Trabajo Fin de Grado
Grado en Ingeniería Electrónica, Robótica y Mecatrónica

Reconocimiento automático de caracteres manuscritos utilizando aprendizaje máquina

Autor: Roberto Gallo García
Tutor: Francisco José Simois Tirado

Dpto. de Teoría de la Señal y Comunicaciones
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla,diciembre 2021

Trabajo Fin de Grado
Grado en Ingeniería Electrónica, Robótica y Mecatrónica

Reconocimiento automático de caracteres manuscritos utilizando aprendizaje máquina

Autor:
Roberto Gallo García

Tutor:
Francisco José Simois Tirado
Profesor Contratado Doctor

Dpto. de Teoría de la Señal y Comunicaciones
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, septiembre 2022

Trabajo Fin de Grado: Reconocimiento automático de caracteres manuscritos utilizando aprendizaje máquina

Autor: Roberto Gallo García

Tutor: Francisco José Simois Tirado

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

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2022

El Secretario del Tribunal

A mi familia

Agradecimientos

Quisiera agradecer en primer lugar a todos aquellos que estuvieron conmigo al empezar la carrera y hoy en día siguen a mi lado viéndome dar estos últimos empujones. A mi familia, que estuvo ahí con cada bache y me ayudó a superarlo y a seguir adelante cuando ni yo mismo podía. A mis amigos del barrio, que siempre me dijeron que podía con todo lo que me echasen y a día de hoy aún me dan ánimos y están a mi lado. A mis amigos del instituto y de la Universidad, de los que aprendí que la distancia o los obstáculos en el camino no separan a las personas y gracias a las cuáles sigo persiguiendo con ilusión este proyecto. También me quisiera agradecer a mi pareja María el apoyo incondicional que me aportó en mis mejores y peores momentos y ha sido una de las mejores personas que me ha dado la carrera. Por último, me gustaría agradecer especialmente a todos aquellos profesores que han sido parte de mi vida; desde que empecé a educarme a mi tutor en este proyecto, Francisco, pues reúnen todas aquellas cualidades que he citado anteriormente y han sido fuente de inspiración y de constancia para mí.

Termino así esta etapa en la que el aprendizaje y el descubrimiento han formado una parte especial de mi vida. Gracias de nuevo a todas las personas que han hecho que esto se pueda llevar a cabo.

Roberto Gallo García

Estudiante de ingeniería

Sevilla, 2022

Resumen

La Inteligencia Artificial y la búsqueda de la replicación de la consciencia y actividades humanas por parte de las máquinas como leer, escribir, cantar etc. ha sido desde hace siglos un sueño y un campo de investigación para la sociedad. En los últimos 60 años afortunadamente estos sueños se están volviendo realidad gracias al desarrollo exponencial de la tecnología para tareas de automatización de actividades realizadas hasta ahora por seres humanos. Una de las actividades con mayor recorrido y avances ha sido el reconocimiento óptico de formas y patrones, con aplicaciones en infinidad de campos y que permiten el proceso de información de nuestro entorno para luego combinarla con otras tareas de automatización y clasificación. Nos encontramos por lo tanto con un campo con una gran proyección a futuro y en el que las máquinas cada vez están más cerca de competir con capacidades humanas de lectura y abstracción.

Dentro del campo del reconocimiento óptico, el reconocimiento de caracteres y en especial el de textos manuscritos ha sido uno de los mayores retos a afrontar debido a la diferencia en la que cada persona escribe un mismo carácter y las condiciones en las que cada texto se nos puede presentar. Estas dificultades son estudiadas y solventadas gracias al Reconocimiento Óptico de Caracteres (ROC).

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

Este proyecto expone el estudio de este campo desde las bases, partiendo de los retos y la tecnología a emplear y posteriormente realizando una comparativa de diferentes modelos, demostrando que el desarrollo de la Inteligencia Artificial ha pasado en solo unas décadas de un reducido grupo de personas a todas aquellas que con inquietud y tiempo contribuyen al avance de la sociedad.

Abstract

Artificial Intelligence and research of machine replication of human conscience and activities like reading, writing, singing etc. have been a dream for centuries and an important investigation field for modern society. Fortunately, for the last 60 years these issues have become more than a dream by reason of exponential development of technology for automatic processes which had completely been developed by humans so far. One of the activities with the most of history is optical recognition for patterns and shapes, with several applications in processing information around us to combine it with other classification and automation processes. In conclusion, we have faced a field with a large future projection in which machines have upgraded their performance and are currently closer to compete with human reading and abstraction abilities.

Within optical recognition development, character recognition and specially handwritten text have been one of the most challenging tasks to face because of the differences in typography for the same character and all the environmental conditions which accompany the text. These difficulties are being studied and solved in Optical Character Recognition (OCR).

Recently, and owing to the study of Neural Networks, the creation of better and more complex rendering architectures and, above all, a more powerful hardware for commercial computers, OCR problems have reached a wider audience who have obtained better accuracy for these systems while reducing errors.

This project contains the study of this field from the roots, starting from the challenges to face and the technology available to use and, after that, making a comparison between different OCR models, showing that AI development has come from something studied by a small group of people to everyone who, with curiosity and time contributes to society progress.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

1 INTRODUCCIÓN

A lo largo de los últimos 20 años, la tecnología de reconocimiento óptico de caracteres se ha usado para transcribir millones de libros, periódicos y artículos a texto que pueda ser encontrado fácilmente usando cualquier buscador en Internet. Para ello, se emplean herramientas capaces de procesar la imagen y luego extraer de las mismas el texto.

Aún así, esta tarea solo puede realizarse de manera eficiente si la imagen se presenta de forma clara y el texto es lo suficientemente visible. La tarea de reconocimiento implica por lo tanto la localización del texto, la segmentación de este en caracteres y la correcta identificación de los mismos. Aunque algunos de los sistemas son capaces de identificar las palabras sin esa segmentación previa, requieren de un mayor coste computacional para obtener la misma precisión que los otros.

Para ello, primero se ha de aplicar técnicas de procesamiento de la imagen como son la detección de contornos, la reducción de ruidos, erosión y dilatado de la imagen entre otras para conseguir que cada uno de los caracteres se presenten de la mejor forma. Eliminar sombras, aplicar filtros de color y binarizar la imagen son otros de los pasos posteriores que se han de seguir para normalizar la imagen. Posteriormente se ha de segmentar la imagen en cada uno de los elementos que lo conforman e introducirlos en nuestro programa ya entrenado para detectar el carácter ante el cual nos encontramos.

Aunque son muchas las técnicas utilizadas para entrenar a nuestro sistema en el reconocimiento de caracteres, la más utilizada es aquella que se basa en Redes Neuronales, similares a las que conforman el cerebro humano y que a través de múltiples iteraciones con datos ya reconocidos poder crear las conexiones necesarias para poder predecir correctamente datos que no forman parte del entrenamiento.

Esta tecnología la aplican hoy en día de forma satisfactoria en numerosas entidades bancarias, centros de marketing, académicos y no solo son capaces agilizar el procesamiento de información sino de mejorar la vida de las personas haciendo que un programa lea en voz alta ficheros de texto para personas con problemas de visión. Las aplicaciones de esta tecnología cada vez abarcan más ámbitos de nuestra sociedad y vida cotidiana.

1.1. Alcance del Proyecto

1.1.1. Objetivo

El objetivo de este proyecto es estudiar y entrenar diferentes modelos de Redes Neuronales para probar su desempeño en el reconocimiento de caracteres tanto en datos de validación como en caracteres escritos a mano en diferentes condiciones. También se probará el programa detecte más de un carácter en la misma imagen y que puede hacerlo en tiempo real dada una entrada de vídeo

Este trabajo podría tener numerosas aplicaciones aparte de las ya mencionadas, pasando de la detección de caracteres a palabras y luego usarlo por ejemplo para el reconocimiento de secciones de un carnet de identidad.

1.1.2. Requisitos

Según la norma ISO 21500 [1] se estructurará el proyecto en bloques diferenciados para realizar un seguimiento adecuado del mismo. De esta forma los requisitos pasaron a ser los siguientes:

1.1.2.1. Requisitos funcionales

- F.1: El sistema leerá correctamente los datos de entrada para el entrenamiento, validación y pruebas.
- F.2: El sistema procesará y mostrará los resultados de cada experimento de forma legible, y será capaz de contrastarlos con predicciones tomadas de los datos de validación
- F.3: El sistema será capaz de procesar imágenes de caracteres individuales y predecirlos
- F.4: El sistema detectará si hay más de un carácter en la imagen y realizará una predicción de cada uno, mostrando el resultado y el carácter en una caja delimitadora
- F.5: El sistema será capaz de realizar predicciones en tiempo real para una entrada de vídeo dada.

1.1.2.2. Requisitos de prestaciones

- P.5: El sistema contará con una entrada de vídeo para realizar las predicciones en tiempo real

1.1.3. Restricciones

- R.5: La calidad de la cámara y las condiciones de iluminación de la estancia podrán no ser las óptimas para la correcta detección de los caracteres.

1.2. Planificación del proyecto

A la hora de realizar cualquier proyecto es muy importante la organización y planificación de éste, marcando los objetivos y los pasos que debemos seguir desde el principio.

La planificación de este proyecto se dividirá en varias etapas que coinciden con las del desarrollo del proyecto. La primera etapa consistirá en la búsqueda de información y estudio del estado del arte de forma que se parta de los conceptos básico en Inteligencia Artificial y cómo funciona el aprendizaje de una red neuronal. Visto los diferentes componentes de un sistema de aprendizaje automático se propondrá un ejemplo sencillo para entenderlos y desde esa base nos enfrentaremos al problema planteado en el proyecto.

En la segunda etapa nos enfocamos en nuestro problema principal y analizamos el entorno de trabajo y los requisitos necesarios para un correcto entrenamiento. Una vez comprobado que se realiza la lectura de los sets elegidos para el entrenamiento, se realizarán distintos entrenamientos con distintas arquitecturas y se compararán los resultados, buscando predecir el comportamiento de cada modelo en un entorno real de trabajo.

Una vez superada la etapa anterior, en una tercera etapa realizaremos las pruebas para caracteres de diferente tipografía de forma que se puedan contrastar las hipótesis del apartado anterior con los resultados obtenidos. Realizado este análisis introduciremos el problema de la detección de múltiples caracteres y cómo sería posible segmentar la imagen de forma eficaz. Realizaremos nuevamente pruebas con los distintos modelos y trasladaremos la solución propuesta para vídeos y comprobar que el sistema se comporta correctamente ante casos en tiempo real.

Por último, se expondrán las conclusiones a las que se llegaron con la información recogida y los experimentos que se realizaron, así como propuestas de optimización y mejora.

1.3. Matriz de verificación

- F.1: El sistema leerá correctamente los datos de entrada para el entrenamiento, validación y pruebas:
    - El sistema muestra correctamente datos de entrenamiento etiquetados (test 1.1)
    - El sistema muestra correctamente datos de validación etiquetados (test 1.2)
    - El sistema lee correctamente imagen y vídeo (test 1.3)
- F.2: El sistema procesará y mostrará los resultados de cada experimento de forma legible, y será capaz de contrastarlos con predicciones tomadas de los datos de validación
    - Obtención de gráficas de precisión y de pérdida para cada uno de los modelos (test 2.1)
    - Obtención de los resultados de predicción para caracteres aleatorios del set de validación (test 2.2)
- F.3: El sistema será capaz de procesar imágenes de caracteres individuales y predecirlos
    - Predicción de diferentes caracteres para los distintos modelos (test 3)
- F.4: El sistema detectará si hay más de un carácter en la imagen y realizará una predicción de cada uno, mostrando el resultado y el carácter en una caja delimitadora
    - Predicción y delimitación para distintas imágenes de múltiples caracteres (test 4)
- F.5: El sistema será capaz de realizar predicciones en tiempo real para una entrada de vídeo dada.
    - Pruebas con entrada de vídeo (webcam) donde aparezcan caracteres, y predicción y delimitación de los mismos (test 5)

Requisito	Nombre del requisito	Verificación				Nombre de prueba	Estado
		I	A	D	T		
F1	Lectura de datos				X	1.1. Lectura de datos de prueba	completado
F1	Lectura de datos				X	1.2. Lectura de datos de validación	completado
F1	Lectura de tados				X	1.3. Leer imagen y vídeo	completado
F2	Procesamiento de resultados				X	2.1. Obtención de gráficas de precisión y pérdidas	completado
F2	Procesamiento de resultados				X	2.2. Obtención de resultados (set de validación)	completado
F3	Pruebas en imágenes				X	3. Predicción para caracteres individuales	completado
F4	Caso de múltiples caracteres				X	4. Predicción para múltiples caracteres	completado
F5	Predicción en tiempo real				X	5. Predicción y delimitación de caracteres en TR	completado

Tabla 1. Matriz de verificación

2 ESTADO DEL ARTE

El reconocimiento óptico de caracteres, usualmente abreviado como OCR por sus siglas en inglés (*Optical Character Recognition*) es la conversión por parte de una computadora de una imagen de texto, bien sea impreso o manuscrito, a texto entendible por el ordenador. En la actualidad sus usos abarcan infinidad de campos; como la lectura de pasaportes, notas bancarias, traducción automática de documentos o en cualquier otro sector en el que se usen archivos impresos. Es la forma más común en la que se digitaliza texto para luego ser almacenado, editado o usado en diversos procesos como transformar el texto en audio (*text-to-speech*) o gestión de bases de datos. [2]

Un programa que implemente OCR deberá, por lo tanto, reconocer "patrones" que permitan distinguir los diferentes caracteres, aprendiendo a través de una red neuronal; y almacenar ese texto escrito en un fichero ASCII. Esto no es una tarea fácil y ha sido desarrollada recientemente gracias al gran desarrollo y éxito del *Deep Learning*, hasta conseguir modelos con una precisión y eficiencia del orden del 99% en el reconocimiento de dígitos manuscritos. Aunque en un principio pudiese parecer que el OCR es una consecuencia del desarrollo de la Inteligencia Artificial, la realidad es que el ser humano lleva intentando implementar esta tecnología desde finales de los años 20. La invención de este sistema se le atribuye a Gustav Tauschek, un inventor austriaco que en 1929 desarrolló un primitivo sistema de reconocimiento de caracteres para las máquinas de cálculo basadas en tarjetas perforadas. El dispositivo, bautizado como la *Máquina de Tauscheck* [3], era un aparato mecánico en el cual se colocaba un fotodetector y una serie de plantillas de forma que cuando la plantilla y el carácter estuviesen alineados, una luz era dirigida hacia el fotodetector, que almacenaba el valor del carácter reconocido. Este sistema, aunque limitado, puso en pie ya desde esa época la necesidad de trasladar texto impreso a un sistema de almacenaje y gestión de datos.

A pesar de tener origen en esa época, no fue hasta 1953 cuando se empezaron a usar versiones comerciales de máquinas que convertían mensajes impresos en lenguajes para almacenarlos en un computador. Estos modelos utilizaban un análisis de la imagen en vez de una comparación pudiendo aceptar variaciones en la fuente, necesidad fundamental para reconocer caracteres manuscritos y base sobre la que funcionan los sistemas de reconocimiento de caracteres ópticos actuales. La máquina llamada *Gismo* [4] fue utilizada para automatizar datos de agencias, leer impresiones en tarjetas de crédito e incluso por la fuerza aérea de los Estados Unidos para transcribir mensajes escritos a máquina.

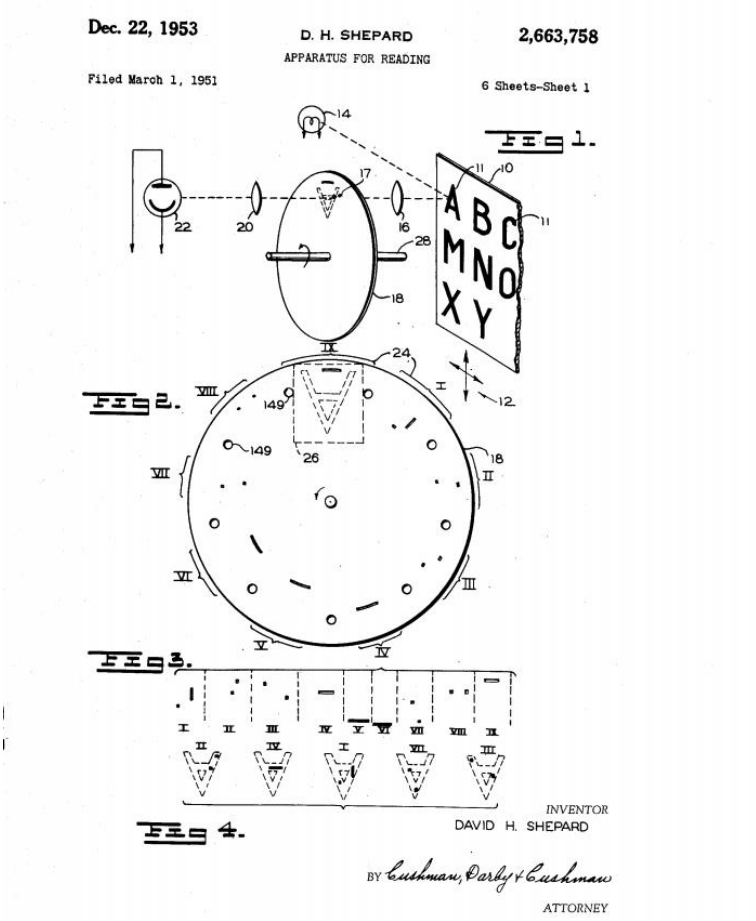


Figura 1. Fragmento de la patente de la Máquina de Gismo. [4]

Los sistemas OCR hasta ahora asumían que el texto seguía una estructura predeterminada y los caracteres se encontraban en posiciones fijas (como podría ser el caso de la matrícula de un coche). En la realidad las imágenes no siguen una plantilla y tendrán variaciones de tipografía, formas, iluminación y orientación por lo que estos sistemas seguían siendo extremadamente limitados.

Los sistemas OCR actuales se consiguieron implementar a finales de los 90 combinando los métodos contemporáneos que se usaban con métodos de reconocimiento de patrones e inteligencia artificial. La mejora de estos sistemas también ha sido posible gracias al desarrollo no solo de los algoritmos sino también de la tecnología encargada de capturar y procesar la información: escáneres, cámaras, ordenadores… Todos estos avances han permitido el procesamiento de más datos tanto por la mayor capacidad de cómputo como por la mejor resolución de los sensores. [5]

A pesar de que los resultados mejoran de forma exponencial con el paso de los años, todavía quedan multitud de retos por afrontar para el reconocimiento óptico de caracteres, como la gran variedad de situaciones de iluminación de los fotogramas, la situación y variedad de fuentes de los caracteres y sobre todo la diferencia de formas en las que varias personas pueden escribir un mismo carácter, lo que dificulta la clasificación con precisión de caracteres.

Con objeto del estudio de estas técnicas de clasificación, así como su eficiencia y mostrar resultados empíricos de diferentes modelos de clasificadores, nace este Trabajo de Fin de Grado, donde se pretende en primer lugar dar unos esbozos y definir las bases que cimientan las redes de Deep Learning, estableciendo los conceptos y terminología que usaremos, el estado del arte de la Inteligencia Artificial tal y como hoy en día la conocemos y se explicará el diseño realizado desde cero del método de entrenamiento y predicción de diversos sistemas.

2.1 Estado del arte de la Inteligencia Artificial

El estudio y perfeccionamiento de la Inteligencia Artificial, a pesar de la creencia general de ser una tecnología reciente, lleva en nuestra sociedad más de 70 años. El inicio del concepto data de octubre de 1950, cuando Alan Turing publicó en la revista *Mind* el artículo *"Computing Machinery and Intelligence"* (Maquinaria computacional e inteligencia) [6], aunque no fue hasta 1956 que se acuñó por primera vez el término *"Inteligencia Artificial"* en una serie de talleres académicos en *Dartmouth College*, Nuevo Hampshire (EE. UU.). [7]

En estos talleres, liderados por el prominente informático John McCarthy, se desarrolló la idea de cómo los ordenadores podían emular el comportamiento humano. McCarthy creía que todo el sistema de resolución de problemas cotidianos que realizaban los seres humanos podía descomponerse en técnicas matemáticas y, por consiguiente, formalizado con algoritmos.

A partir de ese momento y hasta los años 80, se empezaron a crear conceptos que hoy en día consideramos inherentes a la IA, como pueden ser el reconocimiento de patrones, Machine Learning, ciencia cognitiva… El discurso y objetivos pasaron entonces de qué puede hacer la Inteligencia Artificial *por* nosotros a qué podía hacer *para* nosotros.

Durante los años 50 y 60 se desarrollaron las primeras máquinas con una primitiva inteligencia artificial, como la *STUDENT (1964)* [8] que podía resolver problemas algebraicos dada una oración. Por ejemplo, dada la frase *"¿Cuál es el cuadrado del 20% de 45?"* El programa a través de la inferencia lógica de las declaraciones podía mostrar el resultado *81*. Otro ejemplo de desarrollo temprano de estas tecnologías fue *ELIZA* [8], uno de los primeros "*chatbots*" que podía emular y mantener una conversación coherente con el usuario. Este rápido desarrollo generó una reacción en cadena que llevó en poco tiempo a la desconfianza en el poder de la IA. Este período en los 70 se conoció como el "AI Winter" (Invierno de la Inteligencia Artificial) [9] y sirvió para dejar claro que no hay nada de magia en la Inteligencia Artificial, sólo probabilidad, disponibilidad de muestras y las capacidades de nuestra computadora.

La IA tuvo su segundo período de desarrollo a partir de 1980, donde el progreso de los modelos matemáticos relacionados con este sector introdujo las redes neuronales multicapa y feed-forward y el algoritmo de propagación hacia atrás (*backpropagation algorithm*). Como consecuencia de este avance ahora los modelos se actualizaban dependiendo de las entradas que recibían. Fue en esta época y gracias a la combinación de todos los avances anteriores cuando se consiguió por primera vez que una máquina venciese por primera vez al ajedrez a un ser humano.

Motivados por el avance significativo que tuvo este campo, muchos países, incluidos Japón y Estados Unidos, invirtieron en el desarrollo de los llamados ordenadores de quinta generación a lo que se refería como "computadores de inteligencia artificial". Aun así, se encontraron con que la incapacidad de aprender en base a algoritmos a partir de los datos proporcionados; y la de conseguir ciertas cotas de incertidumbre, relevaron a estos ordenadores a un segundo plano mientras en esta época surgían por parte de IBM y Apple los llamados "computadores personales", con un menor coste y más rápidos. El colapso en el mercado de los ordenadores de quinta generación en el mercado llevó a la Inteligencia Artificial a un "segundo invierno", con pocas esperanzas de que resurgiese nuevamente. [9]

A pesar de perder la atención por parte del gran público, el desarrollo para encontrar y mejorar modelos no cesó y fue a finales del siglo XX donde un nuevo paradigma, el Machine Learning (aprendizaje máquina) apareció. En vez de ser programados directamente, los modelos eran entrenados para descubrir "patrones" en los datos. Esto junto con el rápido crecimiento de técnicas de optimización y mejora, como las redes bayesianas, el algoritmo de esperanza-maximización y los árboles de decisión; y el aumento de la potencia computacional de los ordenadores personales provocó que desde inicios del siglo XXI hasta hoy en día el campo de la Inteligencia Artificial se haya extendido hasta llegar al gran público y a numerosos sectores de la sociedad, donde se está viviendo el tercer gran desarrollo de la Inteligencia Artificial que se espera que se extienda en el tiempo a lo largo de varias décadas.

2.2 Conceptos iniciales: Machine Learning, Inteligencia Artificial y Deep Learning

Con el objetivo de que la investigación comience desde las bases de las técnicas del aprendizaje máquina, y de que pueda ser entendida por la audiencia más o menos versada en el tema a tratar, se empezarán definiendo una serie de conceptos que en ocasiones son confundidos entre ellos incluso por personas conocedoras de los mismos. Estos son el Machine Learning, o aprendizaje máquina, IA o Inteligencia Artificial y Deep Learning.

La **inteligencia artificial** es la habilidad o capacidad de las máquinas para emular comportamientos humanos [10], tales como:

- Reconocer estímulos
- Percibir cambios en el entorno para actuar en base a ellos
- Aprender de los errores y actuar para evitarlos si se dan nuevamente

Este concepto de máquina con Inteligencia Artificial considera un sistema perfecto a aquellos que consiguiesen nuestros sentidos, nuestra inteligencia y nuestra razón. En el cine y la ficción muchas veces se muestra además a este tipo de máquinas como autómatas con forma humana, como *C3PO* en la saga *Star Wars*, o *Terminator*. En la actualidad esas máquinas son conceptos imaginarios, pues la tecnología no permite desarrollar todavía ese tipo de sistemas tan complejos, aunque sí existen algunos encargados de realizar determinadas tareas que requieres de los sentidos que poseen los seres humanos, como el reconocimiento y tratamiento de imágenes, reconocimiento de audio, aprendizaje a través de experiencias etc.

Todas estas actividades a pesar de ser triviales para los humanos requieren en ocasiones de una compleja programación para el caso de las máquinas y en otras, de una base de datos casi infinitas. Solamente hay que plantear un caso tan simple como poder distinguir si un animal es un perro o un gato, para llegar a la conclusión que realizar un programa convencional en un lenguaje de estándar resultaría una tarea inabarcable. Para explicar entonces como se dota de esa "inteligencia" a un programa hay que hablar del Machine Learning.

El **Machine Learning** es un campo de la Inteligencia Artificial que utiliza técnicas estadísticas y de análisis de datos para dotar a las computadoras la capacidad de aprender y mejorar automáticamente, sin necesidad de ser programadas para una tarea en específico. [11] Los algoritmos para entrenar a nuestros sistemas variarán de los datos de los que partamos y de la función que desarrollará nuestra computadora. Podemos distinguir tres categorías principales:

- **Aprendizaje supervisado:** Es la técnica más utilizada de ML. Consiste en establecer una función matemática a través de los datos que recibe como entrada y los que espera recibir como salida. La función que se crea posteriormente se utilizará para la predicción de nuevas salidas. El principal inconveniente que presenta este planteamiento es la necesidad de haber trabajado y etiquetado un gran conjunto de datos para entrenar a nuestro sistema y minimizar su error. Se utiliza por ejemplo para asignar a las casas un valor determinado dependiendo de sus características.

- **Aprendizaje no supervisado:** Otra de las principales técnicas del aprendizaje máquina consiste en dotar a nuestro sistema de una base de datos, pero esta no ha sido categorizada previamente. El algoritmo por lo tanto proporcionará a los datos una estructura determinada basada en similitudes y patrones dependiendo además del número de categorías que queramos establecer. Es el algoritmo más parecido a como los humando tratamos la información, pues cuando los niños aprenden tienden a agrupar en el mismo conjunto elemento con características parecidas. Por ejemplo, si un bebé tiene un perro como mascota y se encuentra un gato, lo asociará y tratará como si fuera un perro debido a que para él no existe el grupo "perros" o "gatos" como tal, es en el momento en el que se le induce al niño a crear dos grupos distintos cuando pasarán a distinguirse entre sí y a ser etiquetados.

- **Aprendizaje semi-supervisado:** Es una mezcla de los dos algoritmos vistos hasta el momento. Se le dota al sistema de unos pocos datos de entrenamiento etiquetados, suficientes para crear las categorías,

y datos no etiquetados pero que nuestro algoritmo agrupará dependiendo de sus características. Pretende combinar el ahorro del trabajo manual de etiquetado con la asignación de una etiqueta para un conjunto determinado de características. Normalmente se supone que los datos que nos proporcionan (etiquetados y no etiquetados) vienen de la misma distribución, pero hay que tener cuidado, ya que esto puede no ser así y en el peor de los casos este tipo de aprendizaje podría generar un sesgo en la selección de datos no etiquetados.

- **Aprendizaje reforzado:** Estos algoritmos no se basan en un sistema de acierto-error para el aprendizaje del sistema, sino que dotan al programa de premios o refuerzos positivos dependiendo del desempeño que haya tenido la red en esa situación en concreto. Son usados en aplicaciones que cuentan con multitud de variables y casuísticas; como puede ser el caso de enseñar a un ordenador a jugar al ajedrez, al escondite o incluso a conducir. El problema principal con el que cuentan estas redes, según el ganador de un Premio Turing, Yann Lecun, es "el tiempo que necesita el programa para aprender hasta conceptos relativamente simples". Es notorio el caso de *AlphaStar*, un programa creado por *DeepMind* basado en el aprendizaje profundo que pretendía aprender a jugar al *StarCraft II*. El programa tardó 200 años de juego real (se encontraba en un entorno acelerado) para conseguir su primera victoria. [12]

Habiendo puesto en contexto los diferentes tipos de aprendizaje que existen, pasaremos por último a explicar el concepto de Deep Learning.

El **Deep Learning** o Aprendizaje Profundo es una tecnología de aprendizaje automático que imita el sistema de aprendizaje humano donde a través de múltiples capas de procesamiento y configuración de parámetros internos se aprende a realizar tareas. [13] Entre todas las técnicas que componen este subsector de la Inteligencia Artificial nos centraremos en aquellos que pretenden imitar el comportamiento de las redes neuronales de nuestro cerebro, en el que se crea un sistema que se compone de capas formadas por neuronas, y estas interactúan entre sí intercambiando e interpretando información entre sí. Este tipo de aprendizaje tiene a obtener mejores resultados que otras estructuras cuando la cantidad de datos es muy grande, que es cuando otros algoritmos tienden a saturarse.

El Deep Learning por definición es un subgrupo del Machine Learning, y que se distingue de otros por la manera en la que se extraen las características, no necesitando del procesamiento previo en la que necesitamos indicar las características más importantes.

Esto se puede ver de forma gráfica si tomamos el ejemplo anterior del precio de las casas y lo comparamos con un programa que distinga aves de mamíferos. En el primero es necesario indicar las características más importantes y esas serán nuestras entradas. Superficie, número de habitaciones, ubicación… Cuantas más entradas tenga nuestra red más precisión podremos obtener, pero más largo será el trabajo previo de extracción de características. [14]

Por el contrario, en el segundo ejemplo podríamos utilizar una red neuronal que procese imágenes y realice la extracción de características, eliminando así parte de la intervención humana. Tanto el Machine Learning como el Deep Learning permiten todos los tipos de aprendizajes vistos anteriormente, teniendo mayor o menor éxito dependiendo de la tarea a realizar. Más adelante se hablará de las redes neuronales convolucionales, donde quedarán de manifiesto todas estas capacidades y se implementarán en nuestro programa para el análisis de los caracteres alfanuméricos más allá de ser píxeles en una posición concreta.

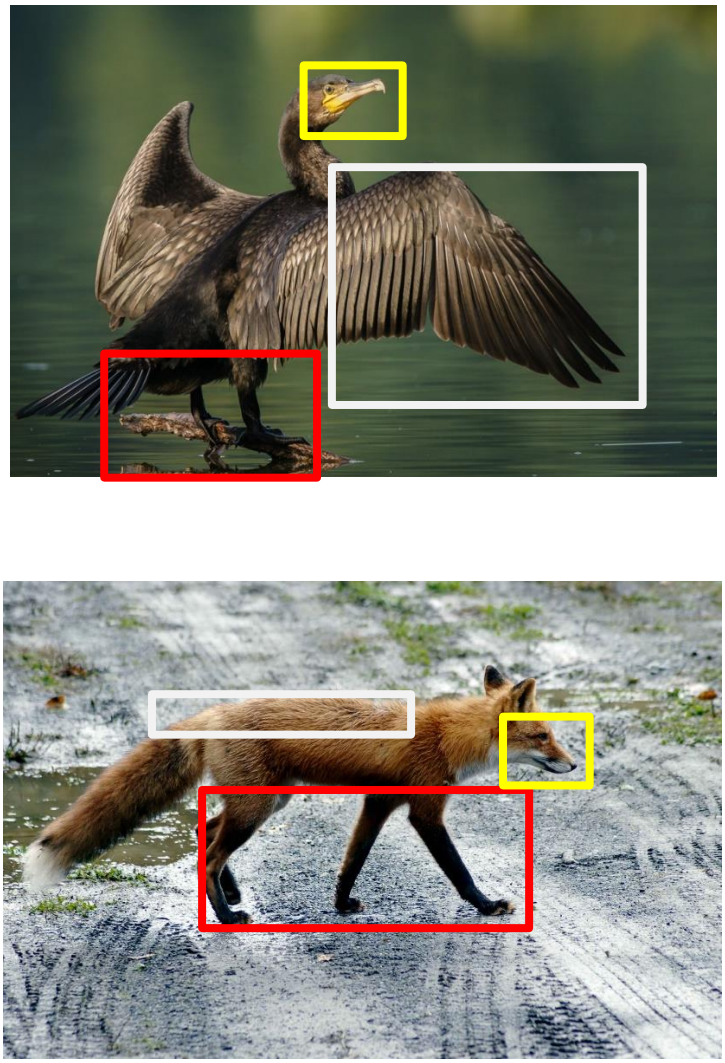


Figura 2. Análisis de las características y diferencias principales entre un cormorán y un zorro. El diseño de nuestra red busca que mediante un algoritmo de aprendizaje el programa sepa cuáles son los "patrones" que debe buscar para determinar si en una imagen aparece un ave o un mamífero

En este Trabajo nos centraremos en el campo del Aprendizaje Profundo, sus componentes y diferentes aproximaciones a problemas que podemos encontrarnos. La combinación del Deep Learning con el aprendizaje supervisado es hoy en día la más utilizada para el procesamiento de imágenes o texto y más tarde comprobaremos hasta qué punto es eficaz reconociendo patrones en una imagen. Debido a que es mediante imágenes como los humanos obtenemos más información, el desarrollo de estos sistemas suponen un avance inconmensurable, tanto para modelos de predicción como de automatización.

3 INTRODUCCIÓN A LAS REDES NEURONALES

Para abordar el reconocimiento de caracteres que se pretende conseguir y que grandes empresas como Google o bancos internacionales utilizan primero se tratarán las redes neuronales desde un punto de vista más simple para luego poder realizar la abstracción a los complejos sistemas multicapa. La explicación más sencilla de lo que es una red neuronal es la de un sistema que recibe una o varias entradas y devuelve una salida como combinación de una o más funciones. [15]

Cualquier sistema neuronal está formado por **perceptrones** que toman las entradas y las multiplica por un **peso (w)**, que es una representación de la importancia de esa entrada. La suma de las entradas multiplicadas por los pesos nos dará un valor que si supera cierto umbral la salida de la neurona pasará de cero a uno. [14]

Tomemos como ejemplo la situación en la que queremos hacer una ruta de senderismo por el campo, habrá que analizar varios factores a tener en cuenta como:

- ¿Disponemos de calzado adecuado?
- ¿Hace buen tiempo?
- ¿Nuestros amigos pueden ir?

Así como en la vida real, la respuesta a estas preguntas tiene diferente peso y determinará en mayor o menor medida si realizamos la actividad. Que un amigo nuestro no pueda venir puede que no haga que cancelemos el plan, pero un pronóstico meteorológico de lluvia y tormenta hará que nuestros planes se pospongan, aunque puedan ir todos nuestros amigos. Cada entrada por lo tanto tendrá un número de peso distinto. Asignamos por ejemplo peso 5 al hecho de que haya buen tiempo, 4 al hecho de llevar buen calzado y por último un 3 a que nuestros amigos vengan. La estructura de ese perceptrón será por lo tanto la siguiente:

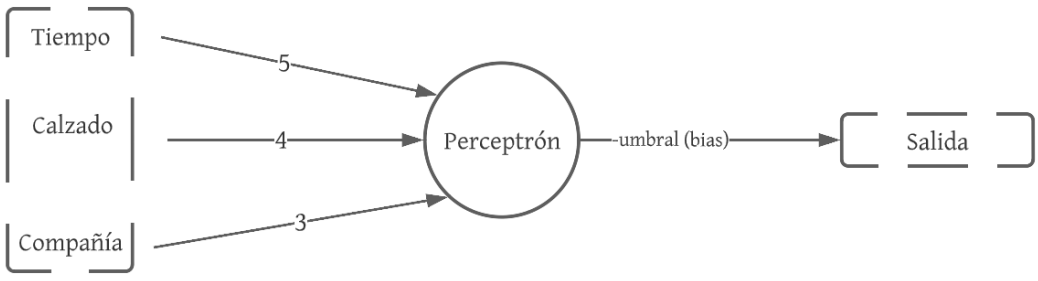


Figura 3. Esquema de un perceptrón con tres entradas y una salida.

