Proyecto Fin de Grado
Ingeniería de Tecnologías Industriales

Uso de redes neuronales para identificación de matrículas

Autor: José Enrique Maese Álvarez
Tutor: Antonio Javier Gallego Len

Dpto. de Ingeniería de Sistemas y Automatización
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2023

Proyecto Fin de Grado
Ingeniería de Tecnologías Industriales

Uso de redes neuronales para identificación de matrículas

Autor:
José Enrique Maese Álvarez

Tutor:
Antonio Javier Gallego Len
Profesor sustituto interino

Dpto. de Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2023

Proyecto Fin de Grado: Uso de redes neuronales para identificación de matrículas

Autor: José Enrique Maese Álvarez

Tutor: Antonio Javier Gallego Len

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2023

El Secretario del Tribunal

A mi familia y amigos.

Agradecimientos

Quisiera expresar mi agradecimiento, en primer lugar, a todos aquellos que me han acompañado durante mi carrera y que siguen a mi lado apoyándome en estos últimos esfuerzos. A mi familia, que estuvo ahí en cada obstáculo y me ayudó a superarlo y seguir adelante cuando yo mismo pensaba que no podía y a mis amigos de toda la vida, que siempre me animaron y me dijeron que era capaz de superar cualquier desafío, y que todavía hoy me dan ánimos y me apoyan.

También quiero agradecer especialmente a todos aquellos profesores que han sido parte de mi vida, desde mi tutor en este proyecto hasta otros que me han inspirado y dado constancia. Han sido una fuente de motivación y aprendizaje para mí.

Concluyo así una etapa en la que el aprendizaje y el descubrimiento han sido una parte especial de mi vida. Agradezco a todas las personas que han hecho posible este logro y han estado a mi lado durante esta carrera.

José Enrique Maese Álvarez

Estudiante de Ingeniería

Sevilla, 2023

Resumen

Durante siglos se ha buscado la replicación de la consciencia y pensamientos humanos por parte de máquinas. Lo que hasta hace unas décadas pertenecía al campo de la ciencia ficción, gracias al desarrollo exponencial de la tecnología estos ideales se están convirtiendo en una realidad cada vez más común. Automatizar tareas que antes solo podían ser realizadas por un ser humano es cada vez más posibles por la evolución de la tecnología.

El reconocimiento óptico de caracteres alfanuméricos u otras formas y patrones es uno de los procesos que más cambios y mejorías han experimentado en este campo. Esto ha permitido a las máquinas procesar información de nuestro entorno para luego combinarla con otras tareas de automatización y clasificación. De esta manera, las máquinas están cada vez más cerca de competir con las capacidades humanas de lectura y abstracción. Sin embargo, esta tarea también es una de las que más retos debe afrontar debido a la variabilidad en la forma en que cada persona escribe un mismo carácter y las condiciones en las que cada texto se nos puede presentar.

Recientemente, gracias al estudio de las redes neuronales, la creación de arquitecturas de procesamiento de información cada vez más complejas, y la implementación de hardware más potente en ordenadores comerciales, se ha logrado llevar todas estas cuestiones a un público más amplio. Esto ha permitido mejorar la precisión de los sistemas de reconocimiento óptico y minimizar los errores.

Este proyecto profundiza en el estudio de este campo desde sus fundamentos, abordando los retos y la tecnología a emplear. Posteriormente, se realiza una comparativa de diferentes modelos, demostrando que el desarrollo de la Inteligencia Artificial ha pasado de ser un campo restringido a un reducido grupo de personas a ser accesible a cualquier individuo con inquietudes y tiempo para contribuir al avance de la sociedad.

Abstract

For centuries, the replication of human consciousness and thoughts by machines has been sought. What until a few decades ago belonged to the realm of science fiction, thanks to the exponential development of technology these ideals are becoming an increasingly common reality. Automating tasks that previously could only be performed by a human being is becoming more and more possible due to the evolution of technology.

Optical recognition of alphanumeric characters or other shapes and patterns is one of the processes that has seen the most changes and improvements in this field. This has enabled machines to process information from our environment and then combine it with other automation and classification tasks. In this way, machines are getting closer and closer to competing with human reading and abstraction capabilities. However, this task is also one of the most challenging due to the variability in the way each person writes the same character and the conditions in which each text can be presented to us.

Recently, thanks to the study of neural networks, the creation of increasingly complex information processing architectures, and the implementation of more powerful hardware in commercial computers, all these issues have been brought to a wider audience. This has made it possible to improve the accuracy of optical recognition systems and minimize errors.

This project studies this field in depth from its foundations, addressing the challenges and the technology to be employed. Subsequently, a comparison of different models is made, demonstrating that the development of Artificial Intelligence has gone from being a field restricted to a small group of people to being accessible to any individual with concerns and time to contribute to the advancement of society.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

1 INTRODUCCIÓN

We are making this analogy that AI is the new electricity. Electricity transformed industries: agriculture, transportation, communication, manufacturing.

- Andrew NG -

En los últimos años, las tecnologías relacionadas con la inteligencia artificial como las redes neuronales se han utilizado en numerosos campos para hacernos la vida más sencilla. Para el caso a tratar, el reconocimiento de los caracteres de una matrícula, se van a emplear herramientas para tratar y procesar las imágenes de forma que podamos extraer de ellas los caracteres individuales.

Sin embargo, esta tarea solo puede realizarse de manera correcta si tenemos en cuenta ciertos criterios. Un reconocimiento óptimo requiere que las condiciones en las que se toma la imagen del vehículo se encuentren dentro de unos parámetros de distancia, altura e iluminación. Además, nuestro sistema requerirá de la segmentación previa de la matrícula y, una vez conseguido, de los caracteres. Existen algunos sistemas de redes neuronales capaces de conseguir los mismos resultados sin segmentación previa, pero no se trabajará con ellos en este proyecto.

El tratamiento de la imagen se dividirá en varias partes. En primer lugar, localizaremos y recortaremos la sección correspondiente al área de la matrícula y a continuación se segmentarán los caracteres de forma individual. Para ello, se van a aplicar técnicas de procesamiento de imagen como la detección de contornos, separación por histograma, erosión, dilatación y localización de figuras por sus proporciones y área. Finalmente, contaremos con una base de datos lo suficientemente amplia como para servir de entrenamiento a algunos de los modelos de redes neuronales convolucionales más utilizados.

Aunque existen distintos modelos de redes neuronales además de otras técnicas de inteligencia artificial capaces de resolver este problema, en este trabajo nos centraremos en tres de las más utilizadas para comparar su arquitectura, funcionamiento y resultados.

1.1 Objetivos

1.1.1 Objetivos generales

El objetivo principal de este proyecto será crear y estudiar diferentes modelos de redes neuronales para comprobar su fiabilidad en el reconocimiento de los caracteres de la matrícula de un vehículo. El proyecto puede tener muchas más aplicaciones además de la mencionada siendo esencial la información que se adquiere acerca de la nueva tecnología relacionada con la inteligencia y visión artificial además de familiarizarse con el

entorno de trabajo usado habitualmente en la industria de este campo. Gracias al estudio de los tres modelos escogidos, se conseguirá el conocimiento necesario para crear cualquier tipo de red neuronal y entrenarla en el entorno de Python. También es importante mencionar los conocimientos teóricos y matemáticos que envuelven todo el campo de la inteligencia artificial.

1.1.2 Objetivos específicos

Aunque el objetivo general es el estudio y aplicación de redes neuronales convolucionales, serán necesarias distintas técnicas de procesamiento de imágenes de visión artificial antes de introducirnos en el problema real. Para ello, debemos crear un algoritmo capaz de llevar a cabo las siguientes tareas:

- Procesar las imágenes de entrada para detectar la posición de los caracteres de la matrícula del vehículo.
- Recortar las imágenes para eliminar toda la información que no pertenezca a los caracteres o la matrícula.
- Conseguir un modelo funcional entrenando una red neuronal para que dados los caracteres de una imagen sea capaz de determinar la matrícula.
- Aplicar el modelo obtenido a imágenes que no pertenecen al entrenamiento para conseguir determinar la matrícula.

1.2 Metodología

Para llevar a cabo el trabajo en primer lugar deberemos crear una base de datos de caracteres etiquetados adecuadamente. Para ello tomaremos imágenes reales con ciertas condiciones de luminosidad, altura y distancia a los vehículos que después serán procesadas por el algoritmo de recorte. Al ser una creación propia tendrá un tamaño limitado, lo que influirá tanto en los resultados obtenidos como en los modelos que podemos entrenar con ella. Una vez obtenida la base de datos podremos entrenar los distintos modelos. Los modelos escogidos (LeNet-5, AlexNet y ResNet50) varían tanto en profundidad como en estructura para comprobar cuál se adecúa mejor a nuestro problema. Se pretende entrenar cada modelo con distintos parámetros para comprobar la relación entre ellos y comprender mejor su funcionamiento.

2 ESTADO DEL ARTE

"I believe that technology can do so much good in the world, it depends on the creator, and whether they thought through the ways it can be used and misused."

- Tim Cook -

La matemática Ada Lovelace, programadora del primer algoritmo creado para ser ejecutado por una máquina, especulaba en 1842 acerca de la imitación del aprendizaje humano por parte de una computadora. Lovelace declaraba que una máquina *"podría componer piezas musicales y científicas de cualquier grado de complejidad y extensión"* [1]. Sin embargo, esta idea distaba aún mucho de lo que hoy en día se conoce como inteligencia artificial. Hubo que esperar un siglo, hasta 1950, hasta que Alan Turing dio origen al concepto actual en el ensayo "*Computing Machinery and Intelligence*" [2]. Pocos años más tarde, en 1956, durante la conferencia de Darmouth (Hanover, Nuevo Hampshire, EEUU) [3] organizada por John McCarthy, se acuñó definitivamente el título "Inteligencia Artificial" y asentaron las bases del campo de estudio. Este evento reunió a personalidades como Marvin Lee Minsk, John McCarthy o Claude Shanon que anticiparon una revolución durante la próxima década.

Figura 2-1. Participantes de la convección de Darmouth. De izquierda a derecha: Oliver Selfridge, Nathaniel Rochester, Ray Solomonoff, Marvin Minsk, Trenchard More, John McCarthy, Claude Shanon.

Durante los años 60 se recorrieron al fin los primeros pasos del campo de la inteligencia artificial. Uno de sus máximos exponentes en esta época fue el programa ELIZA [4]. Desarrollado en el MIT por Joseph

Weizenbaum en 1966, fue uno de los primeros programas en poder procesar lenguaje natural y aparentaba mantener una conversación mediante una serie de frases preprogramadas. Sin embargo, la IA experimenta un retroceso durante la década siguiente. Los tan esperados avances no llegaron. Tanto la limitación de la tecnología como el excesivo optimismo inicial propiciaron el "informe Lightill" (Reino Unido, 1973). En este informe se pone en entredicho la utilidad práctica de la IA y supuso un recorte de la financiación. El campo de la inteligencia artificial se sumó en lo conocido como "Invierno IA" hasta el inicio de la década de os 80.

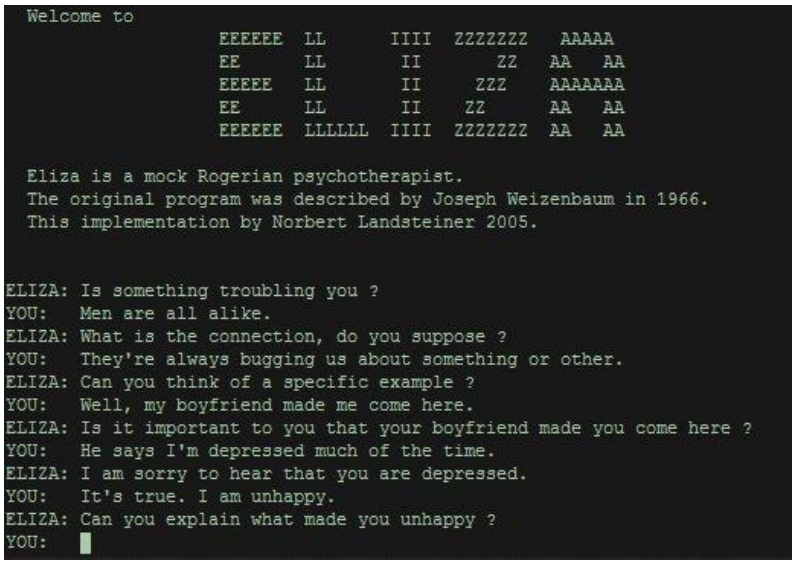

Figura 2-2. Imágenes del programa ELIZA [4].

En 1980, el país nipón renovó la expectación por esta ciencia tras anunciar el "Proyecto de Sistemas Informáticos de Quinta Generación". Los primeros sistemas expertos llegaron a al mercado con un éxito nunca visto hasta la fecha. La inversión en sistemas de inteligencia artificial se contabilizaba en miles de millones de dólares a mediados de la década. Durante este periodo se incorporaron las teorías de la probabilidad y la decisión. Los nuevos métodos incluían modelos como las redes bayesianas o modelos ocultos de Markov, así como algoritmos evolutivos inspirados en conceptos como la recombinación de genes, reproducción y mutaciones.

Algunos investigadores alertaron que este entusiasmo traería consigo una nueva decepción. En efecto, a principios de los 90 se alcanzó el segundo "Invierno IA". Esta vez, sin embargo, no cabía duda de que la inteligencia artificial seguía teniendo mucho recorrido por delante. El aumento de la capacidad computacional, la investigación desarrollada en las últimas décadas y el acceso a grandes cantidades de datos han propiciado la llegada de un periodo dorado de la IA repleto de avances. En palabras del investigador y profesor de la Universidad de Stanford, nos encontramos en una "primavera perpetua de la inteligencia artificial".

2.1 Últimos avances

El ensayo "*Computing Machinery and Intelligence*", de Alan Turing, comenzaba con estas palabras: *"Propongo que se considere la siguiente pregunta, '¿Pueden pensar las máquinas?'"*. En ese mismo ensayo propuso la prueba conocida como Test de Turing que evaluaba la conversación entre un humano y una máquina. Poco después, para evitar ambigüedades, modificó la pregunta: *"¿Existirán computadoras digitales imaginables que tengan un buen desempeño en el juego de imitación?"* Según el matemático, un examinador externo, conocedor de que al menos uno de los participantes era una máquina, debía discernir de cual se trataba. Con esta prueba no se considera la veracidad del conocimiento de la máquina sino su capacidad de expresarlo mediante lenguaje humano.

Algunos programas como ELIZA o PARRY lograron superar el test de Turing en ciertos aspectos. Aun así, un programa capaza de imitar el comportamiento humano a voluntad se ha considerado más cercano a un elemento de ciencia ficción que a la realidad científica durante décadas. Sin embargo, esto está cambiando con

gran rapidez en los últimos años.

OpenAI lanzó en noviembre de 2022 un nuevo modelo de lenguaje optimizado para el diálogo [5]. El chatbot, basado en el modelo GPT-3 (Generative Pre-trained Transformer 3) es capaz de generar respuestas de texto similares a las de una persona a partir de lenguaje natural. Además de mantener una conversación, puede redactar una noticia, escribir un poema o incluso generar código en varios lenguajes de programación. Aunque las respuestas que ofrece no siempre son correctas, la similitud con la forma humana de expresarlo es sorprendente.

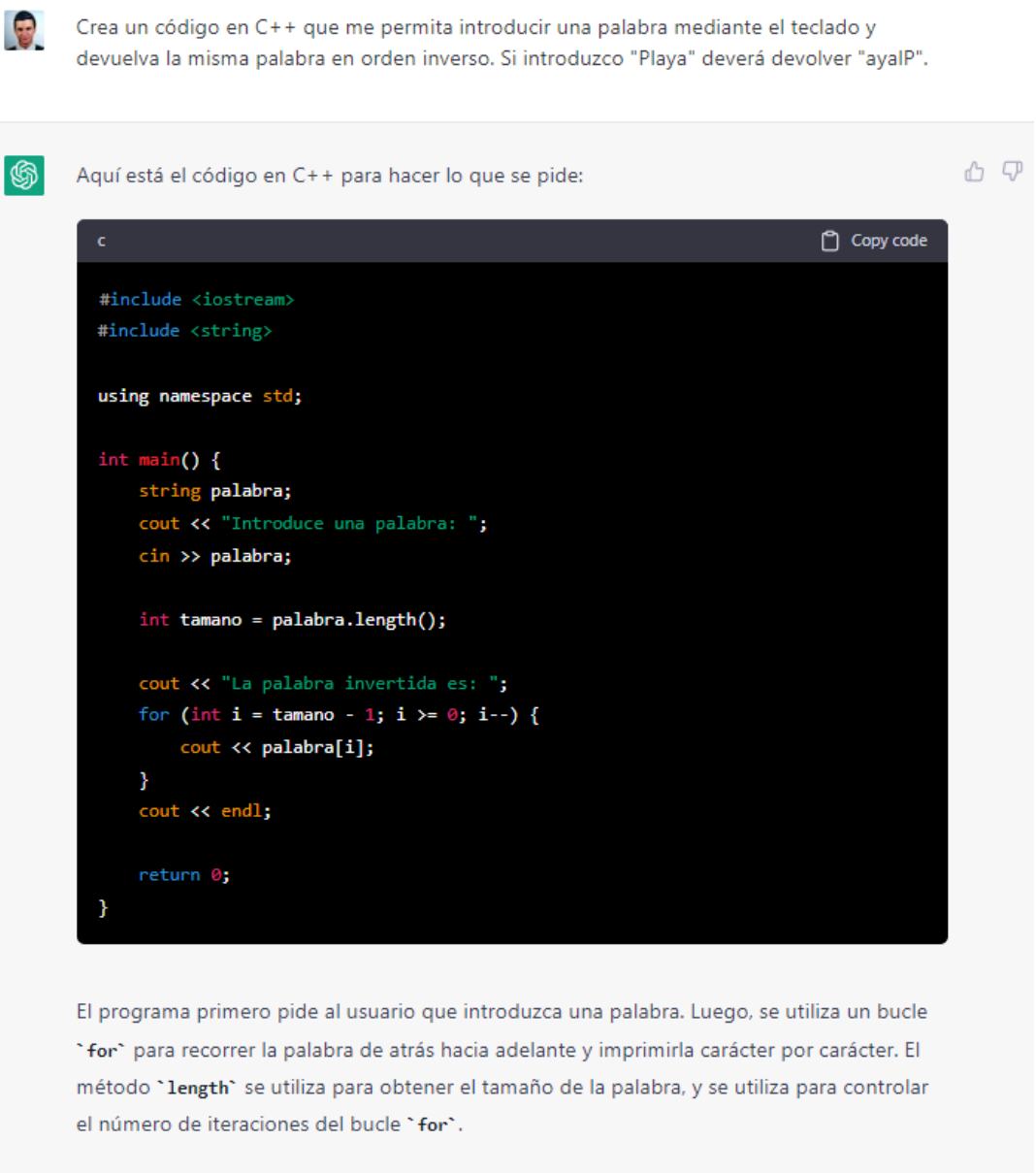

Figura 2-3. Ejemplo de texto y código autogenerados por el programa ChatGPT (modelo GPT3) [5].

Otro campo en los que los modelos de inteligencia artificial están sobresaliendo recientemente es en el ámbito de los programas *Text-to-Image*, capaces de crear una imagen a partir de texto escrito. Comenzando en 2020 con *DALL-E* [6], también de OpenAI, han proliferado durante los últimos años distintos modelos que continuamente mejoran al anterior como Midjourney o Stable Diffusion. Llegados a este punto, muchos consideran obsoleta la pregunta que se hizo Alan Turing hace 70 años y se hacen una nueva: ¿Pueden las máquinas crear arte?

Esta pregunta involucra una discusión más filosófica que técnica y está lejos de responderse.

Figura 2-4. Imagen generada con Midjourney.

Sin embargo, no deben ignorarse los nuevos problemas que pueden crear. Los sistemas de reconocimiento automático de voz (ASR) como VALL-E, de Microsfot, son capaces de imitar cada vez mejor la voz de cualquier persona a partir de un corto ejemplo de grabación. Estos programas, junto con los ya mencionados *Midjourney* y DALL-E, pueden utilizarse para crear noticias falsas entrenándolos con imágenes de figuras públicas. Al ser en ocasiones casi indistinguibles de la realidad, pueden crear situaciones peligrosas o incluso interferir de manera fraudulenta en el ámbito económico y político.

