Desarrollo de una red neuronal profunda para la clasificación y caracterización de prendas de vestir

Titulación: Grado en Ingeniería Informática

Autor: Víctor Herrera Delgado

Supervisado por el tutor Cayetano Guerra Artal

Noviembre 2020

Índice

Tabla de ilustraciones

0.1 Agradecimientos

Antes del comienzo de este documento, quería dar las gracias …

- A mis padres, por siempre apoyarme y darme fuerzas para salir adelante.
- A mi tía y a mi amiga especial A.M.R.M que actuaron como refuerzo de mis padres.
- A D.D.P. y al resto de la "élite informática" por hacerme ver lo que valgo.
- Al resto de mis amigos que me siguieron hablando a pesar de no tener tiempo para ellos.
- Y a mis profesores (y mi tutor, claro está) que han hecho que pudiera llegar aquí.

Capítulo 1

Introducción

1.1 Descripción / Contexto.

Dado la importancia que genera la inteligencia artificial en la era digital en la que vivimos, saber manejarse con ella es algo prácticamente fundamental si se quiere destacar en el mercado. La idea de que un programa te permita clasificar elementos a partir de ejemplos de los que ya uno dispone, facilitando un proceso que podría tardar años y un gran coste si se dejara puramente en manos humanas, o hacer lo mismo con el objetivo de obtener análisis de un posible cliente, al cual queremos ofrecerle un producto concreto en base a un estudio de sus preferencias, es algo bastante atractivo para cualquier empresa grande y no tan grande. No en vano se va a crear un nuevo grado para tratar «data science», uno de los elementos con más importancia en la actualidad en este campo.

Esto es algo que las empresas claramente saben, y no son pocas las que buscan de una u otra manera incorporar estas tecnologías en auge en su forma de trabajar. Actualmente pueden verse distintos tipos de aplicaciones ya instaladas. Si nos vamos a las páginas webs bajo nombres de empresas importantes vemos ejemplos, como chats para consultar problemas que usan chatbots antes de preguntar a alguien de atención al cliente, o incluso sin atención humana, webs de compras online que almacenan gran cantidad de información de compras de otros usuarios, para intentar predecir que podrías querer comprar, antes de si quiera saber de la existencia de un producto, entre muchos otros tipos de análisis que nos hacen día a día para un objetivo u otro.

En general este mundo genera un gran interés a alguien que empieza en la informática y que quiere llegar a hacer grandes cosas. Por ello, como un acercamiento a estos grandes objetivos proponemos un proyecto con el cual adentrarse en la inteligencia artificial plenamente.

Por ello es que, como inicio de una posible aplicación mayor dedicada a ser usada por empresas en el ámbito de la industria de la moda, buscamos la creación de un modelo de red neuronal que permita una clasificación precisa de ropa y otros elementos presentes en tiendas de moda de forma que pudiera ser usado en el futuro por empresas para la búsqueda de prendas de ropas. Un ejemplo de esto sería ver una prenda en un escaparate y a partir de una foto, lograr encontrar en los datos web de la tienda la prenda exacta.

Buscaremos usando tecnologías de inteligencia artificial (concretamente el uso de redes neuronales convolucionales) crear una aplicación que clasifique de forma precisa una imagen de una prenda de vestir en las distintas categorías posibles: tipo de artículo, subcategoría, género, temporada …

1.2 Motivación / Objetivos.

El objetivo principal de este proyecto es contribuir a la inteligencia artificial con un nuevo estudio que usaría redes neuronales convolucionales de forma que el estudiante pudiera profundizar aún más en la rama de la inteligencia artificial, así como en el mundo de la investigación.

El proyecto se dividiría en un grupo de fases a realizar para la realización completa del proyecto.

Una primera fase de investigación para aprender con ayuda del tutor cómo realizar el proyecto de investigación y las distintas herramientas que se podrían utilizar durante su desarrollo. Esto se conseguiría mediante la continua exploración de artículos relacionados con los distintos aspectos que componen el tema tratado en el proyecto y que constarán en la bibliografía. Si bien se trata como una primera fase, se tratará varias veces a lo largo del proyecto a medida que vayan siendo necesarias alternativas para proseguir.

En segundo lugar, se procedería a la búsqueda de un conjunto de datos (dataset) con los suficientes datos incorporados como para que la investigación tenga sentido. En nuestro caso se tratará de un conjunto formado por imágenes y que para cada una de ellas disponga de suficiente información o categorías como para tener la posibilidad de crear un modelo de red neuronal que pueda clasificarlos y que por lo tanto pueda extraer información de ellos.

En tercer lugar, tocará la generación de la red o redes neuronales con las que poder sacar características de las prendas en nuestro dataset. Nos basaremos en las indicaciones del tutor y lo investigado en la primera fase para buscar estructuras que sirvan para el cometido de clasificación de imágenes, generando un nuevo modelo para cada una de las posibles categorías en las que se clasifiquen. Esto incluiría la lectura de los datos y su preparación para ello, la creación de los modelos, el uso de técnicas que nos permitan mejorar los resultados y la evaluación de sus resultados.

Todo el desarrollo del proyecto se realizará usando Python como lenguaje en combinación con el framework Pytorch exclusivamente, ambos de gran uso en el mundo de la inteligencia artificial y en los que se profundizará más adelante. A estos se aunarán otras librerías que se vieron necesarias para las distintas tareas que implica el proyecto, como lectura y análisis del dataset, optimización, etc. Las ejecuciones se realizarán con un servicio en la nube que permita su ejecución con mínimas interrupciones y una GPU adecuada con el trabajo.

Cada uno de los pasos se seguirán con el objetivo de lograr un grupo de modelos aceptables en la clasificación de elementos del mundo de la moda en cada una de sus distintas categorías posibles en el tiempo designado para el proyecto.

1.3 Metodología / Planificación.

La planificación del proyecto intentó seguir lo propuesto en el TFT01 para los tiempos dedicados a cada parte, a pesar de que al final cambiaran ligeramente en su distribución horaria, se mantuvieron vigentes a lo largo del desarrollo.

La dedicación fue de aproximadamente 18 horas semanales, sin establecer días de trabajo concreto ante la adicional dedicación a asignaturas de la carrera y a prácticas de empresa remuneradas. Se hizo un seguimiento semanal de reuniones grupales entre el tutor y el resto de los alumnos bajo su tutela en las cuales cada alumno exponía sus avances durante la semana, ya fuera de código realizado o de resultados obtenidos en ejecuciones. La exposición grupal serviría como una forma para compartir ideas que si bien para un proyecto podían no haber funcionado si podían suponer un avance en uno de los otros. Así mismo, el tutor aportaría feedback de lo realizado y en función de esto se decidiría el objetivo a lograr para la siguiente semana. Las reuniones pasarían a ser por subgrupos de forma telemática luego con la llegada de la pandemia, pero eso no supondría cambio alguno en la dinámica de trabajo.

El desarrollo contó con una gran cantidad de ejecuciones de entrenamiento, ya fuera probando a entrenar con distintos hiperparámetros, buscando distintas formas de configurar el entrenamiento, cambiando la entrada de datos, probando aspectos del código en general (fallando varias veces que acabaron siendo tiempo perdido), buscando la optimización de los hiperparámetros, ejecutándolo sobre las distintas redes, etc.

Los resultados obtenidos se guardaron temporalmente para compararlos con el resto, ya que sería durante el último tercio del tiempo del proyecto cuando realmente se empezaría a escribir la memoria y cuando se harían las ejecuciones que más importancia tendrían para sacar las conclusiones.

Los tiempos de ejecución eran aprovechados para buscar hacer mejores versiones del código existente, expandir el conocimiento con más documentación o simplemente escribir los documentos que servirían de memoria y anexos.

Se dedicó también tiempo del desarrollo para la creación de imágenes propias que se usaron principalmente en la sección de bases teóricas y estado del arte.

1.4 Competencias y aportaciones.

En relación con las competencias especificadas en el documento TFT01 previamente entregado, se han cumplido con las siguientes:

T3. *Capacidad para diseñar, desarrollar, evaluar y asegurar la accesibilidad, ergonomía, usabilidad y seguridad de los sistemas, servicios y aplicaciones informáticas, así como de la información que gestionan.* (G1, G2).

T5*. Capacidad para concebir, desarrollar y mantener sistemas, servicios y aplicaciones informáticas empleando los métodos de la ingeniería del software como instrumento para el aseguramiento de su calidad, de acuerdo con los conocimientos adquiridos según lo establecido en apartado 5 de la resolución indicada*. **(G1, G2).**

Durante este proyecto se ha ido desarrollando poco a poco el código para la creación de los distintos modelos de redes neuronales y para las operaciones relacionadas con estos, asegurando su correcto funcionamiento. Se ha evaluado múltiples veces para su optimización y conseguir así un área de trabajo más sofisticada, permitiendo un uso más eficaz de las herramientas creadas para disminuir el tiempo necesario para la modificación y ejecución de los parámetros que tienen lugar.

T8. *Conocimiento de las materias básicas y tecnologías, que capaciten para el aprendizaje y desarrollo de nuevos métodos y tecnologías, así como las que les doten de una gran versatilidad para adaptarse a nuevas situaciones.* **(G3, N3).**

Además de profundizar una vez más en los conocimientos de programación e inteligencia artificial con los cuales se ha trabajado a lo largo de la carrera, se ha hecho uso de nuevas tecnologías no vistas hasta el momento, como el framework usado y el servicio en la nube entre otros, según fuera necesario. En cada una de las nuevas herramientas y técnicas se hizo uso de las bases ya establecidas durante el estudio académico

CP03. *Capacidad para evaluar la complejidad computacional de un problema, conocer estrategias algorítmicas que puedan conducir a su resolución y recomendar, desarrollar e implementar aquella que garantice el mejor rendimiento de acuerdo con los requisitos establecidos.*

Durante el trabajo se ha investigado la obtención de unos buenos resultados de clasificación de las redes neuronales, incluyendo en su entrenamiento distintas técnicas que mejoren el proceso de entrenamiento y que nos lleven a resultados aceptables e incluso mejores que los obtenidos anteriormente.

CP07. *Capacidad para conocer y desarrollar técnicas de aprendizaje computacional y diseñar e implementar aplicaciones y sistemas que las utilicen, incluyendo las dedicadas a extracción automática de información y conocimiento a partir de grandes volúmenes de datos.*

Este proyecto realiza el entrenamiento de redes neuronales convolucionales para la clasificación, y, por lo tanto, obtención de características a partir de un conjunto extenso de imágenes de elementos del mundo de la moda.

Como aportación que supone este proyecto, si bien existen proyectos que ya se han dedicado a la clasificación de prendas de vestir basados en conjuntos como fashion mnist y Deep fashion, no se han encontrado proyectos que indaguen más allá de la clasificación de prendas de vestir por su tipo. En este proyecto además de usar un conjunto distinto a los ya nombrados, y además de clasificar también las prendas de

vestir de éste (aun conteniendo clases distintas), trata de investigar adicionalmente otro tipo de características que no se han visto en otros trabajos que se tratarán más adelante.

Capítulo 2

Bases teóricas y estado actual.

2.1 Machine learning.

Empecemos con un concepto general, ¿qué es machine learning? Se trata de un subconcepto en la inteligencia artificial y a su vez subcampo de las ciencias de la computación que tiene como objetivo el desarrollo de tecnologías que permitan el aprendizaje por parte de los computadores. En general se trata de la creación de programas y algoritmos que permitan a un ordenador generalizar un comportamiento a partir de una información de entrada. Véase, es una forma de que los ordenadores sean capaces de inducir conocimiento a partir de ejemplos dados.

Dependiendo del tipo de técnica que use la computadora para «aprender», se puede clasificar en los siguientes tipos:

- **Aprendizaje supervisado**: Es cuando la técnica se basa primordialmente en los datos de entrada y salida. Consiste en tener un conjunto de datos con unas características y que se encuentren clasificados en un número de categorías. El sistema aprenderá a clasificar los distintos ejemplos en las categorías posibles usando los datos del conjunto mencionado y comparando los resultados del proceso realizado por el computador con los originales de los datos, corrigiendo su comportamiento en base a esto. De esta forma la máquina aprenderá a sacar unos resultados a partir de entrenar con multitud de ejemplos. Las redes neuronales en las que se basa este proyecto son un ejemplo de este tipo de técnica.
- **Aprendizaje no supervisado**: La técnica de aprendizaje no supervisado, al contrario del supervisado, solo ajusta su comportamiento en función de los datos de entrada no teniendo en cuenta la salida. Los ejemplos de entrada usados no están etiquetados o clasificados ya que no es necesario para entrenarlo. Un ejemplo de este tipo de aprendizaje son los mapas autoorganizados, que mapea los datos de entrada de forma que aporten nueva información en su representación final.
- **Aprendizaje por refuerzo (o semi-supervisado)**: Este tipo se basa en la definición de unos modelos que se dediquen a maximizar un valor, teniendo como base el ambiente en el que trabaja el sistema y las «acciones» que este haga. Son las técnicas más parecidas al comportamiento humano habitual, al buscar con sus «acciones» la maximización de una «recompensa». Un ejemplo fácil de entender sería un videojuego cuyo control tenga la máquina a entrenar y que basándose en la pantalla de juego deba aprender que acciones tomar para mejorar su puntuación. La técnica Q-learning es un ejemplo conocido de este aprendizaje.

Cabe mencionar que ningún tipo es mejor que otro, sino que cada uno tiene ámbitos en los cuales destacan y para los cuales por lo tanto es necesario desarrollar una técnica específica para resolver los problemas que se planteen.

2.2 Redes neuronales.

Concepto general.

Las redes neuronales son actualmente uno de los conceptos más influyentes que existe en el mundo de la inteligencia artificial. Si bien el concepto de las redes neuronales se había planteado con anterioridad, no ha sido hasta estos últimos años (especialmente sobre la última década) cuando realmente ha empezado a instalarse en varios aspectos de nuestro día a día, no solo debido a su excelente rendimiento, sino también en gran parte gracias a la continua mejora del hardware, en especial respecto a las tarjetas gráficas, y a la creciente implementación de nuevas tecnologías en la sociedad, que permiten que los elevados costes de cómputo de estas sean más accesibles.

Los modelos que caen bajo el nombre de redes neuronales toman inspiración del cuerpo humano, concretamente de las neuronas en el cerebro y en cómo trabajan estas para extraer características de la información. Las redes neuronales se basan en capas que se conectan entre ellas, estando cada una de las capas formadas por un número de unidades de procesamiento básicas que imitan a las neuronas. Una neurona natural recibe impulsos, y dependiendo de los impulsos que reciba, la neurona podría activarse o no generando otro impulso eléctrico. Por el otro lado, las unidades básicas de procesamiento de las redes reciben un conjunto de señales con unos valores y en función de sus valores y unos valores internos de la unidad, decidirá si producir un tipo de impulso u otro. Estas señales de entrada se suman de forma ponderada mediante sus correspondientes pesos, de forma que, si tal suma llega a superar un cierto umbral, generará un valor concreto de señal de salida (1 por ejemplo), dejando salir otro valor si no se llega a superar (0 por ejemplo). Las redes neuronales se forman de conjuntos de estas unidades cuya entrada y/o salida están conectadas a otro conjunto de unidades que procesa las nuevas entradas y producen resultados distintos, inspirándose en el funcionamiento de las neuronas en el cerebro.

Diferentes variaciones de estos conjuntos de neuronas, así como distintas organizaciones de sus elementos, permiten la implicación de las redes en gran cantidad de tareas como pueden ser el reconocimiento facial, predicciones de sucesos basados en datos ya existentes, clasificación de objetos, etc.

A continuación, profundizaremos más en su funcionamiento.

El perceptrón

Llamamos **perceptrón** *[14]* a la unidad básica de cómputo que usan las redes neuronales. Es la representación de una neurona real en la informática. Como mencionamos anteriormente, los perceptrones como unidad básica de una red neuronal, son elementos que reciben un conjunto de señales y en función de estos presentan una salida, pero ¿cómo pueden estos aprender o siquiera clasificar algo según las entradas? Vayamos por partes. Para comprender su funcionamiento mejor pensemos en una red para una clasificación simple, que estuviera formada solo por uno de estos perceptrones.

Volvamos a centrarnos en el perceptrón para mostrar un ejemplo de clasificación lineal.

Un ejemplo de problema de clasificación para una red puede ser que teniendo como referencia las características de un portátil (una o más), clasificar si se trata de un portátil de alta gama o no. Pongamos que dependiendo de las pulgadas de la pantalla y de la memoria interna de la que dispone, el elemento entre en el grupo de gama alta o no. Estas y demás características las representaremos como un vector de características en el que cada una de sus posiciones corresponde a un valor determinado referente a una de las características:

$$\vec{e} = (e_1, e_2, e_3, \dots, e_n)$$

Pongamos que tenemos 5 ejemplos, cada uno representado con un vector de características cuya primera posición representa las pulgadas de la pantalla y la segunda posición el tamaño de la memoria interna en gigabytes, de manera que podemos representar los puntos en un espacio 2D.

Como queremos clasificar, deberemos tener mínimo dos clases entre las que diferenciarlos: gama alta y gama baja. El conjunto de muestras se podrá considerar como separable linealmente si representando los puntos en un plano 2D según sus dos entradas (haciendo de x e y) se puede trazar una recta que sea capaz de separar un grupo de otro (OJO: esto se cumple considerando el vector de características de este ejemplo. Para más características estaríamos hablando de pasar a las 3 dimensiones o más, separar los puntos usando planos o hiperplanos respectivamente).

Si el conjunto es linealmente separable, un único perceptrón debería poder clasificarla sin problemas. Ahora, ¿Cómo actuaría el perceptrón como un clasificador lineal para este caso?

El perceptrón, visualmente representado en la ilustración 1, recibe las señales de entrada (e_n). En el caso de nuestro ejemplo recibe el vector de 2 elementos, siendo cada uno de estos elementos el valor de una de las entradas (e_1 y e_2). A su vez, el perceptrón que sabe el número de entradas que le llegan, tiene designado para cada una de ellas un valor (w_i) por el cual se va a multiplicar la entrada. A estos valores los llamaremos de ahora en adelante pesos. El perceptrón realiza un sumatorio de la multiplicación de las entradas por sus correspondientes pesos. Además, cuenta con un valor llamado bias (θ) el cual se le resta al resultado de la operación anterior y con lo que forma la salida del perceptrón. En resumen, realiza la siguiente función:

$$res = (\sum_{j=1}^{n} w_j * e_j) - \theta$$

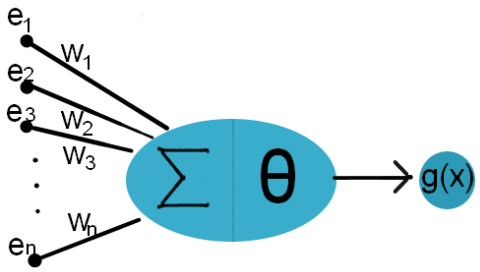


Ilustración 1 Perceptrón básico

Sin embargo, la salida del perceptrón como tal no nos vale para considerarlo una neurona, para ello es necesario que su salida pase primero por una función g(x), que se trata de un discriminador lineal o función de activación, véase una función que solo devuelve 1 o 0 (activación o desactivación) dependiendo de su entrada. Si su entrada es mayor o igual que cero o menor que 0 respectivamente. Esto hará que la función general del perceptrón ya similar a una neurona, f(e) siendo 'e' el vector de entrada, quede así:

$$f(e) = \begin{cases} 1 \rightarrow \sum_{j=1}^{n} w_j * e_j - \theta \geq 0 \\ 0 \rightarrow \sum_{j=1}^{n} w_j * e_j - \theta < 0 \end{cases}$$

Como mencionamos antes, suponiendo que tenemos 2 grupos de puntos de dos clases distintas separables linealmente, imaginemos la recta que las separa en dos espacios usando su ecuación genérica:

$$y = m * x + b$$

Que, en nuestro caso, podría tratarse de la siguiente ecuación:

$$y = (1/3) * x + 1$$

Para conseguir que a partir de esto podamos saber si se trata de un elemento perteneciente a un grupo u otro, lo transformaríamos a una inecuación, de tal forma que los puntos que cumplan la inecuación serán del primer grupo y los que no del segundo:

$$y - \left(\frac{1}{3}\right) * x \geq 1$$

Y que basándonos en el vector de dos características las cuales son representadas como *x* e *y*, podemos cambiar la fórmula a:

$$e1 - \left(\frac{1}{3}\right) * e2 \geq 1$$

Por lo tanto, se induciría que (1/3) sería un peso: w_2, y el peso 1: w_1 correspondería al valor 1. Finalmente, el 1 que lidera la inecuación sería nuestro valor θ .

Vemos de esta manera como el perceptrón basaría su aprendizaje en adecuar los pesos para cumplir con la clasificación propuesta, pero ¿cómo hace para saber cómo modificar su valor a mejor? Empecemos por corregir un problema de la función de activación.

La función de activación g(x) que hemos estado usando hasta el momento, usaba el mayor o igual para definir su salida y esta versión carece de una característica muy útil: no es derivable. Por ello, como función de activación es mucho más usual utilizar una función sigmoide, cuya expresión y forma son las siguientes:

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

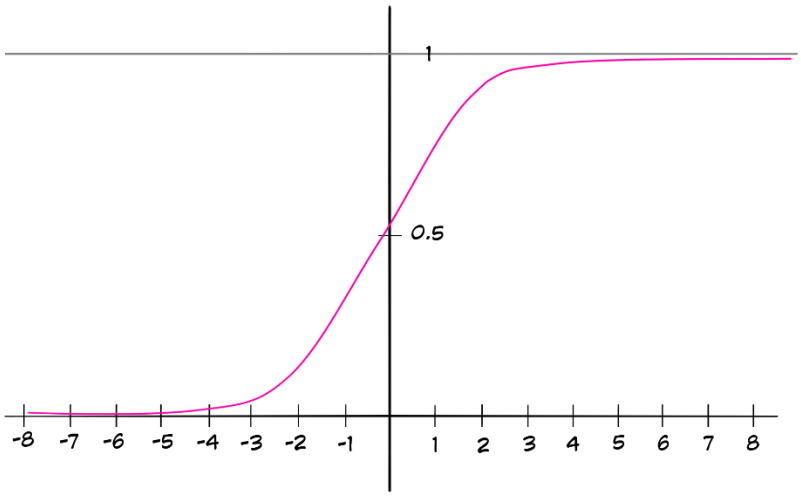


Ilustración 2 Representación gráfica de una función sigmoide

Vemos en la ilustración 2 como al contrario que la anterior, la sigmoide es continua en todo su rango (de -∞ a +∞), teniendo que sin contar un pequeño tramo alrededor de 0, todos los valores negativos de *x* acaban con un valor muy aproximado a 0 y los valores positivos de *x* acaban con un valor muy aproximado a 1. ¿En qué nos beneficia entonces que sea derivable? Para ello tenemos que ir al siguiente paso: el descenso por el gradiente.

En el aprendizaje del perceptrón lo que necesitamos en una primera instancia es intentar eliminar el error, véase la diferencia entre el valor real (la clasificación real de un elemento en nuestro caso) y el valor deducido por nuestro perceptrón. Dado que eliminarlo por completo es complejo, lo que realmente estamos buscando es minimizarlo. Si pensamos en obtener el valor mínimo de una función, lo primero que se nos viene a la cabeza es usar su derivada para ello obteniendo la x al igualar el resultado de esta a 0. Sin embargo, en nuestro caso no podemos resolverlo de forma algebraica, sino numéricamente, moviéndonos poco a poco desde un punto inicial bajando hasta llegar al mínimo.

Sin embargo, esto no es un método eficiente y puede ser mejorado utilizando el método del descenso por el gradiente. Esta última técnica trataría de aprovechar el valor de la pendiente de la función obtenida al realizar la derivada multivariable de la función de activación, al cual llamaríamos gradiente. De esta forma, el gradiente nos dará la dirección y sentido en la que aumenta la pendiente, véase donde el error es mayor, por

lo tanto, nuestro cometido es ir en contrario al gradiente para encontrar el mínimo global. Entonces, ¿cuál es el valor que el perceptrón debe cambiar para poder adaptarse a este descenso? Pues se trata de los valores que más influyen sobre las entradas: los pesos.

Teniendo ya la teoría acerca de la función sigmoide y el descenso por el gradiente ya solo queda saber cómo calcular el error. Por ello para hacer el modelo más adaptable a la sigmoide, el valor $-\theta$ que sirve como separador entre los valores que activan la neurona y los que no, pasará a ser otro de los pesos: w_0 cuyo valor de entrada e_1 asociado será siempre 1, uniéndose al sumatorio y por lo tanto siendo el resultado directo de este el que se tome en cuenta para realizar la función sigmoide:

$$f(e) = Sig(\sum_{j=0}^{n} w_j * e_j)$$

La función de error puede tener distintas formas siempre que cumpla con lo descrito anteriormente. Poniendo que representemos el resultado de la función de activación de una muestra como $h_{\vec{w}}(\vec{e})$, teniendo $\vec{e}$ como el número de inputs de la muestra, una forma de cálculo del error sería la siguiente:

$$err(\vec{w}) = \sum_{j=1}^{m} (h_{\vec{w}}(e^j) - res_j)^2$$

Siendo e_i las entradas correspondientes a una muestra de nuestro conjunto de ejemplos de entrenamiento, res_i el resultado correcto para cada elemento del conjunto de entrenamiento y *m* el número total de muestras.

Finalmente podemos obtener los nuevos valores de los pesos a partir de la siguiente fórmula:

$$\vec{w}_{t+1} := \vec{w}_t - \gamma * \frac{\partial err(\vec{w})}{\partial \vec{w}}$$

En donde γ es un valor muy importante al que llamaremos **tasa de aprendizaje o «learning rate»,** que se trata de un valor estático, que seleccionará el usuario al empezar el entrenamiento de la red, y que definirá la escala de la transformación que sufrirá el nuevo peso en comparación con el antiguo. Es a su vez el valor que indica el paso que se hace con el descenso del gradiente. Cabe destacar que no hay una forma de obtener la tasa de aprendizaje adecuada para cada caso, esta se debe conseguir a través de la experimentación.

Con los datos que tenemos ahora ya podemos dejar que el perceptrón aprenda a clasificar, teniendo como objetivo de su entrenamiento encontrar los pesos que le permitan clasificar con la mayor exactitud posible reduciendo el error.

Un perceptrón podrá diferenciar entre 2 grupos siempre que estos sean linealmente separables, pero cuando no lo son, no basta solo con tener uno. Hay que emplear un conjunto de ellos de manera que por la combinación de separadores lineales sea posible la agrupación. Es entonces cuando tenemos que hablar de su combinación: las redes neuronales *[18]*.

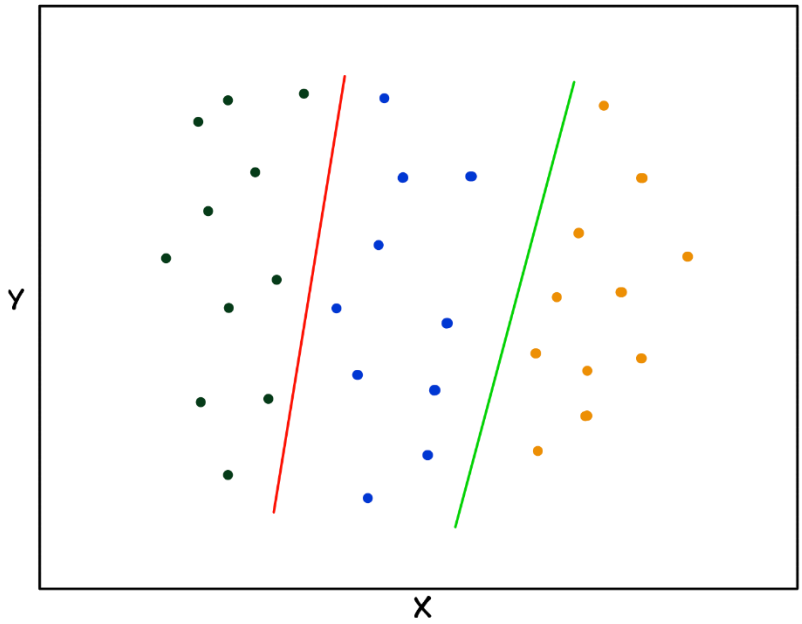


Ilustración 3 Representación de puntos para el ejemplo

Imaginemos un conjunto como el de la ilustración 3. El grupo de puntos de color azul podría ser descrito como aquel cuyos puntos se encuentran a la derecha de una primera recta como es la recta de roja y que a su vez se encuentran a la izquierda de la recta verde. Ya que ninguno de los puntos naranjas cumple los criterios, este puede ser un ejemplo de clasificación. Mirándolo de forma más visual en la ilustración 4, la neurona roja representaría la recta roja (función 1) y la neurona verde la recta verde (función 2), sin embargo, estas dos neuronas que forman la primera capa no son suficiente para determinar si un punto pertenece a un grupo u otro. Es necesaria una segunda capa que combine sus salidas y tome la decisión. En este caso nos bastará con que una neurona que compruebe si ambas neuronas se han activado, véase que haga un sumatorio de las entradas teniendo $\theta = 1.5$ (al tratarse de una combinación AND, el valor debe ser un poco menor que la suma de todas sus entradas activadas).