Si decidimos que nuestro umbral sea 8 tendremos la siguiente función a trozos:

$$f(x) = \begin{cases} 1, & \sum w * x \geq 8 \\ 0, & \sum w * x < 8 \end{cases} \qquad (1)$$

Modificando la estructura pasando el umbral al otro lado de la inecuación y expresando este último como una variable obtenemos una expresión más generalizada dentro del mundo de las redes neuronales y que presenta un nuevo término, la bias.

$$f(x) = \begin{cases} 1, & \sum w * x + b \geq 0 \\ 0, & \sum w * x + b < 0 \end{cases} \qquad (2)$$

La **bias (b o sesgo)** es el valor opuesto al umbral y nos indican como de fácil o difícil es mostrar como salida el umbral. Cuanto mayor sea la bias menor pueden ser nuestras entradas para arrojar un resultado positivo

La salida de un perceptrón asimismo puede ser la entrada de otro y encadenarse formando complejos sistemas introduciendo, ahora sí, el concepto de redes neuronales (véase la figura 4). La modificación de pesos y la bias por lo tanto son los únicos elementos que componen y alteran nuestro sistema y que determinarán la salida que obtendremos. Para obtener el mejor rendimiento de la red cambiaremos los parámetros desde las neuronas más cercanas a la salida hasta las más alejada, por considerar que las primeras tienen mayor impacto en la salida producida. A esta técnica se la conoce como algoritmo de retropropagación o *backpropagation* en inglés.

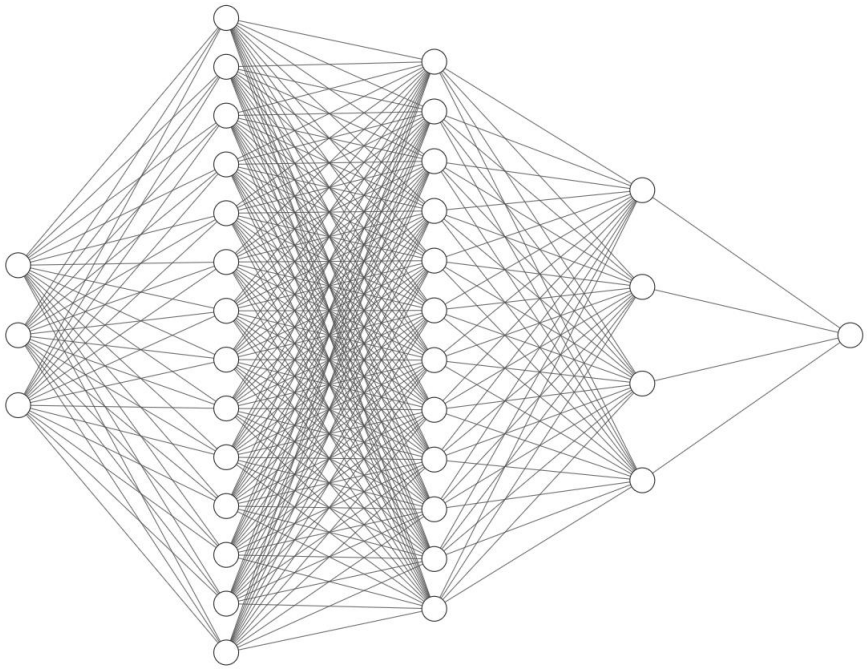


Figura 4. Esquema de red neuronal con tres entradas, tres capas ocultas y una salida. Esquema realizado en NN-SVG (http://alexlenail.me/NN-SVG/)

Se pueden observar los cientos de conexiones que existen entre las neuronas en las diferentes capas, al igual que en el cerebro humano. Cuanto mayor sea el número de conexiones, mayor variedad y facilidad tendrá la red para ajustar los parámetros de cada una de las neuronas.

Entrenar una red neuronal por lo tanto consiste en modificar los pesos y las bias para obtener el resultado deseado o esperado. Esto es, a grandes rasgos, buscar los valores de nuestros hiperparámetros para minimizar lo más

posible la función de coste. Para ello los perceptrones suponen un primer acercamiento al Machine Learning, pero al tener una salida en escalón en forma de ceros y unos provocan a la fuerza que la salida se modifique bruscamente al modificar ligeramente los parámetros de nuestra red. El modelo que buscamos debería cambiar ligeramente su salida si así lo hacen su peso o bias. Para ello es necesario hablar de las neuronas sigmoidales, cuya expresión genera este escalón más suavizado y que por primera vez introduce valores intermedios entre 0 y 1.

$$\phi(z) = \frac{1}{1 + e^{-z}} \qquad (3)$$

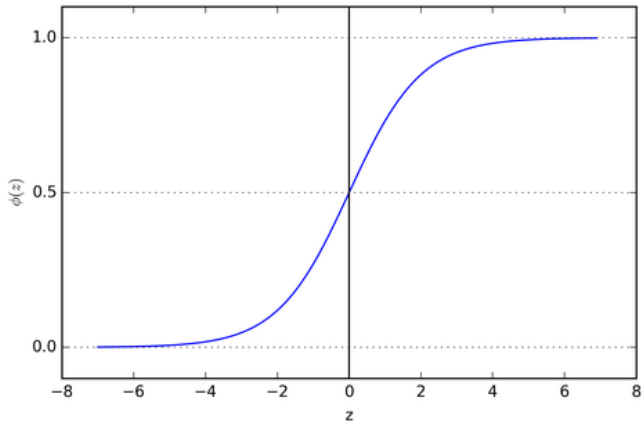


Figura 5. Representación gráfica de la salida en neuronas sigmoidales.

Este cambio implica que si, por ejemplo, pretendemos entrenar una red para que actúe como un reconocedor de mascotas, suponiendo que una salida de 1 es un perro y de 0 es un gato, la red modificará sus parámetros en cuanto de alejada esté de la realidad.

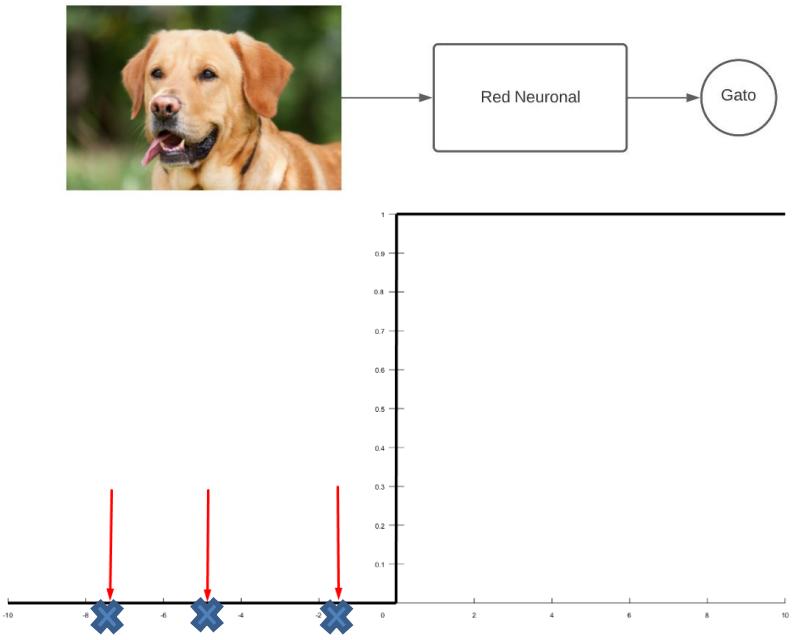
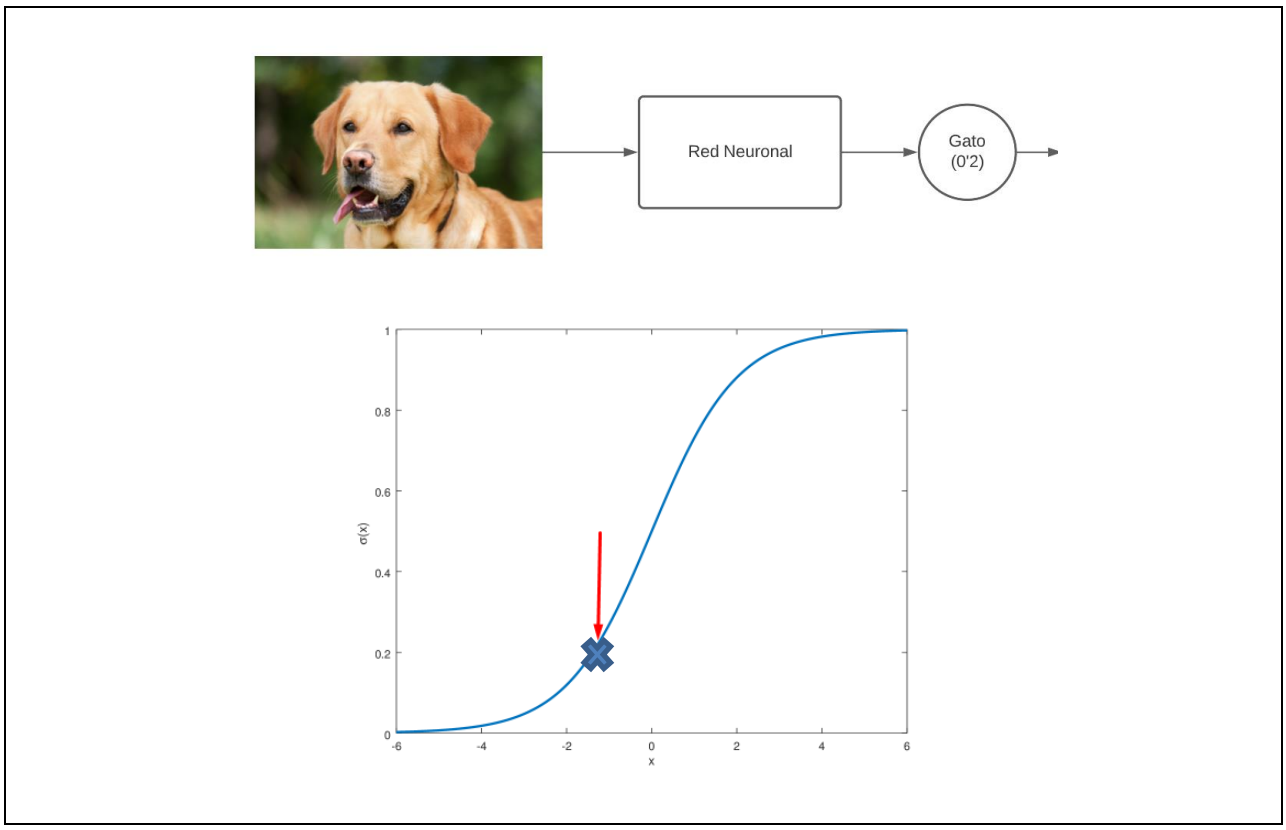


Figura 6. Si nuestra red está compuesta por perceptrones y se obtiene una predicción equivocada, el programa no sabrá cuánto ha de variar los parámetros de la red, ya que no sabemos a qué distancia nos encontramos de una predicción correcta.



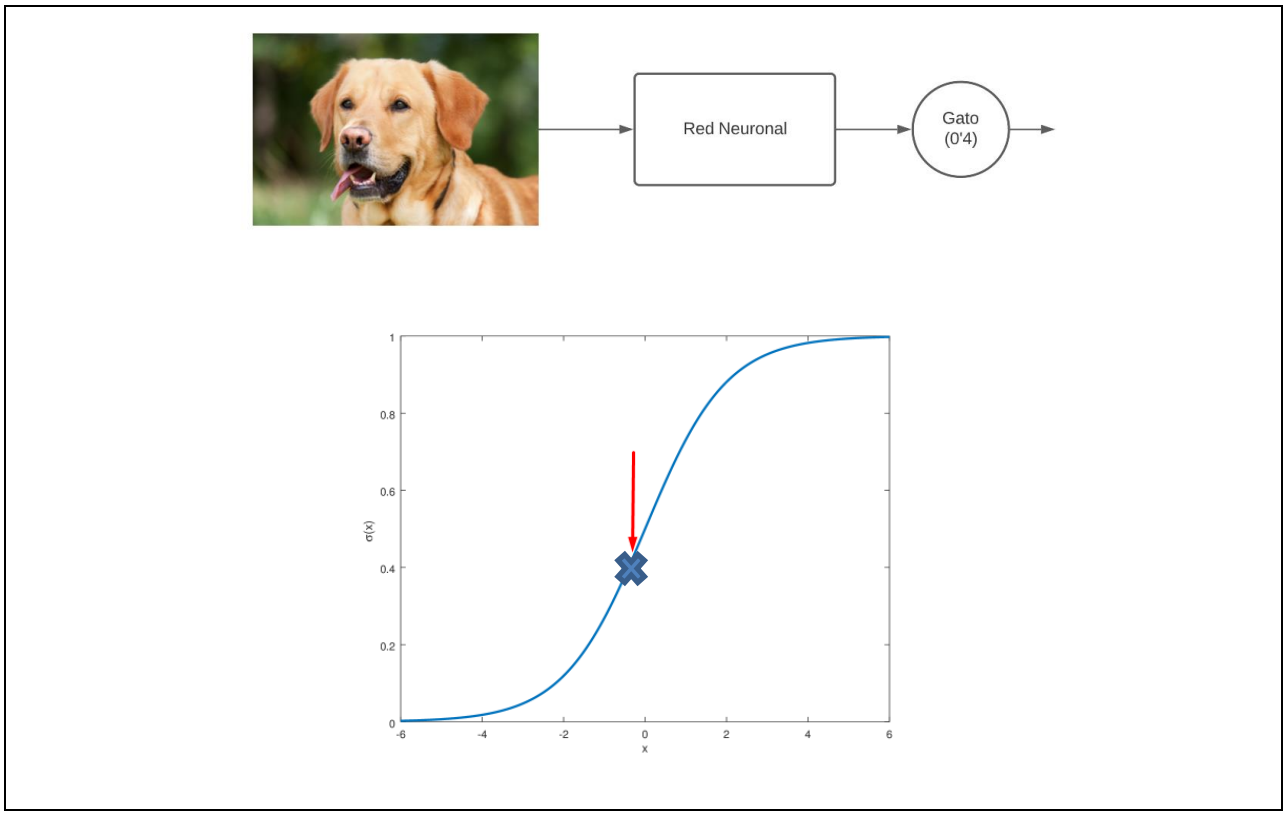


Figura 7. Ejemplos de corrección de pesos en una red con neuronas sigmoidales. Para el segundo caso las neuronas corregirán menos sus valores, pues se encuentran más cerca de predecir el resultado "perro", ya que obtiene un feedback de cuánto se está alejando de la predicción correcta.

Ya sabemos cuál es el criterio que utilizamos para modificar los valores de los parámetros. Pero ¿cómo sabe la red neuronal cuánto hay que modificar los pesos? Para ello se introduce el concepto de **descenso del gradiente**. Para visualizarlo mejor se ha representado una gráfica del error de nuestra red dependiendo de los parámetros introducidos. Los ejes x e y serían los parámetros de nuestra red y el eje z, el error. Buscamos acercarnos lo más posible a algún mínimo de la función partiendo desde un punto aleatorio y para ello nos aprovecharemos de la definición del gradiente, que apunta a la dirección en la que mayor es la pendiente y se lo restaremos al punto en el que estemos. Este proceso se realizará de forma continuada, descendiendo así por la función hasta encontrar el mínimo deseado. [17]

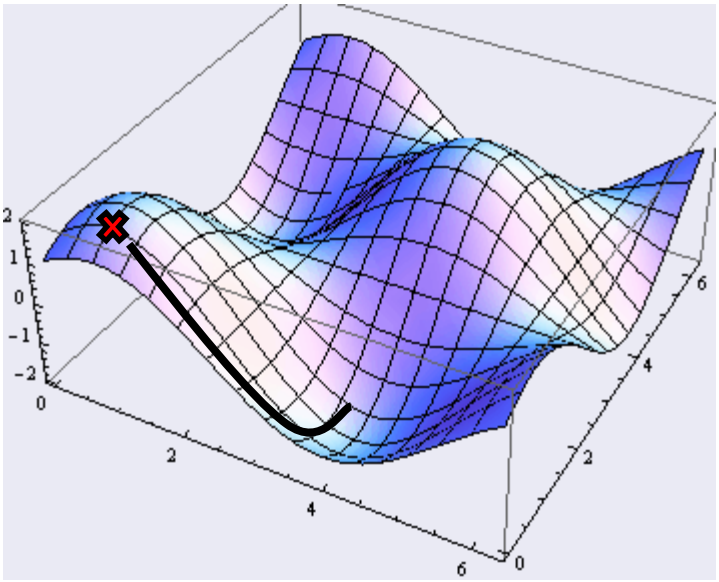


Figura 8. Representación gráfica de cómo actúa el descenso del gradiente en una función de error

Por último y para perfilar esta breve introducción a la estructura de una red neuronal, es necesario establecer los conceptos de capa de entrada, capas ocultas y capa de salida. La capa de entrada es la que recibe los valores dados por el usuario y los únicos que nosotros podemos controlar. Las capas intermedias contienen más neuronas con diferentes pesos y bias. Esto lo utilizaremos para ajustar de mejor forma, el comportamiento de la red. Cuantas más neuronas, más ajustes podrán realizarse para perfeccionar el comportamiento de la red, pero más complicada será de entrenar. Por último, tenemos la capa de salida que muestra los valores entre cero y uno obtenidos por sus neuronas y los resultados de nuestro sistema.

Por lo general las redes suelen tomar una estructura feedforward o sin realimentación, no utilizando las salidas de capas más profundas para capas a la izquierda de estas. Crear un sistema con realimentación podría provocar bucles donde la entrada de una neurona acabe dependiendo de su salida, situación no aconsejable. Aunque no nos extenderemos en el estudio de las redes feedback, son objeto de interés al retratar de manera más fiel cómo funciona y aprende el cerebro humano y algunos modelos cuyas neuronas se activan y desactivan durante un breve periodo de tiempo han probado obtener una curva de aprendizaje mayor en menos tiempo que sus homólogos más simples. [17]

3.1 Diseño de las Redes Neuronales

Explicados los componentes de las redes neuronales, comencemos creando y analizando nuestra primera red neuronal y con la que crearemos un primer modelo para predecir la salida dada una entrada. Como se comentó en el apartado anterior, las neuronas de las redes que usaremos se componen de pesos y bias, pero ¿Es necesario realizar un procedimiento complejo para su entrenamiento? ¿Es capaz nuestro programa de sacar una función válida únicamente a través de entradas y salidas?

Pongamos como ejemplo un programa que introduciendo el número de minutos que hemos estado hablando por

teléfono nos indicase cuánto debemos pagar en nuestra factura. Este ejercicio que en un principio resulta fácil de diseñar con métodos de programación regular servirá para entender de forma práctica cómo se comporta una red neuronal. Nuestra función por calcular será la siguiente

$$C(€) = 0.12 * t(\text{min}) + 0.54 * n + 5 \qquad (4)$$

Lo que quiere decir que el minuto cuesta 12 céntimos y el establecimiento de llamada, cincuenta y cuatro. Además, nuestra tarifa base es de 5 euros.

Por lo tanto, algunos de nuestros costes asociados podrían ser:

Tiempo (minutos)	Número de llamadas	Tarifa (euros)
200	10	**34.40**
145	12	**24.02**
300	20	**51.80**
32	1	**9.38**
0	0	**5**
100	60	**49.4**
1	1	**5.66**

Tabla 2. Muestra del coste del contrato dependiendo del tiempo de llamada y del número de estas

Vamos a programar nuestra red neuronal para que tenga la siguiente forma:

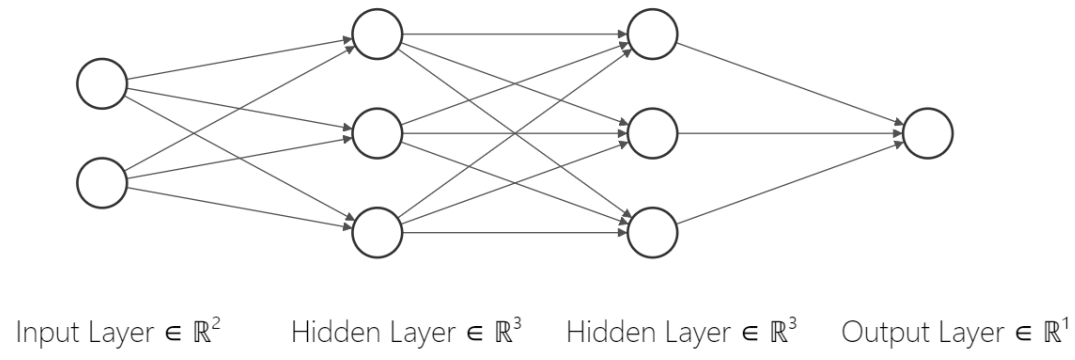


Figura 9. Esquema de la red neuronal propuesta

Nuestro programa internamente en cada iteración modificará los valores de los pesos y sesgos de forma que el error vaya siendo cada vez menor. De esta forma, se establecerán relaciones entre las neuronas que componen la red y sus parámetros. Toda red neuronal sigue el siguiente esquema para ser entrenada:

1. Se inicia la red con pesos y bias aleatorios
2. Calcular error de cada muestra y promediar el error de cada una de ellas.
3. Relacionar cada peso y sesgo con la importancia que han tenido a la hora de generar ese error.
4. Modificar los parámetros de forma proporcional (mayor error, mayor cambio) de forma que el error se reduzca
5. Repetir los pasos 2, 3 y 4 hasta conseguir que el error se reduzca lo máximo posible.

La optimización realizada y el ajuste de pesos y bias se produce, como ya se comentó anteriormente, mediante el algoritmo de retropropagación (*backpropagation*) y la optimización por descenso de gradiente (*gradient descent*).

Utilizando el lenguaje de programación Python definiremos cada una de las capas de la red neuronal y compilaremos el modelo completo

```
oculta1 = tf.keras.layers.Dense(units=3, input_shape=[2])
oculta2 = tf.keras.layers.Dense(units=3)
salida = tf.keras.layers.Dense(units=1)
modelo = tf.keras.Sequential([oculta1, oculta2, salida])
```

Figura 10. Implementación de la red neuronal

Lo siguiente será usar un optimizador para indicar la forma en la que se ajustan los parámetros. La función general para el cálculo de los pesos como explicamos anteriormente es la siguiente, aunque introduciendo un nuevo parámetro:

$$w_{t+1} = w_t - \frac{\delta f(coste)}{\delta w} * lr \qquad (5)$$

Es decir, el peso nuevo sería el peso antiguo menos el gradiente de la función de coste para minimizar el error. El parámetro *lr* sería el llamado factor de entrenamiento o aprendizaje (*learning rate*). Los optimizadores son los encargados de calcular un parámetro adecuado para el mismo. Un valor de lr muy bajo comprometería a nuestra red modificándola de forma muy lenta o estancándose en un porcentaje de error. Por el contrario. Un factor de entrenamiento muy elevado provoca una inestabilidad permanente al realizar cambios de pesos muy bruscos. [18]

Afortunadamente y gracias a los avances en este campo, tenemos a nuestra disposición una gran cantidad de optimizadores que pretenden buscar un mínimo local, siendo los principales [19] [20]:

- Stochastic Gradient Descent (SGD)
- Adaptative Gradient Algorithm (AdaGrad)
- Adadelta
- RMSprop (Root Mean Square Propagation)
- Adam

Las mejoras en los optimizadores más recientes surgen como combinación o a partir de algunos modelos anteriores, por lo que siempre se espera que los modernos sean superiores a los antiguos. Para nuestro ejemplo

utilizaremos el optimizador Adam.

Por último, para evaluar el comportamiento de nuestra función indicaremos qué función de pérdida tomaremos como referencia, que mide la desviación entre predicciones y resultados reales. Unos valores pequeños indicaran que nuestra red neuronal funciona de manera eficiente y ajustando bien el modelo y lo contrario para unos valores altos. Que la función de pérdida vaya tomando cada vez valores más cercanos a cero significa que el ajuste de los pesos se está realizando de forma correcta. Para este ejemplo utilizaremos el error cuadrático medio.

```
modelo.compile(
    optimizer=tf.keras.optimizers.Adam(0.1),
    loss='mean_squared_error'
)
```

Figura 11. Compilación del modelo de ejemplo

Para entrenar nuestro modelo utilizaremos a partir de ahora la función *fit*, que tiene como parámetros las entradas y salidas utilizadas para el entrenamiento y el número de repeticiones para entrenar a la red con los datos de entrenamiento. Una vez finalizada analicemos la función de pérdida:

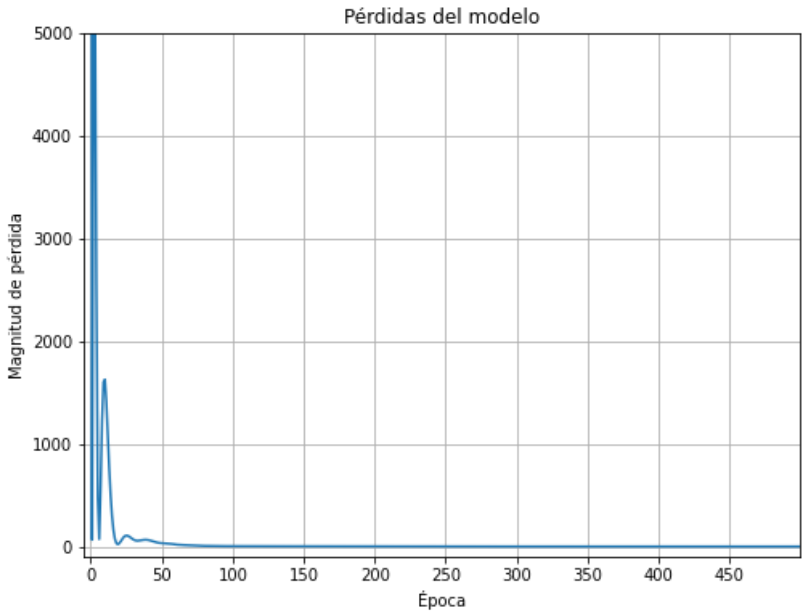


Figura 12. Función de pérdida del modelo de red simple

Podemos observar que a las 100 iteraciones (épocas) el error ha conseguido descender lo suficiente para ajustarse con precisión al modelo esperado. Hagamos una prueba con unos valores que no aparezcan en la tabla proporcionada al programa.

```
print("Hagamos una predicción!")
llamada=np.array([[20,10]], dtype=float)
resultado = modelo.predict([llamada])
print(resultado)
print('El resultado para ' + str(llamada[0][0]) +
      ' minutos en ' + str(round(llamada[0][1]))  + ' llamadas es ' + str(round(resultado[0][0],2)) + '€')
```

```
Hagamos una predicción!
1/1 [==============================] - 0s 12ms/step
[[12.799995]]
El resultado para 20.0 minutos en 10 llamadas es 12.8€
```

```
print("Hagamos una predicción!")
llamada=np.array([[0,0]], dtype=float)
resultado = modelo.predict([llamada])
print('El resultado para ' + str(llamada[0][0]) +
      ' minutos en ' + str(round(llamada[0][1]))  + ' llamadas es ' + str(round(resultado[0][0],2)) + '€')
```

```
Hagamos una predicción!
1/1 [==============================] - 0s 14ms/step
El resultado para 0.0 minutos en 0 llamadas es 5.0€
```

Figura 13. Código y resultados obtenidos

El resultado, aun contando con un muy pequeño error, es lo suficiente preciso para determinar el precio que pagaremos por nuestro consumo. El algoritmo partiendo de la nada ha conseguido encontrar unos valores para los pesos adecuados para que nuestra red neuronal se comporte como la ecuación de la que partimos.

A la vista del éxito de las predicciones y sabiendo lo simple que ha resultado programar este sistema dotado de Inteligencia Artificial podemos tener la falsa idea de que nuestro programa será resultado de extrapolar lo visto hasta ahora con, quizás, una red más densa. El programa y la red presentada presentan entre otras cosas los siguientes inconvenientes:

- **La cantidad de datos es muy pequeña**. Nosotros sólo hemos dotado a nuestra red de unos pocos valores, aunque suficientes para nuestro problema (recordemos que es posible hallar la ecuación de un plano; que sería la representación gráfica de la función dada, sabiendo únicamente 3 puntos). En el problema que queremos resolver necesitaremos muchísimos más datos ya que existen muchos tamaños, fuentes y sangrías diferentes para cada uno de los caracteres
- Y el más importante de todos: nuestro problema tiene una solución **lineal** y en casos similares como el de ecuaciones de rectas, planos o hiperplanos podremos utilizar esta red o cualquier otra que la eficiencia será la misma.

Al estar toda la red formada por neuronas que multiplican los valores de entrada por un peso y le suma un sesgo (operación lineal) para ser entrada de otras neuronas lineales el resultado por comparación también lo será. Por muchas capas que pongamos por lo tanto obtendremos el mismo resultado que si únicamente hubiéramos puesto una capa.

Para un mismo problema lineal las redes de la figura 14 tendrán el mismo comportamiento y alcanzarán unos resultados idénticos, por lo que parece poco intuitivo usar estructuras complejas más difíciles de entrenar.

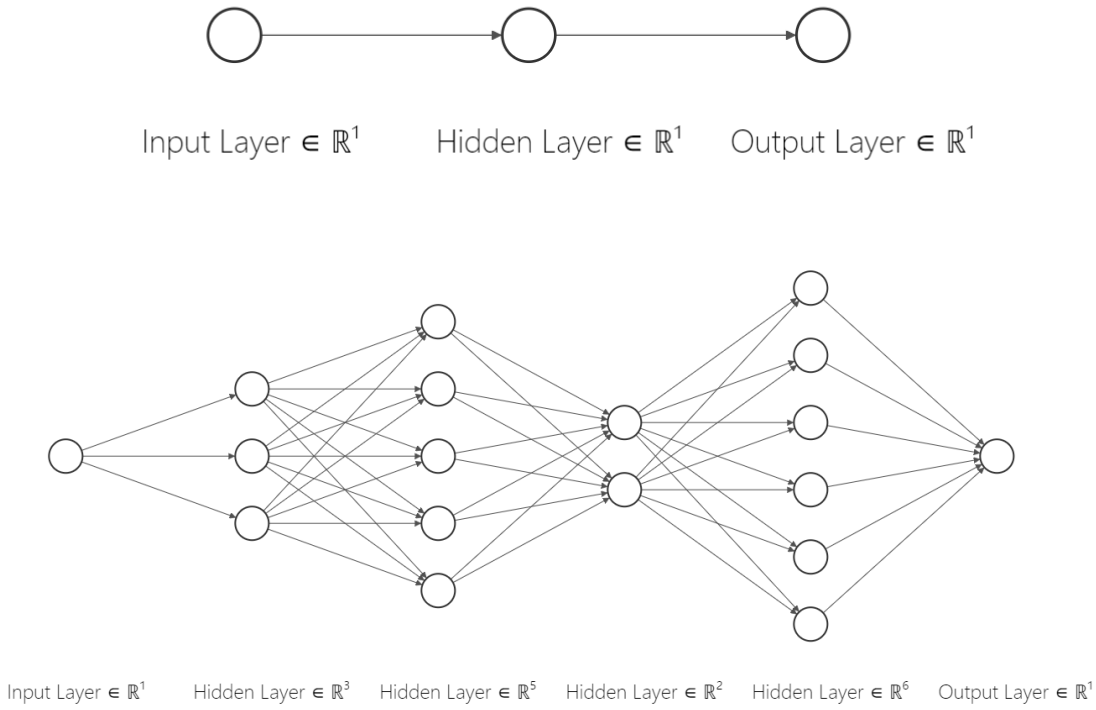


Figura 14. Ejemplos de dos redes neuronales.

Los problemas del mundo real no siguen por lo general un comportamiento lineal, lo que una vez más vuelve a complicar nuestra tarea de lograr un reconocimiento de caracteres lo más preciso posible. En el siguiente apartado intentaremos modificar la red creada para que la combinación de todas salidas las sean ecuaciones no lineales.

3.2 Salir de la linealidad: Funciones de activación

Ya hemos modelado nuestra primera red neuronal pero restringida y limitada por la linealidad. Para solucionar este problema se implementaron las funciones de activación: funciones a las que se le pasa la salida de una neurona antes de entrar a la siguiente y eliminan ese comportamiento lineal. Cada función de activación que definamos tendrá por lo tanto como entrada la suma ponderada de las entradas multiplicadas por los pesos y la suma de la Bias.

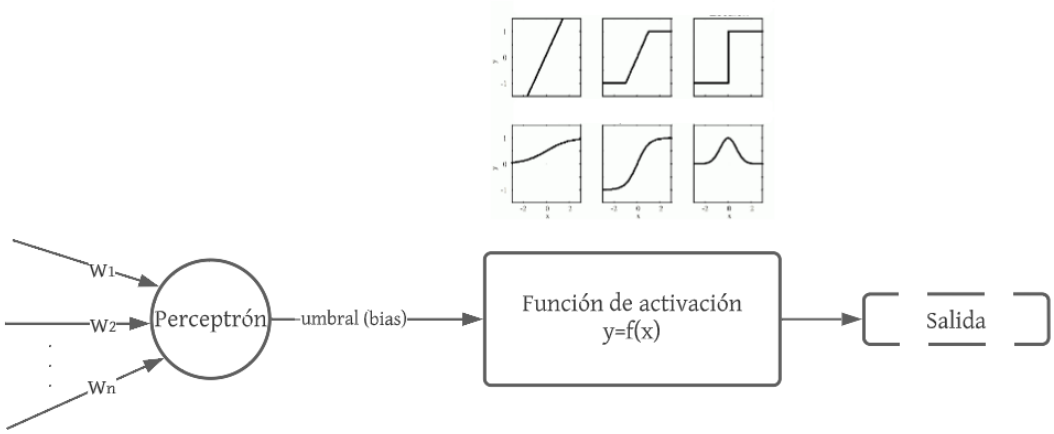


Figura 15. Esquema de una neurona incluyendo la función de activación

Conceptualmente ya vimos dos funciones de activación en la introducción que se dio al mundo de las redes neuronales. La primera es la función de activación en escalón, que no es útil para los sistemas de Inteligencia artificial. Al tener derivada nula en casi la totalidad de su dominio no permite adecuar correctamente los sesgos, por lo que a partir de ahora lo que buscaremos en una función de activación es que sea diferenciable. Esta condición la cumplía, como podemos comprobar, la función de activación sigmoidal.

No obstante, las funciones de este tipo introducen un problema en los modelos llamado "problema del **desvanecimiento del gradiente**" [19]. Al ir mejorando nuestra red neuronal observamos que nuestro gradiente es cada vez más pequeño por lo que irremediablemente nuestro programa al llegar a determinadas iteraciones dejará de mejorar significativamente y mantendrá un error residual. Aun así, estas funciones son útiles en la capa de salida de las redes neuronales y siguen siendo las más utilizadas en las capas de salida para problemas de clasificación binaria como el de perros y gatos. Una mejora de esta función sería la tangente hiperbólica, que cuenta con un mayor recorrido (de -1 a 1) y su derivada es mayor, lo que implica un aprendizaje más rápido. Como nuestro problema pretende clasificar caracteres en más de dos categorías, estas funciones no serán útiles y se ha optado por descartarlas para las capas ocultas.

Actualmente la función de activación más utilizada en Machine Learning y la que aplicaremos más adelante en nuestro proyecto es la llamada ReLU (Rectified Linear Unit), que tiene la siguiente forma:

$$f(x) = \max(0, x) \qquad (6)$$

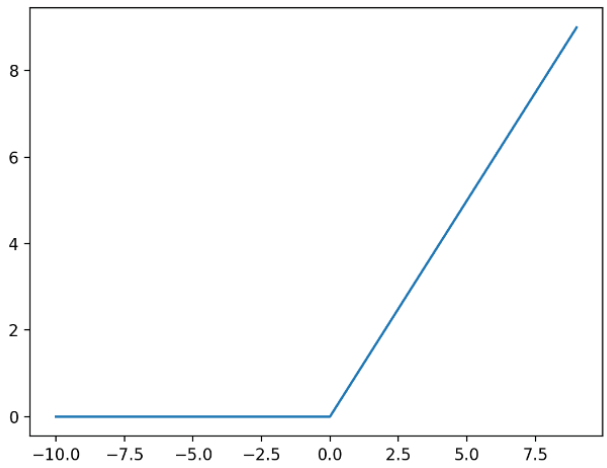


Figura 16. Representación gráfica de la función de activación ReLU

Esta función presenta dos ventajas fundamentales respeto a las anteriores. La primera es que el gradiente de la

función para valores positivos es constante y la función no está acotada, lo que permitirá acelerar el aprendizaje. Otro motivo por el que nos podría interesar elegir esta función es por su bajo coste computacional [21]. Al no depender de exponenciales u otras fórmulas matemáticas más complejas su velocidad de procesamiento es menor y permite obtener resultados similares a los de otras funciones en menos tiempo (hasta 6 veces más rápida que la tangente hiperbólica según el artículo *Deep Sparse Rectified Neural Networks* [22]). Por el contrario, el dominio negativo de la función hace que exista la posibilidad de que se generen "neuronas muertas" que únicamente dan como salida cero, entorpeciendo el aprendizaje. Actualmente, y aunque han surgido nuevas variantes de esta (Leaky ReLU, PReLU y GeLU entre otras), ReLU sigue siendo la función de activación más utilizada en las capas ocultas y será aquella que utilizaremos para implementar tanto nuestro propio modelo como otros desarrollados con base en esta función, como ResNet.

