UNIVERSIDAD EUROPEA DE MADRID

ESCUELA DE ARQUITECTURA, INGENIERÍA Y DISEÑO

MÁSTER UNIVERSITARIO EN

BIG DATA ANALYTICS - MBI

TRABAJO FIN DE MÁSTER

RETO IBM: DEEP LEARNING PARA LA CLASIFICACIÓN DE IMÁGENES DE MARCAS

NOMBRE:

CARLOS AUNIÓN DOMÍNGUEZ

CURSO 2020-2021

TÍTULO: RETO IBM: DEEP LEARNING PARA LA CLASIFICACIÓN DE IMÁGENES DE MARCAS

AUTOR: CARLOS AUNIÓN DOMÍNGUEZ

TITULACIÓN: MÁSTER UNIVERSITARIO EN BIG DATA ANALYTICS

DIRECTOR DEL PROYECTO: LUIS FERNANDEZ ORTEGA

FECHA: Octubre de 2021

RESUMEN

El trabajo conmsiste en resolver un problema de clasificación de marcas mediante reconocimiento de imagen utilizando para ello técnics de Deep Learning con redes convolucionales.

El set de datos constará de un conjunto de imágenes proporcionado por IBM que contiene logos de distintas marcas. Las imágenes estarán ya separadas por marca. No se podrá añadir ninguna imagen extra, pero sí se permitirá nutrir el propio datsaet mediante transformaciones de las propias imágenes existentes.

Palabras clave: Deep learning, reconocimiento de imagen, clasificación.

ABSTRACT

The work consists in solving a classification problem by image recognition using Deep Learning techniques with convolutional networks.

The dataset will consist of a set of images provided by IBM containing logos of different brands. The images will be already separated by brand. It will not be possible to add any extra images, but it will be possible to nurture the datsaet itself through transformations of the existing images.

Key words: Deep learning, classification, image recognition.

Universidad Europea

Índice

Universidad Europea

Índice de Figuras

Capítulo 1. INTRODUCCIÓN

1.1 Planteamiento del problema

El problema a resolver consiste en un reto de clasificación de imágenes presentado por IBM a los estudiantes de la universidad Europea de Madrid. El reto consiste en resolver un problema de clasificación de imágenes. Estas imágenes consisten en un conjunto de 22 marcas comerciales. Este conjunto de dataset es común para todos los participantes. Las marcas que conforman el dataset son AMD, Aquafina, Disney, D-Link, Domino's Pizza, Hellsmann's, IBM, Kitkat, LG, Lipton, McDonalds, Milka, Monter, Nestea, Pac-Man, Pepsi, Pizza Hut, Red Bull, Samsung, Sony, Tic-tac y Universal.

Adicionalmente, existe otra marca que es específica para cada alumno, en este caso la marca es Apple.

1.2 Objetivos del proyecto

El objetivo principal del proyecto es desarrollar un modelo de aprendizaje automático basado en Deep Learning que sea capaz de reconocer una marca en una imagen y clasificarla entre todas las posibles contempladas en el proyecto.

Además, dado mi perfil como participante en el concursto, otro de los principales objetivos de este proyecto es aprender y afianzar los conceptos adquiridos durante el curso. Son muchos los conceptos nuevos o poco profundizados presentados en este reto, como la clasificación de imágenes, el desarrollo de modelos con tensorflow o el Deep learning. Gracias a este proyecto debo profundizar en estos conceptos y ser capaz de forma primordialmente autodidacta, indagar y desarrollar un modelo aceptable para dar solución al reto.

1.3 Estructura del proyecto

Este documento servirá para recoger todos los conceptos aprendidos durante el proyecto, así como para hacer una vista general de todo lo que se ha trabajado y el seguimiento realizado hasta alcanzar la solución final al problema.

El capitulo 1 consistirá en una breve introducción en la que se describe el reto y en qué consiste el trabajo, es decir, cual es el problema a resolver y la forma de proceder para resolverlo. También se presenta en este apartado los objetivos del proyecto.

En capitulo 2 se realizará una investigación a cerca del estado del arte de la materia, explicando en qué consiste el campo y que avances existen, así como las tecnologías desarrolladas y más extendidas y cuál es su aplicación en este proyecto.

En el capitulo 3 realizaré una síntesis de toda la metodología empleada, explicando todos los pasos seguidos hasta alcanzar la solución final, así como las pruebas realizadas y los modelos probados. También resumiré los errores encontrados y los mayores problemas y bloqueos con los que me he topado, así como la solución que encontré a ellos y como llegué hasta ella. Finalmente, presentaré todos los resultados obtenidos y realizaré un breve análisis de los mismos explicando lo que significa cada medida.

El capitulo 4 será una conclusión final, en la que se podrá ver una idea general de los resultados finales del proyecto, el impacto que ha tenido y futuros trabajos relacionados.

Finalmente, se encontrarán los anexos y bibliografía utilizada durante todo el recorrido del proyecto.

Capítulo 2. Conceptos técnicos y estado del arte

2.1 Paradigma

El ámbito en el que se desarrolla este proyecto de fin de master el aprendizaje automático. El aprendizaje automático se subdivide en 3 grupos principales, que son **aprendizaje supervisado**, **aprendizaje no supervisado**, y **aprendizaje reforzado**. En concreto, este reto se calsifica dentro del aprendizaje supervisado.

El aprendizaje supervisado consiste en una técnica para, en base a una entrada, predecir un valor concreto. En función de aquello que se pretende predecir, existen dos sub-grupos en esta categoría. Si aquello que se pretende predecir es un valor **continuo** (i.e. predecir la temperatura de una zona, la umedad, etc.), es decir, predecir algo que no puede tomar un valor concreto dentro de un grupo cerrado de valores; estamos hablado de un problema de **regresión**. Por otro lado, si se pretende predecir un valor **discreto**, siendo este aquel que puede tomar un valor fijo dentro de un intervalo determinado de valores (i.e. número de hijos, marcas, etc.), nos encontraremos ante un problema de **clasificación**. Este último subgrupo es en el que se centra el reto, puesto que se pretende clasificar un conjunto de imágenes dentro de un conjunto limitado de marcas.

El reto a resolver consiste en un problema de clasificación, por lo que en todo momento **me ceñiré a presentar los conceptos relacionados y más relevantes de problemas de clasificación**.

2.2 Tratamiento de imágenes

Las imágenes en informática se sintetizan de una forma muy especial. Cada imagen se entiende como una simplificación de 3 colores: rojo, azul y verde

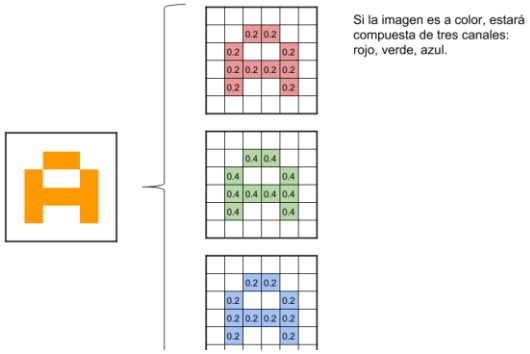

Figura 1. Composición RGB de una imagen

(de las siglas RGB en inglés: **R**ed, **G**reen, **B**lue). Esta sintetización se representa en un conjunto de 3 matrices cuyas dimensiones coinciden con las dimensiones de la imagen en píxeles (i.e. una imagen en 1280x720 se representa en un conjunto de 3 matrices de 1280 filas y 720 columnas). Cada valor de las matrices hace referencia a la cantidad de color rojo, azul o verde que tenemos en un determinado pixel, de modo que cada matriz representa la cantidad de cada color para todos los píxeles de la imagen. Así, cuando la imagen es presentada en el monitor de un ordenador, el motor gráfico indica la cantidad de rojo, azul y verde que debe adquirir cada pixel y representar así todos los espectros de colores.

Los valores contenidos en cada matriz se encuentran siempre normalizados dentro del intervalo [0, 255], tomando así 256 posibles valores (2^8) para facilitar el tratamiento de la información en lenguaje máquina o código binario.

2.3 Modelos y arquitecturas

Tradicionalmente y en el proceso de aprendizaje, es muy común encontrar ejemplos de aprendizaje supervisado basados en entradas cuyos atributos son directamente categóricos o numéricos, siendo utilizados para realizar estas predicciones modelos clásicos como KNN, Naive Bayes, J48 o Random Forest entre muchos otros. Sin embargo, a medida que el problema a predecir escala y se complica, estos modelos pueden quedarse cortos o no ser factibles debido a sus propias limitaciones, tanto temporales como para amoldarse al conjunto de datos en sí. Es por esto, que se ha popularizado cada vez más un concepto que, aunque se desarrolló a finales de la década de 1958, cuando el psicólogo e informático Frank Rosenblatt creó el concepto de perceptrón, no se ha popularizado hasta años recientes, gracias a los avances en computación y en métricas de medición, que inherentemente han traído nuevos problemas más complejos a resolver.

2.3.1 Conceptos clave

Existen algunos conceptos clave a la hora de definir el aprendizaje automático que son imprescindiples a la hora de realizar un trabajo clasificación.

2.3.2 Datasets

En primer lugar, para poder realizar el entrenamiento de un modelo es necesario contar con un buen conjunto de datos. El conjunto de datos es la materia prima del modelo de predicción, es el material que se tiene para poder

entrenar un modelo. Este conjunto de datos se compone de instancias y debe tener algunas características para que sea válido.

Es necesario que el conjunto de datos esté bien nutrido de información, no deben faltar ejemplos de ningún tipo. Es necesario que haya variedad para que el modelo sea capaz de aprender del mayor número de ejemplos posible y que, cuando sea puesto a prueba, los ejemplos que encuentre en "el mundo real" sean similares a los que ha visto durante su entrenamiento.

Se debe tener especial cuidado con que instancias se incluyen en el conjunto de datos ya que, lo normal es que debido al tamaño de un dataset común, sea nutrido de forma automática por sensores. Esto expone al conjunto de datos a contener errores de medición. Un ejemplo de esta situación sería por ejemplo si en la medida de las temperaturas tomadas por un termómetro en la calle, de pronto aparece una medida de 90ºC, lo cual ha de tratarse evidentemente de un error, pero un modelo como tal no es capaz de entenderlo y lo tomará como un valor acertado; este ejemplo se denomina outlier y existen técnicas de detección automáticas para lidiar con ellos. En el contexto del problema presentado, este tipo de error podría darse si una imagen que ha sido clasificada en el conjunto de datos como una marca, en realidad no contiene ninguna marca o pertenece a una marca diferente de la que está marcada.

2.3.3 Aprendizaje y validación

Para construir un modelo y hacer que aprenda del conjunto de datos, es necesario incluir una fase de **training**, pero también una de **testing** para poder evaluar el modelo y saber como de bien ha aprendido.

Para obtener el conjunto de training y el conjunto de testing, se divide en dos subconjuntos el conjunto de datos global. Primero se debe definir que porcentaje del conjunto de datos global será dirigido utilizado para training y que porcentaje será utilizado para testing. Esta proporción puede tomar varios valores, pero para que el aprendizaje sea correcto y los resultados de la validación sean significativos, se suele utilizar (de forma aproximada) un **66%** (2/3) para el conjunto de entrenamiento, y un **33%** (1/3) para el conjunto de testing. Sin embargo, estos valores suelen redondearse hasta alcanzar una distribución de **70/30**.