Toda nueva tecnología trae consigo evidentes avances en diversos frentes. La conducción autónoma y el consecuente aumento de la seguridad vial, la mejora de servicios de atención a clientes, o los avances en medicina gracias a la visión artificial son algunos de estos ejemplos. Aunque no deben omitirse los posibles abusos que pueden llegar a cometerse, los últimos modelos influyen de forma notable en ver con optimismo el camino que tiene por delante la inteligencia artificial.

3 FUNDAMENTOS TEÓRICOS

"By far, the greatest danger of Artificial Intelligence is that people conclude too early that they understand it."

- Eliezer Yudkowsky -

En este capítulo se definirán los conceptos de red neuronal, Machine Learning y Deep Learning. Estudiaremos la arquitectura básica de una red neuronal y los diferentes procesos de aprendizaje. Introduciremos los conceptos esenciales de las redes neuronales convolucionales capaces de tratar imágenes. Por último, analizaremos los métodos de entrenamiento y como evitar los problemas más comunes.

3.1 Machine Learning

En muchas ocasiones se han utilizado los términos de Machine Learning (Aprendizaje Máquina) y Deep Learning (Aprendizaje Profundo) de forma equivalente. Aunque la frontera entre estos conceptos en ocasiones es difusa, intentaremos analizarlos por separado para localizar sus principales características.

El Machine Learning se conoce como una subcategoría de la inteligencia artificial centrada en el uso de conjuntos de datos y algoritmos que imitan el aprendizaje natural. A través de diferentes métodos estadísticos, se entrena el algoritmo en cuestión mediante datos para aprender ciertos patrones y realizar predicciones [7].

Los diversos problemas que puede resolver el Machine Learning se ve reflejado en los distintos modelos algorítmicos. La elección del adecuado dependerá del tipo de problema ante el que nos encontremos, la cantidad de datos o su estructura. Algunos de los algoritmos más conocidos son los árboles de decisión, las redes bayesianas o los modelos de regresión (Máquina de Soporte de Vectores, MSV). En este proyecto, sin embargo, nos centraremos en uno de los modelos que más importancia está teniendo en la última década: las redes neuronales.

3.1.1 Tipos de aprendizaje

La fase de entrenamiento de los distintos modelos puede variar según la finalidad y los datos disponibles. Aunque más adelante vamos a profundizar en el proceso de entrenamiento, ahora explicaremos las diferencias entre el aprendizaje supervisado y no supervisado. [8]

3.1.1.1 Aprendizaje supervisado

Se consideran modelos con aprendizaje supervisado aquellos que durante la fase de entrenamiento requieran datos de entrada y salida etiquetados. Como su nombre indica, al menos una parte del modelo requiere la supervisión humana. La mayoría de los datos disponibles están en bruto, por lo que el etiquetado de grandes conjuntos de datos se considera un proceso con una carga de trabajo alta.

Gracias a los datos de entrada y salida, los modelos de aprendizaje supervisado localizan y aprenden patrones. Estos patrones permiten predecir resultados a partir de nuevas entradas de datos no etiquetados. Esta variedad de aprendizaje se utiliza en modelos predictivos y en la clasificación de distintos tipos de archivos como imágenes o palabras.

3.1.1.2 Aprendizaje no supervisado

Al contrario que en el aprendizaje supervisado, en este caso tratamos el entrenamiento de modelos de datos sin procesar ni etiquetar. El propio algoritmo deberá ser el que aprenda de manera autodidacta, por lo que necesita menos interacción humana que el supervisado. Una vez establecidos los parámetros generales con los que trabajará, el modelo es capaz de procesar grandes cantidades de datos sin intervención humana.

El aprendizaje no supervisado se utiliza en segmentación de datos (clúster de datos) que presentan ciertas similitudes en su disposición o características, así como para entender la relación entre diferentes conjuntos, como recomendaciones automatizadas.

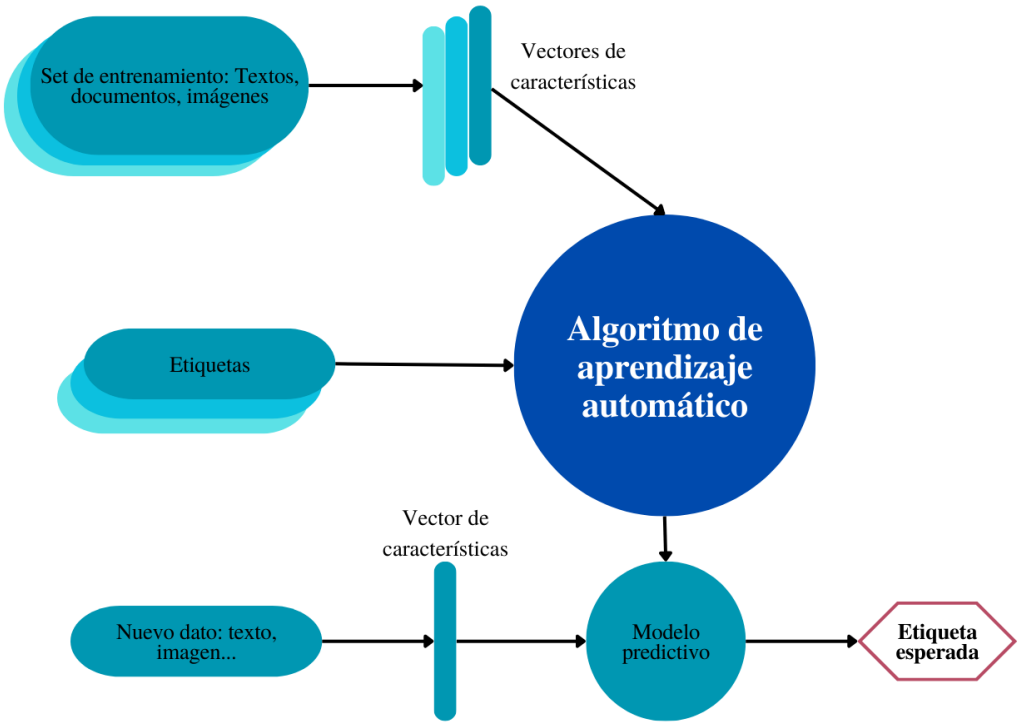

Figura 3-1. Diagrama de funcionamiento del aprendizaje automático no supervisado [8].

3.2 Deep Learning y redes neuronales

El Deep Learning se define como un conjunto de técnicas de aprendizaje automático basado en redes neuronales capaces de reconocer y solucionar problemas complejos a partir de ejemplos. En general, una red neuronal que conste de tres o más capas (incluyendo las entradas y salidas) se considera una red neuronal profunda (Deep Neural Network, DNN) .

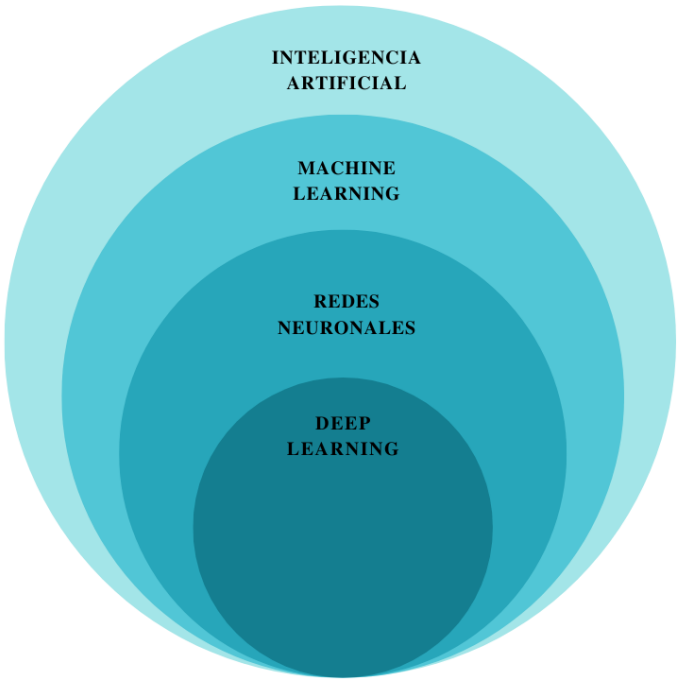

Figura 3-2. Diagrama de Venn del Deep Learning en la Inteligencia Artificial.

Antes de comenzar a estudiar un problema complejo como el procesado de imágenes para obtener caracteres alfanuméricos, debemos entender los conceptos principales de las redes neuronales. Partiendo de la arquitectura más básica (un único perceptrón) se realizará una generalización para comprender las arquitecturas más complejas con múltiples capas.

3.2.1 Perceptrón

El perceptrón, más comúnmente denominado neurona, se puede entender como la mínima unidad de una red neuronal. Se trata de una función matemática basada en un proceso natural biológico. Cada perceptrón recibe múltiples entradas, las evalúa mediante una función no lineal y otorga un resultado.

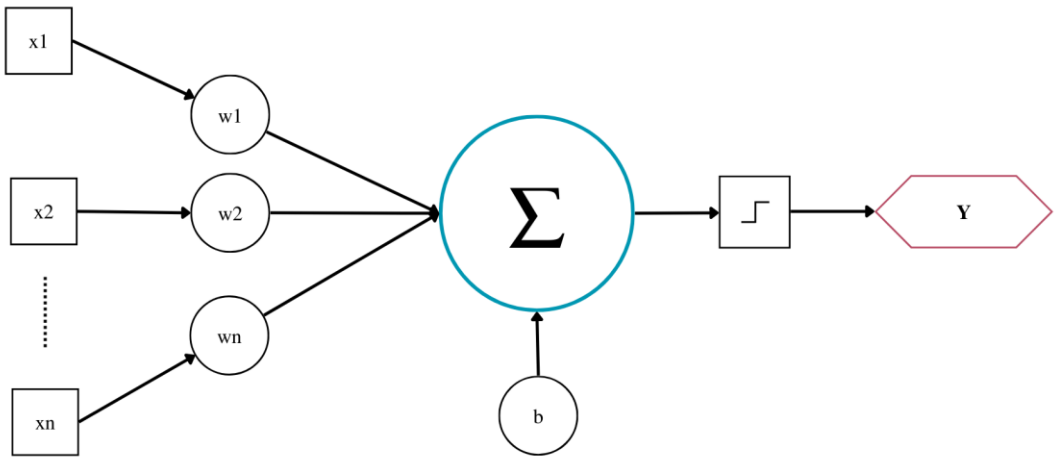

Figura 3-3. Perceptrón.

Si estamos ante un problema con *N* datos de entrada, un perceptrón consistiría en una función matemática donde se multiplican los datos de entrada (x_i), por los coeficientes de peso (w_i). La suma de estos valores se compara con un umbral dado (b). En caso de superar este umbral (valor positivo) la neurona artificial se activará y proporcionará un valor de salida (Y) [9].

$$f(x) = \begin{cases} 1 & si \sum_{i=1}^{n} w_i x_i > b \\ 0 & si \sum_{i=1}^{n} w_i x_i \leq b \end{cases} \quad (3 - 1)$$

A continuación, recolocando los términos obtenemos la función clásica de un perceptrón. Definimos el nuevo término como *bias* (sesgo), siendo este un escalar de valor opuesto al umbral. Como vemos, la neurona artificial se activará en caso de resultado positivo.

$$f(x) = \begin{cases} 1 & si \sum_{i=1}^{n} w_i x_i - b > 0 \\ 0 & si \sum_{i=1}^{n} w_i x_i - b \leq 0 \end{cases} \quad (3 - 2)$$

3.2.2 Red neuronal

Las redes neuronales están compuestas por múltiples perceptrones formando capas. El resultado de los perceptrones de una capa se utiliza como datos de entrada de la siguiente, creando así la conocida estructura de red neuronal. En un primer acercamiento al problema, los pesos y bias son los únicos elementos que modifican la red para obtener el resultado requerido. Por lo tanto, podemos considerar una red neuronal como un gran problema de optimización donde debemos localizar los valores óptimos de estos parámetros de forma que el resultado se adecúe al previsto en el set de entrenamiento.

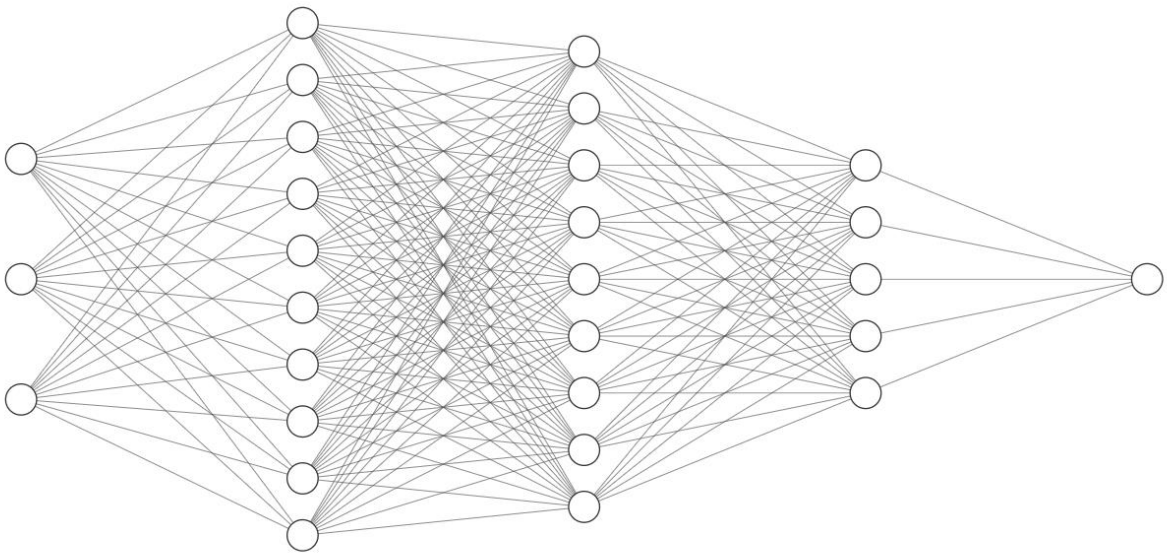

Figura 3-4. Red neuronal profunda multicapa [10].

Podemos apreciar que la salida de cada neurona está conectada a todas las entradas de la capa siguiente,

aunque la importancia de cada unión varía con el peso asignado a cada conexión. Una red neuronal profunda, como se ha mencionado anteriormente, estará compuesta por al menos tres capas ocultas, además de la capa de entrada y salida. Cuanto mayor sea la profundidad de la red y el número de conexiones, más preciso podrá ser el ajuste de los hiperparámetros de la red, aunque también requerirá de un mayor set de entrenamiento y capacidad computacional.

3.2.3 Funciones de activación

En la ecuación 3-2 hemos podido comprobar como un perceptrón modifica la salida siguiendo un modelo en escalón. Este modelo inicial de perceptrón sólo podía resolver problemas de clasificaciones binaria lineales. Podían dividir una nube de datos en dos grupos mediante una recta, aunque no eran capaces de aproximar una función no lineal más compleja. Aunque era útil frente a una aproximación inicial, tiende a alterar bruscamente la salida frente a pequeñas variaciones de los parámetros. Para mejorarlo se recurre a las neuronas con distintas funciones de activación [11].

- Función sigmoidal. Esta función transforma los valores a una escala de 0 a 1. Los valores altos y bajos tienden de manera asintótica a 1 y 0 respectivamente. Aunque presenta una convergencia lenta, es útil para representar probabilidades y tiene un buen rendimiento en la última capa.

$$f(x) = \frac{1}{1 + e^{-x}} \tag{3 - 3}$$

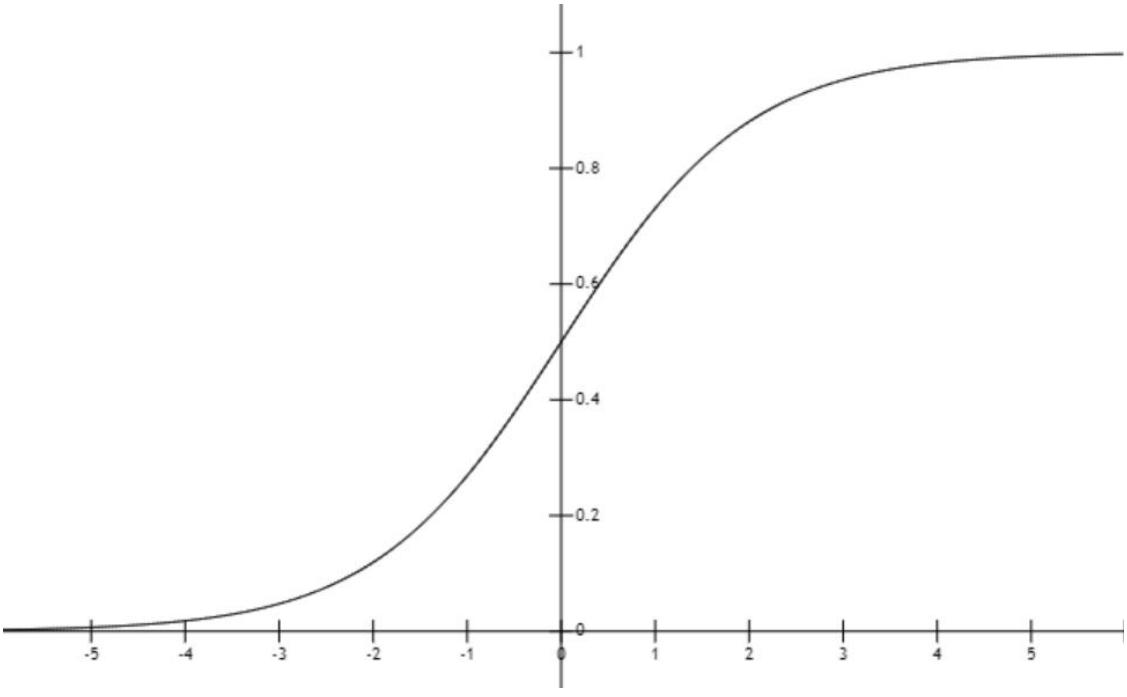

Figura 3-5. Función sigmoidal.

- Función tangente hiperbólica. Muy parecida a la función sigmoidal, aunque en esta ocasión está centrada en 0. Suele utilizarse en problemas de decisión binaria.

$$f(x) = \tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{3 - 4}$$

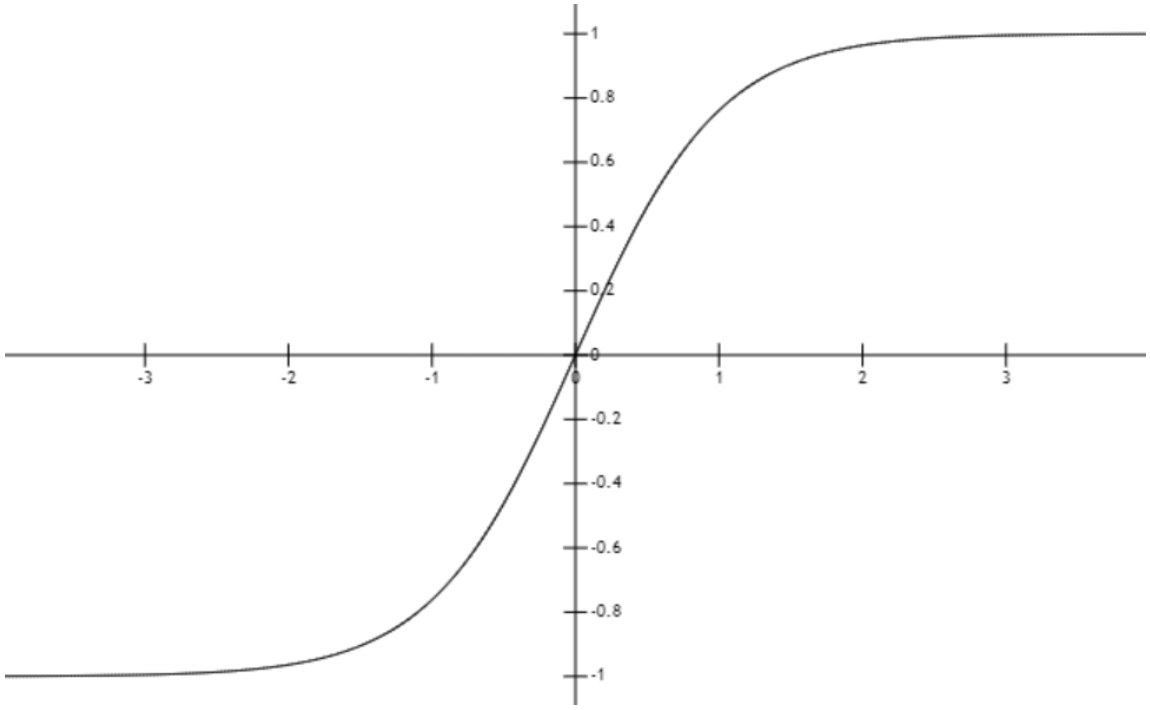

Figura 3-6. Función tangente hiperbólica.

