Facultad de Ciencias Empresariales

UTILIZACIÓN DE TÉCNICAS DE MACHINE LEARNING PARA MEJORAR LA DETECCIÓN DE INFRACCIÓN DE MARCAS

Clave: 201600243

Autor: Belén Armada Carrión

Director: Lucia Barcos Redín

MADRID | Junio 2021

RESUMEN Y PALABRAS CLAVE

Las redes neuronales convolucionales son un algoritmo de Deep Learning, rama dentro del campo de Machine Learning, que destaca especialmente por su capacidad para reconocer, clasificar y/o comparar imágenes. En este trabajo se ha examinado cómo este tipo de redes pueden utilizarse para la automatización de la detección de infracción de marcas. Para ello, se realiza un caso práctico en Python utilizando TensorFlow y Keras como vehículos, en el que se muestra el grado de similitud entre un producto de la página web de Nike de esta temporada y de Asos, en el que se detecta la infracción de marcas de esta última con respecto a la primera. Este caso práctico realizado a gran escala garantizaría la protección sistemática de la autenticidad de las marcas y diseños industriales, a la vez que reduciría enormemente el tiempo empleado actualmente en la detección de falsificaciones por parte de las propias empresas y de despachos de abogados que luchan por la protección de la propiedad industrial de sus clientes.

Palabras clave: redes neuronales convolucionales, infracción de marca, *Deep Learning*, *Machine Learning*, algoritmo, Python, reconocimiento de imágenes.

ABSTRACT AND KEYWORDS

Convolutional neural networks are a Deep Learning algorithm, a modality within the field of Machine Learning, which stands out especially for its ability to recognize, classify and/or compare images. This paper examines how this type of network can be used to automate the detection of trademark infringement. For this purpose, a case study is carried out in Python using TensorFlow and Keras as *frameworks*, showing the degree of similarity between a product on Nike's website this season and one of Asos, in which the latter's trademark infringement is detected with respect to the former's. This case study, carried out on a large scale, would guarantee the systematic protection of the authenticity of trademarks and industrial designs, while greatly reducing the time currently spent on detecting counterfeits by the companies themselves and by law firms fighting for the protection of their clients' industrial property.

Keywords: convolutional neural networks, trademark infringement, *Deep Learning*, *Machine Learning*, algorithm, Python, image recognition.

Índice general

Índice de figuras

Listado de abreviaturas

ANN	Redes Neuronales Artificiales
DNN	Red Neuronal Profunda
DBN	Red de Creencias Profundas
RNTN	Red de Tensor Neuronal Recursivo
RBM	Máquina de Boltzmann Restringida
GAN	Redes Generativas Antagónicas
RNN	Redes Neuronales Recurrentes
NLP	Procesamiento de Lenguaje Natural
LSTM	Redes de Memoria a Largo Plazo
CNN	Redes Neuronales Convolucionales
SVM	Support Vector Machines
PCA	Análisis de Componentes Principales
ReLU	Rectified Lineal Unit
ROC	Receiver Operating Characteristic
AUC	Area Under the ROC Curve
ILSVRC	Imagenet Large Scale Visual Recognition Challenge
TPU	Tensor Processing Unit
ASIC	Application-Specific Integrated Circuit
API	Interfaz de Uso Intuitivo

Introducción

1. Justificación

La infracción de marcas constituye un acto frecuente de competencia desleal entre empresas y su detección no es siempre del todo sencilla o evidente. Existen diversas formas de usurpación de una marca, de entre las que destacan la copia explícita del nombre comercial y la reproducción de una sutil modificación del logo o de diseños característicos de la empresa que se encuentren protegidos jurídicamente. Tales prácticas ilícitas por parte de los competidores pueden crear confusión entre los consumidores e inducirles erróneamente a comprar los productos imitados pensando que adquieren los verdaderos con un mejor precio o descuento. Por tanto, tales empresas defraudadoras estarían aprovechándose de la reputación ajena, a la vez que dañando la imagen y el prestigio reconocido de la empresa original, en el caso de que los productos resultaran ser de mala calidad, lo cual es muy habitual. En definitiva, la infracción de marcas constituye una violación de los derechos de propiedad industrial de la empresa.

Actualmente la detección de infracción de marcas, por parte de las empresas y de los despachos de abogados, que les ayudan en su defensa y requerimiento judicial, se realiza manualmente, por lo general gracias a la delación de un cliente, leal a la empresa, que se encuentra por azar con un producto imitado, tanto en tiendas físicas como online. Tal método es altamente ineficiente, puesto que muchas infracciones nunca llegan al conocimiento de los perjudicados y, en caso de llegar, el proceso de detección puede ser lento y su prueba muy tediosa.

2. Objetivo

Por ello, el objetivo de este trabajo es proponer un método más eficiente y automático, mediante el empleo de técnicas de Machine Learning, que ayude a las empresas a mejorar y acelerar la detección de las infracciones que sufran sus marcas online, para que así puedan minimizar el perjuicio económico y reputacional que conllevan.

3. Metodología

Lo que se pretende es elaborar un modelo basado principalmente en el reconocimiento de imágenes utilizando redes neuronales convolucionales siamesas. Las redes neuronales son uno de los tipos más populares de algoritmos de *Machine Learning*, óptimo para el análisis de imágenes, contrastándolas y devolviendo el porcentaje de similitud entre ellas.

En concreto, la metodología ideada consiste en la utilización de la técnica de Webscrapping de imágenes para descargar todas las imágenes de productos de la misma clase ofrecidos en páginas web competidoras y, posteriormente, aplicar el algoritmo de redes neuronales convolucionales para comparar cada una de ellas con cada una de las imágenes de los productos de "nuestro" cliente. El algoritmo irá devolviendo el grado de similitud entre cada una de ellas. Se realizaría el modelo íntegramente en Python, utilizando TensorFlow y Keras como vehículos. Adicionalmente, se programaría que, en caso de sobrepasarse un predeterminado porcentaje de similitud, salte un aviso alertando de una potencial infracción de marcas. Llegado ese punto, sería la primera vez que tendría que intervenir una persona humana para juzgar si es necesario denunciar la violación y enviar una demanda de requerimiento a la empresa infractora. Como se puede observar, se automatizaría la práctica totalidad del proceso de detección de marcas.

El análisis se realizará en Python. Dado que para realizar el modelo completo se necesita un ordenador con una alta capacidad de almacenamiento y procesamiento, en este trabajo se realizará el análisis únicamente comparando dos imágenes de productos competitivos, a modo de demostración.

Capítulo 1: Las técnicas de *Machine Learning* y su utilización en el campo de propiedad industrial

El aprendizaje automático (en inglés, *Machine Learning*) es una disciplina del campo de la Inteligencia Artificial cuyo objetivo es la creación, mediante algoritmos, de modelos capaces de aprender por si solos, identificando patrones en datos masivos.

1.1. Tipos de algoritmos de *Machine Learning*

Existen tres tipos diferentes de *Machine Learning* según su forma de aprender:

1.1.1 Aprendizaje supervisado

En el aprendizaje supervisado, los algoritmos trabajan con datos "etiquetados" (*labeled data)*, buscando alcanzar una función que asigne la etiqueta de salida adecuada a las variables proporcionadas de entrada (*input data*). Así pues, el algoritmo se entrena con un conjunto de datos históricos, y asigna la etiqueta de salida de un nuevo valor en función a los datos que ya conoce.

Según la autora Recuero de los Santos (2017), entre otros, el aprendizaje supervisado es común tanto en problemas de clasificación, donde la variable objetivo es de tipo categórico, como la identificación de dígitos o diagnósticos, como en problemas de regresión, cuya variable objetivo es numérica, lo cual ocurre en predicciones meteorológicas y de expectativa de vida, entre otros.

Los algoritmos más utilizados como medio en el aprendizaje supervisado son:

1. **Árboles de decisión:** su objetivo es modelar regresiones y clasificaciones. Según Microsoft (2018), para los atributos discretos, el algoritmo hace predicciones basándose en las relaciones entre las columnas de entrada de un conjunto de datos.

2. **Clasificación de Naïve Bayes:** se utiliza el Teorema de Bayes de probabilidad para construir modelos predictivos.
3. **Regresión por mínimos cuadrados:** según el autor Molina (2020), se utiliza para calcular la recta de regresión lineal que minimiza las diferencias entre los valores reales y los estimados por la recta.
4. **Regresión Logística:** se usa para predecir un resultado en función de unos atributos independientes de entrada.
5. **Support Vector Machines (SVM):** se usan en problemas de clasificación y regresión, y abarca tanto aplicaciones médicas de procesamiento de señales como procesamiento del lenguaje natural y reconocimiento de imágenes y voz.
6. **Métodos "Ensemble":** los métodos combinados utilizan varios algoritmos de aprendizaje distintos para lograr un rendimiento predictivo conjunto que supere el que podría adquirirse individualmente con cualquiera de los algoritmos de aprendizaje que lo forman, según el autor Sancho (2018).

1.1.2 Aprendizaje no supervisado

El aprendizaje no supervisado se produce en los casos en los que el ordenador aprende sin disponer de datos "etiquetados" para el entrenamiento. Por ello, tiene un carácter exploratorio y únicamente se puede avanzar describiendo la estructura de los datos, en búsqueda de algún tipo de secuencia que facilite el análisis.

El aprendizaje no supervisado se suele usar tanto en problemas de *clustering*, donde se buscan agrupamientos basados en similitudes para encontrar correlaciones, como en agrupamientos de co-ocurrencias y perfilados (*profiling*). Por ello, los algoritmos más utilizados en aprendizaje no supervisado son los algoritmos de clustering, los de análisis de componentes principales (PCA), los de descomposición en valores singulares (singular value decomposition) y los de análisis de componentes principales (Independent Component Analysis), según la autora Paloma de los Santos.