Sabiendo esto, podemos plantear realizar el **muestreo** (sampling) de varias formas distintas. Podemos seleccionar los ejemplos de forma totalmente aleatoria, lo que, si se tiene un conjunto de datos lo suficientemente nutrido, producirá que se mantenga la proporción de entrada para cada clase.

Otra forma de realizar el muestreo es obligando a formar una proporción equitativa. Para ello, se puede obtener un porcentaje fijo equivalente de cada

clase. Es decir, si el porcentaje elegido es 70/30, se toma un 70% de cada clase para entrenamiento y un 30% para testing.

2.3.4 Predicciones

Los modelos de clasificación realizan una predicción por cada instancia de entrada. Estra predicción representa la clase a la que se cree que pertenece la instancia. Usualmente, estas predicciones se presentan en forma de tabla que indica que cuantos valores predichos son correctos y cuantos incorrectos, así como el valor correcto de las instancias predichas. Esta estructura es conocida como **matriz de confusión** y es una herramienta trementamente útil a la hora de calcular las métricas del modelo, así como ver una imagen aproximada y abstraída de como es capaz de predecir el modelo.

Figura 2. Matriz de confusión

2.3.5 Métricas

Para conocer la bondad de un conjunto de entrenamiento se utilizan una serie de métricas. Estas métricas son fórmulas matemáticas que, aplicadas a las predicciones realizadas por el modelo final construido, obtiene una serie de valores que sirven como marcadores para conocer como de bueno es el modelo.

Existen métricas específicas para cada tipo de problema de machine learning (supervisado, no supervisado o reforzado), de igual modo, existen métricas específicas para los subgrupos, de aprendizaje supervisado (clasificadores y regresores). Sin embargo, como el reto a abordar es un problema de clasificación, solo se tomarán en cuenta sus métricas en este documento.

- **Accuracy**: Es quizás la métrica más popular entre todas. Aporta una vista general de como funciona el modelo. El accuracy representa el porcentaje de veces que, al predecir una clase, el modelo ha acertado la predicción.

- **Precision**. La precisión es una métrica de carácter específico. Existe un valor de precisión para cada clase a predecir e indica el porcentaje de acierto del modelo cuando predice una clase determinada.

- **Recall**. El recall es una métrica específica que indica el porcentaje de acierto del modelo cuando intenta predecir una clase determinada.

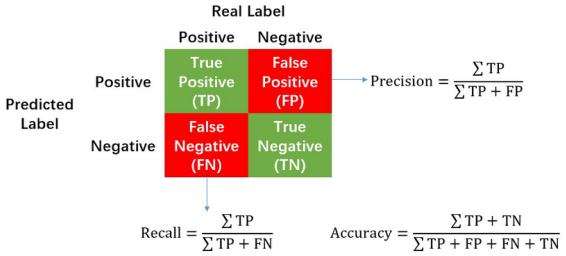

Figura 3. Formulación de métricas

- **F1 score**. El F-score es una combinación de las medidas de precisión y de recall, lo que permite encontrar el rendimiento combinado tanto en precisión como especificidad.

$$F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

2.4 Redes neuronales

2.4.1 Arquitectura del modelo

El perceptrón es la base de un modelo conocido como red neuronal, el cual ha sido el empleado en este trabajo. Las redes neuronales son un modelo computacional que consiste en un conjunto de unidades llamadas neuronas artificiales conectadas entre sí para transferir señales. La información de entrada se transforma en su paso por cada neurona, de modo que finalmente se obtiene una salida, proporcionando una resultado.

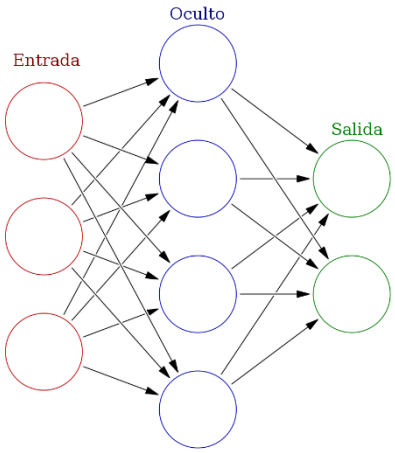

Figura 4. Esquema de una red neuronal

La arquitectura de una red neuronal consta de tres capas. La primera capa es la de **entrada**, en ella se introducen los valores representados como atributos de la instancia a predecir pudiendo ser tanto valores categóricos como numéricos. Esta capa tiene siempre tantas neuronas como atributos tenga el conjunto de datos que se va a predecir, obviando por supuesto el atributo referente a la clase target a precedir.

La segunda capa es la capa **oculta**, esta capa es el corazón de la red neuronal y es donde se realizan todas las transformaciones y operaciones del modelo a los valores de entrada. La capa oculta no tiene un número determinado de neuronas, sin embargo, el aumento del número de neuronas puede permitir al modelo ajustarse mejor a problemas más complejos. En esta capa se ejecutan una serie de operaciones no lineales con los atributos de entrada, esto permite que la regresión estimada calculada con los atributos de entrada se ajuste a problemas que no han de ser exclusivamente lineales. Que un modelo sea capaz de ajustarse a un conjunto de datos cuya relación con sigue un patron lineal, le permitirá ajustarse a este de mejor forma.

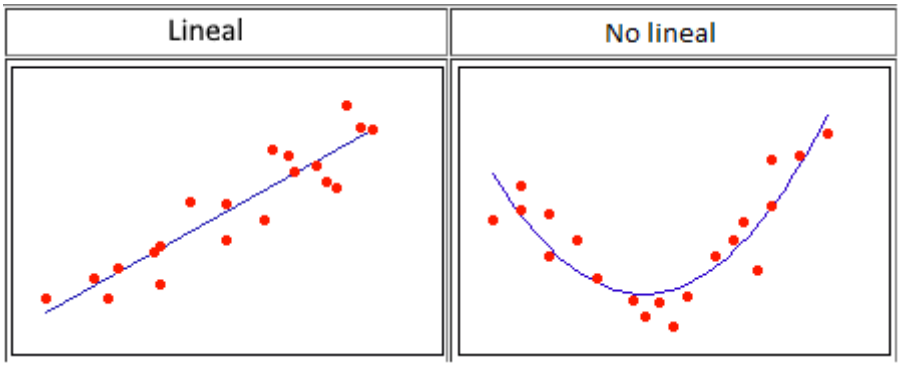

Figura 5. Comparativa entre problema lineal y no lineal

Finalmente, la última capa de la red neuronal será aquella que indicará el tipo de salida. En el caso de clasificación, tenemos una neurona de salida por cada posible clase a predecir, es decir, si existen 23 posibles clases, tendremos 23 neuronas en la capa de salida. Cada neurona obtendrá como resultado un valor dentro del intervalo [0,1]. Este valor obtenido indica la probabilidad estimada de que los atributos que ha recibido como entrada representen una instancia del tipo de su neurona. Cabe mencionar que la salida obtenida va a estar normalizada, es decir, que la suma de todos los resultados será 1. Una salida ideal en la que la clasificación estaría clara, sería aquella en la que todas las neuronas devuelven un valor muy cercano a 0, mientras que una de ellas devuelve un valor muy cercano a 1.

2.4.2 Tipos de redes neuronales

Existen distintos tipos de redes neuronales especializadas para cada tipo de problema, entre las más comunes se encuentran las redes de regresión, cuya transformación realizada en cada neurona consiste en el tunning de una regresión lineal con operaciones no lineales (por lo comentado en el subapartado anterior), muy común a la hora de resolver problemas cuyos atributos de entrada son numéricos.

Por otro lado, encontramos las redes neuronales convolucionales, que son las utilizadas en este proyecto. En este tipo de red neuronal, las neuronas pretenden imitar el proceso biológico de la corteza visual el ojo y son especialmente efectivas a la hora de realizar aprendizaje basado en imágenes.

Antes de comenzar el proceso de entrenamiento de una red neuronal convolucional, es necesario realizar siempre un preprocesado a las imágenes de entrada. Como se ha explicado en la arquitectura, la capa de entrada de una red neuronal consiste en una neurona por atributo, sin embargo, para el caso de las redes convolucionales en la entrada se ha de especificar en tamaño de la foto

en píxeles. Una vez que se conoce cual va a ser el formato de entrada de las imágenes, se establecen las capas ocultas convolucionales.

Una capa convolucional consiste en tomar un grupo de pixeles cercanos de la imagen de entrada y operar matemáticamente aplicando un producto escalar a una pequeña matriz denominada kernel. Este kernel genera una nueva matriz de salida que será la nueva entrada de la siguiente capa que se establezca en la red neuronal. De este modo, cada kernel crea un valor en función de los pixeles que se encuentran alrededor de cada pixel.

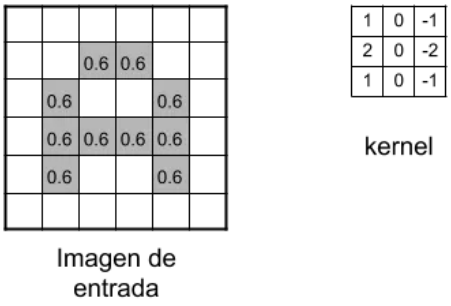

Figura 6. Kernel de entrada convolucional

Es usual aplicar algún filtro de color a la imagen para que todas las imágenes de entrada estén en blanco y negro.

2.4.3 Deep Learning de redes convolucionales

Conforme vamos interpolando capas de convolución la red neuronal va creciendo y aumenta su complejidad. A esto se le conoce como Deep Learning. Sin embargo, no solo existen capas de tipo convolucional que sean relevantes para este tipo de redes.

Un tipo de capa muy utilizado es el **MaxPooling**. Este filtro consiste en recorrer las imágenes del mismo modo que una capa convolucional. Sin embargo, en lugar de aplicar la operación del producto escalar, aplica la función max, de modo que, del conjunto de pixeles contemplados por el kernel, obtiene el valor más grande. Este tipo de capa se introduce normalmente entre

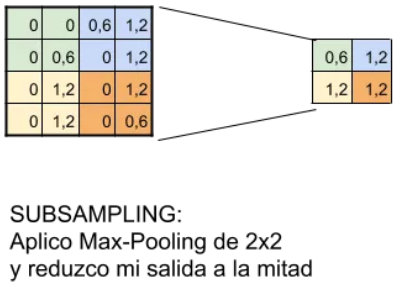

Figura 7. Transformación MaxPooling

Flatten es otro tipo de capa que se incluye usualmente tras las capas convolucionales. Esta capa sirve para aplanar la entrada recibida en forma de matriz y la descompone de modo que pase a ser un simple array bidimensional corriente.

Las capas de tipo **Dense** son aquellas que conectan todas las neuronas contenidas en la capa, con aquellas de la capa siguiente, de modo que si en la primera capa tenemos *n* neuronas y en la segunda capa *m* neuronas, se crearán ***n*m*** relaciones. En muchas arquitecturas conocidas se utilizan este tipo de capas próximas a la capa de salida de la red.