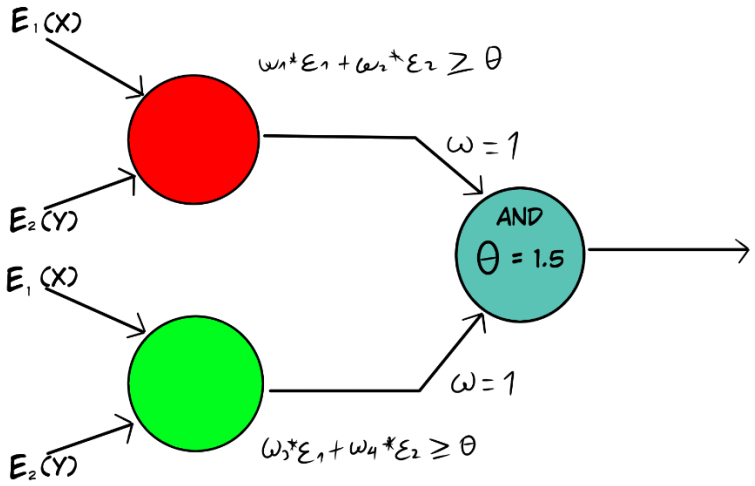


Ilustración 4 Representación de red neuronal básica para el ejemplo

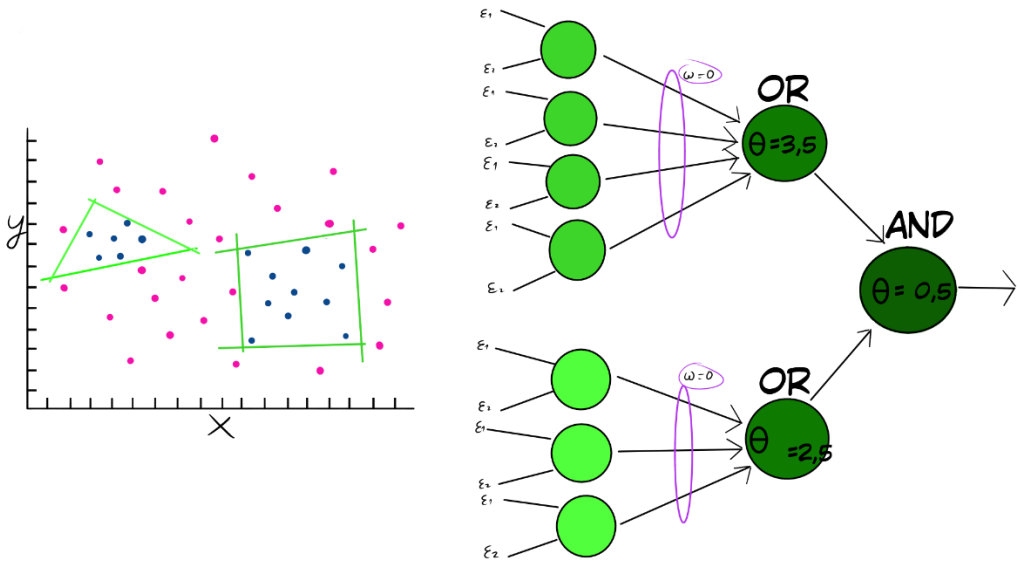


Ilustración 5 Representación de ejemplo más complejo

Para casos más complejos, como uno en el cual los puntos azules pudieran estar todos definidos por ser encontrados entre combinaciones de líneas más complejas, o habiendo varios grupos de puntos azules entre otros, se podrían combinar más neuronas en distintas capas, por ejemplo, buscando estructuras como las de la ilustración 5. Sin embargo, para casos aún más complejos como los que se suelen tratar normalmente, el hecho de establecer conexiones de forma manual buscando cubrir regiones específicas es algo prácticamente imposible para casos de mayor envergadura que los vistos en estos ejemplos. En general, se realizará una conexión completa de una capa con la siguiente (cada salida de una capa es entrada de cada una de las neuronas de la siguiente), siendo el entrenamiento o aprendizaje de la red el encargado de establecer los pesos convenientes para formar las relaciones adecuadas entre capas. Este proceso de aprendizaje sería similar al de una única neurona con el descenso del gradiente, solo que aplicado a una red entera. Entonces es cuando pasamos a hablar de las redes neuronales profundas.

Redes neuronales profundas

Las redes neuronales no sirven solo para clasificar entre dos grupos como se ha estado mostrando hasta ahora. De hecho, en el problema que se tratará en este documento, el número de grupos de clasificación más bajo es de 8 clases, llegando a las 143 clases. Aquí es cuando necesitamos muchas más capas de procesamiento que obtengan distintas características de las cuales como usuarios no nos daríamos cuenta, son necesarias las **redes neuronales profundas** *[10].* Como su nombre indica siguen siendo redes neuronales, sin embargo, estas cuentan con un mayor número de capas de neuronas entre ellos que se conectan unas con otras. Por lo general estas redes se componen de 3 partes (ejemplo en la ilustración 6):

- La capa de entrada: grupo de neuronas que recibe la entrada de datos del exterior, cada una de sus neuronas coge todo el conjunto de datos y lo procesa. Cada neurona genera su propia salida.
- Las capas ocultas o intermedias: son el conjunto de capas internas de la red. Cada capa conecta su salida con la entrada de la siguiente capa y van haciendo las decisiones necesarias para obtener características útiles sobre los datos obtenidos del paso anterior. Cabe destacar que en este ejemplo estamos describiendo la arquitectura **fully-connected**, esto es, capas en las que cada neurona tiene como entrada cada una de las salidas de la capa anterior. Existen algunas redes específicas que aplican técnicas o configuraciones de capas distintas para sus propios objetivos, sin embargo, la fully-connected es la más general.
- La capa se salida: es la capa que va a dar el resultado final. Al igual que las capas ocultas, sus neuronas reciben como entrada todas las salidas de la capa anterior (la última oculta), aunque con un objetivo distinto. Cada una de sus neuronas estará asociada a una de las posibles clasificaciones y dará como salida la probabilidad de que los datos que se han procesado caractericen a su clase. Pongamos un ejemplo que disponga de otra posible clasificación entre 2 clases.

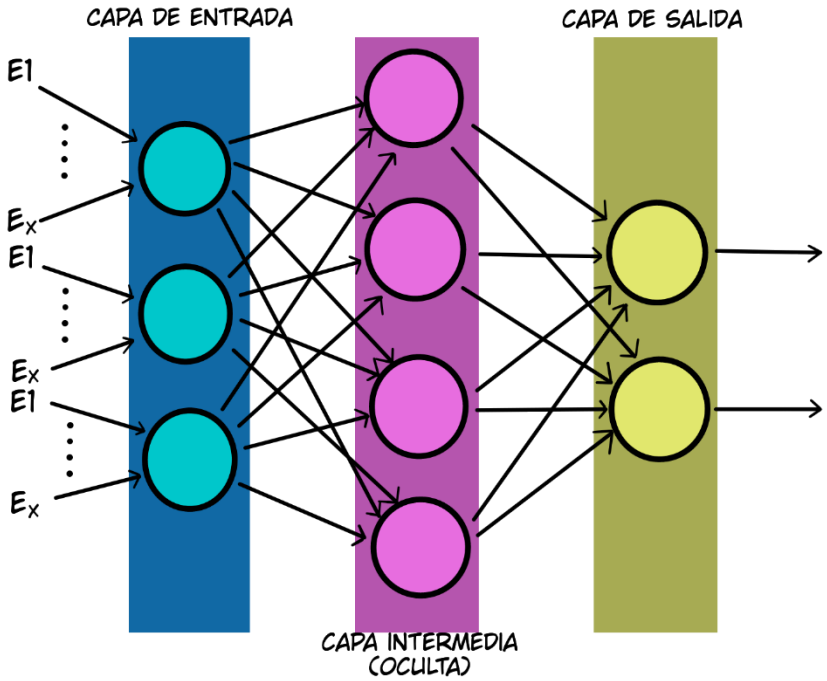


Ilustración 6 Estructura general de una red neuronal

Además, debido a la forma de la capa de salida, ya no realizaremos la función de activación sigmoide sobre cada neurona, sino que usaremos otro método. La **función softmax** se trata de una función similar a la sigmoide, teniendo resultados en el rango entre 0 y 1 pero que, a diferencia de la anterior, esta trabaja sobre un vector, haciendo cada uno de sus componentes un valor en el rango conformando así una distribución de probabilidad (la suma de todos los valores del vector es 1). La representación de la función softmax sería la siguiente:

$$softmax(\vec{v})_i = \frac{e^{v_i}}{\sum_{j=1}^{n} e^{v_j}}$$

En donde v_i es el resultado del cual queremos obtener el valor de softmax y v_j el resultado de cada una de las neuronas (incluida v_i).

Dado que ahora no tenemos una única neurona que diga si pertenece o no a una clase, será necesaria otra forma de interpretar los resultados obtenidos de la capa de salida. Por ello, introducimos la **codificación one-hot**, que se basa en generar un vector con un número de elementos igual al número de clases que pueden identificar a una entrada a la red. Se trata de una expresión del resultado en la cual todos los elementos del vector son representados con un 0, excepto el elemento correspondiente a la clase que mejor se adecua en la capa de salida, la cual se representa con un 1.

Este cambio hace que también sea necesario hacer cambios en el cálculo del error para que tenga en cuenta el error de cada deducción, véase que en este caso deberemos comparar el resultado de la función softmax con las etiquetas en codificación one-hot. El resultado tiende a ideal cuanto más se acerque el elemento de la clase correcta a 1 y más se acerquen a 0 el resto. El error para cada elemento procesado por la red se calcularía de esta forma:

$$err = \sum_{i=1}^{n} (et_i - res_i)^2$$

En el cual et_i es el valor esperado de la etiqueta (0 o 1) para una clase del vector y res_i el valor obtenido en esa posición del vector, siendo n el número de elementos en este. Así, el cálculo del error total sería:

$$totalErr = \frac{1}{m} \sum_{j=1}^{m} [\sum_{i=1}^{n} (et_i^j - res_i^j)^2]$$

Por último, acerca del mismo error, toca añadir el concepto de **«Backpropagation» o retropropagación**. El algoritmo de la retropropagación consiste en empezar calculando las derivadas parciales usadas previamente para el cálculo de errores, solo en función de los pesos de la última capa. A partir de ahí propagaremos (de ahí el nombre) el cálculo de las derivadas parciales para el error a las capas anteriores, hasta llegar a la capa de entrada. De esta forma nos aseguramos de que se corrigen los pesos adecuadamente y de tratar aquellos puntos que son los que más influyen en el error que pueden estar en medio de la red.

2.2 Conceptos adicionales de las redes neuronales

Aun habiendo explicado cómo funcionan las redes neuronales profundas, quedan algunos puntos para llegar a comprender como se entrenan o factores que tienen importancia en su entrenamiento.

Hemos comentado que para entrenar una red neuronal es necesario disponer de un conjunto de muestras con la cual la red pueda tener ejemplos de las distintas

clasificaciones para un grupo de elementos. A estos conjuntos de muestras se les suele denominar **datasets**¸y contienen tanto los datos de los elementos como los grupos en los que se clasifican y que la red debe deducir. Si bien se ha hablado continuamente de un dataset de entrenamiento para que el modelo pueda aprender, la verdad es que es necesario dividir el conjunto en mínimo dos subconjuntos: el que usamos para entrenar la red y otro que nos permita comprobar que nuestro modelo realmente realiza el ejercicio de clasificación. Para ello se crea un conjunto de validación, con elementos distintos al de entrenamiento, aunque similares, que usaremos para ver el porcentaje de acierto que tiene la red al pasar por ella estos datos que, si bien son clasificables de la misma forma, no son exactamente los mismos. Véase que cuando se procesa el conjunto de validación no se está entrenando el modelo sino simplemente se comparan los resultados con los que debería dar para tomar la decisión de si ya se trata de un modelo funcional o si por el contrario es necesario más entrenamiento para que llegue al objetivo deseado.

Hay que tener en cuenta que, para un número elevado de ejemplos en ambos conjuntos, la suma de todos los errores para realizar el aprendizaje de la red puede ser algo masivo. Por ello es que en la práctica no se entrena directamente con el conjunto entero de datos, sino que se toman conjuntos más pequeños de muestras lo suficientemente variados y significativos como para que el resultado del entrenamiento sea parecido al que obtendríamos con el conjunto entero. Estos conjuntos llamados **batchs o mini-lotes** permiten una mejora en la velocidad de entrenamiento del modelo bastante significativo. Cuando se entrenan todos los batchs que se pueden obtener del conjunto se dice que se ha completado una **epoch** o época.

Existe la posibilidad de que durante el continuo entrenamiento y validación de la red nos encontremos con un problema, y es que la red podría no estar progresando más en su aprendizaje a pesar de que la validación nos indica que aún no tiene una buena precisión. Existen 2 posibilidades de que puede provocarlo: podría ser que la red no fuera lo suficientemente compleja como para resolver el sistema a tratar y que por lo tanto sea necesario mejorar el modelo o por el contrario podría tratarse de un problema de overfitting. **Overfitting o sobreajuste** es el caso que se da cuando si bien el entrenamiento está funcionando, el aprendizaje que presenta se adecua demasiado al conjunto de entrenamiento, provocando que aunque los elementos del conjunto de validación tengan características en común con los que se usan para entrenar, los valores que tiene en cuenta la red para la clasificación son muy específicos del primer conjunto y no se adaptan a la obtención de características general que se buscaría para cualquier caso, véase: aprende de casos particulares sin ser capaz de adaptarse a nuevos. Este error puede deberse a crear una red demasiado compleja para el caso a tratar (capas ocultas en exceso sacarán características muy específicas) o sobre-entrenamiento.

Una posible forma de reducir este problema es mediante el **Dropout** *[17]*. Se trata de una técnica que consiste en desactivar de forma aleatoria y basándonos en un porcentaje un conjunto de neuronas en una capa, pudiendo ejecutarlo en varias capas durante el entrenamiento. Al estar desactivadas, las neuronas restantes que están activas deben hacer más por aprender para conseguir mejor precisión, pues como mencionan los descubridores de esta técnica *[17] «La mejor forma de "regularizar" un modelo de tamaño fijo es hacer promedio de las predicciones para todas las configuraciones posibles de parámetros.»*. De esta forma, al cambiar las neuronas

constantemente durante el entrenamiento estamos generalizando el aprendizaje para los múltiples modelos que surgen de la desactivación de neuronas.

Esta es la forma básica de hacer una red, sin embargo, si generamos un modelo, nos daremos cuenta de que es complejo saber a priori qué tamaño debemos usar, la tasa de aprendizaje que ya hemos mencionado, el número de neuronas que formarán algunas de las capas ocultas ("*de en medio*") del modelo, etc. Estos valores son los que comúnmente se llaman **hiperparámetros**, introducidos enteramente por el usuario y difíciles de establecer a primera instancia, pues es complicado obtener su valor óptimo. Es necesario guiarse de los encontrados en otros modelos de redes o los que mejores resultados dan en múltiples ejecuciones para decidir qué valores deben formarlos. El porcentaje que tiene en cuenta la técnica de dropout también es considerado un hiperparámetro generalmente (aunque sus posibles valores están más establecidos).

2.3 Convolución: redes convolucionales.

Como hemos estado observando, las redes neuronales a partir de un conjunto de datos son capaces de modificar sus pesos para intentar resolver un problema dados varios ejemplos del problema, en muchos casos siendo problemas de clasificación. Sin embargo, el único hecho de crear una red neuronal no implica que puedas resolver cualquier problema: la red necesita ser lo suficientemente compleja como para obtener las características necesarias de la entrada que permitan deducir el resultado correcto. De la misma manera, que una arquitectura de red sirva para un caso no significa que vaya a servir para otro distinto. Por ello, además de modelos entrenados para distintas tareas, a lo largo de los años se han realizado varios estudios para desarrollar nuevas arquitecturas de red que faciliten trabajos concretos. En nuestro caso, para sacar características de imágenes es evidente el uso de **redes convolucionales** *[4]***.**

Primero recordemos el concepto de **la convolución**. La convolución es la definición de un operador matemático que representa la integral del producto entre dos funciones, estando una de las funciones desplazada una distancia *t* de la otra. En nuestro caso nos interesa concretamente la versión bidimensional de este operador (t dividido en i y j) y discreta con paso 1:

$$(f * g)(i,j) = \sum_{-\infty}^{\infty}\sum_{-\infty}^{\infty} f(x,y)g(i-x,j-y)$$

¿Qué tiene de especial esto con respecto a lo que ya habíamos utilizado con anterioridad en las redes neuronales? Siendo la versión bidimensional y discreta la que vamos a utilizar, esto significaría que en vez de tener que linealizar la imagen para pasarla por el conjunto de neuronas, como hacíamos anteriormente, podríamos pasarlo por la red usando un concepto más cercano a lo que es la entrada de una imagen: usando matrices. Con la imagen convertida en matrices, por lo general 3 matrices (rgb), cada neurona se encargaría de procesar una parte o bloque de esa matriz. Imaginemos que una neurona capta un bloque de 3x3 de la imagen. La neurona dispondrá de su propio bloque de tamaño 3x3 formado por pesos, denominado **kernel**, y que junto con

el bloque del mismo tamaño de la imagen original formarán las 2 funciones (f y g) que tienen lugar en la convolución, quedando la fórmula con un aspecto similar a este:

$$conv(i,j) = \sum_{y=0}^{2} \sum_{x=0}^{2} kern(x,y) * image(i-x, j-y)$$

Supongamos ahora que tenemos una imagen que pixel a pixel, solo con valores en blanco y negro, representada como la siguiente:

225	225	225	225	225	225	225	225	225	225
225	225	225	225	225	225	225	225	225	225
225	225	225	225	225	225	225	225	225	225
225	225	225	225	225	225	225	225	225	225
0	0	0	0	0	0	225	225	225	225
0	0	0	0	0	0	225	225	225	225
225	225	225	225	0	0	225	225	225	225
225	225	225	225	0	0	225	225	225	225
225	225	225	225	0	0	0	0	0	0
225	225	225	225	0	0	0	0	0	0

-1	-1	-1
0	0	0
1	1	1

Siendo el valor 255 el color blanco y el 0 el color negro y teniendo a su vez un kernel como el siguiente:

225	225	225	225	225	225	225	225	225	225
225	225	225	225	225	225	225	225	225	225
225	225	225	225	225	225	225	225	225	225
225	225	225	225	225	225	225	225	225	225
0	0	0	0	0	0	225	225	225	225
0	0	0	0	0	0	225	225	225	225
225	225	225	225	0	0	225	225	225	225
225	225	225	225	0	0	225	225	225	225
225	225	225	225	0	0	0	0	0	0
225	225	225	225	0	0	0	0	0	0

Comprobamos con los dos bloques mostrados que, si realizamos la convolución sobre una zona todo a blanco (recuadro rojo), el resultado que devolvería es 0. Sin embargo, realizándolo sobre una zona no tan homogénea y que pudiera asemejarse a una línea negra horizontal (recuadro verde), comprobamos que el resultado pasa a ser mayor que 0. Véase, que con este ejemplo hemos creado un kernel que permite la detección de bordes horizontales en una imagen a partir de fijarse en bloques concretos de esta. Véase que al solo realizarlo con bloques individuales dan un único resultado, pero el proceso de convolución de la imagen entera daría lugar a una matriz de datos, cada una con el resultado de realizar la convolución sobre un bloque de la imagen.

Esto llegaría a simular la forma en la que los seres humanos reconocen las imágenes: al no linealizar los datos, conseguimos un recuerdo de la posición de los elementos de la imagen. La salida de esta primera capa de neuronas también será reconocida como bloques como entrada de la siguiente capa convolucional, (gráficamente se vería como en la ilustración c), lo cual provoca que se mantenga esa noción de la posición a lo largo de la red. Este método de reconocer una imagen visualizando sus distintas partes se basa en la manera en que como humanos podemos reconocer a una persona analizando en un mismo instante las distintas facciones de su cara.

Sin embargo, lo que nos importa aquí es la forma de usarlo en forma de capa de una red convolucional, así como entrenarlo de forma que esta aprenda a obtener las características necesarias. Al contrario que las ya mencionadas capas fully-connected, en el caso de las capas convolucionales como mencionamos hace nada, cada neurona se centra únicamente en un bloque de salidas de la capa anterior, así como buscan una disposición bidimensional de las entradas, que visualmente es similar al ejemplo explicado anteriormente en el que se designa un kernel a un bloque de la imagen que es la entrada. Hay que destacar además que todas las neuronas que forman una capa convolucional comparten los mismos pesos, véase el funcionamiento de cada neurona

es una réplica del funcionamiento de las demás neuronas que trabajan con ella. El número de pesos de la capa dependerán del kernel definido para estas. Por ello es que cada capa de una red convolucional estará designada a un caso concreto, por ejemplo, una capa solo para los bordes horizontales, la siguiente para los bordes verticales seguida de una que detecte diagonales, etc.

Otro aspecto que diferencia a las capas convolucionales de las fully-connected es su uso de **tensores**. Los kernel como ya hemos demostrado permiten obtener características de una imagen, pero con una sola característica nos es imposible clasificar nada (a menos que el problema sea tan simple como diferenciar si una imagen presenta formas en horizontal o en vertical, ahí tal vez tendría uso). Un tensor es aquel elemento que se representa en tres dimensiones o más (al contrario de los elementos que hemos tratado hasta ahora que podían ser caracterizados como bidimensionales o matrices). Con el uso de los tensores podemos permitirnos capas que usen más de un filtro de neuronas (como si de un conjunto de capas que toman las mismas entradas se tratara), de forma que, si por ejemplo quisiéramos filtrar tres características, al final se crearía un tensor que representaría los resultados de los 3 filtros, cada uno representado en un **canal** del tensor. Estos resultados en forma de tensor serán entradas a su vez de otra capa convolucional que tomaría también bloques de estos resultados, solo que además estaría teniendo en cuenta los distintos canales generados para ese bloque por la capa convolucional anterior. Estás tomando información de las características que ha encontrado la anterior capa, obtendrán información aún más compleja basándose en esta primera filtración (tomando el ejemplo de las líneas horizontales, si dos bloques contiguos continúan la recta o combinaciones entre estos podrían ser un ejemplo de un segundo filtrado). Si en vez de otra capa convolucional, pasáramos a la parte de clasificación formada por capas fully-connected, dado su forma de actuar sería necesario pasarlo de un tensor a un único vector, véase se linealizaría el tensor. Estas capas fully-connected finalizarían con la capa de salida que permitirá su clasificación. Debido a esto último, las redes convolucionales suelen dividirse en dos partes: el bloque dedicado a la extracción de características (convolucional) y el bloque de clasificación (fully-connected)

Hay que mencionar que al igual que el resto de tipos de redes neuronales, las redes convolucionales aprenden por sí mismas mediante métodos como el descenso por el gradiente, siendo estas mismas las que deciden qué es lo esencial que deben filtrar sus capas (véase, no será el usuario quien diga qué capa debe buscar líneas horizontales, diagonales, etcétera, sino que será la misma red por el aprendizaje la que decidirá qué es lo que al filtrarse define un importante hecho para la clasificación, al igual que se mencionó con las fully-connected).

Queremos aclarar que el tamaño del kernel será al igual que otros valores, un hiperparámetro puesto por el usuario.

A pesar de haber terminado con la explicación general de su funcionamiento, quedan dos cosas importantes que destacar, la función de activación que utilizamos y una forma de simplificar las costosas capas convolucionales:

- **ReLU:** La función ReLU al igual que la función sigmoide o la softmax se trata de una función de activación, una bastante adecuada para las capas convolucionales y que se expresa como:

$$ReLU(x) = max(0, x)$$

Esto provoca que para valores mayores que 0 conserve el valor de la salida, siendo 0 directamente cuando la salida esté por debajo de cero. De esta forma, en el descenso del gradiente la derivada para los valores positivos (los más alejados de 0) tenderá siempre a 1. Sin embargo, para los valores negativos la salida siempre será 0. Esto provoca que solo aquellas neuronas cuyos valores de salida sean positivos influyan en la representación del resultado, provocando a su vez una reducción de su complejidad, así como una mejor propagación del gradiente (al no ser tan pequeños los valores de la salida para sus valores positivos).

- **Max-pooling:** Debido a la obtención de características que ha sido explicada, habrá casos en los cuales debido a una gran cantidad de filtros incluidos o una búsqueda de características muy específica, el tamaño del tensor sea difícil de manejar. Max-pooling consiste en una reducción de los datos de forma que, para cada canal de un tensor entrante, agrupando sus resultados por ejemplo en bloques de 2x2, reduce el valor de las 4 celdas a una única celda que tendrá el valor de la celda que presente el mayor valor del conjunto, obteniendo un nuevo tensor con la mitad de las dimensiones del original. De esta manera no se reduce tan drásticamente la información, pues además estos bloques consisten en aspectos de la imagen muy específicos cuya diferencia entre las celdas adyacentes no es tan grande.

2.3 VGG.

Ahora que ya tenemos el conocimiento necesario, ya es posible crear un modelo que sea de uso para nuestro propósito de clasificación. Sin embargo, ¿es necesario crear todo el modelo desde cero cuando ya hay algunas estructuras que dan resultados de precisión alta para otros casos de clasificación? No. Existen estructuras nacidas de otros proyectos de investigación y que incluso participan en competiciones para declarar qué tecnología de las desarrolladas es mejor. En nuestro caso nos interesa aquellas para la clasificación de imágenes, por lo que nos conviene observar los resultados de la competición ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) o «Reto de Reconocimiento Visual para Gran-Escala de ImageNet» que compara los resultados de distintos modelos obtenidos en investigaciones ante el dataset preparado ImageNet. Durante varios años la arquitectura ganadora del certamen de 2012 Alexnet, desarrollada por el equipo «SuperVision» fue considerada como una arquitectura base de la rama de Machine Learning. Sin embargo, la configuración que vamos a utilizar no se trata de esta, sino de una que vendría a sobrepasar esta arquitectura: **Las redes neuronales VGG** *[16].*

La arquitectura VGG (Visual Geometry Group) es un modelo de red neuronal profunda para reconocimiento de imágenes propuesto por Karen Simonyan y Andrew Zisserman, investigadores del departamento de ciencias de la ingeniería de la Universidad de Oxford. Basada en la arquitectura AlexNet anteriormente mencionada, VGG vio la luz en el certamen de 2014 de la misma competición que ganó en 2012 su predecesora, en donde fue reconocida por su gran eficiencia, quedando en el top 5 con un 92.7% de precisión sobre ImageNet.

El modelo base está prefijado para la entrada de imágenes de 224x224 imágenes RGB durante el entrenamiento, únicamente substrayendo el valor medio RGB a cada pixel. Al contrario que la arquitectura en el que se basa VGG, los kernels de la gran mayoría de capas que forman la red utilizan el menor tamaño posible de kernel (el mínimo que permite un buen reconocimiento de la imagen), usando tamaños de 3x3. En algunas configuraciones de la arquitectura se hace uso también de kernels de 1x1 lo cual llevaría a una simple transformación lineal de la entrada. Por otro lado, el paso convolucional (convolutional stride) se queda en valores de 1 pixel lo que permite que se mantenga la resolución espacial tras el proceso convolucional. El proceso de max-pooling que tiene lugar tras algunas de las capas convolucionales del modelo está fijado siempre a ventanas de 2x2 con pasos de 2 píxeles.

Como hemos comentado anteriormente, el kernel de 3x3 que usa VGG fue una decisión bastante alejada respecto a lo que se veía en otros modelos de la competición (como en AlexNet, en el cual se basa). Así que ¿Por qué se decidió pasar a esta configuración de la previamente usada de 7x7, por ejemplo? En primer lugar, aportaría 2 capas de rectificación adicionales, lo cual influiría haciendo la función de decisión más discriminativa y, en segundo lugar, disminuiría el número de parámetros (pesos) al solo tratar con 9 (3x3) para cada capa. En cuanto a la inclusión del kernel 1x1, esto permite incrementar la no-linealidad de la función de decisión sin por ello tener que afectar a las demás capas convolucionales.

Así que en general, el modelo VGG procesa la entrada de 224x224 por un «trayecto» de capas convolucionales con la configuración explicada con anterioridad tras lo cual se linealiza el resultado y se pasa por un conjunto de tres capas fully-Connected. Este último bloque de capas no cambia en ninguna de las configuraciones de la VGG. Ambas capas, tanto convolucionales como fully-Connected son seguidas de una función de activación ReLU.

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Ilustración 7 Configuraciones aceptadas de estructura VGG. Imagen perteneciente al estudio de K. Simonyan y A. Zisserman[16]

Durante este proyecto, por su complejidad media, estaremos utilizando la configuración VGG16 (configuración D), caracterizada por el uso de 16 capas de pesos (13 capas convolucionales de extracción de características y las 3 capas fully-connected que forman la parte clasificatoria comunes en todas las variaciones) que mostramos a continuación.

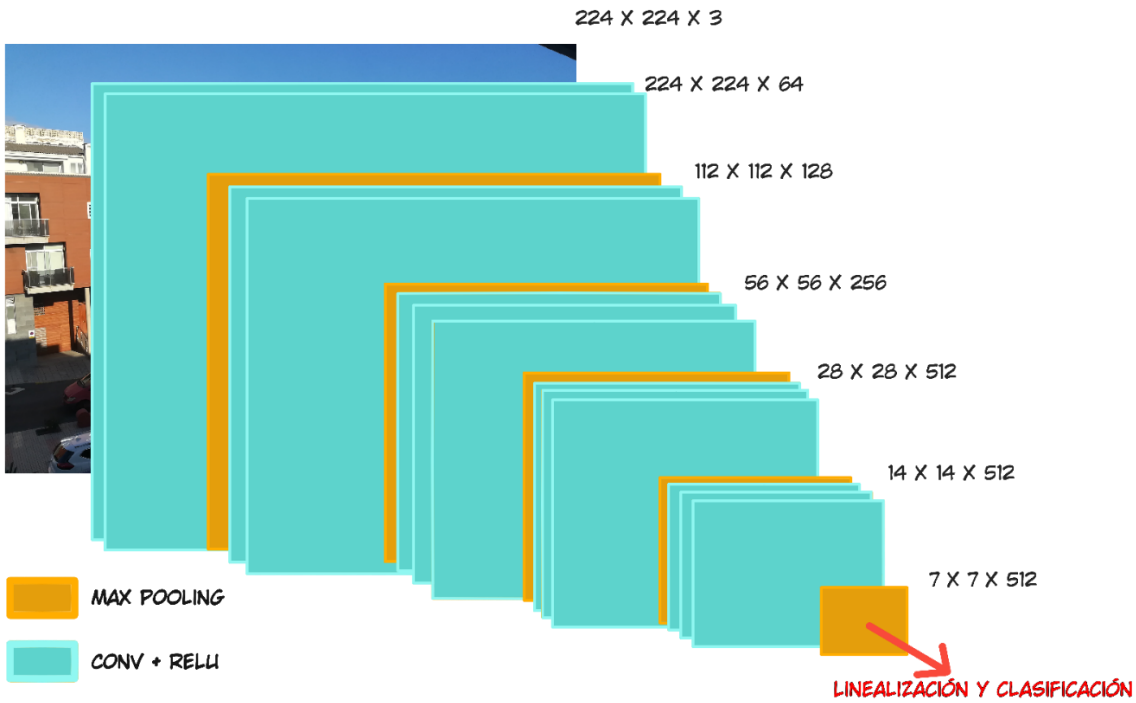


Ilustración 8 Representación visual de la estructura convolucional de una red VGG_16

Con la entrada especificada anteriormente de 224 x 224 x 3, que correspondería a una imagen de 224 de alto y 224 de ancho con las 3 capas RGB, el bloque de capas convolucionales empieza con dos capas ambas con la misma configuración: un kernel de 3 x 3 que se repite en la mayoría de las capas y un ancho de 64, manteniendo las dimensiones de 224 x 224. El resultado se procesa a través de una capa max-pooling que reducirá sus dimensiones a 112 x 112, seguido de otras dos capas convolucionales de 128 de ancho. Continuamos con otra capa max-pooling que reducirá sus dimensiones a 56 x 56 y que dará paso en este caso a 3 capas, ahora con un ancho de 256. Pasando por una tercera capa maxpooling, las dimensiones serán ahora de 28 x 28 y a partir de este punto todas las capas convolucionales a comentar tendrán un ancho de 512. La estructura continúa con 3 capas convolucionales para volver a ser reducidas las dimensiones a 14 x 14 por una cuarta capa max-pooling seguida de otro bloque de 3 capas convolucionales que finalizarán con una última capa max-pooling que terminará dando una salida de 7 x 7 x 512.