- Función ReLu (Rectified Lineal Unit). Modifica los valores introducidos eliminando la parte negativa. Es una función no acotada, por lo que una salida demasiado alta de algunas neuronas puede reducir la funcionalidad de muchas otras. La función ReLu tiene muy buen comportamiento en redes convolucionales y con imágenes, por lo que la usemos más adelante.

$$f(x) = \begin{cases} 0 & si\ x < 0 \\ x & si\ x > 0 \end{cases} \qquad (3 - 5)$$

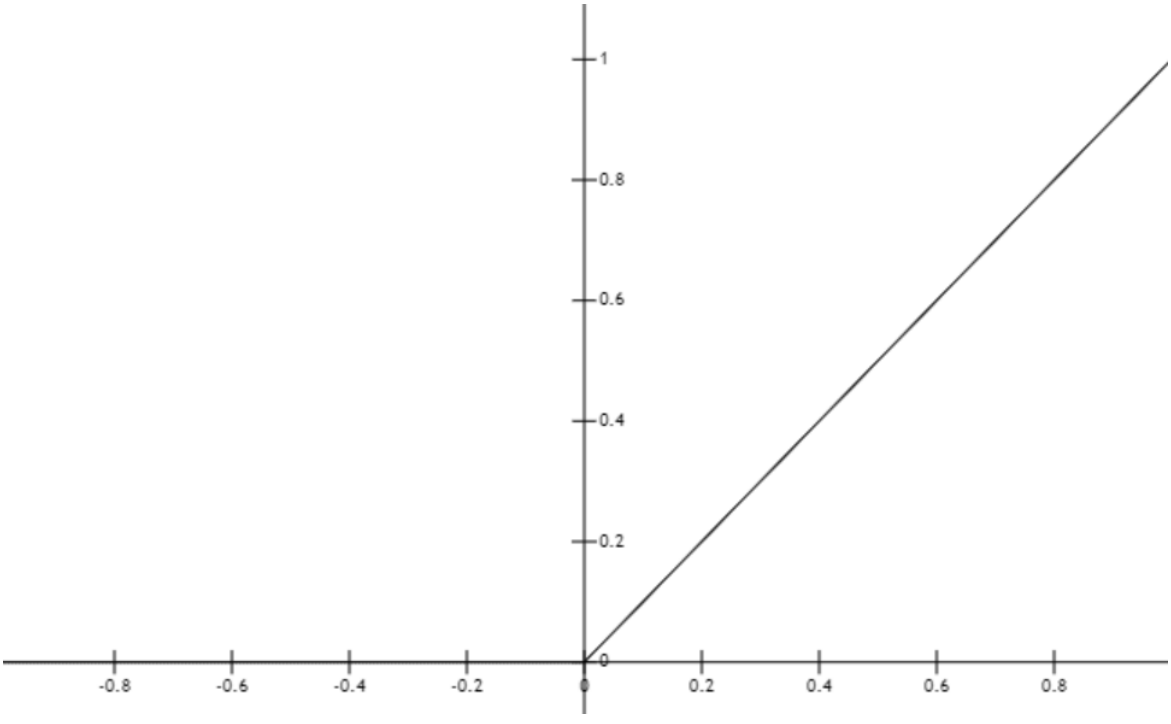

Figura 3-7. Función ReLu.

Existe una variación denominada Leaky ReLu en la que, además, los valores negativos se multiplican por un coeficiente rectificativo, otorgándole menos peso que a los valores positivos, sin llegar a anularlos.

$$f(x) = \max(ax, x) \tag{3 - 6}$$

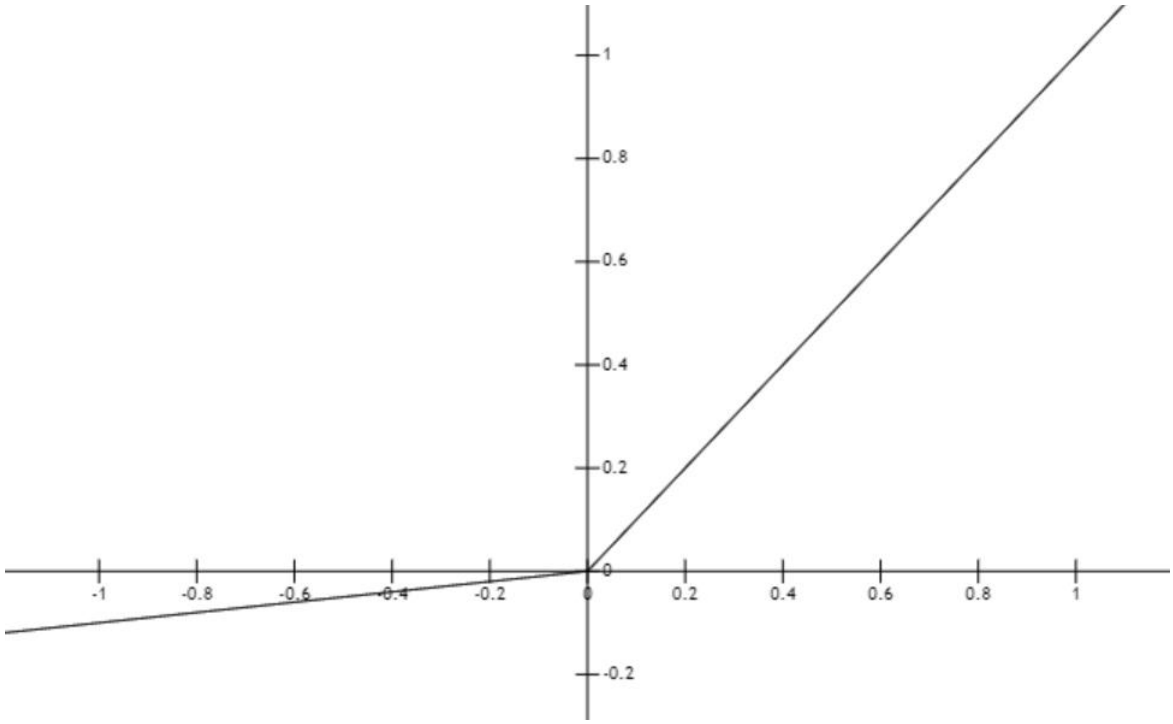

Figura 3-8. Función Leaky ReLu para a = 0,1.

- Función Softmax. Esta función se caracteriza por tener varias salidas que representa en forma de probabilidades, de manera que la suma de todas las salidas es 1. Está acotada entre 0 y 1 y suele utilizarse en la capa de salida de redes de clasificación.

$$f(x_i) = \frac{e^{x_i}}{\sum_{j=0}^{k} e^{x_j}}, \quad i = 0, 1, 2, \dots, k \tag{3 - 7}$$

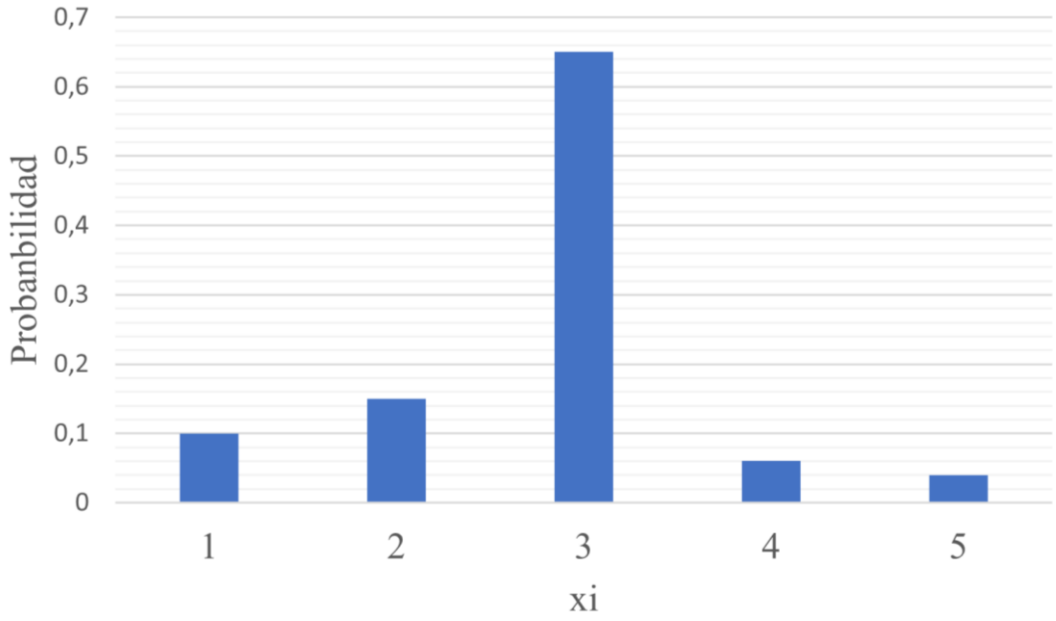

Figura 3-9. Ejemplo de resultado de la función softmax.

3.2.4 Hiperparámetros

Llamamos hiperparámetros a los diferentes parámetros de una red que podemos modificar antes entrenarla. Los hiperparámetros definen la estructura básica de la red, ya que indican el número de capas y de neuronas por capa. Además, también influyen en la tasa de actualización de los parámetros (*learning rate*), en el tipo de optimizador a utilizar (*Adams, Gradient Descent*) o la forma de división de la base de datos (set de entrenamiento y validación, lotes). Una buena elección de los hiperparámetros de una red marcará la diferencia a la hora de entrenarla [12].

Se pueden llegar a construir redes que funciones con una base de datos reducida modificando el número de capas. También, podemos lograr entrenar en poco tiempo una red muy profunda con un alto número de datos de entrada gracias al uso de lotes y de la tasa de actualización.

3.3 Entrenamiento

Como se ha visto anteriormente, el entrenamiento de una red neuronal es en esencia resolver un problema de optimización con un alto número de variables. Debido al gran número de parámetros, su modificación no es un asunto trivial. En 1986 se presentó por primera vez el concepto de propagación inversa (*backpropagation*) que solventaba este problema. La propagación inversa es un método que consiste en optimizar los parámetros de una red neuronal mediante una técnica de descenso por gradiente minimizando el error en cada capa. Su nombre proviene de comenzar la optimización de los parámetros desde la capa de salida de la red hasta la entrada. A continuación, vamos a explicar el procedimiento matemático centrándonos, por simplicidad, en un problema de clasificación binaria a partir de una sola capa oculta.

3.3.1 Forward-propagation

En una primera aproximación al problema, introducimos el concepto de propagación hacia delante o *forward-propagation* [13]. En primer lugar, inicializamos los pesos y bias de la red de forma aleatoria. La inicialización de los pesos y bias no suele ser determinante en el proceso de entrenamiento, aunque una mala inicialización puede hacer que la red converja inevitablemente a cero. Se puede realizar la inicialización de diversas formas:

- Inicialización constante

- Distribución normal
- Método de LeCun (también conocida como *efficient backprop*)
- Método de Glorot/Xavier
- Método de He

El método más utilizado es el de Glorot/Xavier ya que es el utilizado por la biblioteca de *Keras.*

A continuación, se realizar el cálculo de cada neurona de una capa (ecuación 3-8). Para ello se multiplica el peso de la neurona (w) por el valor de cada entrada (x) y se le suma la bias (b). Los valores de entrada serán los datos de set de entrenamiento en el caso de la neurona de entrada o la salida de una neurona de una capa inferior en el resto de los casos.

$$Z = w^T x + b \tag{3 - 8}$$

Finalmente, sustituyendo Z en la función de transferencia escogida (en este caso se ha escogico la función sigmoidal por ser la más utilizada en clasificación binaria) obtenemos $\hat{y}$, el valor predicho.

$$A = \frac{1}{1 + e^{-Z}} = \hat{y} \tag{3 - 9}$$

En el caso general, la ecuación será la siguiente, donde el exponente $[l]$ hace referencia a la capa estudiada.

$$Z^{[l]} = w^{[l]} A^{[l-1]} + b^{[l]} \tag{3 - 10}$$

Mediante el desplazamiento en la dirección natural de la red obtenemos la función de pérdida (*loss function).* Aunque en algunos problemas se podría recurrir a la función de mínimos cuadrados, las redes neuronales de clasificación binaria utilizan en general la función de pérdida de entropía binaria, más conocida por su término en inglés: *binary cross entropy loss.*

$$\mathcal{L} = -[y \cdot \log(\hat{y}) + (1 - y) \cdot \log(1 - \hat{y})] \tag{3 - 11}$$

Los términos de la función 3-11 hacen referencia al valor predicho en la red neuronal ($\hat{y}$) y al valor previsto que debería tener la salida (y). En el ejemplo de clasificación binaria, podemos subdividir a trozos los valores posibles de la salida prevista (ecuación 3-12).

$$\mathcal{L} = \begin{cases} -\log(1 - \hat{y}) & si\ y = 0 \\ -\log(\hat{y}) & si\ y = 1 \end{cases} \tag{3 - 12}$$

Llegados a este punto, podemos calcular la función de coste (*cost function*). Esta será la función que intentaremos optimizar minimizando su valor. En la función 3-13, el parámetro *m* hace referencia al número de datos con el que estamos entrenando la red, por lo que la función buscada es una media de las funciones de pérdida de cada dato de entrenamiento.

$$\mathcal{J}(w, b) = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) \tag{3 - 13}$$

3.3.2 Back-propagation

Finalmente, se realiza la propagación inversa o *back-propagation* [13]. A partir del concepto de derivada como pendiente de una función, lograremos actualizar los parámetros *w* y *b* de manera que converjan hacia el valor que minimice la función de coste. Como podemos apreciar en la ecuación siguiente, el nuevo valor del parámetro a optimizar aumentará o disminuirá en función del signo de la derivada. Para obtener más control sobre este cambio en los parámetros se introduce el concepto de *learning rate* [12]. El *learning rate* aumentar la velocidad de convergencia (si aumenta su valor) o disminuir la oscilación cerca del punto de optimización.

$$w' = w - \alpha \frac{d\mathcal{J}(w,b)}{dw} \tag{3 - 14}$$

$$b' = b - \alpha \frac{d\mathcal{J}(w,b)}{db} \tag{3 - 15}$$

Una red neuronal consta de un gran número de parámetros a optimizar por lo que no se conseguirá un valor cercano al óptimo en un solo paso. Gracias a la repetición del proceso de *forward-propagation* y *back-propagation* se consiguen acercar los parámetros al punto óptimo. Cada repetición recibe el nombre de época.

3.3.3 Posibles problemas

Cuando entrenamos una red neuronal podemos enfrentarnos a diversos problemas que dificultan encontrar la solución óptima. Estos problemas generalmente se encontraron en el momento de la creación de la base de datos y en la elección de los hiperparámetros. Afectan al resultado, no convergiendo al punto óptimo, pero también al tiempo de procesamiento, aumentándolo en exceso. Los problemas más comunes son el *overfitting* y *underfitting* [14].

El problema de *overfitting* hace referencia a redes que tienen resultados perfectos tanto en el set de entrenamiento como en el de validación. Aunque puede parecer que estas redes están trabajando de forma óptima en realidad están sobreentrenando los datos, llegando a memorizar sus resultados esperados. Este aspecto suele darse en redes demasiado pequeñas, con pocos parámetros a optimizar.

Por otra parte, el problema de *underfitting* indica que la red entrenada está mostrando un resultado pobre según lo esperado. El uso de bases de datos pequeñas frente a redes profundas suele acarrear este problema. Sin embargo, puede deberse también a un entrenamiento durante pocas épocas.

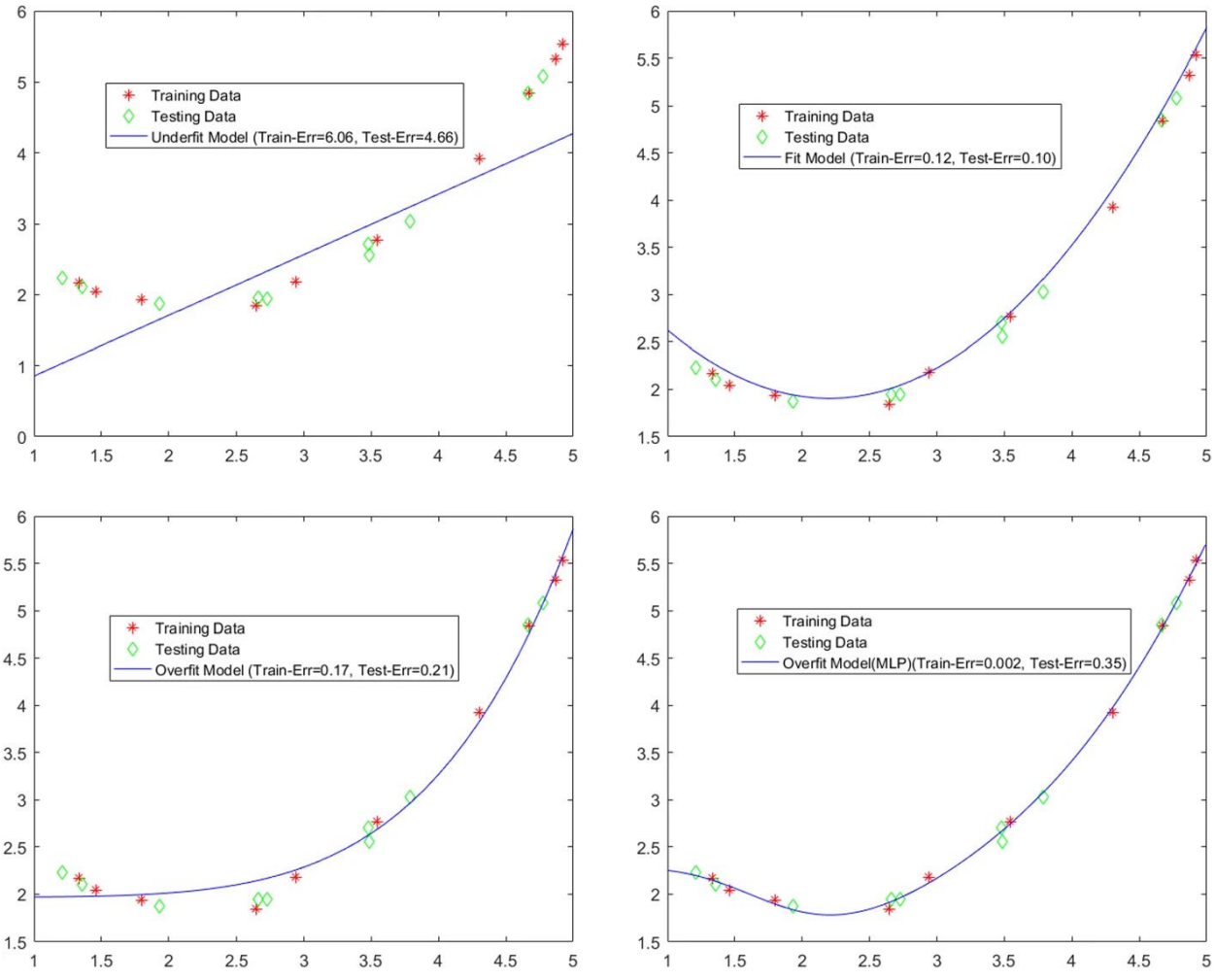

Figura 3-10. Comparación de modelos con *underfitting*, *overfitting* y un ajuste adecuado [14].

Otro de los problemas más comunes son las oscilaciones en torno al punto de optimización. Este problema se da en redes con una tasa de actualización de los parámetros demasiado agresiva, como podemos apreciar en las ecuaciones 3-14 y 3-15. Podríamos asumir que reduciendo esta tasa se eliminaría el problema, sin embargo, esto nos llevaría a otro punto común: el tiempo de entrenamiento. Cuando hablamos de redes profundas con grandes cantidades de datos podemos tener tiempos de ejecución del orden de las horas o incluso días, por lo tanto, cualquier mejora en este aspecto disminuye notablemente el tiempo total del proyecto. Para hacer frente a esto podemos reducir el tamaño de la red, aumentar la tasa de actualización o agrupar los datos en lotes de forma que los parámetros se actualicen menos veces en cada época. Esto último, además, previene el overfitting.