1.1.3 Aprendizaje reforzado

El aprendizaje reforzado consiste en superar cada vez el resultado del modelo mediante un proceso de retroalimentación. Así, aprende a base de prueba y error, pues sus variables de entrada están compuestas por el *feedback* que recibe del mundo exterior como respuesta a sus acciones, el cual va monitorizando y, por ende, mejorando.

Como señala el autor Simeone (2018), su mecanismo no encaja exactamente ni como aprendizaje supervisado, pues no parte de una conexión entre las variables de entrada y de salida, ni como aprendizaje no supervisado, ya que una vez que aprende dispondrá de datos etiquetados.

1.1.4 Polivalencia de las redes neuronales

El caso de las redes neuronales es particular, puesto que según su finalidad pueden encajar en cualquiera de las tres categorías mencionadas anteriormente. Las redes neuronales se utilizan ampliamente en el aprendizaje supervisado y en los problemas de aprendizaje por refuerzo. Asimismo, también se utilizan las redes neuronales dentro del aprendizaje no supervisado, pero es menos común. Ejemplos de ello, según la Profesora Becker de la Universidad de Toronto (1991) y el autor Lee Wei En (2019), incluyen los autocodificadores y mapas autoorganizativos (SOMs), entre otros.

En el ámbito de la propiedad industrial, la herramienta de *Machine Learning* más utilizada son las redes neuronales, en concreto las redes neuronales profundas (en inglés, *Deep Learning*), dado que la clasificación y el reconocimiento de imágenes es complejo. Por ello, en el apartado "*III. Estudio de las técnicas de aplicación en el caso práctico*" se explicará en profundidad la funcionalidad y tipos de las redes neuronales profundas.

1.2. Estado del arte

Hoy en día se aplican herramientas de *Machine Learning* en todos los ámbitos de la sociedad. Sin embargo, en este trabajo interesa su aplicación en el campo de la propiedad industrial.

La propiedad industrial protege las marcas, las patentes (invenciones), los diseños industriales, los dibujos y las denominaciones de origen. La protección por propiedad industrial otorga dos tipos de derechos a sus dueños, el primero es el derecho a utilizar la invención, diseño o signo distintivo, y el segundo es el derecho a prohibir que un tercero haga uso del mismo sin autorización.

Por tanto, en el campo de la propiedad industrial destacan los siguientes avances:

1.2.1 Reconocimiento de imágenes

Es el campo sobre el cual versa el presente trabajo. Mediante el empleo de redes neuronales convolucionales se logra una alta precisión en la catalogación de imágenes, lo cual facilita la detección de infracción de marcas y diseños industriales, entre otros.

1.2.2 Reconocimiento del habla

Sus finalidades principales son mejorar la calidad del audio eliminando ruidos, ayudar a la transcripción de textos y permitir la detección de sonidos o canciones. Las redes neuronales profundas consiguen una alta precisión. Bien es cierto que su aplicabilidad es más relevante en el campo de la propiedad intelectual, también puede ser útil en el campo de la propiedad industrial, en el caso de que se registren eslóganes, característicos de una marca.

1.2.3 Automovilismo

Entre sus funciones más destacadas se encuentran la detección inmediata de obstáculos y señales de tráfico, la predicción de velocidades y, por consiguiente, la prevención de accidentes. La conducción autónoma es un campo que se está desarrollando exponencialmente especialmente gracias a las redes neuronales profundas. En el campo de la propiedad industrial es relevante pues concierne a las patentes. Todos los grandes avances mencionados son invenciones protegidas legalmente mediante la figura jurídica de patentes.

1.2.4 Sanidad

Al igual que en el caso del automovilismo, la salud es un campo especialmente caracterizado por la gran utilización de patentes. Por ejemplo, la mayoría de los fármacos están patentados. Mediante el empleo de algoritmos de Machine Learning se pueden realizar rápidos diagnósticos médicos con tratamientos más acertados, predecir la probabilidad de padecer en un futuro alguna enfermedad basándose en las características, estilo de vida y antecedentes del paciente. Asimismo, se pueden investigar enfermedades y encontrar o mejorar curas. Finalmente, permite dar una atención más cercana, rápida y personalizada a cada paciente.

Capítulo 2: Estudio de las técnicas de aplicación en el caso práctico

2.1. Redes neuronales artificiales

Las redes neuronales artificiales o *artificial neural networks* (ANN, en inglés) se comparan siempre con la cognición humana, pues buscan imitar la estructura del cerebro humano y la secuencia neuronal de este. Las redes neuronales son modelos matemáticos computacionales, paralelos, compuestos de unidades procesadoras adaptativas (llamadas neuronas) con una alta interconexión entre ellas. Visto como un proceso de reconocimiento de patrones, las redes neuronales son una extensión de los métodos estadísticos. Las redes neuronales pueden usarse para dos posibles objetivos. En primer lugar, objetivos de clasificación, para diferenciar unos elementos de otros, puede ir desde ofrecer la solución de funciones lógicas o a la clasificación de imágenes (como realizar distinciones entre tipos de animales o reconocimiento facial). En segundo lugar, para objetivos de regresión, a partir de resultados previos se realiza una aproximación númerica a situaciones diferentes de las cuales se desconoce su resultado.

Las redes neuronales tienen múltiples ventajas. Una neurona funciona de forma no lineal, lo cual permite le operar con sistemas no sistemáticos o caóticos. Así, son tolerantes a fallos, es decir que el fallo en una neurona no obteniendo la salida optima no tiene porque afectar al buen funcionamiento de nuestra red. Igualmente, destacan por su adaptabilidad, ya que su sistema ofrece cierta capacidad a los cambios en el entorno de trabajo, presencia de ruido, modificaciones en la entrada, entre otros. Tiene la capacidad de establecer relaciones complejas entre los datos de entrada. La desventaja principal es que las redes neuronales, en especial las convolucionales, presentan un coste mayor de programación.

2.1.1 Estructura de la red neuronal

Una neurona es el elemento básico de una red neuronal y en esencia es una unidad de procesamiento. Existen tres tipos diferentes de neuronas, según su función en el proceso. Por un lado, están las neuronas de entrada, cuya función es analizar la información procedente del exterior e introducirla en el sistema. Por otro lado, existen las neuronas ocultas, que son el centro de la red, que surgen a partir de la conexión o enlace entre las diferentes neuronas de entrada, produciendo lo que se conoce como sesgo. Cuando hay múltiples capas de neuronas ocultas, se trata de una red neuronal profunda (*Deep Learning*) que se estudiará en el siguiente apartado. Finalmente, se encuentran las neuronas de salida, que se ocupan de mostrar el resultado de la red, tanto de regresión como clasificatorio.

Así, las neuronas de entrada se van ajustando dentro de la red, a lo largo de las distintas capas ocultas, hasta configurar una neurona salida, la cual al inicio del entrenamiento producirá una respuesta distinta a la deseada, pero que después de muchos entrenamientos se perfeccionará, cambiando los pesos de las conexiones, y se logrará el resultado deseado. Para establecer conexiones entre neuronas, creando neuronas ocultas, se debe dar una regla de propagación que relacione las neuronas de entrada y la neurona oculta.

Además de la regla de propagación, a la red neuronal deben aplicarse otros factores (o funciones matemáticas), entre los cuales destacan las reglas de activación y desactivación de las neuronas. La activación es el grado o nivel de excitación de una neurona y es imprescindible para el procesamiento, dado que el resultado de la red neuronal depende del grado de activación de la neurona. La activación varía en cada entrenamiento (conocido como ciclo) en base a cuatro factores, que son: el estado de activación previo de la unidad; la información de entrada que la neurona recibe; los pesos de las conexiones por las que recoge las señales; y finalmente la función de activación utilizada para calcular la activación a partir de dichas entradas. Algunas de las funciones de activación más conocidas incluyen, en primer lugar, la función ReLU (Rectified Lineal Unit), que anula los valores negativos y deja los positivos tal como entran. Esta función ha demostrado ser útil en casos de reconocimiento de imágenes. En segundo lugar, destaca la función lineal, parecida a la ReLU, pero a la inversa, puesto que resalta los valores negativos,

exagerándolos o suavizándolos según interese. En tercer lugar, se encuentra la función sigmoidal, que es especialmente útil pues ayuda a optimizar el coste computacional, pues produce salidas continuas y proporcionales al nivel de activación de la neurona dentro del rango [0,1], con lo cual si el nivel de activación supera el umbral de saturación máximo la salida seguirá siendo 1 y si el nivel de activación es inferior al umbral de saturación mínimo, la salida seguirá siendo 0.

Por otro lado, es conveniente mencionar las reglas de aprendizaje, que son los algoritmos usados para modificar los pesos y el umbral de saturación con el objetivo de reducir el error producido por la red. Dicho error se mide como la diferencia entre la salida real de la red y la deseada, y existen varias funciones para calcularlo, entre las que destacan la función del error cuadrático y la fórmula de la entropía cruzada. Así las cosas, entre las reglas de aprendizaje cabe resaltar en primer lugar la Hebbian rule, que fue la primera regla de aprendizaje. La Hebbian rule optimiza los pesos de las neuronas de la red asumiendo que, si dos neuronas vecinas están activas a la vez, el peso que las une deberá incrementarse, y si no disminuirse. Además, añade que, si la información de entrada de ambas neuronas vecinas es positiva, significa que existe un fuerte peso positivo entre ellas y si una es negativa y la otra positiva, conlleva que existe un gran peso negativo entre ellas. En segundo lugar, destaca la regla de aprendizaje del perceptrón simple (es un tipo de red), por la cual el peso es proporcional al error producido multiplicado por el valor de la entrada. En tercer lugar, se encuentra la regla del método delta, que se aplica en el aprendizaje supervisado, partiendo de que el peso sináptico de una neurona cambia en base al error multiplicado una cuota de aprendizaje y por su entrada, y comparando si la salida es igual que el resultado deseado. En cuarto lugar, está el método del gradiente estocástico, por el cual se modifican los pesos para encontrar mínimos en funciones diferenciables. El algoritmo de retropropagación se usa para modificar los pesos.

