Clasificador de Vestimenta basado en Redes Neuronales

Grado en Ingeniería Informática

Roberto Díaz Badra

Junio 2018

Tutores

José Javier Lorenzo Navarro

Modesto Catrillón Santana

UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA
Escuela de Ingeniería Informática

TFT04

SOLICITUD DE DEFENSA DE TRABAJO DE FIN DE TÍTULO

D/Dª Roberto Díaz Badra, autor del Trabajo de Fin de Título Clasificador de Vestimenta basado en Redes Neuronales, correspondiente a la titulación Grado en Ingeniería Informática, en colaboración con la empresa/proyecto (indicar en su caso) _____

S O L I C I T A

que se inicie el procedimiento de defensa del mismo, para lo que se adjunta la documentación requerida.

Asimismo, con respecto al registro de la propiedad intelectual/industrial del TFT, declara que:

[ ] Se ha iniciado o hay intención de iniciarlo (defensa no pública).
[X] No está previsto.

Y para que así conste firma la presente.

Las Palmas de Gran Canaria, a 1 de Junio de 2018.

El estudiante

Fdo.: _____

A rellenar y firmar **obligatoriamente** por el/los tutor/es

En relación a la presente solicitud, se informa:

[X] Positivamente

[ ] Negativamente
(la justificación en caso de informe negativo deberá incluirse en el TFT05)

Firmado por LORENZO NAVARRO JOSE JAVIER - 42840825J el día 01/06/2018 con un certificado emitido por

Fdo.: _____

Firmado por CASTRILLON SANTANA MOSESTO FERNANDO - 52856297C el día 01/06/2018

DIRECTOR DE LA ESCUELA DE INGENIERÍA INFORMÁTICA

Índice general

Índice de figuras

Agradecimientos

Antes de todo, agradecer...

A mis padres y mi hermano, ya que sin su constancia y apoyo no habría llegado hasta donde estoy.

A mis tutores, por enseñarme el camino.

A Daniel Reyes Parrilla, por nuestra discusiones frikis.

A Virginia, por sus minuciosas correcciones.

Clasificador de Vestimenta basado en Redes Neuronales

El presente proyecto tiene como objetivo, determinar cómo se pueden utilizar las redes neuronales a la hora de clasificar correctamente entre distintos tipos de prendas. Para este enfoque, se ha utilizado una base de datos con miles de imágenes, donde cada imagen está etiquetada con su categoría correspondiente.

Se emplea una metodología basada en prototipos. Mediante la utilización de un algoritmo del estado del arte para este tipo de problemas, el cual ya ha sido entrenado previamente, se han ido realizando pequeñas variaciones para obtener los resultados óptimos.

Posteriormente, se han aplicado diferentes algoritmos de Aprendizaje Automático para acrecentar los resultados y se han evaluado los resultados, proponiendo posibles mejoras para próximos proyectos.

Clothing Classifier based on Neural Networks

The current project has as objective, to determine how to use neural networks to classify different types of clothes. For this approach, we have used a database containing thousands of images, where each image contains a label with the correspondent category.

We apply a methodology based on prototypes. With the use of a state of the art algorithm for this kind of problems, we have been doing small alterations to obtain the optimum results.

Subsequently, we have applied different Machine Learning algorithms to enhance the performance. We also evaluate the results and propose possible improvements for future works.

Capítulo 1

Descripción del proyecto

1.1. Introducción

Desde sus orígenes, el ser humano siempre ha querido mejorar su desempeño, y es ahí donde surge la tecnología. La tecnología, sirve, no solo para aumentar el rendimiento en ciertas tareas sino también para automatizarlas, y de esta forma poder dedicar nuestra atención a otras más importantes.

Con este objetivo de automatizar tareas, surge el campo de la Inteligencia Artificial, y más concretamente, un subcampo de este denominado Aprendizaje Automático ('Machine Learning' en inglés). Este subcampo dispone de numerosas aplicaciones, sin embargo, en este proyecto nos centraremos en los problemas de clasificación.

El objetivo de una tarea de clasificación consiste en mediante unos datos de entrada y la ejecución de algún algoritmo, que pueda aprender las características más importantes de estos datos, generar una salida que catalogue correctamente esos datos. Por ejemplo: Si tenemos una imagen de un gato, y tenemos un algoritmo que pueda diferenciar entre imágenes de perros y gatos, una correcta ejecución de este algoritmo debería ser capaz de inferir que la imagen de entrada se trata de la de un gato, con un porcentaje relativamente alto de acierto.

En este proyecto, nos centramos en este tipo de algoritmos, denominados clasificadores de imágenes, y específicamente nos enfocamos en las llamadas Redes Neuronales Convolucionales, debido a su gran éxito a la hora de llevar a cabo estas tareas.

1.2. Descripción del trabajo

Debido a la ingente cantidad de fotografías y vídeos que diariamente se suben a diferentes plataformas como pueden ser Youtube, Vimeo, Pinterest, Google Fotos o Flickr; se hace cada vez más patente la necesidad de contar con herramientas que permitan analizar toda esa información. Aunque en muchos casos esta información no textual se acompaña de metadatos en forma de etiquetas (tags), no siempre está garantizada su existencia y veracidad, ya que suele recaer en el usuario la anotación.

En esta línea, la anotación automática de imágenes y vídeos, es donde se plantea este trabajo de fin de grado. Así el trabajo ha consistido en la implementación de un clasificador de prendas de vestir basado en redes neuronales, que han demostrado ser un método eficaz en otras tareas de clasificación visual. Para realizar esta tarea, se ha partido de imágenes o fotogramas de un vídeo y mediante un análisis previo se detectará la presencia de personas y a partir de ahí se intentará determinar qué tipos de prendas lleva dicha persona. Las técnicas a utilizar proceden de la Visión por Computador, como son la detección de personas, o la pose de las mismas. A partir del resultado de la detección de la persona se ha utilizado una Red Neuronal Convolucional para realizar la clasificación de las prendas que lleva.

1.3. Objetivos del trabajo

El objetivo de este proyecto, es la implementación de un algoritmo que pueda diferenciar correctamente entre tipos de prendas, mediante el uso de algoritmos de Aprendizaje Automático. Siendo más específicos, los objetivos de este trabajo de fin de grado son:

- Realizar una base de datos con imágenes de personas llevando diferentes tipos de prendas.
- Seleccionar e integrar un detector de personas en imágenes.
- Seleccionar e integrar un detector de poses.
- Implementar diferentes topologías de redes neuronales y estudiar su rendimiento.

1.4. Metodología

La metodología a utilizar para el desarrollo de este proyecto será una basada en prototipos, debido a que por un lado se deben integrar métodos provenientes de distintas librerías, (detector de personas o de pose) y por otro lado el desarrollo del propio clasificador, con su proceso de entrenamiento que se debe realizar previamente al uso del mismo.

Por tanto, se deberán desarrollar por un lado diferentes prototipos que vayan incorporando las funcionalidades a la aplicación, y por otro, se necesitan aplicaciones auxiliares para realizar los procesos de entrenamiento de las redes neuronales, así como de preprocesado de los datos para obtener los conjuntos de datos necesarios para entrenar y evaluar dichas redes. En cuanto a las herramientas que se emplearán, indicar que por un lado se hará uso de librerías de Visión por Computador de código abierto como puede ser OpenCV o Dlib y por otro el desarrollo de las redes neuronales se realizará en el entorno Keras que es una capa que facilita el desarrollo de este tipo de métodos usando como base Tensorflow.

1.5. Justificación de la competencia

En este trabajo, se ha cubierto la siguiente competencia específica de la mención de Computación del Grado de Ingeniería Informática:

CP04 : Capacidad para conocer los fundamentos, paradigmas y técnicas propias de los sistemas inteligentes y analizar, diseñar y construir sistemas, servicios y aplicaciones informáticas que utilicen dichas técnicas en cualquier ámbito de aplicación.

Esta competencia ha sido cubierta, al hacer un estudio del funcionamiento de las Redes Neuronales Convolucionales para tareas de clasificación. Después, se creó un sistema que pudiera diferenciar correctamente entre varios tipos de prendas.

Capítulo 2

Visión por computador

2.1. En qué consiste

El ser humano observa el mundo que le rodea a través del sentido de la vista. El sentido de la vista es aquel que nos permite detectar la energía electromagnética gracias a la luz visible que entra a través del ojo, seguidamente se le envía una señal al cerebro para que así esa imagen pase a ser vista. En efecto, desde el punto de vista biológico, los ojos constituyen prácticamente el principal punto de contacto de nuestra mente con el exterior.

Una persona puede observar un objeto, digamos, una silla, e inmediatamente deducir que se trata de una silla. Para una máquina, sin embargo, no es tan sencillo. La finalidad de la Visión por Computador, es desarrollar sistemas inteligentes con una capacidad similar o superior a la de los humanos, es decir, crear sistemas inteligentes que sepan reaccionar con el entorno que pueden ver, o mejor dicho, que puedan percibir y entender el entorno que les rodea y poder reaccionar acorde a las necesidades del momento.

El campo de la Visión por Computador, a su vez tiene diversos subcampos, tales como la clasificación de imágenes (tarea en la que se centra nuestro proyecto), es decir, poder designar en que categoría se encuentra el objeto de la imagen, o la detección de objetos, dicho de otra forma, localizar y etiquetar todos los objetos que se encuentran en la imagen.

2.2. Aplicaciones

Actualmente los algoritmos de visión por computador son ampliamente utilizados

en el campo de la conducción autónoma, entorno que parece que ha dado lugar a la mayor notoriedad debido a su alta exposición en las redes sociales y en la prensa.

