UNIVERSIDAD DE GRANADA

Departamento de Ciencias de la
Computación e Inteligencia Artificial

Inteligencia de Negocio

Problema 4:
Fashion MNIST

Curso 2020-2021

Cuarto curso del grado de Ingeniería Informática

Luis Miguel Aguilar González
laguilarg99@correo.ugr.es

Índice

1.- Descripción y análisis del problema

En este problema nos enfrentamos principalmente a lo que podría ser un equivalente a MNIST que trata de clasificar imágenes de dígitos escritos a mano en los números que intentan representar. En este caso sucede igual pero con la ropa, que será clasificada en un total de 10 categorías al igual que ocurre con los dígitos.

Label	Class
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot

La estructura del conjunto de datos es igual que la de MNIST: 60000 muestras de entrenamiento y 10000 de test. Cada muestra cómo ya sabemos, es una muestra de 28x28 en escala de grises.

Cuando nos enfrentamos a un problema con la complejidad y dimensión que tiene Fashion MNIST, no solo es necesario establecer un algoritmo que resuelva este problema de clasificación de la forma más óptima posible, si no que es importante también disponer del hardware oportuno para todos los cálculos necesarios. Por eso antes de entrar en analizar el problema quiero introducir las especificaciones de los equipos utilizados y los fundamentos técnicos que hacen a estos poco óptimos para este tipo de tareas.

CUDA son las siglas en inglés para Compute Unified Device Architecture que hace referencia a una plataforma de computación en paralelo incluyendo un compilador y un conjunto de herramientas de desarrollo. Para poder aprovechar la potencia que nos ofrece esta tecnología son necesarios los denominados CUDA cores, una tecnología desarrollada por NVIDIA y que sólo poseen sus GPU.

Poseer una GPU de dicha empresa sería clave en el desarrollo del proyecto pues el tiempo que tarda nuestro programa se reduciría drásticamente no solo por la ventaja obvia que ofrece la ejecución de los procesos en paralelo, sino, porque la comunicación entre la memoria gráfica (VRAM) y estos procesadores es mucho más rápida que la que pudiera existir entre la CPU y la RAM.

Mis equipos cuentan con las siguientes especificaciones:

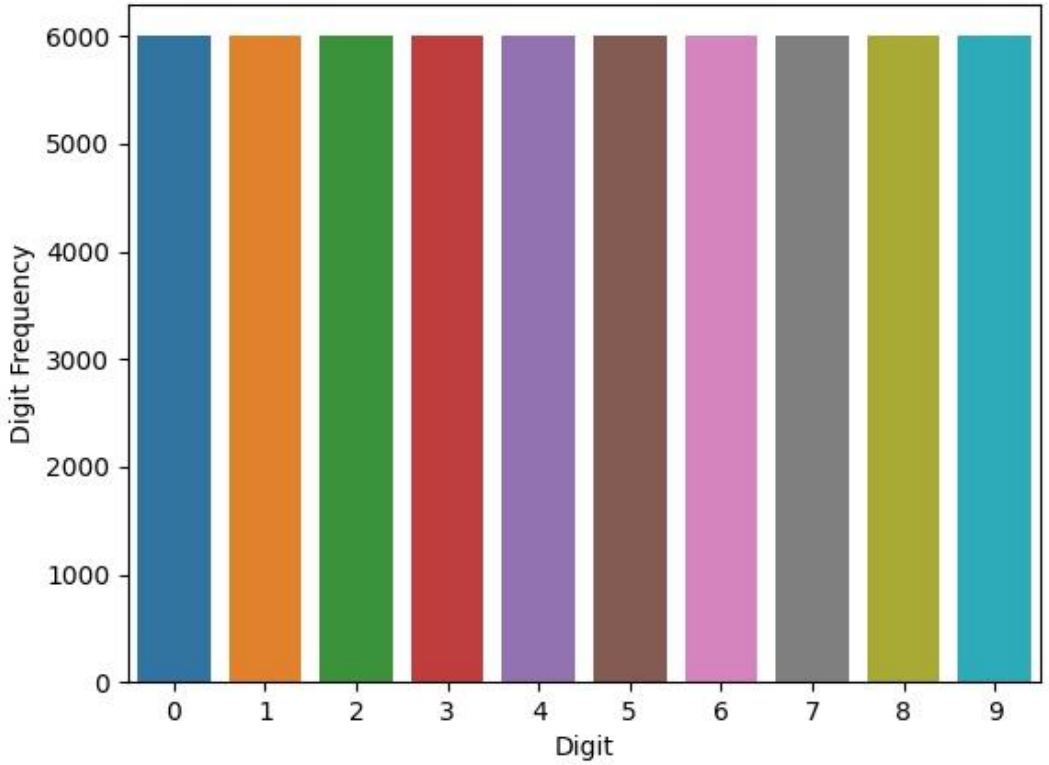
1. Portatil Asus gl553vd
    a. CPU: i7-7700HQ 2’80 GHz 4 Núcleos 8 Hilos
    b. GPU: GTX 1050 mobile 2GB VRAM
    c. RAM: 8GB 2400MHz
    d. 120GB SSD
2. Sobremesa AMD
    a. CPU: Ryzen 3 3300X 3’8GHz 4 Núcleos 8 Hilos
    b. GPU: RX 5700 8GB VRAM
    c. RAM: 16GB 3000 MHz
    d. 480GB SSD