Por último, resulta de interés para nuestro problema estudiar la función Softmax [20] Utilizada principalmente en la salida de las redes neuronales encargadas de tareas de clasificación. Esta función nos dará para cada una de nuestras salidas una probabilidad entre 0 y 1 y se quedará con la mayor. Calculando el exponente natural de cada una de las salidas y dividiéndolo entre el sumatorio de los resultados conseguimos que los números grandes tengan mucho mayor porcentaje que los pequeños. Será por lo tanto de gran utilidad incluir Softmax en nuestra red en cualquiera de las pruebas que hagamos de cara a obtener la fiabilidad con la que nuestra red detecta los caracteres y determinar si el valor detectado es útil o no.

Para concluir con esta introducción, cabe destacar el campo de la Inteligencia Artificial es objeto de estudio e investigación constante, por lo que en los últimos años surgirían otras funciones que sin ser monotónicas han mejorado el rendimiento de ReLU, aunque a costa de introducir una mayor carga computacional. Algunas de las que se han incorporado al vasto catálogo de funciones de activación son Swish (2017) y Mish (2020) por enumerar algunas [23], siendo ya tarea del programador elegir la más adecuada para cada necesidad y capa.

Habiendo explicado cada uno de los componentes de las redes neuronales y los parámetros necesarios para diseñarlas correctamente, finalmente comenzaremos con el problema que motivó la realización de este Trabajo de Fin de Grado y que pasaremos a explicar a continuación: partir de un archivo vacío y construir, obteniendo los resultados óptimos, una red neurona funcional que detecte mediante imágenes o vídeo letras de forma correcta.

4 CREACIÓN DE UN CLASIFICADOR DE IMÁGENES PERCEPTRÓN MULTICAPA

El objetivo principal del proyecto es que nuestro programa tome una imagen de una letra o número y sepa clasificarla correctamente. Para ello es de gran ayuda fijarse en otros clasificadores de imágenes como alguno que simplemente sepa diferenciar números entre sí para abstraer las características que debe tener nuestro programa. [24] [25]

4.1 Propuesta de trabajo y requisitos de hardware y software

Los requisitos autoimpuestos para que el proyecto pueda usarse en casos reales son los siguientes

- El programa debe tener como entrada los píxeles de la imagen. En la realidad las imágenes analizadas pueden variar su resolución por lo que habrá que realizar un preprocesado para cambiar el tamaño de la imagen.
- Las entradas serán en blanco y negro y los píxeles se normalizarán (tomarán valores entre cero y uno) para facilitar el proceso de aprendizaje del algoritmo.
- En la capa de salida obtendremos para cada carácter un número que nos indicará la posibilidad de que el carácter sea ese en concreto. Se tomará como "carácter reconocido" aquel con un mayor porcentaje y que supere cierto umbral.
- Deberemos de adaptarnos a cambios de orientación y posición que pueda tener el carácter en nuestra imagen y reconocerlo correctamente, siempre que el cambio no genere una confusión evidente con otro (por ejemplo, darle la vuelta a un 6 y confundirlo con un 9)
- Nuestro algoritmo deberá tener un porcentaje de éxito adecuado para los estándares actuales.

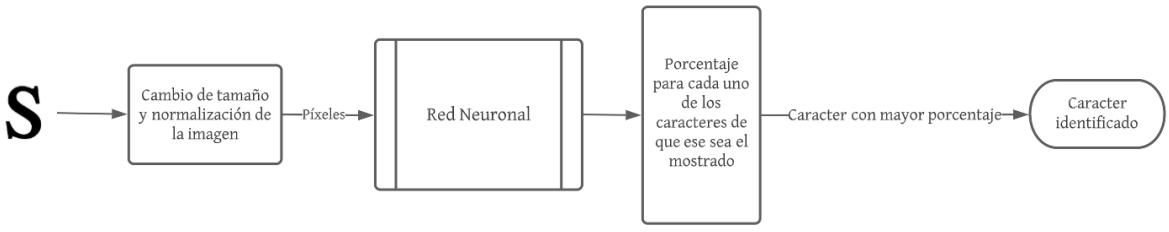


Figura 17. Esquema del programa

4.1.1 Requisitos de Hardware.

Entrenar redes neuronales desde 0 puede ser algo costoso computacionalmente, ya que estas requieren de gran cantidad de datos para conseguir superar a los humanos o al menos lograr la precisión requerida.

Nuestro programa debe ser capaz de determinar la mejor combinación de parámetros para trabajar con una arquitectura que nosotros consideremos "óptima" para la tarea a realizar. En este proyecto se ha utilizado un sistema con las siguientes características:

CPU	I5-7600K CPU 3.80 GHz
RAM	16 GB
ROM	250 GB SDD | 2 TB HDD
GPU	GeForce GTX 1060 6GB OC
S.O.	Windows 10 Pro

Tabla 3. Especificaciones de la computadora local.

Para entrenamientos con un mayor número de iteraciones se ha recurrido a Google Colab, una herramienta para ejecutar código escrito en Python directamente desde el navegador utilizando una computadora virtual. Esta cuenta con una potente tarjeta gráfica para acelerar el proceso de entrenamiento, aunque por contraparte funciona en una máquina virtual y todos los procesos finalizan tras un período de tiempo perdiendo cualquier avance no guardado.

CPU	Intel Xeon CPU 2x2.30 GHz
RAM	12 GB
ROM	25 GB
GPU	Nvidia T4 12 GB

Tabla 4. Especificaciones de la máquina virtual de Google Colab [26].

4.1.2 Requisitos de Software.

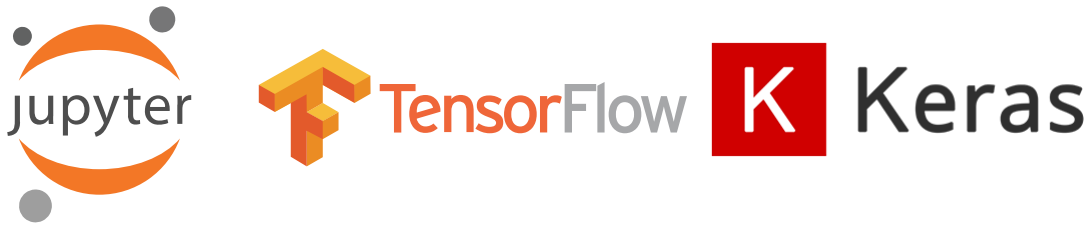


Figura 18. Logotipos de los programas *Jupyter*, *TensorFlow* y *Keras*, herramientas de desarrollo de código y librerías para la implementación de programas de Deep Learning [27] [28] [29]

En el apartado del software en nuestro proyecto emplearemos librerías que usaremos en Python, lenguaje por excelencia en el estudio del Deep Learning, para diseñar de forma más sencilla redes neuronales que luego entrenaremos usando el entorno de Jupyter. Entre todas las que se han incluido, que se encontrarán en el Anexo de este proyecto, destacan Tensorflow y Keras, herramientas utilizadas hoy día en casi la totalidad de proyectos relacionados con la Inteligencia Artificial y cuyas funciones iremos describiendo a lo largo del desarrollo del programa.

La primera fue desarrollada directamente por Google y es una librería de operaciones matemáticas utilizadas en el ámbito del aprendizaje profundo, que incluye las funciones necesarias para:

1. Realizar un preprocesamiento de la información.
2. Construir un modelo propio con las capas que nosotros elijamos.
3. Entrenar y mostrar datos estimados de precisión y error.

En cuanto Keras, esta es una Interfaz de Programación de Aplicaciones (API) relacionadas con el Deep Learning, escrita en Python y usada para una sencilla implementación de redes neuronales con un alto nivel de abstracción y con resultados en tiempo real accesibles al usuario a la vez que se ejecutan por detrás las distintas operaciones computacionales. Esto hace a Keras un entorno algo más lento en comparación a otros entornos de trabajo, pero una herramienta ideal para introducirse en el mundo de la inteligencia artificial y ha sido adoptado por Tensorflow como su API oficial. [30]

4.1.3 Diseño inicial.

En un principio, con los conocimientos adquiridos acerca del aprendizaje de las redes neuronales puede parecer lógico usar una red parecida a la del ejemplo (con funciones de activación ya que nos encontramos ante un problema no lineal) y luego modificar parámetros como número de capas ocultas, neuronas en cada capa entre otros para conseguir que nuestro programa aprenda a reconocer los caracteres. Para obtener la mayor precisión posible hemos de contar con la mayor cantidad posible de datos, tanto para test como para la validación. Aquí introducimos el concepto de los **datasets**, Un dataset no es más que una base de datos con filas que representan los datos que tenemos y cada columna es una variable o atributo de estos [27]. Un ejemplo de este podría ser un fichero o un sitio web que contenga en cada fila una casa de una determinada región y cada columna sea una característica de esta (precio, número de dormitorios, si tiene patio o no…). Por lo general la primera columna corresponde con la variable de salida y el resto de los parámetros son los datos de entrada de nuestra red.

Para la detección de números el dataset más famoso es el **MNIST** (que proviene de las siglas Modified National Institute of Standards and Technology dataset), considerado por algunas fuentes como el "Hola mundo" de la Inteligencia Artificial y en el cual se incluyen 70.000 números escritos a mano tanto por trabajadores del Censo de Estados Unidos y estudiantes y con una resolución de 28x28 píxeles; 60.000 de ellos se usan para entrenar la red y 10.000 sirven para comprobar cómo de bien funciona nuestra red. Esta base de datos proviene del NIST Special Database 19, que contiene también caracteres alfabéticos [31]. En un principio se intentó combinar MNIST con otros datasets de letras para conseguir una recopilación donde apareciesen todos los caracteres [32] [33]. El problema que surgió es la diferencia entre el número de datos de MNIST y el de otros datasets, que hacía que el algoritmo tendiese a identificar con más asiduidad un carácter como letra o como número. También variaban la resolución de las muestras por lo que esta no es una solución adecuada para nuestro problema de toma de muestras

Del MNIST surgiría en 2017 y tomando en cuenta la base creada por NIST el **EMNIST** (Extended MNIST) [34], que es una recopilación de varios datasets, esta vez con letras y números. Entre los datasets encontrábamos:

- ***ByClass*:** Contiene los mismos números de letras y de números que los que había en el NIST, aunque estos entre sí no tienen los mismos números de datos
- ***ByMerge*:** Este dataset contiene el mismo número de elementos que el anterior, pero en este caso se han fusionado aquellas letras que son similares siendo mayúsculas y minúsculas pretendiendo mejorar el porcentaje de clasificación correcta.
- ***Balanced*:** En este caso el número de clases es el mismo que el de ByMerge, pero se ha buscado

equilibrar el número de elementos en cada clase, haciendo de este dataset bastante útil en la mayoría de las tareas de clasificación.

- ***EMNIST Digits*** y ***EMNIST Letters*****:** Recopilación de todos los números y letras respectivamente, con un reparto equitativo en las variables. EMNIST letters va un paso más allá que ByMerge y une las letras mayúsculas y minúsculas para crear únicamente 26 clases.
- ***EMNIST MNIST:*** Dataset que replica la estructura del MNIST, explicado anteriormente.

Figura 19. Ejemplo de letras y números tomado de una de las bases de datos de EMNIST.

Por su aplicabilidad y fiabilidad para obtener resultados equilibrados utilizaremos el *EMNIST Balanced* para entrenar a nuestra red neuronal. Este cuenta con los siguientes datos:

- 112 800 caracteres para el entrenamiento
- 18 800 caracteres para el test
- 3 000 imágenes de cada carácter (2400 para el entrenamiento y 600 para las pruebas)
- 47 clases

Una vez elegido el dataset deberemos normalizar sus entradas para suavizar la carga computacional necesaria. Esto es, a grandes rasgos, transformar las imágenes a blanco y negro y almacenar sus valores de 0 a 1 en vez de 0 a 255 como estamos acostumbrados. Se creó para ello una función que recibiendo como valores de entrada el dataset y la etiqueta de la primera columna (que en nuestro caso es la que alberga los datos de salida) nos da como salida los datos normalizados.

```
def Normalizar(data, label):
    x = data.drop(label ,axis=1)
    x=x.astype('float32')
    x=x/255
    y = data[label]
    x= x.values.reshape(-1,28,28,1)
    return x,y
```

Figura 20. Implementación de la normalización de los datos

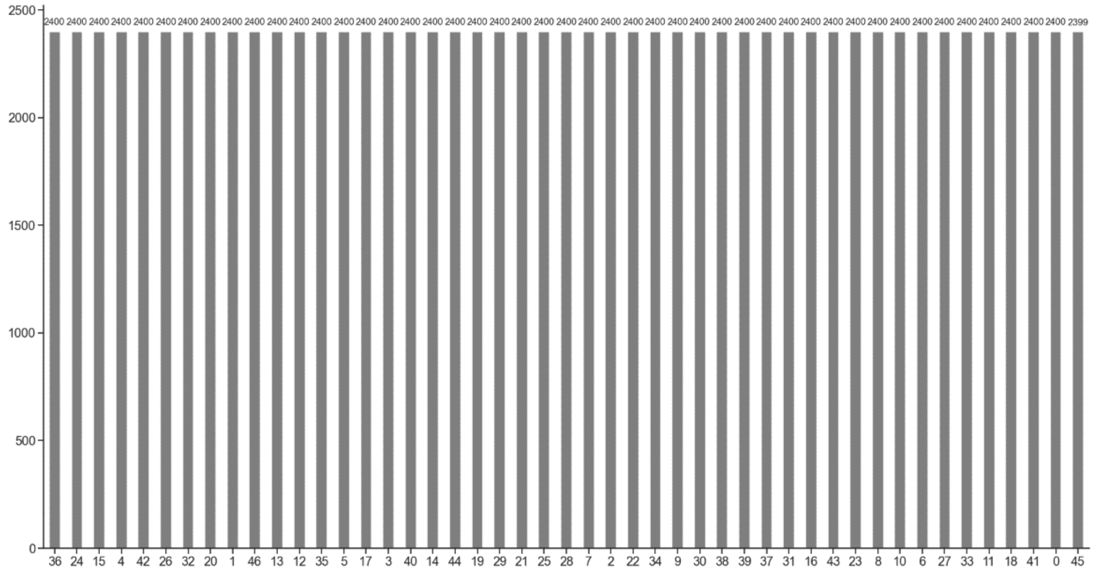


Figura 21. Distribución de los diferentes caracteres y su número de muestras.

Para comprobar que nuestra red está leyendo correctamente los caracteres mostraremos algunos al azar gracias a la librería de Python *matplotlib*. Gracias a esto pude darme cuenta de que los píxeles se leían por columnas y no por filas, lo que hacía que no se aprendiesen correctamente los caracteres. Esto se solucionó añadiendo una función que transponía la matriz de píxeles [35], esta vez consiguiendo el resultado deseado.

```
def Normalizar(data, label):
    x = data.drop(label ,axis=1)
    x=x.astype('float32')
    x=x/255
    y = data[label]
    x= x.values.reshape(-1,28,28,1)
    x=np.transpose(x,[0, 2, 1, 3]) #Hay que transponer los datos porque si no el caracter no se escribe correctamente
    return x,y
```

Figura 22. Código de la normalización con la transposición realizada

Para mejorar la legibilidad en este punto del desarrollo también se creó una lista con los caracteres a identificar para que se mostrasen en vez del número asociado. En la siguiente imagen observamos algunos ejemplos de las letras y números de nuestro dataset [36].

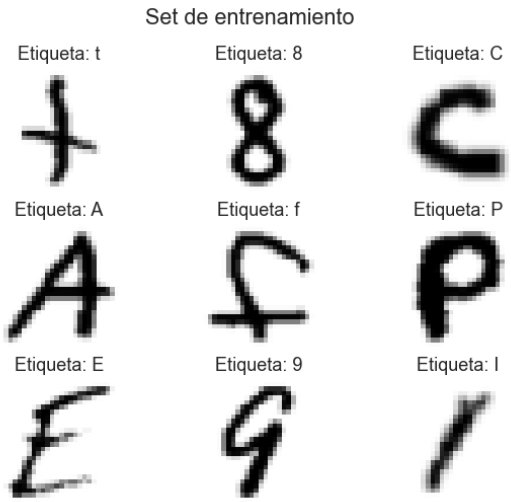


Figura 23. Algunos de los caracteres del set de entrenamiento, etiquetados

Una vez realizados los ajustes previos, se decidió realizar una prueba con una red lineal con funciones de activación. Como se puede observar en la figura 24, la capa de entrada consta de 744 neuronas (correspondientes a todos los píxeles de nuestra imagen de entrada de 28x28). Seguidamente introducimos dos capas ocultas de 25 y de 40 neuronas respectivamente. Por último, la capa de salida tendrá una neurona por cada carácter a reconocer, resultando en 47.

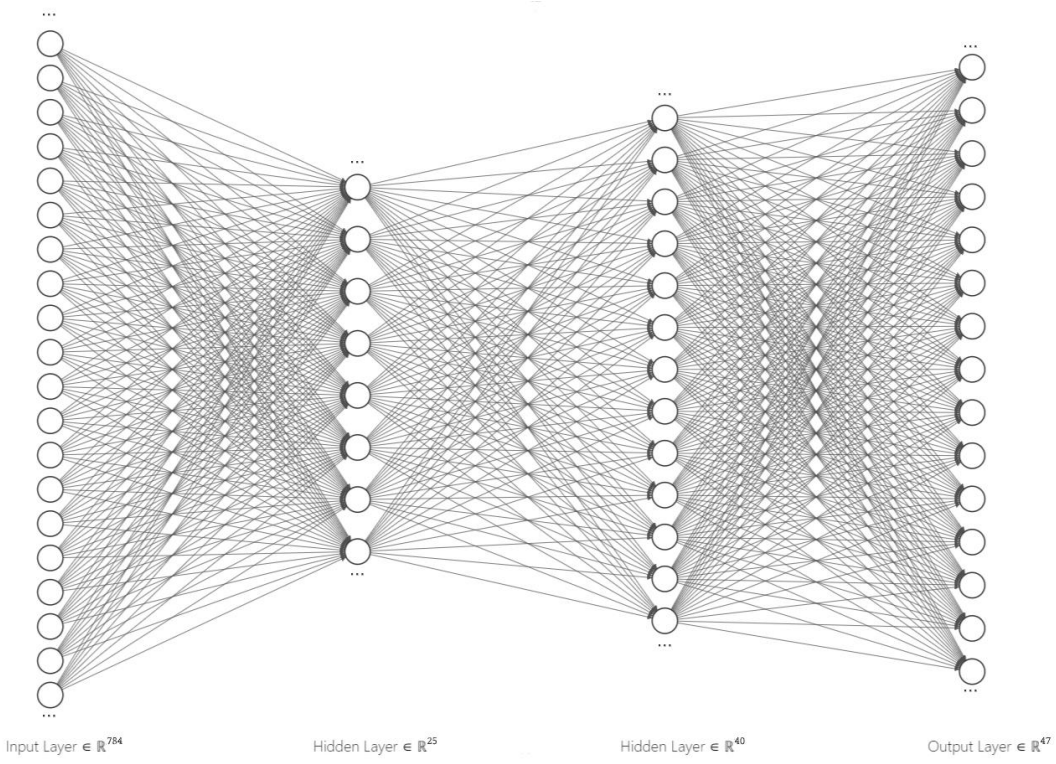


Figura 24. Diagrama de la red neuronal diseñada

```
Vas a usar el MODELO 1: Red Neuronal Simple
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 784)               0

 dense (Dense)               (None, 25)                19625

 dense_1 (Dense)             (None, 40)                1040

 dense_2 (Dense)             (None, 47)                1927

=================================================================
Total params: 22,592
Trainable params: 22,592
Non-trainable params: 0
_________________________________________________________________
Numero de capas de la Red Neuronal
4
```

Figura 25. Resumen de la red neuronal. Tiene como entrada cada uno de los píxeles, dos capas ocultas y en la capa de salida aparecen cada uno de los 47 caracteres reconocibles.

De esta red obtenemos los resultados expuestos en las figuras 26 y 27. Resulta de interés explicar y analizar las gráficas obtenidas, pues estas se utilizarán a partir de ahora en las pruebas que realicemos para valorar como de bien se comporta nuestra red con los datos de entrenamiento y de prueba:

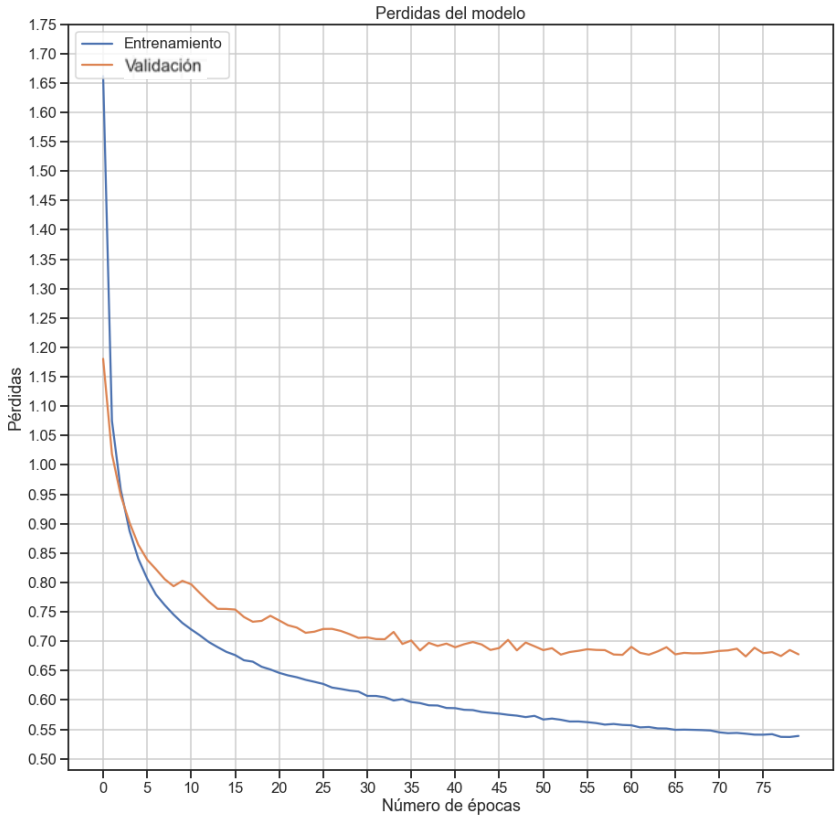


Figura 26. Gráfica de pérdidas del modelo 1

En la figura 26 se muestra la llamada “función de pérdida” o “función de coste” e indica cuánto varía el valor predicho con el valor real de la muestra a lo largo de las épocas (ciclos de propagación hacia detrás y hacia delante para ajustar los pesos de la red). Las pérdidas se calculan como el sumatorio de los errores para cada una de las muestras en los sets de entrenamiento y validación. Como ya se comentó en los apartados introductorios, esta función se utiliza para modificar los pesos de las neuronas y mejorar la red.

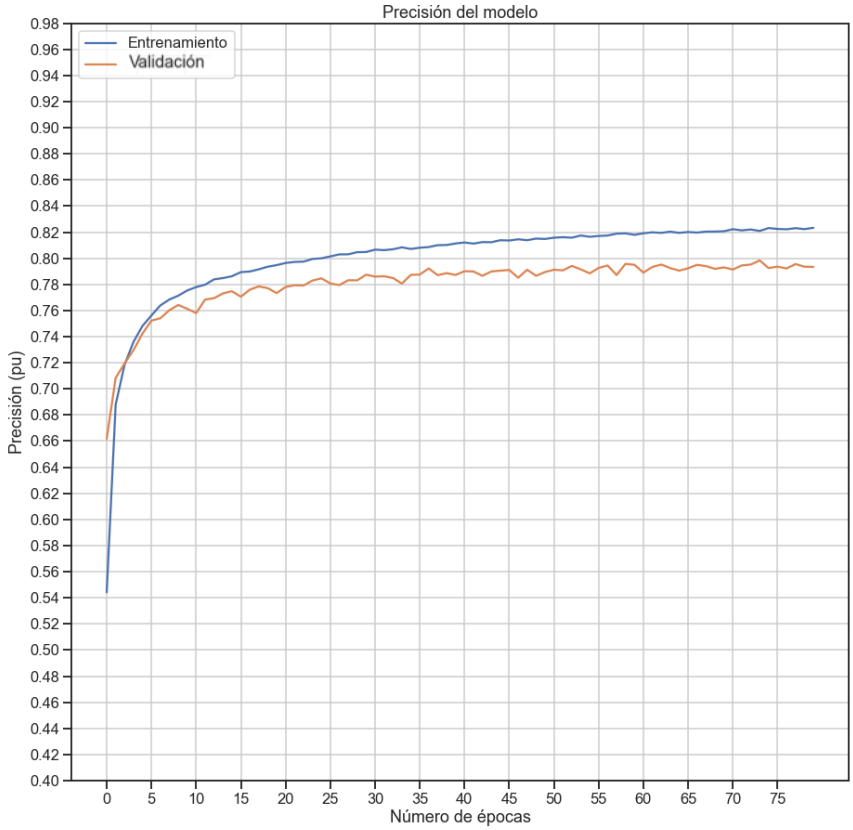


Figura 27. Gráfica de precisión del modelo 1

En la figura 27 observamos la función que nos indica el porcentaje de precisión de la red neuronal a lo largo del entrenamiento. En los entrenamientos esta precisión se traduce en el porcentaje de aciertos de nuestro sistema. Dado una entrada se comparan de forma binaria si el resultado es igual o distinto al esperado y se realiza esta operación para cada una de las muestras de nuestros sets.

Aunque no existe relación matemática entre las dos funciones antes vistas en la mayoría de los casos observaremos que cuando la gráfica de precisión crece la de pérdidas decrece, ya que una menor cantidad de pérdidas suelen implicar que los resultados del sistema serán mejores. Sin embargo, esto no siempre es así, sobre todo cuando se produce un sobreajuste en la red que hace que la precisión aumente, pero las pérdidas suban también.

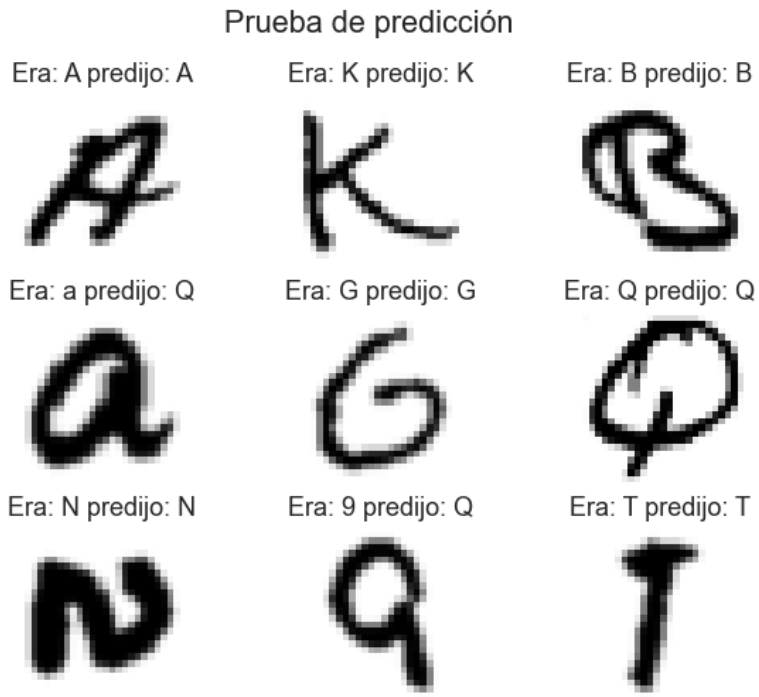


Figura 26. Ejemplos de predicciones con el set de pruebas

Como podemos observar, esta red no produce un resultado del todo insatisfactorio en el entrenamiento, pero a la hora de predecir elementos nuevos nos encontraremos conque no obtenemos ese nivel de precisión. Esto se debe a que nuestra red lo único que ha hecho es relacionar la intensidad de cada píxel con la salida, que para imágenes muy parecidas en tamaño y forma a las del entrenamiento da buenos resultados, pero en el momento que cambiamos la posición o tamaño del número o letra la red nos empieza a dar problemas.

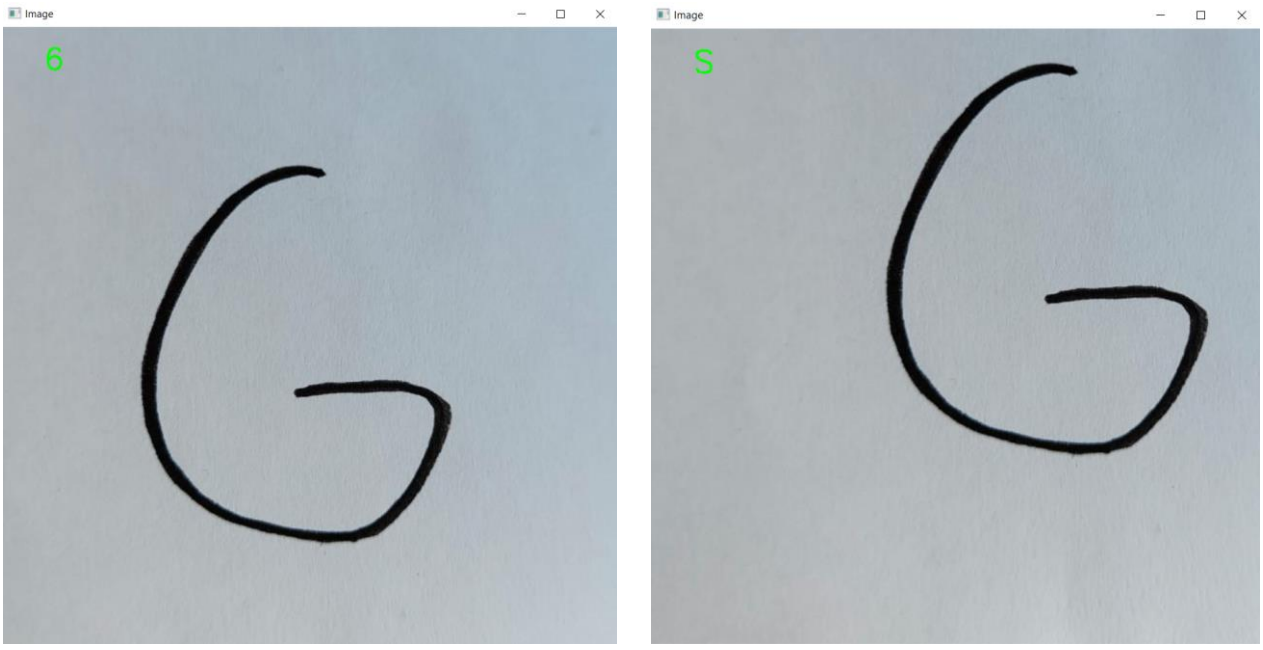


Figura 27. Ejemplos de predicciones con imágenes reales de una G.

En la imagen de la izquierda el programa nos detecta un 6 lo cual es incorrecto, pero hasta cierto punto asumible dado los resultados de precisión de nuestro modelo. Sin embargo, en la segunda imagen la cual es resultado de un desplazamiento de la primera obtenemos un resultado completamente diferente. Este comportamiento no es deseable y nuestro objetivo será eliminarlo. Es a partir de este momento cuando las redes vistas hasta el momento no son suficientes para abarcar nuestro problema y fue tras un análisis de otros modelos que pretendían solucionar tareas de clasificación de imágenes que se introducen otro tipo de redes no vistas hasta ahora: Las Redes Neuronales Convolucionales.

5 CREACIÓN DE UN CLASIFICADOR DE IMÁGENES: REDES NEURONALES CONVOLUCIONALES.

Este tipo de redes nos dan una aproximación distinta al problema de aprendizaje de nuestro programa. Hasta ahora los algoritmos ajustaban los pesos dadas unas entradas y salidas predeterminadas. Esta solución es ideal para entradas en forma de número o cadena de caracteres, siempre que estas definan características como podríamos pensar de un programa que calcula precios de viviendas basándose en entradas numéricas. Un píxel por sí solo no es una característica, pero un grupo de ellos formando un patrón determinado pueden serlo.

En determinado datasets, como el creado por la web de venta de ropa *Zalando* [37], las prendas vienen en una disposición predeterminada e igual al resto de elementos con la misma etiqueta. Aunque gracias a esto podamos facilitar el entrenamiento, también limitamos el aprendizaje del sistema pues un objeto tridimensional puede encontrarse en distintos escenarios y orientaciones.

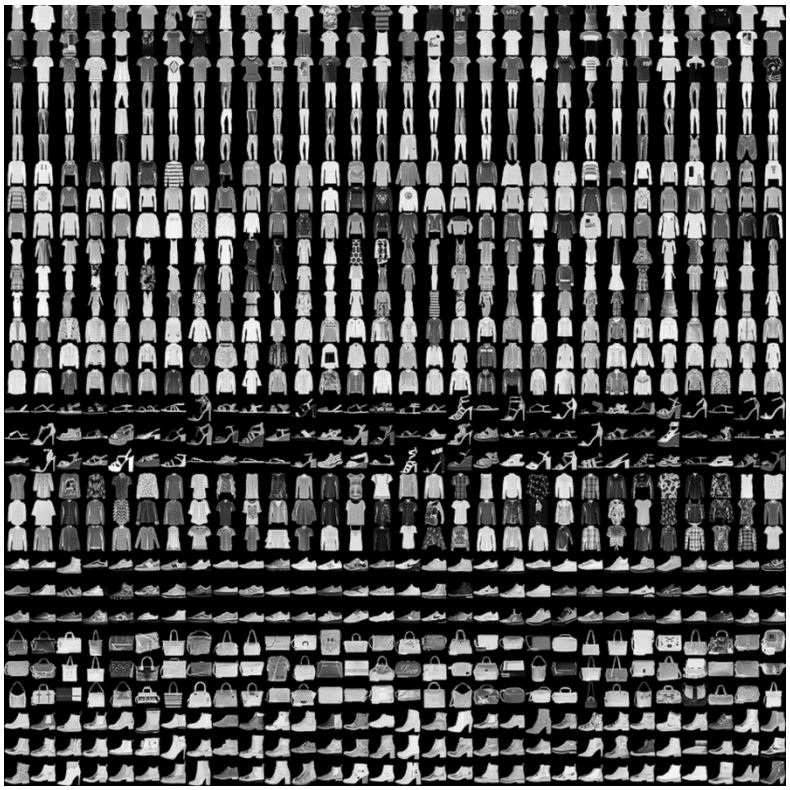


Figura 28. Muestras provenientes del Fashion MNIST Dataset de Zalando. [37]

Figura 29. Varios puntos de vista de una misma zapatilla deportiva (JD, s.f.) [38]

Por la tanto llegamos a la conclusión de que para este tipo de problemas una red neuronal al uso no es ideal, como se ha visto anteriormente. Las redes neuronales convolucionales nos acercan más al razonamiento humano.

Nuestro cerebro asocia a cada objeto patrones y formas. Si las características del elemento visualizado coinciden con algún concepto que tengamos le podremos asociar a ese objeto una salida, sin importar la posición, rotación o tamaño.

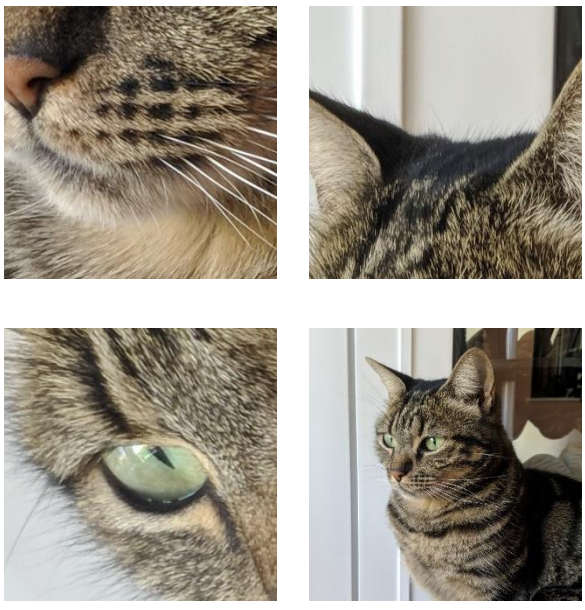


Figura 30. Partes de un gato. Nuestro cerebro internamente al ver la imagen la segmenta en patrones reconocibles, y el conjunto de ellos es lo que le permite identificar al animal como un gato, aunque rotemos, volteemos o modifiquemos de alguna forma la imagen

¿Cómo puede nuestra red neuronal conseguir eso? Para ello es necesario presentar dos nuevas capas que se sitúan después de las capas de entrada. Estas son las capas de convolución y de agrupación, que extraerán las características de cada carácter de manera autónoma para luego introducir estas características en las capas

ocultas y trabajar como las redes que hemos visto hasta ahora. Las dos capas imitan el córtex visual de nuestro cerebro, que se componen de neuronas simples que detectan líneas o ejes en una región muy concreta y las complejas que agrupan la información de neuronas simples para crear formas más concretas y encadenándose entre sí es como formamos patrones reconocibles por nuestro cerebro [39].