Como vemos, las dificultades a las que nos enfrentamos al diseñar una red neuronal están interconectados y cada uno posee múltiples posibles soluciones. Para simplificar este proceso, muchas veces conviene trabajar con redes cuya estructura ya está definida y su fiabilidad comprobada, limitando los hiperarámetros que podemos modificar.

3.4 Redes Neuronales Convolucionales

Las Redes Neuronales Convolucionales (CNN) son una clase de redes neuronales diseñadas específicamente para tratar conjuntos de datos bidimensionales, es decir, imágenes. Al entender una imagen como una matriz con un valor específico en cada píxel, al introducirla en una red neuronal convencional no estaríamos especificando para cada píxel la importancia que precisa dentro de la imagen, por lo que perdemos gran parte de la información. Por el contrario, las Redes Neuronales Convolucionales utilizan el método de convolución para tratar estas imágenes. Estas redes están compuestas por dos capas consecutivas: las capas convolucionales y las capas de agrupación (*pooling layers*).

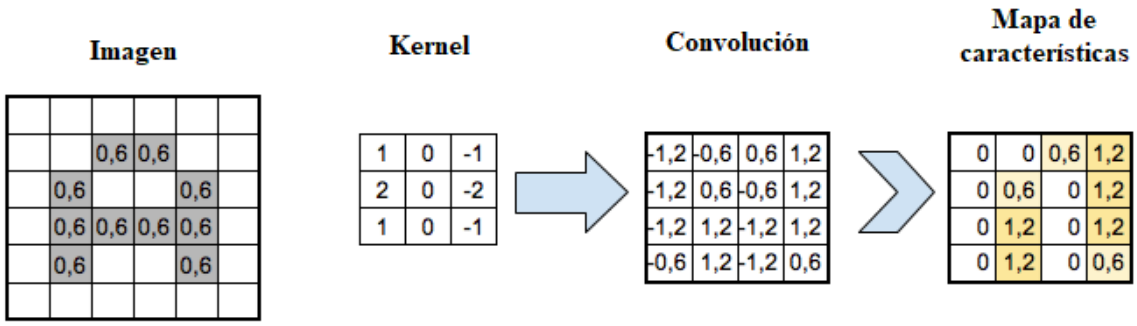

Figura 3-11. Proceso de convolución y obtención del mapa de características [15].

La capa convolucional de una red se encarga en aplicar el proceso matemático de convolución a cada píxel de entrada. La convolución consiste en aplicar un producto escalar a cada elemento y su vecindad contra un filtro (una matriz de menor tamaño denominada Kernel). Tras este paso obtenemos una matriz de salida que será nuestra nueva capa de neuronas ocultas.

El siguiente paso se conoce como capa de agrupación. Estas capas consisten en aplicar una función de activación sobre la matriz obtenida de la convolución. El resultado generado se conoce como mapa de características, ya que nos indica en qué parte de la imagen se han detectado los elementos buscados por el filtro. Tras un primer desarrollo de la capa de convolución y agrupación nuestra red será capaz de discernir la posición de los bordes, cambios en el contraste de la imagen o incluso patrones simples. A medida que aumentamos el número de convoluciones los datos capaces de apreciar aumentan tanto en número como en

complejidad por lo que tras varias repeticiones el modelo será capaz de apreciar patrones más complejos.

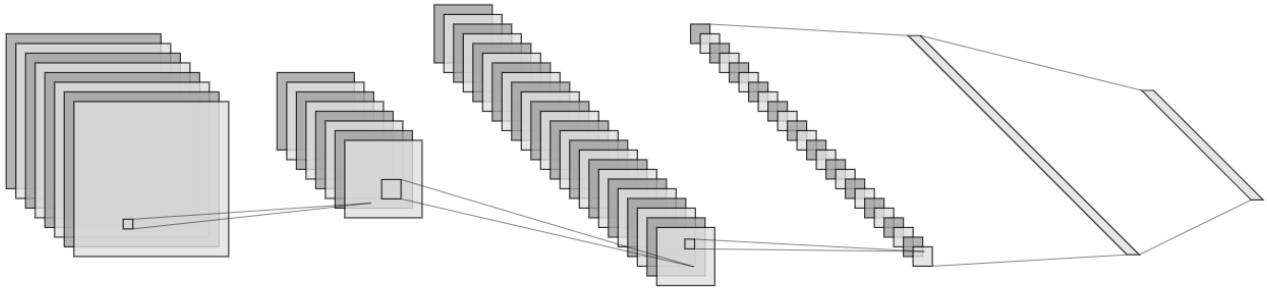

Figura 3-12. Ejemplo de estructura de Red Neuronal Convolucional [10].

4 METODOLOGÍA

The fundamental problem of communication is that of reproducing at one point either exactly or approximately a message selected at another point.

- Claude Shannon -

En este capítulo vamos a describir la metodología seguida a la hora de realizar el proyecto. Comenzaremos explicando la elección del entorno de trabajo comparando su funcionamiento con otros como Matlab. A continuación, desarrollaremos el procedimiento para resolver el problema de manera pormenorizada. Empezaremos con la base de datos escogida y el algoritmo de recorte. Finalmente comentaremos los distintos modelos de redes neuronales escogidos y sus motivos.

4.1 Herramientas

Antes de comenzar a acercarnos al problema nos planteamos que herramienta utilizaríamos para la creación de los algoritmos de Machine Learning y la aplicación de las redes neuronales. Matlab [16] es un lenguaje bien optimizado que recientemente ha incorporado librerías y diversas herramientas de desarrollo de redes neuronales. Estas nuevas librerías permiten crear una red neuronal fácilmente y con poca experiencia siendo muy sencillas de utilizar, sin embargo, debido a su corta vida hay poco material informativo al respecto.

Por otra parte, Python es un lenguaje ampliamente utilizado en el ámbito de la inteligencia artificial. Aunque generalmente es menos eficiente que Matlab, permite trabajar con matrices y conjuntos de datos más fácilmente. Además, debido a su uso extendido, existen numerosa información al respecto de la creación y entrenamiento de redes neuronales a través de distintas librerías. Dado que este proyecto no precisa de gran carga computacional por tener una base de datos relativamente reducida se ha optado por la opción de Python.

Una vez seleccionado el lenguaje de programación debemos escoger la librería principal que utilizaremos para desarrollar las redes neuronales. En general, los dos marcos de aprendizaje profundo más utilizados son TensorFlow [17] y PyTorch [18]. Aunque ambos tienen la misma función final, se ha escogido y trabajar con TensorFlow debido a su mayor simplicidad. Además, TensorFlow tiene incluida Keras, una interfaz de programación de aplicaciones (API) de redes neuronales de alto nivel fácil de usar debido a su modularidad y eficiencia.

4.2 Procedimiento

La finalidad de este proyecto es leer con el menor error posible la matrícula de un vehículo a partir de una imagen. Aunque el procesamiento final de los datos se realizará con herramientas de Deep Learning, las imágenes deberán superar un preprocesado. El procedimiento será el siguiente:

1. Captura de las imágenes para crear la base de datos.
2. Localización de la matrícula del vehículo, recorte y estandarización de la imagen de la matrícula obtenida.
3. Localización de cada carácter de la matrícula, separación, recorte y estandarización de la imagen que contiene a cada carácter.
4. Entrenamiento de la red neuronal. Los datos de entrada serán las imágenes de cada carácter con el tamaño necesario para cada tipo de estructura neuronal. La salida una capa con función de activación Softmax que indique la probabilidad de pertenecer a cada clase.

Los puntos 2 y 3 se realizarán mediante un algoritmo de aprendizaje máquina explicado más adelante. Existen ciertos tipos de redes neuronales como *YOLO* (*You Only Look Once*) [19] capaces de realizar todo el proceso e incluso localizar varias matrículas en una misma imagen. Sin embargo, este tipo de algoritmo precisa de una base de datos mucho mayor por lo que se han escogidos algoritmos que necesitan un tratamiento inicial de los datos de entrada.

4.2.1 Base de datos

La base de datos creada consta de un total de 125 imágenes de vehículos españoles. Se han escogido vehículos posteriores a septiembre del año 2000 para que estén bajo el formato de la normativa europea. Para incluir otros tipos de matrícula, así como matrículas de otros países se necesitaría un algoritmo de recorte más complejo, aunque el modelo de red neuronal no se vería alterado ya que analiza cada carácter por separado. Las imágenes han sido tomadas tanto de la parte delantera como trasera a una altura de 1.5 metros simulando una cámara estática situada en la entrada y salida de un estacionamiento. Se han realizado en diversas situaciones de luz para evitar sesgos en el resultado final.

Figura 4-1. Imagen de vehículo perteneciente a la base de datos.

Según la normativa europea, las matrículas deben tener la bandera de la unión europea situada sobre la inicial del país en cuestión seguida de 7 caracteres alfanuméricos (4 números y 3 letras). Esto hace un total de 875 ejemplos de caracteres de los cuales 500 son numéricos, es decir, 50 ejemplos de media de cada clase de número y 18.75 de cada letra (teniendo en cuenta las letras posibles en una matrícula).

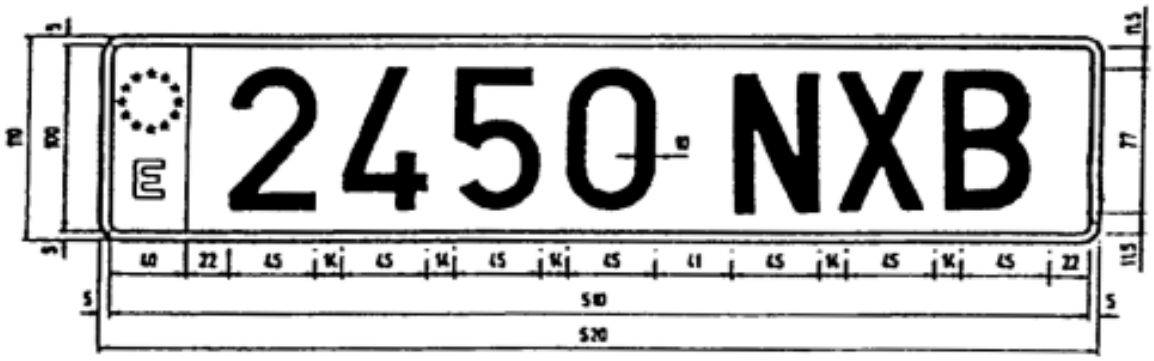

Figura 4-2. Estructura de matrícula española según la normativa europea.

Al tener una base de datos menor en el caso de las letras, se trabajará de ahora en adelante tan solo con la parte numérica de las matrículas. El modelo que seleccionemos finalmente será igualmente válido para ambos casos, suponiendo un entrenamiento con una base de datos mayor.

4.2.2 Algoritmo de recorte

El algoritmo de recorte y preprocesado de imágenes, como se ha mencionado anteriormente, prepara las imágenes de forma que contengan la información necesaria para ser los datos de entrada de las redes neuronales. El algoritmo de en cuestión utiliza diversas técnicas de visión artificial para localizar las matrículas y, una vez recortadas, separar los caracteres. Para ello hacemos uso de la librería OpenCV [20], que proporciona herramientas de libre acceso para trabajar con imágenes en el campo de la visión artificial. Podemos dividir el algoritmo en dos funciones diferenciadas

4.2.2.1 Recorte de matrículas

La primera función del algoritmo de preprocesado se encarga de localizar la parte correspondiente a la matrícula en la imagen y recortarla.

En primer lugar, convertimos la imagen en escala de grises y le aplicamos un filtro Gaussiano para eliminar parte del ruido.

Figura 4-3. Preprocesado de imagen en escala de grises con filtro Gaussiano.

A continuación, extraemos los bordes de la figura mediante el cálculo del gradiente y lo ampliamos con la función *dilate.*

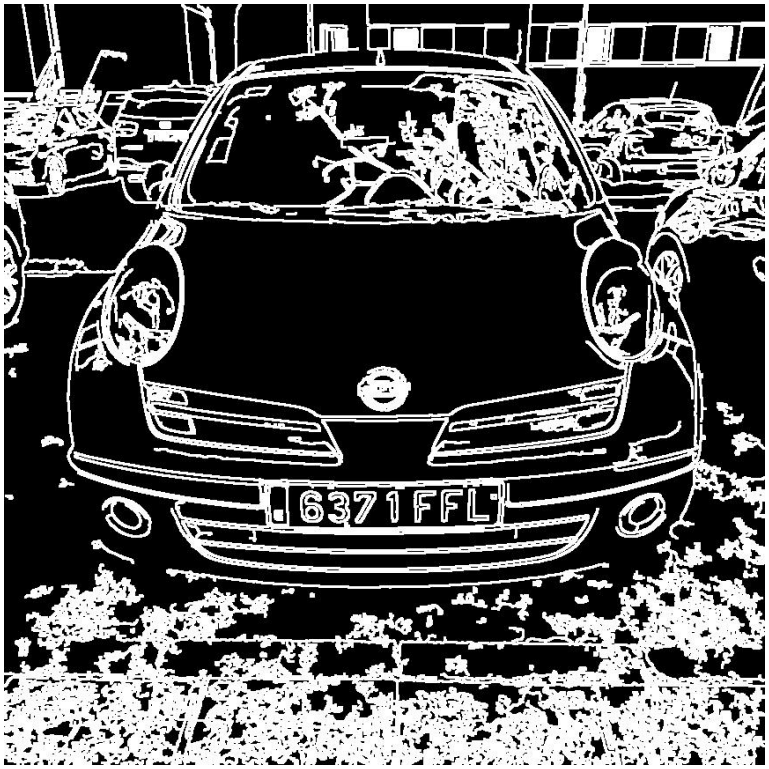

Figura 4-4. Bordes de la imagen del vehículo.

Finalmente recorremos todos los contornos cerrados que han aparecido en la imagen anterior. Escogeremos

aquel que cumpla las condiciones de tamaño y proporcionalidad entre ancho y alto de las matrículas. Debemos tener en cuenta que no podemos tomar valores demasiado precisos ya que las condiciones de distancia al vehículo o ángulo son variables.

Figura 4-5. Matrícula recortada.

Esta función presenta un bajo nivel de dificultad y consigue localizar la matrícula en la práctica totalidad de las imágenes. Sin embargo, el resultado no siempre es óptimo. En ocasiones se le añade un marco a la imagen resultado debido a la geometría del vehículo o no aparece la franja correspondiente a la bandera europea. Estos resultados deberán tenerse en cuenta a continuación a la hora de buscar los caracteres.

Figura 4-6. Ejemplo de recorte subóptimo.

4.2.2.2 Recorte de caracteres

La segunda función del algoritmo de preprocesado se encarga de separar los caracteres de la imagen obtenida anteriormente y recortarlos de forma individual.

En primer lugar, binarizamos la imagen. Además, creamos una pequeña franja blanca al inicio para evitar distinciones entre las matrículas recortadas con la etiqueta europea y sin ella.

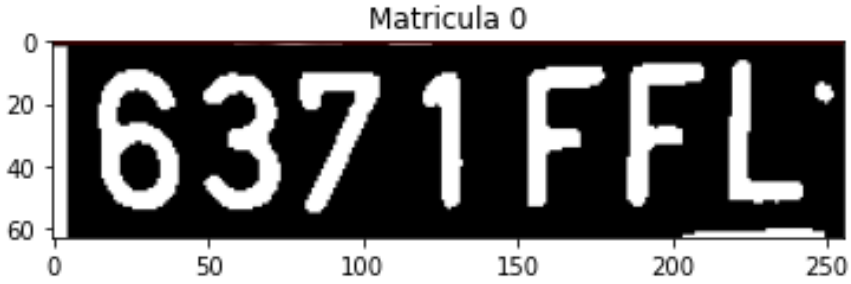

Figura 4-7. Matrícula binarizada.

Para separar los caracteres de la imagen recurrimos al histograma de cada columna de la imagen, es decir, al número de píxeles blancos y negros de cada columna. Como vemos en la imagen anterior, las zonas verticales donde existan mayor número de píxeles blancos corresponderán a un carácter o a la franja inicial. Ciertas imágenes, como la vista en la figura 4-6, pueden presentar un marco que altera los niveles de blanco en la imagen. Por este motivo el límite inferior impuesto para que una zona se reconozca como carácter debe ser

variable en función de las columnas adyacentes.

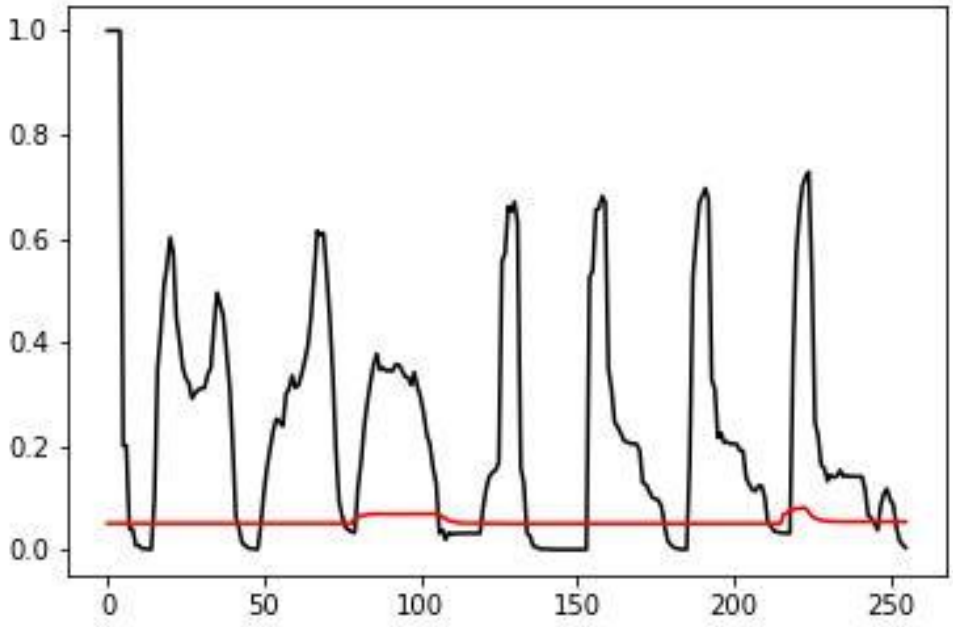

Figura 4-8. Niveles de blanco por columnas. Cada pico corresponde a un caracter. En rojo aparece el límite inferior de separación.

Finalmente, para separar los caracteres, almacenamos cada franja de valores superiores al límite en una imagen distinta. Sabemos que la primera zona corresponde a la etiqueta europea, las siguientes 4 a los números y 3 a las letras.

Figura 4-9. resultado de la separación por caracteres.

Repitiendo este algoritmo en cada una de las 125 imágenes de vehículos obtenemos la base de datos necesaria para entrenar los distintos modelos convolucionales. Los datos de entrada a cada modelo pueden variar en tamaño o formato por lo que, posteriormente, deberemos realizar un procesado extra en algunos casos.

4.2.3 Modelos neuronales

Existen numerosos modelos neuronales convolucionales capaces de reconocer figuras en imágenes. Algunos de estos modelos están pre entrenados con bases de datos de libre acceso como MNIST [21] o ImageNet [22], dirigida al reconocimiento de caracteres manuscritos. Algunos de los modelos más importantes son:

- LeNet-5: Fue una de las primeras redes neuronales convolucionales desarrolladas por Yann LeCun, et al. en 1998, y se utilizó para el reconocimiento de caracteres manuscritos en cheques bancarios [23].
- AlexNet: Fue una de las primeras redes neuronales convolucionales profundas que ganó la competencia de reconocimiento de imágenes ImageNet en 2012. Se utilizó para clasificar imágenes en 1.000 categorías diferentes [24].
- VGG-16: Esta red neuronal convolucional profunda, desarrollada por el equipo de Visual Geometry Group (VGG) en la Universidad de Oxford, fue uno de los mejores modelos en la competencia ImageNet de 2014. Utiliza capas convolucionales con filtros de tamaño 3x3 y pooling para reducir la dimensionalidad de las imágenes [25].
- ResNet50: Esta red neuronal convolucional profunda, desarrollada por el equipo de Microsoft Research en 2015, utiliza una arquitectura residual que ayuda a prevenir el problema de desvanecimiento del gradiente en las capas profundas [26].
- InceptionNet: Esta red neuronal convolucional, también conocida como GoogleNet, fue desarrollada por Google en 2014 y utiliza una arquitectura de módulo de Inception que utiliza filtros de diferentes tamaños en una misma capa [27].
- MobileNet: Esta red neuronal convolucional liviana, también desarrollada por Google, fue diseñada para su uso en dispositivos móviles con recursos limitados y utiliza una arquitectura de capas separables en profundidad para reducir el número de parámetros [28].