Muestro ambos porque cada uno ha formado parte del proyecto en distintos momentos de su desarrollo. Comencé usando el primero para poder aprovechar su tarjeta gráfica para la ejecución del programa como ya he comentado antes, sin embargo, no conseguí configurar adecuadamente los drivers de la gráfica con la versión de Tensorflow y las librerías CUDA por lo que tuve que desistir de su uso, centrando toda la ejecución en el procesador, este cuenta con una tecnología de INTEL denominada Turbo Boost 2.0 con la cual acelera el desempeño del procesador en función de la carga del sistema en ese momento, llegando de los 2’8GHz especificados de frecuencia base, a los 3`8GHz que soporta como carga máxima de forma estable sin llegar a sufrir Thermal Throttling, este efecto reduce la potencia en el procesador por las altas temperaturas que se pudieran producir dada la mala refrigeración del portátil.

El portátil funcionó para la primera fase de desarrollo (resolver el problema con algoritmos clásicos propios de Machine Learning), sin embargo al introducir Deep Learning a la ecuación, como la red neuronal debe estar cargada en memoria (al igual que los datos) y la gestión de la misma no fue la más óptima, la memoria principal no era suficiente incluyendo procesos ejecutados en un segundo plano así que me pasé a mi equipo habitual que cuenta con 16GB de RAM. Viendo que la frecuencia base era igual que la máxima del portátil (y con los mismos núcleos), decidí hacer overclocking del procesador (aumentando el voltaje) en este segundo equipo para así mejorar el rendimiento, alcanzando los 4’30GHz durante la ejecución manteniendo buenas temperaturas.

En la siguiente imagen se puede observar el consumo que realiza el programa en su pico de memoria:

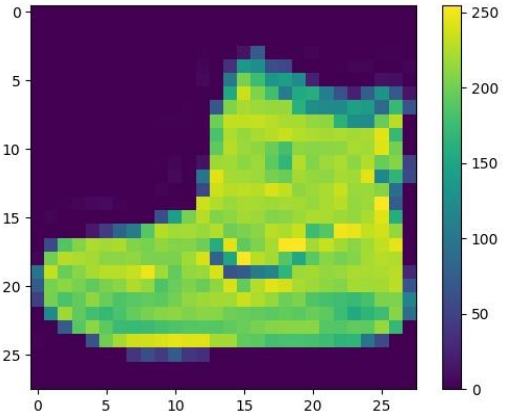
En otros problemas planteados durante la asignatura, el preprocesamiento de los datos era crucial en los resultados finales obtenidos pues nos permitía corregir outliers del conjunto que alteraban las decisiones tomadas por el programa o codificarlos de la forma adecuada para que tuviese una mayor precisión, sin embargo, en esta ocasión el conjunto de datos es correcto, pues existe el mismo número de muestras para cada etiqueta existente como se puede observar a continuación (en la gráfica me refiero a la ropa como dígitos porque ya está codificada), esto es un indicativo de que el conjunto está perfectamente preparado para resolver el problema, además que al representar muestras de forma aleatoria todas ellas tienen su etiqueta correspondiente sin ningún error..