En primer lugar, para la detección de ejes será necesario un proceso de **convolución** de la imagen en la capa del mismo nombre, concepto estudiado en el campo de Sistemas de Percepción y que implica multiplicar un píxel y sus vecinos por una matriz llamada núcleo o kernel y sumar los resultados, obteniendo un nuevo valor para cada píxel.

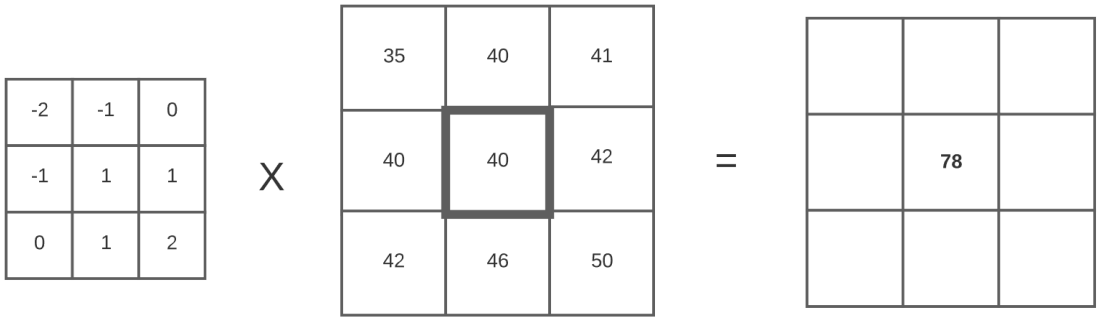


Figura 31. Esquema del proceso de convolución

En la red neuronal nosotros no especificamos los núcleos que se van a usar para la convolución de nuestra imagen, solo el número y el tamaño de estos, ya que al igual que vimos con los pesos, se ajustarán automáticamente.

La segunda capa, la capa de agrupación, reduce el tamaño de la imagen con el objetivo de extraer las características principales y que las mismas no dependan de la posición.

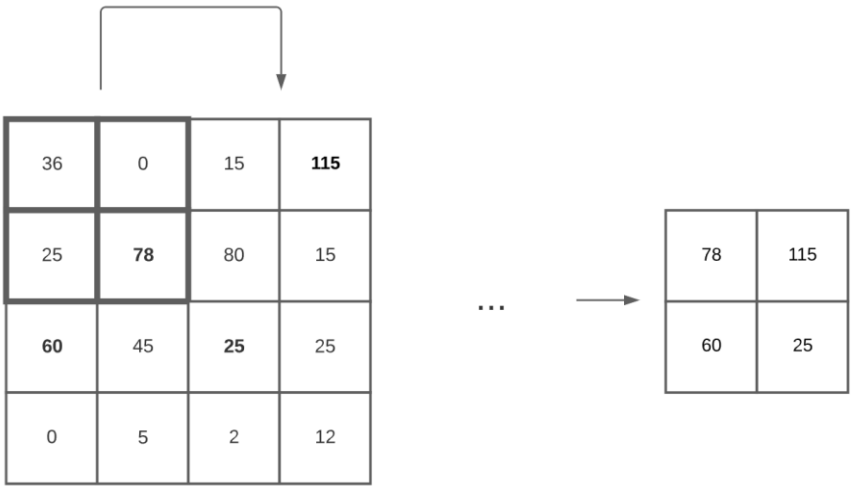


Figura 32. Esquema del proceso de agrupación

Una vez obtenidas las imágenes resultantes, volveríamos a pasarlas por más capas de convolución y agrupación usando normalmente más núcleos al haber reducido el tamaño de nuestra imagen. Una forma de verlo gráficamente es la siguiente:

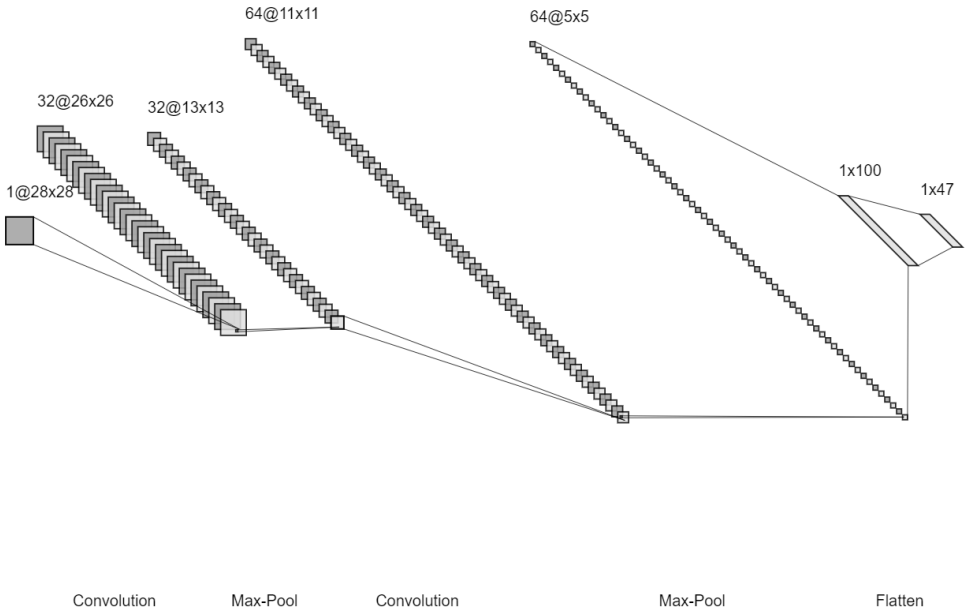
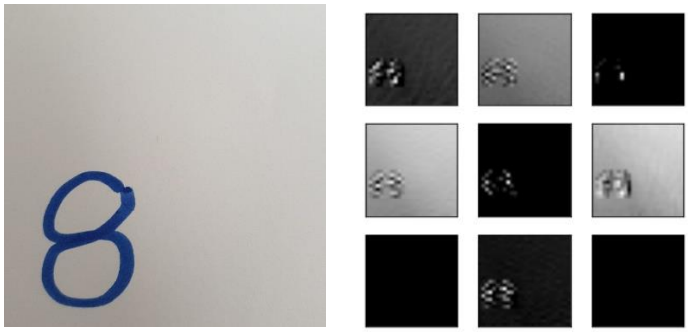


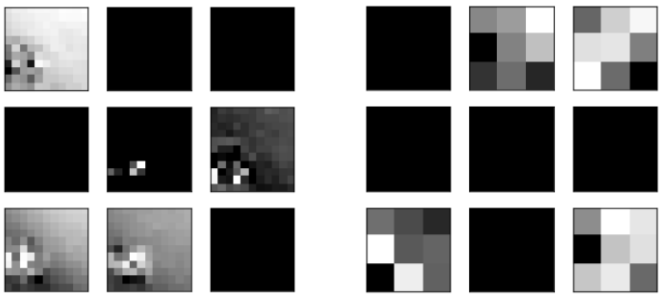
Figura 33. Esquema LeNet de la CNN que utilizaremos

Tras colocar varias de estas capas habremos obtenido las características de entradas necesarias para nuestra red neuronal y podremos pasar al entrenamiento. Antes de continuar, veamos como nuestra red ha extraído las "características de la imagen" para su clasificación [40].



Entrada

Primera convolución



Segunda convolución

Tercera convolución

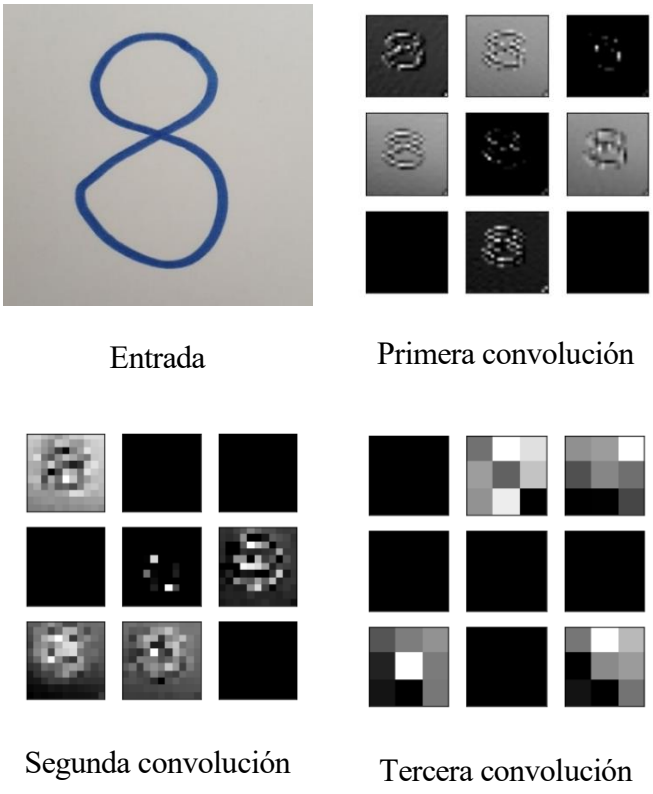


Figura 34. Resultados de la convolución de dos imágenes del carácter 8 pasadas por tres procesos de convolución y agrupación. La última agrupación actuará como las características de cada imagen, teniendo caracteres iguales matrices muy parecidas

5.1 Modelos utilizados

Utilizaremos varias redes neuronales para probar su eficacia y compararlas entre sí, en cada una de ellas mostraremos los cambios que se añaden, así como pruebas de predicción para los datos de la prueba. Distinguiremos cuatro modelos:

1. Red Convolucional simple, únicamente añadiendo las capas de convolución y agrupación
2. Red Convolucional con Dropout
3. Red con arquitectura ResNet
4. Dos redes diferentes para letras y para números

5.1.1 Red Convolucional.

Nuestra red convolucional simple estará compuesta por un par de capas de convolución y agrupación, colocadas en serie, luego una capa oculta de 100 neuronas y finalmente una capa de salida con 47 neuronas, una para carácter que ha de ser reconocido. Las capas ocultas tienen la función de activación ReLu ya vista en apartados anteriores y la de salida la función Softmax. Para este primer modelo realizaremos pruebas con los optimizadores y parámetros de entrenamiento para comprobar la eficacia de cada uno.

```
Vas a usar el MODELO 2: Red Neuronal Convolucional
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 32)        320

 max_pooling2d (MaxPooling2D  (None, 13, 13, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 11, 11, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 5, 5, 64)         0
 2D)

 flatten (Flatten)           (None, 1600)              0

 dense (Dense)               (None, 100)               160100

 dense_1 (Dense)             (None, 47)                4747

=================================================================
Total params: 183,663
Trainable params: 183,663
Non-trainable params: 0
_________________________________________________________________
Numero de capas de la Red Neuronal
7
```

Figura 35. Resumen de la red diseñada.

Necesitamos definir para cada uno de los entrenamientos un tamaño de lote (o BS por sus siglas en inglés), el número de épocas y el número de muestras por lote (steps per epoch).

El tamaño de lote es el número de muestras que se utilizarán para entrenar a la red, calcular el error cuadrático medio y variar los pesos de las neuronas. Esto significa que un número bajo de nuestro BS permite un ajuste más preciso y un menor coste computacional ya que nuestra memoria únicamente se carga con unas pocas muestras para el entrenamiento. Como desventaja obtenemos un comportamiento más ruidoso y el riesgo de estancarnos en un mínimo local utilizando el teorema del Descenso del Gradiente. Por el contrario, un valor más elevado resultaría en un entrenamiento algo más lento y un mayor coste computacional, pero obteniendo mayor confianza en los resultados [41].

El número de muestras por lote a su vez está relacionado tradicionalmente con el BS mediante la siguiente ecuación, ya que permite en cada época utilizar todos los datos disponibles para el entrenamiento.

$$Steps\ per\ epoch = \frac{Longitud\ de\ los\ datos\ de\ entrenamiento}{Batch\ Size} \qquad (7)$$

El número de épocas lo ajustaremos en base a los resultados obtenidos, ya que implica tiempo de computación y puede no ir acompañado de una mejora notoria de la precisión. Utilizaremos en primer lugar unas 50 épocas.

El tipo de función de pérdida que representaremos es la llamada *Categorical Cross-Entropy*, que se utiliza en los problemas de clasificación con más de dos clases. La salida de la función se calcula multiplicando la salida esperada por el logaritmo de la producida y sumando los resultados de cada una de las clases.

$$L = -\sum_{j=1}^{M} y_j * \log(\hat{y}_j) \qquad (8)$$

Siendo M el número de categoría a clasificar y j las clases de nuestro programa.

Si tenemos un clasificador de robots y la red nos arroja los siguientes resultados para una imagen:

$$\hat{y} = \begin{bmatrix} 0.8 \\ 0.12 \\ 0.08 \end{bmatrix} \qquad (9)$$

Siendo la salida esperada la siguiente:

$$y = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \qquad (10)$$

La pérdida para esa imagen en concreto sería

$$L = -(1 * \log(0.8) + 0 * \log(0.12) + 0 * \log(0.08)) = 0.32 \qquad (11)$$

Para calcular este valor para la red neuronal es necesario calcular esta pérdida para cada una de las imágenes

$$L = -\frac{1}{N} * \sum_{i=1}^{N} \sum_{j=1}^{M} y_{ij} * \log(\widehat{y_{ij}}) \qquad (12)$$

Siendo N el número total de muestras de nuestro set.

```
H = modelo.fit(
    train_x, train_y, batch_size=BS, validation_data=(test_x, test_y), steps_per_epoch=len(train_x) // BS, epochs=EPOCHS,
    verbose=1)
print('LISTO')
```

Figura 36. Implementación del entrenamiento utilizando funciones de Keras [42]

Los datos de entrenamiento se dividieron previamente y de forma aleatoria en entrenamiento y validación, para evaluar el comportamiento del modelo a lo largo de la ejecución. Una vez entrenada nuestra red, mostramos las gráficas de precisión y de pérdida:

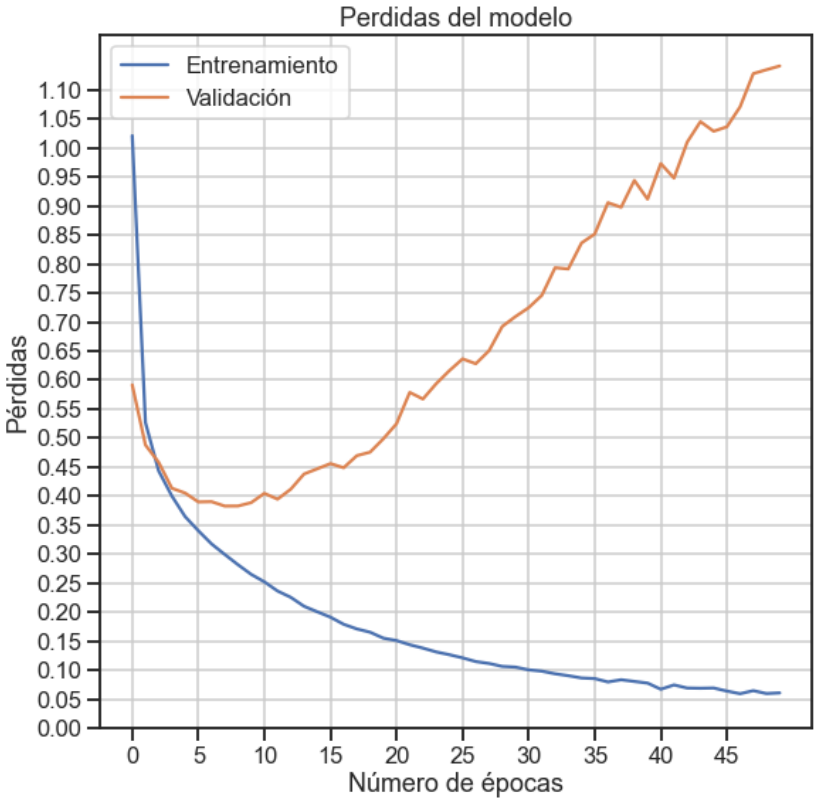


Figura 39. Gráfica de pérdidas para el primer entrenamiento de la CNN

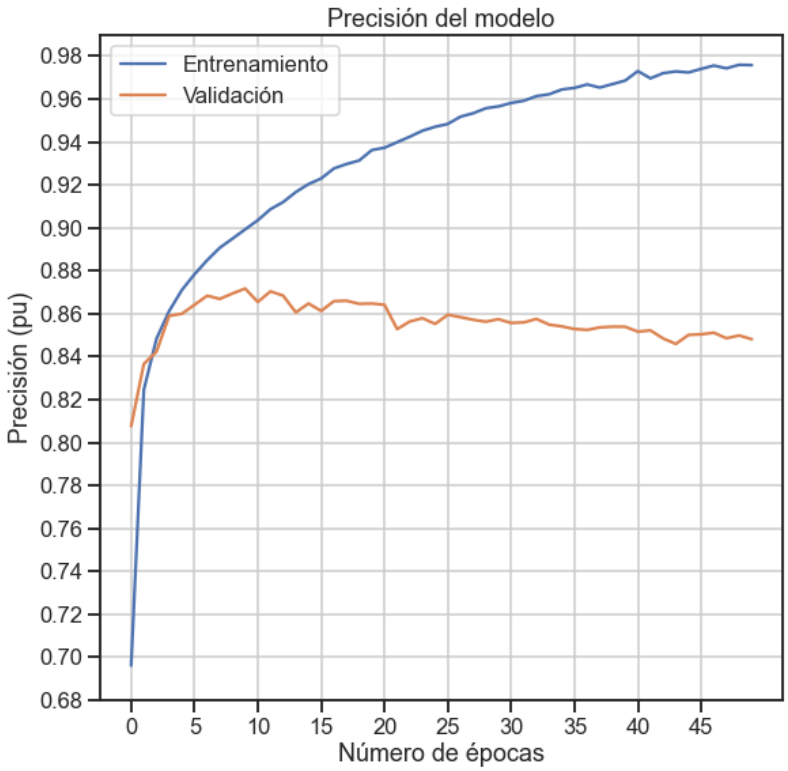


Figura 40. Gráfica de precisión para el primer entrenamiento de la CNN

Durante el entrenamiento observamos un comportamiento dispar entre el entrenamiento y la validación. Mientras el primero va mejorando su precisión y minimizando las pérdidas a lo largo de las iteraciones, los resultados de la prueba actúan a la inversa. Esto se debe al llamado **sobreajuste**, que se da cuando tenemos una red muy grande y se crea un sistema que tiene en cuenta muchos detalles para generar una predicción, pudiendo incluir entre estos el ruido. De esta manera nuestra red consigue "memorizar" nuestro set de entrenamiento, pero las pérdidas de validación van aumentando cada vez más, lo que hace que nuestras predicciones en un entorno de trabajo real no sean exactas y no se pueda dar por válido ese modelo.

Es necesario que para mejorar el comportamiento del modelo realicemos un aumento de datos con el objetivo de modificar las imágenes estirándolas, girándolas y cambiando su tamaño. De esta forma cuando realicemos pruebas con imágenes reales conseguiremos una mayor precisión y los resultados de entrenamiento serán más robustos y parecidos a los que obtendremos en situaciones reales [43]. Usamos la herramienta de Keras *ImageDataGenerator* con valores que modifiquen la forma de los caracteres sin volverlos indistinguibles.

```
# Realizamos el aumento de imagenes de prueba mediante distorsiones, desplazamientos y giros
aug = ImageDataGenerator(rotation_range=20, zoom_range=0.05, width_shift_range=0.1,
                         height_shift_range=0.1, shear_range=0.15, horizontal_flip=False, fill_mode="nearest")
```

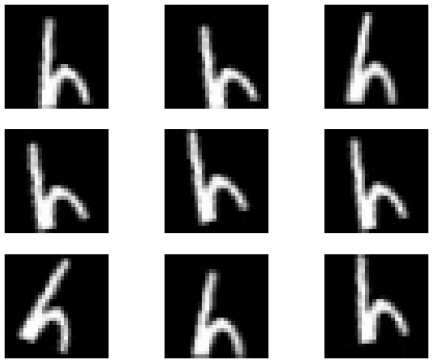


Figura 37. Programación y resultado de aplicar ImageDataGenerator sobre imagen de letra "h"

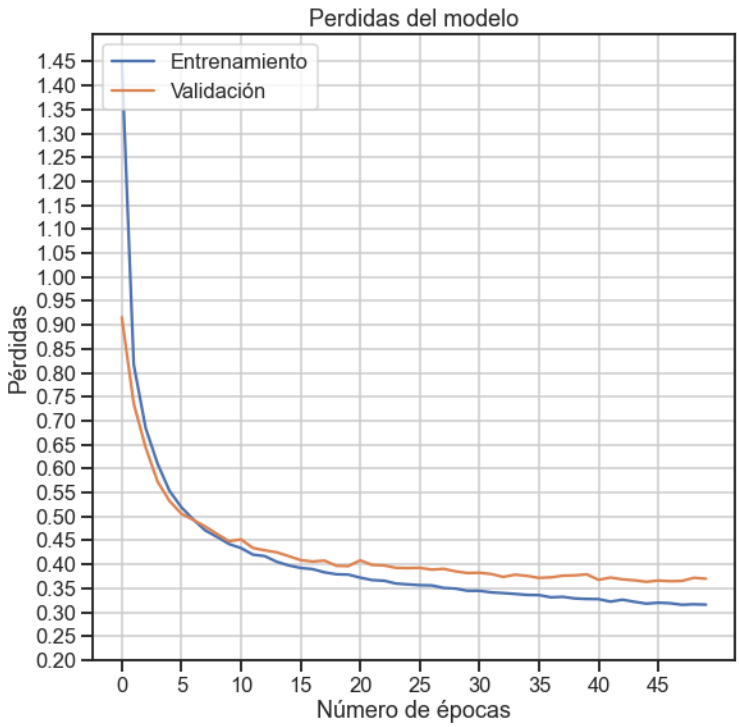


Figura 42. Gráfica de pérdidas con aumento de datos

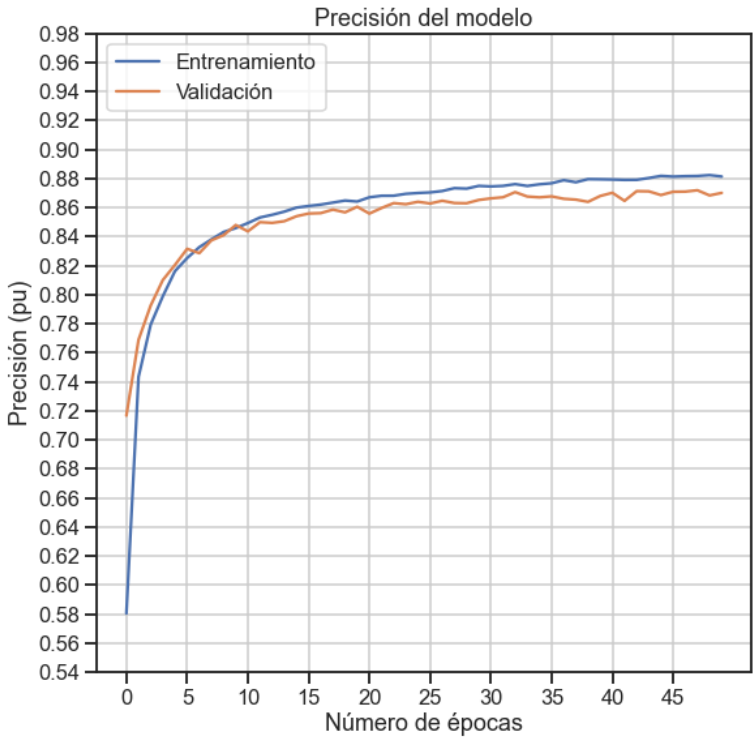
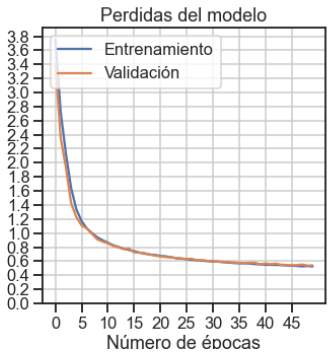
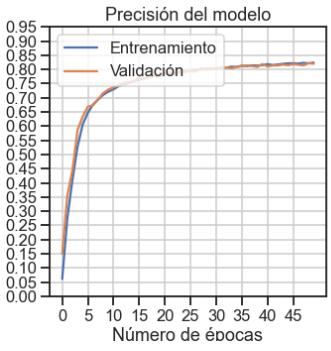


Figura 43. Gráfica de precisión con aumento de datos

Podemos observar que, aunque sigue existiendo cierto ruido en los resultados, estos se estabilizan en torno a un valor fijo y se evita el aumento de las pérdidas.

Realizaremos una comparativa de esta configuración con diferentes optimizadores para ver cómo se comporta nuestro sistema:

Optimizador	Pérdidas	Precisión
SGD		

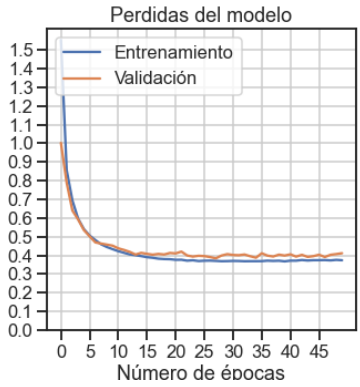
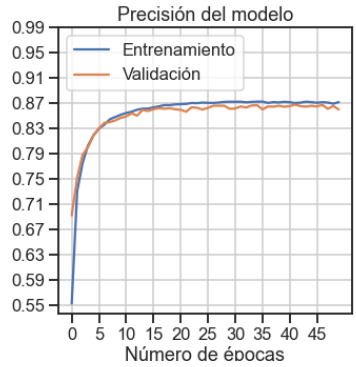
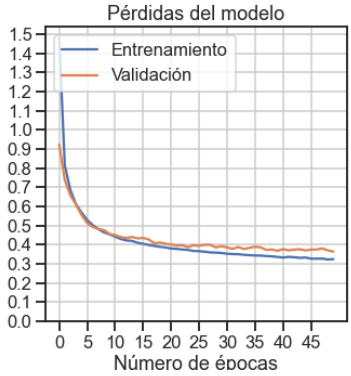
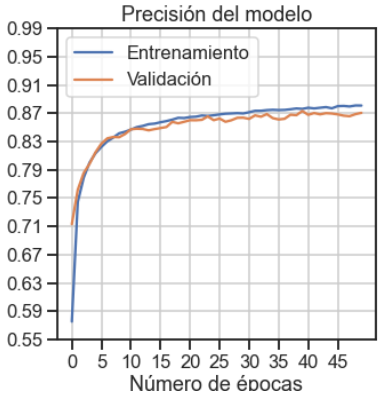
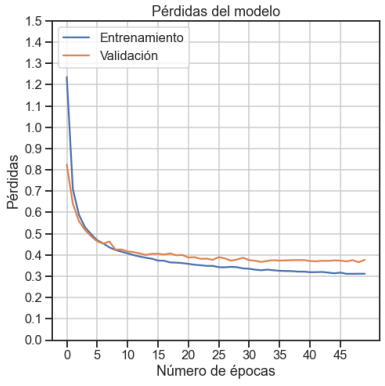
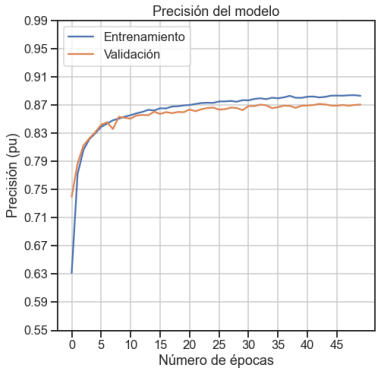
RMSProp		
Adam		

Tabla 5. Comparativa de resultados con distintos optimizadores

En todos los casos la simulación tardó lo mismo y se pudieron sacar las siguientes conclusiones:

- El comportamiento a nivel de ruido es mayor utilizando RMSProp que en otros optimizadores
- Los optimizadores RMSProp y Adam consiguen una precisión similar y superior a la obtenida con SGD
- El optimizador Adam muestra el mejor comportamiento tanto a nivel de ruido como de niveles de precisión y pérdidas, así que será con el que trabajaremos a partir de ahora

A continuación, modificaremos el tamaño del Batch Size y observaremos los resultados:

Batch Size	Pérdidas	Precisión
50		

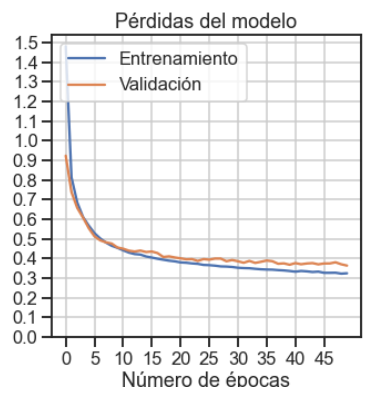
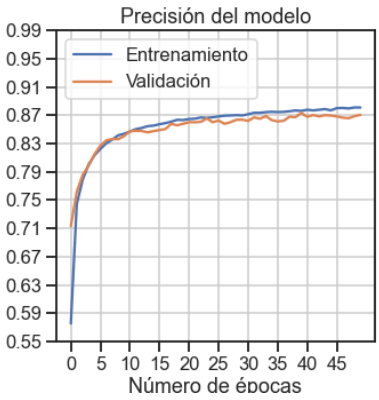
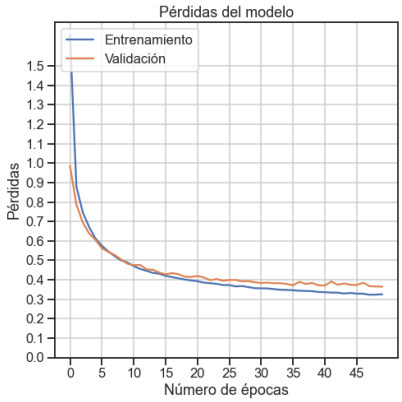
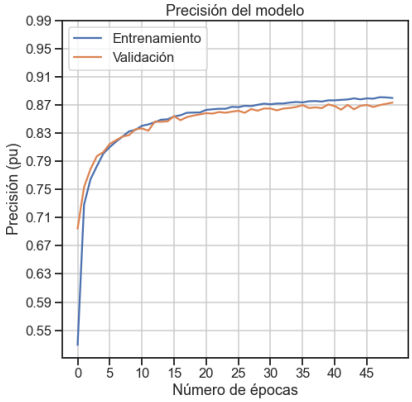
128		
250		

Tabla 6. Comparativa de resultados con distintos tamaños de lote

En este caso las simulaciones seguían tardando el mismo tiempo, ya que el modelo es lo suficiente simple para que el ordenador no utilice por completo la memoria y el procesador tenga tiempo de actualizar los datos. Aun así, podemos darnos cuenta de que se cumple lo que comentamos anteriormente, y los resultados con menor Batch Size son más ruidosos, aunque lleguen con menos épocas al valor de precisión deseado. Tomaremos el tamaño de lote 128 para las siguientes configuraciones y veamos a continuación los resultados de predicción sobre el set de pruebas.

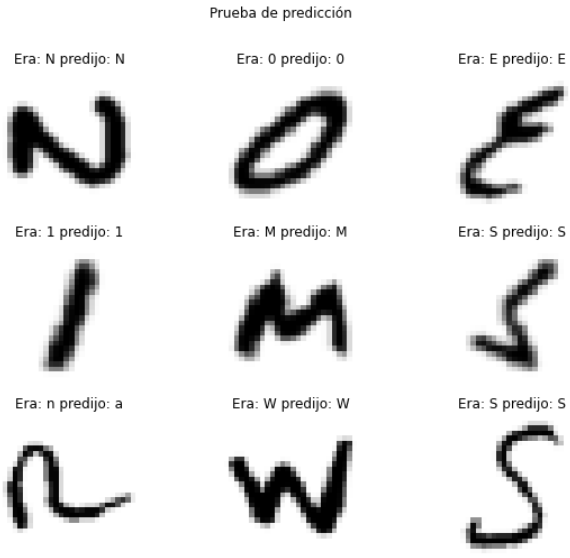


Figura 38. Predicciones obtenidas para el primer modelo

La red es capaz de reconocer la mayoría de los caracteres correctamente y con una precisión bastante elevada. En otras ocasiones se equivoca debido a la similitud con otros caracteres, aunque algo entendible debido al parecido entre algunos caracteres como la G y el 6, la I y la L…

Tomar una red convolucional con más capas ocultas podría ayudar a mejorar la precisión de nuestro modelo. En este caso obtenemos los resultados siguientes para una red con un mayor número de capas.

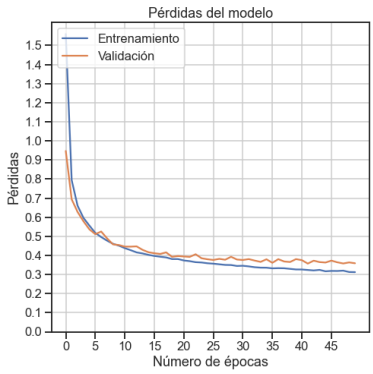
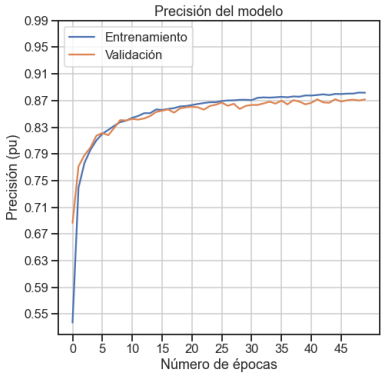
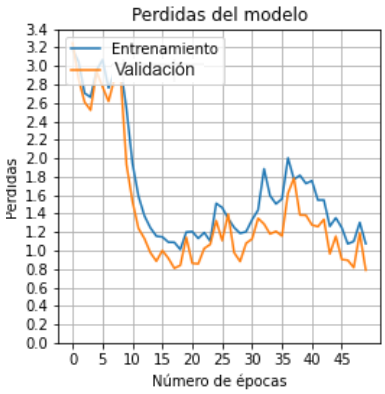
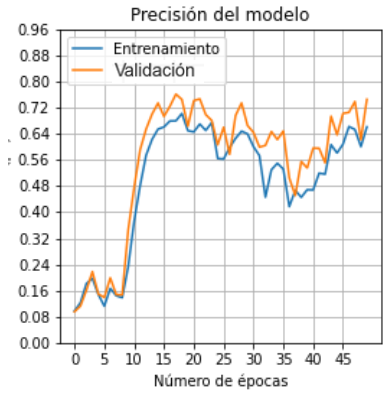
Número de capas	Pérdidas	Precisión
10		
30		

Tabla 7. Comparativa con redes de diferente profundidad

Como podemos observar, una red neuronal más compleja no siempre implicará una mejora de los resultados y será muy importante comprobarlos gráficamente para quedarnos con los mejores modelos y evitar el overfitting.