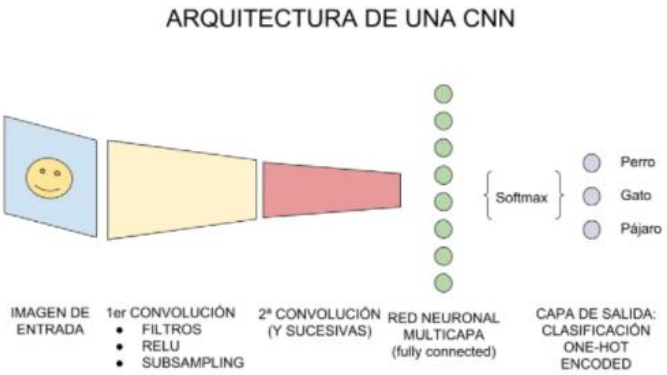

Figura 8. Arquitectura CNN estándar

2.4.4 Parámetros

Al igual que todos los modelos de aprendizaje, las redes neuronales tienen varios parámetros de entrada a tener en cuenta a la hora de **ajustar** nuestro modelo para el aprendizaje.

El **número de neuronas** y de **capas** es, quizás, el parámetro más importante de una red neuronal. Este parámetro definirá la arquitectura que tendrá la red neuronal y afectará directamente a su rendimiento. Cuantas más capas y más neuronas tenga una red, mayor será la capacidad que tenga la red

de amoldarse a un conjunto de datos. Sin embargo, añadir capas y neuronas incrementa en gran medida el coste computacional, tanto de entrenar el modelo como de realizar predicciones, por lo que se debe tener mucho cuidado a la hora de seleccionar esto. Además, aumentar excesivamente la cantidad de capas y neuronas puede producir **overfitting**.

El **ratio de aprendizaje**, también conocido por sus siglas en inglés **lr**, indica el porcentaje de cambio con el que se actualizan los pesos de cada neurona en cada iteración. Su valor oscila siempre entre 0 y 1, aunque suele ser muy próximo a 0, lo normal es encontrar un valor correcto para el learning rate por debajo del 0'1. Establecer este valor a 1 indicaría que no se realiza ninguna transformación y todos los atributos de entrada tienen el mismo peso para predecir el modelo.

2.4.5 Funciones de activación

Las **funciones de activación** son aquellas que se encargan de devolver una salida en un rango determinado que suele ser [0,1]. Es decir, son las responsables de la salida producida por cada neurona. Esta función ha de ser computacionalmente liviana, ya que de no ser así, aumentaría desorbitadamente la complejidad computacional de entrenamiento y predicción del modelo.

Aunque no se utilicen en las capas convolucionales, como hemos visto, en las capas tradicionales que se suelen aplicar al final de las arquitecturas de Deep Learning y es en estas capas donde toman relevancia. Existen infinidad de funciones de activación, alguna de las más comúnmente utilizadas son las siguientes:

- **Función sigmoide**: La función sigmoide transforma los valores introducidos a una escala de [0,1], donde los valores altos tienden la manera asintótica 1 y los valores muy bajos tienden de manera asintótica a 0.

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

- **Función ReLU**: La función de activación ReLU transforma los valores introducidos anulando los valores negativos y dejando los positivos tal y como entran.

$$f(x) = \max(0, x) = \begin{cases} 0 \; for \; x < 0 \\ x \; for \; x \geq 0 \end{cases}$$

- **Función Softmax**: La función Softmax transforma las salidas a una representación en forma de probabilidades, de tal manera que el sumatorio de todas las probabilidades de las salidas sea 1.

$$f(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$$

2.4.6 Entrenamiento

Las redes neuronales tienen una forma muy característica de entrenar. Recordemos que cada neurona de una red neuronal toma los valores de entrada multiplicados por una ponderación para representar la fortaleza de esa conexión. Para encontrar y detectar que ponderaciones son correctas y cuales no, se utiliza una función denominada **back-propagation** (propagación hacia atrás). La diferencia entre el resultado deseado y el resultado actual se calcula mediante la función de pérdida o coste. En otras palabras, la función de pérdida indica el grado de precisión que tiene actualmente la red neuronal al realizar predicciones para una entrada determinada.

$$f(m,b) = \frac{1}{N}\sum_{i=1}^{n}(y_i - (mx_i + b))^2$$

Tras esto, el algoritmo ajusta cada ponderación para minimizar la diferencia entre el valor calculado y el valor correcto. El término “back propagation” viene del hecho de que el algoritmo retrocede desde las capas finales hasta las capas iniciales, ajustando las ponderaciones y los sesgos tras calcular una respuesta. Cuanto menor sea la pérdida para una red, más precisa será. Se puede definir el proceso de aprendizaje como la reducción del resultado de la función de coste. Cada ciclo de corrección del back propagation para reducir la pérdida se denomina época.

Para calcular los pesos en cada época se utiliza una función de optimización, que ajusta los valores de cada neurona para obtener un mejor rendimiento global. Existen varias funciones optimizadoras, algunas de ellas se describen a continuación.

2.5 Optimizadores

La optimización numérica se conoce en matemáticas como un proceso consistente en maximizar o minimizar (en los casos más simples) una función real, eligiendo sistemáticamente valores de entrada tomados de un conjunto permitido y computando el valor de la función. La generalización de la teoría de la optimización y técnicas para otras formulaciones comprende un gran área de la matemática aplicada.

Esta definición, aplicada a las redes neuronales, toma parte en la búsqueda de los pesos adecuados para maximizar la precisión de la red, o para minimizar el error (dependiendo del tipo de problema que tengamos delante). Por tanto, elegir un buen optimizador a la hora de definir la arquitectura de reed neuronal es una parte importante del proceso de investigación.

- **Descenso del gradiente**. El descenso del gradiente es un algoritmo que estima numéricamente donde una función genera sus valores más bajos. Eso significa que busca mínimos locales. Para ello, comenzando en un punto x0 y nos movemos una distancia positiva α en la dirección del gradiente negativo, entonces nuestro nuevo y mejorado x1 se veguiría la siguiente expresión.

$$x_1 = x_0 - \alpha \nabla f(x_0)$$

Partiendo de esta congetura, la optimización consiste en repetir este proceso una y otra vez de forma iterativa hasta alcanzar un mínimo local.

- **Descenso de gradiente estocástico.** El cálculod e la derivada parcial de la función de coste respecto a cada uno de los pesos de la red para cada observación es, dado el número de diferentes pesos y observaciones (millones en Deep learning), inviable. Por tanto, una optimización consiste en introducir un comportamiento estocástico (aleatorio).

- **Descenso de gradiente adaptativo (AdaGrad).** Este algoritmo introduce una interesante variación al concepto de factor de entrenamiento. En lugar de considerar un valor uniforme para todos los pesos, se mantiene un factor de entrenamiento específico para cada uno de ellos. Sería inviable calcular este factor de forma específica, por lo que partiendo de un factor de entrenamiento inicial, así que es escalado y adaptado para cada dimensión con respecto al gradiente acumulado en cada iteración.

- **Adadelta.** Adadelta es una variación de AdaGrad en la que en lugar de calcular el exalado del factor de entrenamiento de cada dimensión teniendo en cuenta el gradiente acumulado desde el principio de la ejecución, se restringe a una ventana de tamaño fijo de los últimos n gradientes.

- **Root Mean Square Propagation (RMSProp).** Es un algoritmo similar, el cual mantiene un factor de entrenamiento diferente para cada dimensión, pero en este caso el escalado del factor de entrenamiento se realiza dividiéndolo por la media del declive exponencial del cuadrado de los gradientes

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(f_i - o_i)^2}$$

- **Estimación de momento adaptativa (ADAM).** Este algoritmo combina las bondades de AdaGrad y RMSProp. Se mantiene un factor de entrenamiento por parámetro y además de lascular el factor para RMSProp, cada factor de entrenamiento se ve afectad por la media del momento del gradiente.

2.6 Estado del arte

2.6.1 Arquitecturas

Actualmente, el campo del Deep learning está muy extendido y se han conseguido enormes avances, existen infinidad de arquitecturas establecidas que se sabe que funcionan muy bien de forma generalizada.

Existe un concurso anual muy famoso llamado **ImageNet Large Scale Visual Recognition Challenge** (ILSVRC) que consiste en construir un modelo que sea capaz de aprender a clasificar imágenes de entre 1000 clases diferentes. A este concurso se presentas las arquitecturas de Deep learning más actuales y prometedoras hasta la fecha para propósito general. Entre estas arquitecturas existen dos que, aunque no tienen una cantidad excesiva de parámetros ni mucha complejidad, consiguen un resultado muy bueno en el concurso: RestNet50 y VGG-16.

2.6.1.1 Resnet-50

ResNet es una estructura de red de 152 capas propuesta por He Kaiming, Sun Jian y otros miembrosde Microsoft Research Asia en 2015, que consiguió ganar el premio ILSVTC-2015 alcanzando una tasa de error de 3.57%.

La idea que plantea esta arquitectura es que no se debe aumentar indefinidamente el número de capas si, se ha comprobado empíricamente que, a diferencia de lo que pueda parecer, la red no siempre mejora su acierto conforme se añaden más capas. De hecho, al añadir capas pueden aparecer

problemas de descenso de gradiente y dimensionalidad, provocando que la red deje de aprender. También, con el aumento de capas, el accuracy llega un punto en el que se estanca y comienza a empeorar. Aquí es donde se plantea la idea de los bloques residuales.

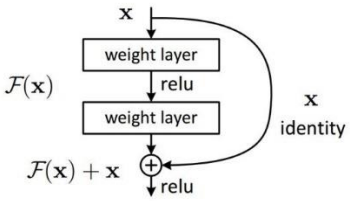

Figura 9. Bloque residual identidad

Se tiene el input x que en la red viaha a través de las capas conv-RELU-conv-RELU. Esta serie de operaciones dan como resultado F(x). Se llama a H(x) = F(x) + x. En una red convolucional tradicional se tiene que conseguir que F(x) y H(x) tengan el mismo valor, así que en lugar de buscar el valor F(x) directamente de x, con los bloques residuales se calcula qué valor se debe añadir a F(x) para que de como resultado H(x). Es decir, el bloque residual añade una pequeña alteración a la entrada para conseguir una representación ligeramente alterada, mientras que, en las CNN tradicionales cuando se pasa de x a F(x), esta última es una representación que no tiene ninguna relación con la entrada original. Esta x también va actualizándose conforme se va entrenando la red, por lo que puede ayudar a la hora de realizar las predicciones.