Cabe destacar que, si bien la configuración D es bastante similar a la configuración C, estas se diferencian en que la configuración C tiene configurado para los tres últimos bloques de tres capas convolucionales que la última capa de cada uno de estos tenga la mencionada configuración en la cual el tamaño del kernel pasa a ser de 1 x 1.

A partir de este punto lo siguiente a realizar es linealizar la salida preparándola así para su clasificación con las capas fully-connected. Este bloque de clasificación está formado por 3 capas:

- Una primera capa cuyo tamaño del vector de entrada es el del resultado de la linealización, 25088 y que como salida tiene un vector de tamaño 4096.
- Una segunda capa con el mismo tamaño de entrada que de salida: 4096.

- Una tercera y última capa cuyo tamaño de entrada es 4096 y cuyo tamaño del vector de salida es el mismo que el conjunto de categorías posibles en las que se puede clasificar: 1000 para el caso original.

Todo acaba finalmente con una capa softmax que nos da los resultados finales de la clasificación.

La arquitectura VGG es, en general una arquitectura que proporciona muy buenos resultados, sin embargo, hay que tener en cuenta que cuenta con la desventaja de que el uso de capas convolucionales la convierte en una arquitectura en extremo lenta de entrenar y que los pesos de la arquitectura pueden resultar un problema para el espacio en disco o ancho de banda que se use.

2.3 Modelos actuales.

El estado del arte en cuanto a estructuras para redes neuronales se refiere, está predominado junto con VGG por otras arquitecturas que merecen ser mencionadas.

AlexNet

Estructura ganadora del certamen ILSVRC del año 2012. *[8]* Se trata de una estructura que usa capas convolucionales, que, si bien tuvo el mejor resultado de su año, ha decaído por la aparición de VGG, la cual se basa en AlexNet.

ResNet

ResNet (o red residual) *[5]* es el nombre con el que se conoce al modelo que presentaría Microsoft y que sería la ganadora del ILSVRC del año 2015. Conociendo como demasiada profundidad en una red neuronal puede llevar a una caída de sus resultados de clasificación, esta red propondría una alternativa usando bloques residuales. Su funcionamiento consiste en incluir en la entrada de algunas capas además de la salida de la capa anterior, la salida de alguna de las capas anteriores que no sean la última. De esta forma, hay datos que se saltan algunas capas convolucionales y por lo tanto el efecto de desaparición del gradiente, ocasionado debido a la profundidad, disminuye en esos caminos alternativos que se crean. Por lo tanto, podemos aumentar la profundidad sin miedo a perder en exactitud de clasificación, teniendo mínimo el mismo resultado que su versión sin las capas extras. Por lo tanto, en este caso, una mayor cantidad de capas solo puede llevar a subir la precisión de la red.

GoogleLeNet / Inception

GoogleLeNet *[19]*, presentada por Google sobre el año 2015, aplica un concepto distinto a la profundidad. El tamaño de kernel puede ser una dura decisión que tomar, pues los tamaños grandes toman datos más generales a lo largo de la imagen, y los tamaños

pequeños son buenos detectando pequeñas características distribuidos por la imagen. Los investigadores de este modelo propusieron en vez de indagar en añadir capas que aumentaran la profundidad o longitud, se centraron en como aumentar el ancho que coge cada capa. Por ello, la red aplica unos módulos llamados Inception formados por cuatro capas procesadas en paralelo (kernel 1x1, kernel 3x3, kernel 5x5 y max-pooling) que concatenan sus resultados y por lo tanto sus distintas formas de caracterizar la entrada. Estos bloques luego se concatenan unos con otros en profundidad. De esta forma, cada bloque inception, visto ahora como capa, obtendrá una mayor cantidad de atributos de la entrada que pasarle a la siguiente capa.

Capítulo 3

Herramientas

3.1 Estudio previo

Durante las primeras semanas el principal trabajo fue el estudio de la temática a trabajar: inteligencia artificial y redes neuronales.

Se consultaron diversas fuentes tanto de carácter general (véase conceptos como las capas convolucionales, data augmentation) así como documentos que mostraban casos específicos de redes neuronales para otros objetivos de los cuales se obtuvo inspiración para algunas de las tecnologías usadas en el código.

Todos los documentos revisados constan en la bibliografía.

3.2 Dataset

Una vez realizados los anteriores pasos pasamos a realizar la búsqueda de un dataset que nos sea útil para el problema a investigar. Tratamos de no usar el dataset “fashion mnist” ya que, si bien trae un amplio número de elementos con los cuales entrenar nuestra red, consideramos que no dispone de las suficientes especificaciones en cuanto a las diferentes clases en las cuales se puede clasificar cada elemento, véase si bien tiene un número aceptable de ejemplos (60000 para el conjunto de entrenamiento), cada uno de estos solo puede clasificarse en una de entre 10 clases, siendo insuficiente para nuestra búsqueda de una obtención de características más profunda.

Por ello, de entre varios datasets buscados por la red nos decantamos por el que ofrece Param Aggarwal en su espacio en Kaggle: Fashion Product Images Dataset ***[2]***. Para quien desconozca de Kaggle, se trata de un servicio web para el desarrollo de inteligencia artificial y que además dispone de una larga lista de datasets proporcionados por la comunidad de la web para su uso público. El dataset usado en cuestión está formado por aproximadamente 44.400 imágenes a color, cada una de ellas identificada con un número. Las imágenes representan múltiples productos que se pueden encontrar en tiendas de moda juntando tanto prendas de arriba, prendas de abajo, ropa interior, calzado como otros objetos que se relacionan menos con la ropa como bolsas y joyería entre otros productos. Cada producto está detallado en el archivo styles.csv que además de las referencias a la localización de cada imagen, incluye para cada una de estas el nombre del producto y las diversas categorías en las que se puede clasificar:

- masterCategory (Categoría maestra): 7 clases que clasifican muy generalmente la categoría del producto.
- subcategory (Subcategoría): 45 clases que profundizan un poco más en el tipo de producto, aunque no lo suficiente en prendas de ropa. Vendría a tratarse de un segundo nivel más específico que MasterCategory.

- articleType (Tipo de artículo): 143 clases que profundizan mucho más en el tipo de cada elemento del dataset, teniendo más clases de prendas de arriba y prendas de abajo. Se trata del tercer nivel de clasificación que es más específico que los dos anteriores.
- baseColour (Color base): 47 clases que identifican el color base de un producto o el color predominante en estos. Esta junto con las de más adelantes ya suponen clasificaciones sin relación con el resto de las categorías. Véase que a partir de aquí no representan niveles como si lo hacen MasterCategory, SubCategory y ArticleType.
- season (Temporada): 4 clases que representan la temporada o estación para la que está pensada el producto.
- year (Año): Año en el que salió el producto. Debido a su arbitrariedad no se tendrá en cuenta para ninguna clasificación.
- usage (Uso): clases que muestran el tipo de uso u ocasión de uso para la cual está pensado dicho producto.
- productDisplayName (Nombre del producto): nombre del producto en tienda, que, si bien pueden ser similares entre ellos, son únicos de cada uno y por lo tanto se descarta su uso.

En la tabla a continuación disponen de una representación de las distintas clases en las que se puede clasificar cada categoría:

CLASIFICACIÓN

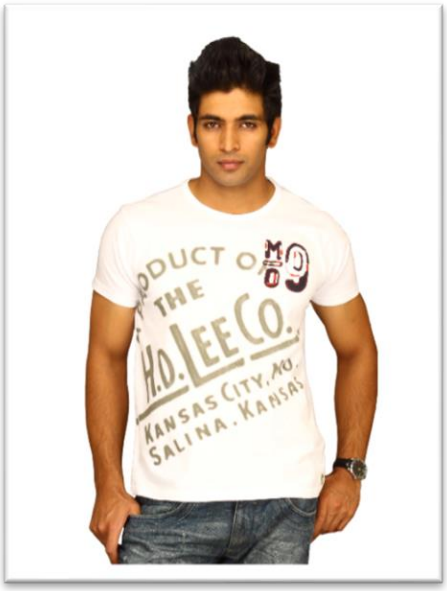
MASTERCATEGORY (CATEGORÍA)	Apparel, Accessories, Footwear, Personal Care, Free Items, Sporting Goods, Home
SUBCATEGORY (SUBCATEGORÍA)	Topwear, Bottomwear, Watches, Socks, Shoes, Belts, Flip Flops, Bags, Innerwear, Sandal, Shoe Accessories, Fragrance, Jewellery, Lips, Saree, Eyewear, Nails, Scarves, Dress, Loungewear and Nightwear, Wallets, Apparel Set, Headwear, Mufflers, Skin Care, Makeup, Free Gifts, Ties, Accessories, Skin, Beauty Accessories, Water Bottle, Eyes, Bath and Body, Gloves, Sports Accessories, Cufflinks, Sports Equipment, Stoles, Hair, Perfumes, Home Furnishing, Umbrellas, Wristbands, Vouchers
ARTICLETYPE (TIPO)	Shirts, Jeans, Watches, Track Pants, Tshirts, Socks, Casual Shoes, Belts, Flip Flops, Handbags, Tops, Bra, Sandals, Shoe Accessories, Sweatshirts, Deodorant, Formal Shoes, Bracelet, Lipstick, Flats, Kurtas, Waistcoat, Sports Shoes, Shorts, Briefs, Sarees, Perfume and Body Mist, Heels, Sunglasses, Innerwear Vests, Pendant, Nail Polish, Laptop Bag, Scarves, Rain Jacket, Dresses, Night suits, Skirts, Wallets, Blazers, Ring, Kurta Sets, Clutches, Shrug, Backpacks, Caps, Trousers, Earrings, Camisoles, Boxers, Jewellery Set, Dupatta, Capris, Lip Gloss, Bath Robe, Mufflers, Tunics, Jackets, Trunk, Lounge Pants, Face Wash and Cleanser, Necklace and Chains, Duffel Bag, Sports Sandals, Foundation and Primer, Sweaters, Free Gifts, Trolley Bag, Tracksuits, Swimwear, Shoe Laces, Fragrance Gift Set,

	Bangle, Nightdress, Ties, Baby Dolls, Leggings, Highlighter and Blush, Travel Accessory, Kurtis, Mobile Pouch, Messenger Bag, Lip Care, Face Moisturisers, Compact, Eye Cream, Accessory Gift Set, Beauty Accessory, Jumpsuit, Kajal and Eyeliner, Water Bottle, Suspenders, Lip Liner, Robe, Salwar and Dupatta, Patiala, Stockings, Eyeshadow, Headband, Tights, Nail Essentials, Churidar, Lounge Tshirts, Face Scrub and Exfoliator, Lounge Shorts, Gloves, Mask and Peel, Wristbands, Tablet Sleeve, Ties and Cufflinks, Footballs, Stoles, Shapewear, Nehru Jackets, Salwar, Cufflinks, Jeggings, Hair Colour, Concealer, Rompers, Body Lotion, Sunscreen, Booties, Waist Pouch, Hair Accessory, Rucksacks, Basketballs, Lehenga Choli, Clothing Set, Mascara, Toner, Cushion Covers, Key chain, Makeup Remover, Lip Plumper, Umbrellas, Face Serum and Gel, Hat, Mens Grooming Kit, Rain Trousers, Body Wash and Scrub, Suits, Ipad
BASECOLOUR (COLOR BASE)	Navy Blue, Blue, Silver, Black, Grey, Green, Purple, White, Beige, Brown, Bronze, Teal, Copper, Pink, Off White, Maroon, Red, Khaki, Orange, Coffee Brown, Yellow, Charcoal, Gold, Steel, Tan, Multi, Magenta, Lavender, Sea Green, Cream, Peach, Olive, Skin, Burgundy, Grey Melange, Rust, Rose, Lime Green, Mauve, Turquoise Blue, Metallic, Mustard, Taupe, Nude, Mushroom Brown, Fluorescent Green, nan
SEASON (TEMPORADA)	Fall, Summer, Winter, Spring, nan
YEAR (AÑO)	2011, 2012, 2016, 2017, 2015, 2014, 2010, 2013, 2018, 2019, 2007, 2009, 2008
USAGE (USO)	Casual, Ethnic, Formal, Sports, Smart Casual, Travel, Party, Home
PRODUCTDISPLAY NAME (NOMBRE DEL PRODUCTO)	Nombres de todos los productos

Ilustración 9 Tabla de categorías y clases del dataset

En cuanto a las imágenes del dataset, el conjunto de imágenes está formado por fotos bastante diversas no solo por las varias prendas de ropa y objetos que representan sino también por las distintas formas en las que se han tomado dichas fotografías. Hay tanto fotos con modelos que visten las prendas de ropa como fotos de estudio únicamente de los elementos a clasificar. Por lo general, para las prendas de arriba se pueden encontrar ambos tipos de fotos, predominando las fotos con modelos. Lo mismo ocurre para los elementos que representan prendas de abajo. Tanto los accesorios, cinturones, joyería u otro tipo de accesorios, así como el calzado y cualquier otro tipo diferente de la ropa como maquillaje, bolsas, carteras entre otros, se representan en su gran mayoría con fotos de estudios en las que aparece el elemento únicamente, salvo algunas excepciones. Por otro lado, los vestidos y trajes de cuerpo completo, la ropa interior (exceptuando calcetines) se suelen representar con fotos de estudio de modelos.

La mayoría de estas imágenes presentan tamaños similares, si bien no iguales, pero lo suficientemente equivalentes como para no resultar problemático para la investigación a desarrollar y serán adaptadas para que todas coincidan con el tamaño requerido para el procesamiento de la red. A continuación, se presentan algunos ejemplos de estas imágenes con su equivalente clasificación obtenida de la fila específica del documento excel que referencia a las imágenes.



ID	3019
Gender	Men
Master Category	Apparel
SubCategory	Topwear
ArticleType	Tshirts
BaseColour	White
Season	Summer
Year	2011
Usage	Casual

Ilustración 10 Ejemplo de parte de arriba. Elemento 3019 del dataset. (derecha: imagen, izquierda: características)

ID	38843
Gender	Women
Master Category	Apparel
SubCategory	Dress
ArticleType	Dresses
BaseColour	Blue
Season	Summer
Year	2012
Usage	Casual

Ilustración 11: Ejemplo de vestido. Elemento 38843 del dataset. (derecha: imagen, izquierda: características)

ID	38993
Gender	Boys
Master Category	Apparel
SubCategory	Bottomwear
ArticleType	Jeans
BaseColour	Blue
Season	Summer
Year	2012
Usage	Casual

Ilustración 12 Ejemplo de parte inferior. Elemento 38993 del dataset. (derecha: imagen, izquierda: características)

ID	43933
Gender	Women
Master Category	Footwear
SubCategory	Shoes
ArticleType	Hells
BaseColour	Black
Season	Winter
Year	2012
Usage	Casual

Ilustración 13 Ejemplo de parte inferior. Elemento 38993 del dataset. (derecha: imagen, izquierda: características)

ID	21268
Gender	Unisex
Master Category	Accesories
SubCategory	Bags
ArticleType	Backpacks
BaseColour	Black
Season	Fall
Year	2011
Usage	Casual

Ilustración 14 Ejemplo de accesorio. Elemento 21268 del dataset. (derecha: imagen, izquierda: características)

Los niveles que se buscan ir completando a medida que avanza el trabajo se basarán en entrenar cada vez buscando una clasificación más profunda (empezaremos por la categoría maestra y una vez obtengamos resultados aceptables pasaremos a entrenar con la subcategoría, luego con tipo de artículo, etc.). Se podrá ver en las siguientes figuras la disposición en clases de los productos para las categorías más importantes:

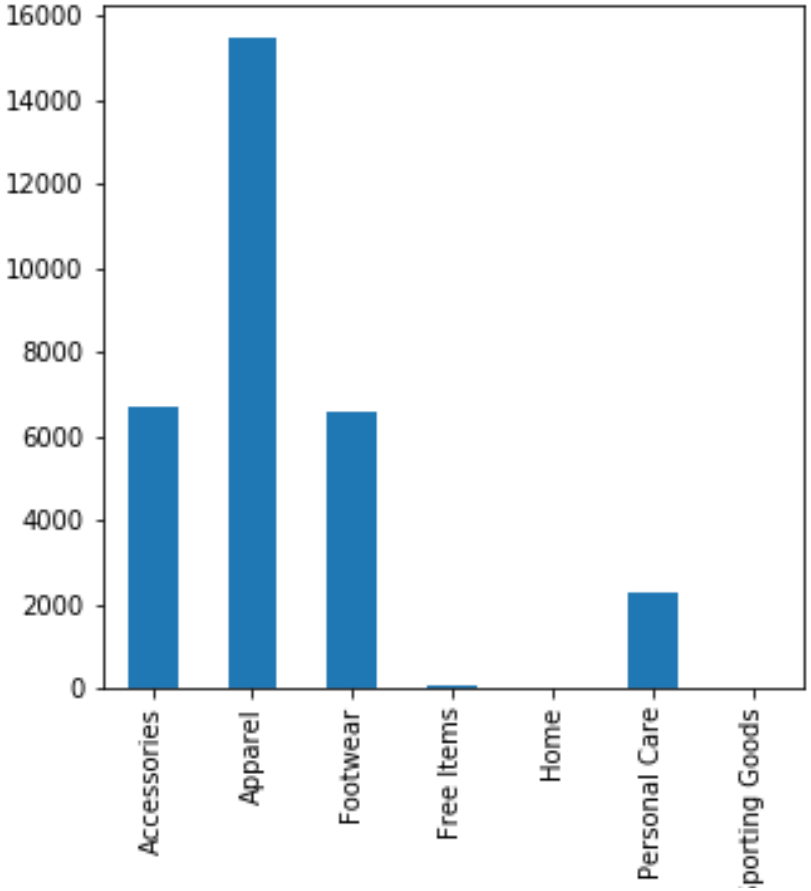


Ilustración 16: Representación gráfica del número de elementos clasificados según MasterCategory

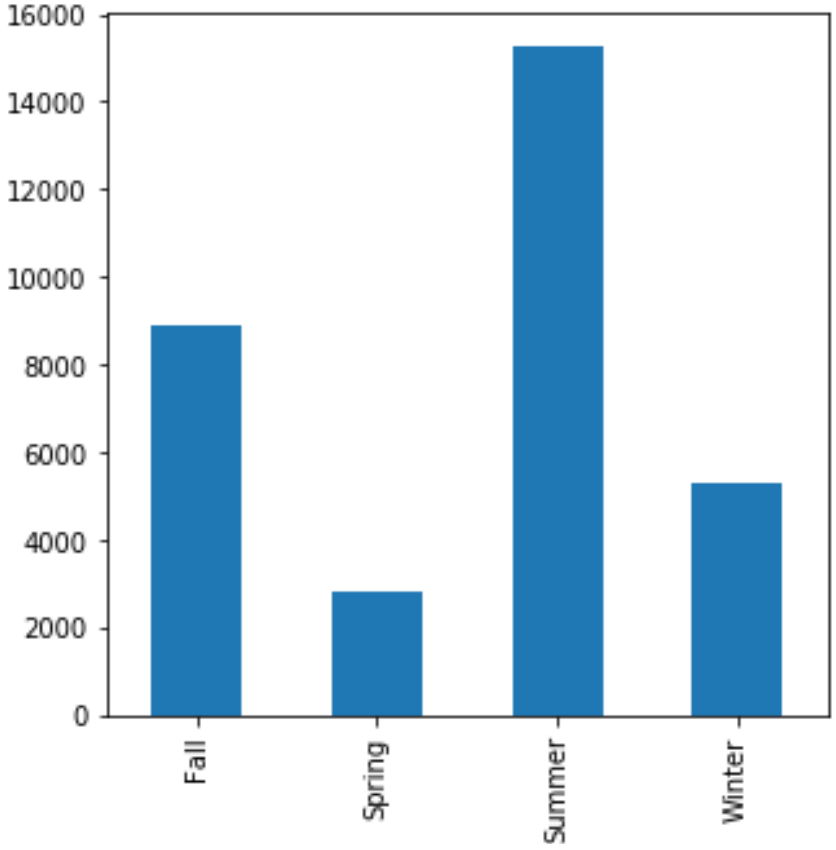


Ilustración 15 Representación gráfica del número de elementos clasificados según Season

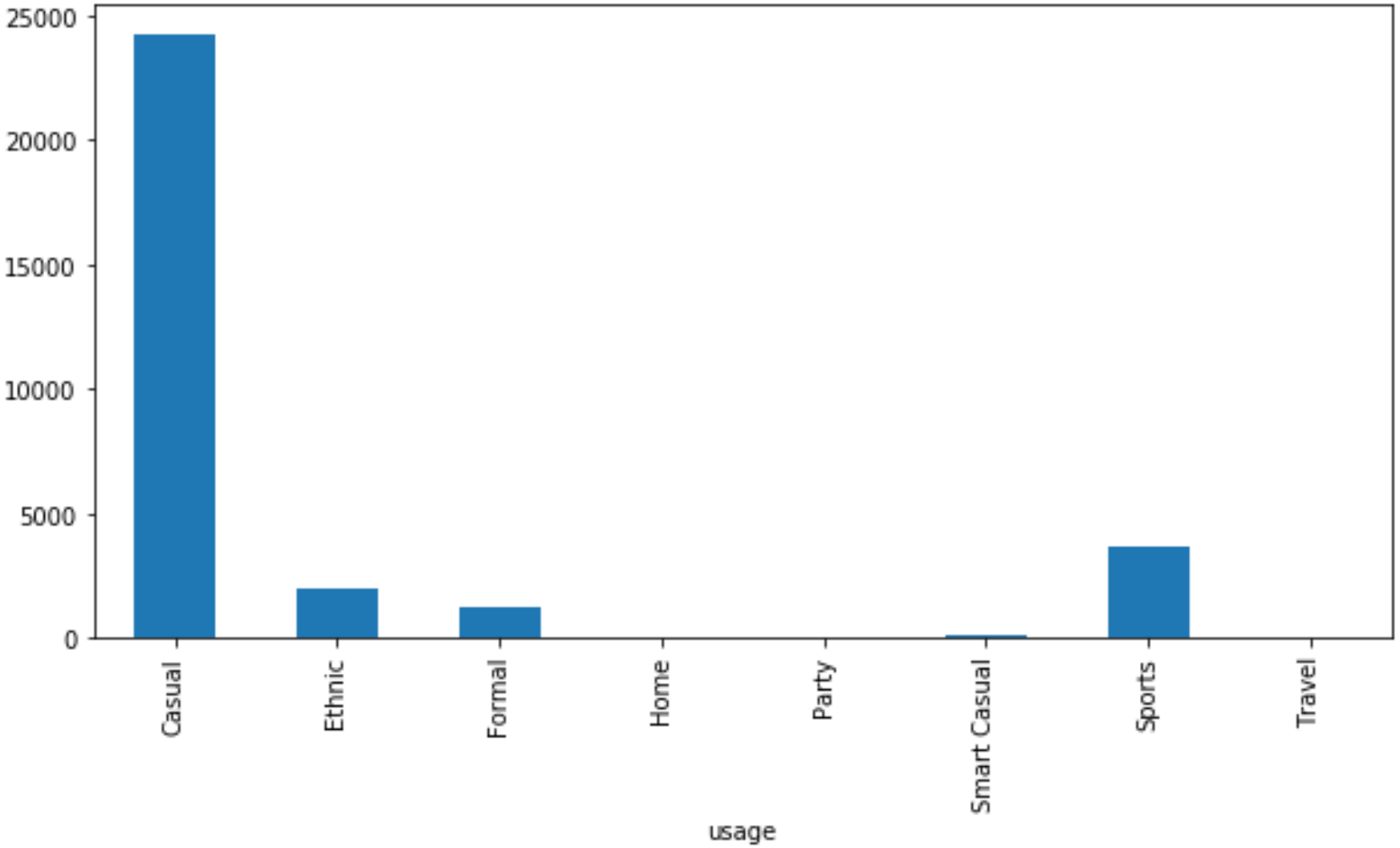


Ilustración 17 Representación gráfica del número de elementos clasificados según Usage

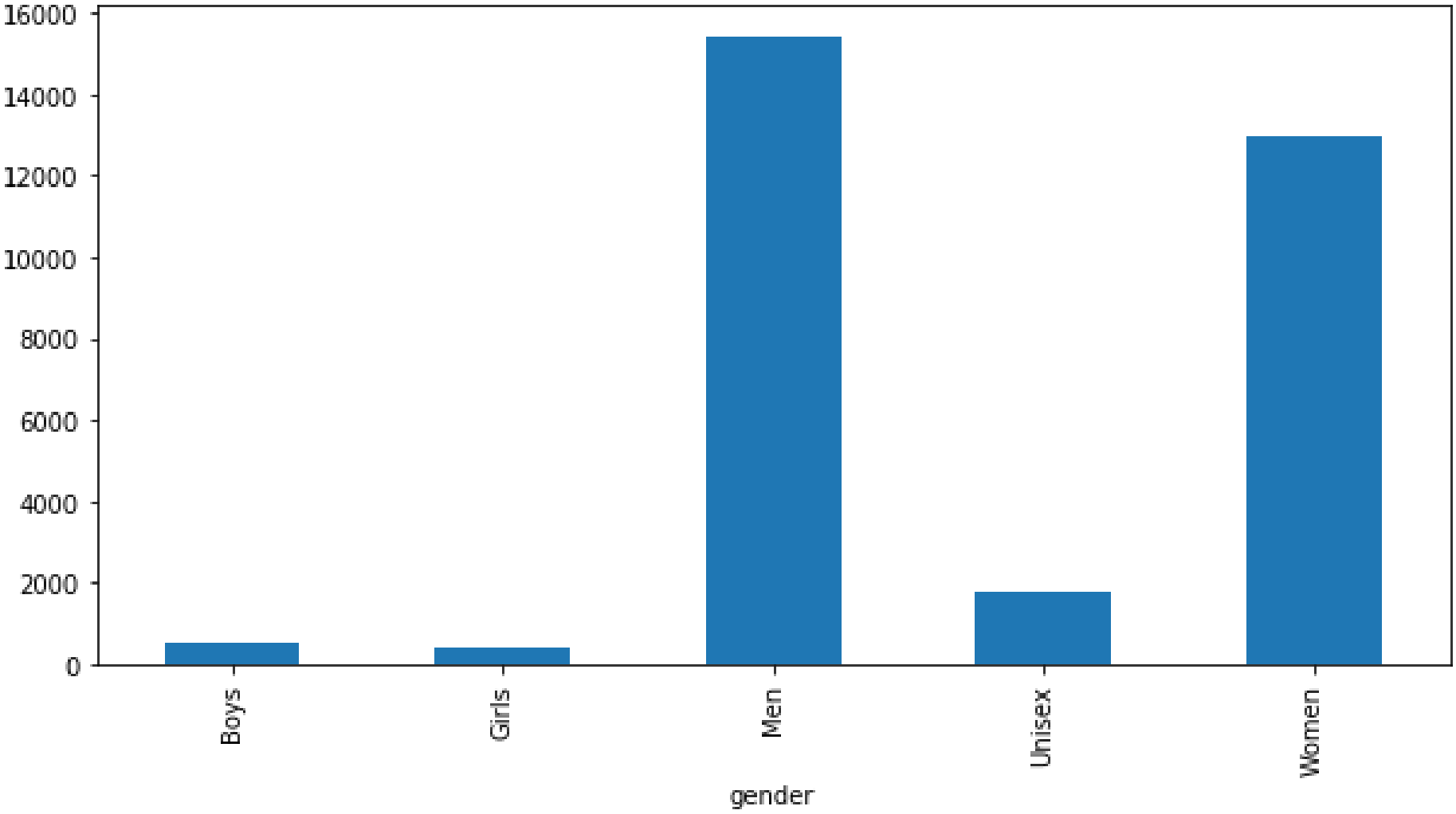


Ilustración 18 Representación gráfica del número de elementos clasificados según Gender