5.1.2 Red Convolucional con Dropout

Para la siguiente red lo que haremos será añadir una regularización mediante Dropout. El Dropout pretende mitigar el efecto de sobreajuste, "apagando" de forma aleatoria algunas neuronas de nuestra red durante el entrenamiento de forma que los valores de los pesos se distribuyan uniformemente y no tengan mayor impacto unas neuronas que otras ya que estaremos trabajando con redes neuronales diferentes, aunque parecidas las unas a las otras [41]. Lo configuraremos con un valor de 0.5, que significa que en cada época cada neurona tendrá un 50% de posibilidades de apagarse. Añadiendo este nuevo elemento a nuestro programa los resultados serán los siguientes:

```
Vas a usar el MODELO 3: Red Neuronal Convolucional con Dropout
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 32)        320

 max_pooling2d (MaxPooling2D  (None, 13, 13, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 11, 11, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 5, 5, 64)         0
 2D)

 dropout (Dropout)           (None, 5, 5, 64)          0

 flatten (Flatten)           (None, 1600)              0

 dense (Dense)               (None, 100)               160100

 dense_1 (Dense)             (None, 47)                4747

=================================================================
Total params: 183,663
Trainable params: 183,663
Non-trainable params: 0
_________________________________________________________________
Numero de capas de la Red Neuronal
8
```

Figura 39. Resumen del modelo con Dropout

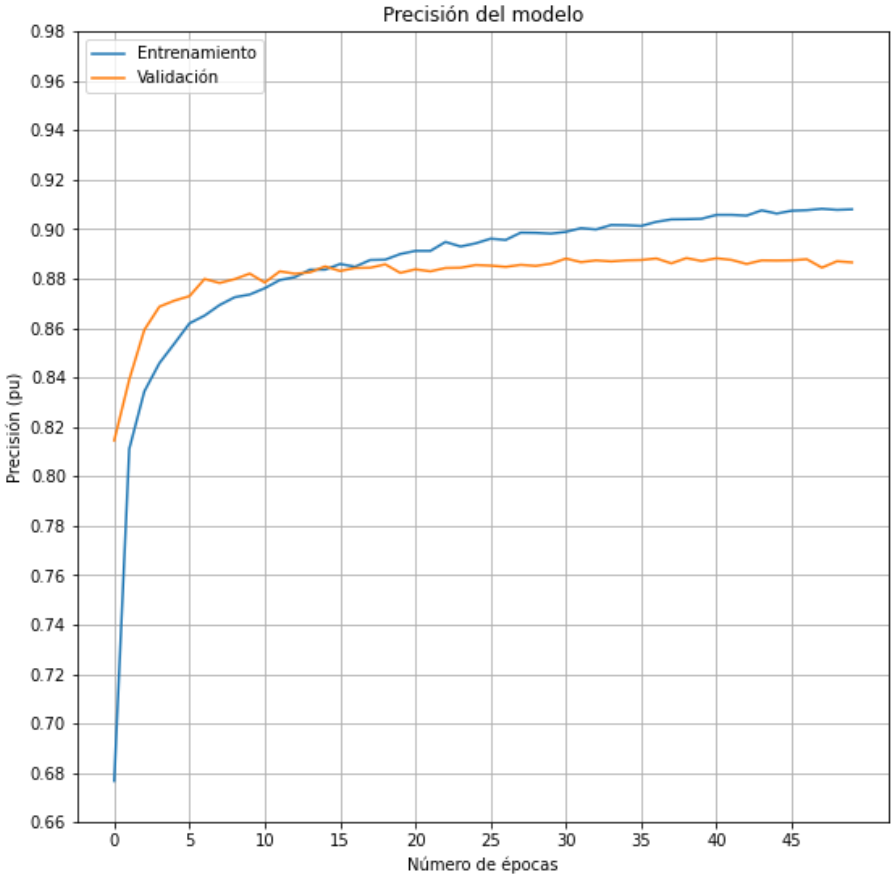


Figura 40. Resultados de pérdidas sin aumento de datos e incluyendo Dropout.

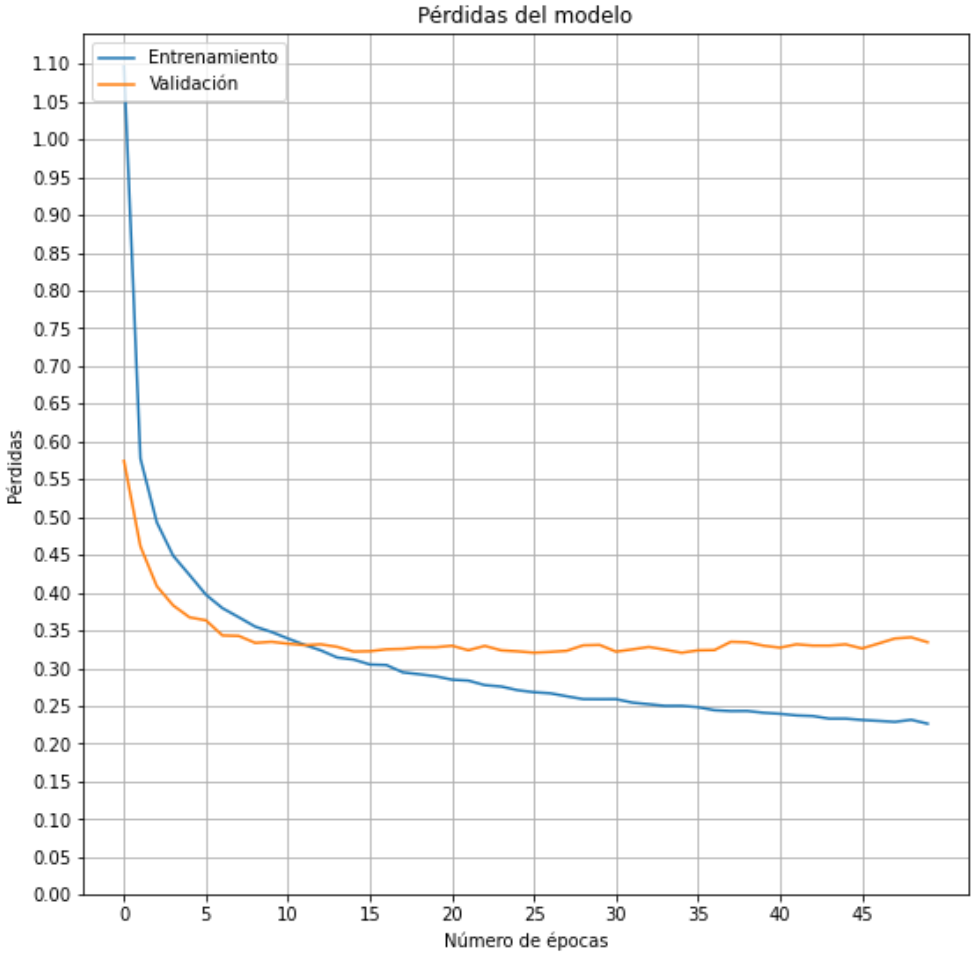


Figura 41. Resultados de precisión sin aumento de datos e incluyendo Dropout.

Como se aprecia en las figuras 49 y 50, vemos que hemos eliminado el efecto de sobreajuste que había en el apartado anterior y los resultados en la validación están en torno al 89% para la precisión y en 0.34 para las pérdidas. Procedamos entonces a incluir el aumento de datos y ver si se observa alguna mejora.

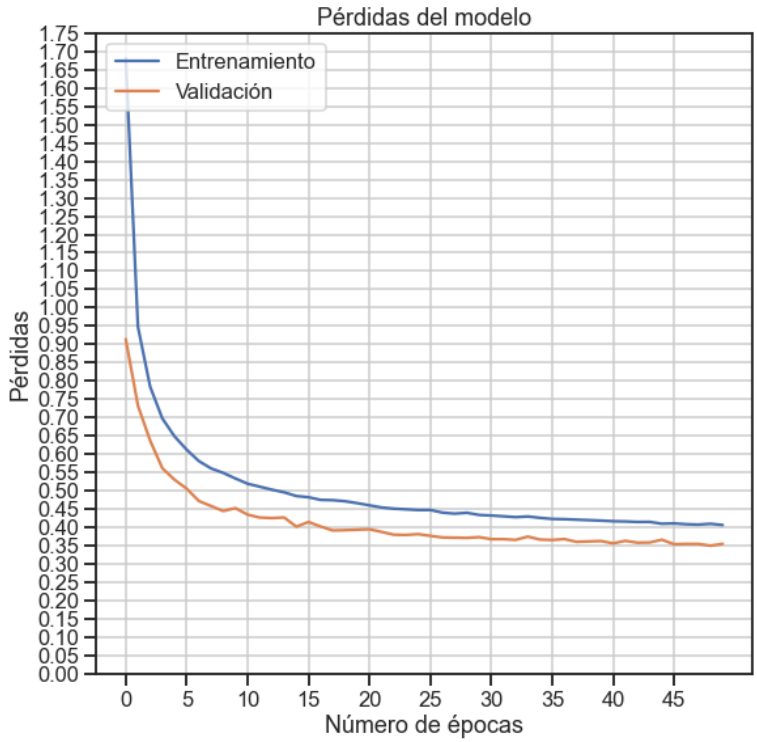


Figura 42. Resultados de pérdidas con aumento de datos e incluyendo Dropout.

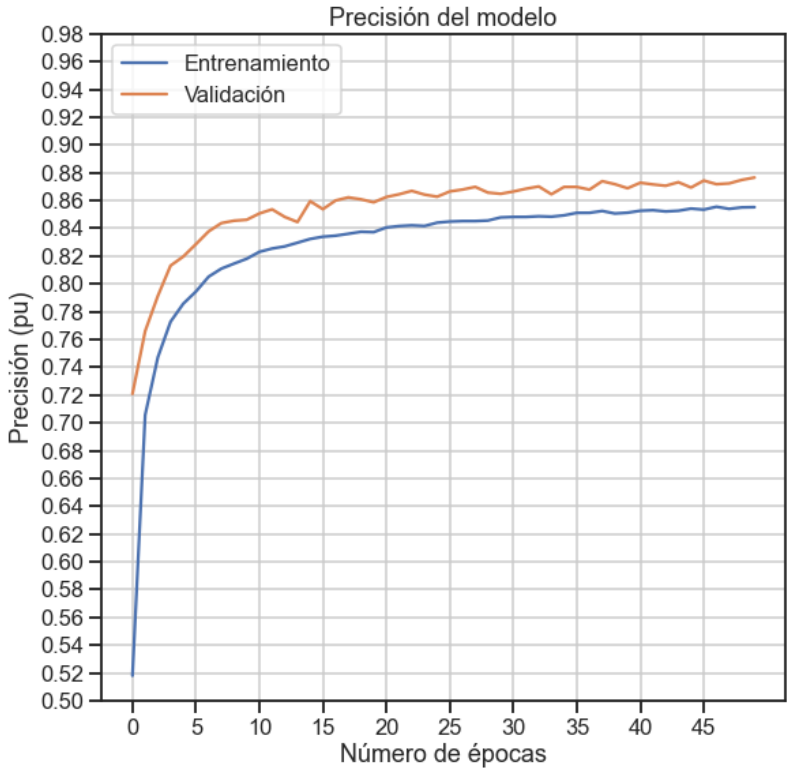


Figura 43. Resultados de precisión con aumento de datos e incluyendo Dropout.

Podemos notar que añadiendo el Dropout los resultados en la prueba mejoran ligeramente (comparar con figuras 42 y 43), pero sobre todo conseguimos reducir el ruido a de las gráficas haciendo este modelo más fiable. Más adelante se realizará una comparativa con caracteres escritos a mano.

Otro de los resultados que podemos observar; con y sin aumento de datos, es que los resultados de validación

son mejores que los de entrenamiento, al menos durante un gran número de épocas. Esto en un principio no debería ser así porque el modelo intenta ajustarse cada vez más a los datos de entrenamiento, pero debido al dropout y a que las neuronas que se activan en cada iteración no son las mismas, al sistema le costará afinar los parámetros en base al entrenamiento a la vez que las pruebas de validación seguirán mejorando a un ritmo más elevado. Una solución para evitar este efecto sería reducir el dropout pero, con el fin de obtener la robustez necesaria para el modelo, se decidió mantener las gráficas de esta forma habiendo justificado su procedencia.

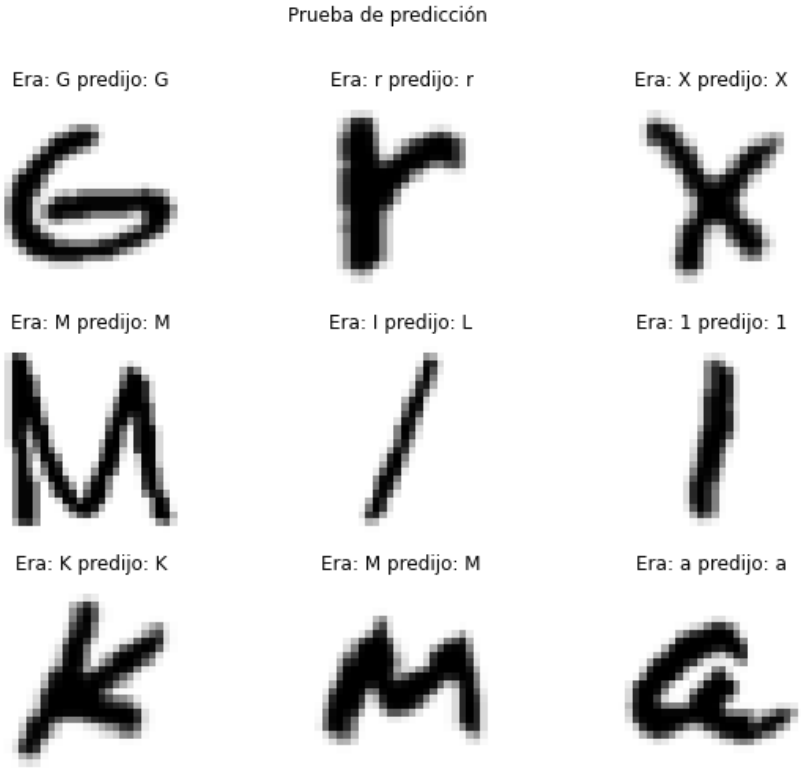


Figura 44. Predicciones obtenidas para el segundo modelo con Dropout

5.1.3 ResNet

En los apartados anteriores llegamos a la conclusión de que incluir más capas ocultas a nuestra red a pesar de hacerla en teoría más robusta y flexible para problemas más complejos pueden introducir el problema del desvanecimiento del gradiente, evitando que se modifiquen los pesos en las capas internas y por lo tanto más profundidad también implique un error mayor.

Sin embargo, en el año 2015, el equipo de Microsoft Research, al mando del científico Kaiming He, presentaron en su artículo “Deep Residual Learning for Image Recognition” el concepto de las redes residuales [44]. Estas introducen una conexión atajo o residual entre capas antes de aplicar una función de activación.

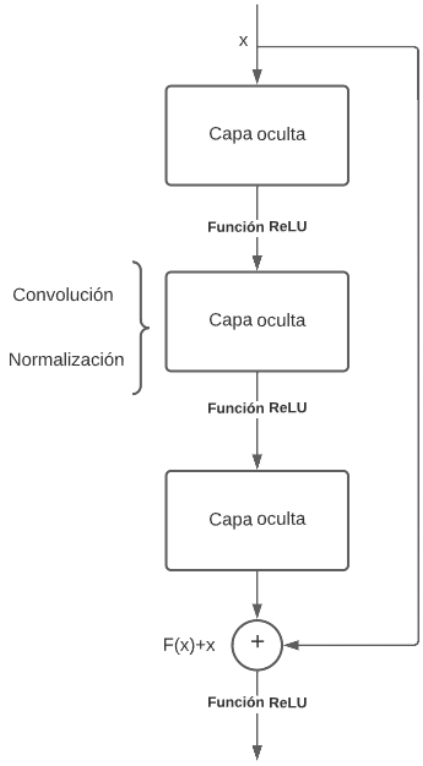
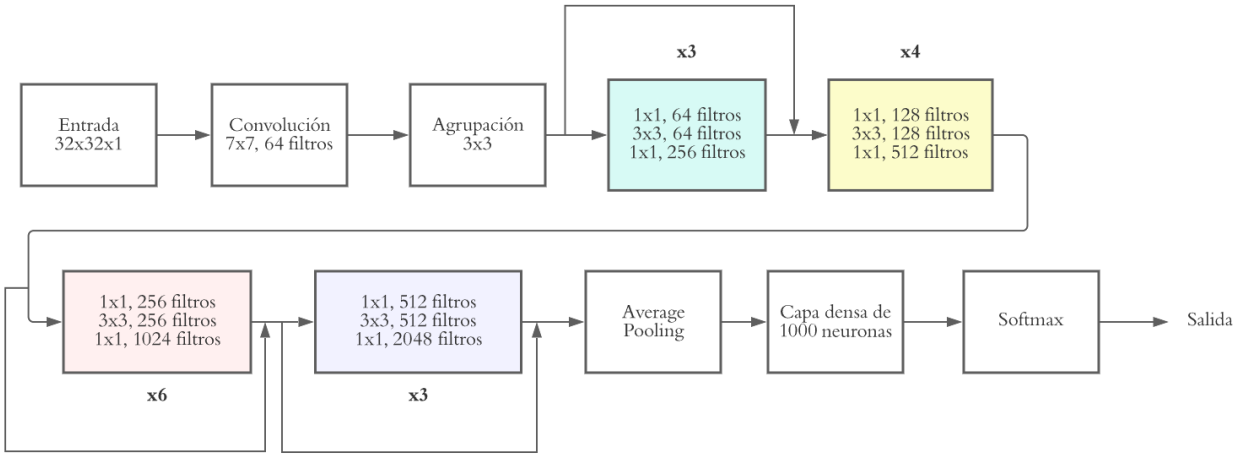


Figura 45. Esquema de una conexión residual.

Al tener esas conexiones lo que ocurrirá es que el gradiente atravesará tanto el camino común como el atajo, mejorando la forma en la que se modifican los pesos de las neuronas en capas más profundas [45]. Gracias a esto podremos tener modelos de cientos incluso miles de capas sin afectar al rendimiento de este y permitiendo operaciones más complejas. Para este caso utilizaremos el modelo ResNet50 que tiene la siguiente estructura, teniendo la conexión residual 3 capas ocultas y teniendo los atajos lugar en las capas de convolución, como se aprecia en la figura 51.




```
==========================================================================================
Total params: 329,971
Trainable params: 325,809
Non-trainable params: 4,162
__________________________________________________________________________________________

102
```

Figura 46. Diagrama y resumen de parámetros de nuestra red ResNet50 [46]

Entrenando al modelo obtenemos los siguientes resultados:

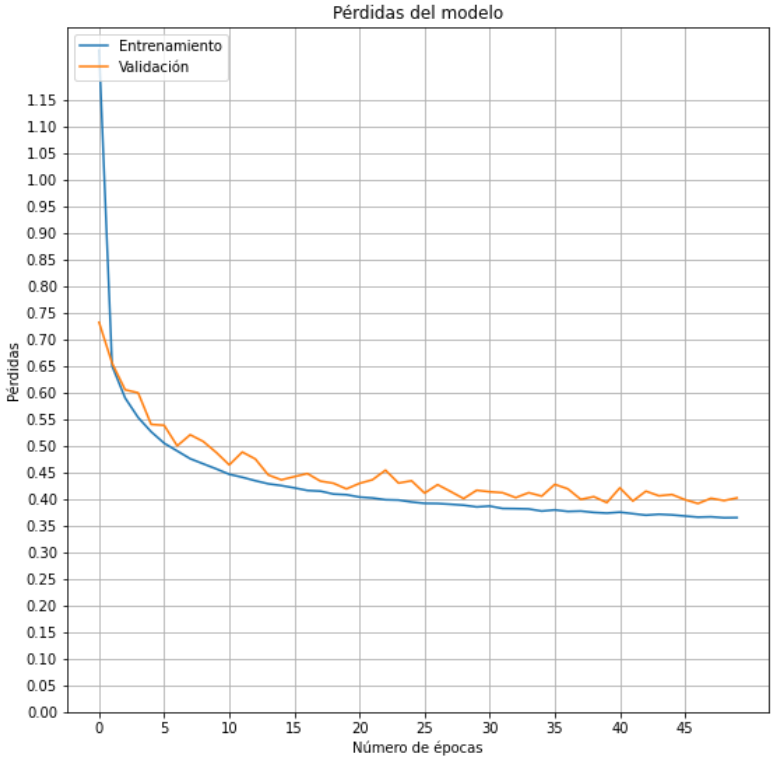


Figura 47. Resultados de pérdidas utilizando ResNet

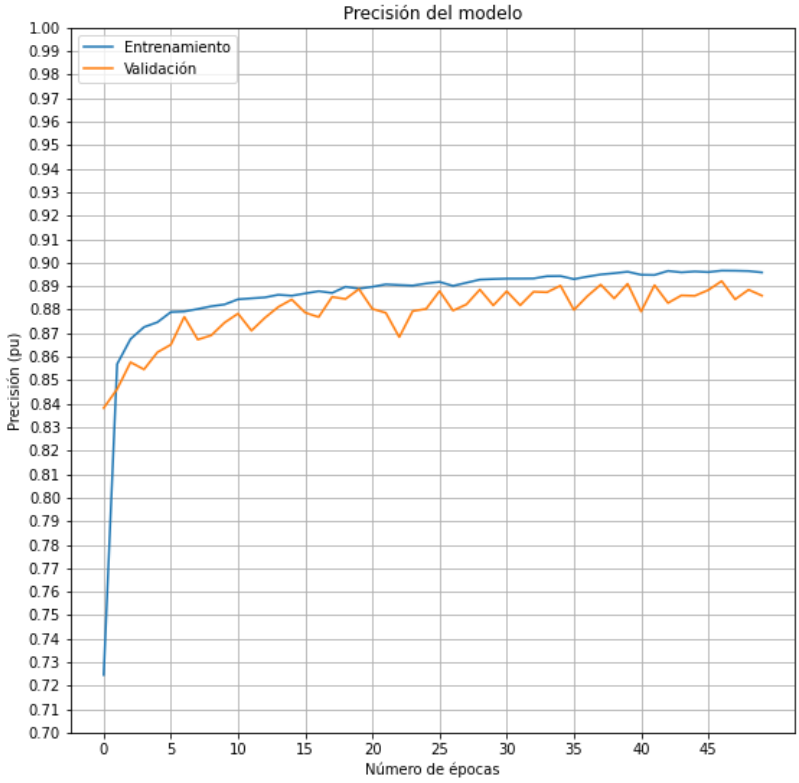


Figura 48. Resultados de precisión utilizando ResNet

Podemos ver que, aunque tengamos gran cantidad de capas su rendimiento sigue siendo muy bueno, aunque los modelos anteriores parecen ser mejores para los datos recibidos

Figura 49. Predicciones obtenidas para el tercer modelo con ResNet

5.1.4 Entrenamiento con dos modelos

Se planteó también un aproximamiento al problema de forma distinta, donde tendríamos dos predictores funcionando al mismo tiempo: uno para las letras y otro para los números, dado que las redes convolucionales obtenían una mayor precisión por separado que las vistas anteriormente. Emplearemos la red neuronal del primer ejemplo, pero duplicada.

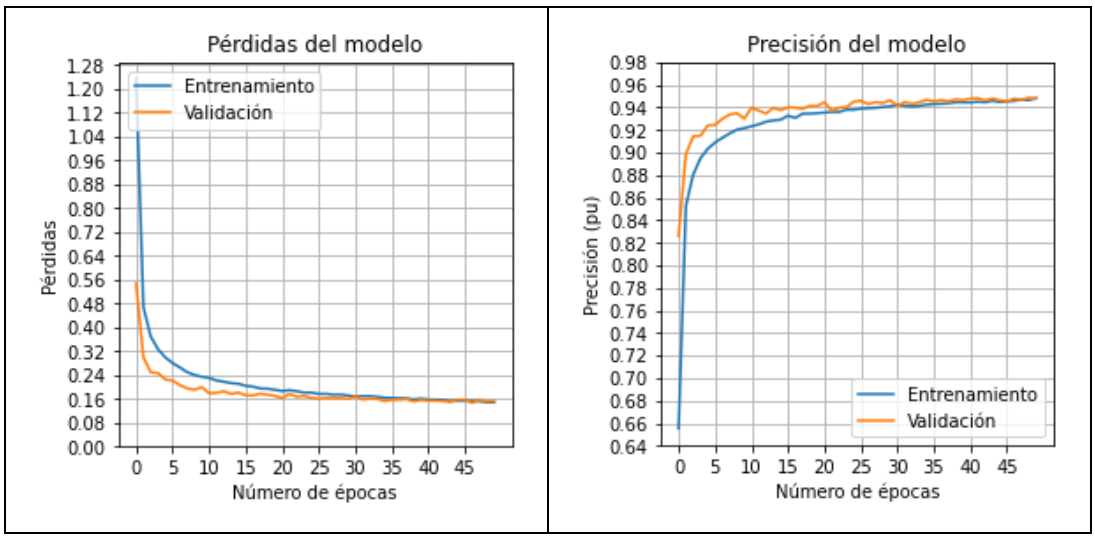


Figura 50. Resultados de pérdidas y precisión únicamente para las letras

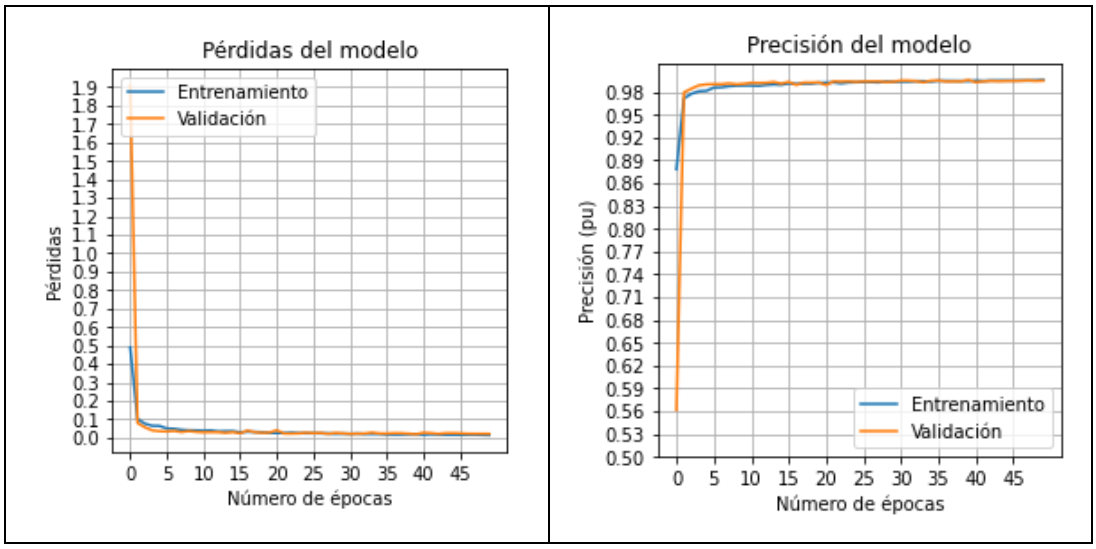


Figura 51. Resultados de pérdidas y precisión únicamente para las letras

En el caso práctico a cada imagen se le aplicarían los dos predictores y aquél que alcanzase un mayor porcentaje de seguridad sería el carácter reconocido.

6 RESULTADOS OBTENIDOS.

6.1 Resultados en imágenes

Para comenzar el análisis primero es necesario ver el desempeño de la red en imágenes con un solo carácter en diferentes posiciones. Para que la imagen sea similar a la de los datos de entrada y con el objetivo de eliminar el ruido en la imagen, se convertirá la imagen a blanco y negro y se aplicará un filtro gaussiano, eliminando el ruido de la imagen. A continuación, binarizaremos la imagen y la reescalaremos a 28x28 píxeles, para que la red neuronal pueda recibirla como entrada [47].

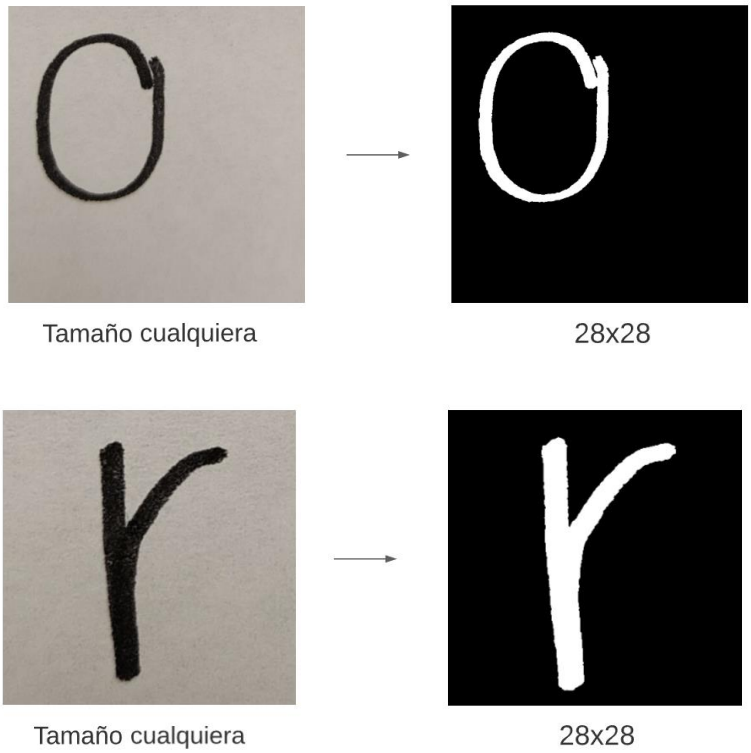


Figura 52. Comparativa del dato de entrada al programa vs dato de entrada a la Red Neuronal

Por último, lanzamos la predicción y la escribimos encima de la imagen original. Se realizarán pruebas para los 4 modelos con diferentes imágenes de caracteres con variaciones en tamaño y forma. Para ellos se utilizará OpenCV, una biblioteca especializada en la visión por computadora [48].

6.1.1 Red Convolucional

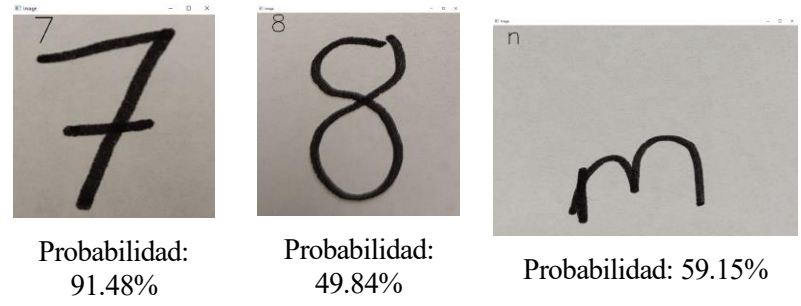


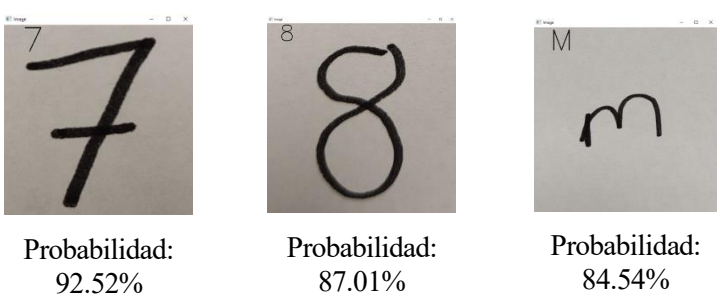
Figura 53. Predicciones utilizando el Modelo 1 (CNN)

Los datos obtenidos en el set de prueba son los siguientes

Caracteres identificados correctamente	16.564
Caracteres identificados incorrectamente	2.235
Porcentaje de acierto	**88.11%**

Tabla 8. Resultados utilizando el Modelo 1 (CNN) en el set de pruebas de *EMNIST Balanced*

6.1.2 Red Convolucional con Dropout



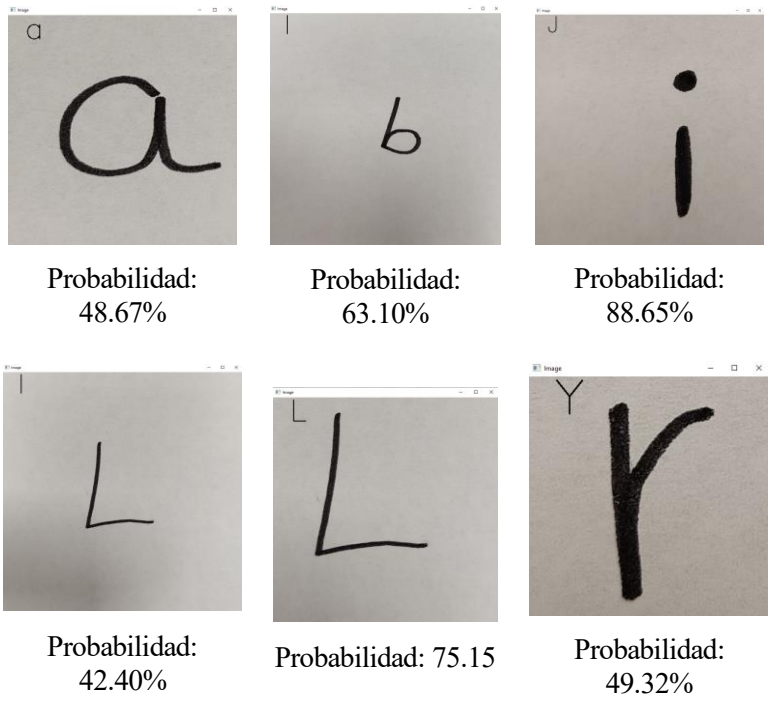


Figura 54. Predicciones utilizando el modelo 2 (CNN+Dropout)

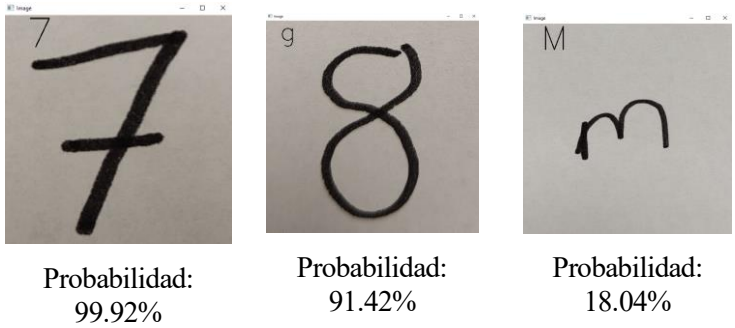
En esta ocasión los resultados obtenidos en el set de prueba son:

Caracteres identificados correctamente	16.583
Caracteres identificados incorrectamente	2.216
Porcentaje de acierto	**88.21%**

Tabla 9. Resultados utilizando el Modelo 2 (CNN+Dropout) en el set de pruebas de *EMNIST Balanced*

6.1.3 ResNet

Para este caso será necesario que las imágenes de entrada sean de 32x32, pues el modelo de ResNet50 empleado debido a las capas de convolución y agrupación por las que está formado



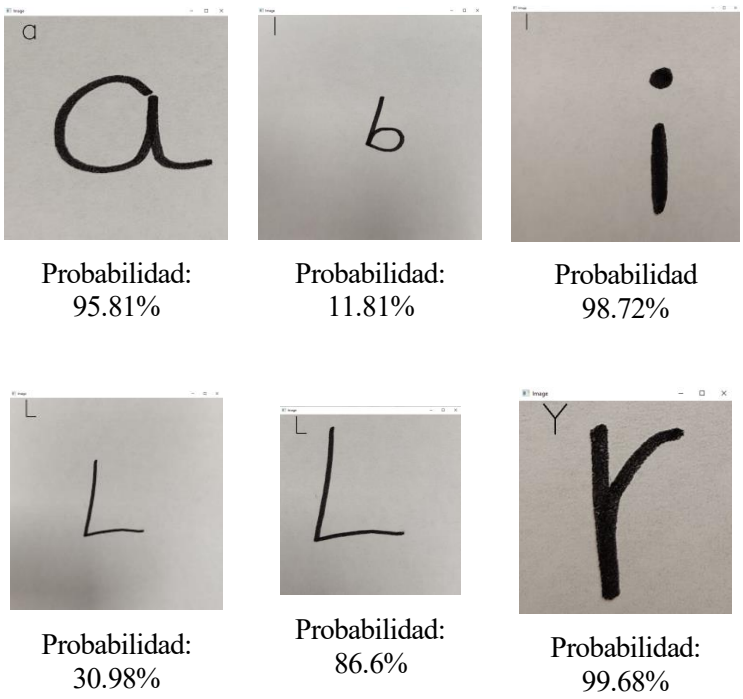


Figura 55. Predicciones utilizando el modelo 3 (ResNet50)

Caracteres identificados correctamente	16.564
Caracteres identificados incorrectamente	2.235
Porcentaje de acierto	**88.11%**

Tabla 10. Resultados utilizando el Modelo 3 (ResNet50) en el set de pruebas de *EMNIST Balanced*

6.1.4 Red con dos modelos

Este modelo requiere de un procedimiento adicional, pues se realizarán dos predicciones (para letras y números) y aquella con una probabilidad más alta se asumirá como la correcta. Tampoco distinguiremos las letras mayúsculas de las minúsculas pues el dataset de las letras contaba únicamente con 26 clases.