En este proyecto utilizaremos los modelos que precisan menos cantidad de parámetros ya que nuestra base de datos es reducida. Por este motivo utilizaremos el modelo LeNet-5 y AlexNet. Además, utilizaremos el modelo ResNet50 que, aunque se trata de una red profunda con un alto número de parámetros, consta de una arquitectura especial capaz de reducir el desvanecimiento del gradiente y por lo tanto puede ser útil en casos con un número de entradas limitadas.

5 ENTRENAMIENTO

"You never fail until you stop trying."

- Albert Einstein -

En este capítulo vamos a describir las estructuras de redes convolucionales escogidas para este proyecto explicando los motivos para su uso. Comenzaremos explicando los distintos tipos de capas que intervienen y posteriormente explicaremos en mayor detalle cada uno de los modelos.

5.1 División de la base de datos

Antes de entrenar cada modelo debemos dividir de manera aleatoria la base de datos con la finalidad de crear un set de entrenamiento y un set de validación. El primero será el encargado de entrenar la red mientras que el segundo nos permite obtener un resultado de su funcionamiento con datos que no pertenecen al entrenamiento. En nuestro caso utilizaremos la división más común:

- Set de entrenamiento: 75%
- Set de validación: 25%

5.2 Modelos

A continuación, se van a presentar los modelos de redes neuronales convolucionales utilizados para realizar el proyecto. En cada caso se especificará los requerimientos de datos de entrada.

5.2.1 LeNet-5

El primero de los modelos que vamos a analizar es LeNet-5. Fue descrita por primera vez por Yann LeCun en 1988 y desarrollo para ser usado en un problema de reconocimiento de caracteres manuscritos. Como demostración se entrenó con el conjunto de datos estándar del MNIST logrando una precisión de aproximadamente el 99,8%.

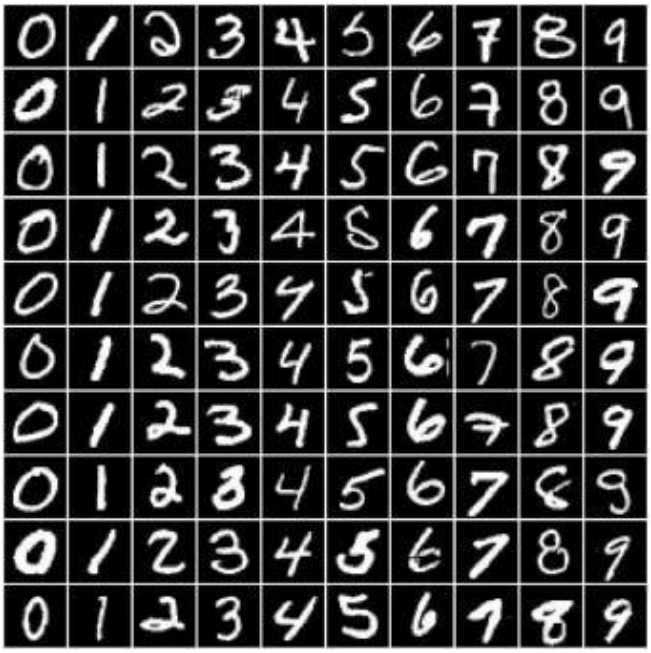

Figura 5-1. Ejemplo extraído de la base de datos MNIST [21].

La red consta de 7 capas. La capa de entrada admite imágenes en escala de grises de tamaño 32x32, el estándar de la base de datos MNIST. El modelo sigue con una capa convolucional y una de agrupamiento también llamada capa de submuestreo. Este conjunto se repite dos veces y media que conformarían la parte de extracción de características. Los mapas de características son las entradas de las capas completamente conectadas para la interpretación y predicción.

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 28, 28, 6)	156
max_pooling2d (MaxPooling2D)	(None, 14, 14, 6)	0
conv2d_1 (Conv2D)	(None, 10, 10, 16)	2416
max_pooling2d_1 (MaxPooling2D)	(None, 5, 5, 16)	0
flatten (Flatten)	(None, 400)	0
dense (Dense)	(None, 120)	48120
dense_1 (Dense)	(None, 84)	10164
dense_2 (Dense)	(None, 10)	850

Total params: 61,706
Trainable params: 61,706
Non-trainable params: 0

Figura 5-2. Capas de la red LeNet-5.

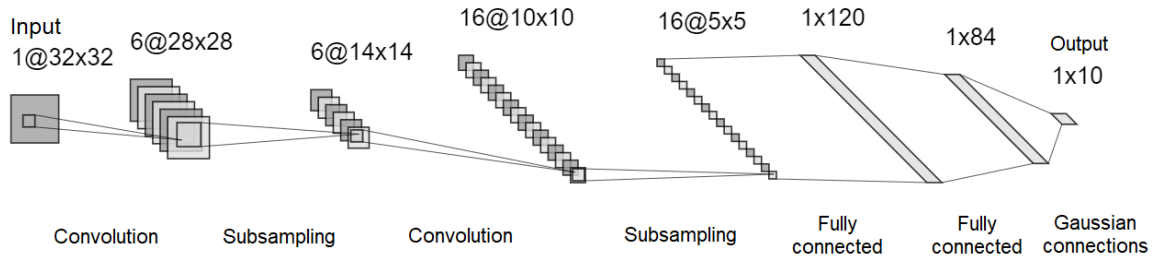

Figura 5-3. Arquitectura de la red neuronal convolucional LeNet-5 [10].

5.2.2 AlexNet

El modelo AlexNet fue abordado en el documento de Alex Krizhevsky, et al. en 2012 denominado *Clasificación de ImageNet con redes neuronales convolucionales profundas*. El documento sirvió de demostración de que es posible desarrollar modelos de extremo a extremo eficaces y profundos para un problema esquivando el uso de técnicas de preentrenamiento.

Algunas de las técnicas novedosas que involucraron son ampliamente utilizadas en las redes convolucionales hoy en día y se han convertido en un estándar. Una de las nuevas técnicas fue la introducción de la función de activación lineal rectificada (ReLu). Esta función actualizó el uso de las ya conocidas función de activación sigmoidal o tangente hiperbólica. Otro cambio importante fue el uso de la función *softmax* en la capa de salida, convirtiéndose rápidamente en un elemento básico en las redes neuronales de clasificación de múltiples clases.

La arquitectura de la red AlexNet es más profunda que la LeNet, pero no podría entenderse sin esta ya que amplía algunos de los patrones ya establecidos. El modelo consta de 5 capas convolucionales en la parte de extracción de características y tres capas completamente conectadas en la parte de clasificación. Al contrario que en el modelo LeNet, se utiliza un patrón de capa convolucional seguido de una segunda capa convolucional. Además, se comenzó a estandarizar el uso de la regularización de Dropout entre las capas totalmente conectadas. Esta regularización consiste en desconectar un cierto número de neuronas de forma aleatoria en cada época con el fin de evitar la sobreparametrización.

Los datos de entrada son imágenes de 224x224 pixeles con tres canales de color.

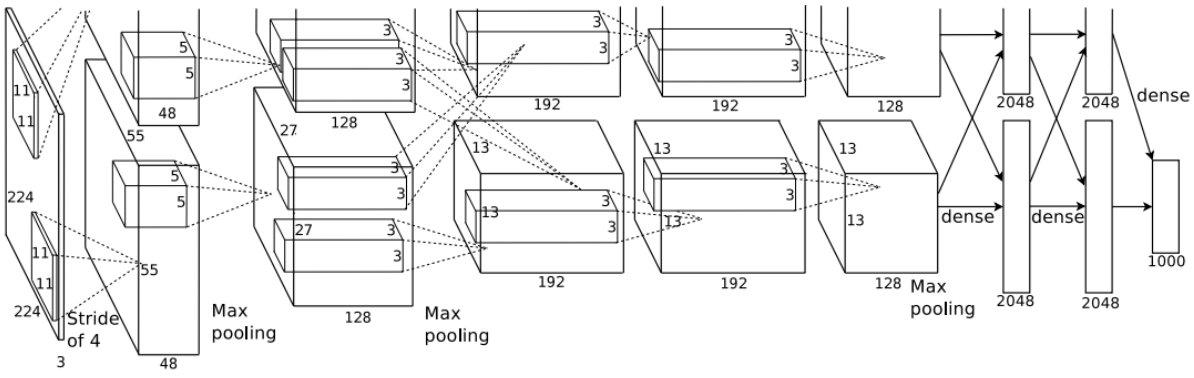

Figura 5-4. Arquitectura de la red neuronal convolucional AlexNet con dos conductos para entrenar con dos GPUs [10].

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_12 (Conv2D)           (None, 55, 55, 96)        34944

batch_normalization (BatchN  (None, 55, 55, 96)        384
ormalization)

max_pooling2d_12 (MaxPoolin  (None, 27, 27, 96)        0
g2D)

conv2d_13 (Conv2D)           (None, 27, 27, 256)       614656

batch_normalization_1 (Batc  (None, 27, 27, 256)       1024
hNormalization)

max_pooling2d_13 (MaxPoolin  (None, 13, 13, 256)       0
g2D)

conv2d_14 (Conv2D)           (None, 13, 13, 384)       885120

batch_normalization_2 (Batc  (None, 13, 13, 384)       1536
hNormalization)

conv2d_15 (Conv2D)           (None, 13, 13, 384)       1327488

batch_normalization_3 (Batc  (None, 13, 13, 384)       1536
hNormalization)

conv2d_16 (Conv2D)           (None, 13, 13, 256)       884992

batch_normalization_4 (Batc  (None, 13, 13, 256)       1024
hNormalization)

max_pooling2d_14 (MaxPoolin  (None, 6, 6, 256)         0
g2D)