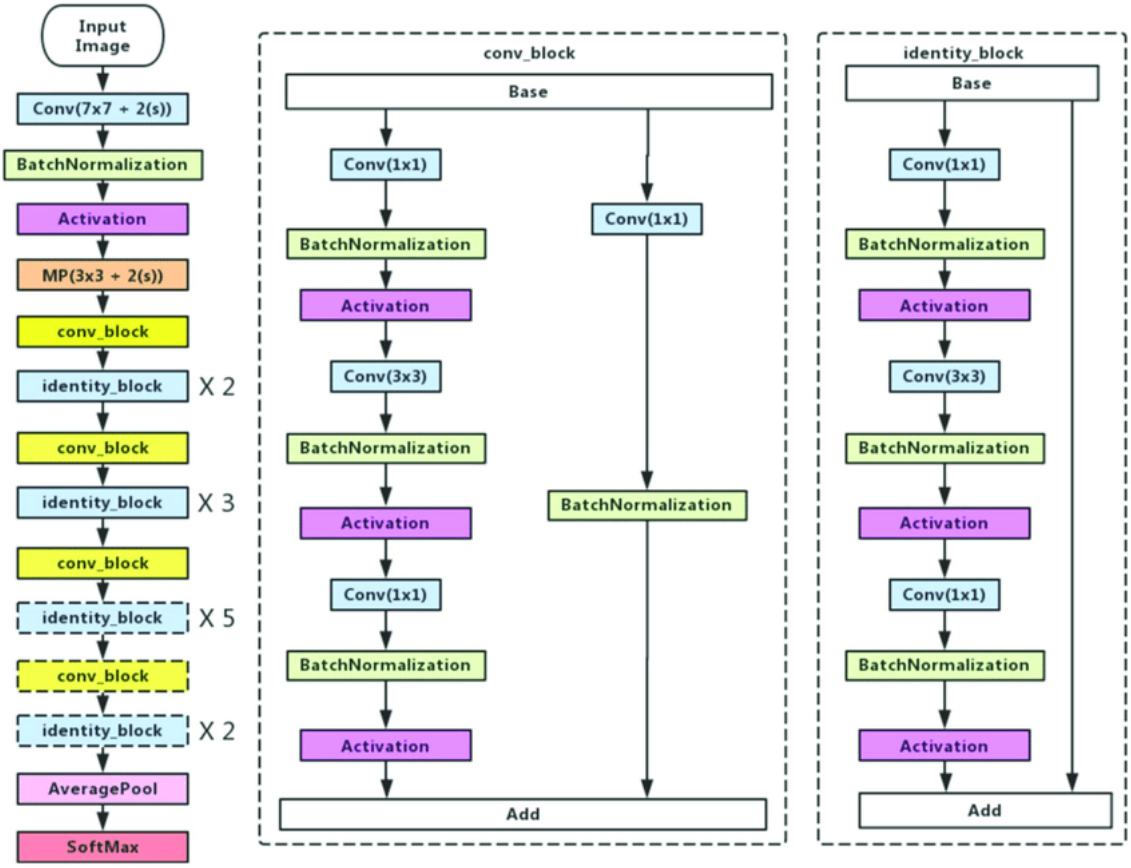

Figura 10. Arquitectura de ResNet-50

En la arquitectura de ResNet50, tenemos dos tipos de bloques residuales, los convolucionales y los bloques identidad.

- **Bloque convolucional**. El bloque convolucional consiste en dos posibles caminos. 3 bloques formados por 1 capa convolucional, 1 capa de normalización y 1 capa de activación. En caso de utilizar el otro camino, no se aplicaría únicamente una operación de suma, sino que utilizaría una capa convolucional y una capa de normalización.
- **Bloque identidad**. Es el explicado en el ejemplo anterior, los posibles caminos son 3 bloques formados por 1 capa convolucional, 1 capa de normalización y 1 capa de activación o la suma de la identidad x.

2.6.1.2 VGG-16

La arquitectura VGGNet se planteó por primera vez e 2014 por Simonyan y Zisserman. Esta arquitectura ha sido muy influyente ya que reforzó el razonamiento intuitivo detrás de las CNN de que para que la red logre una

representación jerárquica lo suficientemente buena d ellos datos, la red ha de tener una estructura de capas profunda.

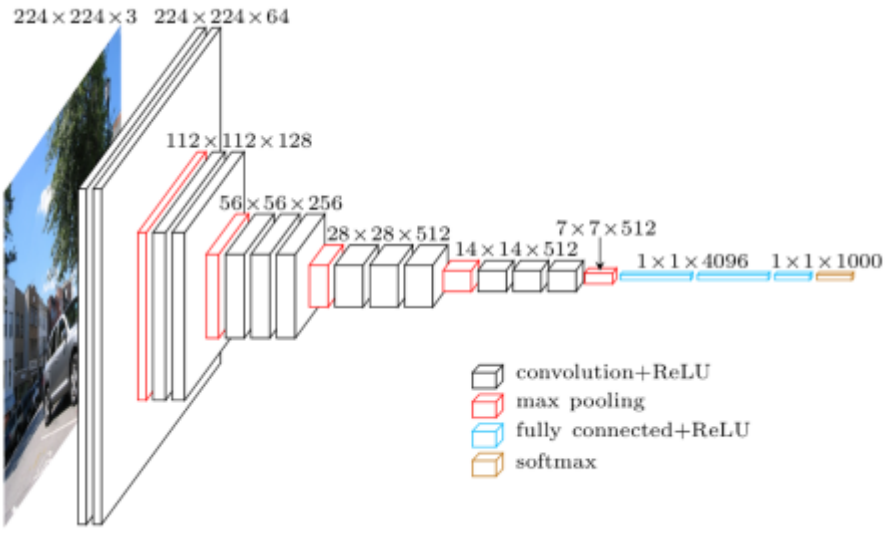

Figura 11. Arquitectura red VGG-16

Uno de los puntos que hizo destacar esta red fue su gran desempeño a pesar de su simplicidad. Está formada por 13 capas convolucionales, tras las que se aplica la función de activación ReLu, con filtros de tamaño 3x3, y capas de MaxPooling 2x2. Esta arquitectura logró una tasa de error de tan solo el 7.2%. Tiene un total de 16 capas ocultas, lo que le da el nombre a la arquitectura.

2.6.1.3 VGG-19

La arquitectua VGG19 es una variación de la arquitectura VGG-16. La principal diferencia entre ellas es que a partir del bloque convolucional de 56x56x256, todos los bloques tienen una capa convolucional adicional.

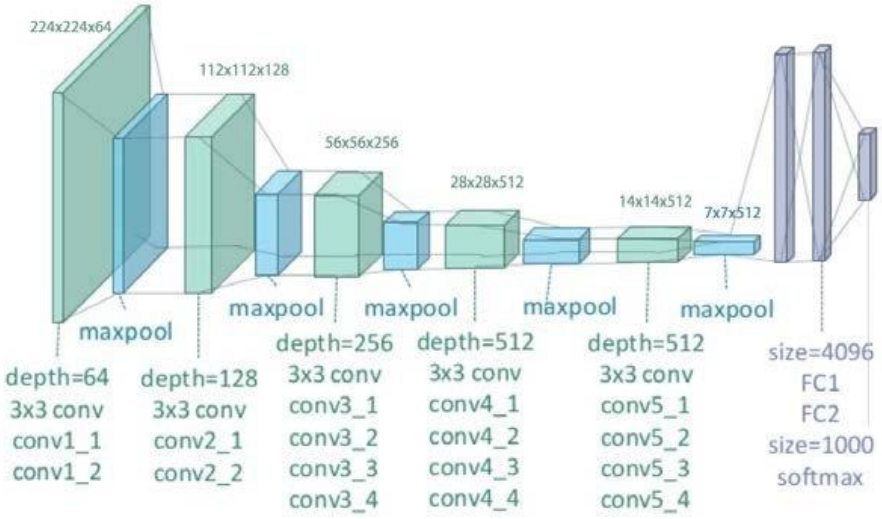

Figura 12. Arquitectura red VGG-19

2.6.2 Transfer Learning

Transfer Learning es un problema de investigación que se centra en almacenar el conocimiento generado por un modelo de aprendizaje automático a la hora de resolver un problema complejo y poder utilizar ese conocimiento para resolver otros problemas similares. Por ejemplo, el conocimiento aprendido por un modelo de inteligencia artificial a la hora de reconocer coches, podría ser utilizado del mismo modo para reconocer camiones.

Su aplicación es muy popular cuando se habla de Deep learning, donde modelos preentrenados son utilizados como punto de partida a la hora de comenzar el aprendizaje de un nuevo problema. Para ello, están disponibles, por ejemplo, los modelos utilizados y entrenados en la competición de imagenet.

2.6.3 Tecnologías

La popularización del aprendizaje automático y del campo de la inteligencia artificial ha venido inherentemente con el desarrollo de múltiples tecnoligías que ayudan a los desarrolladores e investigadores a implementar con mayor facilidad sus trabajos de predicciones y clasificaciones.

2.6.3.1 Tensorflow

Tensorflow es una plataforma desarrollada por Goole de código abierto de extremo a extremo para el aprendizaje automático. Cuenta con un ecosistema integral y flexible de herramientas, bibliotecas y recursos de la comunidad que permite que los investigadores innoven con el aprendizaje automático y los desarrolladores creen e implementen aplicaciones con esta tecnología muy fácilmente.

Esta tecnología está disponible para el lenguaje de programación de propósito general Python, que gracias a su enorme comunidad y soporte, posee librerías complementarias para todo tipo de utilidades matemáticas, estadísticas y de inteligencia artificial persé.

Tensorflow permite implementar de forma muy sencilla algunas arquitecturas de Deep learning ya existentes entre las que se encuentran ResNet50 y VGG-16. Así mismo, da soporte para crear nuevas arquitecturas de Deep learning, permitiendo añadir capas de todo tipo a nuestra red neuronal con una sola línea de código.

Además, tensorflow incluye desde 2017 la API **Keras**, especialmente pensada para la implementación de redes neuronales convolucionales y facilita en gran medida el uso de Deep learning con imágenes.

Como es bien sabido, debido tanto al inmenso número de operaciones simples que se deben llevar a cabo a la hora de entrenar y predecir utilizando Deep learning, como a la ingente cantidad de información que se debe cargar en memoria, las tarjetas gráficas son las grandes aliadas a la hora de realizar esta tarea. Tensorflow admite compatibilidad con las tarjetas gráficas de la marca NVidia, permitiendo utilizar su capacidad a la hora de realizar las actividades de entrenamiento y validación.

2.6.3.2 Open-CV

Open-CV (**Open C**omputer **V**ision) es una librería especializada en visión artificial desarrollada por Intel. Aunque fue desarrollada en C++, actualmente también está disponible para su uso en lenguaje Python. Al ser una librería especializada en el tratamiento de imágenes, ofrece un gran número de funcionalidades para tratarlas, por lo que se convierte en una herramienta muy útil para llevar a cabo el preprocesamiento de las imágenes del conjunto de datos de entrada.

Capítulo 3. Desarrollo del proyecto

El desarrollo del proyecto se ha realizado en su totalidad siguiendo una estrategia específica basada en el perfil de la persona encargada de realizarlo. Para ello, es necesario detallar en primer lugar el perfil.

La persona que realizará el proyecto es un estudiante del Máster universitario en Big Data Analytics que ha superado todos los créditos del curso a excepción del TFM y que previamente completó un Grado en Ingeniería Informática en el que cursó algunas asignaturas relacionadas con la investigación y la inteligencia artificial. Sin embargo, en ningún punto de su carrera ha realizado ningún proyecto relacionado con Deep Learning. Por otro lado, trabaja a tiempo completo como desarrollador web, por lo que conoce varios lenguajes de programación como Java, C++, PHP o Python entre otros.

3.1 Investigación

Para comenzar a desarrollar el proyecto, dado que se parte desde una base de conocimiento superficial y sin conocimiento explícito del mundo del Deep learning ni del tratamiento programático de imágenes, la primera fase de la estratégia ha sido realizar una investigación completa y exhaustiva del ámbito del Deep learning, tanto teórica como práctica (a nivel tecnológico). Esta fase tuvo una duración aproximada de tres semanas en las que se investigó en internet, libros y artículos científicos.