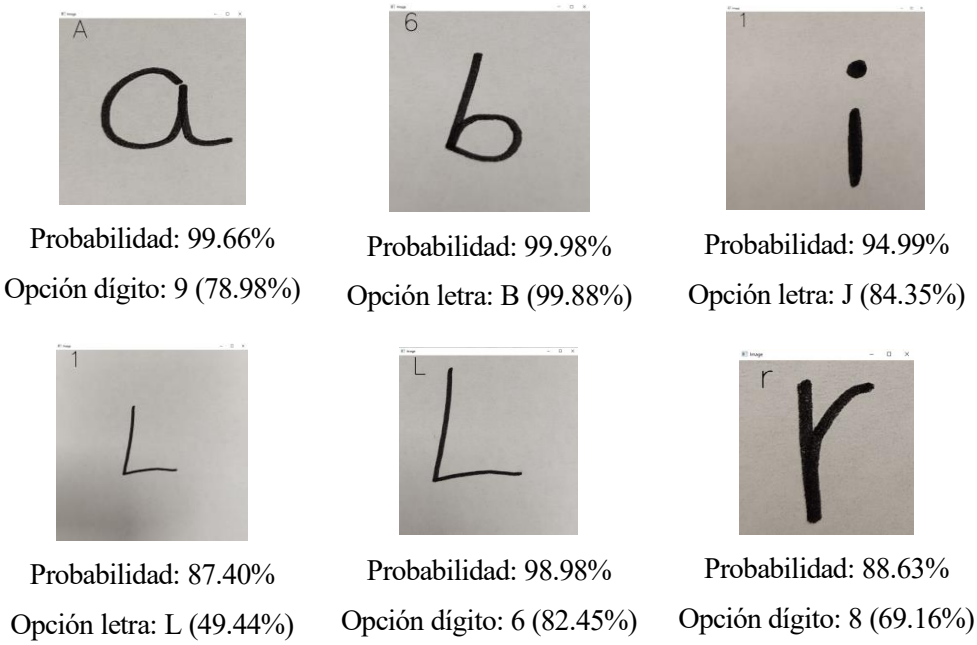


Figura 56. Predicciones utilizando el modelo 4 (Dos modelos)

Caracteres identificados correctamente	23.856 (13.903+9.953)
Caracteres identificados incorrectamente	942 (896+46)
Porcentaje de acierto	**96.20% (93.95%/99.54%)**

Tabla 11. Resultados utilizando el Modelo 4 (CNN Letras + CNN Números) en el set de pruebas de *EMNIST Balanced*

6.1.5 Análisis de los resultados

Tras analizar los resultados vemos que todos los modelos tienen una precisión final menor a la que se obtuvo en el entrenamiento; acertando únicamente 5 ó 6 caracteres de los 9 propuestos. Se han realizado pruebas con más caracteres y los resultados siguen siendo los mismos y únicamente mejoran ligeramente la tasa de acierto. Este menor rendimiento era de esperar pues los datos de entrenamiento y validación tenían un formato muy similar que facilitaba el entrenamiento del sistema. Por lo tanto, tal y como observamos en el entrenamiento los 3 primeros modelos se comportan de forma muy similar. Se aprecia una clara diferencia entre los caracteres que ocupan toda la imagen, donde los modelos tienen menos problemas para realizar una predicción correcta que los que tienen un tamaño menor. Nuestro objetivo por lo tanto será poder extraer de la imagen recibida otra en la que solo esté el carácter y esa reescalarla a 28x28.

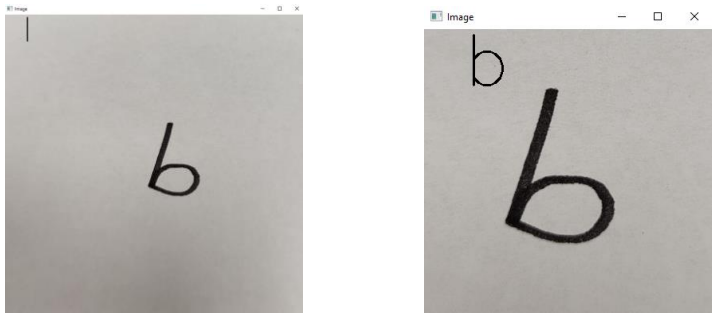


Figura 57. Letra "b" utilizando el modelo ResNet50. En la imagen de la derecha la probabilidad ha sido de un 95.07% frente al 11.81% de la de la izquierda donde la predicción es errónea.

También destaca la precisión del sistema con dos modelos distintos, donde los fallos se han debido a que ha interpretado el carácter como número y no como letra (ya que como letra la deducción era correcta para 3 de los 4 fallos). Se puede por lo tanto suponer que si sabemos el tipo de dato que vamos a recibir (número o letra) nos convendría utilizar el último modelo. Esto se da por ejemplo en la lectura de ciertos documentos identificativos y notas bancarias. Para esos casos podremos asegurar que nuestro sistema funcionará con una gran precisión

6.2 Resolución del problema de detección de múltiples caracteres

Ya hemos conseguido que nuestro programa arroje buenos resultados con la detección de caracteres individuales y cómo mejorarlos, pero en la mayoría de los casos en una misma imagen tendremos no uno, sino varios caracteres, y será tarea del programa segmentar la imagen para la predicción de cada uno de ellos

6.2.1 Programación

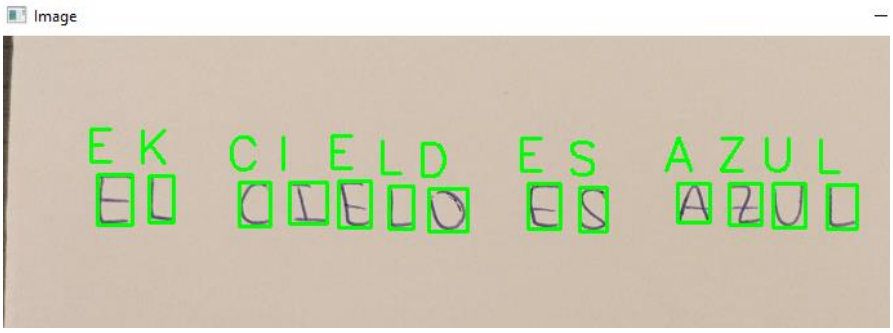
El preprocesamiento de la imagen será en un principio idéntico al de las imágenes de caracteres aislados, aplicando los mismos filtros, pero sin el reescalado a 28x28 píxeles. Lo que haremos ahora será aplicar el algoritmo de Canny para la detección de bordes de la imagen y con el comando de OpenCV *findContours* buscaremos y ordenaremos cada uno de los contornos que son potenciales caracteres [49].

Para cada contorno, aplicaremos el siguiente procedimiento:

1. Comprobar el tamaño de carácter, para evitar que el ruido en la imagen no haya sido detectado como carácter.
2. Si el tamaño es razonable, escalar el ancho o el alto (el más grande) del contorno para comprobar cuanto tenemos que reducir la imagen para que hacerla tan pequeña como la entrada de nuestro sistema.
3. Convertir la imagen a 28x28, manteniendo el carácter centrado.
4. Normalizar y añadir una dimensión a la matriz para que el programa pueda predecir correctamente el contorno. Las dimensiones ahora serían de [28, 28, 1]
5. Añadir esta matriz a la lista de caracteres a predecir, guardando la localización de las cajas delimitadoras (OpenCV nos proporciona este dato).

Una vez hemos enviado los datos de entrada, hay que determinar si el dato se ha detectado correctamente. Para ello se realizaron diversas pruebas y se llegó a la conclusión de que una probabilidad menor al 40% implica una detección incorrecta y que ese contorno realmente no era un carácter. Finalmente se dibuja encima de la imagen los cuadros delimitadores, así como la predicción encima.

6.2.2 Resultados obtenidos




```
[INFO] E - 99.97%
[INFO] K - 56.41%
[INFO] C - 99.99%
[INFO] I - 83.86%
[INFO] E - 100.00%
[INFO] L - 98.55%
[INFO] D - 93.52%
[INFO] E - 99.81%
[INFO] S - 99.73%
[INFO] A - 96.99%
[INFO] Z - 99.15%
[INFO] U - 88.42%
[INFO] L - 99.91%
```

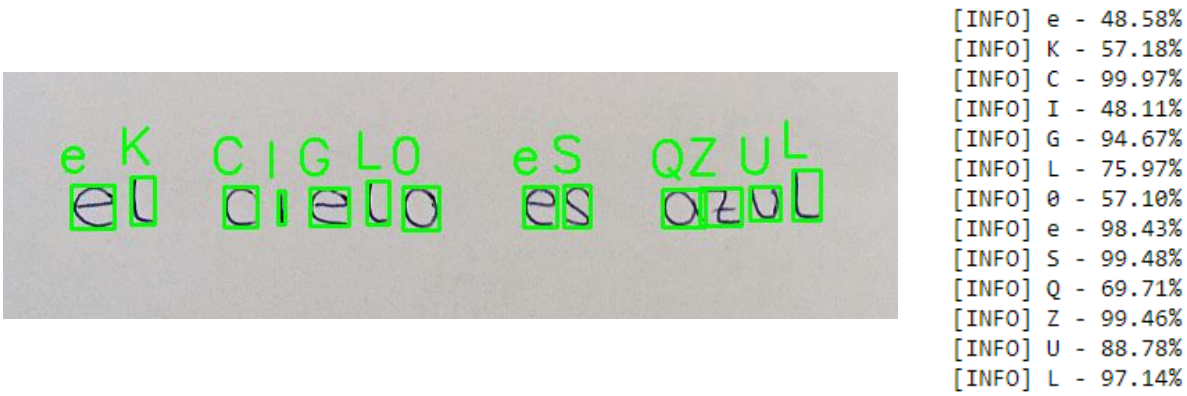


Figura 58. Resultados utilizando CNN para letras mayúsculas y minúsculas, indicando probabilidad de acierto

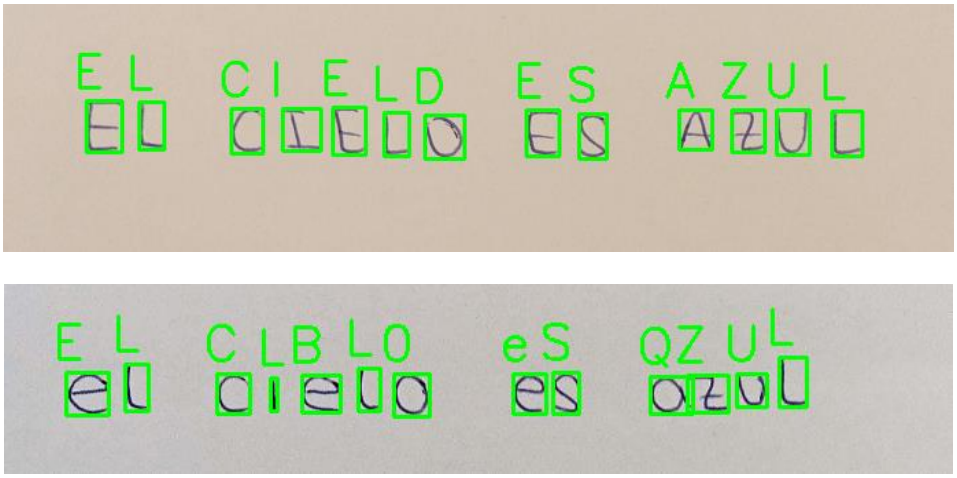


Figura 59. Resultados utilizando CNN+Dropout

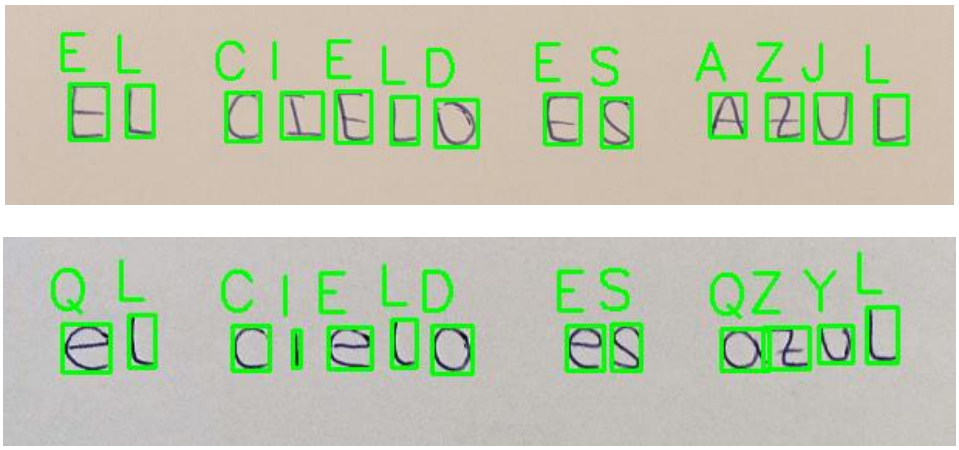


Figura 60. Resultados utilizando ResNet50

Figura 61. Resultados con dos CNN, así como el carácter elegido y su probabilidad en cada caso.

De los resultados obtenidos obtenemos las diferentes conclusiones. La primera de ellas es que, tal y como supusimos en las conclusiones anteriores, el hecho de recortar la entrada y que el carácter ocupe el máximo tamaño posible ha supuesto una mejora significativa en cuanto a la calidad de los resultados, pasando de un 55% de acierto hasta un 78% de media para estos ejemplos, acercándose más a los valores obtenidos en el entrenamiento.

Observamos también que la red CNN+Dropout parece tener un comportamiento similar tanto en mayúsculas como en minúsculas, siendo el más robusto de los 4. Aun así, Los resultados son siempre mejores utilizando mayúsculas. Esto es porque los caracteres en mayúsculas son más reconocibles y tienden menos a confundirse entre sí. Es interesante por lo tanto destacar el paralelismo que existe entre el Deep Learning y la metodología humana, pues en muchos documentos también se pide rellenar los datos en mayúsculas para evitar confusiones entre letras.

Nuevamente destacan los valores arrojados por el último programa. Si se hubiese usado únicamente el predictor de letras habríamos obtenido la mayor precisión de todos los modelos, por lo que la conclusión obtenida tras estos experimentos es que siempre que sea posible trataremos de indicarle a la red si la entrada son caracteres alfabéticos o numéricos.

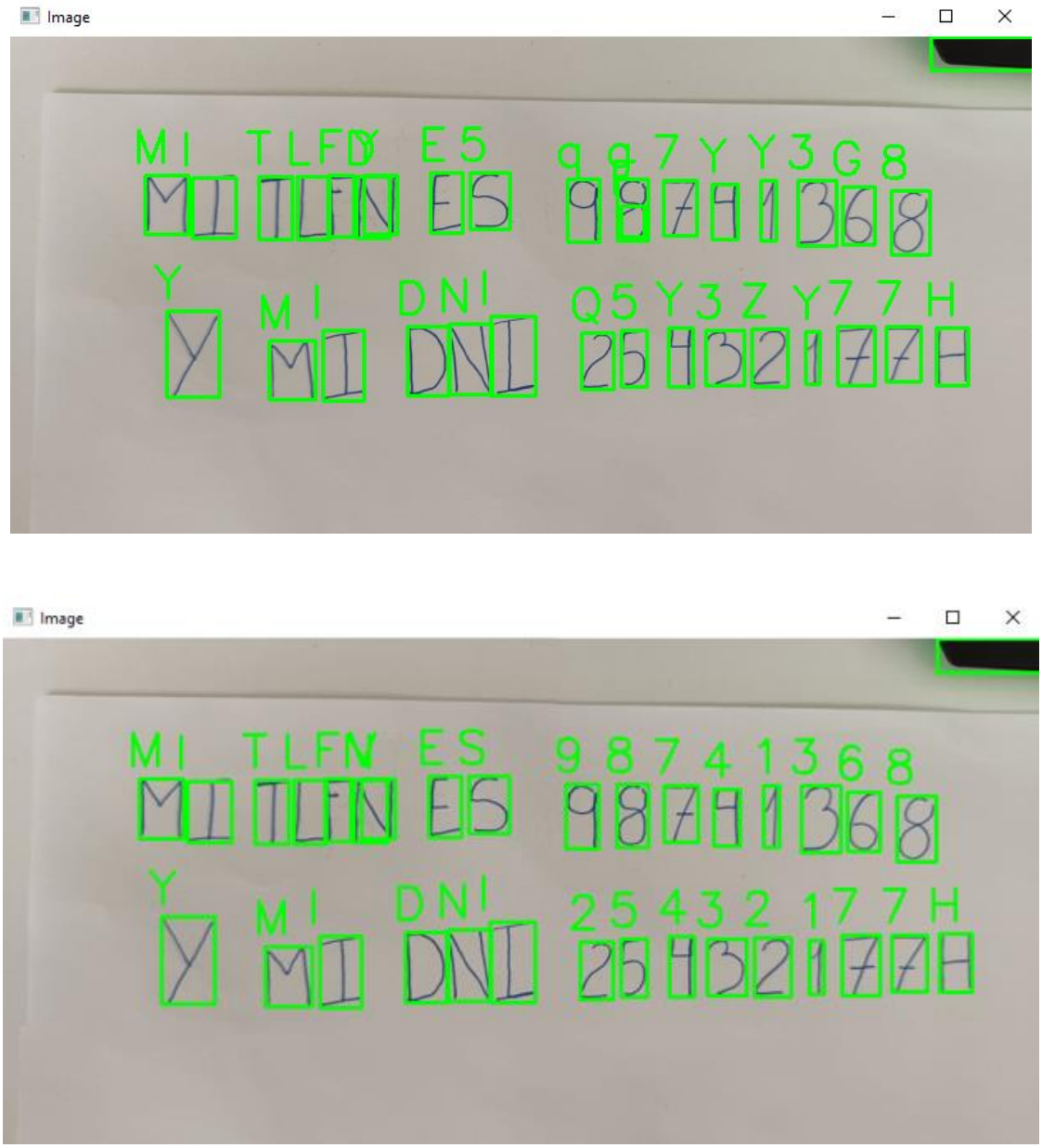


Figura 62. Comparativa entre usar un mismo predictor para letras y números y seleccionar en cada zona de la imagen qué predictor usar

6.3 Estudio e implementación de los Resultados en Tiempo Real

6.3.1 YoloV5

Por último, se ha querido probar y analizar una arquitectura reciente para comparar sus resultados con aquellas ya vistas a lo largo de este proyecto y plantear pruebas en Tiempo Real para futuras líneas. Se ha optado por utilizar YoloV5 (lanzado en mayo de 2020) para este fin.

YoloV5 es la quinta versión del algoritmo YOLO, acrónimo de You Only Look Once (solo miras una vez), que divide la imagen en celdas y de una sola pasada no solo realiza el trabajo de predicción sino además el de detección de objetos, pudiendo tener en una misma imagen más de un objeto y siendo capaz de encontrarlos y clasificarlos. La ventaja que supone este método respecto a los anteriores vistos es la velocidad ya que, según la

documentación proporcionada por su creador, Glenn Jocher, esta herramienta puede trabajar a una frecuencia de hasta 45 imágenes por segundo en una resolución HD, lo que nos da un comportamiento prácticamente en Tiempo Real [51].

A diferencia de las Redes Neuronales Convolucionales (CNN) que analizan toda la imagen de una sola vez para lanzar una predicción, Yolo detecta en primer lugar regiones de interés de la imagen para luego pasar esa región a la red. Se introduce por lo tanto el concepto de R-CNN (Regional CNN) y en ellos se demuestra que el mapa de características de una región de la imagen mantiene la posición en el mapa de características y es idéntico al de la región analizada por separado.

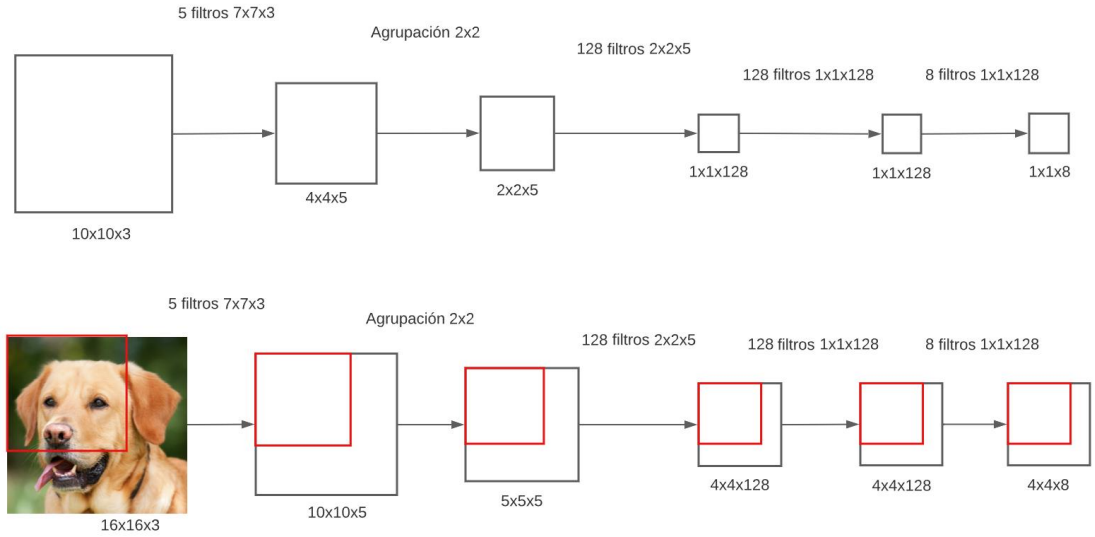


Figura 63. Relación entre el mapa de características de un fragmento de imagen y el fragmento del mapa de características de la imagen completa.

Para cada ventana de nuestra imagen tendremos el siguiente vector asociado:

$$[p_c, c_1, c_2, \dots, x, y, w, h] \qquad (13)$$

p_c: *Probabilidad de que exista un objeto en esa ventana*

c_n: *Clase a la que pertenece el objeto asociado a esa ventana*

x, y: *Coordenadas del centro de la ventana*

w, h: *Ancho y alto relativo de la ventana respecto a la imagen completa*

La imagen dividida en celdas junto con el vector para cada una será la entrada en nuestra CNN para el entrenamiento. Podemos encontrarnos a la hora de predecir una imagen que el mismo objeto se detecte en múltiples ocasiones generando una gran cantidad de cajas delimitadoras. Para evitarlo la red realiza el siguiente procedimiento:

1. Mira el vector salida de cada ventana.
2. Elimina todas aquellas predicciones que cuenten con un p_c menor a cierto umbral, por ejemplo 0,6.
3. Selecciona aquella con el p_c máximo para cada clase.
4. Elimina las regiones que se solapen en exceso con la seleccionada.

Habiendo explicado el procedimiento para la clasificación de objetos, a la hora de implementar el programa para el problema de detección de caracteres nos encontramos con un gran problema, y es que el dataset que estamos utilizando se encuentra en formato .csv y el programa de entrenamiento de YoloV5 para el cálculo de los pesos de la red exige que las imágenes se encuentren en formato .jpg y etiquetadas con su clase y posición y tamaño de los objetos. Para solventar este problema se decidió crear un programa auxiliar que leyese cada imagen, la

guardase en el ordenador en formato jpg y crease una única etiqueta con centro en el centro de la imagen y que abarcase el cien por cien de esta.

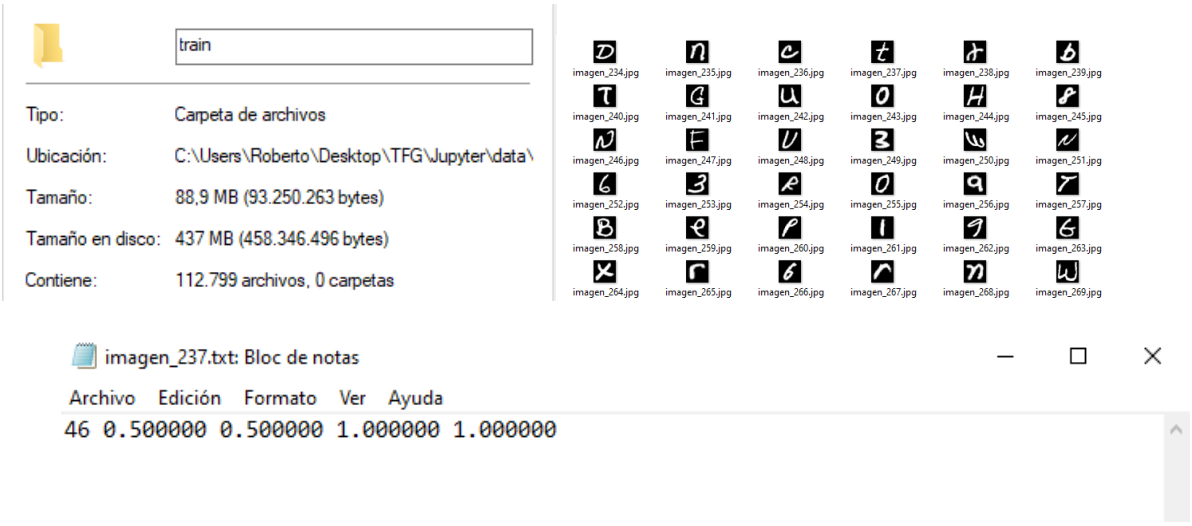


Figura 64. Nueva organización de los datos de entrenamiento y validación; con imágenes y etiquetas

Para este experimento al tener una gran cantidad de datos de entrenamiento en forma de imágenes transformábamos los programas anteriores que se podían entrenar con cierta facilidad en uno muy costoso computacionalmente y para el cuál el ordenador físico no estaba capacitado. Se entrenó por lo tanto utilizando el sistema virtual de Google Colab [52] y debido a las limitaciones del programa se entrenó únicamente por 10 épocas, obteniendo una precisión del 65% pero que iba en aumento. Los resultados en este caso son muy pobres, pero esto se debe en un primer lugar a la limitación temporal para el uso del programa y en segundo lugar debido al tamaño tan reducido de las imágenes. Se llega a la conclusión por lo tanto de que con un dataset más pequeño, pero con un mayor tamaño de imagen y etiquetado manualmente se conseguirían mejores resultados, quedando pendiente realizar pruebas en imágenes y vídeo.

7 CONCLUSIONES

"Nobody phrases it this way, but I think that artificial intelligence is almost a humanities discipline. It's really an attempt to understand human intelligence and human cognition."

- Sebastian Thrun-

El objetivo de este proyecto ha sido el de realizar una aproximación al Deep Learning, concretamente al que conformado por redes neuronales y comprobar cómo funcionan algunos de los programas de OCR que se utilizan hoy en día como puede ser el de Google. Asimismo, se quería hacer una comparativa entre múltiples y modelos y comparar los resultados obtenidos en el entrenamiento con aquellos que se dan en la realidad.

Tras realizar una investigación en profundidad sobre distintos tipos de Redes Neuronales se han encontrado distintas variaciones de CNNs que dan buenos resultados tanto con datos del Dataset EMNIST como con caracteres de distintas personas, confirmando la fiabilidad de los modelos. Además, las predicciones responden bien en problemas de tiempo real, logrando segmentar y localizar los caracteres.

A modo de resumen, los resultados obtenidos para los diferentes modelos han sido los siguientes:

Modelos	CNN	CNN+Dropout	Resnet	2 Modelos CNN
Entrenamiento	88.56%	88.94%	88.48%	**94.05% / 99.72%**
Dataset de prueba	88.11%	88.21%	88.11%	**96.20% (93.95%/99.54%)**
Caracteres Individuales	**77.78%**	55.56%	66.67%	55.56%
Múltiples caracteres	80.77%	**88.46%**	76.92%	76.92%

Tabla 12. Comparativa entre los diferentes modelos en las fases de desarrollo del proyecto.

Podemos concluir que los resultados Los principales objetivos planteados inicialmente se han cumplido de manera satisfactoria, quedando pendientes ciertas mejoras y avances que podrían plantearse en futuras líneas de investigación sobre esta materia.

7.1 Futuras mejoras

Como ya se mencionó en las pruebas realizadas, ciertas mejoras y avances podrían hacerse siguiendo con la investigación y pruebas de más modelos utilizados en reconocimiento óptico de patrones y formas, ya que las mejoras en este campo muchas veces se realizan en cuestión de meses. Como mejoras a implementar se plantean las siguientes:

- Continuar con las pruebas de los modelos propuestos, cambiando parámetros de entrenamiento para optimizar los valores de precisión y pérdidas, así como conseguir mejores resultados en menos épocas. Se propone, por ejemplo, realizar un bucle en el que se cambien ciertos parámetros como el número de capas ocultas de la red, los optimizadores o el tamaño de lote en el entrenamiento y analizar las gráficas obtenidas hasta quedarnos con la mejor. Es importante que los pesos iniciales en el entrenamiento sean los mismos en cada iteración para evitar la aleatoriedad de los experimentos.
- Se ha trabajado con el conjunto de datos EMNIST por ser el más completo para el reconocimiento de caracteres escritos individuales, pero en muchos casos la forma de escribir un carácter es distinta en cada persona (sobre todo para caracteres como la s, la r o la m). Implementar un dataset propio donde se pudiesen incluir esas variaciones podría mejorar la precisión de los resultados, aunque requeriría un gran trabajo de recopilación y etiquetado previo de muestras.
- Probar a realizar entrenamientos para detectar grupos de caracteres o de palabras, ya que muchas veces un único carácter no aporta suficiente. Se podría intentar implementar los modelos utilizando otros datasets y realizar pruebas para textos mecanografiados, ya que textos manuscritos requirían de una gran cantidad de datos y tiempo de entrenamiento para conseguir unos resultados aceptables. Algunos de los datasets empleados en estos casos son el Coco-Text Dataset y el Total-Text Dataset. Para su entrenamiento se utilizan principalmente Redes Neuronales completamente convolucionales (FCNN) como pueden ser la red EAST o la CRAFT [53]
- Probar otros modelos para la detección de caracteres. Para nuestro proyecto se probaron 4 variantes para realizar la comparativa, pero existen infinitud de modelos que aproximan el problema de OCR de formas distintas. Los Transformers, RAMs y CRNNs son modelos que han demostrado ser muy fiables en la detección de texto manuscrito [53]
- Cambio y mejora de las condiciones óptimas. Estudiar el comportamiento en otras circunstancias en las que no se puedan cumplir las mejores condiciones de luminosidad. Con la finalidad de adaptar el dispositivo al medio en vez de adaptar el medio al dispositivo.

7.2. Cronograma

Como ya se explicó en el apartado de introducción y planificación del proyecto, el trabajo se ha compuesto de cuatro fases principales: recopilación de información y estudio del estado del arte, estudio del problema de detección de caracteres manuscritos e implementación de diferentes modelos, realización de pruebas para cada uno de los modelos en diferentes condiciones y elaboración de la documentación.

El tiempo dedicado al desarrollo del proyecto ha sido de unas 400 horas efectivas de trabajo, unos dos meses y medio.

La primera etapa de búsqueda de documentación y formación en Python e Inteligencia Artificial, pues se partió de conocimientos básicos al empezar el proyecto, abarcó las primeras dos semanas. A lo largo de estas se realizaron cursos acerca del Deep Learning a la vez que analizábamos código de problemas similares, como problemas de clasificación binaria o clasificación de números con una fuente determinada.

La siguiente etapa, en la que realizamos la búsqueda y diseño de modelos compatibles para nuestro problema a la vez que se realizaban pequeñas pruebas acerca del funcionamiento de los modelos, ocupó la mayor parte del tiempo del desarrollo del proyecto, unas 150 horas. Esto se debe a la modificación que se estuvo realizando a

los modelos para lograr mejores resultados, a lo que añadíamos otra fase de búsqueda de información de cómo aplicarlos correctamente. Los tiempos de entrenamiento para cada uno de los modelos fueron de 25 minutos excepto el del YoloV5, que duró 3 horas. Se emplearon por lo tanto unas 10 horas únicamente en entrenar a los modelos y comprobar su eficacia.

La tercera etapa supuso unas 100 horas de trabajo, y en esta etapa se buscó información acerca de la segmentación de caracteres en imágenes y se intentó optimizar el procesamiento de las imágenes y el vídeo de forma que se consiguiese entradas a la red neuronal muy parecida a las del entrenamiento para mejorar los resultados. Adicionalmente se realizaron las pruebas para cada modelo y se recopilaron los resultados obtenidos.

Por último, la etapa de documentación, que engloba la realización de este documento, ha supuesto un total de dos semanas, aproximadamente. Pasamos en esta etapa a documentar las pruebas y los estudios realizados previamente sobre el estado del arte.

En el siguiente cronograma puede apreciarse de manera visual la distribución del trabajo en las distintas semanas:

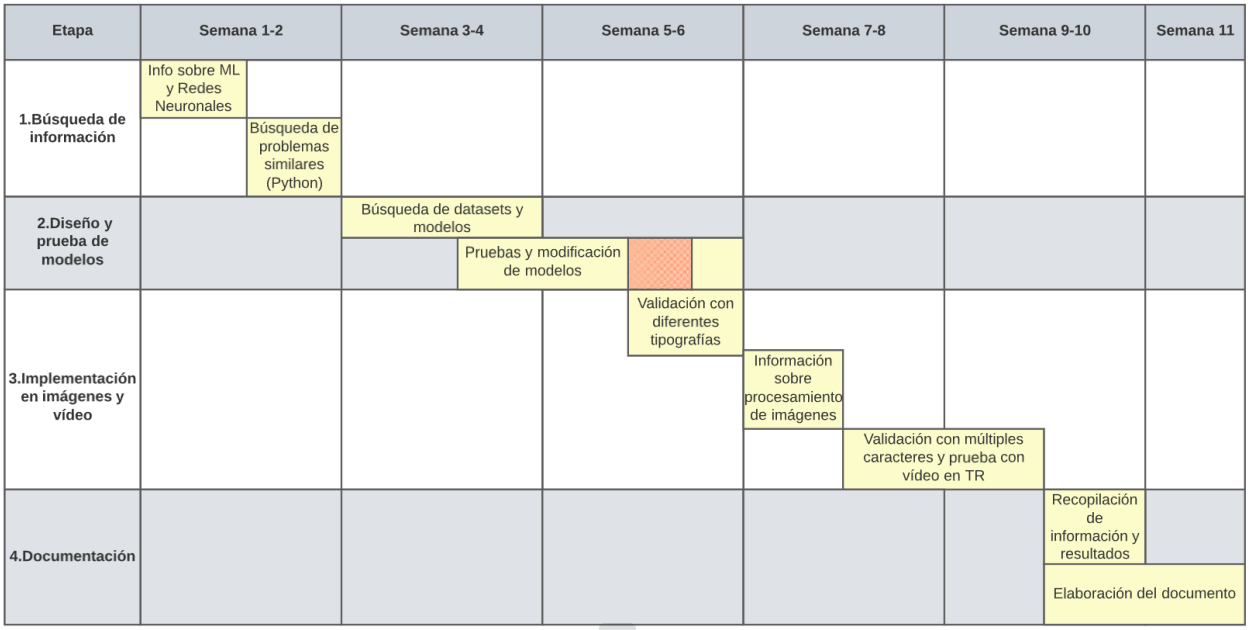


Tabla 13. Diagrama de Gantt

7.3. Presupuesto

El presupuesto que supondría que un ingeniero realizara este trabajo, tomando el sueldo de un titulado de grado superior que se expone en el BOE, en la resolución del 22 de febrero de 2018, de la Dirección General de Empleo [54], sería de unos 6300 €.

Teniendo en cuenta que el salario total anual es de 25.289,02 €, el salario mensual sería de 2.099,09 € y el tiempo trabajado de 3 meses.

En cuanto a los materiales empleados, sólo ha sido necesario un ordenador de sobremesa, cuyo valor a día de hoy es de algo menos de 1000 €.

En caso de realizar avances en el proyecto y aumentar el número de experimentos o mejorar el equipo, habría que tener esto en cuenta de cara al presupuesto final.

REFERENCIAS

[1] STELLINGWERF, Rommert; ZANDHUIS, Anton, 2013. *ISO 21500 Guidance on project management–A Pocket Guide*. Van Haren.

[2] IBM CORPORATION, 5 de enero de 2022. What Is Optical Character Recognition (OCR)? En: *IBM* [en línea]. Disponible en: https://www.ibm.com/cloud/blog/optical-character-recognition

[3] BERCHMANS, Deepa; KUMAR, S. S. Optical character recognition: an overview and an insight. En 2014 International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT). IEEE, 2014. p. 1361-1365. SHEPARD, D. H. (1951). *EEUU Patent No. US2663758A*. En: *History of Information.*

[4] AGARWAL, Rahul, 2021. Deep Learning Based OCR for Text in the Wild. En: *Nanonets* [en línea]. Disponible en: https://nanonets.com/blog/deep-learning-ocr/ [consulta: 26 de junio de 2021]

[5] TURING, Alan M. 1 de octubre de 1950. Computing Machinery and intelligence. En: *Mind*, 59(236). pp. 433-460.

[6] SERRANO, Alberto G. 2012. *Inteligencia Artificial. Fundamentos, práctica y aplicaciones*. San Fernando de Henares, Madrid: RC Libros.

[7] BOBROW, Daniel G. 1964. *Natural language input for a computer problem solving system*. PhD. Massachusetts Institute of Technology.