Por otro lado, ha sido necesario adaptar las imágenes de tal modo que puedan ser procesadas por cada uno de los algoritmos utilizados.
Para cargar dichos datos he usado la propia librería keras de la siguiente forma:

```
fashion_mnist = keras.datasets.fashion_mnist
(train_images, train_labels), (test_images, test_labels) =
fashion_mnist.load_data()
```

Una vez cargados, decidí representar algunas de las muestras (como he comentado antes) con sus etiquetas para comprobar que eran los correctos:



Para todos los algoritmos dividí las imágenes entre 255, de esta forma el ordenador solo tendría que hacer cálculos con números entre 0 y 1 que les resultan más fáciles de procesar.
Para Random Forest y el modelo de red neuronal usé la siguiente operación:

```
train_images = train_images / 255.0

test_images = test_images / 255.0
```

Sin embargo, para la red neuronal pre-entrenada aproveché el preprocesado que esta necesita (InceptionV3 solo acepta imágenes con un tamaño mínimo de 75x75) para cambiar no solo el tamaño, si no también la escala de las mismas.

```
image_gen = ImageDataGenerator(rescale=1./255,
                               width_shift_range=0.2,
                               height_shift_range=0.2,
                               validation_split=0.2)
```

Para la red neuronal simple, la división primera fue suficiente para que pudiese procesar las imágenes. En el caso del algoritmo clásico de machine learning fue necesario aplicar la función flatten() de forma que la dimensión de los arrays de datos pasase de 3 a 2.

```
train_images1 = train_images.flatten().reshape(60000, 784)
```

784 pues es el resultado de 28*28.
Por otra parte para que la red neuronal InceptionV3 admita los datos, será necesario no sólo ampliar las imágenes a 75*75, además habrá que dividirlas en 3 canales, es decir, en lugar de almacenar un solo byte por píxel, tendremos 3, para cada uno de los colores principales RGB (Red Green Blue) quedando nuestro array de datos (train) con la siguiente forma (60000, 75,75,3).

Este último preprocesado lo he realizado de la siguiente forma haciendo uso de una librería específica para imágenes que se incorpora a python de la forma habitual.

```
def change_size(image):
    img = array_to_img(image, scale=False) #returns PIL Image
    img = img.resize((75, 75)) #resize image
    img = img.convert(mode='RGB') #makes 3 channels
    arr = img_to_array(img) #convert back to array
    return arr.astype(np.float64)

train_arr = np.array(train_images).reshape(-1, 28, 28, 1)
train_i = [change_size(img) for img in train_arr]
del train_arr
train_i = np.array(train_i)

train_l = tf.keras.utils.to_categorical(train_labels)
```

Además las etiquetas del conjunto tendremos que transformarlas en categóricas, es decir en lugar de codificarlas como un dígito, les daremos una carácter binario convirtiendo cada una de las etiquetas que puede recibir una entrada del conjunto en 10 columnas en las que solo será 1 en aquella celda que representa la categoría a la que pertenece, siendo 0 el resto.

2.- Descripción de los algoritmos

2.1.- Random Forest

Modelo basado en árboles de decisión, el método genera el árbol de clasificación como un conjunto de árboles de decisión generados a partir de tomar varias muestras de forma aleatoria del conjunto total de datos, con el fin de mejorar la precisión y controlar el sobreajuste. La fortaleza de este método reside en que cada árbol protege al otro de sus errores individuales, funciona muy bien con grandes cantidades de datos, como podría ser nuestro caso, además a lo largo de las prácticas realizadas durante el curso se ha demostrado que por lo general es un algoritmo bastante bueno para conseguir altos valores de precisión.
La profundidad es un dato relevante a la hora de mejorar la precisión, en otras ocasiones he realizado de forma recurrente el cálculo con un bucle for para hacer tuning de dicho parámetro, sin embargo, en esta ocasión por motivos de tiempo y potencia de cómputo he tenido que prescindir de este cálculo y conformarme con la precisión final.

2.2.- Modelo Secuencial Simple.