2.1.2 Tipos de redes neuronales artificiales

Existen dos tipos distintos de redes neuronales, según el número de capas y como estén definidas sus funciones de activación, de propagación, de aprendizaje.

Así pues, en primer lugar, se encuentran las redes neuronales prealimentadas, en inglés conocidas como las *feed-forward propagation*, que funcionan únicamente hacia delante. Dentro de este tipo de red, destacan el perceptrón y el Adaline. El perceptrón es la red neuronal utilizada para la separación de patrones linealmente separables. Así, se construye un algoritmo idóneo para elegir un sub-grupo de entre un grupo de atributos más grande, lo cual es muy útil. Sin embargo, tiene como limitación que no es flexible, debiendo separarse los atributos con un hiperplano. Similar al perceptrón es la red Adaline, cuya diferencia es que, en el momento de ajustar los pesos a lo largo de los entrenamientos, el Adaline es más preciso. Así, considera el grado de corrección de la salida estimada respecto a la deseada utilizando la regla de aprendizaje del método delta y como función de activación la función signo. Para ello, según los autores Kröse y Smagt (1996), se fija para un patrón de entrada, una salida estimada y otra deseada, y seguidamente intenta minimizar la desviación de la red para todos los patrones de entrada mediante la función del error cuadrático medio.

En segundo lugar, se encuentran las redes recurrentes, conocidas como las *feed back-forward propagation* o *back-forward*, que permiten conexiones, tanto para delante como para atrás, entre redes de la misma capa, incluida la misma neurona y hacia capas anteriores. Según los autores Soria y Blanco (2001), las redes recurrentes son más flexibles y adaptables, por lo que permiten alcanzar estados de estabilidad en la activación de ciertas neuronas.

2.1.3 Entrenamiento de la red neuronal

Las redes neuronales tienen dos fases, una primera de entrenamiento y, finalmente, otra de predicción. La fase de entrenamiento consiste en ajustar, mediante múltiples reiteraciones o épocas del proceso, los pesos de las conexiones para lograr la salida deseada. Para ello se requieren dos conjuntos de datos: uno de entrenamiento y otro de validación. Al inicio del entrenamiento, los pesos de la red son elegidos al azar. Sin embargo, utilizando el conjunto de datos de entrenamiento se va ajustando en sucesivas iteraciones los datos, reduciendo al máximo el error cometido por estos. Cuando se terminan las reiteraciones deseadas y los datos de entrenamiento son predichos como se desea, se comprueba la utilidad de la red utilizando los datos de validación. Si la red predice los datos de validación correctamente, se da por terminada la fase de entrenamiento. Por otro lado, si la red no predice de forma correcta los datos de validación, suele deberse principalmente por dos razones: por una incorrecta elección de los datos de entrenamiento (por ser estos poco representativos o insuficientes) o por un sobreajuste (por un sobreentrenamiento la red es incapaz de generalizar).

En cuanto a la fase de entrenamiento, el algoritmo de descenso del gradiente es el más usado para entrenar redes neuronales, que consiste en un cálculo que establece cómo ajustar las ponderaciones y sesgos de la red para que se reduzca su contaminación a la salida. Existen distintos tipos de algoritmos de descenso del gradiente según el número de muestras que se configuren en la red para cada iteración. En primer lugar, existe el descenso del gradiente en lotes (conocido en inglés como *batch*), donde todos los datos disponibles se incluyen en una única vez. Este sistema, aunque es rápido, puede provocar problemas de paralización, puesto que el gradiente se computa utilizando siempre todas las muestras, y finalmente las variaciones son mínimas. Así pues, generalmente es conveniente que la entrada de una red neuronal sea aleatoria. En segundo lugar, se encuentra el descenso del gradiente estocástico, en el cual se implementa solo una muestra aleatoria en cada iteración. Así, el gradiente se calcula únicamente para esa muestra específica, manteniendo convenientemente la aleatoriedad en la entrada. La desventaja de este segundo método es que requiere múltiples iteraciones, para obtener el resultado óptimo, lo que conlleva mucho tiempo. En tercer lugar, está el descenso del gradiente estocástico en mini-lotes (o también conocido como *mini-batch*), que se presenta como

una alternativa intermedia, en el cual se introducen N muestras en cada iteración. Por tanto, según el autor Durán (2019), con este tercer sistema se consigue conservar la aleatoriedad, a la vez que no se ralentiza excesivamente el tiempo de entrenamiento.

Adicionalmente, es relevante la selección de variables de entrada y del número de capas ocultas para construir un modelo eficiente. Generalmente se utilizan dos capas ocultas para la estimación de funciones continuas con ciertas discontinuidades, sin embargo, utilizar dos capas ocultas es suficiente para incrementar las probabilidades de obtener errores locales. Otra manera de seleccionar el número de capas ocultas y de neuronas es entrenando a la vez distintas redes con diferentes estructuras y, después, compararlas para elegir la que cometa un menor error con los datos de validación. Para visualizar el error cometido por la red en un problema de clasificación se utiliza la curva ROC (*Receiver Operating Characteristic*) que muestra la sensibilidad frente a la especificidad del sistema. Una métrica de la eficacia de la curva ROC es la AUC (*Area Under the ROC Curve*).

2.2. Redes neuronales profundas (*Deep Learning*)

Una red neuronal profunda (*Deep Neural Network,* DNN, en inglés) es una red neuronal artificial (*Artificial Neural Network,* ANN, en inglés) compuesta por más de dos capas ocultas (generalmente no lineales) entre las capas de entrada y de salida. Con fines aclaratorios se ofrece un esquema a continuación.

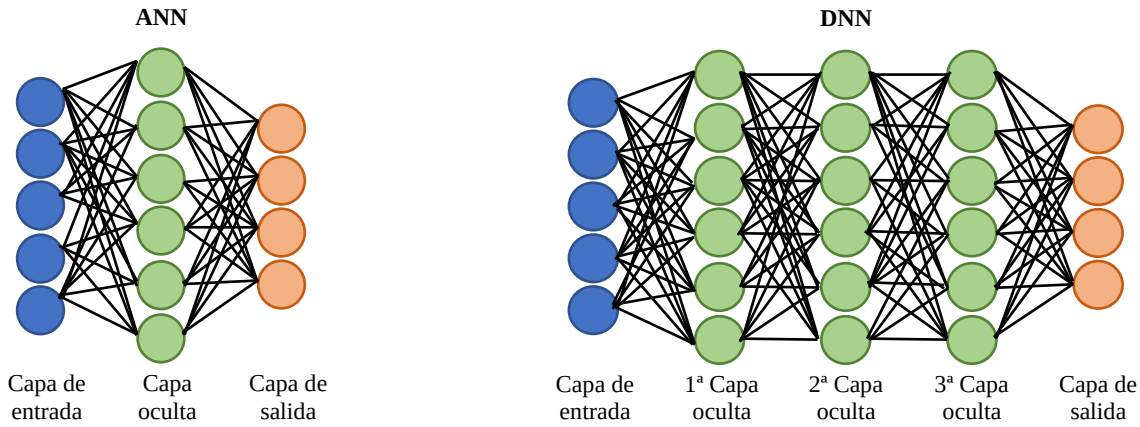

*Figura I: Comparativa de las estructuras típicas de la red neuronal artificial (ANN) y la red neuronal profunda (DNN)*1
Fuente: Elaboración propia

El método que más se usa para optimizar la red neuronal profunda es el método de descenso de gradiente, que es un algoritmo de optimización que se emplea en los entrenamientos para minimizar el error de predicción, lo cual, de acuerdo con el autor Rodríguez (2018), depende de la calidad de sus ponderaciones y sesgos. Así pues, cuando se entrena un conjunto de datos, se calcula continuamente la función de costo, es decir, la divergencia entre el resultado esperado y el resultado real del conjunto de datos etiquetados de entrenamiento. Como se ha mencionado, dicha función se reduce cambiando los pesos y sesgos hasta que se logra obtener el valor más bajo. El gradiente utilizado en el proceso de entrenamiento consiste en una tasa que modifica el costo cuando se varía los valores de peso o sesgo. Asimismo, entre los tipos de redes neuronales anteriormente señaladas, la *Backpropagation* es la más utilizada. Las redes neuronales profundas son las utilizadas en casos complejos de reconocimiento de patrones, pues son los algoritmos más completos y potentes, como ocurre en casos de reconocimiento de rostros humanos, donde cada detalle puede ser trascendente. Sin embargo, la clara

consecuencia derivada de su alta calidad predictiva y precisión es su largo tiempo de procesamiento.

El tipo de red neuronal profunda a utilizar depende de la finalidad para la que se pretenda utilizar, es decir, si lo que se desea es resolver un problema de clasificación o bien simplemente encontrar patrones, tendencias en un problema de aprendizaje no supervisado. En este último caso se suele utilizar la red recurrente denominada Máquina de Boltzmann restringida, la cual se utiliza para el procesamiento de texto y el análisis de sentimientos, entre otros. En definitiva, cuando se requiera un sistema que analice caracteres o voz, se utiliza la red recurrente. También se usa la red recurrente para analizar series de tiempo. Por otro lado, para el reconocimiento de imágenes, se emplea la red de creencias profundas (conocida en inglés como *Deep Belief Network* (DBN)) o la red convolucional. Adicionalmente, si lo que se pretende es lograr el reconocimiento de objetos, se usa una red de tensor neuronal recursivo (en inglés, *Recursive Neural Tensor Network* (RNTN)) o una red convolucional.

La Máquina de Boltzmann restringida (RBM), mencionada anteriormente, es una red de dos capas poco profunda que resuelve el problema del gradiente de fuga (i.e., cuando el gradiente de la capa anterior es más pequeño que el de la capa posterior, provocando que el proceso sea más lento), impidiendo que se compartan conexiones dentro de una misma capa.