[8] NORVIG, Peter, 1992. *Paradigms of Artificial Intelligence Programming*. New York: Morgan Kaufmann Publishers, pp. 151–154. ISBN 1-55860-191-0.

[9] HAENLEIN, Michael; KAPLAN, Andreas., 2019 *A brief history of artificial intelligence: On the past, present, and future of artificial intelligence.* California management review, vol. 61, no 4, pp. 5-14.

[10] BINI, Stefano A. 2018. Artificial Intelligence, Machine Learning, Deep Learning, and Cognitive Computing: What Do These Terms Mean and How Will They Impact Health Care? En: *The Journal of arthroplasty, 33*(8), pp. 2358-2361.

[11] LUNA, Javier, 2018. Tipos de aprendizaje automático. En: *SoldAi* [en línea]. Disponible en: https://medium.com/soldai/tipos-de-aprendizaje-autom%C3%A1tico-6413e3c615e2

[12] ALPHASTAR TEAM, 2019. AlphaStar: Mastering the real-time strategy game StarCraft II. En: *DeepMind* [en línea]. Disponible en: https://www.deepmind.com/blog/alphastar-mastering-the-real-time-strategy-game-starcraft-ii

[13] NIELSEN, Michael A. 2015. *Neural networks and deep learning*. San Francisco, CA, USA: Determination press.

[14] BECKER, Dan, s.f. Intro to Machine Learning. En: *Kaggle* [en línea]. Disponible en: https://www.kaggle.com/learn/intro-to-machine-learning

[15] WOODFORD, Chris, 30 de junio de 2021. Neural Networks. En: *ExplainThatStuff* [en línea] Disponible en: https://www.explainthatstuff.com/introduction-to-neural-networks.html [consulta: 19 de julio de 2021]

[16] FINLAY, Steven, 2020. *Artificial Intelligence for Everyone.* Lancaster University.

[17] LLAMAS, Luis, 10 de febrero de 2019. Machine Learning con Tensorflow y Keras en Python. En: *LuisLlamas.es* [en línea]. Disponible en: https://www.luisllamas.es/machine-learning-con-tensorflow-y-keras-en-python/

[18] JOO, Gihun; PARK, Chihyun; IM, Hyeonseung, 2020. Performance evaluation of machine learning optimizers. *Journal of IKEEE*, vol. 24, no 3, p. 766-776.

[19] VANI, S.; RAO, T. 2019. An Experimental Approach towards the Performance Assessment of Various Optimizers on Convolutional Neural Network. En: *3rd International Conference on Trends in Electronics and Informatics (ICOEI)*, (pp. 331-336). Tirunelveli, India.

[20] RINGA TECH, 25 de mayo de 2022. Funciones de activación a detalle (Redes neuronales). En: *Youtube* [en línea]. Disponible en: https://www.youtube.com/watch?v=_0wdproot34&t=898s

[21] BROWNLEE, Jason, 9 de junio de 2019. A Gentle Introduction to the Rectified Linear Unit (ReLU). En: *Machine Learning Mastery* [en línea]. Diponible en: https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/

[22] GLOROT, Xavier; BORDES, Antoine; BENGIO, Yoshua. junio de 2011. Deep sparse rectifier neural networks. En: *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings. pp. 315-323.

[23] RASAMOELINA, Andrinandrasana David; ADJAILIA, Fouzia; SINČÁK, Peter, enero de 2020. A review of activation function for artificial neural network. En: *2020 IEEE 18th World Symposium on Applied Machine Intelligence and Informatics (SAMI)*. IEEE. pp. 281-286.

[24] SEGURA MORALES, Santiago; DAVID MOGOLLON, Juan; CATALINA VILLAMIL, Angie, 24 de septiembre de 2020. Reconocimiento de caracteres de documentos de identidad utilizando Tesseract y Python en 4 pasos. En: *Data Science AnalyticLab* [en línea]. Disponible en: https://datascience-alw.medium.com/reconocimiento-de-caracteres-de-documentos-utilizando-tesseract-y-python-en-4-pasos-721d861ccf8c

[25] AGARWAL, Rahul, 2021. Deep Learning Based OCR for Text in the Wild. En: *Nanonets* [en línea]. Disponible en: https://nanonets.com/blog/deep-learning-ocr/ [consulta: 26 de junio de 2021]

[26] KAZEMNEJAD, Amirhossein, 2019. How to do Deep Learning research with absolutely no GPUs - Part 2. En: *Amirhossein Kazemnejad's Blog* [en línea]. Disponible en: https://kazemnejad.com/blog/how_to_do_deep_learning_research_with_absolutely_no_gpus_part_2/

[27] TENSORFLOW TEAM, 26 de Enero de 2022. Guardar y cargar modelos. En: *TensorFlow* [en línea]. Disponible en: https://www.tensorflow.org/tutorials/keras/save_and_load [consulta: 18 de mayo de 2022]

[28] KERAS, s.f.. Keras datasets. En: *keras.io* [en línea]. Disponible en: https://keras.io/api/datasets/ [consulta: 27 de julio de 2021]

[29] JUPYTER TEAM, 2015. Jupyter Use and Configure. Recuperado el 26 de Julio de 2021. En: *Jupyter.org* [en línea]. Disponible en: https://docs.jupyter.org/en/latest/projects/architecture/content-architecture.html [consulta: 26 de julio de 2021]

[30] ROSEBROCK, Adrian, 17 de agosto de 2020. OCR with Keras, TensorFlow, and Deep Learning. En: *Pyimagesearch* [en línea]. Disponible en: https://pyimagesearch.com/2020/08/17/ocr-with-keras-tensorflow-and-deep-learning/

[31] GUPTA, Jay, 3 de mayo de 2020. Going beyond 99% — MNIST Handwritten Digits Recognition. En: *Towards Data Science* [en línea]. Disponible en: https://towardsdatascience.com/going-beyond-99-mnist-handwritten-digits-recognition-cfff96337392

[32] FINCHER, Jon, 16 de julio de 2018. Reading and Writing CSV Files in Python. En: *Real Python* [en línea]. Disponible en: https://realpython.com/python-csv/

[33] LYNN, Shane, s.f. Read CSV data quickly into Pandas DataFrames with read_csv. En: *shanelynn.ie* [en línea]. Disponible en: https://www.shanelynn.ie/python-pandas-read-csv-load-data-from-csv-files/ [consulta: 5 de agosto de 2021]

[34] COHEN, Gregory, et al. 2017. *EMNIST: an extension of MNIST to handwritten letters*. En: *International joint conference on neural networks (IJCNN)*. IEEE, 2017. p. 2921-2926.

[35] MB-F, 7 de agosto de 2017. Transpose on 4-d array in numpy. En: *Stack Overflow* [en línea]. Disponible en: https://stackoverflow.com/questions/45547182/transpose-on-4-d-array-in-numpy

[36] VISHAL, 25 de julio de 2021. Python random sample() to choose multiple items from any sequence. En *PYnative* [en línea]. Disponible en: https://pynative.com/python-random-sample/ [consulta: 19 de mayo de 2022]

[37] XIAO, Han; RASUL, Kashif; VOLLGRAF, Roland, 2017. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv*:1708.07747

[38] JD SPORTS, s.f. [Fotografía 3D de Adidas Originals Forum Low]. En: *JD Sports* [en línea]. Disponible en: https://www.jdsports.es/product/blanco-adidas-originals-forum-low/16536270_jdsportses/ [consulta: 25 de junio de 2022]

[39] BROWNLEE, Jason, 17 de abril de 2019. How Do Convolutional Layers Work in Deep Learning Neural Networks? En: *Machine Learning Mastery* [en línea]. Disponible en: https://machinelearningmastery.com/convolutional-layers-for-deep-learning-neural-networks/

[40] DIGITALSREENI, 25 de agosto de 2021. 152 - How to visualize convolutional filter outputs in your deep learning model? En: *Youtube* [en línea]. Disponible en: https://www.youtube.com/watch?v=ho6JXE3EbZ8

[41] RINGA TECH, 16 de agosto de 2021. Redes Neuronales Convolucionales - Clasificación avanzada de imágenes con IA / ML (CNN). En: *Youtube* [en línea]. Disponible en: https://www.youtube.com/watch?v=4sWhhQwHqug&t=1223s

[42] NICOLAI NIELSEN - Computer Vision & AI, 30 de diciembre de 2020. Training, Validation and Test Datasets in Neural Networks and Deep Learning. En: *Youtube* [en línea]. Disponible en: https://www.youtube.com/watch?v=z335ClO7b2k

[43] LYASHENKO, Vladimir, s.f. Data Augmentation in Python: Everything You Need to Know. En: *Neptune.ai* [en línea]. Disponible en: https://neptune.ai/blog/data-augmentation-in-python [consulta: 19 de diciembre de 2021]

[44] HE, Kaiming et al. 2016. Deep residual learning for image recognition. En *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770-778).

[45] RUIZ, Pablo, 8 de octubre de 2018. Understanding and visualizing ResNets. En*: Towards Data Science* [en línea]. Disponible en: https://towardsdatascience.com/understanding-and-visualizing-resnets-442284831be8

[46] CODE WITH AAROHI, 14 de julio de 2020. ResNet Explained Step by Step (Residual Networks). En: *Youtube* [en línea]. Disponible en: https://www.youtube.com/watch?v=Uuc1wdqMFtQ

[47] TUTORIAL KART TEAM, s.f.. OpenCV Resize image using cv2.resize(). En: *TutorialKart* [en línea]. Disponible en: https://www.tutorialkart.com/opencv/python/opencv-python-resize-image/ [consulta: 31 de mayo de 2022]

[48] SAHA, Avinab, 5 de junio de 2017. Read, Write and Display a video using OpenCV. En: *Learn OpenCV* [en línea]. Disponible en: https://learnopencv.com/read-write-and-display-a-video-using-opencv-cpp-python/

[49] PHILLIP, Karl, 13 de febrero de 2018. How to find a letter in an Image with python. En: *Stack Overflow* [en línea]. Disponible en: https://stackoverflow.com/questions/48768604/how-to-find-a-letter-in-an-image-with-python

[50] SABLE, Anuj, 2022. Building Custom Deep Learning Based OCR models. En: Nanonets [en línea]. Disponible en: https://nanonets.com/blog/attention-ocr-for-text-recogntion/ [consulta: 2 de agosto de 2022]

[51] VASQUEZ, Irving, 2 de diciembre de 2021. Explicación de YOLO para detección de objetos. En: *Youtube* [en línea]. Disponible en: https://www.youtube.com/watch?v=3h1_-AaVjWY

[52] APRENDE E INGENIA, 14 de mayo de 2022. APRENDE A DETECTAR OBJETOS EN TIEMPO REAL | Entrena y ejecuta tu propia red neuronal con Yolov5. En: *Youtube* [en línea]. Disponible en: https://www.youtube.com/watch?v=Hb5xHY4e2Mg&t=1105s

[53] MAGESHWARAN, R, 29 de abril de 2021. Scene Text Detection In Python With EAST and CRAFT. En: Medium.com [en línea]. Disponible en: https://medium.com/technovators/scene-text-detection-in-python-with-east-and-craft-cbe03dda35d5

[54] GOBIERNO DE ESPAÑA. Salario de un ingeniero. En: *Agencia Estatal Boletín Oficial del Estado* [en línea]. Disponible en: https://www.boe.es/diario_boe/txt.php?id=BOE-A-2018-3156

GLOSARIO

OCR: Reconocimiento óptico de caracterers

IA/AI: Inteligencia Artificial

DL: Deep Learning

ML: Machine Learning

b: Sesgo

w: peso

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

RAM: Recurrent Attention Model

CRNN: Convolutional Recurrent Neural Networks

R-CNN: Regional CNN

BOE: Boletín Oficial del Estado

ANEXO A

Código correspondiente al ejemplo visto en la introducción de entrenamiento de una red lineal:

```
1. #2022 Roberto Gallo TFG
2. #Ejemplo red lineal
3. #Importamos librerias necesarias
4. import tensorflow as tf
5. import numpy as np
6. import matplotlib.pyplot as plt
7.
8. #Definimos red neuronal
9. capa1 = tf.keras.layers.Dense(units=3, input_shape=[2])
10.     capa2 = tf.keras.layers.Dense(units=3)
11.     salida = tf.keras.layers.Dense(units=1)
12.     modelo = tf.keras.Sequential([capa1, capa2, salida])
13.     #Compilamos indicando perdidas y optimizador
14.     modelo.compile(
15.         optimizer=tf.keras.optimizers.Adam(0.1),
16.         loss='mean_squared_error'
17.     )
18.
19.     #Datos de entrenamiento
20.     llamadas =
    np.array([[200,10], [145,12], [300,20], [32,1], [20,10], [100,60
    ], [1,1]], dtype=float)
21.     coste =
    np.array([34.40, 28.88, 51.80, 9.38, 12.8, 49.40, 5.66],
    dtype=float)
22.     print("Comenzando entrenamiento...")
23.     historial = modelo.fit(llamadas, coste, epochs=2000,
    verbose=False)
24.     print("Modelo entrenado!")
25.
26.     #Grafica de perdidas
27.     plt.figure(figsize = (8,6))
28.     plt.xlabel("Época")
29.     plt.ylabel("Magnitud de pérdida")
30.     plt.xlim([-5, 500])
31.     plt.ylim([-100, 5000])
32.     plt.xticks(np.arange(0, 500, step=50))  # Set label
    locations.
33.     #plt.yticks(np.arange(0, 1000, step=100))  # Set label
    locations.
34.     plt.grid()
35.     plt.title('Pérdidas del modelo')
36.     plt.plot(historial.history["loss"])
37.     plt.savefig('perdidas_modelo_red_simple.png')
38.
39.
```

```
40.     print("Hagamos una predicción!")
41.     llamada=np.array([[20,10]], dtype=float)
42.     resultado = modelo.predict([llamada])
43.     print(resultado)
44.     print('El resultado para ' + str(llamada[0][0]) +
45.           ' minutos en ' + str(round(llamada[0][1]))  + '
  llamadas es ' + str(round(resultado[0][0],2)) + '€')
46.
47.
48.     print("Hagamos una predicción!")
49.     llamada=np.array([[0,0]], dtype=float)
50.     resultado = modelo.predict([llamada])
51.     print('El resultado para ' + str(llamada[0][0]) +
52.           ' minutos en ' + str(round(llamada[0][1]))  + '
  llamadas es ' + str(round(resultado[0][0],2)) + '€')
53.
54.     capa1 = tf.keras.layers.Dense(units=2,
  input_shape=[2],activation="relu")
55.     capa2 = tf.keras.layers.Dense(units=50, activation="relu")
56.     salida = tf.keras.layers.Dense(units=50,
  activation="linear")
57.     modelo2 = tf.keras.Sequential([capa1, capa2, salida])
58.
59.     modelo2.compile(
60.         optimizer=tf.keras.optimizers.Adam(0.1),
61.         loss='mean_squared_error'
62.     )
63.
64.     print("Comenzando entrenamiento...")
65.     historial = modelo.fit(llamadas, coste, epochs=2000,
  verbose=False)
66.     print("Modelo entrenado!")
67.
68.
69.     print("Hagamos una predicción!")
70.     llamada=np.array([[0,0]], dtype=float)
71.     resultado = modelo.predict([llamada])
72.     print('El resultado para ' + str(llamada[0][0]) +
73.           ' minutos en ' + str(round(llamada[0][1]))  + '
  llamadas es ' + str(round(resultado[0][0],2)) + '€')
```

ANEXO B

Código correspondiente a la extracción de características de una red convolucional:

```
#2022 Roberto Gallo TFG
#Extraccion de la capa de caracteristicas
#Importamos librerias necesarias
from keras.models import Sequential
from keras.layers.core import Flatten, Dense, Dropout
from keras.layers.convolutional import Convolution2D,
MaxPooling2D, ZeroPadding2D
from keras.optimizers import SGD
from keras.utils import load_img, img_to_array
import numpy as np
    from matplotlib import pyplot as plt
    from keras.models import Model
    import tensorflow as tf
    import cv2

    #Diseñamos y compilamos red con 3 capas convolucionales
    modelo = tf.keras.Sequential([
        tf.keras.layers.Conv2D(64, (3,3),
input_shape=(28,28,1), activation='relu'),
        tf.keras.layers.MaxPooling2D(2,2), #2,2 es el tamano de
la matriz

        tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
        tf.keras.layers.MaxPooling2D(2,2), #2,2 es el tamano de
la matriz

        tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
        tf.keras.layers.MaxPooling2D(2,2), #2,2 es el tamano de
la matriz


        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(units=100, activation='relu'),
        tf.keras.layers.Dense(47, activation='softmax')
    ])


    modelo.compile(
        optimizer='adam',
        loss=tf.keras.losses.SparseCategoricalCrossentropy(),
        metrics=['accuracy']
    )

    print(modelo.summary())

    #Visualizamos el kernel en una capa
```

```
42.     layer = modelo.layers #Las capas convolucionales son la 0 y
  la 2
43.     filters, biases = modelo.layers[0].get_weights()
44.     print(layer[0].name, filters.shape)
45.
46.     #Mostramos los filtros
47.     fig1=plt.figure(figsize=(8, 12))
48.     columns = 8
49.     rows = 8
50.     n_filters = columns * rows
51.     for i in range(0, n_filters):
52.         f = filters[:, :, :, i]
53.         fig1 =plt.subplot(rows, columns, i+1)
54.         fig1.set_xticks([])  #Turn off axis
55.         fig1.set_yticks([])
56.         plt.imshow(f[:, :, 0], cmap='gray')
57.     plt.show()
58.
59.     #Definimos un modelo mas corto para obtener como salida la
  convolucion
60.     conv_layer_index = [0, 2, 4]
61.     outputs
  = [modelo.layers[i].output for i in conv_layer_index]
62.     model_short = Model(inputs=modelo.inputs, outputs=outputs)
63.     print(model_short.summary())
64.
65.     #La entrada es 28x28, asi que ajustamos la imagen de entrada
  a este tamanio
66.     img = load_img('extraccion_caracteristicas/imagen_1.jpeg',
  target_size=(28, 28)) #VGG user 224 as input
67.
68.     # convert the image to an array
69.     img = img_to_array(img)
70.     img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
71.     img = img.astype("float32") / 255.0
72.     img = np.expand_dims(img, axis=0)
73.
74.     # Generamos la salida de nuestra red
75.     feature_output = model_short.predict(img)
76.
77.     columns = 8
78.     rows = 8
79.     for ftr in feature_output:
80.         fig=plt.figure(figsize=(12, 12))
81.         for i in range(1, columns*rows +1):
82.             fig =plt.subplot(rows, columns, i)
83.             fig.set_xticks([])  #Turn off axis
84.             fig.set_yticks([])
85.             plt.imshow(ftr[0, :, :, i-1], cmap='gray')
86.         plt.show()
```

ANEXO C

Código correspondiente al entrenamiento de los modelos CNN:

```
1. #Importamos librerias necesarias
2.
3. import numpy as np
4. import tensorflow as tf
5. import torch
6. import pandas as pd
7. import matplotlib.pyplot as plt
8. import seaborn as sns
9. from keras.models import Sequential
10.     from keras.layers import Dense, Conv2D, Flatten,
   MaxPooling2D, BatchNormalization, Activation, Dropout
11.     from tensorflow.keras.optimizers import Adam
12.     from tensorflow.keras.preprocessing.image import ImageDataGe
   nerator
13.     from keras.layers.convolutional import Conv2D
14.     from keras.layers.convolutional import MaxPooling2D
15.     from keras import backend as K
16.     from keras.utils import np_utils
17.     from keras.utils import load_img, img_to_array
18.     from sklearn.model_selection import train_test_split
19.     import cv2
20.     from imutils import build_montages
21.     import random
22.     from torch.utils.data.sampler import SubsetRandomSampler
23.     import models
24.     from datetime import datetime
25.
26.     #Llamamos y almacenamos los Datasets de prueba
27.
28.     print("ADQUIRIENDO DATASET...")
29.     data_train= pd.read_csv('emnist-balanced-train.csv') #Datos
   de entrenamiento
30.     data_test= pd.read_csv('emnist-balanced-test.csv') #Datos
   para el test
31.     print("DATASET CARGADO")
32.
33.     #Funcion para normalizar los datasets (Hacerlos de tipo
   float32
34.     #y con valores entre 0 y 1)
35.
36.     def Normalizar(data, label):
37.         x = data.drop(label ,axis=1)
38.         x=x.astype('float32')
39.         x=x/255
40.         y = data[label]
41.         x= x.values.reshape(-1,28,28,1)
```

```
42.         x=np.transpose(x,[0, 2, 1, 3]) #Hay que transponer los
  datos porque si no el caracter no se escribe correctamente
43.         return x,y
44.
45.     #El entrenamiento y el test tienen etiquetas distintas
46.     train_x, train_y= Normalizar (data_train,'45')
47.     test_x, test_y = Normalizar(data_test, '41')
48.
49.     #Imprimimos diagrama de distribución de nuestros datos de
  entrenamiento. En el eje x se representaran
50.     #las variables y en el eje y, el numero de veces que aparece
51.     sns.set(style="ticks", context="talk",font_scale = 1)
52.     #plt.style.use("dark_background")
53.     plt.figure(figsize = (30,16))
54.     ax =
  train_y.value_counts().sort_values(axis=0,ascending=False).plot(
  kind='bar',
55.         grid = False,
56.         fontsize=20,
57.         color='grey')
58.     plt.xticks(rotation=0)
59.     for p in ax.patches:
60.         height = p.get_height()
61.         ax.text(p.get_x()+ p.get_width() / 2., height + 30,
  height,
62.                 ha = 'center', size = 15)
63.     sns.despine()
64.
65.     plt.savefig('figures/distribucion_dataset.png')
66.
67.     #Lista con las diferentes etiquetas para su posterior
  traduccion
68.     #y visualizacion
69.
70.     clases=['0','1','2','3','4','5','6','7','8','9','A','B',
71.             'C','D','E','F','G','H','I','J','K','L','M','N',
72.             'O','P','Q','R','S','T','U','V','W','X','Y','Z',
73.             'a','b','d','e','f','g','h','n','q','r','t']
74.
75.     #Tomamos 9 muestras aleatorias del dataset de prueba
76.     train_samples = random.sample(range(0, len(train_x)), 9)
77.     #Mostramos los numeros que ha cogido por si es necesario
  contrastarlo con el dataset
78.     print(train_samples)
79.
80.     #Imprimimos las muestras
81.     fig,axes = plt.subplots(4,4,figsize=(10,8))
82.     plt.suptitle('Set de entrenamiento')
83.     for i in train_samples:
84.         plt.subplot(3, 3, train_samples.index(i)+1)
85.         plt.imshow(train_x[i], cmap='binary')
86.         plt.title(f'Etiqueta: {clases[train_y[i]]}')
87.         plt.axis('off')
88.     plt.savefig('figures/ejemplos_dataset.png')
89.
90.
```

```
91.     modelo_a_compilar=3
92.
93.
94.     #Modelo secuencial normal
95.     if modelo_a_compilar==1:
96.         print('Vas a usar el MODELO 1: Red Neuronal
   Simple')
97.
98.         modelo = tf.keras.Sequential([
99.             tf.keras.layers.Flatten(input_shape=(28,28,1)),
100.            tf.keras.layers.Dense(units=25, activation='relu'),
101.            tf.keras.layers.Dense(units=40, activation='relu'),
102.            tf.keras.layers.Dense(47, activation='softmax')
103.        ])
104.
105.
106.    elif modelo_a_compilar==2:
107.        print('Vas a usar el MODELO 2: Red Neuronal
   Convolucional')
108.        modelo = tf.keras.Sequential([
109.            tf.keras.layers.Conv2D(32, (3,3),
   input_shape=(28,28,1), activation='relu'),
110.            tf.keras.layers.MaxPooling2D(2,2), #2,2 es el
   tamano de la matriz
111.
112.            tf.keras.layers.Conv2D(64, (3,3),
   activation='relu'),
113.            tf.keras.layers.MaxPooling2D(2,2), #2,2 es el
   tamano de la matriz
114.
115.            tf.keras.layers.Flatten(),
116.            tf.keras.layers.Dense(units=100,
   activation='relu'),
117.            tf.keras.layers.Dense(47, activation='softmax')
118.        ])
119.
120.    #Modelo con dropout
121.
122.    elif modelo_a_compilar==3:
123.        print('Vas a usar el MODELO 3: Red Neuronal
   Convolucional con Dropout')
124.        modelo = tf.keras.models.Sequential([
125.        tf.keras.layers.Conv2D(32, (3,3), activation='relu',
   input_shape=(28, 28, 1)),
126.        tf.keras.layers.MaxPooling2D(2, 2),
127.
128.        tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
129.        tf.keras.layers.MaxPooling2D(2,2),
130.
131.        tf.keras.layers.Dropout(0.5),
132.        tf.keras.layers.Flatten(),
133.        tf.keras.layers.Dense(100, activation='relu'),
134.        tf.keras.layers.Dense(47, activation="softmax")
135.    ])
136.
```

```
137.     elif modelo_a_compilar==4:
138.         print('Vas a usar el MODELO 4: Prueba de Red Neuronal
   con numerosas capas ocultas')
139.
140.         modelo = tf.keras.models.Sequential([
141.         tf.keras.layers.Conv2D(32, (3,3), activation='relu',
   input_shape=(28, 28, 1)),
142.         tf.keras.layers.MaxPooling2D(2, 2),
143.
144.         tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
145.         tf.keras.layers.MaxPooling2D(2,2),
146.
147.         #tf.keras.layers.Dropout(0.5),
148.         tf.keras.layers.Flatten(),
149.         tf.keras.layers.Dense(100, activation='relu'),
150.         tf.keras.layers.Dense(100, activation='relu'),
151.         tf.keras.layers.Dense(100, activation='relu'),
152.         tf.keras.layers.Dense(100, activation='relu'),
153.         tf.keras.layers.Dense(47, activation="softmax")
154.         ])
155.
156.     #Compilar el modelo
157.     modelo.compile(
158.         optimizer='adam',
159.         loss=tf.keras.losses.SparseCategoricalCrossentropy(),
160.         metrics=['accuracy']
161.     )
162.
163.     modelo.summary()
164.     print('Numero de capas de la Red Neuronal')
165.     print(len(modelo.layers))
166.
167.     #Separamos test y entrenamiento
168.     X_train, X_valid, Y_train, Y_valid =
   train_test_split(train_x, train_y, test_size=0.25,
   random_state=42)
169.
170.
171.     print(train_y.shape)
172.     print(test_y.shape)
173.     print(train_x[1].shape)
174.     # Realizamos el aumento de imagenes de prueba mediante
   distorsiones, desplazamientos y giros
175.     aug = ImageDataGenerator(rotation_range=20, zoom_range=0.05,
   width_shift_range=0.1,
176.                              height_shift_range=0.1,
   shear_range=0.15, horizontal_flip=False, fill_mode="nearest")
177.
178.     #Loads in image path
179.     image_path='data/images/train/imagen_26.jpg'
180.     img = load_img(image_path, target_size= (28,28))
181.     img_tensor = img_to_array(img)
182.     img_tensor = np.expand_dims(img_tensor, axis=0)
183.     #Uses ImageDataGenerator to flip the images
184.     #Creates our batch of one image
185.     pic = aug.flow(img_tensor, batch_size =1)
```

```
    plt.figure(figsize=(10,8))
    #Plots our figures
    for i in range(1,10):
        plt.subplot(3, 3, i)
        batch = pic.next()
        image_ = batch[0].astype('uint8')
        plt.imshow(image_)
        plt.axis('off')
    plt.show()
    EPOCHS = 50
    INIT_LR = 1e-1
    BS = 128
    print('COMENZANDO ENTRENAMIENTO...')
    now = datetime.now().time() # time object
    current_time = now.strftime("%H:%M:%S")
    print("Hora de inicio =", current_time)
    H = modelo.fit(
        aug.flow(train_x, train_y, batch_size=BS),
  validation_data=( X_valid, Y_valid),
  steps_per_epoch=len(train_x) // BS, epochs=EPOCHS,
        verbose=1)
    '''H = modelo.fit(
       train_x, train_y, batch_size=BS, validation_data=(
  X_valid, Y_valid), steps_per_epoch=len(train_x) // BS,
  epochs=EPOCHS,
       verbose=1)'''
    print('LISTO')
    now = datetime.now().time() # time object
    end_time = now.strftime("%H:%M:%S")
    print("Hora de finalizacion =", end_time)


    #Mostramos los resultados del entrenamiento vs test para la
  precisión

    plt.figure(figsize = (9,9))
    print(H.history.keys())
    plt.plot(H.history['accuracy'])
    plt.plot(H.history['val_accuracy'])
    plt.title('Precisión del modelo')
    plt.ylabel('Precisión (pu)')
    plt.xlabel('Número de épocas')
    plt.legend(['Entrenamiento', 'Test'], loc='upper left')
    plt.xticks(np.arange(0, 50, step=5))  # Set label
  locations.
    plt.yticks(np.arange(0.54, 1, step=0.02))  # Set label
  locations.
    plt.grid()
    plt.savefig('figures/precision_modelo.png')
    plt.show()


    #Mostramos los resultados del entrenamiento vs test para las
  pérdidas
    plt.figure(figsize = (9,9))
```

```
233.     plt.plot(H.history['loss'])
234.     plt.plot(H.history['val_loss'])
235.     plt.title('Perdidas del modelo')
236.     plt.ylabel('Pérdidas')
237.     plt.xlabel('Número de épocas')
238.     plt.legend(['Entrenamiento', 'Test'], loc='upper left')
239.     plt.xticks(np.arange(0, 50, step=5))  # Set label
   locations.
240.     plt.yticks(np.arange(0, 1.5, step=0.05))  # Set label
   locations.
241.     plt.grid()
242.     plt.savefig('figures/perdidas_del_modelo.png')
243.     plt.show()
244.
245.     test_samples = random.sample(range(0, len(test_x)), 9)
246.
247.     print(test_samples)
248.
249.     fig,axes = plt.subplots(4,4,figsize=(10,8))
250.     plt.suptitle('Prueba de predicción')
251.     for i in test_samples:
252.         #print(test_y)
253.         #print(X_test.shape)
254.         probs = modelo.predict(test_x[np.newaxis, i])
255.         print(probs)
256.         prediction = probs.argmax(axis=1)
257.         print(prediction)
258.         label_predict = clases[prediction[0]]
259.         plt.subplot(3, 3, test_samples.index(i)+1)
260.         #imagen=train_x.loc[[i]].values.reshape(28,28)
261.         #imagen=np.transpose(imagen)
262.         #print(prediction[i])
263.         #print(clases[label_predict[i]])
264.         plt.imshow(test_x[i], cmap='binary')
265.         plt.title(f'Era: {clases[test_y[i]]} predijo:
   {label_predict}')
266.         plt.axis('off')
267.         plt.savefig('figures/pruebas_prediccion_modelo.png')
268.
269.     # save the model to disk
270.     print()
271.     if modelo_a_compilar==1:
272.
273.         modelo.save('saved_model/my_modelSimple',
   save_format="h5")
274.     elif modelo_a_compilar==2:
275.         modelo.save('saved_model/my_modelConv',
   save_format="h5")
276.     elif modelo_a_compilar==3:
277.         modelo.save('saved_model/my_modelDrop',
   save_format="h5")
278.     elif modelo_a_compilar==3:
279.         modelo.save('saved_model/my_modelMal',
   save_format="h5")
```

ANEXO D

Código perteneciente al Algoritmo de entrenamiento del ResNet50:

```
#2022 Roberto Gallo TFG
#Entrenamiento ResNet
#Importamos librerias necesarias

import numpy as np
import tensorflow as tf
import torch
import pandas as pd
import matplotlib.pyplot as plt
    import seaborn as sns
    from keras.models import Sequential
    from keras.layers import Dense
    from keras.layers import Dropout
    from keras.layers import Flatten
    from tensorflow.keras.optimizers import Adam
    from tensorflow.keras.preprocessing.image import ImageDataGe
nerator
    from keras.layers.convolutional import Conv2D
    from keras.layers.convolutional import MaxPooling2D
    from keras import backend as K
    from keras.utils import np_utils
    import cv2
    from imutils import build_montages
    import random
    from torch.utils.data.sampler import SubsetRandomSampler
    import models
    from sklearn.preprocessing import LabelBinarizer

    #Llamamos y almacenamos los Datasets de prueba
    print("ADQUIRIENDO DATASET...")
    data_train= pd.read_csv('emnist-balanced-train.csv')
    data_test= pd.read_csv('emnist-balanced-test.csv')
    print("DATASET CARGADO")

    #Funcion para normalizar los datasets (Hacerlos de tipo
float32
    #y con valores entre 0 y 1)

    def Normalizar(data, label):
        x = data.drop(label ,axis=1)
        x=x.astype('float32')
        x=x/255
        y = data[label]
        x= x.values.reshape(-1,28,28,1)
        x=np.transpose(x,[0, 2, 1, 3])
        return x,y

    #El entrenamiento y el test tienen etiquetas distintas
```

```
47.     train_x, train_y= Normalizar (data_train,'45')
48.     train_x
  = [cv2.resize(image, (32, 32)) for image in train_x]
49.     train_x = np.expand_dims(train_x, axis=-1)
50.
51.     test_x, test_y = Normalizar(data_test, '41')
52.     test_x = [cv2.resize(image, (32, 32)) for image in test_x]
53.     test_x = np.expand_dims(test_x, axis=-1)
54.
55.     #Lista con las diferentes etiquetas para su posterior
  traduccion
56.     #y visualizacion
57.
58.     clases=['0','1','2','3','4','5','6','7','8','9','A','B',
59.             'C','D','E','F','G','H','I','J','K','L','M','N',
60.             'O','P','Q','R','S','T','U','V','W','X','Y','Z',
61.             'a','b','d','e','f','g','h','n','q','r','t']
62.
63.     #Con este bloque imprimimos y mostramos diferentes
  caracteres de
64.     #nuestras pruebas
65.
66.     train_samples = random.sample(range(0, len(train_x)), 9)
67.     print(train_samples)
68.     fig,axes = plt.subplots(4,4,figsize=(10,8))
69.     plt.suptitle('Training set')
70.     for i in train_samples:
71.         plt.subplot(3, 3, train_samples.index(i)+1)
72.         plt.imshow(train_x[i], cmap='binary')
73.         plt.title(f'Etiqueta: {clases[train_y[i]]}')
74.         plt.axis('off')
75.
76.     print('Vas a usar el MODELO ResNet50')
77.
78.     EPOCHS = 50
79.     INIT_LR = 1e-1
80.     BS = 128
81.     model =
  models.ResNet.build(32, 32, 1, 47, (3, 3, 3), (64, 64, 128, 256),
  reg=0.0005)
82.     model.compile(loss="categorical_crossentropy",
  optimizer='adam', metrics=["accuracy"])
83.
84.     le = LabelBinarizer()
85.     train_y = le.fit_transform(train_y)
86.     ounts = train_y.sum(axis=0)
87.
88.     test_y = le.fit_transform(test_y)
89.
90.     classTotals = train_y.sum(axis=0)
91.     classWeight = {}
92.     model.summary()
93.     print(len(model.layers))
94.
95.     #Separamos test y entrenamiento
```

```
96.     X_train, X_valid, Y_train, Y_valid =
   train_test_split(train_x, train_y, test_size=0.25,
   random_state=42)
97.
98.
99.     print(train_y.shape)
100.    print(test_y.shape)
101.    print(train_x[1].shape)
102.    # Realizamos el aumento de imagenes de prueba mediante
   distorsiones, desplazamientos y giros
103.    aug = ImageDataGenerator(rotation_range=20, zoom_range=0.05,
   width_shift_range=0.1,
104.                             height_shift_range=0.1,
   shear_range=0.15, horizontal_flip=False, fill_mode="nearest")
105.
106.    #Loads in image path
107.    image_path='data/images/train/imagen_26.jpg'
108.    img = load_img(image_path, target_size= (28,28))
109.    img_tensor = img_to_array(img)
110.    img_tensor = np.expand_dims(img_tensor, axis=0)
111.    #Uses ImageDataGenerator to flip the images
112.    #Creates our batch of one image
113.    pic = aug.flow(img_tensor, batch_size =1)
114.    plt.figure(figsize=(10,8))
115.    #Plots our figures
116.    for i in range(1,10):
117.        plt.subplot(3, 3, i)
118.        batch = pic.next()
119.        image_ = batch[0].astype('uint8')
120.        plt.imshow(image_)
121.        plt.axis('off')
122.    plt.show()
123.
124.    print("[INFO] compiling model...")
125.
126.    EPOCHS = 50
127.    INIT_LR = 1e-1
128.    BS = 128
129.    print("[INFO] training network...")
130.
131.    H = model.fit(
132.        aug.flow(X_train, Y_train, batch_size=BS),
   validation_data=( X_valid, Y_valid),
   steps_per_epoch=len(X_train) // BS, epochs=EPOCHS,
133.        verbose=1)
134.
135.    print('LISTO')
136.
137.    #Mostramos los resultados del entrenamiento vs test para la
   precisión
138.
139.    plt.figure(figsize = (9,9))
140.    print(H.history.keys())
141.    plt.plot(H.history['accuracy'])
142.    plt.plot(H.history['val_accuracy'])
```

```
143.    plt.title('Precisión del modelo')
144.    plt.ylabel('Precisión (pu)')
145.    plt.xlabel('Número de épocas')
146.    plt.legend(['Entrenamiento', 'Test'], loc='upper left')
147.    plt.xticks(np.arange(0, 50, step=5))  # Set label
   locations.
148.    plt.yticks(np.arange(0.7, 1, step=0.01))  # Set label
   locations.
149.    plt.grid()
150.    plt.savefig('figures/precision_modelo.png')
151.    plt.show()
152.
153.    #Mostramos los resultados del entrenamiento vs test para las
   pérdidas
154.    plt.figure(figsize = (9,9))
155.    plt.plot(H.history['loss'])
156.    plt.plot(H.history['val_loss'])
157.    plt.title('Perdidas del modelo')
158.    plt.ylabel('Pérdidas')
159.    plt.xlabel('Número de épocas')
160.    plt.legend(['Entrenamiento', 'Test'], loc='upper left')
161.    plt.xticks(np.arange(0, 50, step=5))  # Set label
   locations.
162.    plt.yticks(np.arange(0, 1.2, step=0.05))  # Set label
   locations.
163.    plt.grid()
164.    plt.savefig('figures/perdidas_del_modelo.png')
165.    plt.show()
166.
167.    test_samples = random.sample(range(0, len(test_x)), 9)
168.
169.    print(test_samples)
170.    fig,axes = plt.subplots(4,4,figsize=(10,8))
171.    plt.suptitle('Prueba prediccion')
172.    for i in test_samples:
173.        probs = model.predict(test_x[np.newaxis, i])
174.        print(probs)
175.        prediction = probs.argmax(axis=1)
176.        print(prediction)
177.        label_predict = clases[prediction[0]]
178.        plt.subplot(3, 3, test_samples.index(i)+1)
179.        temp = np.where(test_y[i] == 1)
180.        temp=temp[0].astype('int32')
181.        plt.imshow(test_x[i], cmap='binary')
182.        plt.title(f'Era: {clases[temp[0]]} Predijo:
   {label_predict}')
183.        plt.axis('off')
184.
185.    #Guardar modelo entrenado
186.    print('Guardando modelo')
187.    model.save('saved_model/model_resnet', save_format="h5")
```