Otra de las aplicaciones de la visión artificial, que más beneficio podría acarrearle a la humanidad, es la detección temprana de enfermedades. Como se puede apreciar, este campo de la inteligencia artificial parece ser el futuro para gran cantidad de entornos.

2.3. Procesamiento de Imágenes

El procesamiento de imágenes, es el paso anterior al de la visión por computador. El objetivo del procesamiento de imágenes, consiste en extraer las características esenciales de una imagen, tales como esquinas o líneas. Los métodos de extracción de características, se pueden clasificar en dos tipos, descriptores clásicos y métodos con aprendizaje de características. Normalmente, un extractor de características, coge la entrada de una imagen y devuelve un vector de características.

La potencia de cualquier sistema visual, radica, en la calidad de la relación entre extractor de características y clasificador, del cual hablaremos más adelante. Debido a esta relación, podrían haber extractores de características, que proporcionaran las características óptimas, y que no hiciera falta un clasificador perfecto, o podría darse el caso inverso, en que el extractor no fuera muy sofisticado e hiciera falta un clasificador más eficiente. No obstante, no existen ni extractores de características, ni clasificadores perfectos. A continuación se hablará de dos de los descriptores más utilizados a lo largo de los años.

2.3.1. Histograma de Gradientes Ordenados

Histograma de Gradientes Ordenados [Dalal and Triggs, 2005], se trata de un descriptor de características utilizado para la detección de objetos. La idea tras este algoritmo radica en que la forma de un objeto puede ser descrita por medio del histograma de dirección de los gradientes.

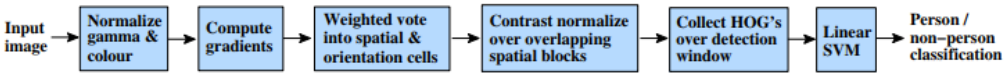

Figura 2.1: La estructura de la cadena del descriptor de características y detección de objetos propuesta por Dalal and Triggs. Imagen sacada de [Dalal and Triggs, 2005]

2.3.2. Transformada de Características Invariante a Escalas

Transformada de Características Invariante a Escalas, o Scale-Invariant Feature Transform en inglés [Lowe, 2004], proporciona un conjunto de características que son robustas ante rotaciones, cambios de tamaño de la imagen y son parcialmente invariantes a cambios en iluminación y ángulos de cámara 3D.

Están muy bien localizadas tanto en dominios espaciales como de frecuencia, reduciendo así la probabilidad de disrupción por oclusión. Gran número de características pueden ser extraídas de imágenes típicas con algoritmos eficientes.

Adicionalmente, las características son muy distintivas, lo que permite que una sola característica sea correctamente identificada, con alta probabilidad ante una gran base de datos de características, permitiendo así una base para reconocimiento de objetos.

2.4. Clasificadores

Los clasificadores, son el núcleo central de los algoritmos de aprendizaje automático. La finalidad de un clasificador, en el campo de la Visión por Computador, es usar el vector de características para proporcionar una categoría a una imagen.

El campo del Aprendizaje Automático, se divide en tres categorías: aprendizaje supervisado, aprendizaje no supervisado y aprendizaje por refuerzo. En el caso del aprendizaje supervisado, un algoritmo es entrenado con un conjunto de datos etiquetados, para conseguir mapear correctamente una entrada x, con una salida y. En aprendizaje no supervisado, sin embargo, los datos de entrenamiento, no se encuentran etiquetados, por lo que la tarea del algoritmo, es encontrar patrones en los datos. En aprendizaje por refuerzo, el algoritmo permite, que un agente inteligente, interaccione libremente con su entorno, para que aprenda de forma autónoma a interactuar con él. En este caso el entorno proporciona algún tipo de recompensa

al sistema.

El proyecto se ha centrado, en algoritmos de aprendizaje supervisado, ya que se disponía de un conjunto de datos previamente etiquetados, DeepFashion, el cual se mencionará más adelante. A continuación se hablará, de dos de los algoritmos tradicionales más utilizados a lo largo de los años.

2.4.1. Máquinas de Soporte Vectorial

Las Máquinas de Soporte Vectorial, más conocidas como SVM [Vapnik, 2013], son un algoritmo de aprendizaje supervisado utilizado tanto para problemas de clasificación como de regresión. La idea principal de las SVM es que dado un conjunto de puntos, encuentra un hiperplano que separa el conjunto de entrenamiento en dos clases. Se busca el hiperplano que posee la distancia máxima entre estas clases, por ello, las SVM también son conocidas como clasificadores de margen máximo.

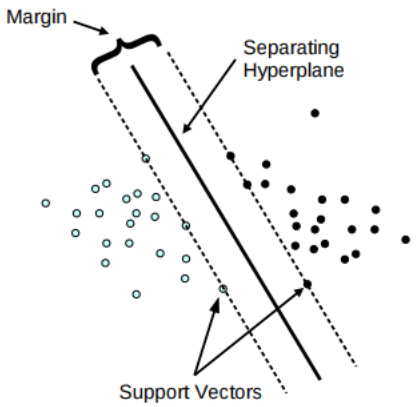

Figura 2.2: Caso lineal de una SVM. Imagen sacada de [Meyer and Wien, 2001]

2.4.2. Bosques Aleatorios

Los Bosques Aleatorios, más conocidos como Random Forests [Breiman, 1999], son una combinación de árboles de decisión, otro algoritmo de clasificación. Un árbol de decisión, es un algoritmo muy intuitivo de entender. Se trata de un grafo que comienza en el nodo raíz. El árbol recorre una de sus hojas, dependiendo del valor de entrada. De esta forma va recorriendo el árbol completo hasta dar con una

solución final.

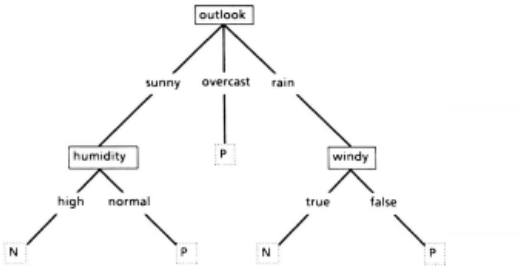

Figura 2.3: Ejemplo de un árbol de decisión sencillo. Imagen sacada de [Quinlan, 1986].

Los Bosques Aleatorios son una combinación de árboles de decisión, de tal forma, que cada árbol depende de los valores de un vector aleatorio, con la misma distribución para todos los árboles en el bosque. El error de generalización para los bosques, converge en un límite a medida que el número de árboles en el bosque se incrementa. El error de generalización de un bosque de árboles clasificadores, depende de la fuerza de los árboles individuales en el bosque y de la correlación entre ellos [Breiman, 2001].

Capítulo 3

Aprendizaje Profundo

Si habláramos actualmente de Aprendizaje Automático, seguramente estaríamos haciendo referencia al subcampo de las redes neuronales, conocido como Aprendizaje Profundo, o Deep Learning en inglés. Este campo ha cogido especial aceptación en los últimos años, debido a la gran mejoría de las redes neuronales, con respecto a otros algoritmos. La gran diferencia en precisión, posiciona a estas redes, a la cima de la pirámide de algoritmos disponibles para el aprendizaje automático.

A pesar de la gran diferencia de rendimiento, con respecto al resto de algoritmos, las redes neuronales necesitan una gran potencia de cómputo. Es por esto que a pesar de que existen desde hace muchos años, no han sido especialmente relevantes hasta hace poco, ya que la potencia de cómputo existente no era la suficiente. Gracias al desarrollo de los procesadores, y especialmente al de las tarjetas gráficas, se ha podido observar el increíble potencial del Aprendizaje Profundo, poniendo estos algoritmos en boca de todos, y utilizándolos en todas las aplicaciones posibles, teniendo un notorio éxito tanto en Procesamiento de Lenguaje Natural, como en Visión por Computador.

Las redes neuronales se dieron a conocer gracias a una de sus variantes denominada Red Neuronal Convolucional, Convolutional Neural Network [LeCun et al., 1995], que demostró ser capaz de reconocer con un alto porcentaje de acierto, carácteres escritos del 0 al 9.

El Aprendizaje Profundo tiene varias descripciones de alto nivel muy conocidas, como que se trata de un subcampo del Aprendizaje Automático, que está basado en aprender varios niveles de presentación, correspondientes a una jerarquía de características, donde conceptos de más alto nivel, son definidos desde algunos de más bajo nivel, y los mismos conceptos de bajo nivel, pueden ayudar a definir algunos de más alto nivel [Deng et al., 2014].

3.1. Redes neuronales con prealimentación

Las redes neuronales están inspiradas por el funcionamiento del cerebro humano. Una red neuronal se compone de neuronas, las cuales como cualquier algoritmo de aprendizaje automático solo buscan aprender a transformar una entrada de datos en una determinada salida. Se llama red debido a que diversas neuronas se conectan entre sí para formar una red neuronal.

El objetivo de una red con prealimentación, es aproximar alguna función $f*$. Por ejemplo, para un clasificador, $y = f*(x)$ que mapea una entrada x a una categoría y. Una red con prealimentación define un mapeo $y = f(x; \theta)$ y aprende el valor de los parámetros θ que tienen como resultado la mejor aproximación a la función, [Goodfellow et al., 2016], capítulo 6. Estos modelos se dicen con prealimentación ya que la información fluye a través de la función que está siendo evaluada desde x, a través de las computaciones intermedias usadas para definir f, y finalmente a la salida y.