La investigación fue bastante fructífera, ya que no solo sirvió para afianzar los conocimientos con los que se partía, sino que también sirvió para proporcionar una buena imagen periférica de la situación actual del campo del aprendizaje profundo. Toda la información recolectada durante el proceso de investigación se encuentra resumida en *Capítulo 2. Conceptos técnicos y estado del arte.*

Tras haber realizado un proceso de investigación exhaustiva, finalmente la tecnología seleccionada para implementar toda la parte técnica del reto, así como para realizar todas las pruebas ha sido el lenguaje de programación Python. Esta decisión se sustenta en que el lenguaje es uno de los que tiene mayor soporte por parte de su comunidad, una buena parte de la cual es dedicada a la ciencia de datos. Por tanto, Python es un lenguaje ideal para realizar todas las pruebas y poder presentar un modelo final.

Para realizar el tratamiento de las imágenes, se va a hacer uso de la librería OpenCV, una librería muy sencilla de usar desarrollada en C++ y con soporte para Python, que aporta muchísima versatilidad e infinidad de funcionalidades a la hora de tratar las imágenes.

Para desarrollar las arquitecturas de Deep learning y representar los resultados, se ha utilizado la librería Tensorflow. Tensorflow está implementado en Python y en C++ y permite implementar con gran versatilidad y simpleza modelos complejos de Deep learning. Además, incluye la libería Keras, especializada en el desarrollo de arquitecturas de Deep learning para imágenes. Además, Tensorflow ha sido implementado teniendo en mente la arquitectura CUDA, por lo que existe la posibilidad de cargar el modelo y ejecutarlo directamente en una GPU de Nvidia.

3.2 Desarrollo y pruebas

Para comenzar con el apartado de desarrollo, lo primero que se ha de hacer es analizar los datos. Hay que comprobar de que tipo son, que tamaño tienen, cuantas clases hay y cuales son sus características.

El dataset con el que se ha de trabajar consiste en un con junto de imágenes de 23 marcas diferentes entre las que se encuentran AMD, Apple, Aquafina, Disney, D-Link, Domino's Pizza, Hellsmann's, IBM, Kitkat, LG, Lipton, McDonalds, Milka, Monter, Nestea, Pac-Man, Pepsi, Pizza Hut, Red Bull, Samsung, Sony, Tic-tac y Universal. De estas 23 imágenes tenemos un total de 1611 imágenes en total, teniendo 70 imágenes de cada marca y 71 de una de ellas. Por tanto, podría decirse que el conjunto de datos está bastante balanceado. Antes de hacer nada, es necesario separar el conjunto de datos en dos subconjuntos, uno para entrenamiento y otro para testing. Como los conjuntos de datos ya están completamente balanceados en el número de imágenes, he decidido que la mejor opción será utilizar un 70% de las imágenes de cada clase para entrenar el modelo y, del mismo modo, un 30% de las imágenes de cada clase para validar el modelo. De este modo nos aseguramos que al validar el modelo obtengamos una imagen clara de como se comportará con cada marca.

Para comenzar a desarrollar un modelo, la primera aproximación debería ser comprobar como se comportará con el conjunto de datosen crudo, sin hacerle mayor preprocesado que redimensionar las imágenes para que se ajusten al tamaño de entrada requerido por el modelo. El modelo utilizado en esta primera fase de la experimentación, será un modelo hecho a mano imitando la arquitectura de VGG-16, formada por 4 bloques de dos capas convolucionales y una capa de maxpooling, y un bloque de tres capas densas y una capa de

aplanado. Para la ejecución se utiliza de igual modo un **Earlystopper** con la paciencia establecida a 5, con la intención de detener el entrenamiento en si por muchas épocas que llevase a cabo no se consigue mejorar la eficacia del modelo. Que tenga la paciencia establecida a 5 significa que si tras 5 épocas de entrenamiento el modelo no es capaz de mejorar su desempeño, la ejecución finalizará.

Como era de esperar, los resultados obtenidos fueron bastante malos, obteniendo un accuracy de a penas un 32.21% durante el entrenamiento con el conjunto de validación.

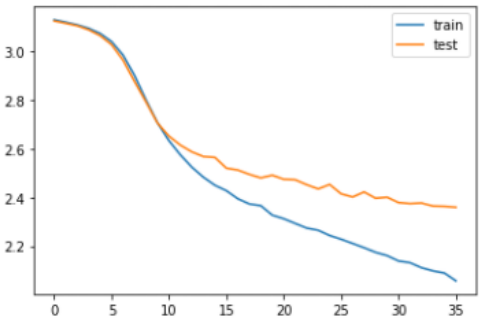

Figura 14. Grafica loss entrenamiento 1

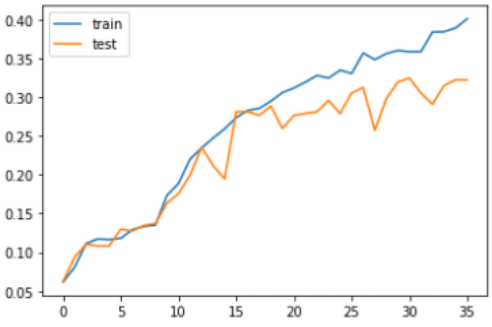

Figura 13. Grafica accuracy entrenamiento 1

```
74/74 [==============================] - 28s 374ms/step - loss: 2.1113 - accuracy: 0.3840 - val_loss: 2.3771 - val_accuracy: 0.2909
Epoch 34/120
74/74 [==============================] - 28s 374ms/step - loss: 2.0983 - accuracy: 0.3843 - val_loss: 2.3648 - val_accuracy: 0.3149
Epoch 35/120
74/74 [==============================] - 27s 367ms/step - loss: 2.0894 - accuracy: 0.3891 - val_loss: 2.3635 - val_accuracy: 0.3221
Epoch 36/120
74/74 [==============================] - 27s 367ms/step - loss: 2.0574 - accuracy: 0.4010 - val_loss: 2.3596 - val_accuracy: 0.3221
Epoch 00036: early stopping
```

Figura 15. Resultados entrenamiento 1.

Sin embargo, podemos encontrar algo interesante, ya que como se ve en la NOTA figura X, durante el entrenamiento el modelo obtuvo un accuracy de validación del 32.21%. Sin embargo, también realicé un proceso de validación manual con esos mismos datos para obtener la matriz de confusión, sin embargo, los datos fueron bastante diferentes.

accuracy			0.2895	418
macro avg	0.2392	0.2927	0.2529	418
weighted avg	0.2376	0.2895	0.2510	418

Figura 16. Resultados validación 1

El accuracy obtenido durante la validación fue de un 28.95%, un 4% menos que durante el entrenamiento, a pesar de haber utilizado el mismo modelo y los mismos datos de validación. Tras una pequeña investigación, llegué a la

conclusión de que obtener valores diferentes en las métricas entrenamiento y en las métricas de la evaluación del modelo es normal debido a que se comportan de forma totalmente distinta. Entrenando el modelo, se utiliza un tamaño de bolsa específico para ir mandando las imágenes al modelo y operar con ellas, de modo que el valor resultante es una media de todos los valores obtenidos en bolsa. Sin embargo, esto no sucede en la evaluación final manual realizada, por lo que inevitablemente los valores obtenidos serán inevitablemente distintos.

La realidad es que hay muy pocos datos existentes para tantas marcas distintas, no son suficientes imágenes para que la red pueda diferenciarlas. Sin embargo, en el reto se prohíbe explícitamente nutrir el dataset con imágenes que no hayan sido proporcionadas por IBM, pero sí está permitido utilizar las imágenes proporcionadas para aumentar el dataset. Por tanto, el siguiente paso de las pruebas será comprobar el impacto de aumentar el número de imágenes de entrenamiento.

Tras investigar como poder solventar este problema, se llegó a la conclusión de que la mejor opción era realizar un proceso llamado **Data Augmentation**. Este prepocesado consiste en tomar cada imagen del dataset y aplicarle una serie de modificaciones simples para aumentar el tamaño del dataset. Para la siguiente prueba obtendremos 4 imágenes por cada imganen original, y se harán transformaciones que permitan no distorsionar el formato de la imagen de la marca. Para realizarlo, se utilizarán en conjunto las librerías de Keras y de Open-CV.

Los resultados obtenidos al realizar las pruebas con el mismo modelo, pero utilizando los datos modificados mediante el proceso de Data Augmentation, podemos ver que efectivamente la falta de datos era parte del problema, pues los resultados mejoran en gran medida. En esta ocasión los resultados obtenidos durante las validaciones del entrenamiento son significativamente mejores, alcanzando un accuracy global del 55.05%.

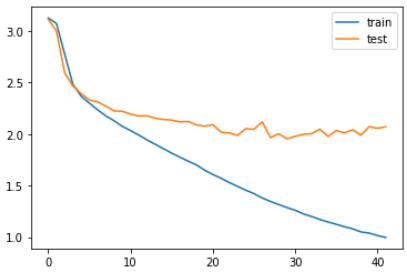

Figura 18. Gráfica loss entrenamiento 2

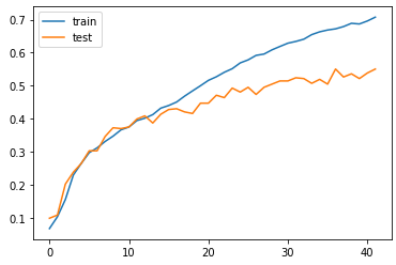

Figura 17. Grafica accuracy entrenamiento 2

```
373/373 [==============================] - 107s 286ms/step - loss: 1.0506 - accuracy: 0.6890 - val_loss: 1.9894 - val_accuracy: 0.5361
Epoch 40/120
373/373 [==============================] - 107s 287ms/step - loss: 1.0395 - accuracy: 0.6870 - val_loss: 2.0729 - val_accuracy: 0.5216
Epoch 41/120
373/373 [==============================] - 107s 288ms/step - loss: 1.0153 - accuracy: 0.6961 - val_loss: 2.0556 - val_accuracy: 0.5385
Epoch 42/120
373/373 [==============================] - 107s 288ms/step - loss: 0.9970 - accuracy: 0.7077 - val_loss: 2.0705 - val_accuracy: 0.5505
Epoch 00042: early stopping
```

Figura 19. Resultado entrenamiento 2

Por otro lado, los resultados obtenidos en el proceso de validación fueron del mimso modo significativamente mejores, logrando mantener el accuracy global del 55.05%. Sin embargo, los valores obtenidos para las métricas específicas siguen sin ser nada buenos, pues parece haber algunos logos de marcas con los que tiene dificultades para acertar, como por ejemplo el de disney.