flatten_6 (Flatten)          (None, 9216)              0

dense_18 (Dense)             (None, 4096)              37752832

dropout (Dropout)            (None, 4096)              0

dense_19 (Dense)             (None, 4096)              16781312

dropout_1 (Dropout)          (None, 4096)              0

dense_20 (Dense)             (None, 10)                40970

=================================================================
Total params: 58,327,818
Trainable params: 58,325,066
Non-trainable params: 2,752
_________________________________________________________________
```

Figura 5-5. Capas de la red AlexNet.

5.2.3 ResNet50

El último de los modelos de redes neuronales convolucionales a tratar es la Red Residual o ResNet. Consiste en un modelo muy profundo desarrollado por Kaiming He, et al. en 2016 en el documento titulado *Aprendizaje profundo residual para el reconocimiento de la imagen*. Con 152 capas de profundidad sería un modelo demasiado extenso para nuestro trabajo si no fuese por la idea de bloques residuales que conexionan distintas capas a modo de acceso directo. Estas conexiones en la arquitectura de la red, denominadas conexiones de acceso directo proyectadas mantienen la entrada estable mientras avanzan a una capa más profunda.

La estructura de la red comienza siendo una red plana similar inspirada en otros modelos como VGG con capas convolucionales seguidas sin capas de agrupación entre ellas. Al final de la parte de detección de características se coloca una agrupación promedio con una función de activación softmax. Esta red simple se modifica hasta pasar a ser una red residual gracias al añadido de conexiones de acceso directo que definen bloques residuales. Los datos de entrada son imágenes con tres canales de color y 180x180 píxeles.

Layer (type)	Output Shape	Param #	Connected to
input_2 (InputLayer)	[(None, 180, 180, 3)]	0	[]
conv1_pad (ZeroPadding2D)	(None, 186, 186, 3)	0	['input_2[0][0]']
conv1_conv (Conv2D)	(None, 90, 90, 64)	9472	['conv1_pad[0][0]']
conv1_bn (BatchNormalization)	(None, 90, 90, 64)	256	['conv1_conv[0][0]']
conv1_relu (Activation)	(None, 90, 90, 64)	0	['conv1_bn[0][0]']
pool1_pad (ZeroPadding2D)	(None, 92, 92, 64)	0	['conv1_relu[0][0]']
pool1_pool (MaxPooling2D)	(None, 45, 45, 64)	0	['pool1_pad[0][0]']
conv2_block1_1_conv (Conv2D)	(None, 45, 45, 64)	4160	['pool1_pool[0][0]']
conv2_block1_1_bn (BatchNormal ization)	(None, 45, 45, 64)	256	['conv2_block1_1_conv[0][0]']
⋮			
conv5_block3_2_bn (BatchNormal ization)	(None, 6, 6, 512)	2048	['conv5_block3_2_conv[0][0]']
conv5_block3_2_relu (Activatio n)	(None, 6, 6, 512)	0	['conv5_block3_2_bn[0][0]']
conv5_block3_3_conv (Conv2D)	(None, 6, 6, 2048)	1050624	['conv5_block3_2_relu[0][0]']
conv5_block3_3_bn (BatchNormal ization)	(None, 6, 6, 2048)	8192	['conv5_block3_3_conv[0][0]']
conv5_block3_add (Add)	(None, 6, 6, 2048)	0	['conv5_block2_out[0][0]', 'conv5_block3_3_bn[0][0]']
conv5_block3_out (Activation)	(None, 6, 6, 2048)	0	['conv5_block3_add[0][0]']
avg_pool (GlobalAveragePooling 2D)	(None, 2048)	0	['conv5_block3_out[0][0]']

Total params: 23,587,712
Trainable params: 0
Non-trainable params: 23,587,712

Figura 5-6. Primeras y últimas capas de la red ResNet50.

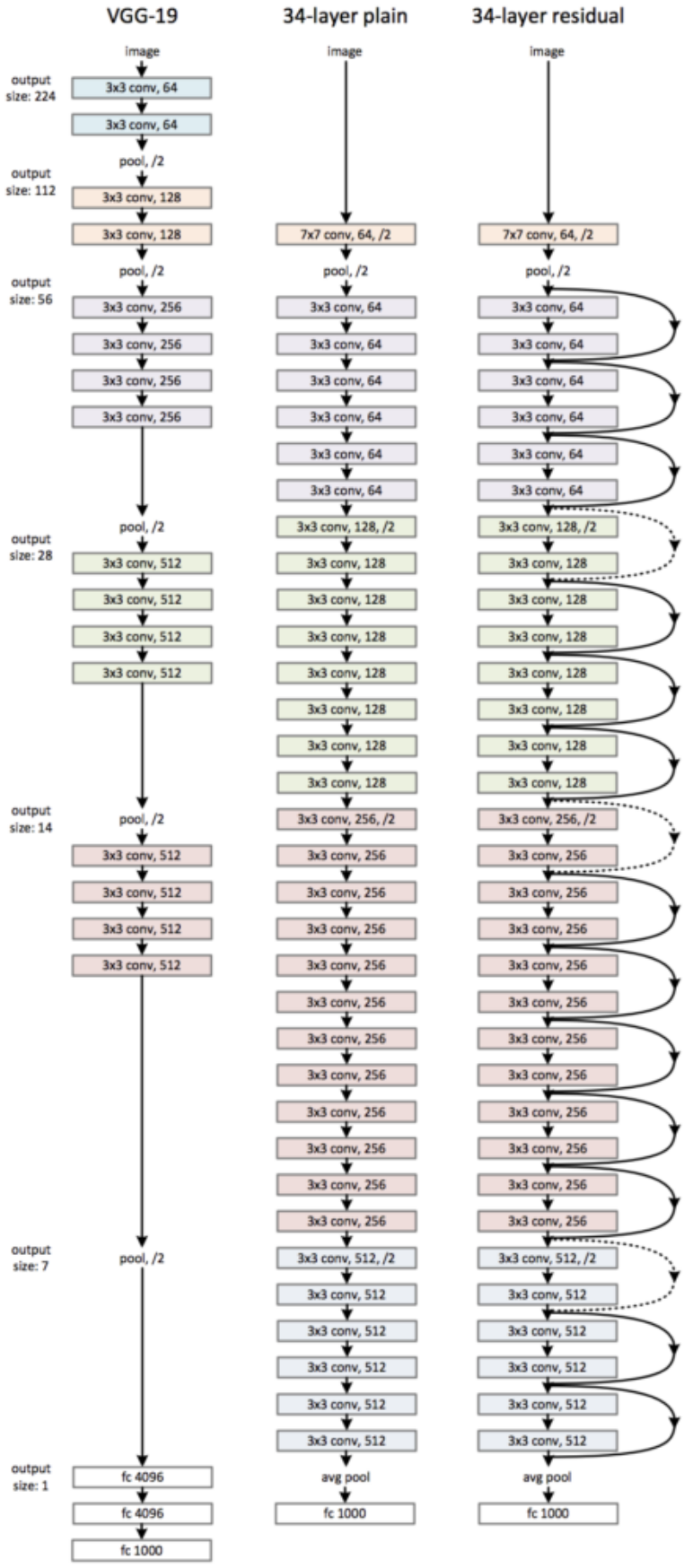

Figura 5-7. Comparación de izquierda a derecha de una red VGG, red simple de 34 capas y una red residual de 34 capas [26].

6 RESULTADOS

"I'd take the awe of understanding over the awe of ignorance any day."

- Douglas Adams -

En este capítulo vamos a entrenar los modelos neuronales convolucionales vistos anteriormente. Para analizar los efectos de distintos hiperparámetros sobre nuestra red vamos a entrenar cada modelo con distintos valores del *learning rate* y distintos números de épocas. Finalmente decidiremos qué conjunto de hiperparámetros da un resultado óptimo según las funciones de pérdida y la precisión conseguida.

Los tres modelos tienen una estructura distinta tanto en estructura como en profundidad, por lo que alteraremos ligeramente los hiperparámetros (*learning rate* y número de épocas) a la hora de entrenarlos. LeNet-5 es un modelo con poca profundidad y, por lo tanto, un número de parámetros a optimizar relativamente bajo. AlexNet es un modelo ligeramente más profundo que LeNet, por lo que comenzaremos con un mayor número de épocas. Además, disminuiremos los valores del *learning rate* ya que para valores mayores (menos precisos) se alcanza la sobreparametrización rápidamente. Finamente, el modelo ResNet50 es mucho más profundo que los anteriores, por lo que exigirá un entrenamiento con mayor número de épocas y un *learning rate* que aporte más precisión.

Modelo	***Learning rate***	**Épocas**
LeNet-5	0.1 – 0.01 – 0.001 – 0.0001	10 – 15 – 20 – 25
AlexNet	0.01 – 0.001 – 0.0001 – 0.00001	15 – 20 – 25 – 30
ResNet50	0.001 – 0.0001 – 0.00001 – 0.000001	30 – 35 – 40 – 45

Tabla 6-1. Conjunto de hiperparámetros utilizados para entrenar cada modelo.

Cada modelo se entrena un total de 16 veces combinando los posibles valores del *learning rate* y el número de épocas. A continuación, se mostrarán los resultados de la función de pérdida y la precisión obtenidos en la última época de cada conjunto tanto para el set de entrenamiento como el de validación. Mostraremos la gráfica del conjunto de hiperparámetros que consiga mejores resultados.

6.1 LeNet-5

Hiperparámetros:

- *Learning rate:* 0.1, 0.01, 0.001, 0.0001.
- Número de épocas: 10, 15, 20, 25.

LeNet-5			Set de entrenamiento		Set de validación	
Posición	**Learning rate**	**Épocas**	**Pérdida**	**Precisión (%)**	**Pérdida**	**Precisión (%)**
1	**0.1**	**10**	2.2992	13.85	2.3092	11.84
2		**15**	2.2998	10.82	2.3137	11.84
3		**20**	2.2968	11.69	2.3151	11.84
4		**25**	2.2911	11.69	2.3144	11.84
5	**0.01**	**10**	2.2828	13.85	2.3138	11.84
6		**15**	0.9750	64.07	1.3757	60.53
7		**20**	0.3818	90.04	2.1457	73.68
8		**25**	2.2834	13.85	2.3133	11.84
9	**0.001**	**10**	0.0099	100	0.7569	84.21
10		**15**	0.0017	100	0.5238	85.53
11		**20**	0.0017	100	0.4477	90.79
12		**25**	0.0010	100	0.5864	86.84
13	**0.0001**	**10**	1.4777	66.23	2.5827	48.68
14		**15**	0.8112	0.7879	1.8554	67.11
15		**20**	0.2552	94.37	1.4513	65.79
16		**25**	0.0999	99.13	1.1687	75.01

Tabla 6-2. Resultado del entrenamiento del modelo LeNet-5.

Los resultados obtenidos del entrenamiento de una red neuronal no pueden analizarse exclusivamente en función de un parámetro. En el caso del modelo a estudiar, podemos ver como la mejor precisión en el set de validación (90.79%) se consigue en la posición 11 con un *learning rate* de 0.001 durante 20 épocas. Sin embargo, en el set de entrenamiento se consigue una precisión del 100%, por lo que podemos asumir que se está produciendo sobreparametrización, es decir, el modelo acaba por memorizar las entradas del set de entrenamiento, por lo que perdería validez.

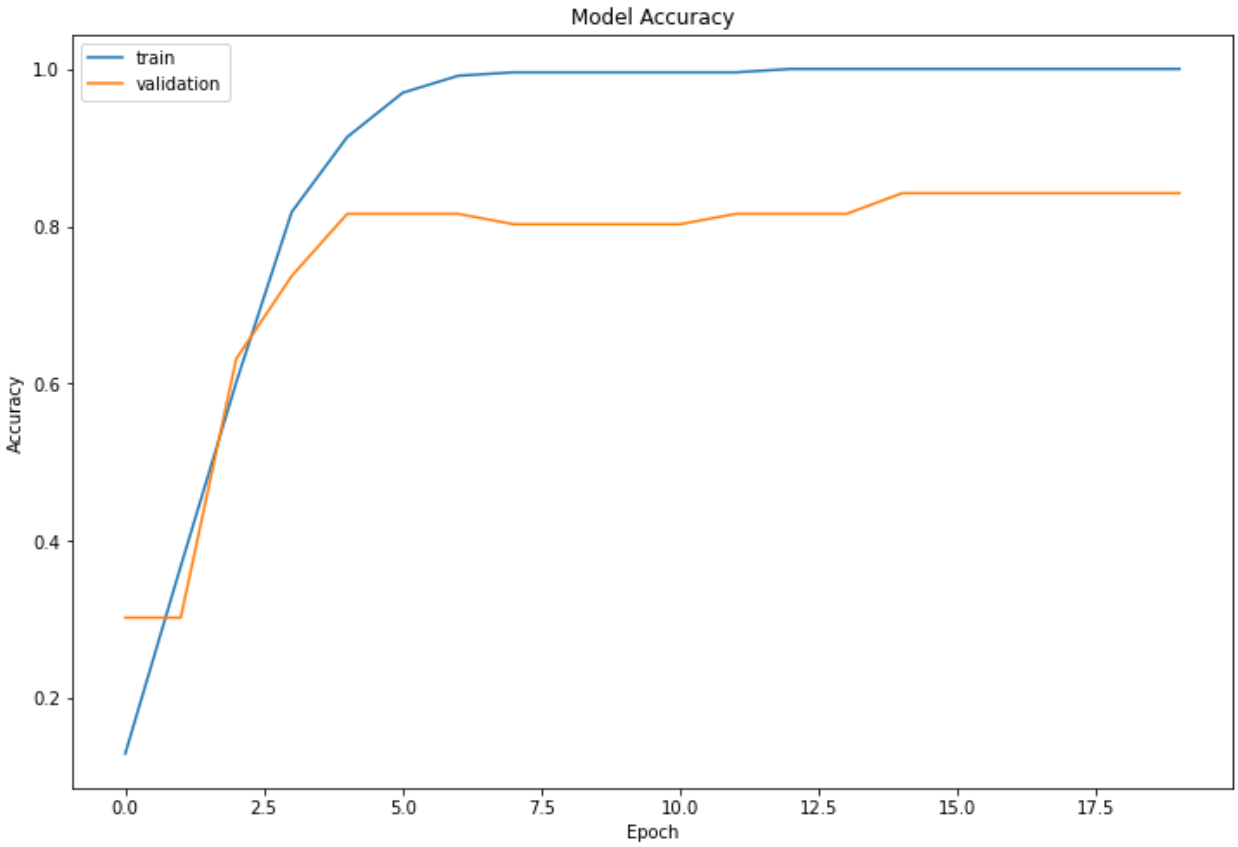

Figura 6-1. Precisión del modelo LeNet-5. *Learning rate* = 0.001, épocas = 20.

El otro parámetro para tener en cuenta es la función de pérdida (*los function*). Esta función evalúa la desviación de las predicciones realizadas por nuestra red neuronal y los valores reales de las observaciones usadas durante el entrenamiento. La función de pérdida debe ser tomada como un parámetro a minimizar para conseguir una mayor eficiencia de la red siempre sin perder de vista la posibilidad de sobreparametrización.

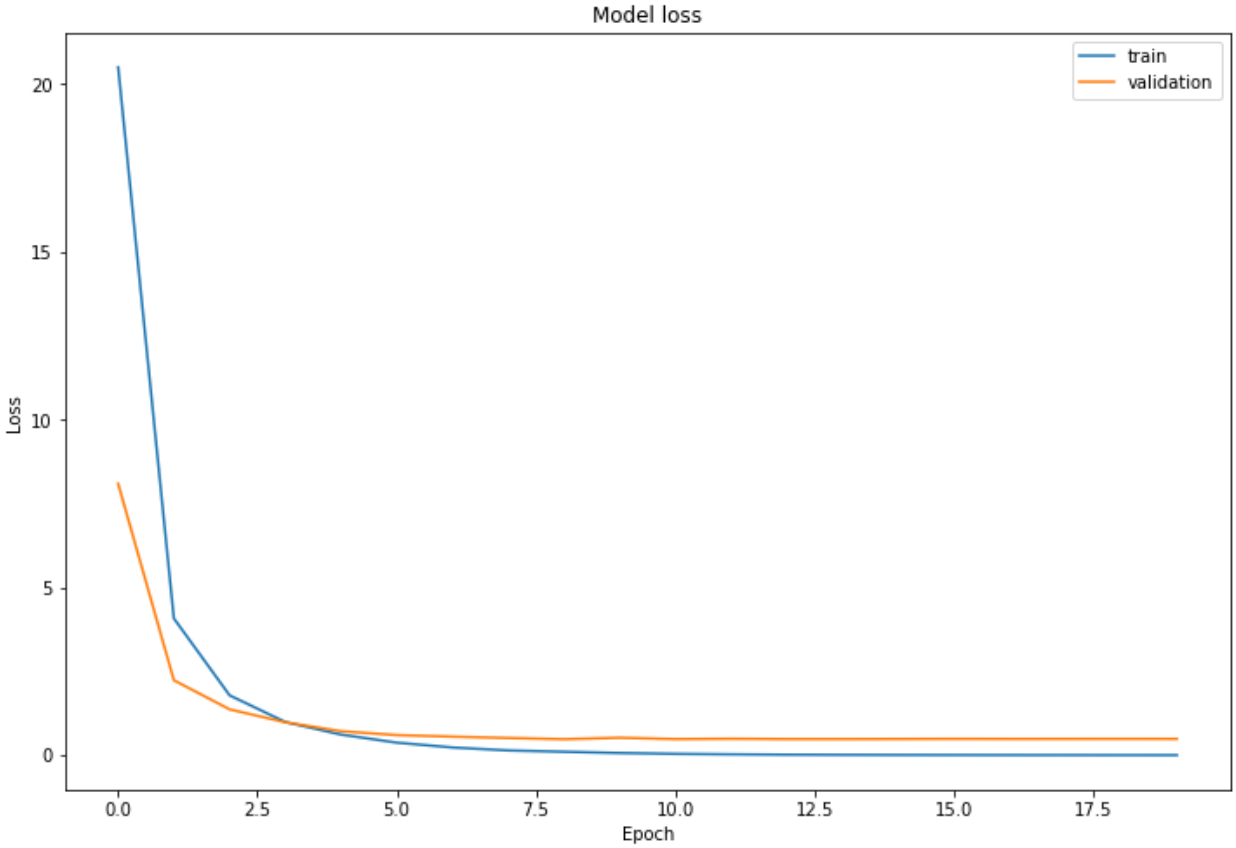

Figura 6-2. Pérdida del modelo LeNet-5. *Learning rate* = 0.001, épocas = 20.

El mejor resultado del set de validación sin padecer sobreparametrización lo encontramos en la posición 16 con un *learning rate* de 0.00001 durante 25 épocas (75.01%). Se trata de un resultado conservador y aún lejos del conseguido en el set de entrenamiento de dicha posición (90.04%) que podremos mejorar en próximos modelos.

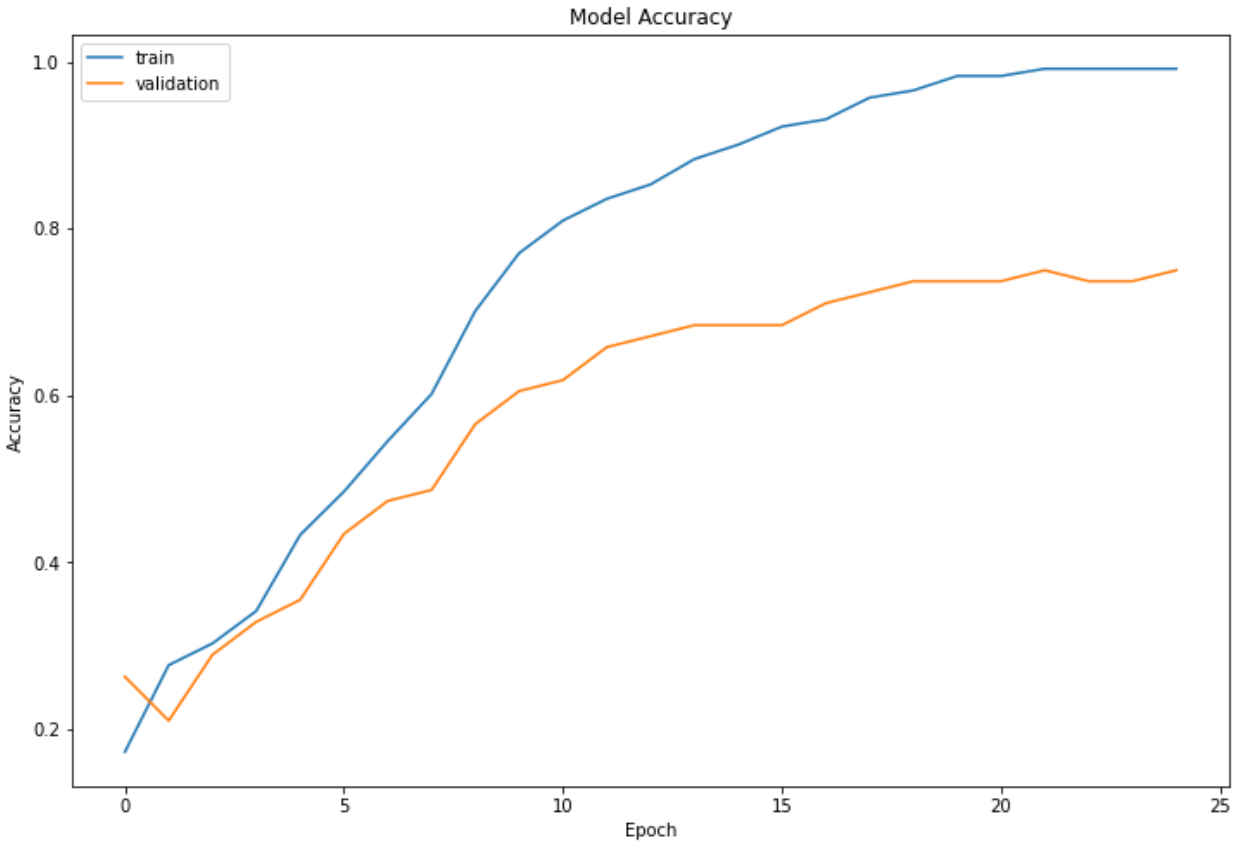

Figura 6-3. Precisión del modelo LeNet-5. *Learning rate* = 0.0001, épocas = 25.

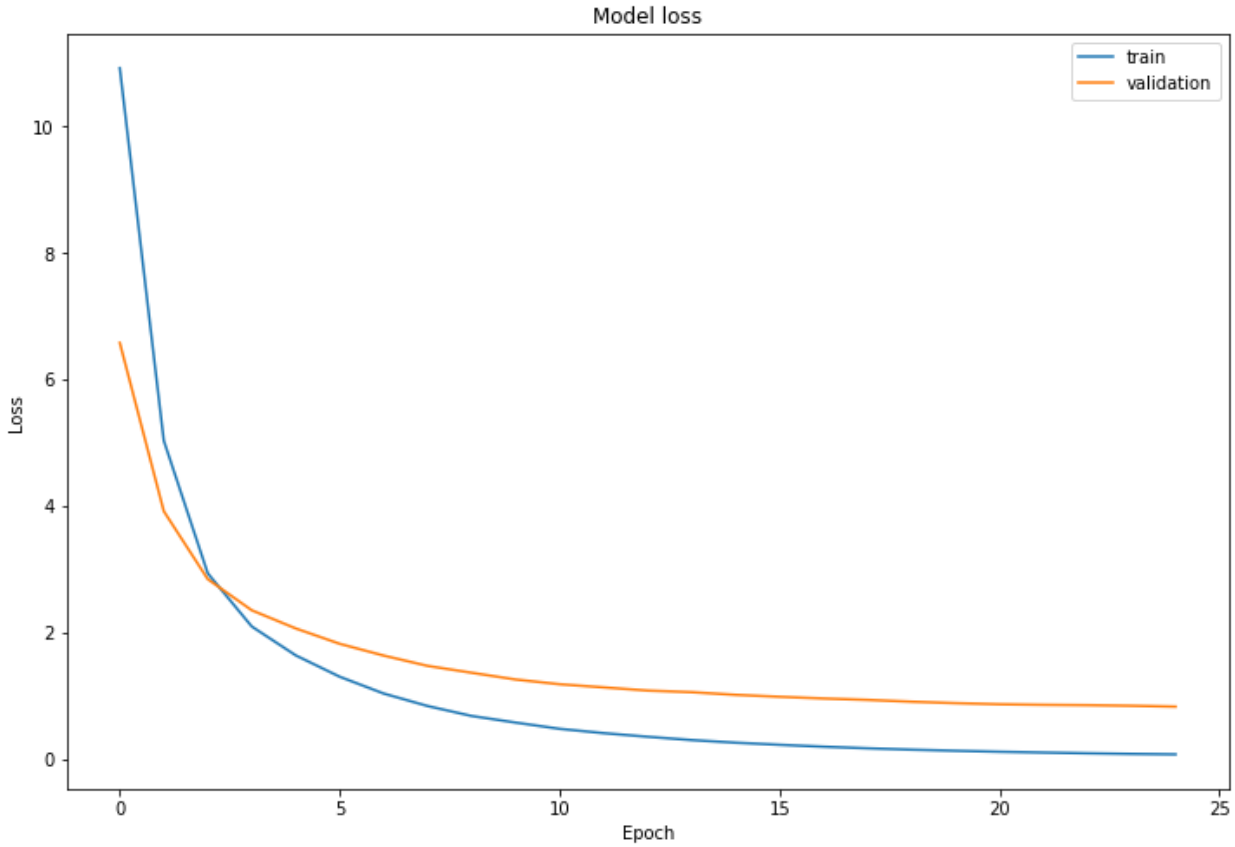

Figura 6-4. Pérdida del modelo LeNet-5. *Learning rate* = 0.0001, épocas = 25.

6.2 AlexNet

Hiperparámetros:

- *Learning rate:* 0.01, 0.001, 0.0001, 0.00001.
- Número de épocas: 15, 20, 25, 30.

AlexNet			Set de entrenamiento		Set de validación	
Posición	**Learning rate**	**Épocas**	**Pérdida**	**Precisión (%)**	**Pérdida**	**Precisión (%)**
1	**0.01**	**15**	0.0138	100	0.2195	96.05
2		**20**	0.0107	99.57	0.2747	94.74
3		**25**	0.0031	100	0.3254	96.05
4		**30**	0.0021	100	0.2850	96.05
5	**0.001**	**15**	0.0967	98.27	0.2258	94.74
6		**20**	0.0968	97.40	0.2212	93.42
7		**25**	0.0618	98.27	0.1953	94.74
8		**30**	0.0245	100	0.1589	96.05
9	**0.0001**	**15**	1.2251	64.50	1.0703	80.26
10		**20**	0.8773	71.86	0.6505	90.79
11		**25**	0.7329	77.49	0.4624	88.16
12		**30**	0.5295	82.68	0.3544	92.11
13	**0.00001**	**15**	3.8897	17.75	2.2517	21.05
14		**20**	3.7216	19.48	2.0958	30.26
15		**25**	3.8978	17.75	2.0539	28.95
16		**30**	3.7319	18.18	2.1039	28.95

Tabla 6-3. Resultado del entrenamiento del modelo AlexNet.

El modelo AlexNet obtiene mejores resultados que el anterior. En la posición 7, con un *learning rate* de 0.001 y 25 épocas conseguimos una precisión de 94.74% en el set de validación. Este resultado se asemeja más al obtenido en el set de entrenamiento (98.27%) y mejora la función de pérdida obtenida en la red AlexNet.

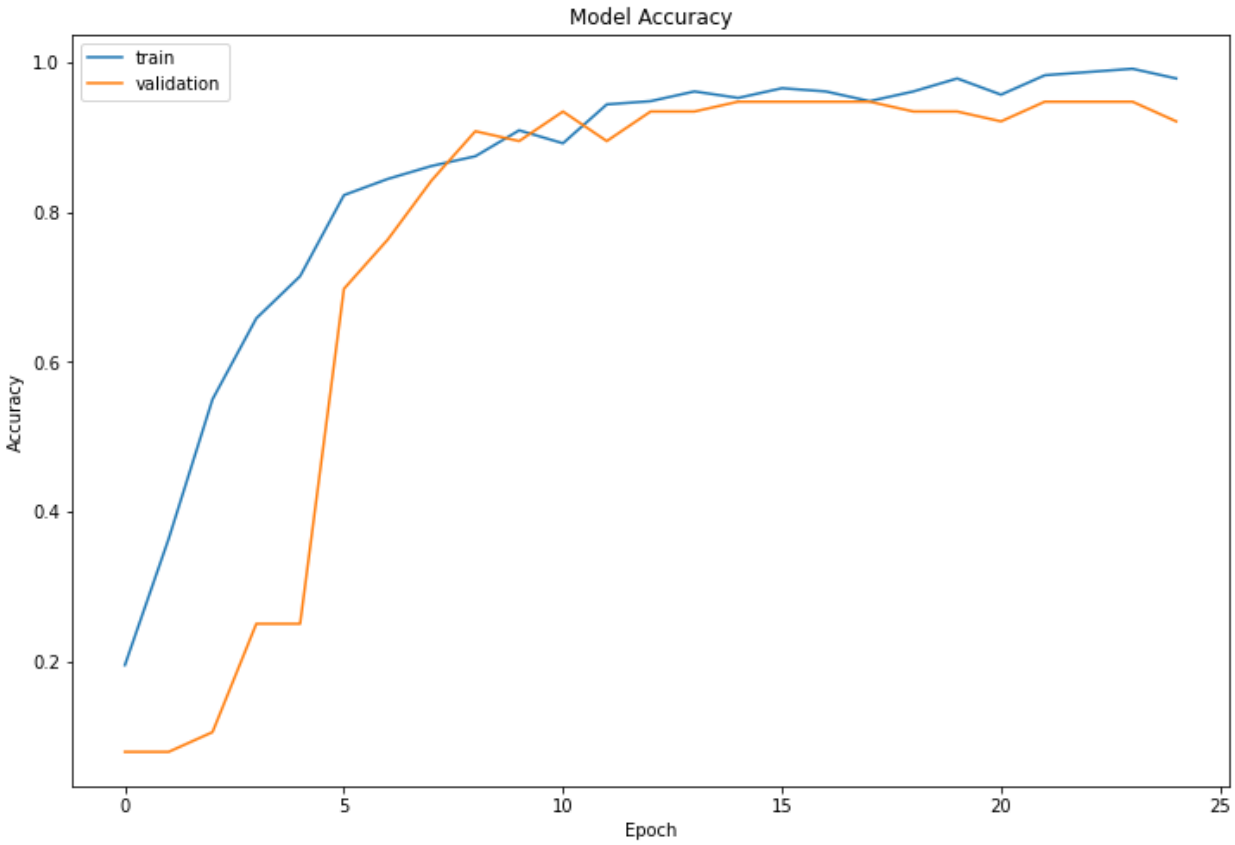

Figura 6-5. Precisión del modelo AlexNet. *Learning rate* = 0.001, épocas = 25.

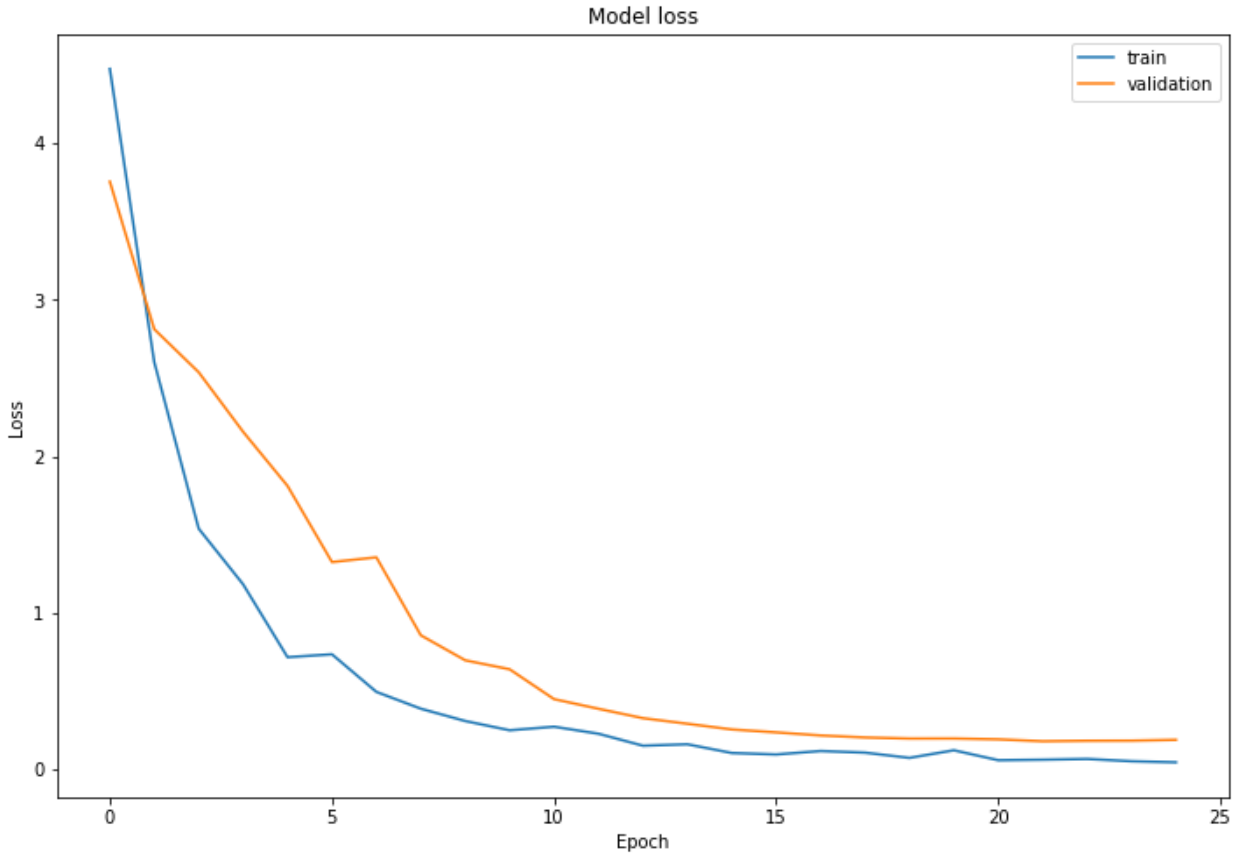

Figura 6-6. Pérdida del modelo AlexNet. *Learning rate* = 0.001, épocas = 25.

6.3 ResNet

Hiperparámetros:

- *Learning rate:* 0.001, 0.0001, 0.00001, 0.000001.
- Número de épocas: 30, 35, 40, 45.

ResNet			Set de entrenamiento		Set de validación	
Posición	**Learning rate**	**Épocas**	**Pérdida**	**Precisión (%)**	**Pérdida**	**Precisión (%)**
1	**0.001**	**20**	0.0008	100	0.1931	96.05
2		**25**	0.0006	100	0.1755	96.05
3		**30**	0.0004	100	0.1915	96.05
4		**35**	0.0005	100	0.1473	96.05
5	**0.0001**	**20**	0.0181	100	0.1449	96.05
6		**25**	0.0125	100	0.1276	96.05
7		**30**	0.0091	100	0.1236	96.05
8		**35**	0.0079	100	0.1268	96.05
9	**0.00001**	**20**	0.6211	94.37	0.7185	89.47