2.2.1 Tipos de redes neuronales profundas

2.2.1.1 Redes de creencias profundas (DBN)

Las redes de creencias profundas (en inglés, *Deep Belief Network* (DBN)) son redes neuronales profundas que se construyen añadiendo a la Máquina de Boltzmann restringida (RBM) un método de entrenamiento inteligente. Así, se consigue atacar el problema de la desaparición de gradiente, que es un supuesto que se produce en la retropropagación de error a través del descenso de gradiente estocástico. Así pues, en una red de creencia profunda, cada red de Boltzmann restringida aprende toda la información de entrada, por lo que se ajusta global e íntegramente el modelo en base a la información de entrada, asemejándose, según los autores Goodfellow, Bengio y Courville (2016), a la lente de una cámara enfocando paulatinamente una imagen.

2.2.1.2 Redes generativas antagónicas (GAN)

Las redes generativas antagónicas (GAN) son redes neuronales profundas que comprenden dos redes, que reciben el adjetivo de “antagónicas” por estar contrapuestas la una con la otra. Dichas redes neuronales tienen un gran potencial, puesto que son capaces de aprender de repetir cualquier distribución de datos, tanto fotográfica, como musical como literaria, entre otras. Las GANs se componen de dos partes, por un lado, el generador, que reproduce nuevas versiones de los datos reales y, por otro lado, el discriminador, que valora su autenticidad, devolviendo un 1 si es verdadera y un 0 si es falsa.

2.2.1.3 Redes neuronales recurrentes (RNN)

Las redes neuronales recurrentes (RNN) son redes neuronales profundas que se usan en aplicaciones como el modelado de lenguaje o el procesamiento de lenguaje natural (NLP). Una característica esencial de la RNN es el uso de información secuencial, es decir, para predecir la siguiente palabra en una frase, la RNN necesita conocer las palabras previas, como ocurre en los navegadores online en los que anticipan las posibles siguientes palabras. Por ello, se entiende que las RNN tiene una memoria que almacena información sobre lo que se ha calculado previamente y la capacidad de dirigirse tanto para adelante como para atrás. Esto difiere del resto de redes neuronales, en las cuales las entradas y salidas son independientes entre sí.

El adjetivo "recurrentes" proviene de que repiten la misma acción para cada componente de la secuencia. Las redes de memoria a largo plazo (LSTM) son las RNN más utilizadas.

Al igual que las redes neuronales convolucionales, las RNN se han utilizado para generar descripciones de imágenes no etiquetadas.

2.2.1.4 Redes neuronales convolucionales (CNN)

Las redes neuronales convolucionales (CNN) son redes neuronales profundas, de múltiples capas (en ocasiones superando la veintena), que se usan abundantemente en el reconocimiento automático de imágenes y de visión artificial, y de reconocimiento de voz. Así pues, los datos de entrada son generalmente imágenes. Dada la relevancia que tienen en el presente trabajo, se estudiarán en profundidad en el siguiente apartado.

2.2.2 Estudio en profundidad de las redes neuronales convolucionales

Las redes neuronales convolucionales son un tipo de red neuronal profunda que surgió a partir de una modificación del perceptrón multicapa, del cual se diferencia en el uso de capas convolucionales y subselectivas.

Las redes neuronales convolucionales se pueden visualizar conceptualmente como un filtro en movimiento que atraviesa las imágenes que va recibiendo, a una proximidad predeterminada de las neuronas. De acuerdo con la página web Código Fuente (2019), las CNN minimizan el número de parámetros a ajustar, manejando eficazmente la alta dimensionalidad de las imágenes.

El investigador informático francés Yann LeCun fue el desarrollador pionero de las redes neuronales convolucionales, cuando en 1998 entrenó LeNet, una CNN capaz de clasificar imágenes digitales escritas a mano con un 99% de precisión. Asimismo, en el concurso *ImageNet Large Scale Visual Recognition Challenge* de 2015, un ordenador empleando redes neuronales convolucionales fue capaz reconocer objetos más rápido que un humano. Facebook usa las CNN como software de reconocimiento facial.

2.2.2.1 Estructura de las redes neuronales convolucionales

Las redes neuronales convolucionales en la mayoría de los casos parten de una imagen, y se componen de una serie de capas de convolución que alternan con unas de submuestreo (en inglés, *pooling*), para lograr una predicción, a la cual otorgan una probabilidad de acierto o porcentaje de similitud. La convolución es un tipo de operación lineal. A continuación, se ofrece a modo de ejemplo un diagrama básico de la estructura de las CNN partiendo de una imagen obtenida mediante Webscrapping de la página web de Nike de su colección de esta temporada de mochilas.

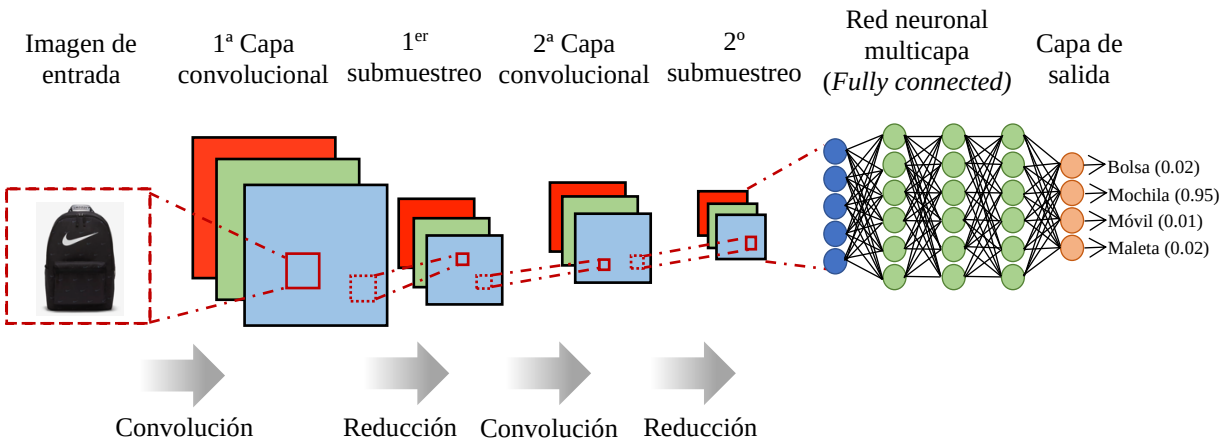

Figura II: Estructura típica de la red neuronal profunda convolucional (CNN)[2]

Fuente: Elaboración propia

La dimensión de entrada en las CNN depende de si la imagen recibida está en color, en cuyo caso será tridimensional, o en blanco y negro, para la cual será bidimensional. Es importante la dimensión de entrada puesto que se deberá adaptar el núcleo de la red (i.e., las capas ocultas) a ese número de dimensión.

Por ello, si la imagen introducida está a color, será tridimensional, y la red tendrá tres capas bidimensionales correspondientes a los colores primarios de la luz (rojo, verde y azul, denominadas RGB). En definitiva, si la imagen de entrada está a color, la convolución de la red neuronal se traduce en la construcción de tres capas (una para color, que juntas son la profundidad) convoluciones bidimensionales (altura x ancho), y después se suman los resultados. Para que funcione correctamente la red neuronal, la dimensión de la capa de entrada debe ser igual a la dimensión del núcleo, lo cual frecuentemente no ocurre. Para cuadrar ambas dimensiones, una solución frecuente es rellenar con ceros la dimensión menor para aumentarla, lo cual se denomina proceso de *zero-padding*. Así, primero se añade una columna de ceros en la derecha o abajo donde se necesite, y si se precisa más se completa secundariamente con ceros en la izquierda o arriba. Con fines aclaratorios se proporciona un ejemplo a continuación.

$$I = \begin{pmatrix} 1 & 3 \\ 2 & 2 \end{pmatrix} \qquad pad\,(I, 1) = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 3 & 0 \\ 0 & 2 & 2 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

En definitiva, como se estudiará a continuación, las CNN están compuestas por capas convolucionales, que se alternan con capas de submuestreo. El número de capas depende de la complejidad de la imagen, a mayores pixeles, más capas serán necesarias (i.e., más veces se repetirá el proceso iterativo) hasta alcanzar un conjunto de matrices que, mediante un proceso de *flattering*, se puedan unir sus elementos en un único vector de tamaño manejable para ser introducido como entrada de una red neuronal *feedforward* que finaliza el problema de clasificación o regresión de forma tradicional.

2.2.2.2 Funcionamiento de las redes neuronales convolucionales

a. Pre-procesamiento: tratamiento de los píxeles

En primer lugar, como se ha mencionado anteriormente, la CNN recibe como información de entrada una imagen, en concreto sus pixeles. El número de neuronas necesarias en la capa de entrada equivale al número de píxeles de la imagen, multiplicado por 3 (en caso de ser una imagen a color, como se ha explicado en el apartado anterior), por 2 en caso (de ser en blanco y negro) o por 1 (si hay solo 1 color, por ejemplo, en una escala de amarillos).

Adicionalmente, de acuerdo con Barrios (2021), cada píxel adopta un valor entre 0 y 255 según su intensidad del color (así, por ejemplo, un píxel color rojo puro tendrá un valor de 255). Por ello, es conveniente normalizar los valores, dividiéndolos entre 255 para transformarlos entre 0 y 1. La finalidad es reducir el peso o carga procesal de la imagen y así reducir su costo y tiempo de procesamiento, dado que las imágenes generalmente tienen un número elevado de píxeles.

b. Capas convolucionales

Las capas convolucionales constituyen la parte más característica de las CNN. En ellas, se procede agrupando los píxeles según su cercanía en la imagen de entrada y operando matemáticamente mediante el producto escalar contra una matriz reducida denominada **kernel**. Lo que se pretende mediante los kernels es comprimir la máxima información (o la esencial) de los pixeles en una matriz de salida más reducida y simplificada, en la que, por ende, se utilicen menos neuronas. En cada capa convolucional habrá kernels diferentes y conjuntamente se les conoce como filtros. El resultado, de multiplicar cada uno de los valores del Kernel por el valor del pixel que se encuentra en la misma posición, es una nueva matriz que se denomina mapa de características.