Se trata de una primera aproximación a las redes neuronales con el modelo más simple posible, este se conforma de una primera capa Flatten que convierte nuestras imágenes en arrays, de esta forma convertimos nuestros datos en lineales para que puedan ser procesados por la siguiente capa Dense. Sería lo equivalente a hacer numpy.ravel que convierte nuestra matriz de 3-D en un array 1-D.
Luego la capa Dense con activación de tipo ReLu (Rectified Lineal Unit) transforma los valores introducidos anulando los valores negativos y dejando los positivos tal y como entran:

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

Por otro lado, la última capa Dense con activación softmax transforma las salidas a una representación en forma de probabilidades, de tal manera que el sumatorio de todas es 1 y motivo por el cual en la columna Output Shape ha sido especificado por 10 pues es el número de categorías posibles de clasificación.

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

En la siguiente página queda especificada la estructura del modelo tal y como lo muestra la función:

```
print(Secuencial.summary())
```

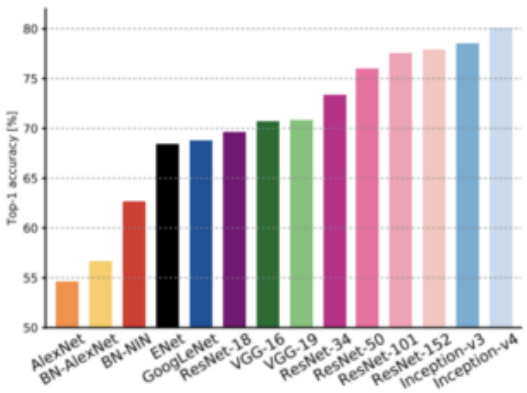
Model: "sequential"

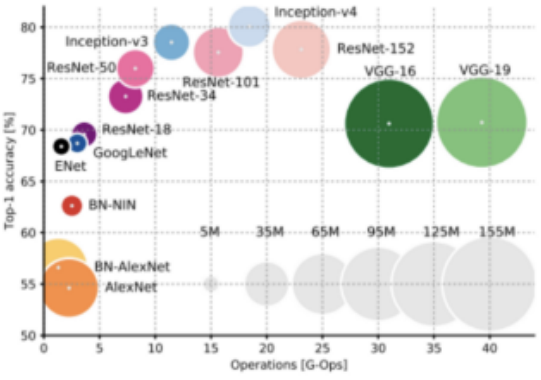
Layer (type)	Output Shape	Param #
flatten_1 (Flatten)	(None, 784)	0
dense_2 (Dense)	(None, 128)	100480
dense_3 (Dense)	(None, 10)	1290

Total params: 101,770
Trainable params: 101,770
Non-trainable params: 0

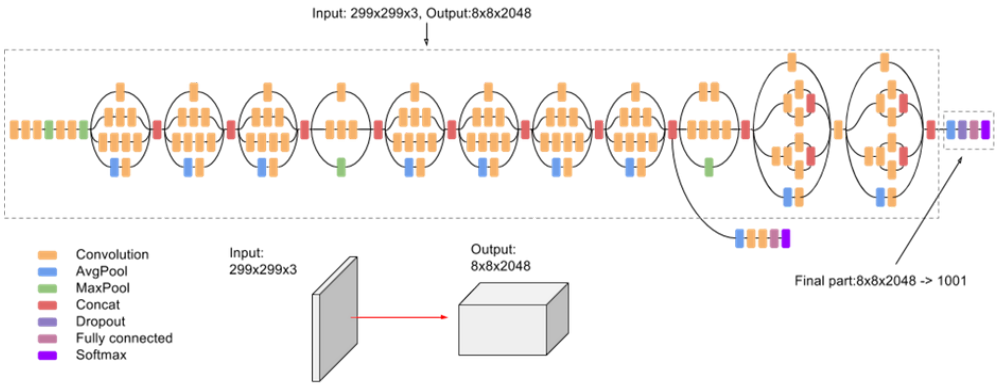
2.3.- Modelo Secuencial. InceptionV3.

Tras investigar las distintas redes pre entrenadas, seleccioné InceptionV3 porque es de las que mejores precisiones consigue y que tiene soporte con Keras. A la hora de añadir esta como una capa más específico que los pesos utilizados son los proporcionados por imagenet que es una de las mayores bases de datos de imágenes organizada según la jerarquía de WordNet (que es a su vez una base de datos de palabras en inglés), en la que cada nodo está representado por cientos y miles de imágenes.