ANEXO E

Código correspondiente al entrenamiento con dos modelos CNN para letras y números

```
1. #2022 Roberto Gallo TFG
2. #Entrenamiento separando letras y numeros
3. #Importamos librerias necesarias
4. import numpy as np
5. import tensorflow as tf
6. import torch
7. import pandas as pd
8. import matplotlib.pyplot as plt
9. import seaborn as sns
10.     from keras.models import Sequential
11.     from keras.layers import Dense, Conv2D, Flatten,
   MaxPooling2D, BatchNormalization, Activation, Dropout
12.     from tensorflow.keras.optimizers import Adam
13.     from tensorflow.keras.preprocessing.image import ImageDataGe
   nerator
14.     from keras.layers.convolutional import Conv2D
15.     from keras.layers.convolutional import MaxPooling2D
16.     from keras import backend as K
17.     from keras.utils import np_utils
18.     from imutils import build_montages
19.     import random
20.     from torch.utils.data.sampler import SubsetRandomSampler
21.
22.     #Llamamos y almacenamos los Datasets de prueba
23.     print("ADQUIRIENDO DATASET...")
24.     letters_train= pd.read_csv('emnist-letters-train.csv')
25.     letters_test= pd.read_csv('emnist-letters-test.csv')
26.     digits_train= pd.read_csv('emnist-mnist-train.csv')
27.     digits_test= pd.read_csv('emnist-mnist-test.csv')
28.     print("DATASET CARGADO")
29.
30.     #Funcion para normalizar los datasets (Hacerlos de tipo
   float32
31.     #y con valores entre 0 y 1)
32.     def Normalizar(data, label):
33.         x = data.drop(label ,axis=1)
34.         x=x.astype('float32')
35.         x=x/255
36.         y = data[label]
37.         x= x.values.reshape(-1,28,28,1)
38.         x=np.transpose(x,[0, 2, 1, 3])
39.         return x,y
40.
41.     #El entrenamiento y el test tienen etiquetas distintas
42.     train_letters_x, train_letters_y=
   Normalizar (letters_train,'23')
```

```
43.     train_digits_x, train_digits_y=
  Normalizar (digits_train,'4')
44.
45.     test_letters_x, test_letters_y=
  Normalizar (letters_test,'1')
46.     test_digits_x, test_digits_y= Normalizar (digits_test,'1')
47.
48.     #Este bloque se utiliza para comprobar que nuestros datos de
  entrenamiento
49.     #se distribuyen uniformemente en los números.
50.     sns.set(style="ticks", context="talk",font_scale = 1)
51.     plt.style.use("dark_background")
52.     plt.figure(figsize = (20,10))
53.     ax =
  train_digits_y.value_counts().sort_values(ascending=False).plot(k
  ind='bar',
54.         grid = False,
55.         fontsize=20,
56.         color='grey')
57.     plt.xticks(rotation=0)
58.     for p in ax.patches:
59.         height = p.get_height()
60.         ax.text(p.get_x()+ p.get_width() / 2., height + 30,
  height,
61.                 ha = 'center', size = 30)
62.     sns.despine()
63.
64.     #Este bloque se utiliza para comprobar que nuestros datos de
  entrenamiento
65.     #se distribuyen uniformemente en las letras.
66.
67.     sns.set(style="ticks", context="talk",font_scale = 1)
68.     plt.style.use("dark_background")
69.     plt.figure(figsize = (20,10))
70.     ax =
  train_letters_y.value_counts().sort_values(ascending=False).plot(
  kind='bar',
71.         grid = False,
72.         fontsize=20,
73.         color='grey')
74.     plt.xticks(rotation=0)
75.     for p in ax.patches:
76.         height = p.get_height()
77.         ax.text(p.get_x()+ p.get_width() / 2., height + 30,
  height,
78.                 ha = 'center', size = 30)
79.     sns.despine()
80.
81.     clases_letters=['0','A','B',
82.             'C','D','E','F','G','H','I','J','K','L','M','N',
83.             'O','P','Q','R','S','T','U','V','W','X','Y','Z']
84.     clases_digits=['0','1','2','3','4','5','6','7','8','9']
85.
86.     #Con este bloque imprimimos y mostramos diferentes
  caracteres de
87.     #nuestras pruebas
88.
```

```
89.     train_samples
  = random.sample(range(0, len(train_letters_x)), 9)
90.
91.     print(train_samples)
92.
93.     fig,axes = plt.subplots(4,4,figsize=(10,8))
94.     plt.suptitle('Training set letras')
95.     for i in train_samples:
96.         plt.subplot(3, 3, train_samples.index(i)+1)
97.         plt.imshow(train_letters_x[i], cmap='binary')
98.         plt.title(f'Etiqueta:
  {clases_letters[train_letters_y[i]]}')
99.         plt.axis('off')
100.
101.    #Con este bloque imprimimos y mostramos diferentes
  caracteres de
102.    #nuestras pruebas
103.
104.    train_samples
  = random.sample(range(0, len(train_digits_x)), 9)
105.
106.    print(train_samples)
107.
108.    fig,axes = plt.subplots(4,4,figsize=(10,8))
109.    plt.suptitle('Training set letras')
110.    for i in train_samples:
111.        plt.subplot(3, 3, train_samples.index(i)+1)
112.        plt.imshow(train_digits_x[i], cmap='binary')
113.        plt.title(f'Etiqueta:
  {clases_digits[train_digits_y[i]]}')
114.        plt.axis('off')
115.
116.    modelo = Sequential()
117.    modelo.add(Conv2D(32, kernel_size = 3,
  activation='relu',input_shape=(28, 28, 1)))
118.    modelo.add(BatchNormalization())
119.    modelo.add(Conv2D(32, kernel_size = 3, activation='relu'))
120.    modelo.add(BatchNormalization())
121.    modelo.add(Conv2D(32, kernel_size = 5, strides=2,
  padding='same', activation='relu'))
122.    modelo.add(BatchNormalization())
123.    modelo.add(Dropout(0.4))
124.
125.    modelo.add(Conv2D(64, kernel_size = 3, activation='relu'))
126.    modelo.add(BatchNormalization())
127.    modelo.add(Conv2D(64, kernel_size = 3, activation='relu'))
128.    modelo.add(BatchNormalization())
129.    modelo.add(Conv2D(64, kernel_size = 5, strides=2,
  padding='same', activation='relu'))
130.    modelo.add(BatchNormalization())
131.    modelo.add(Dropout(0.4))
132.
133.    modelo.add(Conv2D(128, kernel_size = 4, activation='relu'))
134.    modelo.add(BatchNormalization())
135.    modelo.add(Flatten())
136.    modelo.add(Dropout(0.4))
137.    modelo.add(Dense(47, activation='softmax'))
```

```
138.     #Compilar el modelo
139.     modelo.compile(
140.         optimizer='adam',
141.         loss=tf.keras.losses.SparseCategoricalCrossentropy(),
142.         metrics=['accuracy']
143.     )
144.
145.     #Separamos test y entrenamiento
146.     X_letters_train, X_letters_valid, Y_letters_train,
   Y_letters_valid = train_test_split(train_letters_x,
   train_letters_y, test_size=0.25, random_state=42)
147.
148.     #Separamos test y entrenamiento
149.     X_digits_train, X_digits_valid, Y_digits_train,
   Y_digits_valid = train_test_split(train_digits_x, train_digits_y,
   test_size=0.25, random_state=42)
150.
151.     #Realizar aumento de datos
152.     aug = ImageDataGenerator(rotation_range=10, zoom_range=0.05,
   width_shift_range=0.1, height_shift_range=0.1, shear_range=0.15,
   horizontal_flip=False, fill_mode="nearest")
153.
154.     print('Entrenamiento letras')
155.     EPOCHS = 50
156.     INIT_LR = 1e-1
157.     BS = 128
158.     H = modelo.fit(
159.         aug.flow(train_letters_x, train_letters_y,
   batch_size=BS), validation_data=( X_letters_valid,
   Y_letters_valid), steps_per_epoch=len(train_letters_y) // BS,
   epochs=EPOCHS,
160.         verbose=1)
161.     print('LISTO')
162.
163.     #Mostramos los resultados del entrenamiento vs test para la
   precisión
164.
165.     plt.figure(figsize = (4,4))
166.     print(H.history.keys())
167.     plt.plot(H.history['accuracy'])
168.     plt.plot(H.history['val_accuracy'])
169.     plt.title('Precisión del modelo')
170.     plt.ylabel('Precisión (pu)')
171.     plt.xlabel('Número de épocas')
172.     plt.legend(['Entrenamiento', 'Test'], loc='lower right')
173.     plt.xticks(np.arange(0, 50, step=5))  # Set label locations.
174.     plt.yticks(np.arange(0.7, 1, step=0.02))  # Set label
   locations.
175.     plt.grid()
176.     plt.savefig('figures/precision_modelo.png')
177.     plt.show()
178.
179.
180.     #Mostramos los resultados del entrenamiento vs test para las
   pérdidas
181.     plt.figure(figsize = (4,4))
182.     plt.plot(H.history['loss'])
```

```
183.     plt.plot(H.history['val_loss'])
184.     plt.title('Perdidas del modelo')
185.     plt.ylabel('Pérdidas')
186.     plt.xlabel('Número de épocas')
187.     plt.legend(['Entrenamiento', 'Test'], loc='upper left')
188.     plt.xticks(np.arange(0, 50, step=5))  # Set label locations.
189.     plt.yticks(np.arange(0, 1.8, step=0.1))  # Set label
   locations.
190.     plt.grid()
191.     plt.savefig('figures/perdidas_del_modelo.png')
192.     plt.show()
193.
194.     modelo.save('saved_model/modelo_solo_letras',
   save_format="h5")
195.     print('Entrenamiento digitos')
196.     EPOCHS = 50
197.     INIT_LR = 1e-1
198.     BS = 128
199.     H = modelo.fit(
200.         aug.flow(train_digits_x, train_digits_y, batch_size=BS),
   validation_data=( X_digits_valid, Y_digits_valid),
   steps_per_epoch=len(train_digits_y) // BS, epochs=EPOCHS,
201.         verbose=1)
202.     print('LISTO')
203.     modelo.save('saved_model/modelo_solo_digitos',
   save_format="h5")
```

ANEXO F

Programa dedicado a la predicción de caracteres individuales (CNNs y ResNet):

```
#2022 Roberto Gallo TFG
#Prediccion usando los tres primeros modelos
#Importamos librerias necesarias
from tensorflow.keras.models import load_model
from imutils.contours import sort_contours
import numpy as np
import argparse
import imutils
import cv2
    import matplotlib.pyplot as plt

    print("[INFO] loading handwriting OCR model...")
    model = load_model('saved_model/model_resnet') #ruta del modelo guardado
    caracteres=['imagenes/caracter_individual/0.jpeg', 'imagenes/caracter_individual/7.jpeg',
                'imagenes/caracter_individual/8.jpeg', 'imagenes/caracter_individual/8_deformado.jpeg',
                'imagenes/caracter_individual/a.jpeg', 'imagenes/caracter_individual/b_girada.jpeg',
                'imagenes/caracter_individual/i.jpeg', 'imagenes/caracter_individual/l_mayuscula.jpeg',
                'imagenes/caracter_individual/l_mayuscula_desplazada.jpeg', 'imagenes/caracter_individual/l_minuscula.jpeg',
                'imagenes/caracter_individual/m.jpeg', 'imagenes/caracter_individual/r.jpeg']
    clases=['0','1','2','3','4','5','6','7','8','9','A','B','C','D','E','F','G','H','I','J',
    'K','L','M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z','a','b','d','e','f','g','h','n','q','r','t']


    im = cv2.imread(caracteres[2])
    # calculate mean value from RGB channels and flatten to 1D array
    vals = im.mean(axis=2).flatten()
    # plot histogram with 255 bins
    b, bins, patches = plt.hist(vals, 255)
    plt.xlim([0,255])
    plt.show()
    j=0
    for i in caracteres:

        image = cv2.imread(i)
        scale_percent = 50
        width = int(image.shape[1] * scale_percent / 100)
        height = int(image.shape[0] * scale_percent / 100)
        dim = (width, height)
```

```
39.
40.         # resize image
41.         image_res = cv2.resize(image, dim, interpolation =
    cv2.INTER_AREA)
42.         gray = cv2.cvtColor(image_res, cv2.COLOR_BGR2GRAY)
43.         blurred = cv2.GaussianBlur(gray, (5, 5), 0)
44.         edged = cv2.Canny(blurred, 10, 100)
45.
46.         cnts = cv2.findContours(edged.copy(),
    cv2.RETR_EXTERNAL,
47.             cv2.CHAIN_APPROX_SIMPLE)
48.         cnts = imutils.grab_contours(cnts)
49.
50.
51.         #print(image.shape)
52.         image = cv2.threshold(blurred, 0, 255,
53.             cv2.THRESH_BINARY_INV | cv2.THRESH_OTSU)[1]
54.
55.         width = 28 #int(image.shape[1] * scale_percent / 100)
56.         height = 28 #int(image.shape[0] * scale_percent / 100)
57.         dim = (width, height)
58.         cv2.imshow("cd", image)
59.         image = cv2.resize(image, dim, interpolation =
    cv2.INTER_AREA)
60.
61.
62.
63.         cv2.waitKey(0)
64.         image = image.astype("float32") / 255.0
65.
66.         image= image.reshape(-1,28,28,1)
67.         print(image.shape)
68.
69.         image = np.array(image, dtype="float32")
70.         #print(image)
71.         preds = model.predict(image)
72.         i = np.argmax(preds)
73.
74.         prob = preds[0][i]
75.         label = clases[i]
76.         #print(prob)
77.         #print(label)
78.
79.         print("[INFO] {} - {:.2f}%".format(label, prob * 100))
80.
81.         image = cv2.imread(caracteres[j])
82.         j=j+1
83.
84.
85.         cv2.putText(image_res, label, (50, 70),
86.             cv2.FONT_HERSHEY_SIMPLEX, 3, (0, 0, 0), 2)
87.
88.
89.         # show the image
90.         cv2.imshow("Image", image_res)
```

```
91.          cv2.waitKey(0)
92.
93.
94.      image = image.astype("float32") / 255.0
95.      image = np.expand_dims(image, axis=0)
96.      print(image.shape)
97.
98.      # OCR the characters using our handwriting recognition model
99.      preds = model.predict(image)
100.
101.     clases=['0','1','2','3','4','5','6','7','8','9','A','B','C',
    'D','E','F','G','H','I','J',
102.     'K','L','M','N','O','P','Q','R','S','T','U','V','W','X','Y',
    'Z','a','b','d','e','f','g','h','n','q','r','t']
103.
104.     i = np.argmax(preds)
105.     print(preds)
106.     print(i)
107.     prob = preds[0][i]
108.     label = clases[i]
109.     print(prob)
110.     print(label)
111.
112.     print("[INFO] {} - {:.2f}%".format(label, prob * 100))
113.
114.     image = cv2.imread('imagenes/imagen_G_desplazado.jpeg')
115.
116.     cv2.putText(image, label, (50, 50),
117.         cv2.FONT_HERSHEY_SIMPLEX, 1.2, (0, 255, 0), 2)
118.
119.
120.     # show the image
121.     cv2.imshow("Image", image)
122.     cv2.waitKey(0)
```

ANEXO G

Código correspondiente a la predicción de caracteres individuales (2 Redes CNN para letras y números):

```
#2022 Roberto Gallo TFG
#Prediccion usando el cuarto modelo
#Importamos librerias necesarias
from tensorflow.keras.models import load_model
from imutils.contours import sort_contours
import numpy as np
import argparse
import imutils
import cv2
    import matplotlib.pyplot as plt
    print("[INFO] loading handwriting OCR model...")
    model1 = load_model('saved_model/modelo_solo_letras') #ruta del modelo guardado
    model2 = load_model('saved_model/modelo_solo_digitos') #ruta del modelo guardado
    caracteres=['imagenes/caracter_individual/0.jpeg', 'imagenes/caracter_individual/7.jpeg',
                'imagenes/caracter_individual/8.jpeg', 'imagenes/caracter_individual/8_deformado.jpeg',
                'imagenes/caracter_individual/a.jpeg', 'imagenes/caracter_individual/b_girada.jpeg',
                'imagenes/caracter_individual/i.jpeg', 'imagenes/caracter_individual/l_mayuscula.jpeg',
                'imagenes/caracter_individual/l_mayuscula_desplazada.jpeg', 'imagenes/caracter_individual/l_minuscula.jpeg',
                'imagenes/caracter_individual/m.jpeg', 'imagenes/caracter_individual/r.jpeg']
    clases_letras=['0','A','B',
            'C','D','E','F','G','H','I','J','K','L','M','N',
            'O','P','Q','R','S','T','U','V','W','X','Y','Z']
    clases_digitos=['0','1','2','3','4','5','6','7','8','9']

    im = cv2.imread(caracteres[2])
    # calculate mean value from RGB channels and flatten to 1D array
    vals = im.mean(axis=2).flatten()
    # plot histogram with 255 bins
    b, bins, patches = plt.hist(vals, 255)
    plt.xlim([0,255])
    plt.show()

    j=0
    for i in caracteres:

        image = cv2.imread(i)
```

```
37.         scale_percent = 50 #
38.         width = int(image.shape[1] * scale_percent / 100)
39.         height = int(image.shape[0] * scale_percent / 100)
40.         dim = (width, height)
41.
42.         # resize image
43.         image_res = cv2.resize(image, dim, interpolation =
   cv2.INTER_AREA)
44.         gray = cv2.cvtColor(image_res, cv2.COLOR_BGR2GRAY)
45.         blurred = cv2.GaussianBlur(gray, (5, 5), 0)
46.         edged = cv2.Canny(blurred, 10, 100)
47.         cnts = cv2.findContours(edged.copy(),
   cv2.RETR_EXTERNAL,
48.             cv2.CHAIN_APPROX_SIMPLE)
49.         cnts = imutils.grab_contours(cnts)
50.
51.
52.         image = cv2.threshold(gray, 120, 255,
53.             cv2.THRESH_BINARY_INV)[1]
54.
55.         width = 28 #int(image.shape[1] * scale_percent / 100)
56.         height = 28 #int(image.shape[0] * scale_percent / 100)
57.         dim = (width, height)
58.         image = cv2.resize(image, dim, interpolation =
   cv2.INTER_AREA)
59.
60.
61.         cv2.imshow("cd", image)
62.         cv2.waitKey(0)
63.         image = image.astype("float32") / 255.0
64.         image= image.reshape(-1,28,28,1)
65.         print(image.shape)
66.         image = np.array(image, dtype="float32")
67.         preds_letra = model1.predict(image)
68.         i = np.argmax(preds_letra)
69.         #print(preds)
70.         #print(i)
71.         prob_letra = preds_letra[0][i]
72.         label_letra = clases_letras[i]
73.     #-------------
74.         preds_digito = model2.predict(image)
75.         i = np.argmax(preds_digito)
76.         prob_digito= preds_digito[0][i]
77.         label_digito = clases_digitos[i]
78.
79.         if prob_letra>prob_digito:
80.             label=label_letra
81.             prob=prob_letra
82.         else:
83.             label=label_digito
84.             prob=prob_digito
85.         #print(prob)
86.         #print(label)
87.
88.         print("[INFO] {} - {:.2f}%".format(label_letra,
   prob_letra * 100))
```

```
89.         print("[INFO] {} - {:.2f}%".format(label_digito,
  prob_digito * 100))
90.
91.         image = cv2.imread(caracteres[j])
92.         j=j+1
93.
94.
95.         cv2.putText(image_res, label, (50, 70),
96.             cv2.FONT_HERSHEY_SIMPLEX, 3, (0, 0, 0), 2)
97.
98.
99.
100.        cv2.imshow("Image", image_res)
101.        cv2.waitKey(0)
```

ANEXO H

Programa diseñado para predicción de varios caracteres (CNNs y ResNet)

```
#2022 Roberto Gallo TFG
#Prediccion de varios caracteres usando los tres primeros modelos
#Importamos librerias necesarias
from tensorflow.keras.models import load_model
from imutils.contours import sort_contours
import numpy as np
import argparse
import imutils
import cv2
    print("[INFO] loading handwriting OCR model...")
    model = load_model('saved_model/model_resnet') #ruta del modelo guardado
    image = cv2.imread('imagenes/letras_numeros.jpeg')

    scale_percent = 50 # percent of original size
    width = int(image.shape[1] * scale_percent / 100)
    height = int(image.shape[0] * scale_percent / 100)
    dim = (width, height)

    # resize image
    image = cv2.resize(image, dim, interpolation = cv2.INTER_AREA)
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    blurred = cv2.GaussianBlur(gray, (5, 5), 0)
    edged = cv2.Canny(blurred, 60, 100)
    cv2.imshow("Image", edged)
    cv2.waitKey(0)
    cnts = cv2.findContours(edged.copy(), cv2.RETR_EXTERNAL,
            cv2.CHAIN_APPROX_SIMPLE)
    cnts = imutils.grab_contours(cnts)
    cnts = sort_contours(cnts, method="left-to-right")[0]
    #Crear cortina blanca
    temp = np.ones(edged.shape,np.uint8)*255
    # Dibuje el contorno: la temperatura es la cortina blanca, los contornos son el contorno, -1 es la imagen completa, luego el color, el grosor
    cv2.drawContours(temp,cnts,-1,(0,255,0),3)
    cv2.imshow("contours",temp)
    cv2.waitKey(0)

    #Inicializar la lista de contornos y caracteres asociados

    chars = []

    # loop
    for c in cnts:
        (x, y, w, h) = cv2.boundingRect(c)
```

```
44.
45.         #Nos aseguramos que los contornos no son demasiados
    grandes ni demasiado pequenios
46.         if (h >=(image.shape[0])/30) and (h<20*w) and (w<10*h):
47.             roi = gray[y:y + h, x:x + w]
48.             thresh = cv2.threshold(roi, 0, 255,
49.                 cv2.THRESH_BINARY_INV | cv2.THRESH_OTSU)[1]
50.             (tH, tW) = thresh.shape
51.
52.             if tW > tH:
53.                 thresh = imutils.resize(thresh, width=32)
54.
55.             else:
56.                 thresh = imutils.resize(thresh, height=32)
57.
58.             #Determinamos cuanto hay redimensionar la imagen
59.             (tH, tW) = thresh.shape
60.             dX = int(max(0, 32 - tW) / 2.0)
61.             dY = int(max(0, 32 - tH) / 2.0)
62.
63.             # Forzamos una dimension de 28x28
64.             padded = cv2.copyMakeBorder(thresh, top=dY,
    bottom=dY,
65.                 left=dX, right=dX,
    borderType=cv2.BORDER_CONSTANT,
66.                 value=(0, 0, 0))
67.             padded = cv2.resize(padded, (32, 32))
68.
69.             #Preparamos la imagen para su clasificacion
70.             padded = padded.astype("float32") / 255.0
71.             padded = np.expand_dims(padded, axis=-1)
72.             print(padded.shape)
73.
74.             #Actualizamos lista
75.             chars.append((padded, (x, y, w, h)))
76.
77.     #Lanzamos prediccion
78.     preds = model.predict(chars)
79.     clases=['0','1','2','3','4','5','6','7','8','9','A','B','C',
    'D','E','F','G','H','I','J',
80.     'K','L','M','N','O','P','Q','R','S','T','U','V','W','X','Y',
    'Z','a','b','d','e','f','g','h','n','q','r','t']
81.
82.     for (pred, (x, y, w, h)) in zip(preds, boxes):
83.         i = np.argmax(pred)
84.         prob = pred[i]
85.         label = clases[i]
86.             #Dibujamos los cuadros delimitadores
87.         print("[INFO] {} - {:.2f}%".format(label, prob * 100))
88.         if (prob*100)>40:
89.             cv2.rectangle(image, (x, y), (x + w, y +
    h), (0, 255, 0), 2)
90.             cv2.putText(image, label, (x - 10, y - 10),
91.                 cv2.FONT_HERSHEY_SIMPLEX, 1.2, (0, 255, 0), 2)
92.
```

```
93.
94.     # Mostramos la imagen
95.     cv2.imshow("Image", image)
96.     cv2.waitKey(0)
97.
98.     #Prueba sobre un dataset completo
99.     print("ADQUIRIENDO DATASET...")
100.    data_test= pd.read_csv('emnist-balanced-test.csv')
101.    print("DATASET CARGADO")
102.
103.    def Normalizar(data, label):
104.        x = data.drop(label ,axis=1)
105.        x=x.astype('float32')
106.        x=x/255
107.        y = data[label]
108.        x= x.values.reshape(-1,28,28,1)
109.        x=np.transpose(x,[0, 2, 1, 3]) #Hay que transponer los
  datos porque si no el caracter no se escribe correctamente
110.        return x,y
111.
112.    test_x, test_y= Normalizar (data_test,'41')
113.    test_x = [cv2.resize(image, (32, 32)) for image in test_x]
114.    test_x = np.expand_dims(test_x, axis=-1)
115.    correctas=0
116.    incorrectas=0
117.    prueba=0
118.    for i in range(len(test_x)):
119.            #print(test_y)
120.            #print(X_test.shape)
121.            probs = model.predict(test_x[np.newaxis,
  i],  verbose=False)
122.            #print(probs)
123.            prediction = probs.argmax(axis=1)
124.            #print(prediction)
125.            #label_predict = clases[prediction[0]]
126.            #plt.subplot(3, 3, test_samples.index(i)+1)
127.            #imagen=train_x.loc[[i]].values.reshape(28,28)
128.            #imagen=np.transpose(imagen)
129.            #print(prediction[i])
130.            #print(clases[label_predict[i]])
131.            #plt.imshow(test_x[i], cmap='binary')
132.            if test_y[i]==prediction[0]:
133.                correctas=correctas+1
134.            else:
135.                incorrectas=incorrectas+1
136.            prueba=prueba+1
137.            print(prueba)
138.    print("Datos correctos:")
139.    print(correctas)
140.    print("Datos incorrectos:")
141.    print(incorrectas)
142.    print("Total:")
143.    print(correctas+incorrectas)
```

ANEXO I

Código correspondiente a la predicción de varios caracteres (2 Redes CNN)

```
1. #2022 Roberto Gallo TFG
2. #Prediccion de varios caracteres usando el cuarto modelo
3. #Importamos librerias necesarias
4. from tensorflow.keras.models import load_model
5. from imutils.contours import sort_contours
6. import numpy as np
7. import argparse
8. import imutils
9. import cv2
10.     print("[INFO] loading handwriting OCR model...")
11.     model_letras =
    load_model('saved_model/modelo_solo_letras') #ruta del modelo
    guardado
12.     #model_digitos =
    load_model('saved_model/modelo_solo_digitos') #ruta del modelo
    guardado
13.     model_digitos =
    load_model('saved_model/modelo_solo_digitos') #ruta del modelo
    guardado
14.     image = cv2.imread('imagenes/letras_numeros.jpeg')
15.
16.     scale_percent = 50
17.     width = int(image.shape[1] * scale_percent / 100)
18.     height = int(image.shape[0] * scale_percent / 100)
19.     dim = (width, height)
20.
21.     image = cv2.resize(image, dim, interpolation =
    cv2.INTER_AREA)
22.     gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
23.     blurred = cv2.GaussianBlur(gray, (5, 5), 0)
24.     edged = cv2.Canny(blurred, 60, 100)
25.     cv2.imshow("Image", edged)
26.     cv2.waitKey(0)
27.     cnts = cv2.findContours(edged.copy(), cv2.RETR_EXTERNAL,
28.             cv2.CHAIN_APPROX_SIMPLE)
29.     cnts = imutils.grab_contours(cnts)
30.     cnts = sort_contours(cnts, method="left-to-right")[0]
31.     #Crear cortina blanca
32.     temp = np.ones(edged.shape,np.uint8)*255
33.     # Dibuje el contorno: la temperatura es la cortina blanca,
    los contornos son el contorno, -1 es la imagen completa, luego el
    color, el grosor
34.     cv2.drawContours(temp,cnts,-1,(0,255,0),3)
35.
36.     cv2.imshow("contours",temp)
37.     cv2.waitKey(0)
```

```
38.     chars = []
39.
40.     for c in cnts:
41.         (x, y, w, h) = cv2.boundingRect(c)
42.
43.         # nor too large
44.         if (h >=(image.shape[0])/25) and (h<20*w) and (w<10*h):
45.             roi = gray[y:y + h, x:x + w]
46.             thresh = cv2.threshold(roi, 0, 255,
47.                 cv2.THRESH_BINARY_INV | cv2.THRESH_OTSU)[1]
48.             (tH, tW) = thresh.shape
49.             if tW > tH:
50.                 thresh = imutils.resize(thresh, width=28)
51.
52.             else:
53.                 thresh = imutils.resize(thresh, height=28)
54.
55.             (tH, tW) = thresh.shape
56.             dX = int(max(0, 28 - tW) / 2.0)
57.             dY = int(max(0, 28 - tH) / 2.0)
58.
59.             padded = cv2.copyMakeBorder(thresh, top=dY,
  bottom=dY,
60.                 left=dX, right=dX,
  borderType=cv2.BORDER_CONSTANT,
61.                 value=(0, 0, 0))
62.             padded = cv2.resize(padded, (28, 28))
63.             padded = padded.astype("float32") / 255.0
64.             padded = np.expand_dims(padded, axis=-1)
65.             print(padded.shape)
66.
67.             chars.append((padded, (x, y, w, h)))
68.
69.     boxes = [b[1] for b in chars]
70.     chars = np.array([c[0] for c in chars], dtype="float32")
71.     preds_letras = model_letras.predict(chars)
72.     preds_digitos = model_digitos.predict(chars)
73.     clases_letras=['0','A','B','C','D','E','F','G','H','I','J',
74.     'K','L','M','N','O','P','Q','R','S','T','U','V','W','X','Y',
  'Z']
75.
76.     clases_digitos=['0','1','2','3','4','5','6','7','8','9']
77.
78.     k=0
79.     for (pred_letras,pred_digitos, (x, y, w,
  h)) in zip(preds_letras,preds_digitos, boxes):
80.
81.         i=np.argmax(pred_letras)
82.         j=np.argmax(pred_digitos)
83.         prob_letra = preds_letras[k][i]
84.         label_letra = clases_letras[i]
85.     #-------------
86.         prob_digito= preds_digitos[k][j]
87.         label_digito = clases_digitos[j]
88.
89.         k=k+1
```

```
90.          label=label_letra
91.          prob=prob_letra
92.          print("[INFO] {} - {:.2f}%".format(label_letra,
   prob_letra * 100))
93.          print("[INFO] {} - {:.2f}%".format(label_digito,
   prob_digito * 100))
94.          print("")
95.
96.          if (prob*100)> 80:
97.              cv2.rectangle(image, (x, y), (x + w, y +
   h), (0, 255, 0), 2)
98.              cv2.putText(image, label, (x - 10, y - 10),
99.                  cv2.FONT_HERSHEY_SIMPLEX, 1.2, (0, 255, 0), 2)
100.
101.
102.
103.     # show the image
104.     cv2.imshow("Image", image)
105.     cv2.waitKey(0)
```

ANEXO J

Transformación de Dataset para que pueda servir de entrada de entrenamiento para una Red YoloV5:

```
#2022 Roberto Gallo TFG
#Extraccion de imagen y etiqueta para usarlos como datos de entrenamiento en la Red YoloV5
#Importamos librerias necesarias
import os
import numpy as np
import tensorflow as tf
import torch
import pandas as pd
import matplotlib.pyplot as plt
    import seaborn as sns
    from keras.models import Sequential
    from keras.layers import Dense, Conv2D, Flatten, MaxPooling2D, BatchNormalization, Activation, Dropout
    from tensorflow.keras.optimizers import Adam
    from tensorflow.keras.preprocessing.image import ImageDataGenerator
    from keras.layers.convolutional import Conv2D
    from keras.layers.convolutional import MaxPooling2D
    from keras import backend as K
    from keras.utils import np_utils
    from sklearn.model_selection import train_test_split
    import cv2
    from imutils import build_montages
    import random
    from torch.utils.data.sampler import SubsetRandomSampler
    import models
    from PIL import Image
    #Llamamos y almacenamos los Datasets de prueba

    print("ADQUIRIENDO DATASET...")
    data_train= pd.read_csv('emnist-balanced-train.csv')
    data_test= pd.read_csv('emnist-balanced-test.csv')
    print("DATASET CARGADO")
    def Normalizar(data, label):
        x = data.drop(label ,axis=1)
        x=x.astype('float32')
        y = data[label]
        x= x.values.reshape(-1,28,28,1)
        x=np.transpose(x,[0, 2, 1, 3])

        return x,y

    train_x, train_y= Normalizar (data_train,'45')
    for i in range(train_x.shape[0]):
        cv2.imwrite(f'imagen_creada/imagen_{i}.jpg',train_x[i])
        #img = cv2.imread(f'imagen_creada/imagen_{i}.jpg')
```

```
45.         if os.path.exists(f'etiqueta_creada/imagen_{i}.txt'):
46.             os.remove(f'etiqueta_creada/imagen_{i}.txt')
47.         file = open(f'etiqueta_creada/imagen_{i}.txt', 'x')
48.         file.write(f'{train_y[i]} 0.500000 0.500000 1.000000
    1.000000')
49.         file.close()
50.     #Para el de validacion
51.     def Normalizar(data, label):
52.         x = data.drop(label ,axis=1)
53.         x=x.astype('float32')
54.         y = data[label]
55.         x= x.values.reshape(-1,28,28,1)
56.         x=np.transpose(x,[0, 2, 1, 3])
57.
58.         return x,y
59.
60.     test_x, test_y= Normalizar (data_test,'41')
61.     for i in range(test_x.shape[0]):
62.         cv2.imwrite(f'data/images/val/imagen_test_{i}.jpg',test_
    x[i])
63.         #img = cv2.imread(f'imagen_creada/imagen_{i}.jpg')
64.         if os.path.exists(f'data/labels/val/imagen_{i}.txt'):
65.             os.remove(f'data/labels/val/imagen_{i}.txt')
66.         file = open(f'data/labels/val/imagen_test_{i}.txt', 'x'
    )
67.         file.write(f'{test_y[i]} 0.500000 0.500000 1.000000
    1.000000')
68.         file.close()
```