3.1.1. Perceptrón Mono capa

El ejemplo más sencillo de red neuronal es una perceptrón monocapa [Rosenblatt, 1958], la cual está formada por una sola neurona. La entrada de datos se multiplica con una serie de pesos. A la sumatoria de estas entradas por los pesos se le añade un bias. Este resultado, pasa por una función de activación no lineal, esto quiere decir que la neurona se activa si pasa cierto umbral, o no activándose si está por debajo.

$$Z_i = \sum W_i * X_i + b_i$$

$$A_i = f(Z_i)$$

3.1.2. Perceptrón Multi Capa

Red Perceptrón Multi Capa, Multi-layer Perceptron [Svozil et al., 1997]. Está formada por neuronas que a su vez están ordenadas por capas. La primera capa es

la capa de entrada, las capas intermedias se denominan capas ocultas y la última capa se llama capa exterior. Todas las neuronas de una capa están conectadas con todas las neuronas de la capa siguiente.

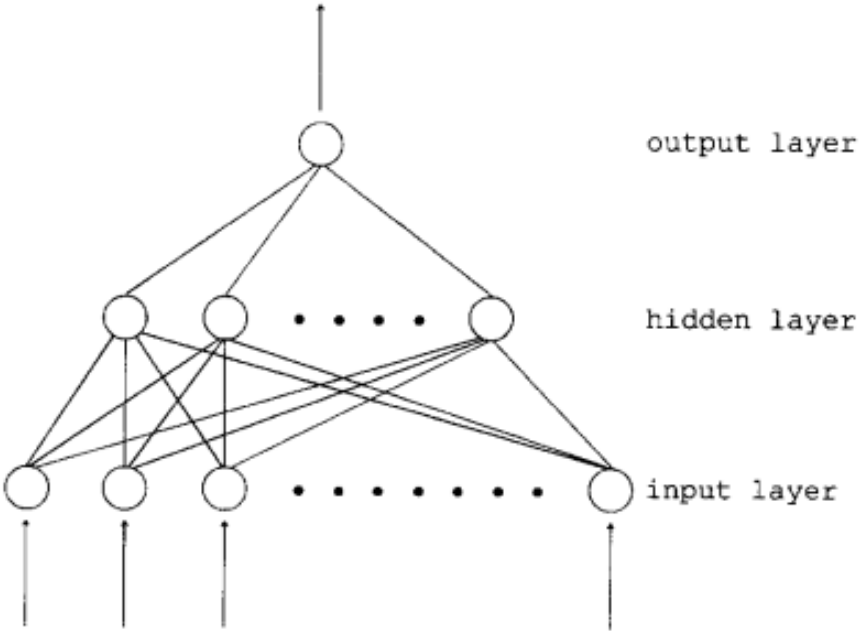

Figura 3.1: Ejemplo de la estructura de una Perceptrón Multi Capa. Imagen sacada de Svozil et al. [1997]

Las neuronas de una capa están conectadas con las de la siguientes capas mediante unos pesos w. El valor del coeficiente w muestra el grado de relevancia de esa conectividad de la red. El valor de la salida de la neurona de la capa posterior viene determinado por las siguientes funciones:

$$Z_i = \sum W_i * X_i + b_i$$

$$A_i = f(Z_i)$$

En las funciones anteriores, W_i representa los pesos de las capas, X_i las entradas de las capas y b_i representa el bias. La unidad bias, se puede interpretar como un valor que se añade a la función para entrenar el modelo para que pueda predecir

mejor los datos.

La salida de las neuronas es transmitida como entrada a las neuronas de la siguiente capa. El objetivo de la red neuronal es modificar el valor de los pesos, para así minimizar el valor de la función objetivo, que puede venir determinada por la diferencia de la suma del cuadrado de las diferencias entre el valor computado y el valor esperado.

$$E = \sum \frac{1}{2}(y - \hat{y})^2$$

3.2. Redes Neuronales Profundas

Las redes neuronales profundas o deep neural networks, DNN, permiten la existencia de modelos computaciones compuestos por múltiples capas de procesamiento que pueden aprender representaciones de los datos con múltiples niveles de abstracción, [LeCun et al., 2015]. Las DNN son capaces de encontrar una estructura entre un gran conjunto de datos a través del algoritmo de retropropagación comentado anteriormente, para indicar como un modelo debe modificar sus parámetros para inferir de esta forma las características necesarias para cada tipo de problema.

3.3. Descenso por gradiente

Descenso por gradiente es el proceso más utilizado en el campo de las redes neuronales por el cual una red neuronal adapta los pesos de sus conexiones para obtener una diferencia mínima entre el valor computado por la red y el verdadero valor que deberíamos esperar [Ramchoun et al., 2016]. Para la actualización de sus pesos se toma:

$$w_{ij}^{(k+1)} = w_{ij}^{(k)} - \lambda(\frac{\partial E}{\partial w_{ij}})^{(k)}$$

$$b_i^{(k+1)} = b_i^{(k)} - \lambda(\frac{\partial E}{\partial b_i})^{(k)}$$

En las funciones anteriores λ es el ratio de aprendizaje que debe ser mayor de 0. Este método no garantiza la llegada a un mínimo global. Este proceso suele converger en mínimos locales, pero para nuestro problema tampoco es muy relevante ya que igualmente dan buenos resultados.

3.3.1. Retropropagación

Para calcular el gradiente de forma eficiente se utiliza la técnica de retropropagación. Este método consiste en calcular las derivadas requeridas de forma recurrente. Empezando por la capa exterior, y yendo de forma recursiva hasta llegar a la capa de entrada. Esto es así, ya que el error de la salida se propaga desde la capa exterior, a través de las capas ocultas, hasta la capa de entrada.

3.4. Entrenamiento

Para entrenar una red neuronal, hacen falta al menos dos conjuntos de datos, uno para entrenamiento y otro para testear. Al inicio la red neuronal comienza con valores aleatorios y avanza iterativamente hasta actualizarlos. En primer lugar realiza un feed-forward, este consiste en ejecutar las funciones hasta llegar a calcular la función objetivo. Una vez tenemos este valor se realiza el gradient descent para actualizar los valores de la red. Cada vez que se realiza una iteración de este proceso con todo el conjunto de entrenamiento, se denomina epoch. Se realizarán tantas epochs como sean necesarias para que el modelo haga predicciones de manera efectiva.

El entrenamiento se puede hacer con varios tamaños de lotes, esto quiere decir que o bien se puede entrenar todo el conjunto de entrenamiento de forma vectorizada, o bien se pueden ir actualizando los valores de la red por lotes. En lugar de entrenar todo el conjunto a la vez, se pasan por lotes de un tamaño previamente definido, hasta que todos los elementos del conjunto hayan sido reconocidos.

3.5. Redes Neuronales Convolucionales

Las redes neuronales convolucionales, o CNN, son un tipo de redes neuronales específicamente utilizadas para datos de grandes dimensiones. El nombre de red neuronal convolucional indica que se trata de una red que emplea una operación

matemática denominada convolución. La convolución es un tipo especial de operación lineal. Las redes convolucionales son simplemente redes que usan convolución en alguna de sus capas. [Goodfellow et al., 2016], capítulo 9.

Una capa típica de una red convolucional consiste de tres etapas. En la primera etapa, la capa ejecuta diversas convoluciones en paralelo para producir un conjunto de activaciones lineales. En la segunda etapa, cada activación lineal es ejecutada a través de una función de activación no lineal. Esta etapa es también conocida como etapa de detección. En la tercera etapa, utilizamos una función de submuestreo para modificar la salida de la capa.

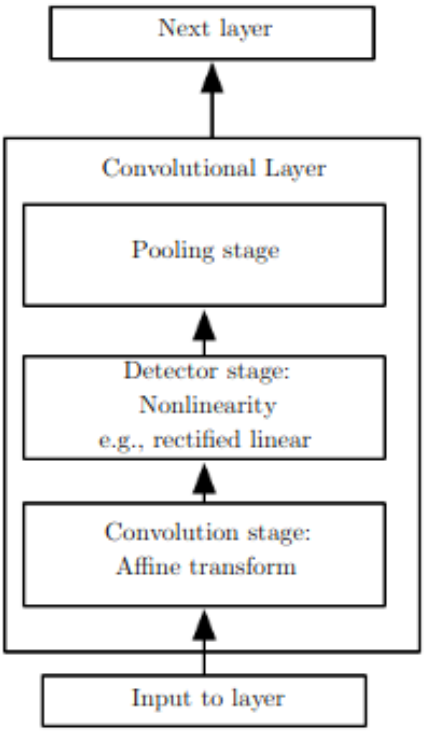

Figura 3.2: Ejemplo de una capa típica de convolución. Imagen sacada de [Goodfellow et al., 2016], capítulo 9

3.5.1. Capa Convolucional

La capa convolucional es el componente más importante de una CNN. Contiene un conjunto de filtros denominados kernels, que realizan la función de convolución

con una entrada dada para generar una salida que se trata de un mapa de características. Un filtro es una cuadrícula formada por números discretos. Los números en cada celda de la cuadrícula son los pesos del kernel y son aprendidos durante el entrenamiento de la CNN.

El objetivo del kernel es ir reduciendo las dimensiones de la entrada, la cual suele ser un array multidimensional, e ir infiriendo cuáles son las características más reseñables de los datos. La salida de este proceso de menor dimensión es lo que se conoce como mapa de características.

3.5.2. Relleno

El relleno es una técnica empleada cuando queremos mantener el tamaño de los datos constantes o incluso aumentar su tamaño. Esto se puede conseguir aplicando zero-padding, o cero relleno. La idea básica es incrementar el tamaño del mapa de características de forma que se obtiene un mapa de características con el volumen deseado.

Las convoluciones con padding son divididas en tres categorías dependiendo de la implicación de zero-padding.