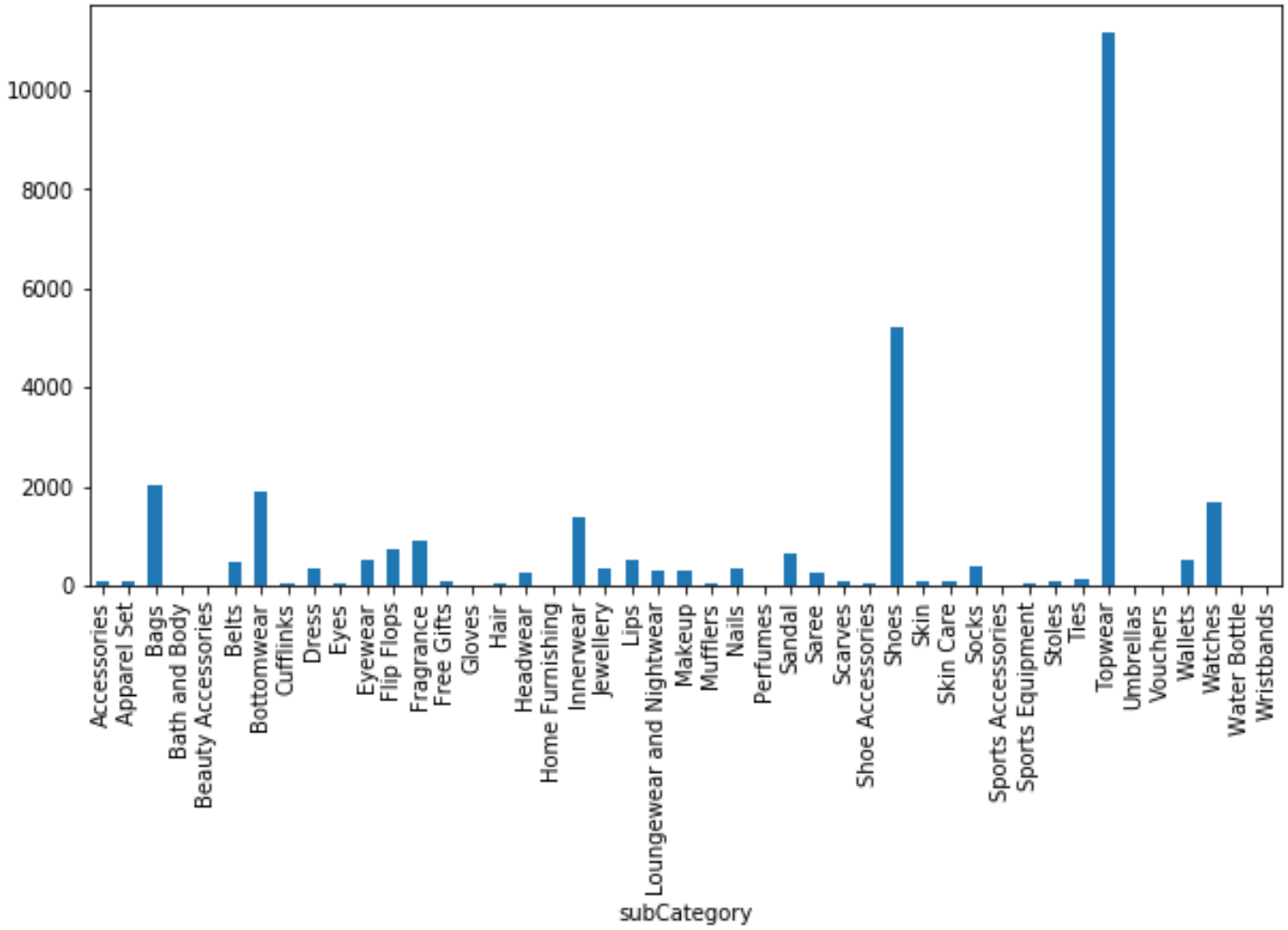


Ilustración 19 Representación gráfica del número de elementos clasificados según SubCategory

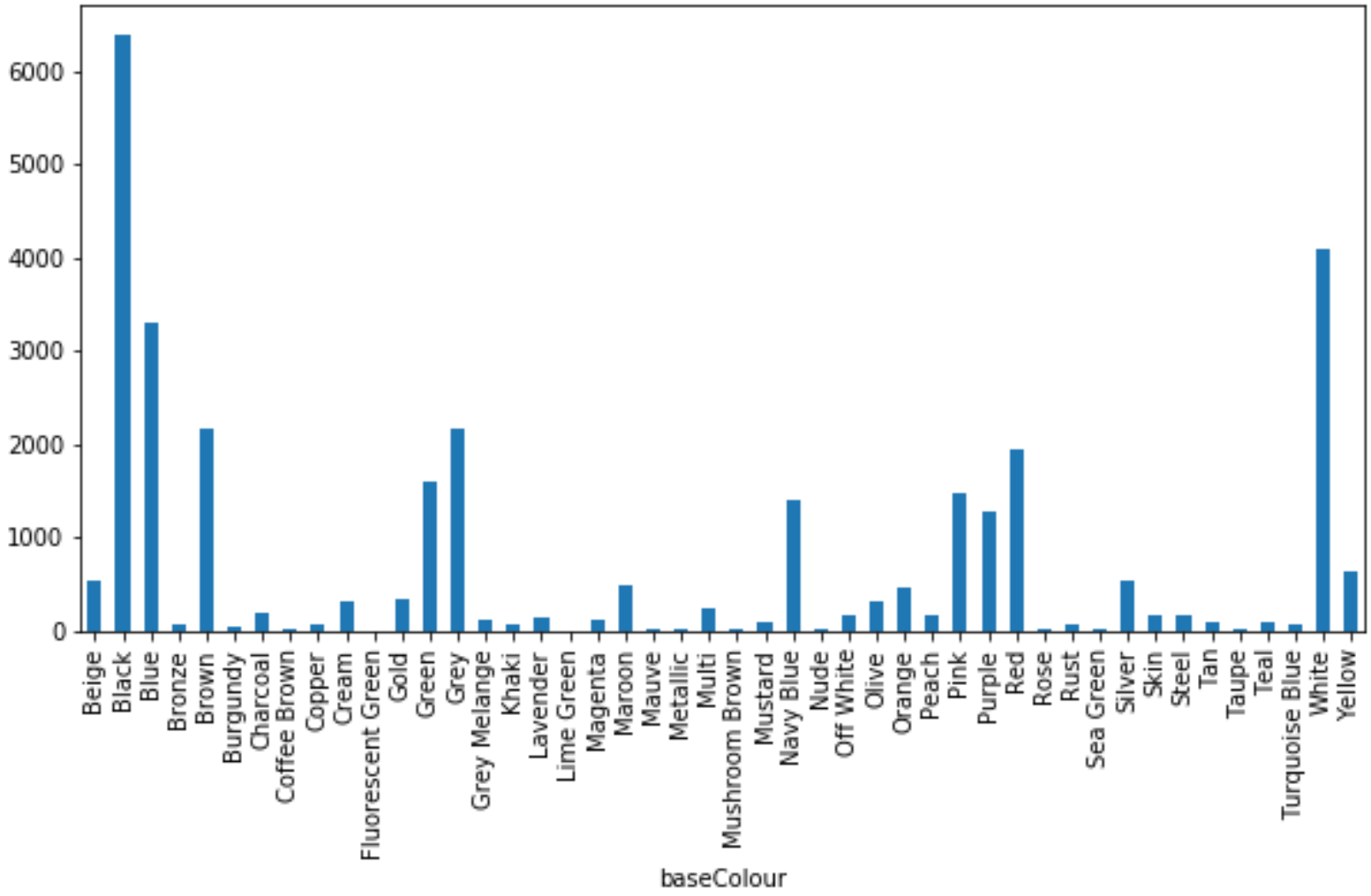


Ilustración 20 Representación gráfica del número de elementos clasificados según BaseColour

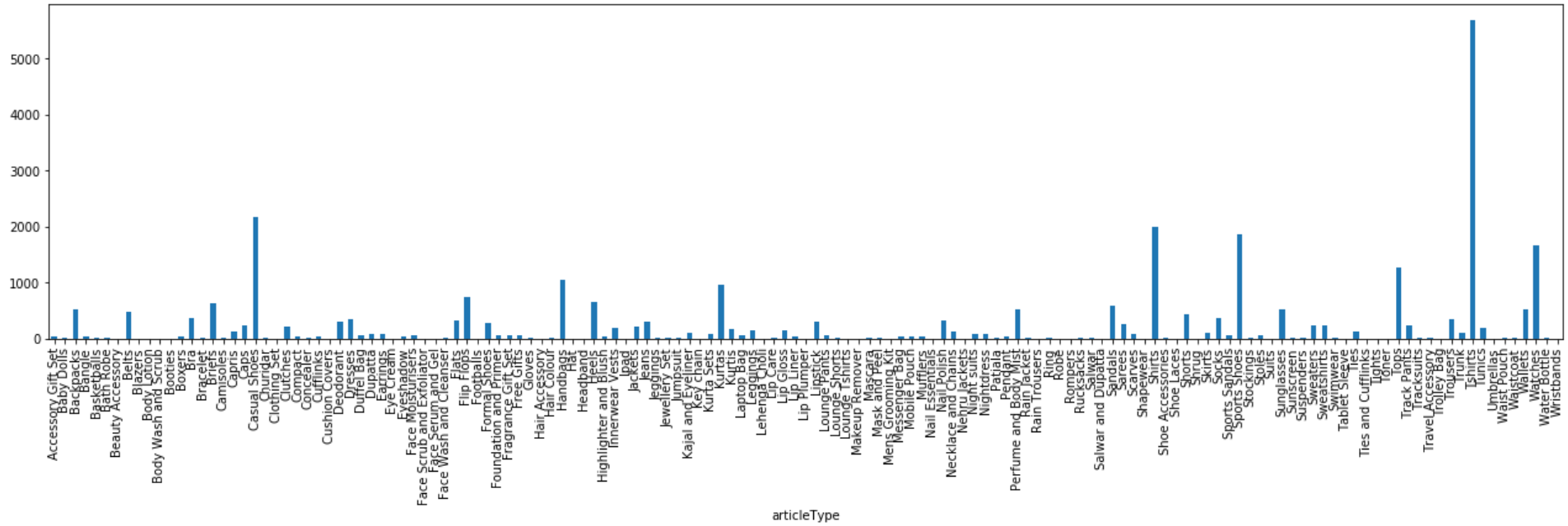


Ilustración 21 Representación gráfica del número de elementos clasificados según ArticleType

Destacamos que, si bien la mayoría de los elementos del dataset son conocidos y vendidos de forma internacional, al estar basados en productos encontrados en las tiendas del país del cual procede el autor del dataset, India, algunos de los elementos resultarán más bien desconocidos o inusuales para el lector, como lo pueden ser los tipos de artículo «salwar» and «dupatta» y «patiala» entre otros.

3.3 Lenguaje, framworks y librerías

Python

Lenguaje de programación que será usado en todo nuestro código. Se trata de un lenguaje interpretado de alto nivel orientado a objetos, además de ser de tipo dinámico y multiplataforma que es uno de los pilares fundamentales de la computación moderna.

El motivo de su elección es que destacan favorablemente la gran cantidad de librerías y herramientas adicionales relacionadas con la computación, aprendizaje de datos y Machine Learning, como Numpy, Pandas, Pytorch, entre otras que hacen que el lenguaje sea de uso fundamental para todo aquel que quiera desarrollar inteligencia artificial.

Pytorch

Pytorch es la herramienta en la que se basará la gran mayoría del desarrollo del código. Se trata de una librería de código abierto dedicada a la computación científica, basada en el lenguaje Python y principalmente dedicada a este lenguaje y cuyos principales objetivos son:

- Servir como un reemplazamiento de la librería de uso similar Numpy en el uso de GPUs
- Servir como plataforma para «Deep learning» que proporcione la mayor velocidad y flexibilidad.

Empezó como un proyecto del departamento de inteligencia artificial de la compañía Facebook cuyo primer lanzamiento fue a finales de 2016, aunque no adquirió relevancia hasta el año siguiente.

La librería presenta múltiples facilidades para la creación, modificación y entrenamiento de redes neuronales, como la capacidad de crear clases específicas para nuestro dataset, facilidad para la transformación de las imágenes, funciones para crear modelos de redes neuronales, así como importar modelos pre-entrenados de arquitecturas conocidas, etc.

La «unidad básica» de la librería es un elemento que representa el tensor, y gracias a su uso por parte de la librería es capaz de integrarse con mayor facilidad con CUDA para mejorar las ejecuciones respecto a las redes neuronales.

PyTorch fue elegida como herramienta base para el proyecto al tratarse de una librería estandarizada que está ganando cada vez más fuerza en el medio por su facilidad de entendimiento y uso y que no para de mejorar, así como que al haber trabajado

anteriormente con Keras veía interesante descubrir una nueva herramienta que ampliara mis conocimientos de cara al futuro.

CUDA

CUDA (acrónimo de «Compute Unified Device Architecture») es una plataforma de computación y modelo de programación proporcionado por la compañía NVIDIA que facilita el uso de GPUs para propósito general. Esto permite que además del uso de tarjetas gráficas para videojuegos o gráficos en general pueda ser usado para el entrenamiento de redes neuronales dada su enorme capacidad para procesar enormes bloques de datos en paralelo.

Livelossplot

Livelossplot es una librería que permite la visualización de gráficas durante el entrenamiento de redes neuronales en documentos Jupyter Notebook que usen frameworks como lo son Keras y Pytorch. Además de la visualización, mantiene los datos referentes a los valores mínimos y máximos obtenidos durante las ejecuciones.

MatPlotLib

Librería fundamental de Python que permite la creación de gráficos de datos.

os

Librería para el manejo del sistema operativo desde Python. Usado para la apertura y/o comprobación de los elementos del dataset (imágenes y Excel)

Pandas

Librería fundamental de Python para el análisis y manipulación de datos. Su uso ha consistido en la lectura de los datos encontrados en los Excel, así como para analizar el dataset que hemos seleccionado.

Seaborn

Librería adicional de Pyhon dedicada a la visualización de datos de forma atractiva e informativa. Usada para la mejor visualización de los «heatmaps» de las matrices de confusión.

Scikit-Learn.metrics

Scikit-learn se trata de una librería de código abierto dedicada enteramente al machine learning, dando soporte para aprendizaje supervisado y no supervisado, ofreciendo además varias herramientas para la selección de modelos, preprocesamiento de datos y evaluación de resultados entre otros campos. En el proyecto fue usada por su sección «metrics», que nos permitía sacar datos conclusivos acerca de los resultados de los modelos creados.

Scikit-Optimize

Scikit Optimize es una librería desarrollada para Python, basada en otras librerías como NunmPy, SciPy y Scikit-learn, que está dedicada a minimizar costosas funciones de tipo caja negra (véase de los cuales únicamente tenemos control de sus datos de entrada y salida, no de los internos). La librería implementa múltiples métodos de optimización basados en modelos secuenciales con distintas opciones disponibles, así como la posibilidad de reanudar búsquedas interrumpidas por medio de los puntos de guardado.

Esta librería será utilizada para la optimización de los hiperparámetros del modelo teniendo como objetivo la búsqueda de la mayor precisión para el modelo. La librería ha sido elegida por recomendación de compañeros que ya habían trabajado con ella.

time

Librería que permite el uso de un número de funciones destinadas al control del tiempo. Usada para la obtención de los tiempos de ejecución

3.4 Servicio en la nube

Dado que el equipo con el que se comenzó el proyecto no tenía potencia suficiente para ejecutar de forma estable las distintas partes de código que se fueron desarrollando a lo largo del tiempo era necesario buscar una alternativa para poder entrenar los modelos de redes que generáramos.

En un primer lugar se hizo uso de **Colab**. Colab es un servicio de Google con el cual poder crear notebooks en la nube, preparados para su programación en Python, así como incluyendo populares librerías relacionadas con Deep learning como Keras, TensorFlow, OpenCV y PyTorch, nuestra principal herramienta de trabajo. Además, dispone al público de una GPU sin costes adicionales para el usuario, concretamente una GPU Tesla K800. Si bien durante las primeras pruebas se hizo uso de este servicio, debido a problemas con él a la hora de importar el dataset y un costo temporal añadido por la inconsistencia de la máquina usada (necesidad de cargar los archivos para cada sesión). Además, podíamos encontrar mejores alternativas de ejecución (tanto respecto al entorno de ejecución como a la GPU a pesar de su posible coste adicional).

Paperspace es al igual que Colab una plataforma en la nube, con la diferencia de que gran parte de sus posibilidades son de pago. A pesar de eso, Paperspace contiene una gran cantidad de opciones para el desarrollador de Inteligencia artificial. Presenta 3 productos distintos:

- Core, destinado a generar entornos y servidores de Windows o Linux con mejores características que las que podría tener uno propio
- Api, que permite crear servicios en la nube.
- Gradient, el producto que realmente nos interesa. Se trata de una plataforma dedicada enteramente a Machine Learning e Inteligencia Artificial. Permite la creación de notebooks que ejecutar con varias opciones de GPU manteniendo el kernel activo durante largas etapas de trabajo, trabajos automatizados, guardado de modelos, así como proporciona una máquina virtual con la que interactuar y en la que se mantendrán los datasets que descarguemos en esta (así como esta misma plataforma proporciona datasets populares para su uso) entre otras opciones.
  Entre las opciones que permite para el kernel del notebook que tenemos creado (permitiendo selección de tiempo de ejecución del kernel y la GPU a seleccionar), optamos por usar una GPU Quadro P5000 para mejorar el tiempo de entrenamiento de los modelos que generamos.

Esta plataforma sería la utilizada durante el resto del proyecto.

Capítulo 4

Desarrollo

Durante la investigación se estuvo ejecutando una enorme cantidad de veces la clasificación para cada categoría, ya que el poco contacto con redes neuronales convolucionales, la introducción apresurada a pytorch y la necesidad del manejo de un servicio en la nube, provocaron una gran cantidad de fallos que se fueron solucionando. En esta sección, sin embargo, solo trataremos los mejores resultados obtenidos y las técnicas usadas para obtenerlos.

4.1 Procesamiento de datos

Una vez seleccionado el dataset, se hizo un análisis de los datos que lo componían (y cuyos números pueden ser vistos en la sección Dataset en el capítulo 3: herramientas), ya que podía ser necesario hacer cambios o pequeñas modificaciones para adaptarlo a nuestro problema a resolver. Utilizaremos todas las clases del dataset, ya que todas tienen importancia en el mundo de la moda.

Observando el conjunto de imágenes se puede ver una categoría que se repite de forma similar al hacer la clasificación de MasterCategory, SubCategory y ArticleType, y es la existencia de la categoría «Free gifts» o «Free ítems». Esta categoría se diferencia de las otras en que al contrario que el resto de ellas en las cuales una prenda como un zapato se diferencia claramente de una prendas de arriba, así como de una prenda inferior, esta clase en específico está formada por elementos que podrían caer perfectamente en cualquiera de las otras clasificaciones, véase que se han encontrado casos de relojes que deberían ser clasificados como accesorios o camisas que deberían ser clasificadas como prendas de arriba que acababan como «Free gifts». En contexto suponemos que se tratan de elementos incluidos en ofertas o promociones en las cuales por un motivo u otro acababan dándose gratuitamente a los clientes. Si bien una red neuronal podría llegar a obtener alguna característica común entre sus componentes, es una clasificación que entorpece a aquellas con las que “comparte” elementos, así que nos desharemos de los elementos bajo esta clasificación.

En varias de las categorías del dataset existían unas categorías en las cuales había elementos que quedaban con valores no válidos que se imprimían como NaN. Dado que estos casos no son de utilidad para nuestra clasificación y además se trata de un pequeño número de elementos de los disponibles, nos desharemos de ellos en las categorías a las cuales afecten con valores inválidos. También se elíminó de la categoría «Master Category» la clase «home», al tener solo un elemento y muy poca relación con el resto de clases.

Cabe destacar que esto ocasionó que categorías con un número muy pequeño de elementos (sobre todo en clasificaciones como ArticleType) acabaran desapareciendo casi al completo y provocando precisión nula en los posteriores entrenamientos.

4.1.3 Re-escalado de las imágenes

Si bien las imágenes son de tamaños similares, no todas tienen el mismo tamaño, aunque a la vista pueda parecerlo. Al mismo tiempo, la red en la que se basa la gran mayoría de este proyecto, la estructura VGG, está preparada para aceptar un tamaño en concreto de la imagen: 224 píxeles x 224 píxeles x 3 (capas de color RGB). Por ello, era necesario re-escalar todas las imágenes a un tamaño de 224x224 si su entrada iba a ser directa, o como alternativa a la entrada normal de datos buscando hacer data augmentation (concepto que se explicará a continuación) un re-escalado a otras dimensiones que acabe con un recortado o «crop» de la imagen a una versión de 224x224, aunque tratándose de una versión bastante más diferente a la original que el re-escalado normal.

4.1.5 Data augmentation

Para variar más los elementos que se clasificaban se han realizado unos añadidos a las imágenes cada vez que son cargados:

- Flip horizontal aleatorio: a algunas de las imágenes se les daría la vuelta de forma de que lo que se mostraba de izquierda a derecha se mostrara al contrario de derecha a izquierda.
- Rotación aleatoria: a algunas de las imágenes se les rotaria hasta 20 grados.
- Recortado aleatorio: la imagen sería primero cambiada a un tamaño de 236x236, y teniendo este tamaño, se haría un corte aleatorio en la imagen para pasarla a los 224x224 que requería la red VGG, siendo este recorte hecho de forma aleatoria sobre cada imagen. Este recorte se habría probado anteriormente empezando por un tamaño de 256x256, pero se descartó al ver que era excesivo para algunos elementos del conjunto.

Estas distintas transformaciones logran que, si bien tenemos un número limitado de imágenes, podemos ampliar la cantidad de ejemplos metiendo también variaciones de las mismas, en distintas posiciones, ya que la red las verá como una foto distinta y de la cual debe inferir la misma lógica. Al estar tratando con un tamaño de imágenes muy dispar, esto significa que aquellas con un menor número de ejemplos dispondrán ahora de un poco más de elementos de los cuales obtener características (no olvidemos que con el resto de las categorías tengan más o menos elementos pasará igual). De esta forma, cambiando las imágenes estamos dando más variedad sobre la que entrenar, tratando por lo tanto con el concepto denominado como DataAugmentation, con la esperanza de que pueda mejorar los resultados a obtener.

4.2 Entrenamiento

Para el entrenamiento, de forma general se ejecutarán por primera vez todos con los mismos parámetros.

En primer lugar, para el aprendizaje de la red se utilizará el **descenso por el gradiente estocástico** *[2]*. Esta técnica es similar a la del descenso por el gradiente común, con la diferencia de que hace simplemente una estimación al obtener el gradiente de cada batch y no del conjunto al completo. Esto permite además de la velocidad de entrenamiento asociada a usar mini-lotes, que la aleatoriedad de cada lote dificulte el estancamiento del proceso de aprendizaje. El learning rate usado por defecto sería el siguiente:

0.003920274771701523

El valor fue elegido tras ser obtenido durante las primeras pruebas de optimización de hiperparámetros sobre las redes.

Hablando de la optimización, su ejecución se realizaría teniendo en cuenta los siguientes posibles rangos y valores para los hiperparámetros principales de la ejecución:

- Batch size: 16, 32, 64
- Learning rate: 0.001 <-> 0.05
- Neuronas de la capa: 512, 1024, 2048, 4096, 8192

La obtención de la mejor exactitud por optimización de hiperparámetros se buscaría utilizando la función de skopt «forest_minimize», que usa un modelo de regresión basado en árbol para minimizar el coste (en este caso el porcentaje que no ha sido clasificado correctamente).

Durante el entrenamiento de la red se tuvieron en cuenta otras técnicas para mejorar los resultados, que se mención a continuación.

4.2.1 Batch normalization

Tras el procesamiento por cada capa convolucional, antes de procesar los resultados usando la función ReLU se usaría antes una capa de **batch normalization** *[7]* o normalización por lotes. La técnica de normalización de lotes consiste en coger la salida de una capa (en nuestro caso, de las capas convolucionales) y estandarizar de forma similar a como hacemos a la entrada de la red los valores. Esto es recomendable dado que la distribución entre el procesamiento de un elemento y el de otro puede dar a oscilaciones muy bruscas del error y por lo tanto de la actualización de los pesos, dificultando el trabajo de obtención de características. Esta estandarización se hará teniendo en cuenta que usamos el descenso por el gradiente estocástico, y, por lo tanto, la media y la desviación aplicadas serán obtenidas a partir de cada batch.

Además, el proceso permite normalmente llegar a buenos resultados con un menor número de épocas involucradas en el entrenamiento. Además, si se le aplica un momentum que trate de que la diferencia de desviación y media no sea tan grande en comparación a la anterior iteración, podremos conseguir un efecto de regularización

(disminución de overfitting). En general, la técnica permite una simplificación de la creación de modelos neuronales.

4.2.2 Dropout

Tras cada capa oculta de la sección de clasificación y tras el procesamiento por la función ReLU que las acompaña, insertamos una función de dropout que hará que descarte el resultado de las neuronas individuales que caigan en su probabilidad, con el objetivo de reducir el posible «overfitting» o sobreajuste que pueda llegar a sufrir la red. El parámetro que regirá la probabilidad de que se descarte o no la neurona será de 0.5 por defecto en cada sitio que se use.

4.2.2 Transfered learning

Si bien la arquitectura VGG es potente y entrenándola adecuadamente podríamos obtener unos resultados de obtención de características bastante mejores, el tiempo que es necesario para esto, sobre todo teniendo en cuenta el número de clasificaciones que vamos a realizar en nuestro cometido, es extremadamente grande e incluso podríamos no llegar al resultado deseado. Además, es posible que, con este entrenamiento, lográramos un modelo de red cuyos pesos estuvieran distribuidos de forma muy similar a los de otra red para otra clasificación pero que igualmente buscan tomar características a partir de imágenes. Por ello es que en este proyecto haremos uso de la técnica de «Transfered learning» que consiste en tomar la configuración de pesos de otra arquitectura previamente entrenada (otra VGG en nuestro caso) que presente una buena exactitud en su cometido. De esta forma ya no será necesario tanto esfuerzo para conseguir unos resultados aceptables, e incluso podríamos solo modificar una parte de la red para ello, como se explica a continuación.

4.2.4 Entrenamiento de la clasificación y fine tunning

Como mencionamos en el punto anterior, con transfered learning podemos obtener unos pesos pre-entrenados con los cuales nos sea más fácil llegar a nuestro objetivo. La parte importante, la convolucional, encargada de extraer características probablemente ya haya sido lo suficientemente trabajada como para poder obtener información de otras imágenes no pertenecientes a su conjunto original sin problemas, sin embargo, no pasa lo mismo para la parte clasificadora formada por capas fully-connected. Es una técnica común en el campo de la inteligencia artificial por ello dejar los pesos de la capa convolucional como están, únicamente entrenando la capa clasificadora, que es al fin y al cabo, la que debe a partir de las características obtenidas en qué clase vamos a meter el elemento procesado.

De todas formas, hay veces en las que sigue pudiendo ser necesario entrenar al menos una parte de los pesos de la parte convolucional de la red. Es aquí donde nace el concepto «fine-tunning» que vamos a usar en el proyecto. Este consiste en además de entrenar la clasificación, permitir el entrenamiento de un número de capas de las más cercanas al final de la parte convolucional, que serán aquellas que más influencia puedan tener en el procesamiento posterior de la parte clasificadora.

En nuestro proyecto se harán pruebas iniciales con 4 configuraciones distintas de entrenamiento:

- Entrenamiento solo de la parte clasificadora.
- Fine-tunning de 1 capa (clasificación + última capa convolucional).
- Fine-tunning de 2 capas (clasificación + dos últimas capas convolucionales).
- Fine-tunning de 3 capas (clasificación + tres últimas capas convolucionales).

Este uso de Fine-tunning es considerado mejor que reentrenar toda la red por el limitado número de elementos del cual dispone el dataset para algunas clases.

4.3 Obtención de resultados

Código general desarrollado.

El proyecto ha contado con varias versiones del código, todas basadas en múltiples ejemplos que ofrecía la organización de pytorch, que se han ido modificando ya fuera para adaptarse a los modelos a entrenar o los objetivos a lograr, o simplemente para que al ejecutarse disminuyera la necesidad de ser controlado por parte del usuario. El código aún siguió siendo modificado durante las últimas etapas del trabajo para cada una de las ejecuciones, normalmente para decidir los datos a coger o los parámetros a elegir. Finalmente, el código usado se estructura como se va a describir a continuación.

Una primera parte destinada a la importación de las librerías descritas anteriormente que se usarían a lo largo del código, así como la inicialización de variables que se tratan de constantes o una variable global llamada mode que permite cambiar la forma en la que vamos a entrenar luego. También es la parte encargada de definir la transformación que sufrirán las imágenes antes de entrar al código.

Seguido viene una clase heredera de la clase dataset de pytorch a la cual instanciaremos para la creación de nuestro dataset. Es el encargado de limpiar los datos y de suministrarlos cuando se le piden mediante get_item(). Además, se le ha aportado una funcionalidad con la cual a partir de la parte convolucional de una red VGG crea y usa unos datos pre-procesados por la parte convolucional, de forma que si queremos entrenar únicamente una sección como lo es la parte clasificatoria de la red, se haga uso solo de la parte clasificatoria con los datos ya preparados que suministrará el objeto dataset. Esto permite una mucho mayor rapidez a la hora de hacer operaciones más exhaustivas y sin tanto peso en la convolución como al optimizar hiperparámetros.

Los datasets se combinan con la posterior creación de dataloaders, unos elementos que son los encargados de controlar la entrada de elementos a la red por batches incluso haciendo algún trabajo más, como aleatorizar su aparición.

Siguiendo con el código, creamos una función solo para facilitar la configuración de los optimizadores que harán uso del learning rate para hacer que la red aprenda. Y hablando de la red continuamos con un grupo de clases que se han definido que representan cada una de las posibles formas en las que le pasaremos el modelo de red al módulo encargado del entrenamiento. Véase que dependiendo de si se trata de una ejecución en la que simplemente buscamos entrenar la parte clasificatoria o si queremos hacer fine-tunning devolveremos una configuración ligeramente distinta. Vienen acompañadas de una función para obtener el tipo deseado según el valor en la variable global MODE.