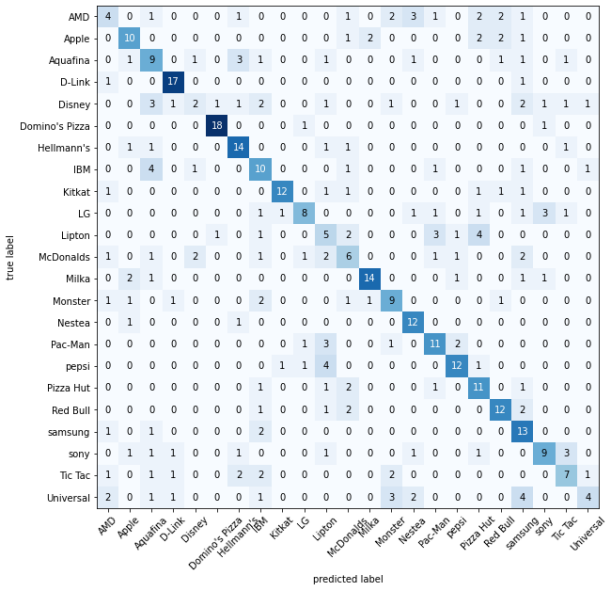

Figura 21. Matriz de confusión exp 2

accuracy			0.5505	416
macro avg	0.5549	0.5500	0.5389	416
weighted avg	0.5588	0.5505	0.5414	416

Figura 20. Resultados validación exp 2

Todas estas pruebas han sido realizadas con una implementación aproximada de la arquitectura VGG-16. Sin embargo, esta arquitectura viene implementada para trabajar con ella en la librería Keras, por tanto, es posible que dicha implementación consiga alcanzar un desempeño mayor a la hora de realizar las predicciones. Por tanto, entrenamos el modelo VGG-16 del paquete tensorflow.keras.applications.vgg16. Este módulo incluye numerosas configuraciones y variaciones, pero para esta prueba vamos a dejar la configuración por defecto. Sin embargo, en lugar de utilizar el parámetro especificado para seleccionar el número de clases a predecir, he optado por agregar una capa densa extra y utilizarla como capa de salida.

Los resultado obtenidos en esta nueva experimentación han sido, de nuevo, mejores que los obtenidos en el anterior experimento. La utilización del modelo VGG-16 junto con las capas de salida extras, han logrado obtener un valor de accuracy durante la validación del entrenamiento de 81.73%, lo que supone una diferencia considerable.

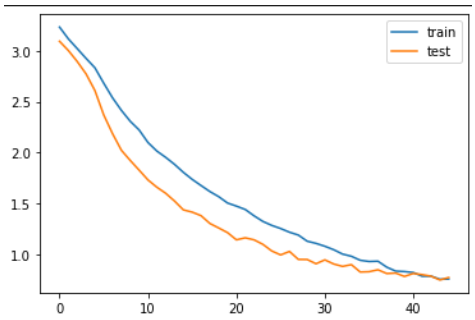

Figura 23. Grafica loss entrenamiento 3

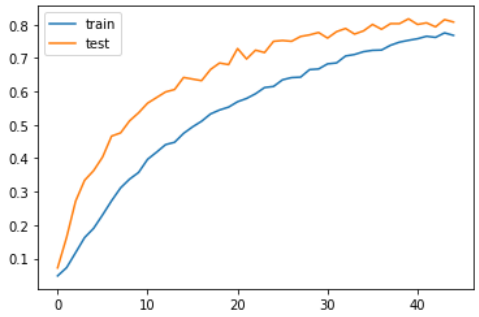

Figura 22. Grafica accuracy entrenamiento 3

```
186/186 [==============================] - 111s 593ms/step - loss: 0.8291 - accuracy: 0.7530 - val_loss: 0.7810 - val_accuracy: 0.8173
Epoch 41/100
186/186 [==============================] - 113s 603ms/step - loss: 0.8197 - accuracy: 0.7577 - val_loss: 0.8103 - val_accuracy: 0.8005
Epoch 42/100
186/186 [==============================] - 112s 601ms/step - loss: 0.7832 - accuracy: 0.7651 - val_loss: 0.7991 - val_accuracy: 0.8053
Epoch 43/100
186/186 [==============================] - 111s 593ms/step - loss: 0.7841 - accuracy: 0.7617 - val_loss: 0.7836 - val_accuracy: 0.7933
Epoch 44/100
186/186 [==============================] - 112s 597ms/step - loss: 0.7536 - accuracy: 0.7752 - val_loss: 0.7465 - val_accuracy: 0.8149
Epoch 45/100
186/186 [==============================] - 110s 592ms/step - loss: 0.7557 - accuracy: 0.7678 - val_loss: 0.7704 - val_accuracy: 0.8077
Epoch 00045: early stopping
```

Figura 24. Resultados entrenamiento 3

Sin embargo, este valor difiere en gran medida con el obtenido en las métricas de la evaluación, que aun así sigue mejorando los resultados del experimento previo, con un accuracy de 67.07%.

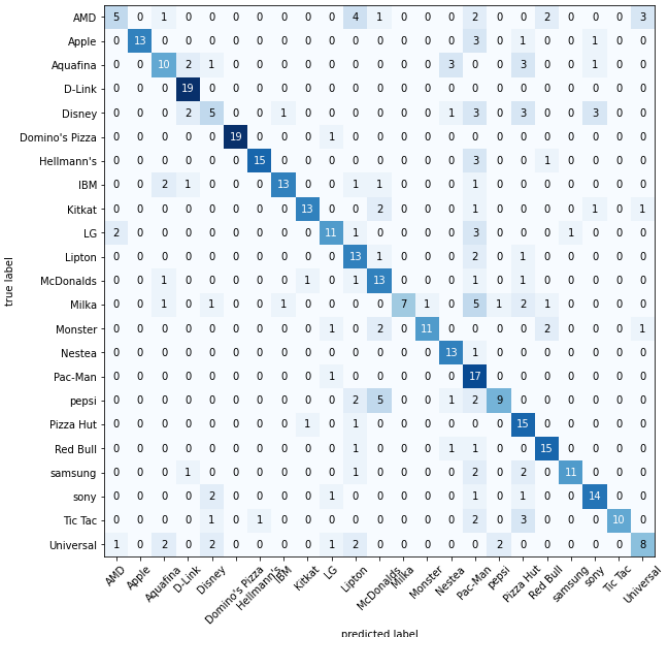

Figura 25. Matriz confusión exp 3

accuracy			0.6707	416
macro avg	0.7329	0.6738	0.6717	416
weighted avg	0.7358	0.6707	0.6714	416

Figura 26. Resultados exp 3

Tras haber probado exhaustivamente como se comporta este modelo siendo entrenado desde 0, vemos que tiene un comportamiento aceptable para la cantidad de datos de la que disponemos en relación a la complejidad del problema, aunque no deja de estar alejado de lo ideal. La siguiente prueba realizada consistirá en utilizar **Transfer Learning**. Para ello, utilizaremos los pesos del modelo VGG-16 entrenada con los datos de imagenet. Esta configuración está disponible al incluir el parámetro *weights='imagenet'.* Al ejecutar este comando, tensorflow descargará automáticamente los pesos de la red entrenada y la cargará. Existe también la posibilidad de seleccionar qué capas se quieren ajustar durante el entrenamiento, sin embargo, para esta prueba voy a dejar todas las capas entrenables.

Los resultados obtenidos en esta prueba han sido muy similares a los obtenidos en la experimentación anterior, no parece haber conseguido un gran cambio, pues, a pesar de haber mejorado le desempeño durante el entrenamiento, este no ha sido el caso en la validación.

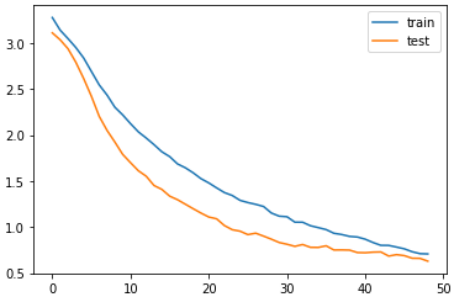

Figura 27. Grafia loss entrenamiento 4

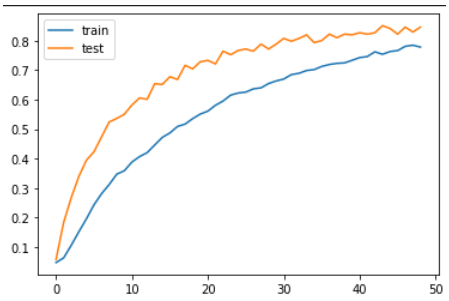

Figura 28. Grafica accuracy entrenamiento 4

```
186/186 [==============================] - 117s 627ms/step - loss: 0.8001 - accuracy: 0.7538 - val_loss: 0.6842 - val_accuracy: 0.8510
Epoch 45/100
186/186 [==============================] - 116s 620ms/step - loss: 0.7819 - accuracy: 0.7631 - val_loss: 0.6995 - val_accuracy: 0.8413
Epoch 46/100
186/186 [==============================] - 115s 618ms/step - loss: 0.7628 - accuracy: 0.7664 - val_loss: 0.6890 - val_accuracy: 0.8221
Epoch 47/100
186/186 [==============================] - 115s 615ms/step - loss: 0.7312 - accuracy: 0.7808 - val_loss: 0.6606 - val_accuracy: 0.8462
Epoch 48/100
186/186 [==============================] - 116s 621ms/step - loss: 0.7103 - accuracy: 0.7846 - val_loss: 0.6592 - val_accuracy: 0.8293
Epoch 49/100
186/186 [==============================] - 115s 617ms/step - loss: 0.7076 - accuracy: 0.7782 - val_loss: 0.6289 - val_accuracy: 0.8462
Epoch 00049: early stopping
```

Figura 30. Resultados entrenamiento 4

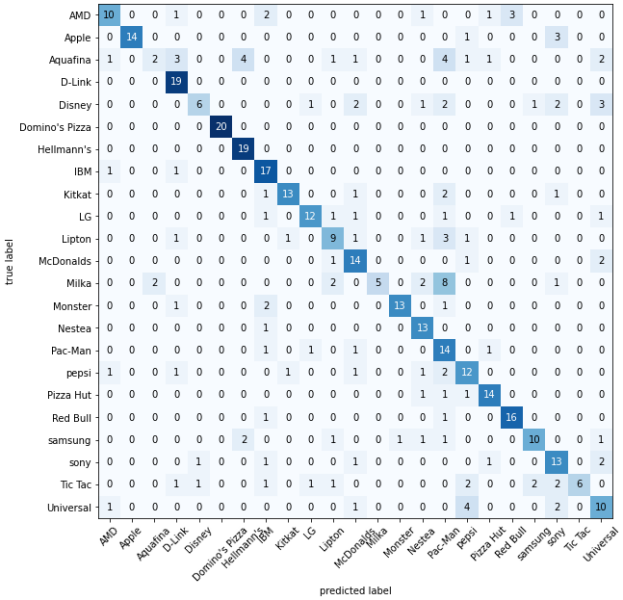

Figura 29. Matriz confusión exp 4

accuracy			0.6755	416
macro avg	0.7220	0.6784	0.6624	416
weighted avg	0.7224	0.6755	0.6601	416

Figura 31. Resultados exp 4

Pareciendo haber llegado a un camino sin salida, tomé la elección de utilizar otro modelo para comprobar su desempeño. En este caso, decidí probar el modelo basado en redes residuales *RestNet50*, que también está disponible como módulo de Keras. Esta arquitectura no se adaptó especialmente bien al problema, puesto que la ejecución de las predicciones para calcular la matriz de confusión y las métricas específicas tuvieron un resultado extraño. Pese a que su valor de accuracy y de loss obtenido durante el entrenamiento fue 0.5289 y 87.25% respectivamente, al evaluar las predicciones del conjunto de test obtuve un valor de accuracy de 3.48%. La explicación de este valor puede verse claramente al observar la matriz de confusión, puesto que el modelo está prediciendo casi todas las instancias como la misma clase de forma errónea.