Asimismo, según autores como Gulli y Pal (2017), otro elemento esencial es la **función de activación**, que por excelencia es la **Rectifier Linear Unit (ReLU)** y se define con la siguiente función: $f(x) = max(0, x)$, donde 'x' es la entrada de la neurona. La finalidad de esta función es evitar que haya números negativos en la salida de la neurona.

c. Submuestreo (*subsampling* o *pooling*)

Antes de realizar una nueva convolución, es necesario reducir el número de neuronas mediante la elección de una muestra, puesto que, de lo contrario, el número de neuronas aumentará exponencialmente de capa en capa convolucional y el tiempo de procesamiento será extremadamente lento. Por ello, se considera que la mejor solución es escoger una muestra que conserve las especialidades más significativas que detectó cada filtro.

Hay varias formas de realizar el muestreo, entre las que destaca el denominado ***max-pooling***. Este sistema de muestreo consiste en subdividir la matriz en grupos de 2x2 e ir conservando el valor más elevado de entre esos 4 pixeles. Así, se reducirá a la mitad el número de píxeles a lo alto y ancho. No obstante, se conservará el número de píxeles de profundidad, pues corresponde al número de dimensiones. Otro sistema conocido de muestreo es el denominado *average-pooling* que se diferencia del *max-pooling* en que el

criterio decisivo es la media aritmética, en lugar del valor más elevado de entre los píxeles del subgrupo.

Los pasos 2 y 3 se repiten sucesivamente hasta reducir el número de neuronas a uno viable.

d. Unión a una red neuronal *feedforward*

Finalmente, se coge la última capa oculta y se aplana, eliminando su múltiple dimensionalidad y convirtiéndola en una capa feedforward. Posteriormente, a dicha capa unidimensional se le aplica una función denominada **SoftMax**, que une la capa oculta con la de salida, que poseerá el número de neuronas igual al número de clases que se quiera clasificar (i.e., si se debe elegir entre mesa o silla, el número resultante de neuronas será 2) en formato *one-hot-encoding* (i.e., [1,0]). Asimismo, la función SoftMax se ocupa de proporcionar la probabilidad de que sea una u otra clase la resultante (p.ej. [0.7,0.3])

En resumen, la capa de entrada se compone de un número de neuronas equivalente al número de pixeles de la imagen. En segundo lugar, la capa de convolución agrupa los píxeles cercanos y calcula el producto escalar considerando el valor de píxel (peso). En tercer lugar, la capa ReLU se utiliza para emplear la función de activación en los elementos de la matriz. En tercer lugar, se realiza un submuestreo de la matriz resultante de la capa convolucional para reducir el número de píxeles, tanto en su altura como en su anchura, pero conservando los de su profundidad. Finalmente, una vez obtenido, mediante la repetición de la combinación de capas convolucionales y muestreos, un tamaño de matriz que conserve la información relevante de la imagen y a la vez sea manejable, se procede a una capa habitual *feedforward* que finalizará la regresión o clasificación deseada.

2.2.2.3 Diferencias entre la red neuronal feedforward multicapa y la red neural convolucional

Las redes neuronales *feedforward multicapa* son redes neuronales tradicionales que también se denominan *fully connected* por estar completamente conectadas todas las neuronas de una capa con las neuronas de la capa previa y la posterior. Así, mediante el método matemático de *Backpropagation,* se ajustan los pesos de todas las interconexiones de las capas, mientras que las CNN no son *fully connected* pues sólo se ajusta el valor de los pesos usados en los distintos kernels (filtros). Como se ha expuesto en el apartado anterior, la red neuronal convolucional tiene al final de la misma una capa *fully connected.*

Existen claras diferencias entre las CNN y las redes neuronales tradicionales en cada una de las capas. En primer lugar, la información que se analiza en la capa de entrada en el caso de las redes neuronales tradicionales son atributos (ej. Color de ojos, sexo, altura, etc.) y en las CNN son píxeles de una imagen. En segundo lugar, en las capas ocultas de una red neuronal tradicional se escoge voluntariamente el número de neuronas, mientras que en la CNN el número de neuronas depende del número de píxeles multiplicado por 1, 2 o 3 dependiendo de si la imagen es en escala de 1 color, en blanco y negro o a color. Asimismo, el número de capas ocultas es discrecional y no significa nada en concreto. Sin embargo, en las CNN las capas ocultas constituyen verdaderos mapas de detección de características de la imagen y a medida que aumenta el número de capas ocultas convolucionales, aumenta igualmente la cantidad de detalle procesado. Así, las primeras capas detectan líneas, las siguientes curvas, y finalmente formas más elaboradas. Finalmente, según Barrios (2021) el número de neuronas en la capa de salida de una red neuronal tradicional es igual al número de categorías en las que se quiere clasificar, mientras que en las CNN se debe minimizar la dimensionalidad (aplanar) de la ultima capa convolucional y mediante la función SoftMax unirlo a una capa de salida tradicional.

2.2.2.4 Redes neuronales convolucionales siamesas

De esencial interés en el presente trabajo, existe una forma de combinar dos redes neuronales convolucionales, a las que se denomina siamesas, con el fin de comparar dos imágenes y devolver el porcentaje de similitud entre ellas. Una aplicación conocida que utiliza redes neuronales convolucionales siamesas es la validación de identidad mediante la verificación facial.

El modelo se compone de dos redes neuronales convolucionales paralelas, como las explicadas anteriormente, que comparten los mismos pesos y sesgos. Por tanto, el objetivo de la función de costo es encontrar la distancia que existe entre las propiedades conseguidas al introducir dos imágenes en el modelo, una por cada una de las redes neuronales convolucionales. La salida de cada subred es una capa totalmente conectada (*Fully Connected*). Generalmente, se calcula la distancia euclidiana entre estas salidas y se las hace pasar por una activación sigmoidea para poder determinar la similitud de las dos imágenes de entrada. Los valores de la función de activación sigmoide más cercanos a "1" implican una mayor similitud entre las imágenes, mientras que los valores más cercanos a "0" indican una menor similitud. De acuerdo con el autor Bastidas (2020), para entrenar la arquitectura de la red siamesa, existe una serie de funciones de pérdida que se pueden utilizar, incluyendo la entropía cruzada binaria, la pérdida de triples y la pérdida contrastiva (*contrastive loss*).
A continuación, se proporciona un esquema con fines aclarativos:

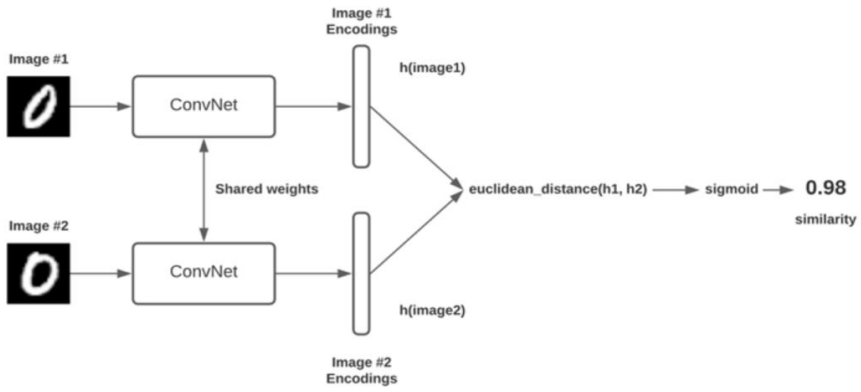

Figura III: Estructura típica de una red neuronal convolucional siamesa. 3
Fuente: Rosebrock (2020)

Capítulo 3: Caso práctico

En el presente apartado se expondrá la parte práctica del trabajo, que consiste en probar la utilidad de las redes neuronales convolucionales en el campo de la propiedad industrial, para detectar automáticamente infracciones de marcas online mediante el reconocimiento de imágenes. Por tanto, la idea subyacente consiste en realizar, en primer lugar, Webscrapping de imágenes de las páginas web que se deseen contrastar, en este caso serán las marcas de ropa Asos y Nike. Lo ideal sería poder comprobar la autenticidad de todos los productos de ambas, pero en el presente trabajo, de todas las imágenes webscrappeadas, se contrastarán, con fines ejemplificativos, únicamente una imagen de una mochila de la colección de Nike de esta temporada con otra de Asos, también a la venta actualmente, para ver si los algoritmos de redes neuronales profundas convolucionales siamesas diseñadas en nuestro modelo detectan (y en cuánto porcentaje) la similitud entre ambas. Así pues, en segundo lugar, se realizará en Python la comparación entre ambas imágenes de las mochilas y se detectará el porcentaje de falsificación mediante las herramientas de TensorFlow y Keras, que se describen a continuación.

3.1. Webscrapping de imágenes

Actualmente existen múltiples formas de descargar imágenes masivamente de páginas web. Siempre se puede programar en Python, pero hoy en día adicionalmente existen extensiones en Google Chrome, como *Image Downloader* y *Web Scraper* que funcionan muy bien para recopilar imágenes de páginas web. Para la realización del caso práctico objeto del presente trabajo, se han descargado las imágenes de mochilas de Asos y Nike utilizando *Image Downloader*, una extensión sencilla e intuitiva.