Posteriormente encontramos el bloque de código destinado al entrenamiento (junto a un pequeño bloque para imprimir los resultados obtenidos). El entrenamiento de cada modelo con este trozo de código procederá de la siguiente forma: en cada época el programa procesará los datos tanto del grupo de entrenamiento como del grupo de validación, mostrando al final de cada una de estas una actualización gráfica de la evolución del aprendizaje tanto en ambos conjuntos dando a conocer su porcentaje de precisión y de pérdida también. El código se encargará de guardar por cada grupo de épocas entrenadas el modelo de red que consiga la mayor precisión sobre el conjunto de validación, permitiendo que aun pudiendo haber overfitting no se pierdan los logros conseguidos.

El código continúa con un añadido para comprobar cuál ha sido el desempeño de la red de forma individual. Un grupo de funciones que pasándoles una red previamente entrenada (por nosotros, no por transfered learning) nos permitirán saber para cada una de las clases a predecir el resultado que obtuvo con cada elemento, para poder usarlo para estudios como la matriz de confusión, la sensibilidad, la precisión y la exactitud (que ya nos la devuelve la función de entrenamiento).

Tras esto, se procede a la realización de la inicialización de las distintas clases y el comienzo de los entrenamientos, aunque cabe destacar el código realizado para la optimización de hiperparámetros con la ayuda de skopt. En esta sección probamos distintas configuraciones para mejorar los resultados de los distintos modelos que se hicieran.

Por último, en cuanto a los resultados mostrados a continuación, cabe mencionar que de todos ellos se examinará, además de su exactitud sobre el conjunto de validación, su matriz de confusión y sensibilidad (porcentaje acertado de cada clase), con el objetivo de sacar conclusiones.

Caso nº 1: Categoría maestra.

La categoría MasterCategory, como ya hemos comentado representa de forma muy general el tipo de artículos que se pueden encontrar en una tienda de moda. Debido a la eliminación de «Free Gifts» y la inutilidad de Home (que solo tiene un elemento), se

queda con una distribución de 5 clases, de las cuales tres de ellas son mayoritariamente prendas de vestir.

Categoría	Tipo de entrenamiento	Mejor precisión	Épocas
Master Category	Entrenamiento clasificación	99.70 %	10
	Fine-tunning de una capa	99.59 %	10
	Fine-tunning de dos capas	99.67 %	10
	Fine-tunning de tres capas	99.69 %	10

Ilustración 22 Tabla exactitud de MasterCategory

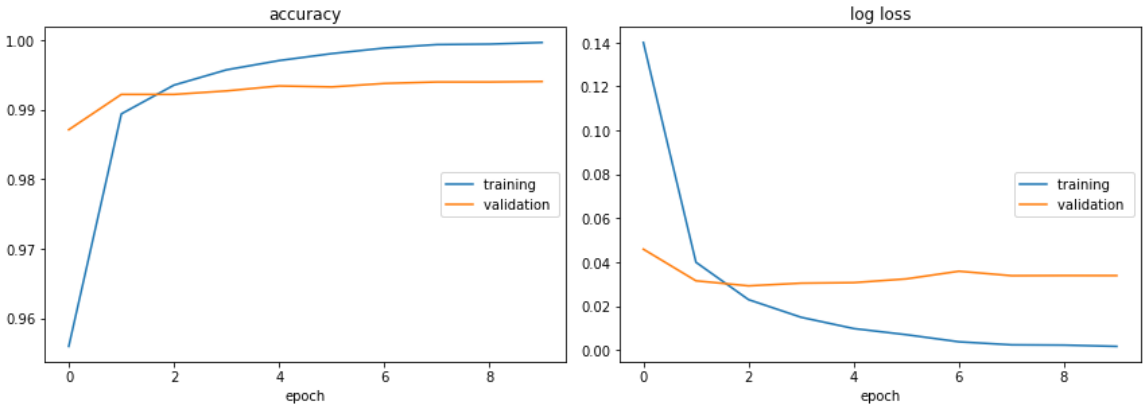


Ilustración 23 Gráfica del entrenamiento de clasificación

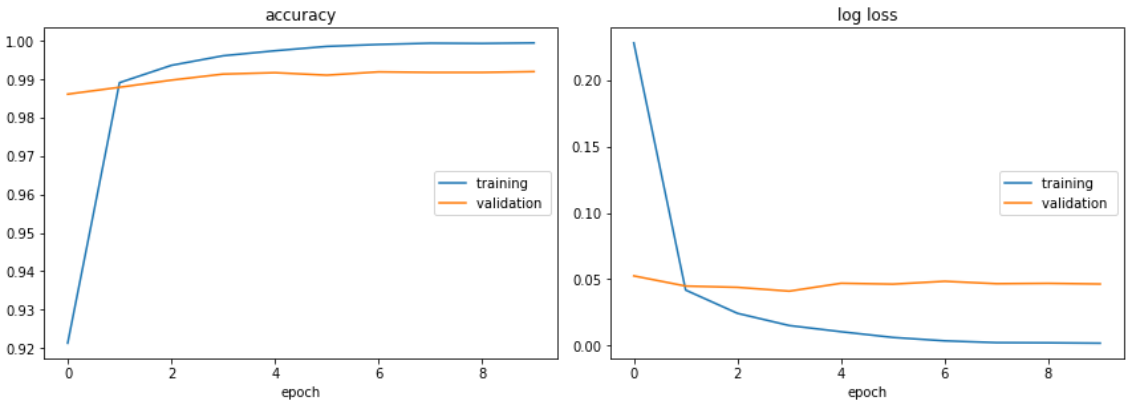


Ilustración 24 Gráfica del entrenamiento fine-tunning 1

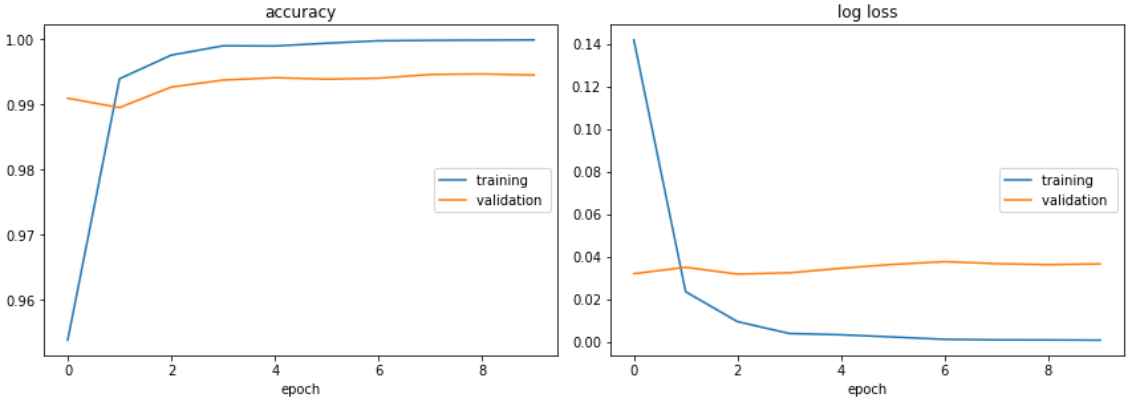


Ilustración 25 Gráfica del entrenamiento fine-tunning 2

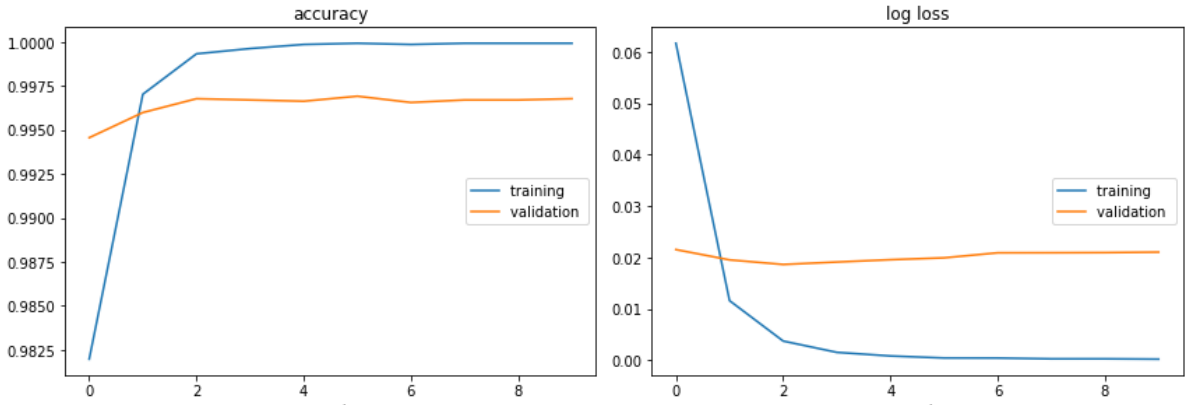


Ilustración 26 Gráfica del entrenamiento fine-tunning 3

Clasificación	Solo clasificación	FT de una capa	FT de dos capas	FT de tres capas	Nº elementos validación
Apparel	99.78 %	99.76 %	99.75 %	99.76 %	6770
Accesories	99.49 %	98.26 %	98.46 %	99.57 %	3541
Footwear	99.76 %	99.77 %	99.73 %	99.70 %	2990
Personal Care	99.85 %	99.55 %	99.85 %	99.85%	230
Sporting Goods	87.50 %	37.50 %	87.50 %	75.00 %	8

Ilustración 27 Tabla se sensibilidad de MasterCategory

Como observamos, los resultados obtenidos son casi perfectos, siendo muy fácil la dife renciación entre las distintas clases al tratarse de elementos que, aun encontrándose todos en un mismo ámbito, son bastante distintos entre sí. Sus mejores valores los con sigue solamente entrenando la clasificación, viendo así que para este caso el fine-tunning no supone ninguna mejora.

Durante la búsqueda de hiperparámetros dedicada a esta categoría se consiguió aumentar la exactitud muy ligeramente a un 99.77% teniendo los siguientes valores:

- Entrada y salida de la capa intermedia: 512 -> 8192
- Learning rate: 0.011975841795078507
- Batch size 64

La matriz de confusión mostrada gráficamente a continuación ayuda a corroborarlo.

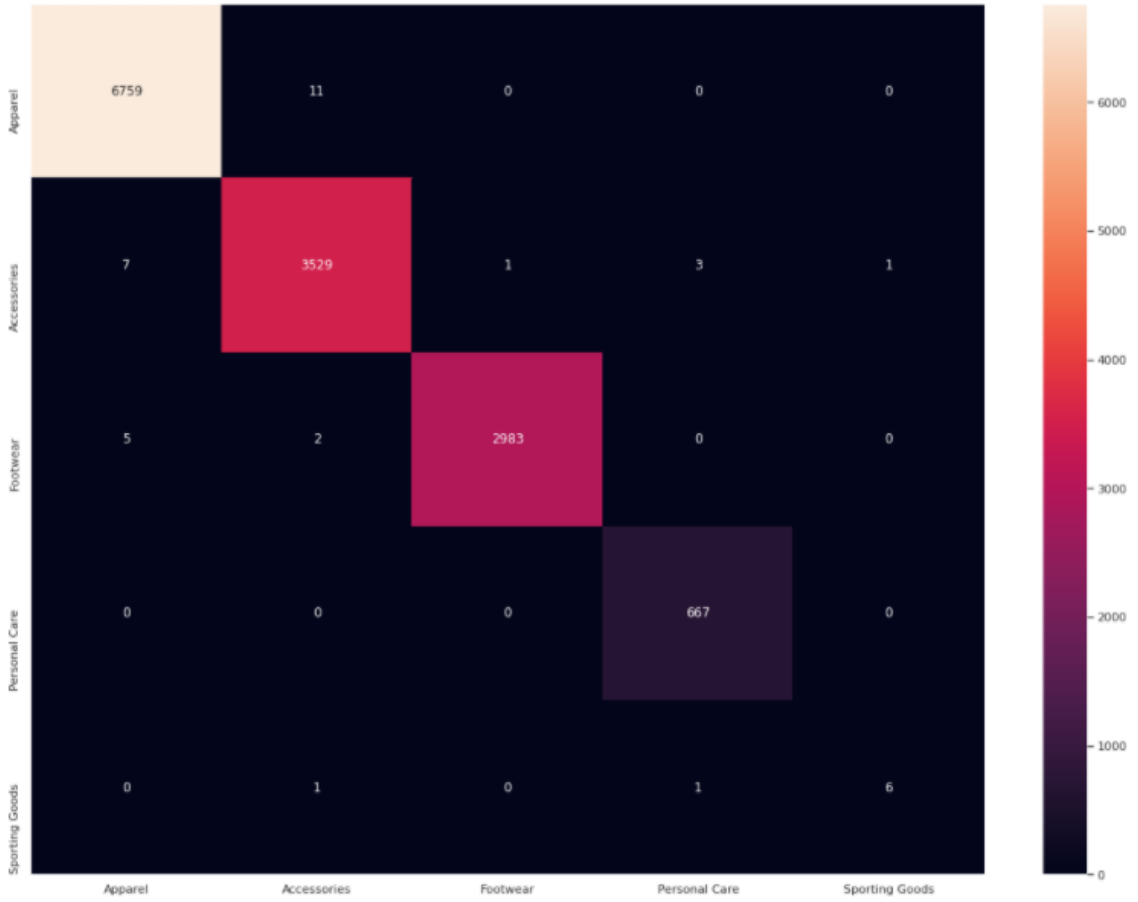


Ilustración 28 Matriz de confusión de MasterCategory

Vemos que existen muy ligeras confusiones entre los distintos elementos, si acaso sien do más predominante una ligera tendencia de «Apparel» a «Accesories» y viceversa. «Sporting goods» aun teniendo pocos elementos presenta mínimo error. En conclusión , para esta pequeña clasificación, los resultados son abrumadores, siempre teniendo en cuenta la simpleza de su tarea.

Caso nº 2: Sub-Categoría.

La categoría SubCategory, como ya hemos comentado representa aún de forma general el tipo de artículo que es, pero profundizando un poco más en ello. Al igual que con la

anterior, se eliminó la categoría equivalente a «Free Items», aquí denominada como «Free gifts».

Categoría	Tipo de entrenamiento	Mejor precisión	Épocas
Sub Category	Entrenamiento clasificación	97.41 %	10
	Fine-tunning de una capa	97.08 %	10
	Fine-tunning de dos capas	97.64 %	10
	Fine-tunning de tres capas	97.59 %	10

Ilustración 29 Tabla de exactitud de SubCategory

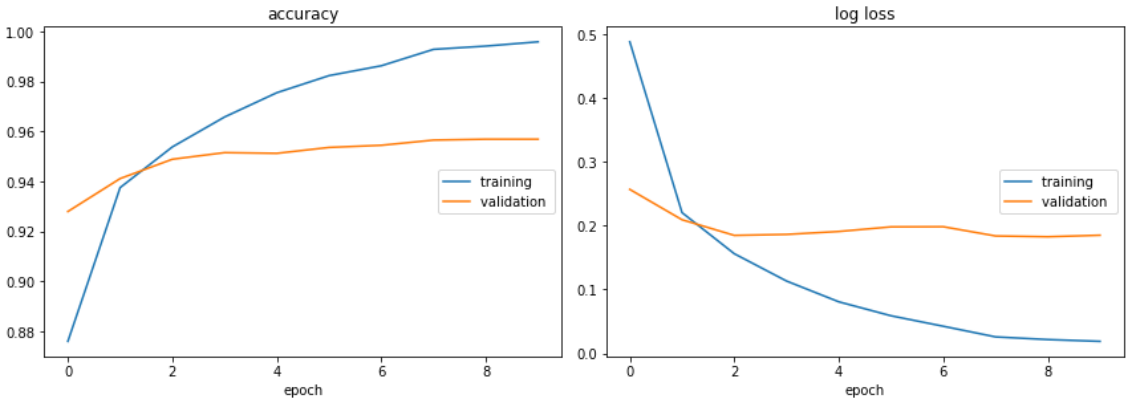


Ilustración 30 Gráfica del entrenamiento de clasificación

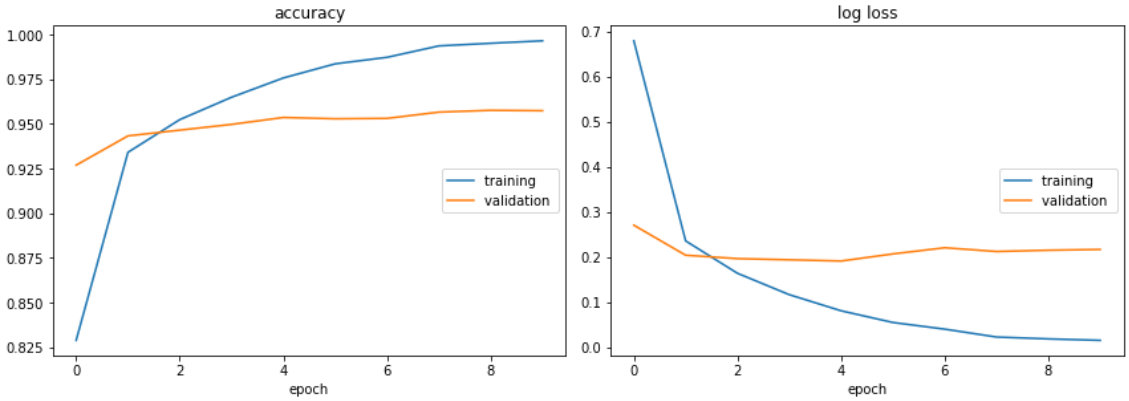


Ilustración 31 Gráfica del entrenamiento fine-tunning 1

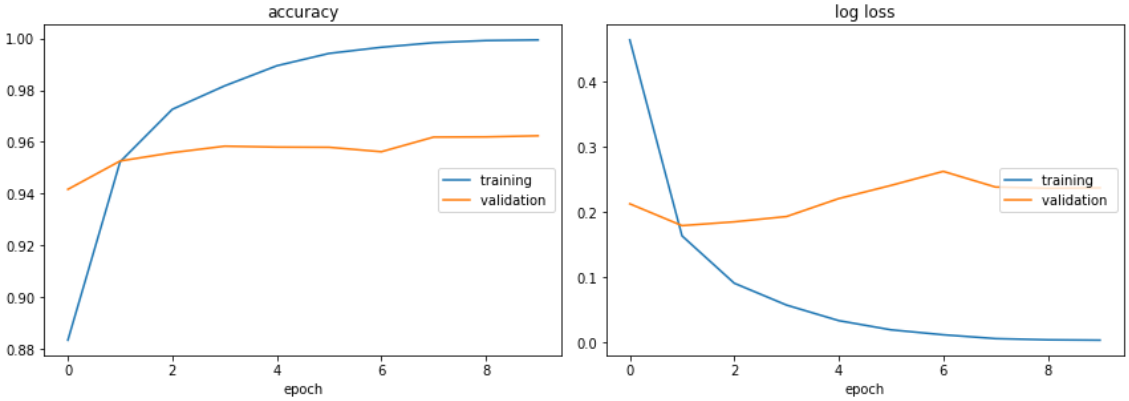


Ilustración 32 Gráfica del entrenamiento fine-tunning 2

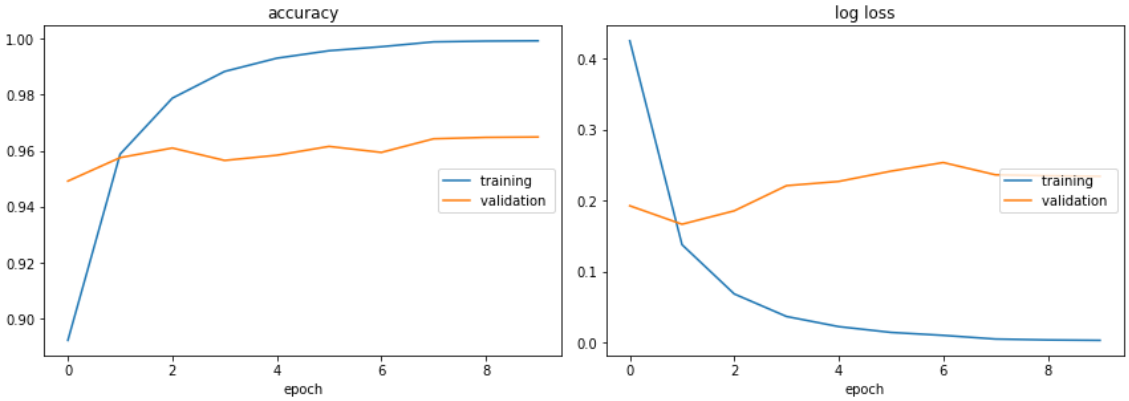


Ilustración 33 Gráfica del entrenamiento fine-tunning 3

Clasificación	Solo clasificación	FT de una capa	FT de dos capas	FT de tres capas	Nº elementos validación
Topwear	99.26 %	98.89 %	99.18 %	99.18%	4867
Bottomwear	98.62 %	98.39 %	98.28 %	98.28 %	872
Watches	99.74 %	99.62 %	99.74 %	99.49 %	787
Socks	98.20 %	97.75 %	98.20 %	98.65 %	222
Shoes	99.04 %	98.86 %	99.07 %	98.91 %	2377
Belts	99.17 %	99.58 %	99.58 %	99.58 %	240
Flip Flops	91.86 %	88.27 %	92.83 %	92.18 %	307
Bags	98.33 %	98.12 %	98.44 %	98.75 %	960
Innerwear	97.59 %	97.77 %	97.77 %	97.94 %	582
Sandal	83.99 %	83.99 %	86.27 %	87.25 %	306
Shoe Accesories	00.00 %	00.00 %	00.00 %	00.00 %	2
Fragrance	98.42 %	98.42 %	99.05 %	98.73 %	316

Jewellery	97.53 %	98.63 %	98.90 %	99.18 %	364
Lips	96.82 %	93.65 %	96.03 %	96.82 %	126
Saree	99.19 %	99.19 %	99.19 %	98.39 %	124
Eyewear	100.0 %	99.70 %	99.70 %	100.0 %	337
Nails	100.0 %	100.0 %	100.0 %	100.0 %	88
Scarves	79.31 %	82.76 %	82.76 %	79.31 %	29
Dress	80.82 %	80.82 %	82.19 %	76.71 %	146
Loungewear and Nightwear	79.58 %	81.69 %	80.98 %	82.39 %	142
Wallets	96.61 %	96.27 %	96.95 %	97.29 %	295
Apparel Set	71.43 %	74.28 %	77.14 %	77.14 %	35
Headwear	100.0 %	98.97 %	98.97 %	100.0 %	97
Mufflers	33.33 %	40.00 %	73.33 %	60.00 %	15
Skin Care	50.00 %	55.55 %	44.44 %	38.89 %	18
Makeup	85.36 %	82.93 %	89.02 %	86.58 %	82
Free Gifts	00.00 %	00.00 %	00.00 %	00.00 %	0
Ties	100.0 %	100.0 %	100.0 %	100.0 %	76
Accessories	81.39 %	79.07 %	81.39 %	81.39 %	43
Skin	64.28 %	50.00 %	64.28 %	78.57 %	14
Beauty Accessories	00.00 %	00.00 %	00.00 %	00.00 %	2
Water Bottle	100.0 %	100.0 %	100.0 %	100.0 %	2
Eyes	27.27 %	00.00 %	36.36 %	36.36 %	11
Bath and Body	00.00 %	00.00 %	50.00 %	50.00 %	4
Gloves	75.00 %	50.00 %	100.0 %	100.0 %	8
Sports Accessories	00.00 %	00.00 %	00.00 %	00.00 %	2
Cufflinks	93.10 %	93.10 %	93.10 %	93.10 %	29
Sports Equipment	100.0 %	85.71 %	100.0 %	00.00 %	7
Stoles	84.37 %	87.50 %	87.50 %	87.50 %	32
Hair	50.00 %	00.00 %	50.00 %	50.00 %	2
Perfumes	00.00 %	00.00 %	00.00 %	00.00 %	5
Home Furnishing	00.00 %	00.00 %	00.00 %	100.0 %	1
Umbrellas	100.0 %	50.00 %	100.0 %	100.0 %	2
Wristbands	100.0 %	00.00 %	100.0 %	100.0 %	1

Vouchers	00.00 %	00.00 %	00.00 %	00.00 %	0

Ilustración 34 Tabla de sensibilidad de SubCategory

De primeras, solo entrenando la parte de clasificación la red ya llega a un valor de un 97.41% de precisión, lo cual es destacable, llegando a un máximo de un 97.64% al aplicar fine-tunning de 2 capas. Para esta categoría vemos que aplicar más capas tiene una muy ligera mejora en la clasificación. Aquí ya se empieza a notar como la poca cantidad de elementos de algunas clases tiende a afectar muy negativamente a sus valores (aunque no siempre ocurre así, ahí está como ejemplo la clase paraguas). Así todo, haciendo fine-tunning conseguimos salvar algunos de estos valores, obteniendo unos resultados más adecuados.

Durante la búsqueda de hiperparámetros dedicada a esta categoría se consiguió aumentar la exactitud muy ligeramente a un 97.69% teniendo los siguientes valores:

- Entrada y salida de la capa intermedia: 1024 -> 512
- Learning rate: 0.002078712010233878
- Batch size 16

Comprobaremos ahora que tanto se confunden los elementos entre ellos.

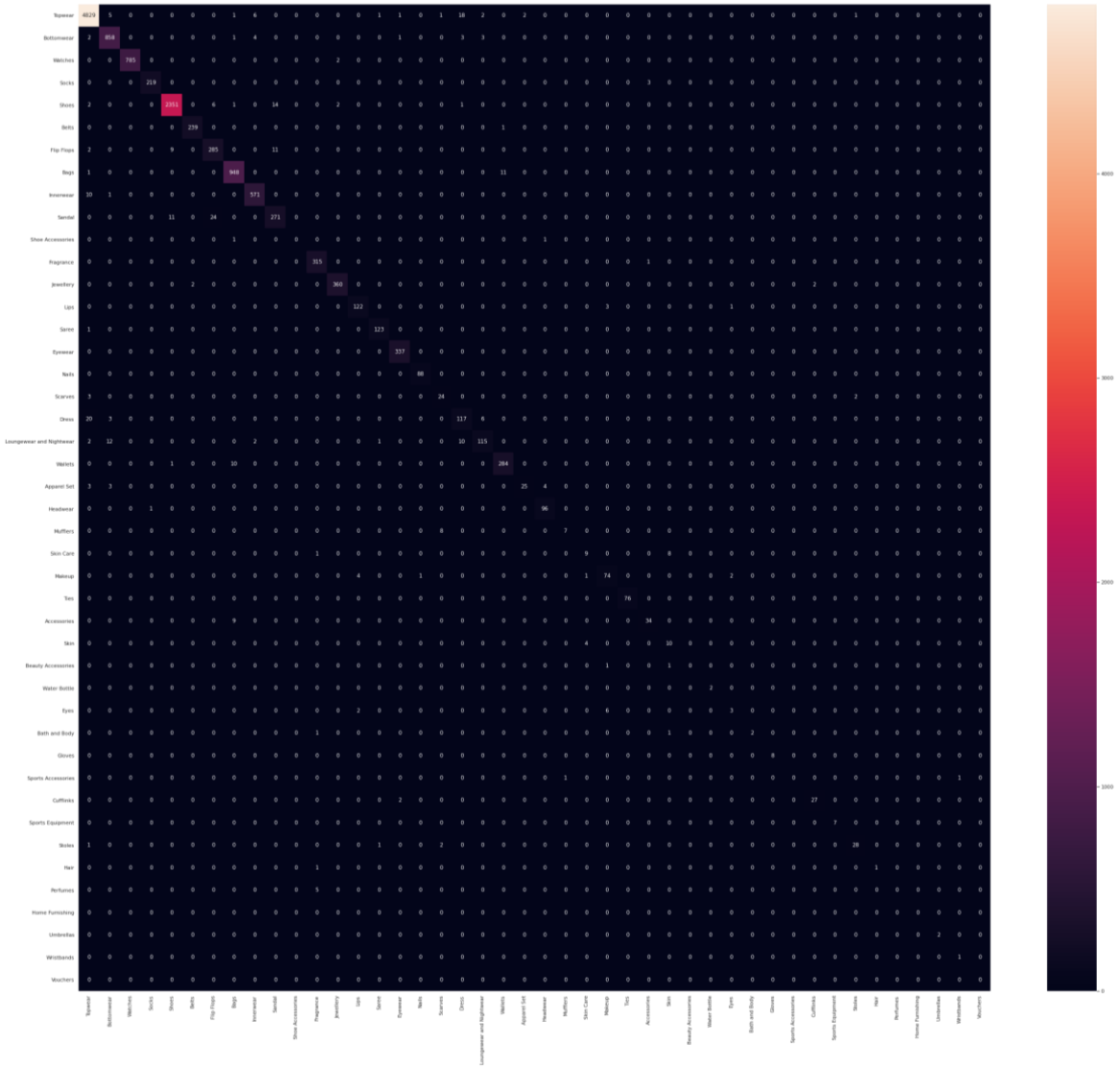


Ilustración 35 Matriz de confusión de SubCategory

De su matriz de confusión podemos observar al igual que en la categoría anterior, que los resultados que obtiene son bastante aceptables, a pesar de ser poco visible su capacidad en los elementos finales, que son por lo general los que menos elementos componen su clasificación. En cuanto a los errores que comete la red, descontando aquellos casos individuales, tenemos algunos casos como los elementos que representan calzado, que confunden algunos elementos entre ellos («shoes» siendo el predominante, seguido de «flip flops» y de «sandals»), los vestidos y la ropa de noche, similares en forma y que además se confunden bastante con las prendas superiores. En cuanto al resto, en algunos de los productos no textiles vemos casos como las fragancias y los perfumes y los productos de piel y de cuidado de la piel, que resultan casos muy similares entre ellos.

Caso nº3 Tipo de artículo

La categoría «ArticleType», como ya hemos comentado representa de la forma más específica que permite este dataset el tipo de elemento que representa la imagen. Las prendas superiores e inferiores tienen variantes. Los calzados y múltiples accesorios aumentan sus posibles clasificaciones con respecto a la anterior. Es la categoría más similar a las encontradas comúnmente en otros datasets de ropa. Al igual que con SubCategory, la clase «Free gifts» queda eliminada del conjunto.