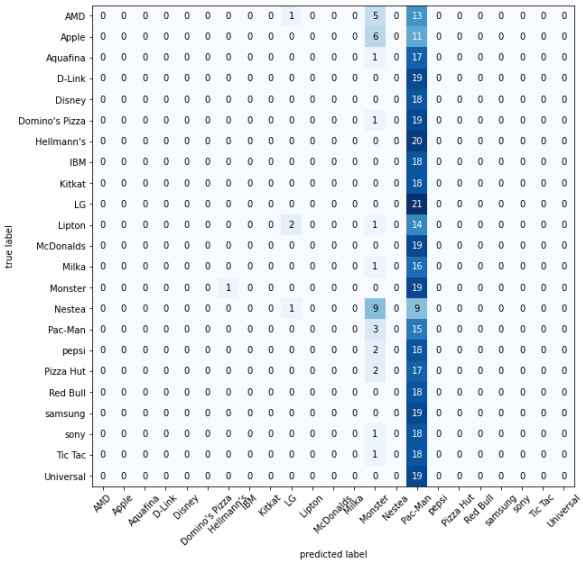

Figura 32. Matriz confusion exp 5

accuracy			0.0348
macro avg	0.0017	0.0362	0.0032
weighted avg	0.0016	0.0348	0.0030

Figura 33. Resultados exp 5

Tras una búsqueda exhaustiva por internet, no pude encontrar una respuesta concluyente a lo que sucedía, por lo que decidí probar con distintos modelos implementados en keras basados en redes residuales. Los modelos testeados fueron **ResNet152V2** y **ResNet50V2**, pero el resultado fue el mismo. Tanto para los mismos datos utilizados con el modelo VGG-16 como para distintos aumentos y modificaciones de preprocesado de Data Augmentation, la salida del modelo era siempre la misma, predecir prácticamente todos los ejemplos como la misma clase. Por este motivo, decidí que para este problema las arquitecturas basadas en redes residuales no se ajustaban correctamente.

Tras esta accidentada parte de la experimentación, pero necesaria para explicar el motivo de excluir las arquitecturas de redes residuales, continué probando otros modelos. En este caso el modelo testeado fue la arquitectura VGG-19, una variante de la arquitectura VGG-19, solo que se diferencia en que implementa más capas que esta.

Aunque este modelo realizó unas predicciones bastante más acertadas que las arquitecturas residuales, de nuevo, empeoró el resultado de VGG-16. Sus limitaciones pueden verse reflejadas perfectamente en las gráficas obtenidas en el entrenamiento, puesto que se percibe que a partir de cierto nivel de precisión muy limitado es incapaz de mejorar el rendimiento obtenido durante el entrenamiento y termina desencadenando en un alto **overfitting**.

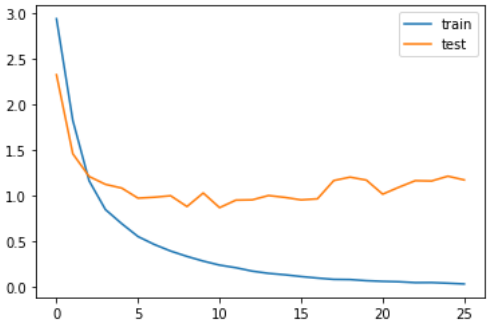

Figura 35. Grafica loss entrenamiento 6

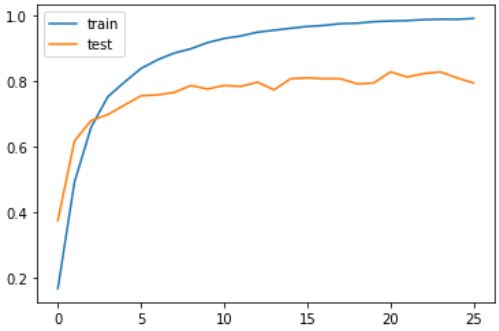

Figura 34. Grafica accuracy entrenamiento 6

El resultado de las métricas calculadas a partir de las predicciones fue igualmente malo, igualando el valor del segundo experimento con un accuracy del 54.26%

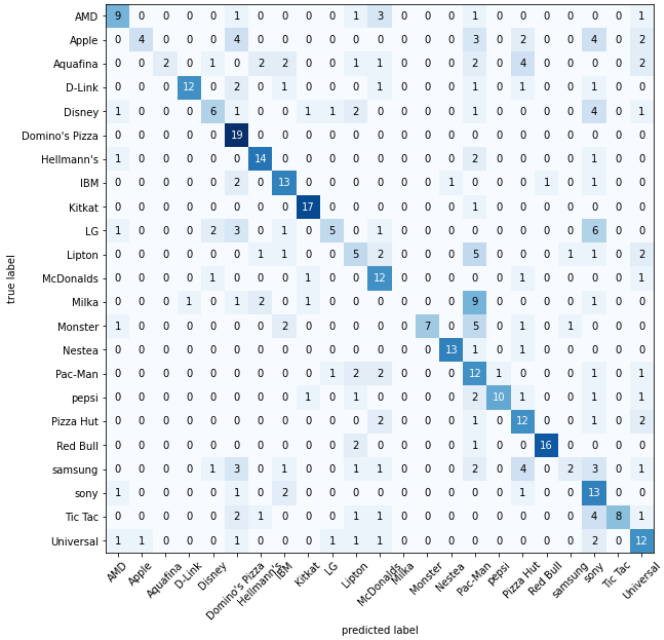

Figura 36. Matriz confusion exp 6

accuracy			0.5426
macro avg	0.6293	0.5408	0.5244
weighted avg	0.6287	0.5426	0.5243

Figura 37. Resultado exp 6

Dado que hasta el momento el único cambio que ha conseguido mejorar el desempeño del modelo ha sido utilizar la técnica de preprocesado Data Augmentation, la próxima prueba consistirá en aumentar el número de imágenes y transformaciones por imagen. En esta prueba, el número de imágenes aumentadas ha sido 10 por cada imagen.

Por desgracia, el aumentar tanto el número de imágenes solo ha conseguido ralentizar el aprendizaje del modelo. Los resultados obtenidos han sido prácticamente iguales a las mejores pruebas con VGG-16. Presento a continuación los resultados de entrenamiento:

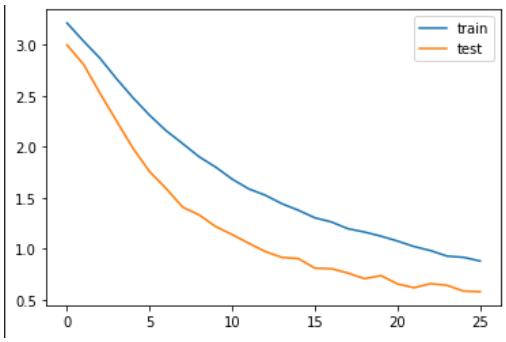

Figura 39. Grafia loss entrenamiento 7

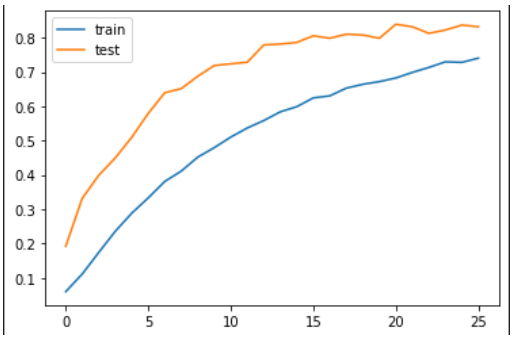

Figura 40. Grafica accuracy entrenamiento 7

```
368/368 [==============================] - 225s 610ms/step - loss: 1.0747 - accuracy: 0.6824 - val_loss: 0.6536 - val_accuracy: 0.8389
Epoch 22/100
368/368 [==============================] - 221s 600ms/step - loss: 1.0209 - accuracy: 0.6987 - val_loss: 0.6172 - val_accuracy: 0.8317
Epoch 23/100
368/368 [==============================] - 222s 603ms/step - loss: 0.9807 - accuracy: 0.7131 - val_loss: 0.6571 - val_accuracy: 0.8125
Epoch 24/100
368/368 [==============================] - 222s 601ms/step - loss: 0.9268 - accuracy: 0.7295 - val_loss: 0.6403 - val_accuracy: 0.8221
Epoch 25/100
368/368 [==============================] - 224s 608ms/step - loss: 0.9149 - accuracy: 0.7280 - val_loss: 0.5836 - val_accuracy: 0.8365
Epoch 26/100
368/368 [==============================] - 236s 639ms/step - loss: 0.8788 - accuracy: 0.7402 - val_loss: 0.5782 - val_accuracy: 0.8317
Epoch 00026: early stopping
```

Figura 38. Resultado entrenamiento 7

Y los resultados de validación:

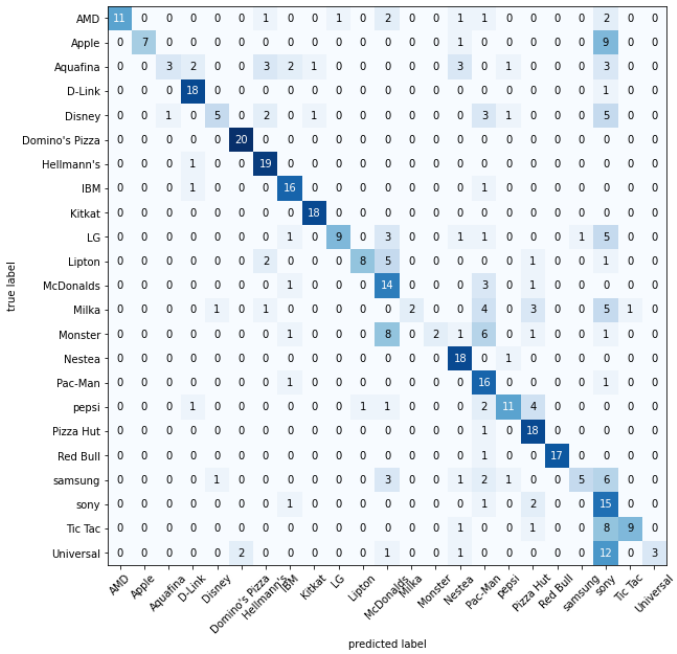

Figura 41. Matriz confusion exp 7

accuracy			0.6125
macro avg	0.7770	0.6103	0.5931
weighted avg	0.7760	0.6125	0.5944

Figura 42. Resultado exp 7

Tras estas pruebas y revisando todas las gráficas he llegado a la conclusión de que posiblemente la paciencia establecida para el mecanismo de Early Stopping sea demasiado pequeña para dejar converjer al modelo completamente. Por tanto la siguiente prueba consistirá en modificar dicho parámetro para comprobar el efecto de utilizar este parámetro mayor o menor. De nuevo, se utilizará un Data Augmentation de 5, puesto que según se ha observado en las pruebas, es un número que retorna un buen desempeño sin desorbitar el coste computacional del entrenamiento.