- Convolución válida, es el caso más sencillo donde no se aplica ningún tipo de relleno. La salida es reducida en ancho y altura.
- Misma convolución, se asegura que la entrada y salida tengan el mismo tamaño. Se aplica zero-padding para conseguirlo.
- Convolución completa, aplica el padding máximo posible a a entrada del mapa de características antes de la convolución. El padding máximo posible, es aquel en el que al menos un valor de entrada válido se ve implicado en todos los casos de convolución.

3.5.3. Capa de Submuestreo

Esta capa trabaja en bloques de la entrada del mapa de características. Lo que hace es combinar las características, para así, decrementar las dimensiones de entrada. Esto operación de combinación se lleva a cabo con una de dos funciones, la de media o la de máximo. La de media simplemente coge el valor medio entre los escogidos, mientras que el máximo coge el valor máximo. Al reducir las dimensiones de la entrada del mapa de características, se consigue una representación de

características, que es invariante a cambios moderados en la escala del objeto, la pose y la traslación de una imagen [Goodfellow et al., 2016].

3.5.4. Función no lineal

Las capas con pesos de una CNN, suelen estar seguidos por una activación no lineal. Esta función de activación toma un valor real como entrada y lo comprime en un pequeño rango como [0, 1] y [-1, 1]. La aplicación de este tipo de función es muy relevante, ya que permite a la red neuronal aprender mapeos no lineales. Si no hubiera funciones no lineales, la unión de varias capas con pesos sería equivalente a utilizar un mapeo lineal, [Khan et al., 2018].

3.6. Aprendizaje Por Transferencia

El aprendizaje por transferencia, es una técnica utilizada en el aprendizaje profundo, por el cual en vez de entrenar específicamente una red desde cero, debido al extremo nivel de cómputo requerido que puede ser de días, se opta por tomar una red ya entrenada, para una determinada tarea y utilizarla para la nuestra. Esto funciona, ya que esa red previamente entrenada ya ha sido capaz de inferir las características correctas de un objeto, por lo que lo único que habría que hacer sería retocar las capas finales, para que esta red sea capaz de tener resultados positivos para nuestro problema.

La gran ventaja es que en lugar de necesitar cientos de miles, si no millones de datos para nuestro problema, y tener que entrenar el modelo durante días, podemos partir de muchos menos datos y mucho menos tiempo de computo y aún así tener resultados excelentes. Después, según el volumen de datos que tengamos, se pueden retocar más o menos capas.

3.7. Estado del arte

A continuación se hablará del estado del arte en el campo de la visión por computador, es decir, los algoritmos que han dado mejores resultados en los últimos años.

3.7.1. LeNet

La arquitectura LeNet [LeCun et al., 1998], es una de las CNN más básicas que fue aplicada al problema de reconocimiento de dígitos escritos. Una variante muy exitosa de esta estructura es la llamada LeNet-5, llamada así porque tiene cinco capas con pesos.

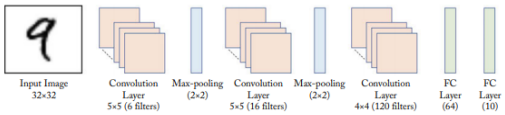

Figura 3.3: Arquitectura de LeNet-5. Imagen sacada de [Khan et al., 2018]

3.7.2. AlexNet

Alexnet, [Krizhevsky et al., 2012], fue la primera CNN a gran escala, que llevó al resurgimiento de las redes neuronales profundas en el campo de la visión por computador. Esta arquitectura ganó el ImageNet LargeScale Visual Recognition Challenge (ILSVRC) en 2012 por una gran diferencia.

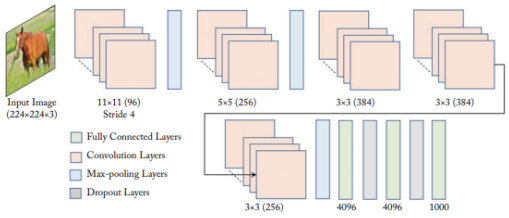

Figura 3.4: Arquitectura de Alexnet. Imagen sacada de [Khan et al., 2018]

3.7.3. VGG-Net

La VGG-Net, [Simonyan and Zisserman, 2014], en uno de los modelos de CNN más populares desde su introducción en 2014 aunque no fue el ganador del concurso ILSVRC'14. Los autores introdujeron un conjunto de configuraciones de la red, y las configuraciones D y E (llamadas también VGG-16 y VGG-19) son las más exitosas.

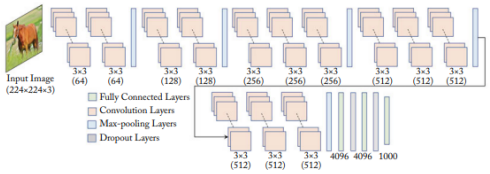

Figura 3.5: Arquitectura de VGG-Net. Imagen sacada de [Khan et al., 2018]

3.7.4. ResNet

La red residual, ResNet, [He et al., 2016] creada por Microsoft, ganó el ILSVRC 2015 con una gran diferencia en rendimiento. La característica más notable de este tipo de red, es la habilidad de saltarse conexiones lo que permite entrenar fácilmente arquitecturas de CNN muy profundas.

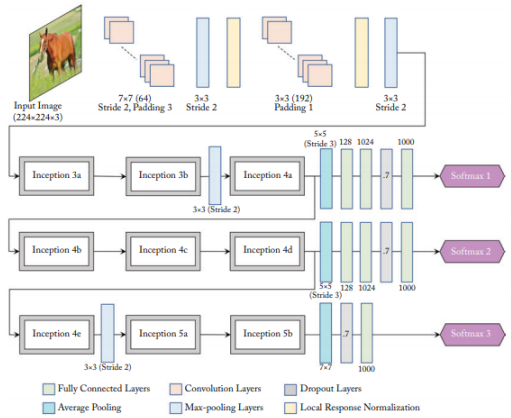

Figura 3.6: Arquitectura de ResNet. Imagen sacada de [Khan et al., 2018]

Capítulo 4

Entorno de trabajo

4.1. DeepFashion

DeepFashion, [Liu et al., 2016] es el conjunto de datos, o dataset en inglés, con el que hemos trabajado. Se trata de una base de datos de prendas de ropa a gran escala, con más de 800.000 imágenes. DeepFashion posee mucha información de las prendas, no solo la categoría en la que se encuentra, sino también atributos descriptivos, cuadros delimitadores y puntos de referencia de la ropa.

Figura 4.1: Ejemplo del conjunto de datos escogido. Imagen sacada de la página oficial de DeepFashion.

Este conjunto de datos dispone de datos para cuatro tipos de estudios. Desde predicción de categorías y atributos, detección de puntos de referencia, búsqueda de

prendas en tienda y búsqueda de prendas de consumidor en tienda. En este proyecto, nos hemos centrado en la parte de predicción de categorías y atributos, más concretamente en categorías.

Este subconjunto del conjunto de datos, poseía 289.222 imágenes, 50 categorías de prendas y 1000 atributos. Cada imagen además, tenía anotada el cuadro delimitador y el tipo de prenda. Más adelante explicaremos como se analizó el conjunto de datos para finalmente poder utilizarlo.

4.2. Python

Python es un lenguaje de programación de código abierto. Se trata de un lenguaje interpretado, de tipado dinámico y multiplataforma.

Actualmente es un lenguaje muy utilizado en el campo de análisis de datos gracias al uso de algunas de sus librerías, como son Pandas y Numpy. También es ampliamente utilizado en el mundo del Aprendizaje Profundo gracias a numerosos frameworks existentes.

4.2.1. Numpy

Numpy es el paquete de Python para computación científica. Es una librería que permite el manejo de arrays multidimensionales, y la realización de operaciones matemáticas a muy alta velocidad. Numpy está bajo la licensia BSD.

4.2.2. Pandas

Pandas es una librería de código abierto bajo la licencia BSD. Proporciona herramientas de análisis de datos y estructuras de sencilla utilización.

4.2.3. os

Esta librería permite una forma de utilizar las funcionalidades del sistema operativo.

4.2.4. Matplotlib

Matplotlib es una librería de Python que permite la creación de gráficos en 2D.

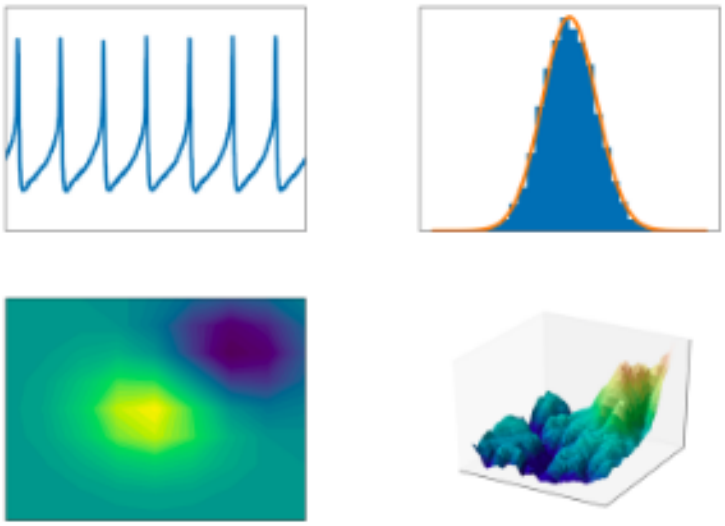

Figura 4.2: Ejemplo de gráficas con matplotlib. Imagen sacada de la página oficial de matplotlib.

4.2.5. urllib.request

Librería que proporciona herramientas para abrir URLs. En nuestro caso se ha utilizado esta librería para descargar la base de datos que utilizamos.

4.2.6. random

Módulo que implementa un generador de números aleatorios para varias distribuciones.

4.2.7. math

Módulo que permite el acceso a funciones matemáticas básicas especificadas por el estándar C. En nuestro caso simplemente la hemos utilizado para redondear

números hacia abajo.