Categoría	Tipo red	Mejor precisión	Épocas
Article Type	Entrenamiento clasificación	90.28 %	15
	Fine-tunning de una capa	89.53 %	15
	Fine-tunning de dos capas	91.22 %	15
	Fine-tunning de tres capas	91.21 %	15

Ilustración 36 Tabla de exactitud de ArticleType

Clasificación	Solo clasificación	FT de una capa	FT de dos capas	FT de tres capas	Nº elementos validación
Shirts	96.62 %	97.01 %	97.68 %	97.49 %	1036
Jeans	91.19 %	93.78 %	92.75 %	95.34 %	193
Watches	99.74 %	99.87 %	99.74 %	99.49 %	787
Track Pants	84.37 %	85.42 %	87.50 %	89.58 %	96
Tshirts	97.47 %	96.34 %	97.29 %	96.75 %	2214
Socks	98.18 %	98.18 %	98.64 %	98.64 %	220
Casual Shoes	87.04 %	87.14 %	88.45 %	87.36 %	918
Belts	98.76 %	98.35 %	98.76 %	98.76 %	242
Flip Flops	92.51 %	92.51 %	91.20 %	91.86 %	307
Handbags	96.58 %	93.54 %	96.20 %	97.15 %	526
Tops	73.69 %	75.43 %	77.35 %	76.65 %	574
Bra	99.34 %	98.68 %	99.34 %	99.34 %	152
Sandals	83.51 %	84.56 %	85.61 %	87.72 %	285
Shoe Accessories	00.00 %	00.00 %	00.00 %	00.00 %	2
Sweatshirts	82.72 %	83.95 %	87.65 %	87.65 %	81

Deodorant	95.73 %	94.87 %	97.43 %	97.43 %	117
Formal Shoes	93.78 %	88.04 %	87.08 %	87.56 %	209
Bracelet	40.00 %	35.00 %	60.00 %	65.00 %	20
Lipstick	98.75 %	98.75 %	98.75 %	98.75 %	180
Flats	39.10 %	33.97 %	47.43 %	42.31 %	156
Kurtas	95.23%	94.04 %	95.06 %	95.40 %	587
Waistcoat	33.33 %	33.33 %	33.33 %	33.33 %	12
Sports Shoes	87.87 %	88.03 %	88.98 %	91.02 %	635
Shorts	94.77 %	92.16 %	96.73 %	96.73 %	153
Briefs	96.41 %	96.06 %	96.41 %	96.06 %	279
Sarees	99.19 %	99.19 %	99.19 %	99.19 %	124
Perfume and Body Mist	96.30 %	96.30 %	96.82 %	96.82 %	189
Heels	92.16 %	89.11 %	88.67 %	90.63 %	459
Sunglasses	100.0 %	100.0 %	100.0 %	100.0 %	337
Innerwear Vests	85.90 %	88.46 %	85.90 %	89.74 %	78
Pendant	84.13 %	90.48 %	95.24 %	90.48 %	63
Nail Polish	100.0 %	98.86 %	100.0 %	100.0 %	88
Laptop Bag	53.57 %	57.14 %	60.71 %	64.28 %	28
Scarves	82.76 %	86.21 %	89.65 %	86.21 %	29
Rain Jacket	00.00 %	00.00 %	00.00 %	00.00 %	4
Dresses	85.00 %	80.71 %	83.57 %	87.14 %	140
Night suits	90.48 %	85.71 %	95.24 %	95.24 %	42
Skirts	87.23 %	74.47 %	85.11 %	91.49 %	47
Wallets	96.96 %	95.61 %	97.97 %	96.28 %	296
Blazers	100.0 %	100.0 %	100.0 %	100.0 %	2
Ring	92.10 %	94.74 %	97.37 %	97.37 %	38
Kurta Sets	100.0 %	92.86 %	96.43 %	100.0 %	28
Clutches	79.57 %	74.19 %	76.34 %	78.49 %	93
Shrug	00.00 %	00.00 %	00.00 %	00.00 %	1
Backpacks	96.64 %	95.38 %	96.64 %	96.64 %	238
Caps	100.0 %	98.95 %	100.0 %	100.0 %	95
Trousers	89.59 %	89.02 %	92.48 %	93.06 %	173
Earrings	96.38 %	97.10 %	98.55 %	97.83 %	138
Camisoles	83.33 %	66.67 %	100.0 %	75.00%	12

Boxers	91.67 %	91.67 %	91.67 %	91.67 %	12
Jewellery Set	94.11 %	100.0 %	100.0 %	100.0 %	17
Dupatta	94.59 %	94.59 %	94.59 %	94.59 %	37
Capris	68.18 %	69.70 %	78.79 %	75.75 %	66
Lip Gloss	93.75 %	93.75 %	93.75 %	100.0 %	32
Bath Robe	33.33 %	16.67 %	33.33 %	66.67 %	6
Mufflers	46.67 %	66.67 %	66.67 %	73.33 %	23
Tunics	21.43 %	17.14 %	32.86 %	31.43 %	70
Jackets	77.78 %	77.78 %	75.00 %	76.39 %	72
Trunk	89.13 %	93.48 %	95.65 %	91.30 %	46
Lounge Pants	66.67 %	66.67 %	53.33 %	60.00 %	15
Face Wash and Cleanser	28.57 %	42.86 %	71.43 %	42.86 %	7
Necklace and Chains	85.71 %	87.50 %	89.28 %	89.28 %	56
Duffel Bag	81.48 %	85.18 %	81.48 %	77.78 %	27
Sports Sandals	33.33 %	28.57 %	52.38 %	42.86 %	21
Foundation and Primer	80.00 %	100.0 %	100.0 %	75.00 %	49
Sweaters	71.43 %	70.33 %	75.82 %	76.92 %	91
Trolley Bag	00.00 %	00.00 %	00.00 %	00.00 %	0
Tracksuits	90.00 %	90.00 %	90.00 %	90.00 %	10
Swimwear	62.50 %	37.50 %	25.00 %	25.00 %	8
Shoe Laces	00.00 %	00.00 %	00.00 %	00.00 %	0
Fragrance Gift Set	80.00 %	80.00 %	80.00 %	73.33 %	15
Bangle	75.00 %	71.87 %	71.87 %	62.50 %	32
Nightdress	75.81 %	69.35 %	82.26%	75.81 %	62
Ties	100.0 %	100.0 %	100.0 %	98.68 %	76
Baby Dolls	40.00 %	40.00 %	60.00 %	60.00 %	5
Leggings	77.92 %	75.32 %	77.92 %	77.92 %	77
Highlighter and Blush	100.0 %	70.00 %	100.0 %	100.0 %	10
Travel Accessory	50.00 %	33.33 %	50.00 %	33.33 %	6
Kurtis	42.10 %	35.53 %	51.31 %	47.37 %	76
Mobile Pouch	57.89 %	42.10 %	57.89 %	57.89 %	19

Messenger Bag	31.25%	18.75 %	43.75 %	50.00 %	16
Lip Care	00.00 %	00.00 %	100.0 %	100.0 %	2
Face Moisturisers	80.00 %	90.00 %	80.00 %	90.00 %	10
Compact	93.33 %	86.67 %	93.33 %	93.33 %	15
Eye Cream	00.00 %	00.00 %	00.00 %	00.00 %	0
Accessory Gift Set	96.97 %	100.0 %	100.0 %	100.0 %	33
Beauty Accessory	00.00 %	00.00 %	00.00 %	00.00 %	2
Jumpsuit	00.00 %	16.67 %	16.67 %	16.67 %	6
Kajal and Eyeliner	80.00 %	76.67 %	66.67 %	83.33 %	30
Water Bottle	100.0 %	100.0 %	100.0 %	100.0 %	3
Suspenders	100.0 %	100.0 %	100.0 %	100.0 %	11
Lip Liner	70.00 %	40.00 %	70.00 %	60.00 %	10
Robe	00.00 %	00.00 %	00.00 %	00.00 %	2
Salwar and Dupatta	50.00 %	50.00 %	100.0%	100.0 %	2
Patiala	66.67 %	66.67 %	77.78 %	66.67 %	9
Stockings	78.57 %	85.71 %	78.57 %	92.86 %	14
Eyeshadow	80.00 %	80.00 %	90.00 %	90.00 %	10
Headband	00.00 %	00.00 %	00.00 %	00.00 %	1
Tights	00.00 %	00.00 %	00.00 %	00.00 %	2
Nail Essentials	00.00 %	00.00 %	00.00 %	00.00 %	3
Churidar	40.00 %	60.00 %	80.00 %	80.00 %	5
Lounge Tshirts	00.00 %	00.00 %	00.00 %	00.00 %	0
Face Scrub and Exfoliator	00.00 %	00.00 %	00.00 %	00.00 %	0
Lounge Shorts	11.11 %	11.11 %	22.22 %	44.44 %	9
Gloves	87.50 %	87.50 %	100.0 %	100.0 %	8
Mask and Peel	40.00 %	00.00 %	20.00 %	60.00 %	5
Wristbands	66.67 %	66.67 %	66.67 %	66.67 %	3
Tablet Sleeve	00.00 %	00.00 %	00.00 %	00.00 %	3
Ties and Cufflinks	00.00 %	00.00 %	00.00 %	00.00 %	2
Footballs	66.67 %	100.0 %	100.0 %	100.0 %	3
Stoles	81.25 %	81.25 %	87.50 %	87.50 %	32
Shapewear	00.00 %	33.33 %	33.33 %	33.33 %	3

Nehru Jackets	00.00 %	00.00 %	00.00 %	00.00 %	3
Salwar	63.64 %	63.64 %	72.73 %	63.64 %	11
Cufflinks	100.0 %	100.0 %	100.0 %	100.0 %	27
Jeggings	27.27 %	27.27 %	27.27 %	18.18 %	11
Hair Colour	50.00 %	00.00 %	50.00 %	50.00 %	2
Concealer	00.00 %	00.00 %	20.00 %	20.00 %	5
Rompers	100.0 %	100.0 %	100.0 %	100.0 %	3
Body Lotion	00.00 %	00.00 %	00.00 %	00.00 %	0
Sunscreen	28.57 %	14.28 %	28.57 %	28.57 %	7
Booties	100.0 %	100.0 %	100.0 %	100.0 %	2
Waist Pouch	00.00 %	00.00 %	50.00 %	25.00 %	4
Hair Accessory	00.00 %	00.00 %	00.00 %	00.00 %	0
Rucksacks	00.00 %	00.00 %	00.00 %	12.50 %	8
Basketballs	100.0 %	75.00 %	100.0 %	100.0 %	4
Lehenga Choli	00.00 %	00.00 %	00.00 %	00.00 %	1
Clothing Set	00.00 %	00.00 %	00.00 %	00.00 %	3
Mascara	33.33 %	66.67 %	33.33 %	33.33 %	3
Toner	00.00 %	00.00 %	00.00 %	00.00 %	1
Cushion Covers	00.00 %	00.00 %	00.00 %	00.00 %	0
Key chain	00.00 %	00.00 %	00.00 %	00.00 %	0
Makeup Remover	00.00 %	00.00 %	00.00 %	00.00 %	0
Lip Plumper	100.0 %	50.00 %	100.0 %	100.0 %	2
Umbrellas	100.0 %	100.0 %	100.0 %	100.0 %	2
Face Serum and Gel	00.00 %	00.00 %	00.00 %	00.00 %	1
Hat	00.00 %	00.00 %	00.00 %	00.00 %	1
Mens Grooming Kit	00.00 %	00.00 %	00.00 %	00.00 %	1
Rain Trousers	00.00 %	00.00 %	00.00 %	00.00 %	0
Body Wash and Scrub	00.00 %	00.00 %	00.00 %	00.00 %	1
Suits	00.00 %	00.00 %	00.00 %	00.00 %	0
Ipad	00.00 %	00.00 %	00.00 %	00.00 %	0

Ilustración 37 Tabla de sensibilidad de ArticleType

En esta tabla al contrario que en la anterior se notan muchos más problemas de clasificación, principalmente teniendo categorías que en la división de validación y entrenamiento no tuvieron suficientes elementos como para dividirse correctamente y por lo tanto aparecen nulos en validación. Claramente debido a esto la sensibilidad de la red respecto a las clases más pequeñas es ridículamente baja. Solo se consiguen salvar aquellas clases lo suficientemente distintivas del resto. Afortunadamente, muchas de estas pertenecen a accesorios u consumibles muy concretos que no tiene tanta importancia en nuestro objetivo de clasificación y que por lo tanto son desechables. Además, esto no se aplica a todos y aún con pocos elementos hay algunos que llegan a destacar, y no se puede llegar a confirmar del todo que el número de elementos influya mucho. Como aporte adicional, se realizó un estudio de la correlación entre el número de elementos de cada clase y su porcentaje de acierto, llegando entre el 0% y el 100% a valores sobre el 45% de relación. Aun así, los porcentajes obtenidos en la mayoría de sus clases son aceptables.

Durante la búsqueda de hiperparámetros dedicada a esta categoría se consiguió aumentar la exactitud a un 91.54% teniendo los siguientes valores:

- Entrada y salida de la capa intermedia: 1024 -> 4096
- Learning rate: 0.005194759474568325
- Batch size 16

Veremos a continuación en la matriz de confusión los motivos por los cuales algunos se confunden.

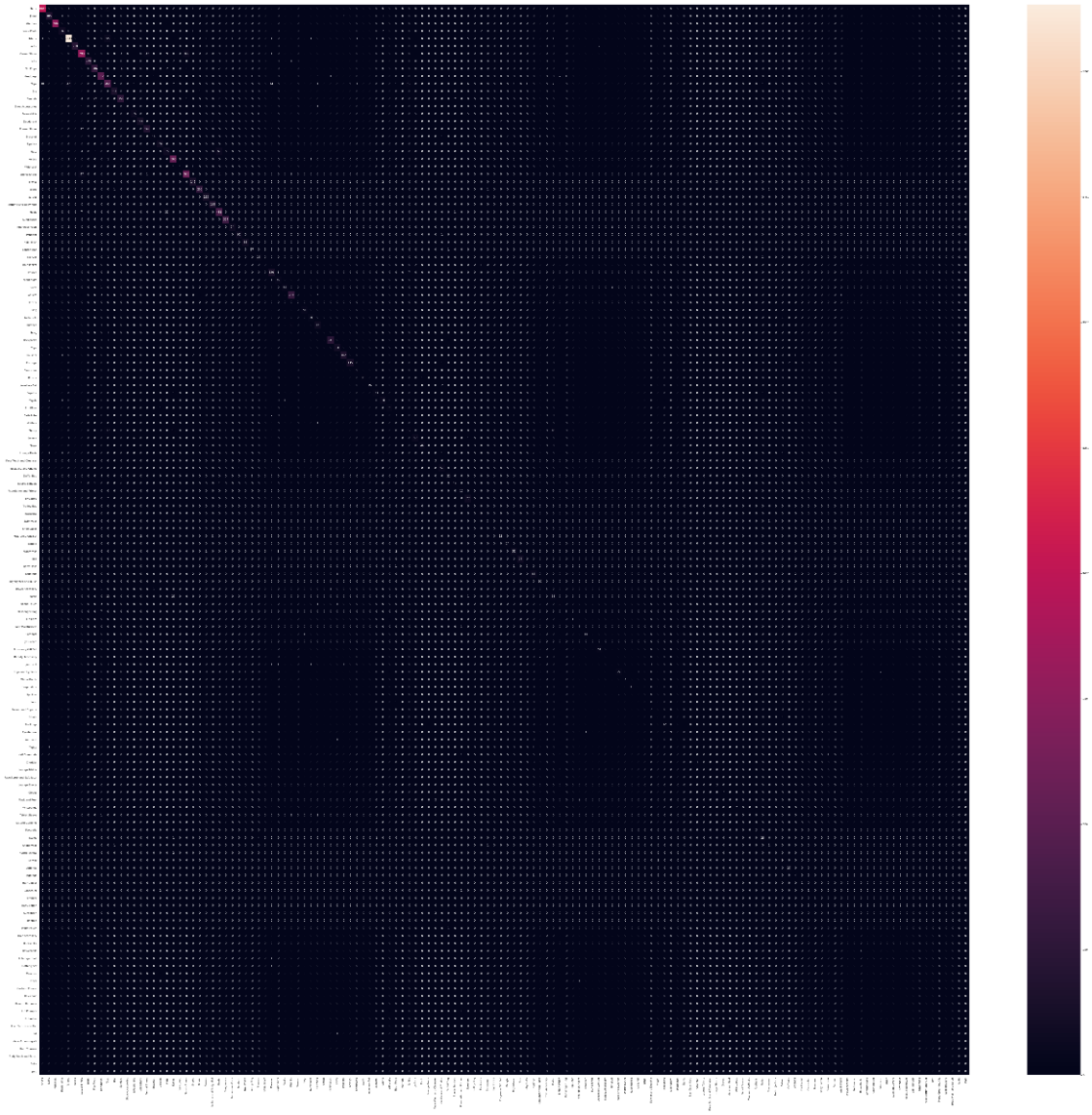


Ilustración 38 Matriz de confusión de ArticleType

Al contrario que en las dos anteriores categorías, en el caso de los tipos de artículo además de encontrarse un mayor número de errores dada una menor exactitud que presentaba la red, también se encuentran más “dispersas” las confusiones pues estamos tratando con un número aún mayor de categorías de las cuales no siempre se tienen suficientes ejemplos. Vamos a tratar algunos de los casos más graves (aquellos con un número más elevado de errores) de confusión:

En primer lugar, entre las confusiones de prendas de arriba, podemos observar que la clase «tops» tiende a confundirse bastante con otras, sobre todo con la clase «t-shirts» y algo menos con «shirts», curiosamente estas últimas dos entre si no presentan casi problemas. Además, a pesar de su general diferencia en tamaños, se ven varios casos, aunque no demasiados, de equivocaciones con prendas largas como «dresses», «kurtas» o «tunics». La red presenta errores con estas prendas largas, siendo confundidas unas con otras, aunque con mucha menos frecuencia de la que se podría esperar de elementos tan parecidos.

Al contrario que con los vestidos y como se vio en «Master Category», los zapatos tienden a confundirse unos con otros con frecuencia siendo entre todas las mezclas que realiza más predominantes sobre:

- «Sandals», «flip flops» y «sport sandals», comprensible siendo varios tipos de zapatillas.
- «Formal shoes» con «casual shoes», posiblemente por algún estilo similar.
- «Casual shoes» con «sport shoes», que se confunden bastante, seguramente por el estilo de playeras de algunos zapatos casuales.
- «Flats» y «heels», que se confunden frecuentemente, seguramente por la forma de estos, aunque encontramos interesante que no sea tan capaz de distinguir el tacón.

Por último, destacamos algunos fallos no tan comunes pero que se han dado como los de «jackets» con otras partes de arriba (principalmente «shirts») o las de «wallets» con «clutches», entendible por su parecido.

El resto de las clases o se confunden con poca frecuencia o ninguna, o no se disponen de suficientes elementos suyos como para poder inducir algún razonamiento o ser diferenciados de similares de mayor magnitud. De todas formas, teniendo en cuenta la cantidad de clasificaciones para esta categoría de los cuales dispone el dataset y los principales errores que se han comentado, podemos deducir que hace un buen trabajo en su cometido.

Caso nº 4: Color base.

Probaremos a pasar a otra de las categorías a ver si conseguimos que otros aspectos no tan basados en sus características sean obtenidos. Empezaremos con la categoría baseColour. BaseColour como ya hemos comentado anteriormente, representa el color predominante o base de un elemento. Por ejemplo, si una camisa blanca tuviera un gran dibujo en el centro, su color base seguiría siendo blanco. Los resultados obtenidos con el fine-tunning fueron los siguientes.

Categoría	Tipo red	Mejor precisión	Épocas
BaseColour	Entrenamiento clasificación	58.16 %	20
	Fine-tunning de una capa	60.69 %	20
	Fine-tunning de dos capas	64.78 %	20
	Fine-tunning de tres capas	66.72 %	20

Ilustración 39 Tabla de exactitud de BaseColour

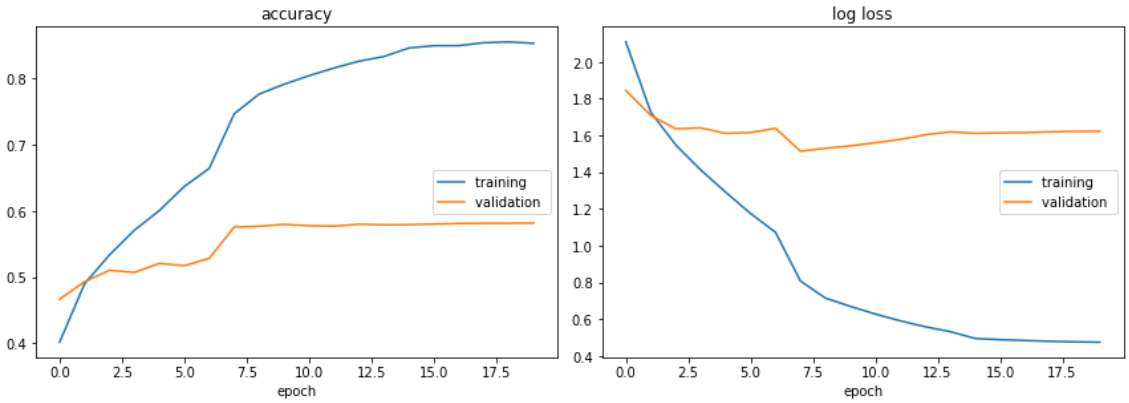


Ilustración 40 Gráfica del entrenamiento de clasificación

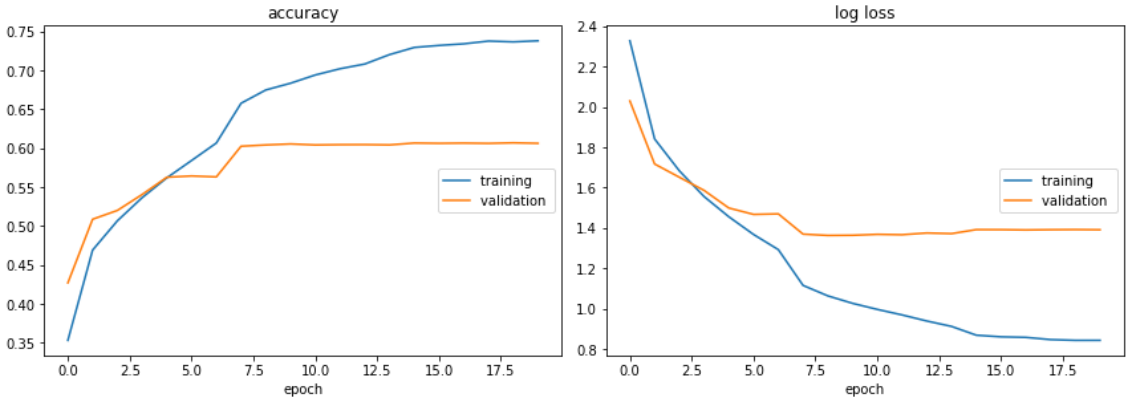


Ilustración 41Gráfica del entrenamiento fine-tunning 1

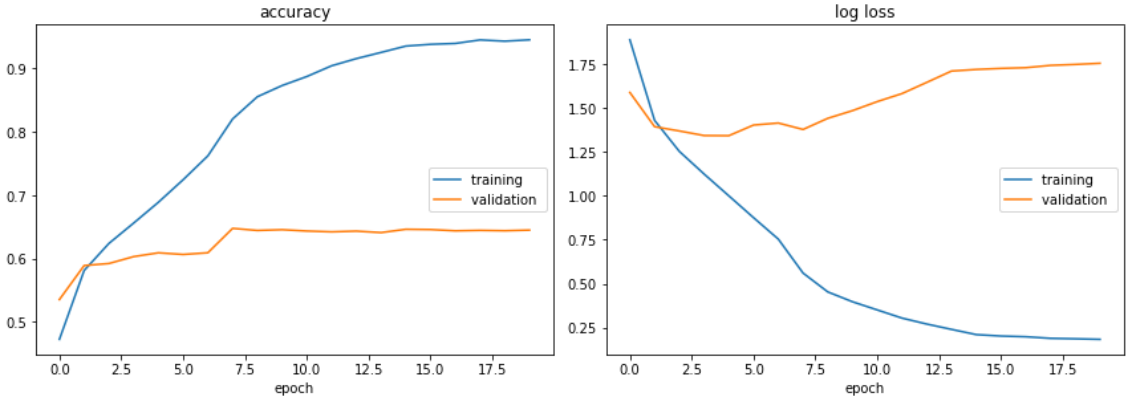


Ilustración 42 Gráfica del entrenamiento fine-tunning 2

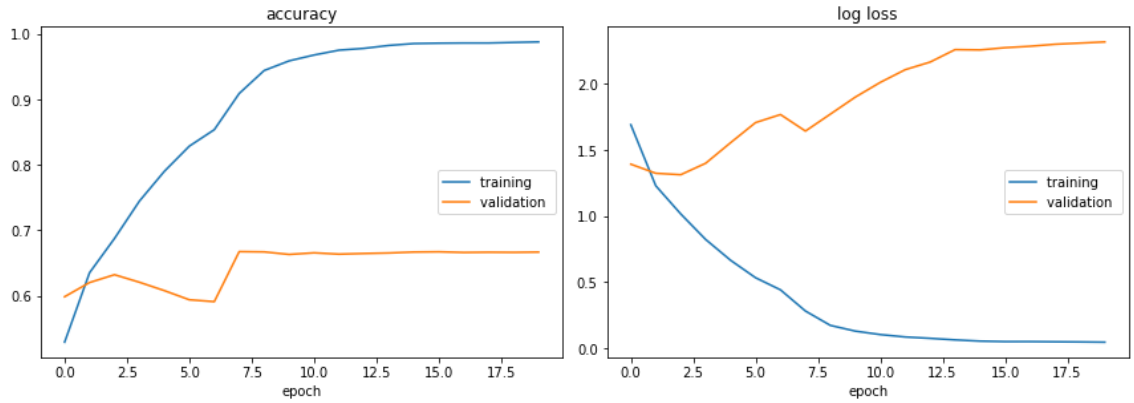


Ilustración 43 Gráfica del entrenamiento fine-tunning 3

La exactitud que proporciona en estos casos la red VGG y sus ligeras modificaciones con fine-tunning, es la peor exactitud obtenida de entre todas las clases clasificadas hasta el momento, llegando como máximo a un 66.72 % de acierto en validación, siendo en la que más influye la técnica de fine-tunning. Esto se hace ver mucho más a la hora de estudiar la sensibilidad de la red.

Clasificación	Solo clasificación	FT de una capa	FT de dos capas	FT de tres capas	Nº elementos validación
Navy Blue	32.73 %	33.81 %	46.11 %	45.75 %	553.0
Blue	57.96 %	62.89 %	68.07 %	72.68 %	1563
Silver	61.28 %	57.66 %	61.28 %	66.02 %	359
Black	80.53 %	80.83 %	84.44 %	84.04 %	2989
Grey	39.51 %	46.69 %	49.61 %	49.72 %	891
Green	47.85 %	55.16 %	62.61 %	49.72 %	698
Purple	29.86 %	38.04 %	50.10 %	51.53 %	489
White	77.84 %	78.35 %	78.01 %	80.92 %	1751
Beige	31.87 %	33.47 %	42.23 %	41.43 %	251
Brown	61.47 %	64.95 %	62.84 %	68.81 %	1090
Bronze	9.090 %	00.00 %	9.091 %	13.64 %	22
Teal	5.263 %	5.263 %	5.263 %	7.895 %	38
Copper	3.846 %	7.692 %	3.846 %	7.692 %	26
Pink	50.87 %	55.38 %	59.37 %	60.07 %	576
Off White	21.15 %	23.08 %	28.85 %	26.92 %	52
Maroon	20.00 %	15.55 %	27.22 %	30.55 %	180
Red	69.53 %	73.96 %	74.97 %	76.61 %	791

Khaki	28.26 %	26.09 %	28.26 %	28.26 %	46
Orange	43.67 %	48.10 %	58.86 %	56.33 %	158
Coffee Brown	00.00 %	00.00 %	00.00 %	00.00 %	18
Yellow	62.93 %	70.27 %	77.99 %	77.61 %	259
Charcoal	17.86 %	13.09 %	27.38 %	33.33 %	84
Gold	51.77 %	54.42 %	56.64 %	56.64 %	226
Steel	40.66 %	41.76 %	54.94 %	43.96 %	91
Tan	14.70 %	20.59 %	20.59 %	20.59 %	34
Multi	31.90 %	24.14 %	25.86 %	28.45 %	116
Magenta	4.444 %	2.222 %	4.444 %	20.00 %	45
Lavender	6.122 %	2.041 %	20.41 %	26.53 %	49
Sea Green	00.00 %	00.00 %	00.00 %	00.00 %	6
Cream	11.86 %	14.41 %	14.41 %	15.25 %	118
Peach	19.64 %	12.50 %	21.43 %	25.00 %	56
Olive	22.95 %	25.41 %	45.90 %	48.36 %	122
Skin	83.33 %	72.92 %	77.08 %	91.66 %	84
Burgundy	00.00 %	00.00 %	00.00 %	00.00 %	17
Grey Melange	27.66 %	23.40 %	36.17 %	36.17 %	47
Rust	00.00 %	13.04 %	4.348 %	30.43 %	23
Rose	00.00 %	00.00 %	00.00 %	00.00 %	8
Lime Green	00.00 %	00.00 %	00.00 %	00.00 %	1
Mauve	00.00 %	00.00 %	00.00 %	00.00 %	8
Turquoise Blue	00.00 %	00.00 %	00.00 %	4.167 %	24
Metallic	11.76 %	00.00 %	00.00 %	00.00 %	17
Mustard	19.23 %	26.92 %	30.77 %	38.46 %	26
Taupe	00.00 %	00.00 %	00.00 %	00.00 %	2
Nude	00.00 %	00.00 %	00.00 %	00.00 %	7
Mushroom Brown	00.00 %	00.00 %	00.00 %	00.00 %	4
nan	00.00 %	00.00 %	00.00 %	00.00 %	3
Fluorescent Green	00.00 %	00.00 %	00.00 %	00.00 %	1

Ilustración 44 Tabla de sensibilidad de BaseColour

Se trata además del modelo en el cual el overfitting es más visible, pues si bien en las gráficas actuales no es visible, en varias pruebas realizadas para el modelo se observó que la exactitud en validación caía drásticamente, aunque cabe destacar que con el entrenamiento único de la clasificación y el fine-tunning de 1 sola capa no son tan elevados los aciertos en el conjunto de entrenamiento. Como en anteriores casos, las

clases con mayor número de elementos son aquellas en las que se suele ver mayor cantidad de aciertos (salvo skin, que suponemos que se debe a que la red clasifica de otra forma que se trata de piel y no de una prenda).