Primero, se descarga la extensión *Image Downloader* desde Chrome Web Store y se añade automáticamente a la barra superior del explorador. Segundo, hay que meterse en la página web de la que se desee descargar las imágenes y pulsar el icono de *Image Downloader*. Así, aparecerá un grid de 3 columnas exponiendo las imágenes de esa

página, y marcando la casilla de “seleccionar todo” se pueden descargar todas las imágenes de esa página web. Adicionalmente, se puede crear una carpeta para que se almacenen todas las imágenes juntas ahí en nuestro ordenador. Por otro lado, si no se desea descargar todas las imágenes, se puede ir eligiendo una a una las que descargar.
A continuación, se proporcionan dos imágenes ilustrando como se realizaría la descarga de imágenes usando *Image Downloader*.

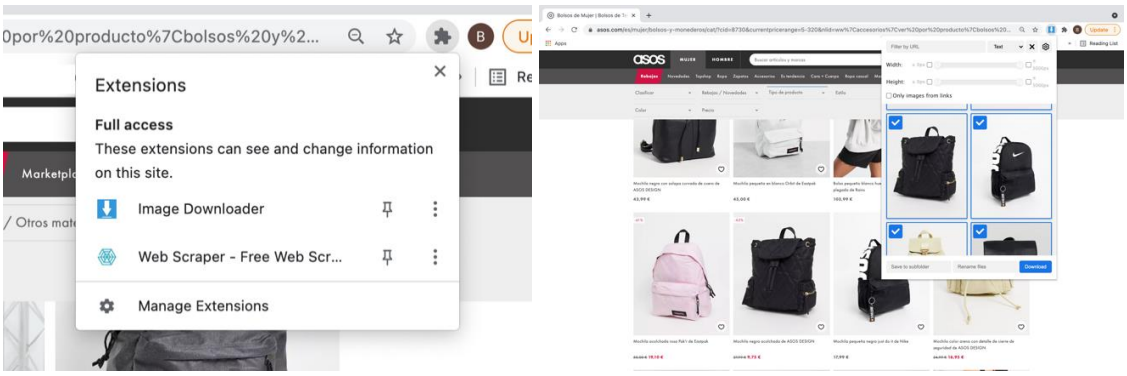

Figura IV: Realización de Webscrapping de Imágenes usando la extensión Image Downloader[4]

Fuente: Elaboración propia

3.2. Lenguaje de programación: Python

Entre los lenguajes de programación más empleados en el área de la Inteligencia Artificial sobresalen dos: Python y R. Sin embargo, en lo que respecta a las redes neuronales, en concreto las redes neuronales convolucionales, el lenguaje de programación dominante es Python, dado que cuenta con un gran número de librerías y *frameworks* como TensorFlow, Keras, Theano o PyTorch. Así las cosas, para la realización del caso práctico en cuestión se va a utilizar Python, y concretamente las herramientas TensorFlow y Keras.

3.2.1 TensorFlow

TensorFlow es una biblioteca de software de código abierto para realizar cálculos numéricos utilizando diagramas de flujo de datos. Los nodos de los gráficos representan operaciones matemáticas, y las aristas de los gráficos representan las matrices de datos multidimensionales (denominados tensores) que se pasan entre ellos. TensorFlow es una plataforma ideal para construir y entrenar redes neuronales, y para descubrir y descifrar patrones y correlaciones.

TensorFlow fue desarrollado inicialmente por investigadores e ingenieros de la División de Investigación de Inteligencia Artificial del equipo de Google Mind para realizar investigaciones sobre el aprendizaje automático y las redes neuronales profundas. En mayo de 2016, Google anunció la TPU (*Tensor Processing Unit*), un diseño especial de ASIC (*Application-Specific Integrated Circuit*) de aprendizaje automático adaptado a TensorFlow. La TPU es un acelerador de Inteligencia Artificial programable, diseñado para utilizar o ejecutar modelos, pero no para su entrenamiento.

La biblioteca TensorFlow se ha utilizado ya en el pasado para un amplio abanico de sectores, desde para mejorar la fotografía de los Smartphones y ayudar a analizar radiografías médicas hasta para el mejor procesamiento de imágenes. Así las cosas, una de las más destacadas aplicaciones de TensorFlow es la llamada *DeepDream*, que es un software automatizado de procesamiento de imágenes. *DeepDream* es un programa de visión artificial que emplea una red neuronal convolucional para buscar y superar patrones en imágenes mediante pareidolia algorítmica, es decir, mediante la construcción de una apariencia alucinógena, lo cual produce imágenes sobreprocesadas voluntariamente, como si fuera un sueño. Como dato curioso, las imágenes resultantes están adquiriendo valor en el campo del arte.

Según el autor Buhigas (2018), TensorFlow se ideó teniendo en cuenta su condición de código abierto, y lo fácil que es su ejecución (pudiendo realizarse tanto en la nube como localmente) y su escalabilidad. Es una biblioteca muy flexible y adaptable a multitud de dispositivos y aplicaciones.

3.2.2 Keras

Keras es una biblioteca de código abierto esencial para la programación en Python de redes neuronales, lanzada en 2015, que brinda una enorme variedad de estructuras fácilmente adaptables y flexibles. Así, se permite construir cómodamente capas de convolución y de submuestreo (en concreto de *max pooling*) a la vez que diseñar la correspondiente función de activación y regla de aprendizaje. En definitiva, el objetivo final de la biblioteca es aligerar la construcción de redes neuronales. Por lo tanto, en lugar de operar como un *framework* independiente, Keras funciona como una interfaz de uso intuitivo (API), para poder hacer uso de múltiples *frameworks* de aprendizaje automático y desarrollarlos, entre los que destaca el ya mencionado TensorFlow.

La biblioteca Keras ofrece una serie de ventajas considerables. En primer lugar, Keras tiene un diseño sencillo, accesible y proporcionando un *feedback* de ayuda en caso de errores. Todo esto mejora la productividad e incrementa el número de usuarios deseando utilizarlo. En segundo lugar, dado que se puede complementar con *frameworks* de aprendizaje profundo, como se ha mencionado anteriormente, esta simplicidad no provoca restricciones funcionales, pudiéndose hacer el modelo todo lo complejo que se desee. En tercer lugar, Keras es flexible y ampliamente compatible con sistemas importantes como iOS de Apple CoreML, Android (en concreto Keras TensorFlow Android Runtime) y Google Cloud. Finalmente, es compatible con múltiples motores de *backend* (es decir, configuradores de bases de datos), pudiendo combinar y alternar varios de ellos. Además, según la página web Digital Guide Ionos (2020) no es necesario decantarse por un *framework* concreto, pues es viable cambiar de *backend* fácilmente.

3.3. Comentario del caso práctico en Python

Se distinguen dos partes diferenciadas en la programación de las redes neuronales convolucionales siamesas: la fase de entrenamiento (*train*) y la fase de predicción (*test*).

En relación a la fase de entrenamiento, para realizar el caso práctico, en primer lugar, hay que configurar las imágenes de entrada, es decir, especificar las dimensiones de las imágenes de entrada (alto, ancho, dimensión (es decir, número de colores)). En este caso

las mochilas de Nike y de Asos, con unas dimensiones de (592, 592, 2) y de (476, 608, 2), respectivamente. Asimismo, se debe establecer el tamaño (conocido en inglés como *batch size*) y el número de iteraciones (o épocas), que en este caso se han fijado como 32 y 10, respectivamente.

En segundo lugar, se debe importar las librerías necesarias, las cuales previamente se habrán instalado utilizando, entre otros, el siguiente comando en la terminal: **~ % pip3 install TensorFlow**. Destacan varios paquetes, de acuerdo con el autor Leban (2020), entre los que se encuentran **build_siamese_model**, que permite construir el modelo de CNN siamesas, y **lambda**, que introduce la distancia Euclidea en la CNN siamesa.

En tercer lugar, es importante cargar las imágenes que se van a comparar, las cuales se encontrarán ya descargadas en el disco duro personal (por haberse hecho ya el Webscrapping explicado anteriormente). Según el autor Das (2020), las imágenes se cargan utilizando la función **cv2.imread**. Asimismo se debe normalizar el valor de sus píxeles entre [0, 1], dividiendo entre 255 los píxeles del conjunto de datos de entrenamiento de ambas imágenes. La función **plt.show** devuelve las siguientes imágenes, que incluyen el número de píxeles, como se muestra a continuación.

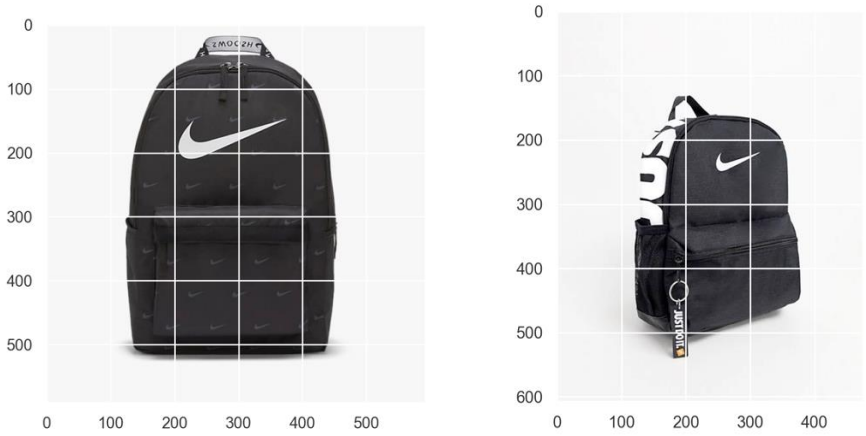

Figura V: Imágenes de entrada del caso práctico 5

Fuente: Código del Anexo en Python

En cuarto lugar, se configura la CNN siamesa, utilizando la función **build_siamese_model** y se construye utilizando la función **lambda (utils.euclidean_distance) ([featsnike, featsasos])**. Así, se calcula la distancia Euclidea entre ambas imágenes.