4.3. OpenCV

OpenCV es una librería de visión por computador. Posee una licencia BSD. Su código está escrito en C++, y puede ser utlizado en varios lenguajes como C++, Java o Python. Además de funciones para el reconocimiento de objetos, posee funciones para la manipulación y procesamiento de imágenes. Debido a que nuestro proyecto ha sido desarrollado en Python, y a que esta librería es muy sencilla de utilizar, convierten a OpenCV en una herramienta excelente para este tipo de proyectos.

En nuestro proyecto esta librería será utilizada solamente para la manipulación de imágenes, no hará falta usar sus herramientas de detección.

4.4. Dlib

Dlib es otra librería multiplataforma escrita en C++. Posee diversas herramientas para muchas tareas diferentes, desde robótica hasta utilidades de testeo de software. En este caso ha sido utlizada para el reconocimiento de personas, ya que posee una herramienta que encuentra 68 puntos de referencia faciales, lo cual la convierte en una herramienta idónea para reconocimiento facial.

4.5. Keras

Keras es una API de alto nivel para el manejo de redes neuronales. Se puede ejecutar sobre CNTK, Tensorflow o Theano, las cuales son APIs de redes neuronales de más bajo nivel. Al ser una librería de alto nivel, permite un rápido prototipado de diferentes modelos. Una de sus grandes características, es la habilidad de poder ejecutarse directamente en la GPU, lo cual permite una ejecución substancialmente más veloz.

4.6. Weka

Weka es una colección de algoritmos de aprendizaje automático para tareas de minería de datos. Puede ser utilizada desde una interfaz gráfica, o bien directamente desde el código en Java. Weka posee tanto herramientas de visualización, preprocesamiento de datos, clasificación y regresión entre otras. Es un herramienta idónea para probar rápidamente varios algoritmos, para comprobar cual es el mejor para el problema en cuestión. Esta es la cualidad que hacen a esta herramienta tan atractiva, el rápido prototipado de los diferentes tipos de algoritmos.

4.7. Google Colaboratory

Google Colab es una plataforma en la nube que permite ejecutar código en Python de forma gratuita. La gran ventaja de Google Colab es que permite la utilización de una GPU Tesla K80 sin coste alguno. En este entorno es donde se ejecutará la mayoría de nuestro código, principalmente el entrenamiento de los modelos de redes neuronales. Esta plataforma permite tanto la ejecución de código en Python como de código de terminal como si se tratara de un entorno Linux. A su vez una de las ventajas de Google Colab es su sencilla integración con los documentos en Drive, ya que puede importar y exportar ficheros de forma rápida y con muy pocas líneas de código, mediante una API muy intuitiva.

Capítulo 5

Metodología

Tras haber aclarado el problema a tratar, explicado el estado del arte para este tipo de tareas, y hablado del entorno de trabajo utilizado, ahora explicaremos en detalle todo el proceso desarrollado para llevar a cabo este proyecto.

5.1. Pre-procesamiento de datos

En primer lugar, se realizó un pre-procesado de los datos, esto quiere decir, que analizamos los datos para ver cuales eran los que más nos interesaban hasta crear un conjunto de datos óptimo para nuestro problema en cuestión.

Como hemos dicho previamente, partimos de la base de datos DeepFashion, con 289.222 imágenes compuestas por 50 clases y 1000 atributos. Tras analizar esta base de datos llegamos a la conclusión de que no necesitaríamos estos 1000 atributos, ya que sería un problema básico de clasificación, por lo que la metodología a emplear sería entrenar un modelo con imágenes de esas categorías e intentar predecir un resultado correcto para cada imagen.

5.1.1. Categorías con más de 5000 imágenes

Lo primero que se hizo fue observar la cantidad de instancias que teníamos para cada categoría, para de esta forma poder hacernos una mejor idea de las dimensiones de nuestra base de datos. Tras crear un histograma para observar la distribución de cantidad de imágenes por categoría, se observó que era una distribución un tanto anormal, ya que habían clases para las que habían cerca de 70.000 imágenes y para otras un número muy cercano a 0. Para casos como el último nombrado,

lógicamente no tendría sentido entrenar un modelo, ya que directamente para esas clases no habría posibilidad alguna de entrenamiento.

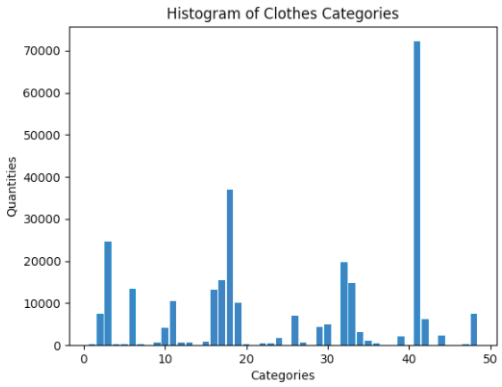

Figura 5.1: Histograma con la cantidad de imágenes por categoría.

Se permitió que permanecieran en el conjunto de datos, solo las categorías que tuvieran 5000 o más imágenes, ya que menos podría afectar al entrenamiento, al no ser suficientes como para inferir las características. Tras esto, pasamos de tener 50 categorías a tener 15.

5.1.2. Unir categorías Top y Tank

Después se analizaron las imágenes restantes, observando detenidamente una buena cantidad del muestreo total, para asegurarnos así de que las imágenes estaban correctamente clasificadas. se decidió por unir las categorías Tank y Top en una sola, que pasó a llamarse Top, debido a sus grandes similitudes, no era posible diferenciarlas. Tras esta unión el total de categorías se quedaba en 14.

Figura 5.2: Ejemplo de imagen de la categoría Tee.

Figura 5.3: Ejemplo de imagen de la categoría Tank.

5.1.3. Eliminar imágenes pequeñas

Con las imágenes restantes, se analizó cuales de ellas tenían un tamaño demasiado limitado, como para ser aceptables tras redimensionar. Finalmente, se eliminaron aquellas que tenían un ancho o alto inferior a 50 píxeles. Después de esto, la clase Leggings pasó a tener 4582 imágenes, el cual es un número inferior al umbral que se había fijado anteriormente de 5000 elementos, sin embargo se permitió mantener esta clase, ya que la insignificante diferencia con 5000 no supondría ningún dilema para este problema.

5.1.4. Balanceo del conjunto de datos

Finalmente, se decidió para entrenar, crear un subconjunto de imágenes en el que cada categoría tuviera 5000 elementos, para permitir así una distribución similar para cada categoría, de forma que ninguna categoría tuviera ventaja sobre otra. Esto fue así ya que el conjunto de datos originales tenía una distribución de imágenes muy dispar, ya que tenía 70000 para alguna categoría mientras que menos de 20 para otras.

5.1.5. Configuración del entrenamiento

A la hora de entrenar nuestro modelo, se hizo uso de una clase de Keras denominada ImageDataGenerator, de la cual se hablará más adelante, pero para cuyo uso se necesitaba una estructura de ficheros determinada. Esto quiere decir que era requisito que las imágenes estuvieran almacenadas en directorios que tuvieran como nombre la clase de estas imágenes. También requería que hubieran directorios padres de estos subdirectorios denominados train, para entrenamiento, valid, para validation y test, para testeo.

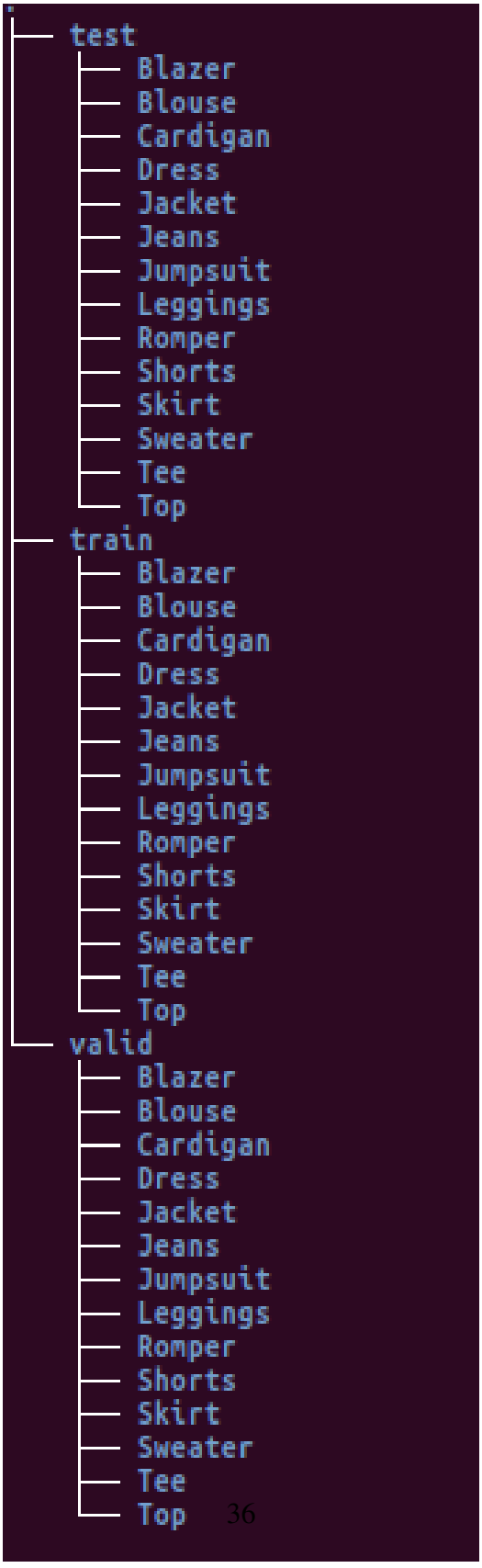

Figura 5.4: Estructura de ficheros para ImageDataGenerator

5.1.6. Anotación de contenedores

Llegando al final del tratamiento de los datos, se crearon directorios como los expuestos en el apartado anterior, en el que en lugar de tener imágenes, tenían ficheros que contenían las coordenadas del cuadro delimitador de cada imagen.