Durante la búsqueda de hiperparámetros dedicada a esta categoría se consiguió aumentar la exactitud a un 66.83% teniendo los siguientes valores:

- Entrada y salida de la capa intermedia: 512 -> 4096
- Learning rate: 0.0012332514118440143
- Batch size 16

Pasemos a observar los datos obtenidos de la matriz de confusión.

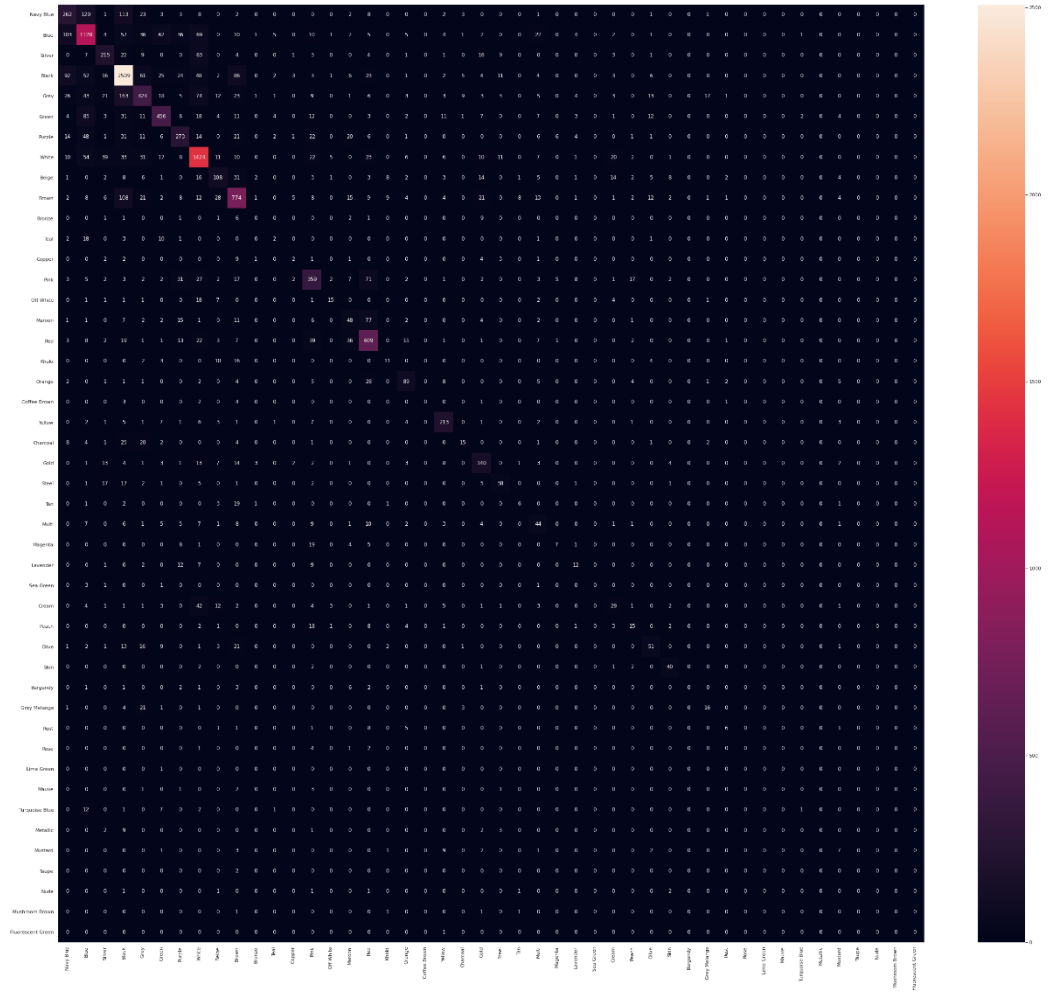


Ilustración 45 Matriz de confusión de BaseColour

Examinando los resultados de la matriz de confusión, si bien sigue destacando la falta de buenos resultados, podemos llegar a tener indicios de por qué. Además de los pocos ejemplos de cada una de las posibles clasificaciones, una enorme parte de ellos se tratan de distintas tonalidades del mismo color que para algunos usuarios no

acostumbrados a distinguir colores serían nombrados todos bajo un mismo nombre, por ejemplo, marrón para el mismo, bronce, cobre, marrón café, marrón champiñón, etc y así con otros colores. Se ve en varios casos como, por ejemplo, en el color «grey melange», que únicamente clasifica bien 17 de sus elementos, teniendo 25 clasificados como «grey». Esto se repite con distintas tonalidades de color, así como con otros casos no tan evidentes que dependiendo del matiz del color pueden ser confundidos como el violeta («purple»), que tiene bastantes confusiones con las tonalidades de azul («blue» y «navyblue») a la vez que con las rojas («pink» y «red»), teniendo en cuenta que es una combinación de ambos colores. Además, la gran mayoría de los colores se confunden constantemente con los colores blanco y negro. Al igual que en casos anteriores el número de elementos también pasa factura a la hora de clasificar.

Como comentamos anteriormente, el conjunto de entrenamiento llegó a tener muy buenos resultados (por encima del 90% con tres capas de fine-tunning), así que la conclusión a la que llegamos es que, a pesar de estar en la misma clase, los componentes de cada una no representan del todo las mismas características, y por lo tanto el cambio de conjunto hace que varios de los colores clasificados en validación no sean reconocidos como los correspondientes. Creemos que se trata de varias diferencias entre tonalidades consideradas el mismo color que si bien a ojos humanos se ven similares, la red hace uso de límites bien establecidos en base al conjunto de entrenamiento que no corresponden con una parte de las tonalidades que se encuentran en validación.

Se probó a deshacerse de la normalización de la entrada, por si suponía alguna mejora el disponer de los colores “puros” a la hora de clasificarlos, sin embargo, solo conllevó a un gran ejemplo de overfitting, haciendo caer en picado la exactitud.

Caso nº5 Género

La categoría «Gender», como ya hemos comentado, representa el sexo para el cual se ha hecho el elemento a clasificar. Esta clasificación no se divide solo en hombres y mujeres, sino que también tiene datos referentes a la edad del individuo, pudiendo ser de niño o niña. También contiene la clase unisex, que engloba hombres y mujeres, que, si bien puede confundir a la red, nos interesa comprobar su eficacia.

Categoría	Tipo red	Mejor precisión	Épocas
Gender	Entrenamiento clasificación	93.66 %	20
	Fine-tunning de una capa	92.46 %	20
	Fine-tunning de dos capas	94.22 %	20
	Fine-tunning de tres capas	94.35 %	20

Ilustración 46 Tabla de exactitud de Gender

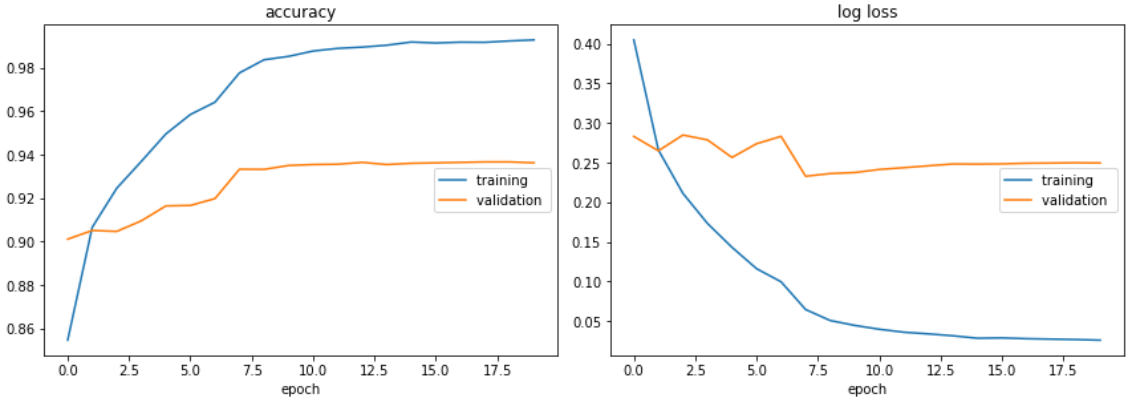


Ilustración 47 Gráfica del entrenamiento de clasificación

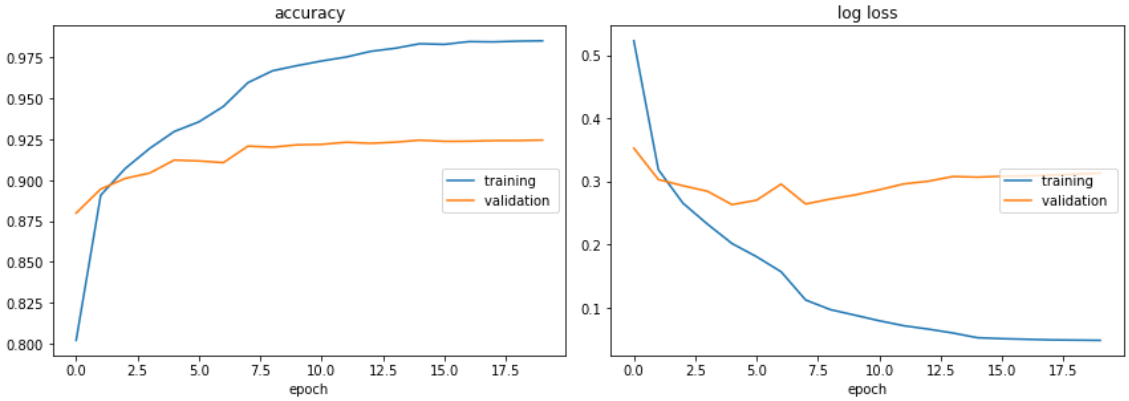


Ilustración 48 Gráfica del entrenamiento fine-tunning 1

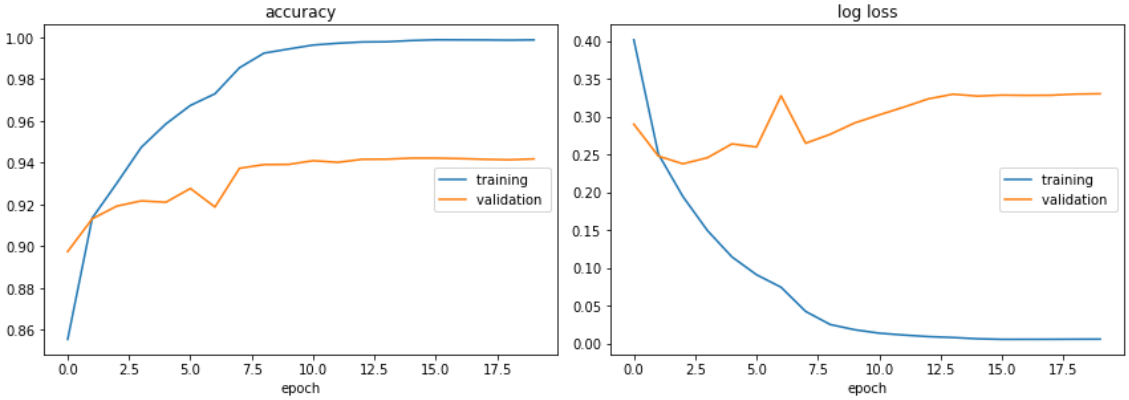


Ilustración 49 Gráfica del entrenamiento fine-tunning 2

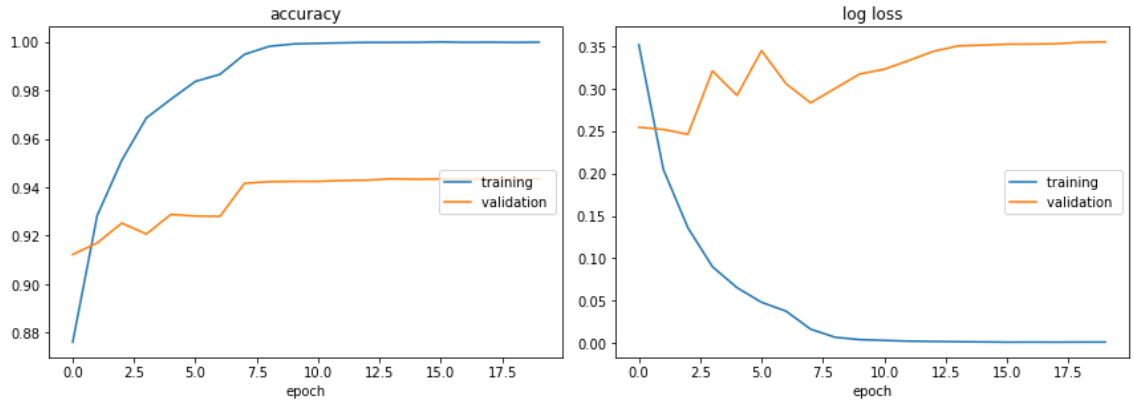


Ilustración 50 Gráfica del entrenamiento fine-tunning 3

De forma general como en casos anteriores logramos unos resultados excelentes, con un máximo de un 94.35% de acierto al entrenar usando fine-tunning de tres capas de profundidad. En cuanto a la sensibilidad de la red, vemos que individualmente como de costumbre las dos clases más influyentes en el conjunto son las que mejor son clasificadas. Aun con la enorme diferencia de elementos en comparación a las prendas utilizadas por menores, estas últimas no se quedan atrás, superando ambos el 75% de exactitud. Unisex como era de esperar no es tan fácil de clasificar, probablemente porque coinciden tanto con hombres como con mujeres.

Clasificación	Solo clasificación	FT de una capa	FT de dos capas	FT de tres capas	Nº elementos validación
Men	96.04 %	94.98 %	96.54 %	96.44 %	6997
Women	94.62 %	93.62 %	95.01 %	95.44 %	5836
Boys	84.73 %	80.53 %	86.26 %	87.79 %	262
Girls	79.02 %	75.12 %	83.90 %	82.44 %	205
Unisex	68.64 %	66.12 %	69.53 %	69.38 %	676

Ilustración 51 Tabla de sensibilidad de Gender

Durante la búsqueda de hiperparámetros dedicada a esta categoría se consiguió aumentar la exactitud a un 94.51% teniendo los siguientes valores:

- Entrada y salida de la capa intermedia: 8192 -> 512
- Learning rate: 0.014513678337298972
- Batch size 64

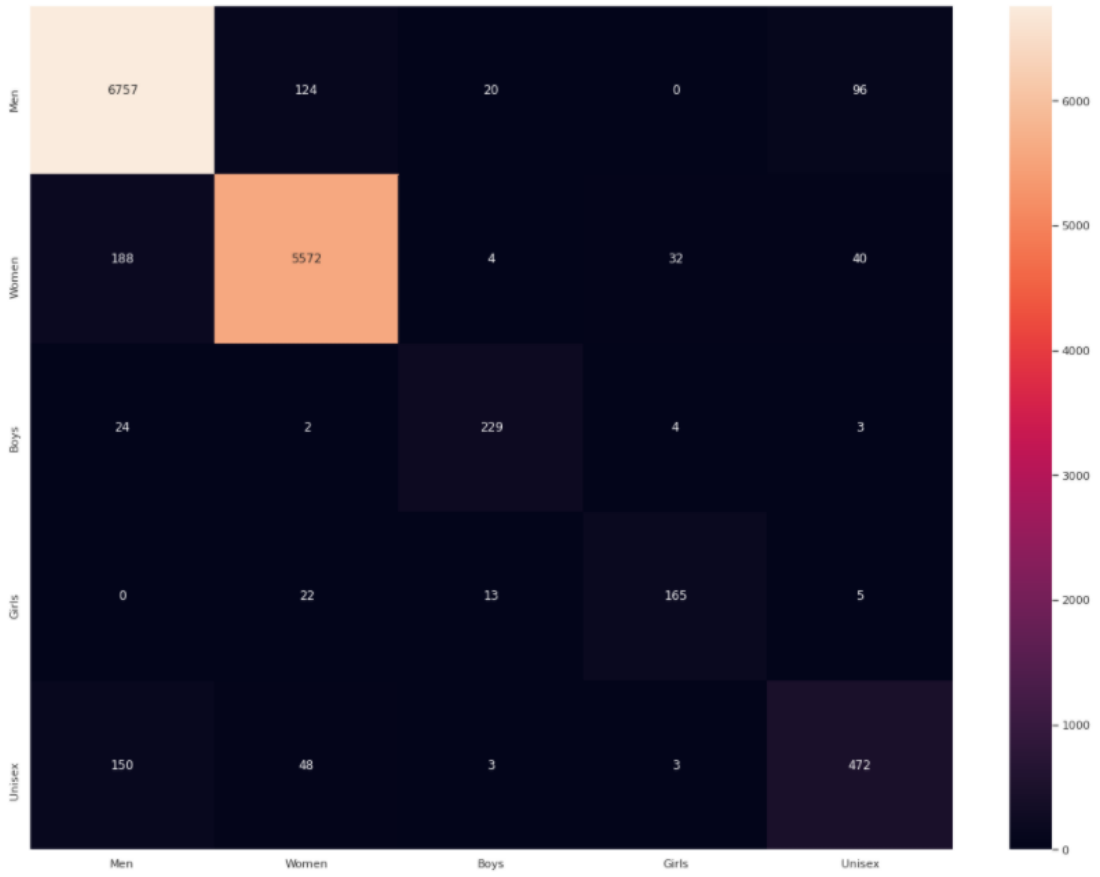


Ilustración 52 Matriz de confusión de Gender

La matriz nos permite apreciar que la clasificación que se realiza respecto al género es bastante acertada. La mayoría de los errores que comete a la hora de clasificar son o cuando están en la misma edad o cuando se trata del mismo sexo. Véase, hay algunas prendas de hombres que confunde con ropas de mujeres, así como con ropas de niños, y a su vez la ropa de niña se confunde ligeramente con la de niño y con la de mujer. Claramente unisex supone bastantes confusiones cuando se compara con las prendas de hombres y mujeres adultos, al compartir características con ambos.

En conclusión, la red es capaz de diferenciar excelentemente un género de otro, e incluso la edad a la que está destinada. Distinguir valores unisex, sigue siendo más complejo, sobre todo considerando que entre sexos aún hay características que los hacen confundirse entre ellos, pero es destacable que lo diferencie con un porcentaje mayor al 50%.

Caso nº6 Estación o temporada

La categoría «Season», como ya hemos comentado representa la temporada del año para la cual estaría pensada el elemento del dataset. Esta, al contrario que otras, es menos distinguible y podría suponer un mayor problema de clasificación para la red.

Categoría	Tipo red	Mejor precisión	Épocas
Season	Entrenamiento clasificación	78.72 %	20
	Fine-tunning de una capa	76.03 %	20
	Fine-tunning de dos capas	79.49 %	20
	Fine-tunning de tres capas	81.21 %	20

Ilustración 53 Tabla de exactitud de Season

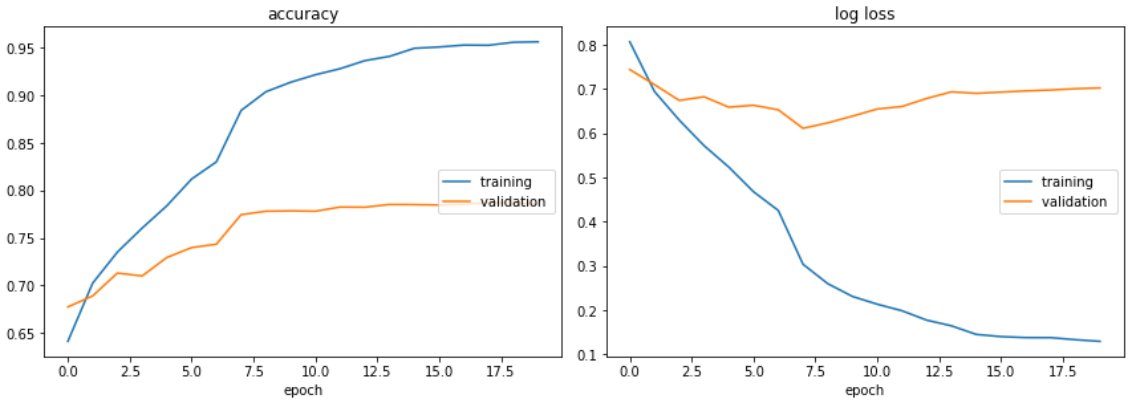


Ilustración 54 Gráfica del entrenamiento de clasificación

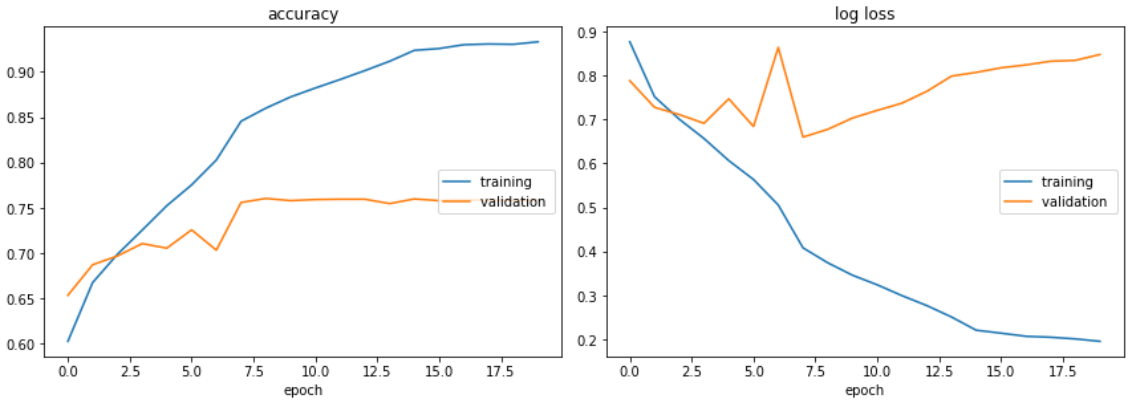


Ilustración 55 Gráfica del entrenamiento fine-tunning 1

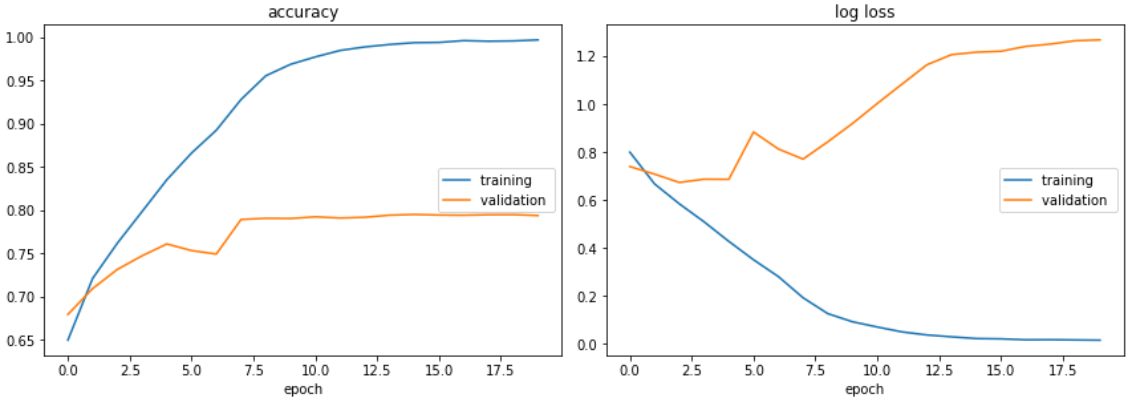


Ilustración 56 Gráfica del entrenamiento fine-tunning 2

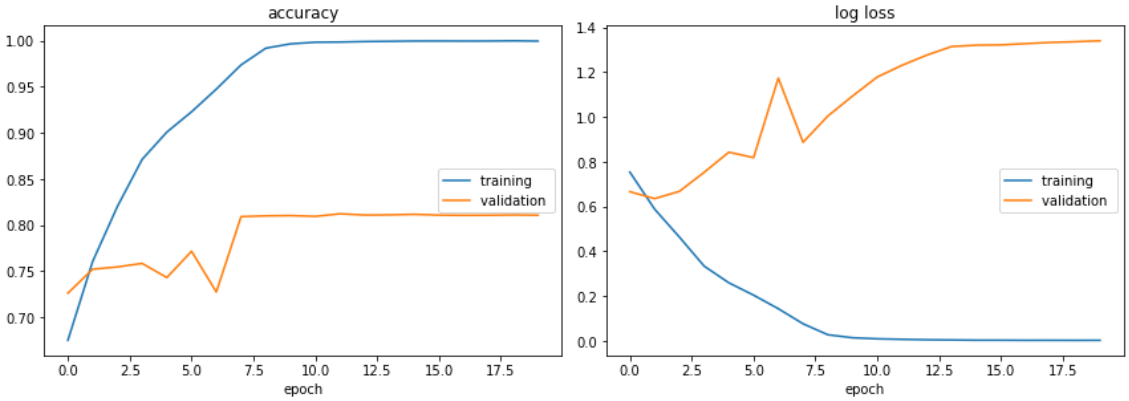


Ilustración 57 Gráfica del entrenamiento fine-tunning 3

A pesar de tratarse de una característica que incluso a las personas les puede costar distinguir (yo incluido), sobre el conjunto de validación la red obtiene un acierto de un 81.21% en su mejor versión (tras entrenarla con fine-tunning de 3 capas de profundidad). Si bien no se trata de un porcentaje tan alto como en otros casos, teniendo en cuenta lo dicho es sorprendente.

En cuanto a los resultados individuales, a pesar de que entre las clases hay bastante diferencia de elementos, la diferencia de porcentajes no es tan alta pues se ve que no influye tanto en esta clasificación cuando siendo otoño («Fall») la segunda con más elementos es la que menos consigue acertar, incluso por debajo de primavera («Spring»).

Clasificación	Solo clasificación	FT de una capa	FT de dos capas	FT de tres capas	Nº elementos validación
Fall	67.16 %	66.97 %	70.48 %	72.51 %	3685
Summer	84.57 %	81.72 %	84.41 %	85.88 %	6780
Winter	79.12 %	73.18 %	78.56 %	80.92 %	2673

Spring	81.03 %	78.98 %	82.46 %	82.58 %	838

Ilustración 58 Tabla de sensibilidad de Season

Tras una extensa búsqueda por optimizar los hiperparámetros del modelo no se llegaron a encontrar variaciones que mejoraran la exactitud del modelo base en lo más mínimo.

Veamos si la matriz de confusión obtenida aclara algún aspecto de la sensibilidad.

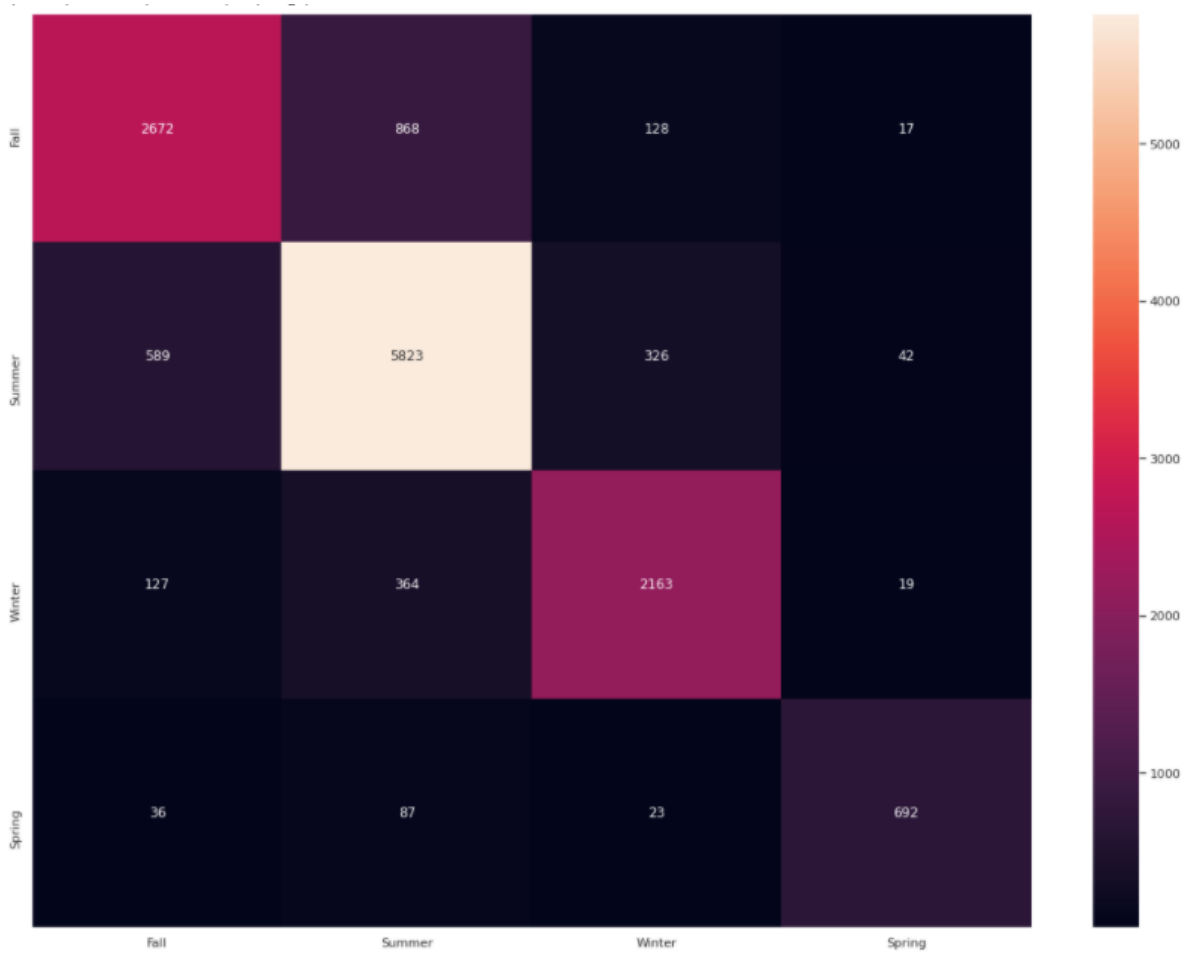


Ilustración 59 Matriz de confusión de Season

La matriz de confusión en este caso no aporta tanta información, ya que para las tres categorías con más elementos las equivocaciones son varias. Incluso destacamos como si bien es normal que se confundan más con verano, por ser el que más tiene, encontramos pocas confusiones entre invierno y otoño en comparación a las encontradas con verano. Vemos por otra parte que difícilmente se confunden con primavera (también influyendo su conjunto de elementos), siendo verano la que más cosas tiene en común con ella desde el punto de vista de la de menos elementos.