A continuación una imagen de la especificación oficial de Google sobre la red neuronal InceptionV3:



Después añadimos 1 capa Flatten y 2 Dense como especificamos en la red neuronal simple para adaptar la salida de la misma a nuestro problema siendo la última capa al igual que en el otro caso activada por la función softmax que devuelve una serie de probabilidades para cada una de las etiquetas posibles siendo la suma igual a 1.

Model: "sequential"

Layer (type)	Output Shape	Param #
inception_v3 (Functional)	(None, 1, 1, 2048)	21802784
flatten (Flatten)	(None, 2048)	0
dense (Dense)	(None, 128)	262272
dense_1 (Dense)	(None, 10)	1290

Total params: 22,066,346
Trainable params: 22,031,914
Non-trainable params: 34,432

Cabe destacar el aumento del número de parámetros, el primer modelo de red neuronal tiene un tiempo de ejecución de la función fit aproximadamente de 20s, mientras que este segundo modelo con la red pre entrenada llega a tardar un total de 53 minutos aproximadamente.

3.- Estudio experimental

En esta sección estudiaremos principalmente los resultados de precisión de los distintos algoritmos con validación cruzada, en el caso del último algoritmo (Red neuronales pre-entrenadas) no he realizado la validación cruzada pues dado el número de parámetros a analizar podría llegar a tardar un total de en torno a 4 horas, lo cual sería un problema si tuviese que ejecutar el programa en varias ocasiones dado el tiempo con el que contamos para la realización del proyecto, además InceptionV3 es una red que ha sido testada en varias ocasiones pues ya existen multitud de estudios de la misma que corroboran la bondad de sus resultados.

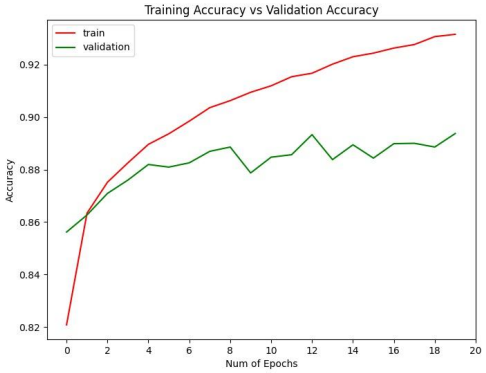
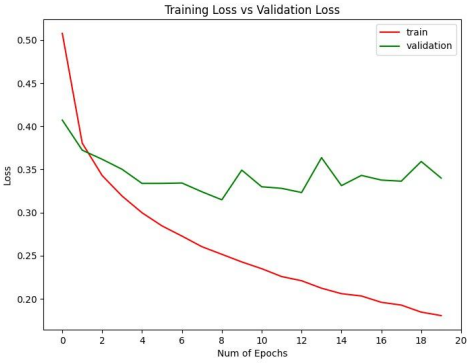
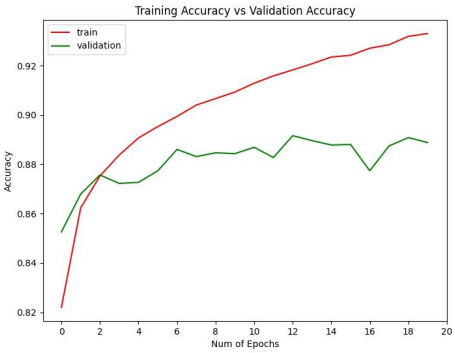
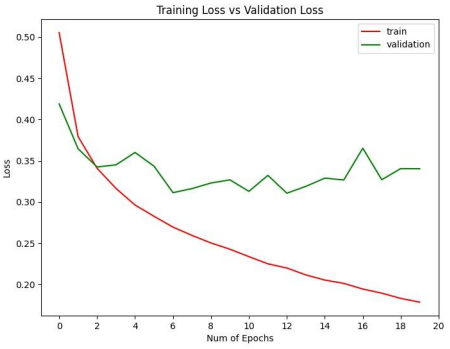
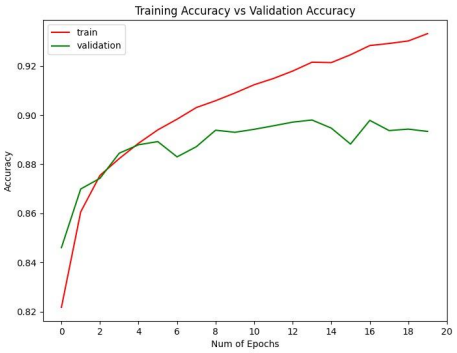
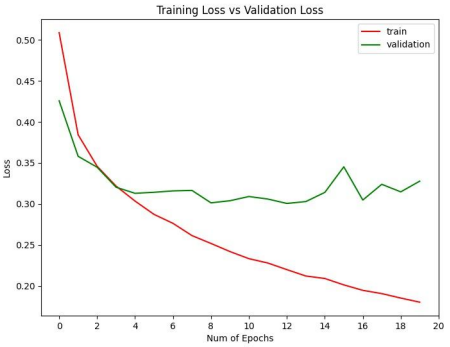
3.1.- Primera aproximación. Machine Learning.