Finalmente, se compila el modelo usando la función **model.compile (loss="binary_crossentropy", optimizer="adam", metrics=["accuracy"]**) y se guarda el modelo usando la función **model.save(config.MODEL_PATH)**. Adicionalmente, se puede representar gráficamente lo conseguido en la fase de entrenamiento utilizando la función **utils.plot_training(history, config.PLOT_PATH)**.

Por otro lado, en relación a la fase de predicción, de nuevo en primer lugar se debe importar los paquetes necesarios, entre los que se encuentran load_model (función perteneciente a las bibliotecas de Keras y TensorFlow para cargar desde el disco duro, la red neuronal siamesa entrenada previamente), list_images, plt (para mostrar gráficos e imágenes), np, argparse y cv2.

Posteriormente, se cargar el modelo previamente entrenado utilizando la función **load_model(config.MODEL_PATH)** y las imágenes usando la función **cv2.imread**. Para ello, se inicia un bucle en el que se carga cada imagen, se crea una copia de ellas para poder visualizarlas gráficamente posteriormente, y se escalan las dimensiones de los píxeles en el rango [0, 255] a [0, 1], al igual que se hace en la fase de entrenamiento.

Una vez cargadas las imágenes, se utiliza la función **model.predict([nikeimage, asosimage])** para hacer predicciones sobre la similitud entre ambas imágenes, lo cual devuelve como resultado las puntuaciones de similitud entre las dos imágenes.

Finalmente, se muestran las dos imágenes utilizando la librería **matplotlib** y su correspondiente puntuación de similitud, que estará en el rango [0,1] y cuanto más cercana al valor 1, mayor será. Generalmente, se considera que si dos imágenes tienen una similitud mayor a 0.5 pertenecen a la misma clase, y si es menor de 0.5 a diferentes categorías. Con todo ello, se ayuda a mejorar la determinación del grado de infracción de marcas, incrementando su objetividad y precisión.

Capítulo 4: Conclusiones y futuras líneas de investigación

El objetivo del presente trabajo era proponer un método más eficiente y automático, mediante el empleo de técnicas de *Machine Learning*, que pueda ayudar a las empresas a mejorar y acelerar la detección de las infracciones que sufran sus marcas online, para que así puedan minimizar el perjuicio económico y reputacional que conllevan.

De entre las técnicas de *Machine Learning*, en el campo de la propiedad industrial son especialmente relevantes las redes neuronales, en concreto las redes neuronales profundas (conocido en inglés como *Deep Learning*), y entre ellas las convolucionales siamesas.

En relación con las redes neuronales artificiales, éstas están compuestas por tres tipos diferentes de capas: de entrada, oculta y de salida. Cuando hay más de dos capas ocultas se entiende generalmente que se trata de una red neuronal profunda. En cuanto a las fases de las redes neuronales, estas tienen dos: una primera de entrenamiento, en la cual se ajusta, mediante múltiples reiteraciones, los pesos de las conexiones, y una segunda fase de predicción. En cuanto a las redes neuronales profundas (DNN) es una red neuronal artificial compuesta por más de dos capas ocultas entre las capas de entrada y de salida. El método que más se usa para optimizar la red neuronal profunda es el método de descenso de gradiente, que es un algoritmo de optimización que se emplea en los entrenamientos para minimizar el error de predicción, lo cual depende de la calidad de sus ponderaciones y sesgos. Existen distintos tipos de redes neuronales profundas, entre las que destacan las redes de creencias profundas, las redes generativas antagónicas, las redes neuronales recurrentes y las redes neuronales convolucionales.

En concreto, interesan especialmente las redes neuronales profundas convolucionales (CNN), pues se usan abundantemente en el reconocimiento automático de imágenes y de visión artificial. Así pues, los datos de entrada son generalmente imágenes. En relación a su estructura y funcionamiento, se debe destacar lo siguiente. En primer lugar, la capa de entrada se compone de un número de neuronas equivalente al número de pixeles de la imagen. En segundo lugar, la capa de convolución agrupa los píxeles cercanos y multiplica cada uno de los valores del Kernel por el valor del pixel que se encuentra en la misma posición. En tercer lugar, se realiza un submuestreo de la matriz resultante de la

capa convolucional para reducir el número de píxeles, tanto en su altura como en su anchura, pero conservando los de su profundidad, y así reducir la carga procesal, que tiende a ser altísima. Finalmente, una vez que, mediante la repetición de la combinación de capas convolucionales y submuestreos, se obtenga un tamaño de matriz que conserve la información relevante de la imagen y a la vez sea manejable, se pasa a una capa habitual *feedforward* de CNN que resultará en la predicción deseada.

Asimismo, es muy relevante mencionar la existencia de un tipo de redes neuronales convolucionales, denominado siamesas, por ser las que se aplican cuando se quiere comparar dos imágenes y predecir un porcentaje de similitud. En esencia es una red neuronal compuesta por dos partes, primero una combinación de dos redes neuronales convolucionales paralelas, con los mismos pesos y sesgos, y segundo una red neuronal multicapa en la que se realiza la comparación, mediante el cálculo de la distancia Euclidea entre ambas imágenes, y se devuelve el porcentaje de similitud.

Finalmente, tras haber realizado una revisión de la literatura pertinente sobre las técnicas de *Machine Learning* aplicables, se ha materializado la utilidad de las redes neuronales profundas convolucionales siamesas mediante la realización de un caso práctico en Python a modo de demostración. Así, en primer lugar, se ha utilizado la técnica de Webscrapping de imágenes (en concreto, *Chrome Image Downloader*) para descargar todas las imágenes de mochilas de esta colección de las páginas web de Asos y Nike. En segundo lugar, se han utilizado las herramientas de Keras y TensorFlow en Python para comparar una mochila de Nike con otra de Asos. El algoritmo devuelve el grado de similitud entre ellas. Se ha realizado el caso práctico íntegramente en Python, utilizando TensorFlow y Keras como vehículos.

En definitiva, las redes neuronales convolucionales siamesas son una herramienta efectiva, que proporciona grandes beneficios para la detección de infracción de marcas online y la lucha contra la violación de los derechos de propiedad industrial de las empresas. No obstante, se observan aun áreas de mejora que se pueden desarrollar en un futuro. En primer lugar, sería conveniente reducir aun más el coste computacional y el tiempo de procesamiento del modelo, que aun es considerablemente largo. Asimismo, para desarrollar este proyecto a gran escala, (realizando Webscrapping de imágenes de muchas páginas web de competidores) se requiere una gran capacidad de almacenamiento

en el disco duro, y la carga computacional es muy alta. Por tanto, el desarrollo de alternativas al Webscrapping de imágenes es indudablemente una línea de investigación futura de gran interés y utilidad. Por todo ello, se puede concluir que las redes neuronales convolucionales siamesas son una herramienta que ha mostrado ser útil para la detección de infracción de marcas y que tiene un alto potencial de desarrollo.

Bibliografía

Barrios, J. (2021). *Redes neuronales convolucionales.* Disponible en: https://www.juanbarrios.com/redes-neurales-convolucionales/#Como_estan_construidas_y_como_funcionan

Bastidas, M. (2020). *Redes Siamesas: aplicación de deep learning para la validación de identidad.* Disponible en: https://reconoserid.com/redes-siamesas-aplicacion-de-deep-learning-para-la-validacion-de-identidad/

Becker, S. (1991). *Unsupervised Learning Procedures for Neural Networks*. Interntional Journal of Neural Systems, 2, 17-33. Disponible en: https://www.researchgate.net/publication/220117478_Unsupervised_Learning_Procedures_for_Neural_Networks

Buhigas, J. (2018). *TensorFlow, la plataforma para Inteligencia Artificial de Google*. Disponible en: https://puentesdigitales.com/2018/02/14/todo-lo-que-necesitas-saber-sobre-tensorflow-la-plataforma-para-inteligencia-artificial-de-google/

CódigoFuente. (2019). *Redes neuronales profundas – Tipos y Características.* Disponible en: https://www.codigofuente.org/redes-neuronales-profundas-tipos-caracteristicas/

Das, A. (2020). Convolution Neural Network for Image Processing — Using Keras. Disponible en: https://towardsdatascience.com/convolution-neural-network-for-image-processing-using-keras-dc3429056306

Digital Guide Ionos (2020). *Keras: biblioteca de código abierto para crear redes neuronales.* Disponible en: https://www.ionos.es/digitalguide/online-marketing/marketing-para-motores-de-busqueda/que-es-keras/

Durán, J. (2019). *Descenso del Gradiente Aplicado a Redes Neuronales*. Disponible en: https://medium.com/metadatos/todo-lo-que-necesitas-saber-sobre-el-descenso-del-gradiente-aplicado-a-redes-neuronales-19bdbb706a78

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning (Adaptive Computation and Machine Learning series)*. MIT Press.

Gulli, A., & Pal, S. (2017). *Deep Learning with Keras*. Packt Publishing Ltd.