10		**25**	0.5184	97.40	0.6363	92.11
11		**30**	0.3780	99.57	0.4699	88.16
12		**35**	0.3059	99.57	0.4022	96.03
13	**0.000001**	**20**	2.4999	9.96	2.5025	13.16
14		**25**	1.9744	35.93	2.0046	34.21
15		**30**	2.1185	31.60	2.1391	31.58
16		**35**	2.1968	24.67	2.3385	25.00

Tabla 6-4. Resultado del entrenamiento del modelo ResNet.

Como podemos ver en la tabla 6-3, el modelo ResNet50 consigue resultados ligeramente superiores al modelo AlexNet. En la posición 12 obtiene una precisión en el set de validación del 96.03% (para un *learning rate* de 0.00001 y 35 épocas) frente al 94.74% conseguido en el modelo AlexNet. Sin embargo, estos resultados podrían alternarse si aumentásemos la base de datos y, por lo tanto, la precisión de ambos.

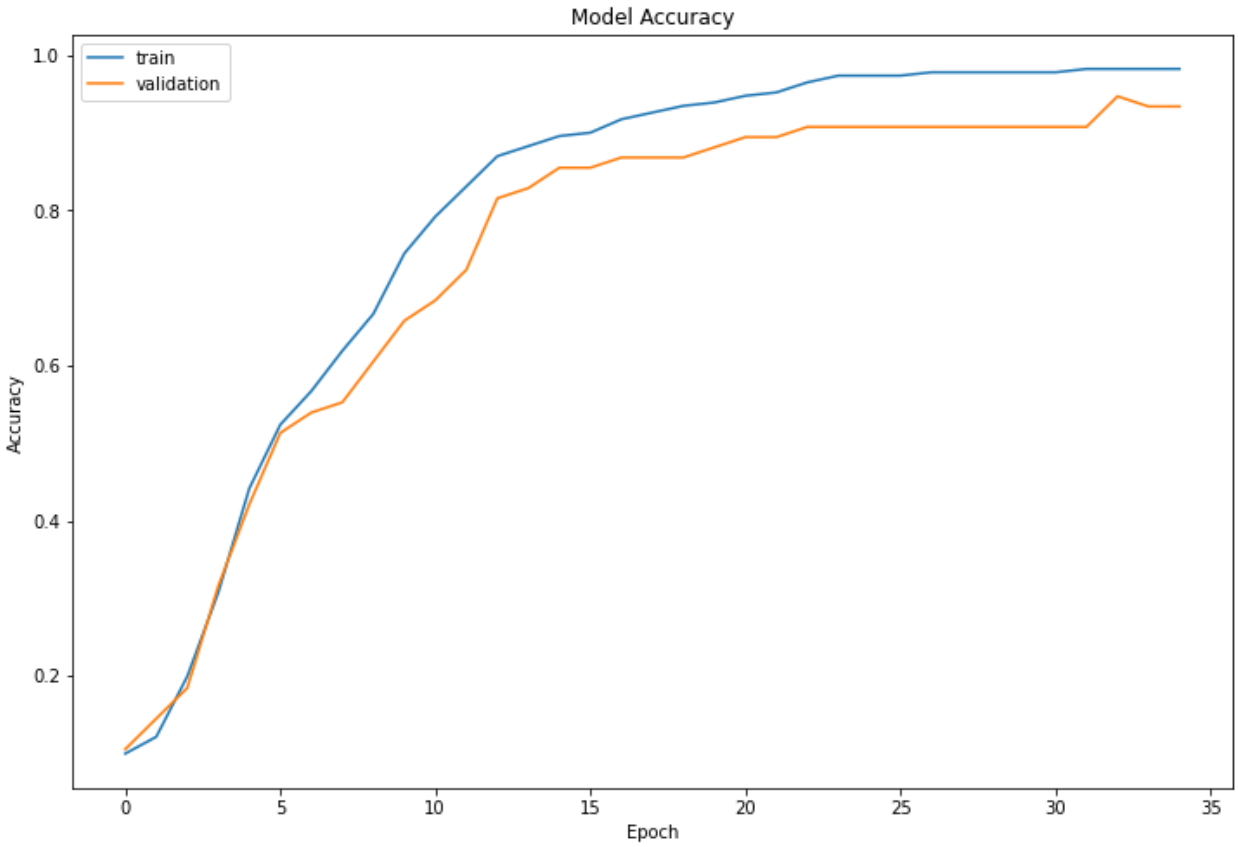

Figura 6-7. Precisión del modelo ResNet50. *Learning rate* = 0.00001, épocas = 35.

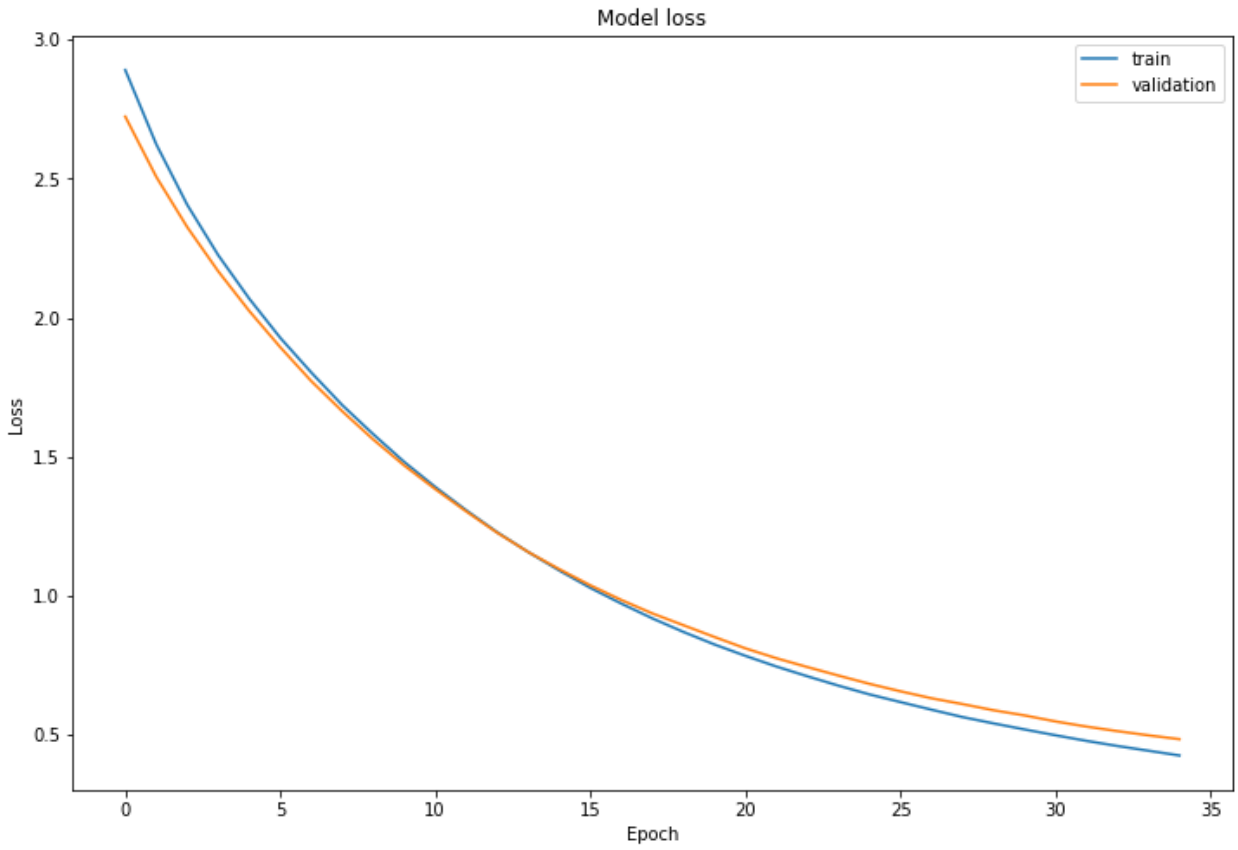

Figura 6-8. Pérdida del modelo ResNet50. Learning rate = 0.00001, épocas = 35.

7 CONCLUSIONES

"We can build a much brighter future where humans are relieved of menial work using AI capabilities."

- Andrew NG -

El objetivo de este proyecto era realizar una aproximación al aprendizaje profundo y, más concretamente, a las redes neuronales convolucionales. Para ello se ha centrado en la resolución de un problema de visión artificial. Paralelamente, se ha realizado una comparativa entre distintos modelos de redes neuronales convolucionales para analizar sus resultados frente al problema propuesto tanto en el entrenamiento como en la realidad. Por otra parte, el proyecto a sido de gran utilidad para iniciarse en un lenguaje de programación tan extendido como Python, así como para aprender acerca de las librerías con un uso más extendido en el campo de la inteligencia artificial.

Tras entrenar los tres modelos propuestos sobre la misma base de datos se han encontrado distintas variaciones de hiperparámetros que aportan buenos resultados.

Modelos	Set de entrenamiento	Set de validación
LeNet-5	99.13%	75.01%
AlexNet	98.27%	94.74%
ResNet50	99.57%	96.03%

Tabla 7-1. Comparativa entre los distintos modelos.

Con estos resultados podemos llegar a varias conclusiones interesantes acerca del uso y entrenamiento de los modelos escogidos frente a una base de datos de tamaño limitado. En primer lugar, podemos apreciar que el modelo LeNet-5 aporta los peores resultados de las tres. Aunque este modelo está demostrado que puede alcanzar más del 99.9% de acierto en el campo del reconocimiento de caracteres, al entrenarse con una base de datos pequeña y tener pocos parámetros a entrenar, alcanza cerca del 75% de precisión en el set de validación antes de producirse la sobreparametrización. En segundo lugar, comprobamos que una capa más profunda, como AlexNet o ResNet50, mejoran notablemente este resultado alcanzando el 95% de precisión en el set de validación. Cabe mencionar que, aunque ambas obtienen un resultado parecido, ResNet50 está menos expuesta a la sobreparametrización debido a su arquitectura de bloques residuales. Sin embargo, para definir con mayor fiabilidad cuál de los dos modelos es más eficiente sería necesaria una base de datos más extensa.

Podemos concluir que los resultados obtenidos han cumplido con los principales objetivos de manera

satisfactoria, dejando espacio a ciertas mejoras y avances en lo relativo tanto de la herramienta de recorte como del modelado de redes neuronales profundas.

7.1 Futuras mejoras

Como se ha mencionado anteriormente, podrían realizarse ciertas mejoras para implementar mejores modelos o alterar los ya existentes para obtener resultados más precisos. Algunas de las mejoras a implementar serían:

- Continuar realizando pruebas con los modelos escogidos alterando con mayor exactitud los hiperparámetros para conseguir mejorar los resultados ya obtenidos.
- Crear una red neuronal con una estructura propia. Para ello deben escogerse el número de capas, el tamaño de los lotes de entrenamiento o los optimizadores entre otros hiperparámetros. La elección de tantos factores complica notablemente el proceso de optimización, pero permite un resultado más personalizado en ciertos casos.
- Aumentar la base de datos. Una posible solución sería utilizar bases de datos ya creadas como MNIST, sin embargo, los resultados serían más completos en el caso de una base de datos propia. En nuestro caso, podría colocarse una cámara estática en la entrada o salida de un estacionamiento para tomar los datos con las condiciones más cercanas al caso real.
- Utilizar otros modelos de redes neuronales. Una vez mejorada la base de datos podríamos probar a entrenar modelos con mayor número de capas o más complejos. El primer modelo que probar debería ser el ya mencionado YOLO (*You Only Look Once*). Este modelo permite reconocer con alta fiabilidad todo tipo de formas y figuras en una imagen y es capaz de reconocer caracteres separados en la misma.
- Por último, podríamos localizar los casos en los que el modelo falla para analizar el motivo y poder corregir los fallos con más precisión.

Es importante mencionar que el campo de la inteligencia artificial, y más de la visión artificial, se encuentra en un momento de continuo cambio con procesos de mejora constantes, por lo que en cuestión de meses podrían aparecer nuevos modelos que mejoren los ya existentes o simplifiquen su diseño.

REFERENCIAS

[1] L. C. Aiello, «The multifaceted impact of Ada Lovelace in the digital age,» *Artificial Intelligence,* vol. 235, pp. 58-62, 2016.

[2] A. M. Turing, «Computing machinery and intelligence,» *MIND,* vol. 49, pp. 433-460, 1950.

[3] J. Moore, «The Dartmouth College Artificial Intelligence Conference: The Next Fifty Years,» *AI Magazine,* vol. 27, nº 4, 2006.

[4] S. Natale, «If software is narrative: Joseph Weizenbaum, artificial intelligence and the biographies of ELIZA,» *Sage Journals,* vol. 21, nº 3, 2019.

[5] OpenAI, «chatgpt,» 2022. [En línea]. Available: https://openai.com/blog/chatgpt.

[6] OpenAI, «DALL-E,» 2020. [En línea]. Available: https://openai.com/product/dall-e-2.

[7] D. G. d. l. P. Ramírez, «www.youtube.com/@DotCSV,» 1 Noviembre 2017. [En línea]. Available: https://www.youtube.com/watch?v=KytW151dpqU&list=PL-Ogd76BhmcC_E2RjgIIJZd1DQdYHcVf0&ab_channel=DotCSV.

[8] J. Luna Gonzalez, «Medium,» 8 febrero 2018. [En línea]. Available: https://medium.com/soldai/tipos-de-aprendizaje-autom%C3%A1tico-6413e3c615e2.

[9] A. NG, «DeepLearning.AI, Deep Learning Specialization, Neural Network and Deep Learning,» [En línea]. Available: https://www.coursera.org/specializations/deep-learning.

[10] A. Lenail, «NN-SVG,» [En línea]. Available: http://alexlenail.me/NN-SVG/.

[11] B. Ding, H. Qian y J. Zhou, «Activation functions and their characteristics in deep neural networks,» de *2018 Chinese Control And Decision Conference (CCDC)*, 2018, pp. 1836-1841.

[12] A. NG, «DeepLearning.AI, Deep Learning Specialization, Improving Deep Neural Networks: Hyperparameter Tuning, Regularization, and Optimization,» [En línea]. Available: https://www.coursera.org/specializations/deep-learning.

[13] A. NG, «Deeplearning.AI, Convolutional Neural Networks,» [En línea]. Available: https://www.coursera.org/specializations/deep-learning.

[14] M. Mahdi Bejani y M. Ghatee, «A systematic review on overfitting control in shallow and deep neural networks,» *Artificial Intelligence Review,* vol. 54, pp. 6391-6438, 2021.

[15] J. I. Bagnato, «¿Cómo funcionan las Convolutional Neural Networks? Visión por Ordenador,» de *Aprende Machine Learning*, 2018.

[16] Matlab, «MathWorks,» mayo 2022. [En línea]. Available: https://es.mathworks.com/products/matlab/matlab-vs-python.html. [Último acceso: enero 2023].

[17] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu y X. Zheng, TensorFlow: A system for large-scale machine learning, 2016.

[18] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani y Chila, PyTorch: An Imperative Style, High-Performance Deep Learning Library, 2019.

[19] A. NG, «Deep Learning Specialization, Sequence Models,» [En línea]. Available: https://www.deeplearning.ai/courses/deep-learning-specialization/.

[20] V. Arevalo, J. González-Jiménez y G. Ambrosio, «OpenCV. La Librería Open Source de Visión Artificial,» *Linux Free Magazine,* vol. 10, pp. 141-147, 2005.

[21] L. Deng, «The mnist database of handwritten digit images for machine learning research,» *IEEE Signal Processing Magazine,* vol. 29, nº 6, pp. 141-142, 2012.

[22] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li y L. Fei-Fei, «ImageNet: A large-scale hierarchical image database,» de *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009.

[23] Y. B. L. B. Y. a. H. P. LeCun, «Gradient-based learning applied to document recognition,» *Proceedings of the IEEE,* vol. 86, p. 2278–2324, 1999.

[24] A. Krizhevsky, I. Sutskever y G. E. Hinton, «ImageNet Classification with Deep Convolutional,» *Communications of the ACM,* vol. 60, nº 6, pp. 84-90, 2012.

[25] K. Simonyan y A. Zisserman, «Very Deep Convolutional Networks for Large-Scale Image Recognition,» de *International Conference on Learning Representations*, 2015.

[26] K. He, X. Zhang, S. Ren y J. Sun, «Deep Residual Learning for Image Recognition,» de *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, USA, 2016, pp. 770-778.

[27] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke y A. Rabinovich, «Going deeper with convolutions,» de *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Boston, MA, USA, 2015, pp. 1-9.

[28] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto y H. Adam, «MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications,» *CoRR,* 2017.

GLOSARIO

OCR: Reconocimiento óptico de caracteres

IA/AI: Inteligencia Artificial

DL: Deep Learning

ML: Machine Learning

b: Sesgo , bias

w: Peso/importancia de la neurona

lr: Learning rate

ReLU: Rectified Linear Unit

API: Application Programming Interfaces

CNN: Convolutional Neural Network

BS: Batch Size

log: Logaritmo en base 2

ResNet: Residual Network

FCNN: Full Convolutional Neural Network

CRNN: Convolutional Recurrent Neural Networks

R-CNN: Regional CNN

ANEXO A

Código correspondiente a la función de recorte de matrículas.