En esta primera aproximación como ya he comentado antes realizada a través de Random Forest se han conseguido resultados bastante buenos aunque peores que los conseguidos en los modelos que veremos a continuación .

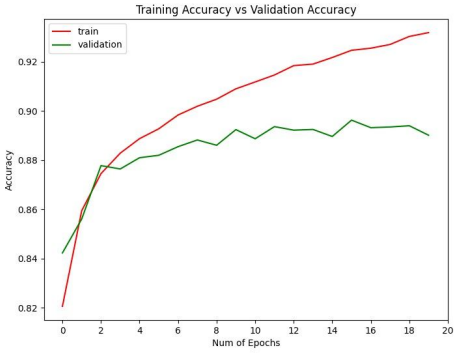
[0.8835 0.87766667 0.88483333 0.88358333 0.88083333]
Random Forest cv_scores mean:0.8820833333333333

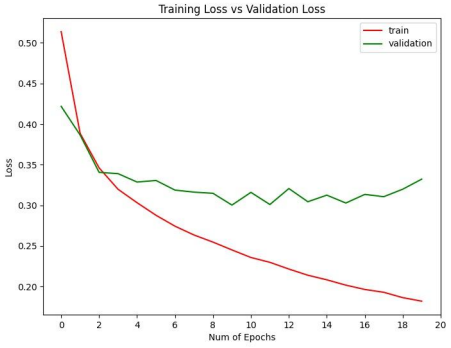
3.2.- Deep Learning.Modelo Secuencial.

En este caso además de la validación cruzada contamos con distintas gráficas que muestran la curva de aprendizaje del modelo (train en rojo y test en verde). Además de la tasa de error, con ambos conjuntos.