Kröse, B., & Smagt, P. (1996). *An introduction to neural networks.* Disponible en: https://www.infor.uva.es/~teodoro/neuro-intro.pdf

Leban, J. (2020). *Image recognition with Machine Learning on Python, Convolutional Neural Network*. Disponible en: https://towardsdatascience.com/image-recognition-with-machine-learning-on-python-convolutional-neural-network-363073020588

Lee, J. (2019). *Autoencoders: Neural Networks for Unsupervised Learning*. Disponible en: https://medium.com/intuitive-deep-learning/autoencoders-neural-networks-for-unsupervised-learning-83af5f092f0b

Microsoft. (2018). *Algoritmo de árboles de decisión de Microsoft.* Disponible en: https://docs.microsoft.com/es-es/analysis-services/data-mining/microsoft-decision-trees-algorithm?view=asallproducts-allversions

Molina, M. (2020). *La distancia más corta. El método de los mínimos cuadrados.* Disponible en: https://anestesiar.org/2020/la-distancia-mas-corta-el-metodo-de-los-minimos-cuadrados/

Recuero de los Santos, P. (2017). *Tipos de aprendizaje en Machine Learning: supervisado y no supervisado.* Disponible en: https://empresas.blogthinkbig.com/que-algoritmo-elegir-en-ml-aprendizaje/

Rodríguez, D. (2018). *Implementación del método descenso del gradiente en Python*. Disponible en: https://www.analyticslane.com/2018/12/21/implementacion-del-metodo-descenso-del-gradiente-en-python/

Rosebrock, A. (2020). *Siamese networks with Keras, TensorFlow, and Deep Learning*. Disponible en: https://www.pyimagesearch.com/2020/11/30/siamese-networks-with-keras-tensorflow-and-deep-learning/

Sancho, F. (2018). *Métodos combinados de aprendizaje*. Disponible en: http://www.cs.us.es/~fsancho/?e=106

Simeone, O. (2018). *A Very Brief Introduction to Machine Learning With Applications to Communication Systems.* Disponible en: https://arxiv.org/pdf/1808.02342.pdf

Soria, E., & Blanco, A. (2001). *Redes neuronales artificiales*. Disponible en: https://www.acta.es/medios/articulos/informatica_y_computacion/019023.pdf

Anexo: Código en Python

Código en Python, utilizando TensorFlow y Keras, para llevar a cabo el reconocimiento de imágenes

```
# Importar los paquetes necesarios
import os
# Especificar las dimensiones de la imagen
Nike_SHAPE = (592, 592, 2)
Asos_SHAPE = (476, 608, 2)
# Especificar el número y tamaño de iteraciones
tamañoiteraciones = 32
numeroiteraciones = 10

# Definir la ruta del directorio de salida base
BASE_OUTPUT = "output"

# Usar la ruta de salida base para derivar la ruta al modelo serializado junto con el gráfico del historial de entrenamiento

MODEL_PATH = os.path.sep.join([BASE_OUTPUT, "siamese_model"])
PLOT_PATH = os.path.sep.join([BASE_OUTPUT, "plot.png"])

# Importar los paquetes necesarios
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import GlobalAveragePooling2D
from tensorflow.keras.layers import MaxPooling2D

def build_siamese_model(inputShape, embeddingDim=48):
    # Especificación de las entradas para la red del extractor de características
    inputs = Input(inputShape)

    # Definir la primera capa convolucional + función de activación ReLU + submuestreo +  capas de salida
    x = Conv2D(64, (2, 2), padding="same", activation="relu")(inputs)
    x = MaxPooling2D(pool_size=(2, 2))(x)
    x = Dropout(0.3)(x)

    # Definir la segunda capa convolucional + función de activación ReLU + submuestreo +  capas de salida
    x = Conv2D(64, (2, 2), padding="same", activation="relu")(x)
    x = MaxPooling2D(pool_size=2)(x)
    x = Dropout(0.3)(x)
```

```
    # Preparar los resultados finales
    pooledOutput = GlobalAveragePooling2D()(x)
    outputs = Dense(embeddingDim)(pooledOutput)

    # Construir el modelo
    model = Model(inputs, outputs)

    # Devolver el modelo cuando lo pida la función definida
    return model #este paquete permite construir el modelo de CNN siamesas


# Importar los paquetes necesarios
import tensorflow.keras.backend as K
import matplotlib.pyplot as plt
import numpy as np

# Esta función se define para establecer todo lo relativo a la carga de las imágenes en cuestión
def make_pairs(images, labels):
    # Crear dos listas vacías que incluyan las dos imágenes a comparar y etiquetas mostrando si la pareja es positiva o negativa, es decir, si pertenece a la misma categoría o no. Generalmente se considera que si dos imagenes tienen una similitud mayor a 0.5 pertenecen a la misma clase y si es menor de 0.5 a diferentes categorías.
    pairImages = []
    pairLabels = []

    # Calcular el número total de clases presentes en el conjunto de datos
    # para después construir una lista de índices para cada etiqueta de clase que ofrece los índices de todos los ejemplos con una etiqueta determinada
    numClasses = len(np.unique(labels))
    idx = [np.where(labels == i)[0] for i in range(0, numClasses)]

    # Cargar las imágenes
    for idxA in range(len(images)):
        # Para coger la imagen y la etiqueta que pertenece a la iteración
        currentImage = images[idxA]
        label = labels[idxA]

        # Elegir al azar una imagen que pertenezca a la misma clase (etiqueta)
        idxB = np.random.choice(idx[label])
        posImage = images[idxB]

        pairImages.append([currentImage, posImage])
        pairLabels.append([1])

        negIdx = np.where(labels != label)[0]
        negImage = images[np.random.choice(negIdx)]

        pairImages.append([currentImage, negImage])
        pairLabels.append([0])

    # Devolver el modelo cuando lo pida la función definida
```

```
    return (np.array(pairImages), np.array(pairLabels))

def euclidean_distance(vectors):
    # Separar los vectores en listas diferentes
    (featsA, featsB) = vectors

    # La suma de la distancia al cuadrado entre los vectores
    sumSquared = K.sum(K.square(featsA - featsB), axis=1,
        keepdims=True)

    # Devolver el modelo cuando lo pida la distancia entre las imágenes
    return K.sqrt(K.maximum(sumSquared, K.epsilon()))

def plot_training(H, plotPath):
    # Crear una gráfica que muestre la historia del entrenamiento
    plt.style.use("ggplot")
    plt.figure()
    plt.plot(H.history["loss"], label="train_loss")
    plt.plot(H.history["val_loss"], label="val_loss")
    plt.plot(H.history["accuracy"], label="train_acc")
    plt.plot(H.history["val_accuracy"], label="val_acc")
    plt.title("Training Loss and Accuracy")
    plt.xlabel("Epoch #")
    plt.ylabel("Loss/Accuracy")
    plt.legend(loc="lower left")
    plt.savefig(plotPath)
    #La función 'utils' permite computar la distancia Euclidea, entre otras cosas

from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Input
from tensorflow.keras.layers import Lambda #Introduce la distancia Euclidea en la CNN siamesa
import numpy as np
import cv2

# Cargar las dos imágenes que se desea comparar y normalizar el valor de sus píxeles entre [0, 1]
nikeimage = cv2.imread('/Users/belenarmada/Desktop/Nike/mochila-sportswear-heritage-G5Wt0h.jpeg')
asosimage = cv2.imread('/Users/belenarmada/Desktop/Asos/14054054-1-black.jpeg')

trainX = nikeimage / 255.0
trainY = asosimage / 255.0

trainX = np.expand_dims(trainX, axis=-1)
trainY = np.expand_dims(trainY, axis=-1)

(pairTrain, labelTrain) = make_pairs(nikeimage, asosimage)
(pairTest, labelTest) = make_pairs(nikeimage, asosimage)

# Configurar la CNN siamesa
nikeimg = Input(shape=Nike_SHAPE)
asosimg = Input(shape=Asos_SHAPE)
```

```
featureExtractornike = build_siamese_model(Nike_SHAPE)
featureExtractorasos = build_siamese_model(Asos_SHAPE)
featsA = featureExtractornike(nikeimg)
featsB = featureExtractorasos(asosimg)

# Construir la CNN siamesa
distance = Lambda(euclidean_distance)([featsA, featsB])
outputs = Dense(1, activation="sigmoid")(distance)
model = Model(inputs=[nikeimg, asosimg], outputs=outputs)

# Compilar el modelo
model.compile(loss="binary_crossentropy", optimizer="adam",
metrics=["accuracy"])

# Entrenar el modelo
history = model.fit([pairTrain[:, 0], pairTrain[:, 1]], labelTrain[:], validation_data=([pairTest[:, 0], pairTest[:,
1]], labelTest[:]), tamañoiteraciones = config.tamañoiteraciones, numeroiteraciones =
config.numeroiteraciones)

# Guardar el modelo
model.save(MODEL_PATH)

# Representar graficamente el entrenamiento
plot_training(history, PLOT_PATH)

# Importar los paquetes necesarios
from tensorflow.keras.models import load_model
from imutils.paths import list_images
import matplotlib.pyplot as plt
import numpy as np
import argparse
import cv2

# Construir y analizar los argumentos
ap = argparse.ArgumentParser()
ap.add_argument("-i", "--input", required=True,
   help="path to input directory of testing images")
args = vars(ap.parse_args())

# Cargar el modelo
model = load_model(MODEL_PATH)

# Cargar las imágenes
for (i, (pathA, pathB)) in enumerate(pairs):
   nikeimage = cv2.imread(pathA, 0)
   asosimage = cv2.imread(pathB, 0)

   # Crear una copia de ambas imágenes para su visualización
   origA = nikeimage.copy()
   origB = asosimage.copy()
```

```
    nikeimage = np.expand_dims(nikeimage, axis=-1)
    asosimage = np.expand_dims(asosimage, axis=-1)

    nikeimage = np.expand_dims(nikeimage, axis=0)
    asosimage = np.expand_dims(asosimage, axis=0)

    # Escalar
    nikeimage = nikeimage / 255.0
    asosimage = asosimage / 255.0

    # Utilizar el modelo para hacer predicciones sobre la similitud entre ambas imágenes, y con ello
sugiriendo o no si se ha producido una infracción de marcas
    preds = model.predict([nikeimage, asosimage])
    proba = preds[0][0]

    # Crear la figura
    fig = plt.figure("Pair #{}".format(i + 1), figsize=(4, 2))
    plt.suptitle("Similarity: {:.2f}".format(proba))

    # Mostrar la primera imagen
    ax = fig.add_subplot(1, 2, 1)
    plt.imshow(origA, cmap=plt.cm.gray)
    plt.axis("off")

    # Mostrar la segunda imagen
    ax = fig.add_subplot(1, 2, 2)
    plt.imshow(origB, cmap=plt.cm.gray)
    plt.axis("off")

    # Mostrar el gráfico
    plt.show()
```