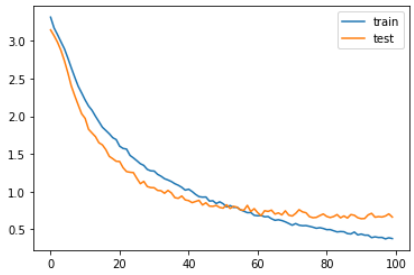

Figura 44. Grafica loss entrenamiento 8

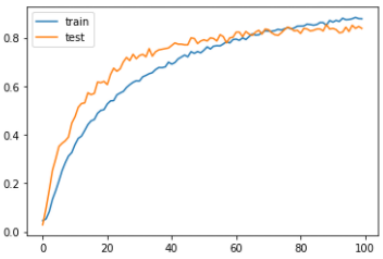

Figura 43. Grafica accuracy entrenamiento 8

```
Epoch 96/100
187/187 [==============================] - 115s 614ms/step - loss: 0.3912 - accuracy: 0.8752 - val_loss: 0.6683 - val_accuracy: 0.8255
Epoch 97/100
187/187 [==============================] - 114s 605ms/step - loss: 0.3922 - accuracy: 0.8784 - val_loss: 0.6629 - val_accuracy: 0.8516
Epoch 98/100
187/187 [==============================] - 113s 600ms/step - loss: 0.3727 - accuracy: 0.8839 - val_loss: 0.6764 - val_accuracy: 0.8385
Epoch 99/100
187/187 [==============================] - 115s 611ms/step - loss: 0.3874 - accuracy: 0.8789 - val_loss: 0.7040 - val_accuracy: 0.8464
Epoch 100/100
187/187 [==============================] - 114s 608ms/step - loss: 0.3770 - accuracy: 0.8782 - val_loss: 0.6624 - val_accuracy: 0.8385
```

Figura 45. Resultado entrenamiento 8

Efectivamente el aumentar el número parece haber conseguido reducir el underfitting. De este modo, se han conseguido también mejores resultados en las métricas de evaluación del modelo.

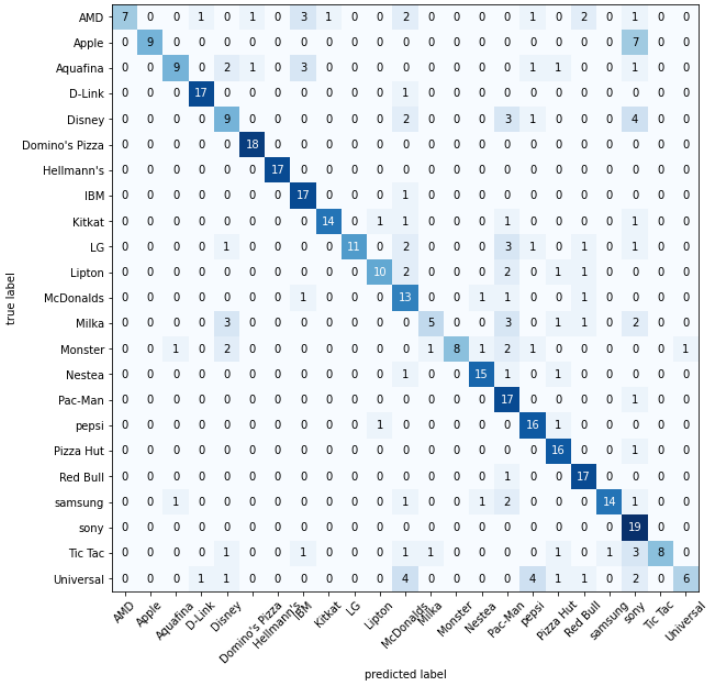

Figura 46. Matriz confusion exp 8

accuracy			0.7105
macro avg	0.7930	0.7103	0.7053
weighted avg	0.7928	0.7105	0.7048

Figura 47. Resultado exp 8

Capítulo 4. Conclusiones

A lo largo de este reto he podido descubrir el maravilloso mundo de las redes neuronales y el Deep learning, habiendo llegado incluso a trabajar en un problema de de clasificación completo como primera toma de contacto. A pesar de no conocer nada de este mundillo, durante el proyecto he aprendido infinidad de conceptos y aplicaciones de este campo de la inteligencia artificial, desde medidas de evaluación hasta arquitecturas de modelos.

Aunque considero haber realizado un trabajo muy completo, creo que hay muchos aspectos por mejorar y técnicas que no he descubierto (aún) ni se me han ocurrido aplicar. Y, aunque no haya conseguido un resultado expléndido, estoy convencido de que he hecho un buen trabajo en el reto.

Siguen quedando muchas aplicaciones y puntos de mejora que serían posible implementar y probar disponiendo de más tiempo o de un equipo más potente que mi ordenador personal con una tarjeta gráfica de hace más de 5 años. No obstante, he disfrutado y aprovechado cada equivocación y cada cambio que no suponía un avance en el proyecto, ya que ello me abría la puerta a estudiar y explorar nuevos caminos, de igual modo que todo lo aprendido.

ANEXOS

Anexo I: División del cunjunto de datos

Script en Python utilizado para separar el conjunto de datos.

```
import numpy as np
import shutil
import os

path = 'C:/Users/Carlos/Desktop/Master/TFM/Datasets/Original'
w_path = 'C:/Users/Carlos/Desktop/Master/TFM/Datasets/test/'
dir_list = os.listdir(path)

os.mkdir(w_path)
for dir in dir_list:
    try:
        os.mkdir(w_path + dir)
    except:
        print("Error")

    for image_dir in dir_list:
        img_list = os.listdir(path + "/" + image_dir)
        num_images = int(
            len(img_list) * .3)  # take 30% of imgs to have stratified
prediction and training set with 70/30 proportion
        random_files = np.random.choice(img_list, num_images)
        for img in random_files:
            shutil.copyfile(path + "/" + image_dir + "/" + img,
                            w_path + "/" + image_dir + "/" + img)  # copy
file for test set
        for img in random_files:
            try:
                os.remove(path + "/" + image_dir + "/" + img)  # copy
file for test set
            except:
                ""
```

Anexo II: Data Augmentation

Scrypt de Python utilizado para el preprocesado de imágenes.

```
import os
os.add_dll_directory("C:/Program Files/NVIDIA GPU Computing
Toolkit/CUDA/v11.2/bin")
import numpy as np
import cv2
from keras.preprocessing.image import ImageDataGenerator, load_img, image

MAIN_FOLDER='C:/Users/Carlos/Desktop/Master/TFM/Datasets/'
images_increased = 5

try:
    os.mkdir(MAIN_FOLDER + "train")
except:
    print("Error")

data_path= MAIN_FOLDER + "Original"
data_dir_list = os.listdir(data_path)

datagen = ImageDataGenerator(
    rotation_range=10,
    zoom_range=0.2,
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=True,
    vertical_flip=True
)
width_shape, height_shape = 224, 244
i=0
num_images=0
for image_dir in data_dir_list:
    img_list=os.listdir(data_path + '/' + image_dir)
    try:
        os.mkdir(MAIN_FOLDER + "train/" + image_dir)
    except:
        print("Error")

    for image_file in img_list:
        img_path = data_path + '/'+ image_dir + '/' + image_file

        imge=load_img(img_path)

        imge=cv2.resize(image.img_to_array(imge), (width_shape,
height_shape), interpolation = cv2.INTER_AREA)
        x= imge/255
        x=np.expand_dims(x,axis=0)
        t=1
```

```
        for output_batch in datagen.flow(x,batch_size=1):
            a=image.img_to_array(output_batch[0])
            imagen=output_batch[0,:,:]*255
            imgfinal = cv2.cvtColor(imagen, cv2.COLOR_BGR2RGB)
            cv2.imwrite(MAIN_FOLDER + "train/" + image_dir + "/%i%i.jpg"
%(i,t) , imgfinal)
            t+=1

            num_images+=1
            if t>images_increased:
                break
        i+=1
    print("Done with " + image_dir)

print("images generated",num_images)
```

BIBLIOGRAFÍA

Koziarski, Michał & Cyganek, Boguslaw. (2017). Image recognition with deep neural networks in presence of noise – Dealing with and taking advantage of distortions. Integrated Computer-Aided Engineering. 24. 1-13. 10.3233/ICA-170551.

Bishop, C. C. N. O. M. (1996). *Neural Networks for Pattern Recognition* (1.ª ed.). Clarendon Press.

Brownlee, J. (2019, 16 septiembre). *A Gentle Introduction to Transfer Learning for Deep Learning*. Machine Learning Mastery. https://machinelearningmastery.com/transfer-learning-for-deep-learning/#:%7E:text=Transfer%20learning%20is%20a%20machine%20learning%20technique%20where%20a%20model,on%20a%20second%20related%20task.&text=%E2%80%94%20Page%20526%2C%20Deep%20Learning%2C,when%20modeling%20the%20second%20task.

Brownlee, J. (2021). *Machine Learning Mastery With Python: Understand Your Data, Create Accurate Models and Work Projects End-To-End*. Independently published.

C., & de Cienciacinetica, V. T. L. E. (2020, 22 junio). *Redes neuronales convolucionales en inteligencia artificial (CNN)*. Inteligenc[IA]. https://inteligencia.tech/2018/06/06/redes-convolutivas-en-inteligencia-artificial/

Calvo, D. (2018, 7 diciembre). *Función de activación – Redes neuronales*. Diego Calvo. https://www.diegocalvo.es/funcion-de-activacion-redes-neuronales/

Dwivedi, P. (2019, 27 marzo). *Understanding and Coding a ResNet in Keras - Towards Data Science*. Medium. https://towardsdatascience.com/understanding-and-coding-a-resnet-in-keras-446d7ff84d33

Hassan, M. U. (2021, 24 febrero). *VGG16 – Convolutional Network for Classification and Detection*. Neurohive. https://neurohive.io/en/popular-networks/vgg16/

ImageNet. (2012). Imagenet. https://www.image-net.org/

Juan Barrios. (2019, 18 junio). *Redes neuronales convolucionales son un tipo de redes neuronales*. https://www.juanbarrios.com/redes-neurales-convolucionales/#:%7E:text=Las%20Redes%20neuronales%20convolucionales%20son,V1)%20de%20un%20cerebro%20biol%C3%B3gico.

K. (2019, 2 mayo). *Inteligencia artificial: ¿Cómo aprenden las redes neuronales?* Microsoft Docs. https://docs.microsoft.com/es-es/archive/msdn-magazine/2019/april/artificially-intelligent-how-do-neural-networks-learn

Module: tf | TensorFlow Core v2.6.0. (2015). TensorFlow. https://www.tensorflow.org/api_docs/python/tf?hl=es-419

Saha, S. (2021, 17 agosto). *A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way*. Medium. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

Velasco, L. (2020, 4 septiembre). *Optimizadores en redes neuronales profundas: un enfoque práctico*. Medium. https://velascoluis.medium.com/optimizadores-en-redes-neuronales-profundas-un-enfoque-pr%C3%A1ctico-819b39a3eb5

Vickery, R. (2020, 27 mayo). *10 Lesser-Known Python Libraries for Machine Learning*. Medium. https://towardsdatascience.com/10-lesser-known-python-libraries-for-machine-learning-fca7ad32e53c