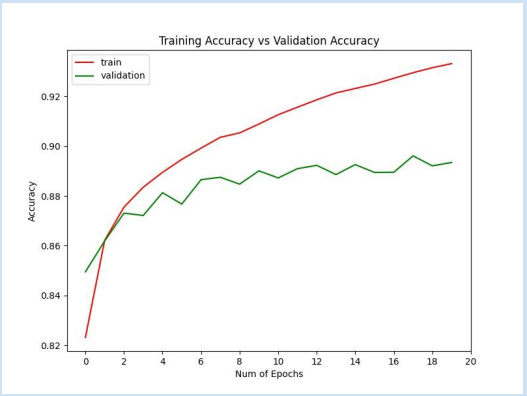
K-Fold	Train [Accuracy]	Loss
1		
2		
3		

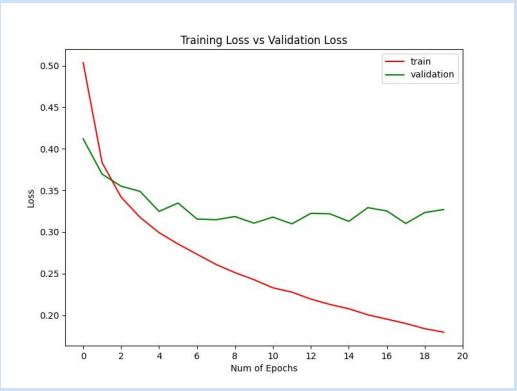
4





5

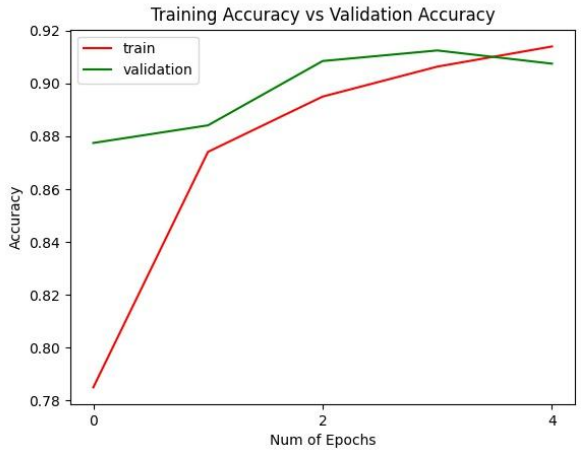


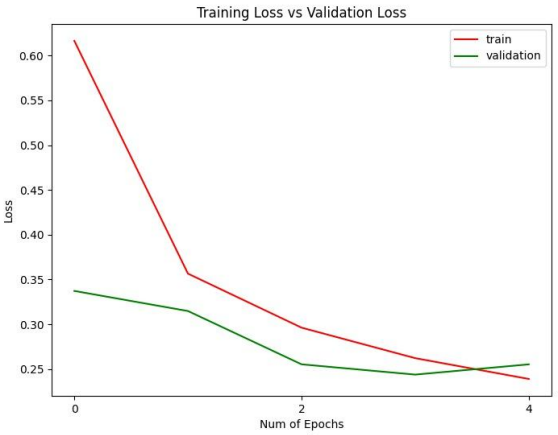


[0.8937143087387085, 0.888785719871521, 0.8933571577072144, 0.8901428580284119, 0.8934285640716553]
Secuencial Model cv accuracy mean:0.8918857216835022

Como se puede observar este modelo es muy simple, sin embargo, consigue mejores resultados medios que Random Forest cuyo tiempo de ejecución es incluso mucho mayor.

3.3.- Uso de redes pre-entrenadas





Accuracy: 0.9149 Val_accuracy: 0.9075

Sin duda el modelo que mejor precisión consigue y que mayor validación obtiene por la comunidad científica en numerosos estudios, sin embargo, habría que valorar si interesa usar dado nuestro caso una red neuronal más simple aún perdiendo algo de precisión.

4.- Planteamiento de futuro

Lo primero que haría sin duda es intentar configurar adecuadamente la gráfica de mi portátil para comprobar si con esa potencia de cómputo entraríamos dentro de un tiempo de ejecución de la función fit coherente para llevar a cabo la investigación de forma adecuada a lo largo de su desarrollo, el problema no implica una complejidad suficiente como para que un equipo con unas características técnicas medias respecto a la tecnología actual a nivel profesional pueda conseguir tiempos lo suficientemente bajos como para ejecutar los algoritmos que puedan ser propuestos a lo largo del proyecto con gran facilidad.

Una vez acomodado nuestro entorno de trabajo a nivel hardware, podemos optimizar la gestión de memoria que realiza nuestro programa para que no ocurra como he especificado al comienzo del trabajo, eliminando aquellos arrays y variables que no sean necesarias en cada momento.

El fichero de datos y el modelo deben estar cargados en memoria para su ejecución, por eso podrían investigarse no solo la correcta gestión de la misma, también formas de reducir el tamaño de estos como es el caso de Tensorflow lite que permite adaptar modelos desarrollados con sus librerías a dispositivos de bajo rendimiento como pueden ser teléfonos móviles o dispositivos IoT. Podrían mirarse también si otros lenguajes de programación como puede ser C++ que tienen soporte para la API de Tensorflow conseguirían mejores valores en cuanto a la gestión de memoria y a tiempos de ejecución se refiere.

Una vez descubierto el mejor entorno de desarrollo posible para nuestro problema sin recurrir a un gran presupuesto que nos proporcione unas características que no llegaríamos a aprovechar y con un entorno software que nos ayude a aprovechar la mismas al máximo, podríamos comenzar a investigar la combinación de distintas metodologías para abordar el problema.