Consideramos los resultados obtenidos aceptables, pues si bien se comenten bastantes confusiones entre sus clases, dado el concepto que estamos tratando en el cual una ropa sacada en otoño no tiene por qué diferenciarse mucho de una de invierno, consigue clasificar correctamente la gran mayoría.

Caso nº7 Uso

La categoría «Usage», como ya hemos comentado representa el uso que se le daría a cada elemento en el dataset.

Categoría	Tipo red	Mejor precisión	Épocas
Usage	Entrenamiento clasificación	93.02 %	20
	Fine-tunning de una capa	92.06 %	20
	Fine-tunning de dos capas	93.19 %	20
	Fine-tunning de tres capas	93.52 %	20

Ilustración 60 Tabla de exactitud de Usage

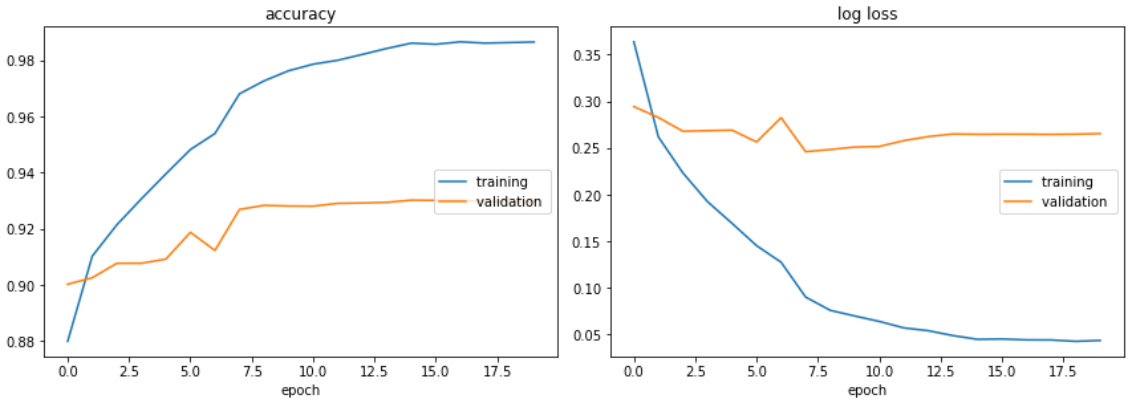


Ilustración 61 Gráfica del entrenamiento de clasificación

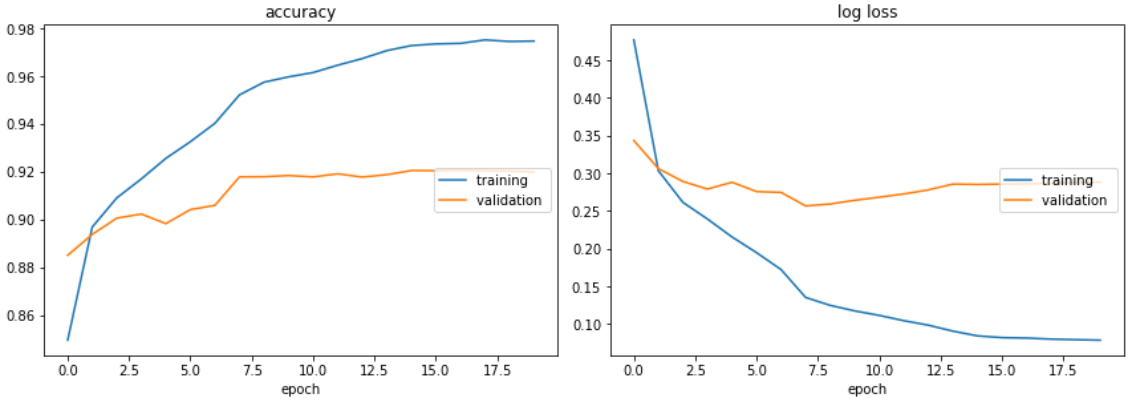


Ilustración 62 Gráfica del entrenamiento fine-tunning 1

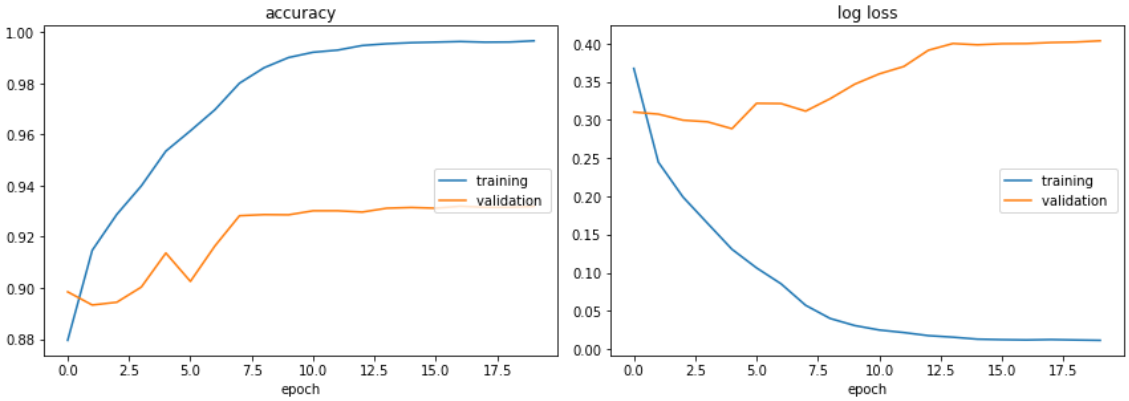


Ilustración 63 Gráfica del entrenamiento fine-tunning 2

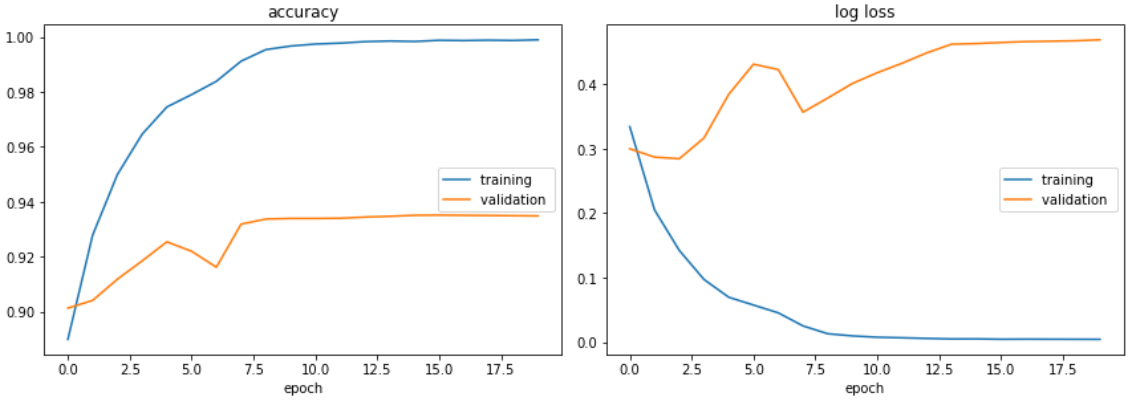


Ilustración 64 Gráfica del entrenamiento fine-tunning 3

En general, con una exactitud general máxima de un 93.52% conseguido durante el entrenamiento con fine-tunning de 3 capas de profundidad, habiendo superado el 90% podemos deducir que nuestra red hace un excelente ejercicio de clasificación. Si bien no es una clasificación tan abstracta como la temporada, es bastante sorprendente los resultados que se obtienen de su entrenamiento.

A pesar de esto, si pasamos a comprender el número desde una perspectiva individual, vemos que de las 8 clases solo 4 de ellas obtienen buenos resultados. De estas 4 «casual» es la que más destaca por todos los ejemplos de los que dispone, siguiéndole

étnica con un porcentaje adecuado y razonable sabiendo que se tratan de ropas más inusuales. Los resultados de «formal» y «casual» siguen siendo bastante aceptables, aunque podrían haber mejorado.

Clasificación	Solo clasificación	FT de una capa	FT de dos capas	FT de tres capas	Nº elementos validación
Casual	96.27 %	95.32 %	96.00 %	96.24 %	10885
Ethnic	91.03 %	89.75 %	92.02 %	92.81 %	1015
Formal	82.49 %	81.34 %	84.14 %	83.76 %	788
Sport	75.56 %	74.92 %	78.11 %	79.23 %	1252
Smart Casual	00.00 %	00.00 %	00.00 %	00.00 %	21
Travel	14.28 %	00.00 %	00.00 %	00.00 %	7
Party	00.00 %	00.00 %	00.00 %	00.00 %	8
Home	00.00 %	00.00 %	00.00 %	00.00 %	0

Ilustración 65 Tabla de sensibilidad de Usage

Durante la búsqueda de hiperparámetros dedicada a esta categoría se consiguió aumentar la exactitud a un 93.65% teniendo los siguientes valores:

- Entrada y salida de la capa intermedia: 512 -> 2048
- Learning rate: 0.017035007776142457
- Batch size 64

Procederemos a obtener la matriz de confusión para examinarlo más a fondo.

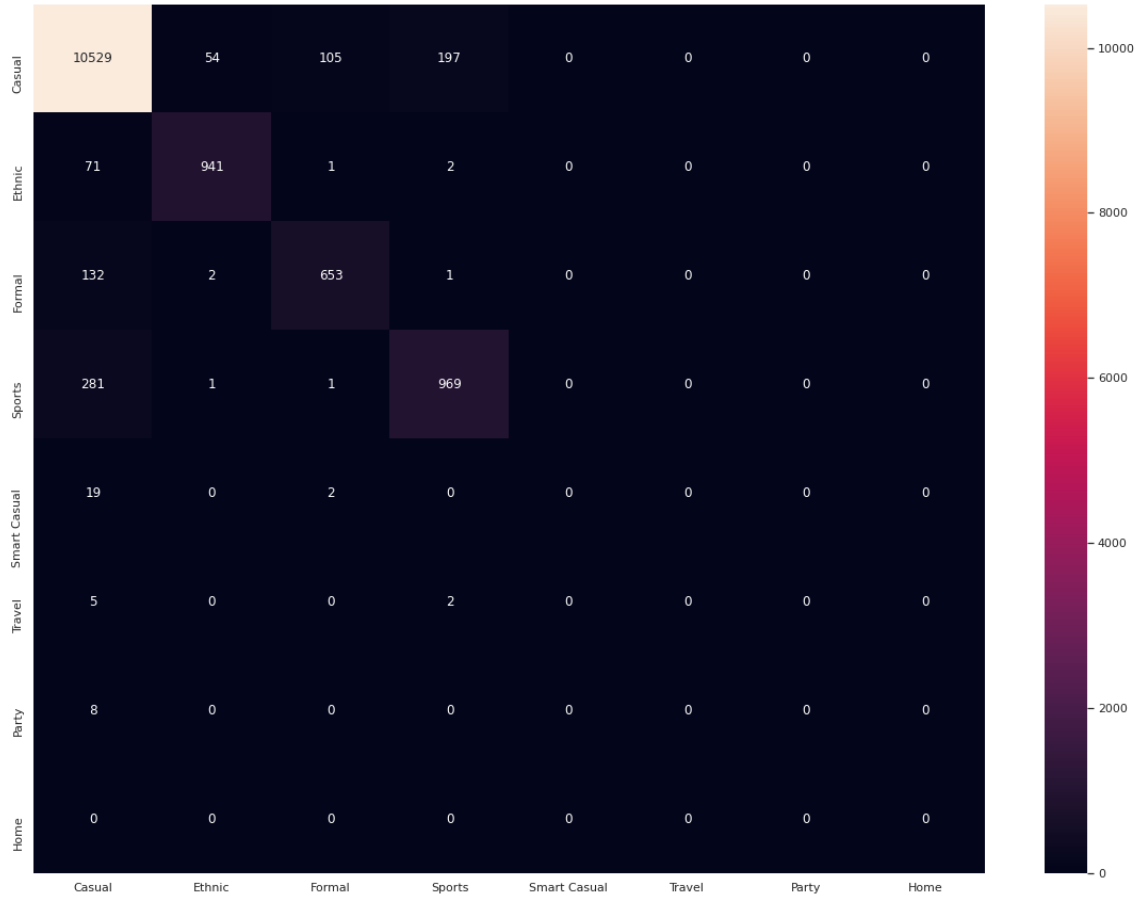


Ilustración 66 Matriz de confusión de Usage

La matriz de confusión en este caso nos muestra como las clases con un número de elementos aceptables obtienen una buena clasificación. Se notan los fallos principalmente al compararlos con Casual que cuenta con la mayor cantidad de elementos en todo el dataset. El estilo étnico es el más diferenciable siendo con el que menos confunde la red con «casual», y prácticamente no confundiéndola con el resto. «Formal» es más confundible pero no tanto como «sports», que siendo el segundo que más elementos presenta es también el que menos diferencia la red al compararlo con el primero. El resto de las clases, debido a su poca cantidad de elementos caen en su gran mayoría en casual. Afortunadamente, debido al uso que representan estas últimas, podrían ser clasificadas como cualquiera de los 4 más prominentes («Smart casual» de hecho se ve que se confunde con «casual» y con «formal», como su nombre indica).

En conclusión, si reducimos los resultados a las 4 categorías prioritarias, podemos estar contentos con lo obtenido.

4.4 Resultados

Finalmente hagamos un resumen de los resultados obtenidos del estudio.

La categoría «MasterCategory» presenta una exactitud extremadamente alta, casi perf ecta (99.77%), teniendo para cada clase una exactitud casi perfecta, exceptuando una sola clase que no llega tan alto pero que es comprensible dado su bajo número de ele mentos. Unos resultados aceptables dado el pequeño número de clases con las que trabaja la red y la clara diferencia que hay entre ellos.

Con la categoría «SubCategory», si bien tenemos unos resultados también excelentes (97.69%), en este caso se empiezan a ver las confusiones que afectarán mayormente luego a «ArticleType». Tiene confusiones entre calzados similares y algunos tipos de vestidos mayoritariamente, pero en general se desenvuelve bien clasificándolos.

En «ArticleType» la clasificación baja hasta un 91.54%, lo cual sigue siendo una considerable exactitud. Ahora con más ejemplos de prendas superiores se ven más fallos de precisión, sobre todo ocasionadas por la clase tops en comparación al resto de prendas de arriba y entre algunos tipos de calzado similares. Ambos se clasifican bien, pero sus fallos siempre tienden a confundirse con estas clases similares. También sufre confusiones con algunos tipos de vestidos largos y otros accesorios similares. El mayor problema de esta categoría recae en el bajo número de elementos de ejemplo que componen a algunas de las clases, que impide que se lleguen a clasificar en absoluto, exceptuando algunos casos de tipos de artículos fácilmente distinguibles. Afortunadamente, muchas de estas clases pertenecían a accesorios más alejados de las prendas de vestir o que podrían haberse combinado en una única clase junto a aquellas con las que se les confunde. Aun con este gran problema, aquellos artículos con una cantidad más adecuada de ejemplos tienden a obtener buenos resultados, demostrando lo capaz que es un modelo de red VGG de clasificar distintas prendas de vestir y accesorios de moda.

«BaseColour» usando redes VGG permite una clasificación ineficiente, consiguiendo solo un 50. Al igual que con otras categorías, en evaluación la clasificación no genera problemas, por lo tanto, sabemos que no se trata de que la red VGG tenga problemas para clasificar los colores, pero no es tan capaz como las otras de transmitirlo. Los colores se confunden fácilmente unos con otros, sobresaliendo con falta de precisión aquellos cuya tonalidad o luminosidad sean similares (distintos tipos de marrones se confunden entre ellos, y el azul marino y negro también). Deducimos que la red acota muy justamente las tonalidades encontradas durante el entrenamiento, provocando que aquellas del grupo de validación los suficientemente “afortunadas” para caer entre sus límites sean clasificadas correctamente. Además, si bien están más uniformemente distribuidos los elementos de sus clases que la anterior categoría, sigue habiendo graves diferencias. Por último, no se han examinado todas las imágenes, pero conociendo lo ambiguo que puede resultar clasificar colores para una persona, no sorprendería que la misma tonalidad pudiera estar en rojo o rosa para dos prendas con la misma tonalidad dependiendo de quién las evaluara, lo cual claramente influye negativamente en la red.

Con «Season» obtenemos una exactitud de un 81.21%, lo cual no está mal, pero no es tan buena como con otras clasificaciones. Individualmente en cuanto a sensibilidad, cada clase clasifica bien más de un 80% exceptuando otoño, que llega a superar por poco el 70%. Al contrario que en otros casos, sus confusiones no son tan esperadas pues si bien se esperaba que verano y primavera tendieran a confundirse, no se

esperaba tanta confusión de todas en general con verano, ya que se creía que tendrían más peso las confusiones entre invierno y otoño, que, si bien no tienen poca, destaca más la anterior. Creemos que pueda tener que ver una gran cantidad de elementos variados bajo Summer que albergue muchos ejemplos similares a los del resto. Cabe decir que, aun así, la exactitud es bastante aceptable en el caso a tratar, dado que la temporada para la que está hecha cada prenda es difícil de distinguir hasta para una persona. También influirán en la confusión aquellos accesorios destinados a una temporada, pero que al no ser tan influyentes como prendas de vestir a la hora de abrigarse o no, estos generen confusión.

«Gender» revela unos resultados excelentes de hasta un 94.51%. Las dos clases principales hombres y mujer generan los mejores resultados para el gran número de elementos que representan, y chica y chico no se le quedan muy atrás. Claramente unisex es la más difícil de clasificar por sus coincidencias con el resto de las clases, pero logra casi un 70% (aunque sospechamos que gran parte de ellos se debe a los accesorios). Podemos deducir que se trata de un modelo que promete con lo especificado, y podemos comprobar como las confusiones que comete, si bien no son pocas, son razonables: confunde principalmente aquellos elementos destinados a la misma edad y mismo sexo (pocas o nulas confusiones se dan entre chica y hombre u chico y mujer).

Finalmente, «Usage» tiene genera también unos buenos resultados con un 93.65% de exactitud, aunque viéndolo bien, esto se debe a hacer caso omiso a algunas de sus clases. Esto debido a que la clase casual tiene una enorme cantidad de elementos que opacan al resto, de tal forma que todas acaban tendiendo a confundirse con ella en mayor o menor medida. A pesar de esto, consigue distinguir el resto de ellas entre sí mismas, exceptuando las de menor número de elementos, los cuales debido a su poca cantidad no llegan a clasificarse de forma exacta en ningún caso, quedando casi todas las de sus elementos bajo casual. Esto tendría más peso si no fuera porque exceptuando «party», que puede distinguirse, «sports» podría juntarse con «travel», y «smart-casual» queda exactamente como lo que su nombre indica. Dado el conjunto tratado no podemos asumir con total seguridad que clasifique bien, pero viendo los casos de las 4 clases de mayor importancia tendemos a creer que sí.

Con esto finalmente tenemos un estudio realizado sobre múltiples características de los elementos de moda presentes en las tiendas, y además algunos clasificadores que pueden destacar para su posterior uso.

Capítulo 5

Distribución temporal del proyecto

En base a la distribución temporal que se había desarrollado para el documento TFT01 se ha ido elaborando a lo largo del desarrollo del trabajo un seguimiento del tiempo dedicado a cada uno de los apartados que se han dedicado a su elaboración.

Como se mencionó, se ha estaba trabajando usando dos tipos de iteraciones:

- Iteraciones semanales: Cada semana durante el curso (salvo contadas excepciones) se realizaban reuniones de seguimiento entre alumno y profesor para comentar los progresos hechos y plantear las tareas a realizar para la siguiente semana.
- Iteraciones temáticas: Dado que se estuvo tratando con varias características para un mismo dataset, se realizaron iteraciones para cada una de estas, tratando los mismos apartados y expandiéndolos en caso de no obtener los resultados deseados con estos.

En cuanto a su coincidencia con lo especificado en el TFT01 aclaramos ante duda que la duración estimada para cada fase varió ligeramente en comparación a lo pensado, pero el número de horas no fue alterado. La extensión de tiempo desde su propuesta hasta la entrega del proyecto se vería retrasada debido a la realización de estudios y actividades externas durante su desarrollo. Este plan se trata claramente de una aproximación al tiempo que realmente se le ha dedicado basándonos en el trabajo semanal.

Fases	***Duración Estimada*** ***(horas)***	***Tareas*** ***(nombre y descripción, obligatorio al menos una por fase)***
Estudio previo / Análisis	50	Tarea 1.1: Estudio de las redes neuronales convolutivas.
		Tarea 1.2: Aprendizaje del uso de PyTorch.
		Tarea 1.3: Búsqueda y estudio del dataset.
Diseño / Desarrollo / Implementación	110	Tarea 2.1: Desarrollo redes neuronales.
		Tarea 2.2: Entrenamiento de redes neuronales.
		Tarea 2.3: Optimización de hiperparámetros.
Evaluación / Validación / Prueba	100	Tarea 3.1: Comprobación de la validez de la red convolutiva y sus resultados.
		Tarea 3.2: Desarrollo de gráficos y estudios posteriores.
Documentación / Presentación	40	Tarea 4.1: Documentación del código y exposición en github.
		Tarea 4.2: Redacción de la memoria del TFT.

		Tarea 4.3: Creación de las tablas y diseño de imágenes.
		Tarea 4.4: Preparación para la presentación del TFT.

Ilustración 67 Distribución temporal del proyecto

Capítulo 6

Conclusiones y trabajo futuro

Durante la duración de este proyecto se ha estado en todo momento tratándose como una investigación profesional. Se han realizado los distintos pasos que compondrían cualquier estudio en cuanto a inteligencia artificial: la propuesta de un tema a tratar, el estudio de las tecnologías y técnicas vigentes para su uso, la búsqueda de un conjunto de datos que satisficiera nuestras necesidades y su análisis, la implementación de las redes neuronales adecuadas, la optimización de sus resultados y finalmente la muestra de estos en un documento escrito. Todo esto llevó a unos resultados que sin ser los mejores, resultan aceptables para los casos que estamos tratando (exceptuando el color base) y prueban la validez de este proyecto como investigación y como punto de inicia para futuras investigaciones o mejoras.

6.1 Aspectos a mejorar / Trabajo a futuro

A lo que hemos llegado actualmente ha sido a una escueta investigación acerca de las posibilidades de clasificar prendas de vestir y demás elementos de moda únicamente a partir de sus imágenes usando redes neuronales.

Debido al corto tiempo del que se dispone para realizar el trabajo de fin de grado no se puedo tomar el riesgo de crear nuestro propio dataset con todas las categorías que en un principio se tenían en mente, y en vez de eso se eligió un dataset con una buena cantidad de características disponibles (categoría, subcategoría, tipo de artículo, género, etc.) que si bien dista un poco de la idea inicial que se tenía del proyecto, suponen un buen acercamiento. Para un posterior estudio, se agradecería la creación de un dataset mucho más completo incluyendo cada una de las características específicas de su tipo de artículo. Disponer de si un elemento de tipo tiene un tipo de cuello o una longitud de mangas específicas resultaría mucho más interesante comercialmente que solo saber que es una camiseta y el género y estación que le corresponden. Véase, el primer caso de uso si se buscara mejorar este proyecto sería la creación de un nuevo dataset o modificación de datos del actual de forma que para cada tipo de artículo (o al menos subtipo) dispusiéramos de un abanico más amplio de características específicas de cada uno de ellos que dieran más juego a la hora de extraer datos de ropa en imágenes.

Combinando el estudio realizado con el que se realizaría a futuro con el dataset ampliado, podríamos llegar a estudiar la creación de una red neuronal capaz de clasificar a partir de un tipo las distintas características que lo forman por separado, como en el caso de las camisas el tipo de cuello, tamaño de mangas, si tiene bolsillos, si tiene botones etc, para lograr individualmente por cada tipo la mayor cantidad de características posible que nos permitan los modelos entrenados. Con el estudio hecho hasta el momento tendríamos para permitir rellenar a partir de una imagen con una exactitud adecuada los campos: categoría maestra, subcategoría, tipo de artículo y género, y con resultados aceptables temporada y uso.

Esto último implicaría también la investigación de su rendimiento usando otras arquitecturas como las comentadas de forma general junto a la VGG como estado del arte en la sección de bases teóricas. Es bastante posible que alguna de ellas pueda lograr mejores valores. Así mismo, no se descartaría el uso de técnicas que por el tiempo limitado y el conjunto de elementos tratados, no fueron elegidas y que podrían aportar a un mejor rendimiento de los modelos.

Teniendo en cuenta la velocidad a la que puede luego obtener las características a partir de una imagen, todo uso posterior que se hiciera con este proyecto podría dar a lugar a varias aplicaciones que a mi parecer serían de gran interés en el mundo de la moda. Dos ejemplos que se me ocurren desde el punto de vista de una marca de moda serían:

- De cara al cliente de la marca: Una aplicación en la cual el cliente a partir de una prenda de ropa u objeto que pudiera vender la marca pudiera sacar una foto del objeto que sería clasificada por nuestras redes neuronales y que llevaran al usuario a elementos similares en los cuales pudiera estar interesado.
- De cara al dependiente de la marca: Una máquina que ante la entrada de nuevos productos que meter en la base de datos o a la hora de enviar la ropa a distintas secciones, agilizara el proceso clasificando al momento mediante imágenes las prendas que le van llegando.

6.2 Conclusión personal

En lo que a experiencia personal respecta, si bien había trabajado anteriormente con inteligencia artificial, se trata de la primera vez en la que se ha tenido que realizar una investigación de principio a fin con varias tecnologías cuyo funcionamiento desconocía al momento de empezar el trabajo. Si bien arduo, ha sido una experiencia gratificante en la cual he podido profundizar más en un campo tan importante actualmente como lo es la inteligencia artificial, expandiendo mis conocimientos de ella que han calado más profundo en mí que si se hubiera tratado de cualquier otro trabajo de una asignatura. El hecho de realizarlo además de la misma forma en la que se realizaría en el mundo laboral me da aún más confianza en mi forma de trabajar y en los resultados que he logrado. Estoy agradecido con mi tutor por haberme dado la oportunidad de haber trabajado bajo su tutela y espero lograr mucho en el futuro con los conocimientos que me ha aportado en estos últimos meses.

Anexos

Repositorio con la versión limpia y resumida del código desarrollado:

https://github.com/VictorHerreraDelgado/FashionClassifier

Referencias / Bibliografía

[1] C. M. Bishop. Neural networks for pattern recognition. Oxford university press, 1995.

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

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

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

[5] 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.

[6] Geoffrey E. Hinton and Nitish Srivastava and Alex Krizhevsky and Ilya Sutskever and Ruslan R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors", arXiv preprint arXiv: 1207.0580, 2012.

[7] Sergey Ioffe and Christian Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift", arXiv preprint arXiv: 1502.03167, 2015.

[8] 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.

[9] 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.

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

[11] 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.

[12] Raymond Perrault, Yoav Shoham, Erik Brynjolfsson, Jack Clark, John Etchemendy, Barbara Grosz, Terah Lyons, James Manyika, Saurabh Mishra, and Juan Carlos Niebles, "The AI Index 2019 Annual Report", AI Index Steering Committee, Human-Centered AI Institute, Stanford University, Stanford, CA, December 2019.

(c) 2019 by Stanford University, "The AI Index 2019 Annual Report" is made available under a Creative Commons AttributionNoDerivatives 4.0 License (International) https://creativecommons.org/licenses/by-nd/4.0/legalcode

The AI Index is as an independent initiative at Stanford University's Human-Centered Artificial Intelligence Institute (HAI).

[13] B. D. Ripley. Pattern recognition and neural networks. Cambridge university press, 1996.

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

[15] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. arXiv:1409.0575, 2014.

[16] K. Simonyan and A. Zisserman. “Very Deep Convolutional Networks for Large-Scale Image Recognition.” arXiv preprint arXiv:1409.1556, 2014.

[17] N. Srivastava and G. Hinton and A. Krizhevsky and I. Sutskever and R. Salakhutdinov, “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”, Journal of Machine Learning Research,2014, Volume 15, Pages 1929-1958

[18] 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.

[19] Christian Szegedy and Wei Liu and Yangqing Jia and Pierre Sermanet and Scott Reed and Dragomir Anguelov and Dumitru Erhan and Vincent Vanhoucke and Andrew Rabinovich, “Going Deeper with Convolutions”, arXiv preprint arXiv: 1409.4842, 2014

[20] M. J. A. van Wezel and L. J. Hamburger and Y. Napolean, “Fine-grained Classification of Rowing teams”, arXiv preprint arXiv: 1912.05393, 2019.

[21] Matthew D Zeiler and Rob Fergus, “Visualizing and Understanding Convolutional Networks”, arXiv preprint arXiv: 1311.2901, 2013.

Param Aggarwal, “Fashion Product Images Dataset”, Kaggle, June 2019. https://www.kaggle.com/paramaggarwal/fashion-product-images-dataset