5.1.7. Ajustar dimensiones de las imágenes

Tomando como base lo realizado en el apartado anterior, se creó una estructura de directorios similar pero en lugar de los ficheros, tenían las imágenes solo con el cuadro delimitador, es decir, no todo la imagen sino solo lo que se necesitaba, y redimensionadas para que tuvieran el tamaño 150x200, mayor alto que ancho, ya que suele ser la distribución que suele tener la imagen de una persona en una foto.

Figura 5.5: Imagen original.

Figura 5.6: Imagen después de haber recortado el bounding box y de haberla redimensionado.

5.2. Familiarización con OpenCV y Dlib

En un primer momento, se iban a utilizar conjuntamente OpenCV y Dlib para hallar el cuadro delimitador de las imágenes a recortar, por lo que hubo que familiarizarse con varias funciones de estas librerías para estas tareas.

A pesar de que el conjunto de datos de DeepFashion ya posee un fichero con las coordenadas de los cuadros delimitadores de las imágenes, se pensó que podría no ser completamente certero y que este se podría mejorar utilizando estas librerías. Sin embargo, tras hacer una comparación entre ambas propuestas, se optó por utilizar las coordenadas proporcionadas por DeepFashion. Se pasó así a descartar la utilización de Dlib, que habría sido utilizada solamente para la tarea de reconocimiento facial, y se pasó a utilizar solo OpenCV para la tarea de manipulación de imágenes, las cuales eran recortar, redimensionar y guardar las imágenes.

5.3. Familiarizarse con Keras

Una vez se hubo realizado todo el pre-procesado de las imágenes, se estaba preparado para comenzar el diseño de la red neuronal. Para esta tarea, se optó por utilizar Keras, ya que se creó pensando en que se pudiera diseñar y probar modelos dinámicamente. Para esto, hubo que familiarizarse con este framework, el cual gracias a su sencillez, no hubo que extender su aprendizaje más de lo necesario.

En este ejemplo se ve como se implementan las librerías necesarias. La clase Sequential es una de las APIs de Keras que representa un conjunto de capas apiladas. La clase Dense corresponde con una capa de neuronas. En este caso, este red tiene una sola capa con cuatro neuronas, una entrada de un vector de 10 elementos, y se inicializan los pesos de forma aleatoria..

```
from keras.models import Sequential
from keras.layers import Dense

model = Sequential()
model.add(Dense(4, input_dim=10, kernel_initializer='random_uniform'))
```

Figura 5.7: Ejemplo sencillo de red neuronal en Keras.

5.4. Aprendizaje por Transferencia con VGG-16

Tras meditar las posibles aproximaciones hacia este problema, se optó por realizar aprendizaje por transferencia utilizando una red VGG-16, de la cual se habló en capítulos anteriores. Se aprovechó que Keras proporciona una librería con la cual podemos acceder a esta red ya entrenada con los pesos de Imagenet, el cual es un conjunto de datos de millones de imágenes en el que el objetivo es que clasifique correctamente 1000 categorías.

5.5. Implementación en Keras

```
from keras import applications
from keras.preprocessing import image
from keras.applications.vgg16 import preprocess_input
import numpy as np

img_rows, img_cols, img_channel = 224, 224, 3

base_model = applications.VGG16(weights='imagenet', include_top=True, input_shape=(img_rows, img_cols, img_channel))

img = image.load_img("cat2.jpeg", target_size=(224, 224))
x = image.img_to_array(img)
print(x.shape)
x = np.expand_dims(x, axis=0)
x = preprocess_input(x)
base_model.summary()
i = base_model.predict(x)
print(i.argmax())
```

Figura 5.8: Función que proporciona Keras para utilizar la VGG-16 preentrenada.

Para la implementación en Keras, se hizo uso de una clase denominada ImageDataGenerator. El conjunto de datos que finalmente se utilizó, era demasiado grande como para poder ser guardado en memoria, de forma, que lo que se hizo fue ir cogiendo muestras poco a poco hasta verlas todas. Esta es la tarea de ImageDataGenerator, va cogiendo imágenes en pequeños lotes de un tamaño definido con el parámetro ”batch_size”, hasta haber cogido todas las imágenes del conjunto.

Además de implementar VGG-16, se hicieron modificaciones en la red para mejorar su precisión. Se hicieron cambios tales como, modificar el algoritmo de optimización, el ratio de aprendizaje y el número de neuronas por capa entre otros. Más adelante en el apartado de Experimentos, se hablará detalladamente de las modificaciones con las que se obtuvieron mejores resultados.

5.6. Implementación en Google Colab

Google Colab es una herramienta existente en la nube de Google, que puede acceder a los archivos que una persona tenga en su cuenta de Google Drive mediante una API en Python. Una particularidad de Google Colab es que solo se puede utilizar una sesión continuada por un máximo de 12 horas, por lo que cada 12 horas hay que reiniciar el proceso.

```
!pip install -U -q PyDrive
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials
import os

# Authenticate and create the PyDrive client.
# This only needs to be done once per notebook.
auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)

def seeFiles():
  file_list = drive.ListFile({'q': "'root' in parents and trashed=false"}).GetList()
  for file1 in file_list:
    print('title: %s, id: %s' % (file1['title'], file1['id']))

def downloadFile(Id, name):
  file_id = Id
  downloaded = drive.CreateFile({'id': file_id})
  downloaded.GetContentFile(name)

def createFile(name):
  file5 = drive.CreateFile()
# Read file and set it as a content of this instance.
  file5.SetContentFile(name)
  file5.Upload() # Upload the file.
```

Figura 5.9: Ejemplo de las funciones necesarias para interactuar con los archivos de la cuenta de Google Drive de un usuario.

La función seeFiles mostraría todos los archivos que tiene el usuario en el directorio root. La función downloadFile descarga un archivo de Drive a la sesión actual de Google Colab. La función createFile, coge un archivo existente en la sesión de Google Colab y lo descarga a la cuenta de Google Drive.

Siempre que se iniciaba una sesión en Google Colab, se realizaba el mismo proceso, el cual consistía siempre en las siguientes tareas:

- Descargar los archivos de Drive a la sesión de Google Colab
- Descomprimir los archivos .zip
- Crear los lotes de entrenamiento
- Entrenar el modelo
- Guardar el resultado

Debido a que Google Colab solo nos permitía una ejecución máxima de 12 horas, y que debido a que la plataforma está aún en desarrollo y de vez en cuando surgían errores de conexión, se tuvieron que idear soluciones a estos problemas. Afortunadamente la librería de Keras tiene implementadas unas funciones de callbacks que

permitían ir guardando el modelo que mejor precisión tuviera en cada iteración del entrenamiento. Además, permitía la realización de una parada temprana, Early Stopping, en caso de que el porcentaje de acierto fuera disminuyendo, tratando así de evitar el problema de overfitting o sobreentrenamiento.

5.7. Planificación

Después de haber realizado un desarrollo minucioso del proyecto, se llevó a cabo el mismo tiempo para las tareas que el especificado en el TFT01.

Fases	***Duración Estimada*** *(horas)*	***Tareas*** *(nombre y descripción, obligatorio al menos una por fase)*
Estudio previo / Análisis	60	Tarea 1.1: Análisis y evaluación de las bases de datos de vestimenta de libre acceso disponibles.
		Tarea 1.2: Estudio y selección de las librerías de visión por computador.
		Tarea 1.3: Estudio de las redes neuronales convolucionales.
		Tarea 1.4: Familiarización con la librería Keras.
Diseño / Desarrollo / Implementación	100	Tarea 2.1: Creación de la base de datos de vestimenta a partir de las existentes.
		Tarea 2.2: Integración del detector de personas y obtención del área de interés.
		Tarea 2.3: Integración del detector de poses.
		Tarea 2.4: Implementación y entrenamiento de la red neuronal.
Evaluación / Validación / Prueba	100	Tarea 3.1: Evaluación de los resultados de la detección de personas.
		Tarea 3.2: Evaluación de diferentes topologías de la red neuronal implementada.
		Tarea 3.3: Evaluación de la inclusión de la información de pose en los resultados.
Documentación / Presentación	40	Tarea 4.1: Redacción de la memoria del TFG
		Tarea 4.2: Realización de la presentación
		Tarea 4.3: Ensayos presentación

Figura 5.10: Plan de trabajo especificado en el TFT01.

Capítulo 6

Experimentos

A continuación se hablará de como se llegó a tener una red con un 70 % de acierto. Para este dilema, como se dijo anteriormente, se probó a utilizar una red VGG-16, pero modificando la parte final. Se hicieron modificaciones de numerosos hiperparámetros, desde el ratio de aprendizaje, el número de neuronas por capa, etc. A continuación se hablarán de algunos de los cambios que se hicieron, y al final del capítulo se hablará de las estructuras que mejores resultados obtuvieron.

6.1. Número de neuronas

Originalmente se probó la red VGG-16 normal, que consistía en añadir a continuación de las capas convolucionales, dos capas con 4096 neuronas cada una y posteriormente una con el número de clases con una función de activación Softmax, sin embargo, esto no dió muy buenos resultados.

Seguidamente, se pasó a tener simplemente una capa de neuronas con 1024 neuronas conectada a una con el número de clases por neuronas. Debido al menor número de neuronas que había que entrenar, el entrenamiento era mucho más rápido.