Las razones por las que se propuso Fashion MNIST en un origen, fue porque MNIST es fácil, está muy usado y no es capaz de representar la complejidad de los problemas actuales, sin embargo, las técnicas usadas para MNIST (un problema con un largo recorrido de estudio) adaptadas para Fashion MNIST son también efectivas y no se ha tardado mucho en conseguir valores de precisión bastante altos (0.967 el mejor en la página de github de zalando). Es difícil en un rango de 6 meses mejorar la precisión del modelo actual siendo una sola persona, además con gente que cuenta con años de experiencia en el sector. Dicho esto podemos establecer dos objetivos y abordar uno de ellos, por un lado pretendemos mejorar la eficiencia del modelo con buenos valores de precisión (para ser ejecutados en equipos de bajo rendimiento p.e.) y el otro es formar un equipo de trabajo para mejorar los valores de precisión actuales.

El primero se puede desarrollar prolongando la investigación sobre tecnologías antes mencionadas como Tensorflow Lite y lenguajes de programación además del mejor aprovechamiento de la tecnología CUDA posible.

El segundo objetivo se puede desarrollar de la siguiente forma, dividiría el equipo de trabajo en dos partes coordinadas que trabajarían de forma periódica por objetivos propuestos en reuniones de equipo con la misma periodicidad. La primera parte del equipo trabajaría en el preprocesamiento de los datos para comprobar que la base de datos es correcta y mejorar lo máximo posible de alguna forma las imágenes (en relación a su dimensión y datos que aporta cada una) para adaptarlas a los modelos propuestos y coordinados en la reunión de objetivos por parte del segundo equipo que sería el que guiaría el ritmo de desarrollo.

Para un trabajo como este sólo será necesario un equipo de pocas personas, además de que cuantas menos personas integren los equipos mayor coordinación podrá haber entre ambos por los posibles cambios que pudiera haber entre una reunión y otra, además de que el equipo Hardware necesario sería de menor tamaño cubriendo aun así todos los procesos que pudieran ser ejecutados en su mayor momento de carga de trabajo. Por otro lado, si tuviese que encarar yo solo este segundo objetivo comenzaría simulando los mejores modelos actuales y comprender en profundidad que ha llevado a los creadores de los mismos a tomar cada una de las decisiones de diseño y a partir de ahí intentaría ver en qué partes se podría mejorar (cómo haría el segundo equipo en el proyecto antes comentado) con un preprocesamiento lo más básico posible, una vez mejorados los valores de precisión, comenzaría a estudiar un preprocesamiento distinto hasta mejorar de nuevo (la misión del primer equipo), así haría un desarrollo escalonado ocupando el papel de un equipo y otro en cada momento en función del avance de cada uno de ellos marcando distintos objetivos .

5.- Bibliografía

https://github.com/zalandoresearch/fashion-mnist
https://www.tensorflow.org/tutorials/keras/classification
https://towardsdatascience.com/transfer-learning-using-pre-trained-alexnet-model-and-fashion-mnist-43898c2966fb
https://towardsdatascience.com/transfer-learning-from-pre-trained-models-f2393f124751#:~:text=A%20pre%2Dtrained%20model%20is,VGG%2C%20Inception%2C%20MobileNet).
https://keras.io/api/applications/
https://wngaw.github.io/transfer-learning-for-image-classification/
http://personal.cimat.mx:8181/~mrivera/cursos/aprendizaje_profundo/preentrenadas/preentrenadas.html
http://www.eamonfleming.com/projects/fashion-mnist.html
https://www.kaggle.com/saumandas/intro-to-transfer-learning-with-mnist
https://github.com/graydenshand/FASHION-MNIST-CLASSIFICATION__Random-Forest-vs-Multi-Layer-Perceptron/blob/master/Fashion_MNIST.py
https://www.kaggle.com/gpreda/cnn-with-tensorflow-keras-for-fashion-mnist
https://www.machinecurve.com/index.php/2020/02/18/how-to-use-k-fold-cross-validation-with-keras/
https://www.diegocalvo.es/funcion-de-activacion-redes-neuronales/
http://www.image-net.org/
https://cloud.google.com/tpu/docs/inception-v3-advanced
https://www.tensorflow.org/lite/convert?hl=es
https://www.tensorflow.org/api_docs/cc