```
'''
José E. Maese Álvarez.
TFG: Uso de redes neuronales para identificación de matrículas.
Funciones de creacion de base de datos
'''

import cv2
import numpy as np
import matplotlib.pyplot as plt

def RecorteMatricula(img):
    #1. Cargar imagen
    imagColor = cv2.imread('C:/Users/josen/OneDrive/Documents/Python
Scripts/TFG/Imagenes/Coches/' + str(img) + '.jpg')
    imagColor = cv2.resize(imagColor, (864, 864) )

    #2. Cargar imagen en escala de grises
    imagGris = cv2.cvtColor(imagColor, cv2.COLOR_BGR2GRAY)

    #3. Filtro de reduccion de ruido
    imagFiltrada = cv2.GaussianBlur(imagGris, (5, 5), 0)
    cv2.imwrite('C:/Users/josen/OneDrive/Documents/Python
Scripts/TFG/Imagenes/Pruebas/' + str(img) + '_filtro_gauss.jpg',
imagFiltrada)

    #4. DetecciÃ³n de bordes con Canny. Incluye algoritmo Sobel,
supresion de
    # no-max (pixeles fuera del borde) y umbral por histeresis.
    imagBordes = cv2.Canny(imagFiltrada, 30, 200)
    imagBordes = cv2.dilate(imagBordes, None, iterations=1)
    cv2.imwrite('C:/Users/josen/OneDrive/Documents/Python
Scripts/TFG/Imagenes/Pruebas/' + str(img) + 'bordes.jpg', imagBordes)


    #5. Localizacion de contornos y creacion de rectangulos en cada
uno
    (contornos, jerarquia) = cv2.findContours(imagBordes,
cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
    #Ordena y dibuja solo los 10 contornos mÃ¡s grandes
    contornos = sorted(contornos, key = cv2.contourArea, reverse =
True)[:10]
    #6. Escoger el rectangulo correspondiente a la matricula:
    matrizDetectada = 0
    for c in contornos:
       area = cv2.contourArea(c)
       epsilon = 0.02*cv2.arcLength(c, True)
       approx = cv2.approxPolyDP(c, epsilon, True)
       # vertices = len(approx)
       x,y,w,h = cv2.boundingRect(approx)
       aspect_ratio = float(w)/(h)
```

```
        if (aspect_ratio > 3.6 and aspect_ratio < 6):
            if area>5000 and area<18000:
                matrizDetectada = 1
                break

    #7. Recortamos la matricula
    if matrizDetectada == 1:

        # Masking the part other than the number plate
        mask = np.zeros(imagGris.shape, np.uint8)
        new_image = cv2.drawContours(mask, [approx], 0, 255, -1,)
        new_image = cv2.bitwise_and(imagGris, imagGris, mask=mask)


        #Recortamos la matricula
        (x, y) = np.where(mask == 255)
        (topx, topy) = (np.min(x), np.min(y))
        (bottomx, bottomy) = (np.max(x), np.max(y))
        imagMatricula = imagGris[topx:bottomx+1, topy:bottomy+1]

        cv2.imwrite('C:/Users/josen/OneDrive/Documents/Python
Scripts/TFG/Imagenes/Pruebas/' + str(img) + '.jpg',
imagMatricula)
    else:
        print('Error al detectar la matricula del coche ' +str(img))
```

ANEXO B

Código correspondiente a la función de recorte de caracteres.

```
'''
José E. Maese Álvarez.
TFG: Uso de redes neuronales para identificación de matrículas.
Funciones de creacion de base de datos
'''

import cv2
import numpy as np
import matplotlib.pyplot as plt

def RecorteCaracteres(img, alpha1, alpha2, beta, limite_binarizado,
tam):
        imagMat = cv2.imread('C:/Users/josen/OneDrive/Documents/Python
Scripts/TFG/Imagenes/Matriculas/' + str(img) + '.jpg')
        if imagMat is None:
            print('Error al recortar la matricula del coche ' +
str(img))
            return
        height = 64
        width = 256

        # Aplicamos filtros a la imagen y binarizamos
        imagColor = cv2.resize(imagMat, (width, height) )
        imagGris = cv2.cvtColor(imagColor, cv2.COLOR_BGR2GRAY)
        imagFiltrada = cv2.GaussianBlur(imagGris, (5, 5), 0)
        ret, imagBin = cv2.threshold(imagFiltrada, limite_binarizado,
255, cv2.THRESH_BINARY_INV)
        imagBin[:, 0:5]=255
        imagMatriz = np.asarray(imagBin)

        plt.figure(1)
        plt.imshow(imagMatriz,cmap = 'gray')
        plt.close(1)

        grad = np.sum(imagMatriz, axis=0)/(255*height)
        for i in range(5,256):
            grad[i] = alpha1*grad[i] + (1 - alpha1)*grad[i-2]

        # Creamos el limite inferior que separa cada caracter
utilizando su nivel de blanco en cada columna
        x = np.array(range(0, 256))
        y = grad.copy()
        limCaracter = grad.copy()
        limCaracter[:] = 0

        for i in range(30, 226):
            limCaracter[i] = np.amin(y[i-30:i+30]) + beta
        limCaracter[0:30] = np.amin(y[0:30]) + beta
        limCaracter[225:256] = np.amin(y[225:256]) + beta
```

```
        alpha2 = 0.6
        for i in range(5,256):
            limCaracter[i] = alpha2*limCaracter[i] + (1 -
alpha2)*limCaracter[i-2]

        plt.figure(2)
        plt.plot(x, y, color="black")
        plt.plot(x, limCaracter, color="red")
        #plt.title('Matricula '+str(img))
        plt.xlabel('Columnas de la imagen')
        plt.ylabel('Nivel de blanco')
        plt.savefig('C:/Users/josen/OneDrive/Documents/Python
Scripts/TFG/Imagenes/Pruebas/hist.jpg', facecolor='w',
bbox_inches="tight", pad_inches=0.3, transparent=True)

        #imagHist = np.asarray(x,y)
        #plt.show(imagHist)
       # plt.savefig('C:/Users/josen/OneDrive/Documents/Python
Scripts/TFG/Imagenes/Pruebas/imagHist', facecolor='w',
bbox_inches="tight", pad_inches=0.3, transparent=True)
        plt.close(2)

        #Obtenemos la posicion de cada caracter diferenciando entre
letra y numero
        j = 0
        flag = 1     #No esta en caracter
        inicioCaracter = np.zeros(8)
        finCaracter = np.zeros(8)
        margen = 3
        for i in range(256):
            if (grad[i] > limCaracter[i] and flag == 0):
                flag = 1
                inicioCaracter[j] = i - margen
                #print('Inicio caracter en ' + str(inicioCaracter[j]))
            if (grad[i] < limCaracter[i] and flag == 1):
                flag = 0
                finCaracter[j] = i + margen
                #print('Fin caracter en ' + str(finCaracter[j]))
                j = j + 1
                if j == 8:
                    break

        #Recorto cada caracter
        inicioCaracterX = np.zeros(7)
        finCaracterX = np.zeros(7)
        inicioCaracterY = np.zeros(7)
        finCaracterY = np.zeros(7)
        caracter = np.zeros(7)

        for i in range(7):
            inicioCaracterX[i] = inicioCaracter[i+1]
            finCaracterX[i] = finCaracter[i+1]
            inicioCaracterY[i] = 3
            finCaracterY[i] = 60
            # print(str(finCaracterX[i]))
            caracter =
imagGris[int(inicioCaracterY[i]):int(finCaracterY[i]),
```

```
int(inicioCaracterX[i]):int(finCaracterX[i])]
            if finCaracterX[i] == 0:
                break
            caracter = cv2.resize(caracter, (tam, tam))
            cv2.imwrite('C:/Users/josen/OneDrive/Documents/Python
Scripts/TFG/Imagenes/Caracteres/' + str(img) + '_' + str(i) + '.jpg',
caracter)
```

ANEXO C

Código correspondiente a la función del modelo LeNet-5.

```
'''
José E. Maese Álvarez.
TFG: Uso de redes neuronales para identificación de matrículas.
Funciones de LeNet-5
'''
import matplotlib.pyplot as plt
import pandas as pd
import cv2
import funciones as fun
import keras
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Flatten
from keras.layers import Dense
from keras.models import load_model
import tensorflow as tf
from tensorflow.keras.preprocessing import
image_dataset_from_directory
import os
os.environ['KMP_DUPLICATE_LIB_OK']='True'

def Lenet(learn, epocas):
    BATCH_SIZE = 16
    IMG_SIZE = (32, 32)
    directory = "Imagenes/Numeros/"
    train_dataset =
image_dataset_from_directory(directory='Imagenes/Numeros',
                                                shuffle=True,
                                                color_mode='grayscale'
,
                                                batch_size=BATCH_SIZE,
                                                image_size=IMG_SIZE,
                                                validation_split=0.25,
                                                subset='training',
                                                seed = 1)

    test_dataset =
image_dataset_from_directory(directory='Imagenes/Numeros',
                                               shuffle=True,
                                               batch_size=BATCH_SIZE,
                                               color_mode='grayscale',
                                               image_size=IMG_SIZE,
                                               validation_split=0.25,
                                               subset='validation',
                                               seed = 1)


    AUTOTUNE = tf.data.experimental.AUTOTUNE
    train_dataset = train_dataset.prefetch(buffer_size=AUTOTUNE)
    test_dataset = test_dataset.prefetch(buffer_size=AUTOTUNE)
```

```
    data_augmentation = fun.data_augmenter()

        ##############################################################
#        ##############################################################
##

    model = Sequential()
    model.add(Conv2D(filters = 6, kernel_size = 5, strides = 1,
activation = 'relu', input_shape = (32,32,1)))
    model.add(MaxPooling2D(pool_size = 2, strides = 2))
    model.add(Conv2D(filters = 16,  kernel_size = 5, strides = 1,
activation = 'relu', input_shape = (14,14,6)))
    model.add(MaxPooling2D(pool_size = 2, strides = 2))
    model.add(Flatten())
    model.add(Dense(units = 120, activation = 'relu'))
    model.add(Dense(units = 84, activation = 'relu'))
    model.add(Dense(units = 10, activation = 'softmax'))


    # conv_model = convolutional_model((32, 32, 1))
    opt = keras.optimizers.Adam(learning_rate=learn)
    model.compile(optimizer = opt, loss =
'sparse_categorical_crossentropy', metrics = ['accuracy'])
    model.summary()
    history = model.fit(train_dataset, epochs=epocas,
validation_data=test_dataset)
    model.save("LeNet_Model.h5")

#####################################################################
#####################################################################


    df_loss_acc = pd.DataFrame(history.history)

    plt.figure(1)
    df_loss= df_loss_acc[['loss','val_loss']]
    df_loss.rename(columns={'loss':'train','val_loss':'validation'},i
nplace=True)
    df_loss.plot(title='Model
loss',figsize=(12,8)).set(xlabel='Epoch',ylabel='Loss')
    plt.savefig('C:/Users/josen/OneDrive/Documents/Python
Scripts/TFG/Archivos/Imagenes/Lenet/Loss: ' + str(learn) + ', ' +
str(epocas) +'.jpg', facecolor='w', bbox_inches="tight",
pad_inches=0.3, transparent=True)
    plt.close(1)

    plt.figure(2)
    df_acc= df_loss_acc[['accuracy','val_accuracy']]
    df_acc.rename(columns={'accuracy':'train','val_accuracy':'validat
ion'},inplace=True)
    df_acc.plot(title='Model
Accuracy',figsize=(12,8)).set(xlabel='Epoch',ylabel='Accuracy')
    plt.savefig('C:/Users/josen/OneDrive/Documents/Python
Scripts/TFG/Archivos/Imagenes/Lenet/Acc: ' + str(learn) + ', ' +
str(epocas) +'.jpg', facecolor='w', bbox_inches="tight",
pad_inches=0.3, transparent=True)
    plt.close(2)
```

```
    LeNet_pred = model.predict(test_dataset)
    resultados_Lenet = df_loss_acc.to_numpy()

    return (resultados_Lenet[-1,0], resultados_Lenet[-1,1],
resultados_Lenet[-1,2], resultados_Lenet[-1,3])
```

ANEXO D

Código correspondiente a la función del modelo AlexNet.

```
'''
José E. Maese Álvarez.
TFG: Uso de redes neuronales para identificación de matrículas.
Funciones de AlexNet
'''
import matplotlib.pyplot as plt
import pandas as pd
import cv2
import numpy as np
import funciones as fun
import keras
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPool2D
from keras.layers import Flatten
from keras.layers import Dense
from keras.layers import BatchNormalization
from keras.layers import Dropout
from keras.models import load_model
from keras.utils.np_utils import to_categorical
import tensorflow as tf
from tensorflow.keras.preprocessing import
image_dataset_from_directory
import os
os.environ['KMP_DUPLICATE_LIB_OK']='True'

# learn = 0.001
# epocas = 25
def AlexNet(learn, epocas):
    BATCH_SIZE = 16
    IMG_SIZE = (227, 227)
    train_dataset = image_dataset_from_directory(directory =
'Imagenes/Numeros',
                                                 shuffle=True,
                                                 color_mode = 'rgb',
                                                 batch_size=BATCH_SIZE,
                                                 image_size=IMG_SIZE,
                                                 validation_split=0.25,
                                                 subset='training',
                                                 seed = 1)

    test_dataset = image_dataset_from_directory(directory =
'Imagenes/Numeros',
                                                shuffle=True,
                                                batch_size=BATCH_SIZE,
                                                color_mode = 'rgb',
                                                image_size=IMG_SIZE,
                                                validation_split=0.25,
                                                subset='validation',
                                                seed = 1)
```

```
    class_names = train_dataset.class_names

    AUTOTUNE = tf.data.experimental.AUTOTUNE
    train_dataset = train_dataset.prefetch(buffer_size=AUTOTUNE)
    test_dataset = test_dataset.prefetch(buffer_size=AUTOTUNE)
    data_augmentation = fun.data_augmenter()


    model = Sequential()
    model.add(Conv2D(filters=96, kernel_size=(11,11), strides=(4,4),
activation='relu', input_shape=(227,227,3)))
    model.add(BatchNormalization())
    model.add(MaxPool2D(pool_size=(3,3), strides=(2,2)))
    model.add(Conv2D(filters=256, kernel_size=(5,5), strides=(1,1),
activation='relu', padding="same"))
    model.add(BatchNormalization())
    model.add(MaxPool2D(pool_size=(3,3), strides=(2,2)))
    model.add(Conv2D(filters=384, kernel_size=(3,3), strides=(1,1),
activation='relu', padding="same"))
    model.add(BatchNormalization())
    model.add(Conv2D(filters=384, kernel_size=(3,3), strides=(1,1),
activation='relu', padding="same"))
    model.add(BatchNormalization())
    model.add(Conv2D(filters=256, kernel_size=(3,3), strides=(1,1),
activation='relu', padding="same"))
    model.add(BatchNormalization())
    model.add(MaxPool2D(pool_size=(3,3), strides=(2,2)))
    model.add(Flatten())
    model.add(Dense(4096, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(4096, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(10, activation='softmax'))

    opt = keras.optimizers.SGD(learning_rate=learn)
    model.compile(loss='sparse_categorical_crossentropy',
optimizer=opt, metrics=['accuracy'])
    model.summary()

    history = model.fit(train_dataset, epochs=epocas,
validation_data=test_dataset)
    model.save("AlexNet_Model.h5")


    df_loss_acc = pd.DataFrame(history.history)

    plt.figure(1)
    df_loss= df_loss_acc[['loss','val_loss']]
    df_loss.rename(columns={'loss':'train','val_loss':'validation'},in
place=True)
    df_loss.plot(title='Model
loss',figsize=(12,8)).set(xlabel='Epoch',ylabel='Loss')
    plt.close(1)

    plt.figure(2)
    df_acc= df_loss_acc[['accuracy','val_accuracy']]
    df_acc.rename(columns={'accuracy':'train','val_accuracy':'validati
```

```
on'},inplace=True)
    df_acc.plot(title='Model
Accuracy',figsize=(12,8)).set(xlabel='Epoch',ylabel='Accuracy')
    plt.close(2)

    AlexNet_pred = model.predict(test_dataset)

    resultados_AlexNet = df_loss_acc.to_numpy()

    return (resultados_AlexNet[-1,0], resultados_AlexNet[-1,1],
resultados_AlexNet[-1,2], resultados_AlexNet[-1,3])
```

ANEXO E

Código correspondiente a la función del modelo ResNet50.

```
'''
José E. Maese Álvarez.
TFG: Uso de redes neuronales para identificación de matrículas.
Funciones de ResNet50
'''
import matplotlib.pyplot as plt
import pandas as pd
import cv2
import funciones as fun
import numpy as np
import keras
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Flatten
from keras.layers import Dense
from keras.models import load_model
import tensorflow as tf
from tensorflow.keras.preprocessing import
image_dataset_from_directory
import os
os.environ['KMP_DUPLICATE_LIB_OK']='True'

# learn = 0.00001
# epocas = 40
def ResNet(learn, epocas):
    BATCH_SIZE = 16
    IMG_SIZE = (180, 180)
    directory = "Imagenes/Numeros/"
    train_dataset = image_dataset_from_directory(directory =
'Imagenes/Numeros',
                                                 shuffle=True,
                                                 color_mode = 'rgb',
                                                 batch_size=BATCH_SIZE,
                                                 image_size=IMG_SIZE,
                                                 validation_split=0.25,
                                                 subset='training',
                                                 seed = 1)

    test_dataset = image_dataset_from_directory(directory =
'Imagenes/Numeros',
                                                 shuffle=True,
                                                 batch_size=BATCH_SIZE,
                                                 color_mode = 'rgb',
                                                 image_size=IMG_SIZE,
                                                 validation_split=0.25,
                                                 subset='validation',
                                                 seed = 1)
```

```
    AUTOTUNE = tf.data.experimental.AUTOTUNE
    train_dataset = train_dataset.prefetch(buffer_size=AUTOTUNE)
    test_dataset = test_dataset.prefetch(buffer_size=AUTOTUNE)
    data_augmentation = fun.data_augmenter()


    model = Sequential()

    pretrained_model=
tf.keras.applications.ResNet50(include_top=False,
                                                    input_shape=(180,
180,3),
                                                    pooling='avg',cla
sses=10,
                                                    weights='imagenet
')

    pretrained_model.trainable = False
    pretrained_model.summary()

    model.add(pretrained_model)
    model.add(Flatten())
    model.add(Dense(512, activation='relu'))
    model.add(Dense(10, activation='softmax'))

    opt = keras.optimizers.Adam(learning_rate=learn)
    model.compile(optimizer=opt,
loss='sparse_categorical_crossentropy',metrics=['accuracy'])
    history = model.fit(train_dataset, validation_data=test_dataset,
epochs=epocas)

    model.save("ResNet_Model.h5")

    plt.figure(1)
    df_loss_acc = pd.DataFrame(history.history)
    df_loss= df_loss_acc[['loss','val_loss']]
    df_loss.rename(columns={'loss':'train','val_loss':'validation'},in
place=True)
    df_loss.plot(title='Model
loss',figsize=(12,8)).set(xlabel='Epoch',ylabel='Loss')
    plt.close(1)

    plt.figure(2)
    df_acc= df_loss_acc[['accuracy','val_accuracy']]
    df_acc.rename(columns={'accuracy':'train','val_accuracy':'validati
on'},inplace=True)
    df_acc.plot(title='Model
Accuracy',figsize=(12,8)).set(xlabel='Epoch',ylabel='Accuracy')
    plt.close(2)

    ResNet_pred = model.predict(test_dataset)

    resultados_ResNet = df_loss_acc.to_numpy()
    return (resultados_ResNet[-1,0], resultados_ResNet[-1,1],
resultados_ResNet[-1,2], resultados_ResNet[-1,3])
```