Se probó también, a cambiar el número de neuronas de 1024 a 512, y a hacer combinaciones de varias de estas capas. Finalmente las estructuras que mejor funcionaban, fueron las que tenían una capa adicional además de la que contiene el número de clases, y que esta capa tuviera o bien 1024 o 512 neuronas.

```
model.add(Flatten(name='flatten'))
model.add(Dense(4096, activation='relu', name='fc1'))
model.add(Dense(4096, activation='relu', name='fc2'))
model.add(Dense(14, activation='softmax', name="output"))
```

Figura 6.1: Estructura original de la red.

6.2. Algoritmos de optimización

Una de las cosas que requirió de más experimentos, fue la utilización de los diferentes tipos de algoritmos de optimización, no solo por su variedad, sino porque a su vez cada uno poseía diferentes hiperparámetros que podían ser tuneados.

Se probaron gran cantidad de los algoritmos más utilizados para este tipo de problemas. Se probaron RMSprop [Hinton et al.], Adadelta [Zeiler, 2012], Adam [Kingma and Ba, 2014] Y Stochastic Gradient Descent [Bottou, 2010].

6.3. Regularización

Una de las herramientas más utilizados en el mundo del aprendizaje automático es la regularización. Esta herramienta consiste en evitar que el modelo acabe desarrollando un sobre-entrenamiento, esto consiste en que el modelo se acostumbre demasiado a las características del conjunto de datos de entrenamiento y no sea capaz de generalizar para datos que no haya visto previamente.

Esto es algo que ocurre con mucha frecuencia a la hora de desarrollar un algoritmo de aprendizaje automático, por ello es necesario ir llevando un seguimiento de las capacidades de efectividad de estos algoritmos a través de las diferentes iteraciones. Gracias a que observamos el porcentaje de precisión, se puede parar el entrenamiento justo cuando el algoritmo empiece a sobre-entrenarse. Sin embargo, esto no evita que el algoritmo llegue a sobre-entrenar.

Para evitar el sobre-entrenamiento, se aplicaron en algunos casos, capas de Dropout, el cual es un método que consiste en desactivar algunas neuronas en una iteración del entrenamiento, para que así el modelo no dependa excesivamente de alguna neurona en concreto.

Tras aplicar Dropout, se pudo ver un incremento significativo en el aprendizaje.

6.4. Normalización por lotes

La normalización por lotes [Ioffe and Szegedy, 2015], es una técnica que permite acelerar en gran medida el tiempo que necesita un clasificador para aprender. Normalmente, los datos de entrada al algoritmo se normalizan para tener valores entre 0 y 1. La idea detrás de la normalización por lotes es aprovechar esta ventaja que tienen los datos de entrada, y añadírsela a las capas ocultas de la red neuronal.

6.5. Tamaño de entrada de datos

Por defecto, la red VGG-16 tiene una entrada de datos de tamaño 224x224x3, sin embargo, las imágenes después del preprocesado realizado eran de 150x200x3. En un principio se entrenó la red cambiando el tamaño de entrada por el de las imáges preprocesadas y se comprobaron los resultados, sin embargo tras varias pruebas, se optó por poner el tamaño de entrada que viene por defecto, por lo que hubo que redimensionar de nuevo las imágenes. Curiosamente, al poner el tamaño de datos original, a pesar de que es superior al de las preprocesadas y por tanto tendría que buscar mas características, la red ofrecía mejores resultados que anteriormente.

6.6. Aumento de datos

Otra de las cosas que se probó para mejorar el desempeño de la red fue usar aumento de datos, o data augmentation. Esto consiste en aumentar la cantidad de imágenes del conjunto de datos, esto se hace modificando las imágenes originales, cambiando su tamaño, poniéndolas en ángulos diferentes, o invirtiéndolas entre otras opciones. Esta técnica suele utilizarse cuando el conjunto de datos es bastante pequeño, como es este caso.

Esta técnica también es muy útil cuando las imágenes son tomadas en unas condiciones limitadas, ya que el aumento de datos permite obtener imágenes con una gran variedad de condiciones, desde iluminación, al ángulo desde que está tomada la foto. El aumento de datos también es muy útil en casos en los que ya partimos con un gran conjunto de imágenes, debido a lo que se ha dicho previamente, puede que haya una gran cantidad de imágenes pero que estas sean tomadas con condiciones limitadas, y la técnica de aumento de datos permite subsanar en gran medida este inconveniente.

Esto, lógicamente, ayudaba a mejorar la red, ya que permitía que la red discerniera de mejor forma las características esenciales de cada tipo de prenda.

6.7. Fine-tuning

Una de las aproximaciones que se tomaron para mejorar el resultado de la red, consistió en retocar algunas de las capas convolucionales de la VGG-16, que en un principio estaban congeladas y solo servían para coger características. Se cogió solo la capa convolucional anterior a nuestra red neuronal y se entrenó tanto esta capa como nuestra red neuronal. Solo se entrenó esta capa y no todo el conjunto debido a que se poseía una cantidad de datos muy limitada. La opción de entrenar todo la red incluidas todas las capas convolucionales habría sido óptima si se hubiera tenido un gran volumen de imágenes, sin embargo, como se dijo anteriormente solo se tenían un conjunto de 5000 imágenes por categoría.

6.8. Mejores resultados

Ahora se hablarán de las estructuras y aproximaciones que dieron lugar a los mejores resultados.

```
def create_model():
  vgg16 = keras.applications.vgg16.VGG16(weights='imagenet', input_shape=(224, 224, 3), include_top=False)

  model = Sequential()

  for layer in vgg16.layers:
      model.add(layer)


  for layer in model.layers[:-4]:
    layer.trainable = False

  model.add(Flatten(name='flatten'))
  model.add(Dense(512, activation='relu', name='fc1'))
  model.add(BatchNormalization())
  model.add(Dropout(0.8))
  model.add(Dense(14, activation='softmax', name="output"))
  model.compile(keras.optimizers.Adam(lr=0.0001), loss='categorical_crossentropy', metrics=['accuracy'])
  return model
```

Figura 6.2: Estructura de la red que mejores resultados dio.

Como se puede apreciar, esta red usa tanto Dropout, como BatchNormalization. También utiliza el algoritmo de optimización Adam con un ratio de aprendizaje de 0.0001. Esta red ofreció una precisión del 70 %. Usaba todo lo mencionado anteriormente, además de aumento de datos y un fine-tuning de algunas capas convolucionales.

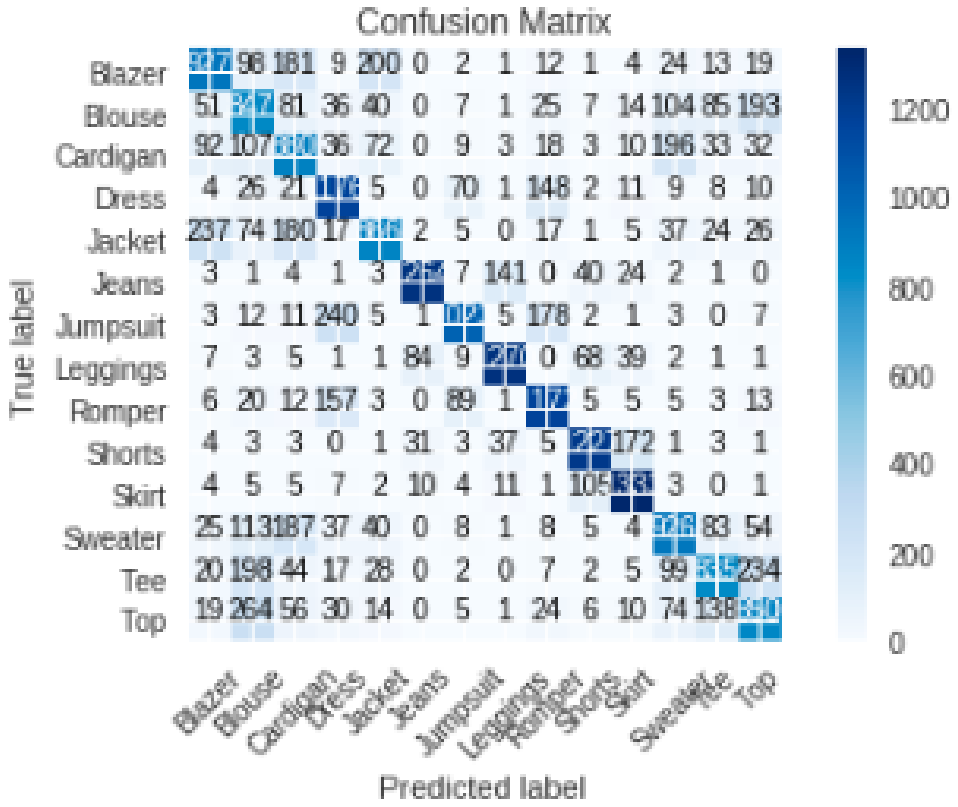

Figura 6.3: Matriz de confusión generada tras probar el conjunto de testeo.

6.9. Análisis de resultados

Tras los numerosos experimentos, es decir, las modificaciones de la arquitectura de la red, se llegó a un 70 % de precisión máxima para nuestro conjunto de testeo. A pesar de que es un buen resultado, se buscaba el mayor porcentaje de acierto posible por lo que se optó por tomar diferentes aproximaciones a partir de estos resultados.

Figura 6.4: Ejemplo de predicción fallida en la que el algoritmo dice que la prenda es pertenece a Cardigan, pero en realidad pertenece a Blouse.

6.10. Aumentando resultados en Weka

Una de las opciones tomadas, fue quitar la parte Fully Connected a excepción de la capa conectada inmediatamente a la última capa convolucional. De esta capa con 512 neuronas, se tomaba un vector de características de las mismas dimensiones. Esto se hizo así, ya que las capas convolucionales y los pesos de esta última capa, deberían estar entrenadas como para al menos inferir correctamente las características de las prendas, a pesar de que luego no fuera capaz de clasificarlas correctamente. El objetivo era entrenar otros algoritmos de aprendizaje automático, en lugar de con redes neuronales para ver si obteníamos mejores resultados
.
Se utilizó Weka, para probar estos otros algoritmos, ya que como se mencionó anteriormente, se trata de una herramienta con una interfaz gráfica de muy fácil utilización que permite un prototipado muy veloz. La entrada de datos preferente de Weka es un archivo de tipo ARFF, el cual es un archivo CSV con algunas peculiaridades. Los valores de los atributos se escriben igual que en formato CSV, pero para los nombres de las columnas, se tiene que escribir "@attribute", seguido del nombre del atributo y su tipo de valor. Para generar este archivo, se partió del conjunto de datos que se tenía para entrenar. Se ejecutaron predicciones por cada imagen y se obtuvo el valor de cada resultado, de dimensión 512, en una matriz. Posteriormente, se pasó a depositar el contenido de esta matriz en el fichero ARFF.

```
@relation weather

@attribute outlook {sunny, overcast, rainy}
@attribute temperature numeric
@attribute humidity numeric
@attribute windy {TRUE, FALSE}
@attribute play {yes, no}

@data
sunny,85,85,FALSE,no
sunny,80,90,TRUE,no
overcast,83,86,FALSE,yes
rainy,70,96,FALSE,yes
rainy,68,80,FALSE,yes
rainy,65,70,TRUE,no
overcast,64,65,TRUE,yes
sunny,72,95,FALSE,no
sunny,69,70,FALSE,yes
rainy,75,80,FALSE,yes
sunny,75,70,TRUE,yes
overcast,72,90,TRUE,yes
overcast,81,75,FALSE,yes
rainy,71,91,TRUE,no
```

Figura 6.5: Ejemplo de la estructura de un archivo ARFF.

Una vez se obtenía el archivo ARFF, se abría en Weka, y se pasaba a probar los diversos algoritmos. Como se dijo anteriormente, los algoritmos tradicionales más utilizados para esta tarea eran las SVM y los Bosques Aleatorios, por lo que fueron los algoritmos escogidos. Debido a que el Bosque Aleatorio es el algoritmo más sencillo, y por tanto más veloz a la hora de ejecutarse, se probó este primero. Seguidamente, se probó el SVM con kernel Lineal y kernel con función de base radial (RBF).

- Random Forests : 76.55 %
- SVM con kernel Lineal : 76.71 %
- SVM con kernel RBF : 76.52 %

Como se puede apreciar en los resultados anteriores, este procedimiento proporcionó mejores resultados que solo una red neuronal.

6.11. Entrenando una red ensamblada

Otra de las aproximaciones que se tomó, consistía en utilizar una red ensamblada. Esto quiere decir, que en lugar de utilizar una sola red neuronal, se entrenaron diferentes redes y se unieron para así mejorar su rendimiento. En este caso, se entrenó una red que diferenciaba entre parte de arriba, parte de abajo y cuerpo completo.

También se entrenaron tres redes diferentes, una para cada parte del cuerpo como se acaba de decir. Esto se entenderá mejor de la siguiente manera. Imaginemos que la prenda a clasificar es un vaquero, en este caso, la primera red dirá que se trata de una parte de abajo, por lo que le pasará la imagen a la red que clasifica solo prendas inferiores, y esta a su vez dirá que es un vaquero. Esta aproximación tuvo un éxito del 71 %.

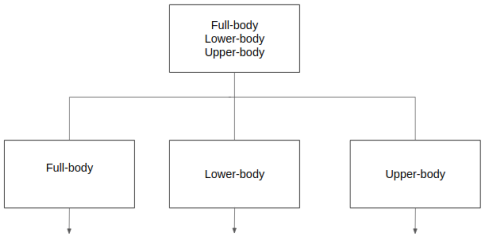

Figura 6.6: Estructura de la red ensamblada.

6.12. Probando clasificación top-k

La última aproximación tomada, consistió en utilizar clasificación top-k [Gong et al., 2013]. La ventaja de este tipo de clasificación radica en tomar los k mejores resultados dados por la predicción. Por ejemplo si tomáramos k = 2, nos daría las dos categorías con mayor probabilidad de acierto. Esto se puede conseguir debido a que la capa exterior de la red tiene una función de activación Softmax, la cual de divide las probabilidades de las categorías más probables, siendo la categoría más viable la más cercana a 1.

En nuestro caso se probó la clasificación top-k solamente para la red entrenada para todas las categorías completas, es decir, aquella que en un principio nos daba un 69 % de acierto. Después de probar con k = 2, el algoritmo daba un porcentaje de acierto del 85.17 %, también se probó para k = 3, dando un 91.36 %. No tendría sentido hacerlo con un valor mayor para k debido a que solo tenemos un total de 14 categorías.

Capítulo 7

Conclusiones

En este trabajo se han comentado los métodos de visión por computador utilizados previamente a la resurgencia de las redes neuronales. Se han implementado varias arquitecturas de CNNs, y se han utilizado CNNs no solo como clasificadores sino también como descriptores de características para permitir que otros algoritmos realizaran la clasificación.

Se ha seguido todo el proceso, desde la elección de la base de datos y el preprocesado de ésta, hasta finalmente la generación y utilización del clasificador. Comprobando así de primera mano, las diferentes tareas que serían necesarias, para el desarrollo de un producto como este.

Se han tratado de implementar las mejores técnicas existentes para nuestro problema, aunque nuestros resultados no son excelentes, pero sí están bastante bien para una primera aproximación.

7.1. Aspectos a Mejorar

Tras haber realizado todo el proceso de entrenamiento y testeo, y ver que nuestro clasificador fallaba, a la hora de distinguir, entre las categorías que pertenecían a prendas de la parte de arriba del cuerpo, se analizaron las imágenes del conjunto de datos, y se llegó a la conclusión de que habían imágenes de categorías mezcladas. No eran muchas, pero sí parece que fueron suficientes, como para impedir que nuestro algoritmo, tuviera una certeza absoluta a la hora de predecir.

Creo que como usuario de la base de datos, habría sido inconsistente, analizar todas las imágenes para asegurarnos de que estuvieran bien clasificadas, por los creado-

res de la base de datos. Así que se podría probar a trabajar con una base de diferente.

7.2. Trabajo Futuro

Una aplicación de nuestro clasificador, sería la posibilidad de utilizarlo como medio para encontrar prendas en páginas de tiendas de ropa. Sería algo así como implementar un Shazam para ropa.

A nivel extendido, se podría crear un recomendador de ropa que vestir, que examine la ropa disponible y le diga al usuario qué vestir. Muy útil cuando el usuario no sabe qué ponerse.

Otra aplicación que podría realizarse, sería la de diseñar una arquitectura que pudiera describir correctamente todas las prendas que lleva una persona, en lugar de simplemente clasificar.

Se podría tomar este trabajo de fin de grado, como punto de partida para entrenar un clasificador para cualquier tipo de datos, es decir, en lugar de entrenar un clasificador prendas, se podría entrenar para clasificar diferentes tipos de objetos.

7.3. Valoración Personal

Nunca antes había realizado un proyecto de Inteligencia Artificial, pero gracias a este, me he hecho una idea de como sería todo el proceso, para llevar a cabo uno en el mundo real. Actualmente el campo del Aprendizaje Automático es muy atractivo laboralmente hablando, ya que hay gran demanda hoy en día. Gracias a la realización de este proyecto y a todo lo que he tenido que aprender para realizarlo, me veo más capacitado para poder realizar cualquier tipo de trabajo, relacionado con Aprendizaje Profundo, ya que aunque haya algún tema que no domine, siempre puedo aprender sobre el mismo.

Bibliografía

L. Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

L. Breiman. Random forests. *UC Berkeley TR567*, 1999.

L. Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893. IEEE, 2005.

L. Deng, D. Yu, et al. Deep learning: methods and applications. *Foundations and Trends® in Signal Processing*, 7(3–4):197–387, 2014.

Y. Gong, Y. Jia, T. Leung, A. Toshev, and S. Ioffe. Deep convolutional ranking for multilabel image annotation. *arXiv preprint arXiv:1312.4894*, 2013.

I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

G. Hinton, N. Srivastava, and K. Swersky. Lecture 6a overview of mini-batch gradient descent (2012). *Coursera Lecture slides https://class. coursera. org/neuralnets-2012-001/lecture*.

S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

S. Khan, H. Rahmani, S. A. A. Shah, and M. Bennamoun. A guide to convolutional neural networks for computer vision. *Synthesis Lectures on Computer Vision*, 8 (1):1–207, 2018.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

Y. LeCun, Y. Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *nature*, 521(7553):436, 2015.

Z. Liu, P. Luo, S. Qiu, X. Wang, and X. Tang. Deepfashion: Powering robust clothes recognition and retrieval with rich annotations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1096–1104, 2016.

D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

D. Meyer and F. T. Wien. Support vector machines. *R News*, 1(3):23–26, 2001.

J. R. Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

H. Ramchoun, M. Amine, J. Idrissi, Y. Ghanou, and M. Ettaouil. Multilayer perceptron: Architecture optimization and training. *IJIMAI*, 4(1):26–30, 2016.

F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

D. Svozil, V. Kvasnicka, and J. Pospichal. Introduction to multi-layer feed-forward neural networks. *Chemometrics and intelligent laboratory systems*, 39(1):43–62, 1997.

V. Vapnik. *The nature of statistical learning theory*. Springer science & business media, 2013.

M. D. Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.