Proyecto de Aprendizaje Profundo

Diseño de redes neuronales y aprendizaje profundo

Adrian Pineda Sánchez A00834710

Profesor:

Dr. Santiago Enrique Conant Pablos

02/12/2023

Índice

Introducción

Figura 1.1 (Representación de la base de datos de Fashion MNIST)

Fashion MNIST es un conjunto de datos estándar en el campo de la visión por computadora y el aprendizaje automático. Esta base de datos ha sido propuesta como un reemplazo directo del conjunto clásico de datos MNIST, diseñado originalmente para evaluar algoritmos de aprendizaje automático. Fashion MNIST consta de un conjunto de entrenamiento con 60,000 ejemplos y un conjunto de pruebas con 10,000 ejemplos. Cada muestra es una imagen de tamaño 28x28 píxeles, representando prendas de vestir, y está asociada con una de 10 clases diferentes. Esta base de datos ha ganado popularidad en la comunidad de investigación y desarrollo de inteligencia artificial como un nuevo desafío para evaluar y comparar modelos de clasificación de imágenes

La situación problema se basa en la solución del problema de clasificación de prendas de vestir de la misma base de datos mediante un algoritmo o modelo utilizado, donde para abordar este reto, optamos por modelos basados en Convolutional Neural Networks (CNN).[1] Estas arquitecturas están especialmente diseñadas para reconocer patrones espaciales en datos visuales como imágenes. Dada la naturaleza de Fashion MNIST, donde las prendas de vestir se presentan

como imágenes, las CNN resultan óptimas para identificar características cruciales, como bordes, texturas y formas, fundamentales para diferenciar entre las distintas categorías de prendas.

Las redes neuronales convolucionales (CNN) han transformado la clasificación de imágenes con sus ventajas únicas, aprendiendo automáticamente características como bordes y texturas, evitando la necesidad de diseñar manualmente esas características. Su capacidad para aplicar filtros a regiones locales de imágenes les otorga invarianza a las traslaciones, permitiéndoles reconocer patrones sin importar su ubicación. Las CNN son arquitecturas fundamentales en el reconocimiento de objetos y la clasificación de imágenes, y han evolucionado para mejorar su precisión y generalización. Esta capacidad les permite aprender patrones en imágenes, audio y datos temporales, facilitando la identificación de clases y categorías. [1]

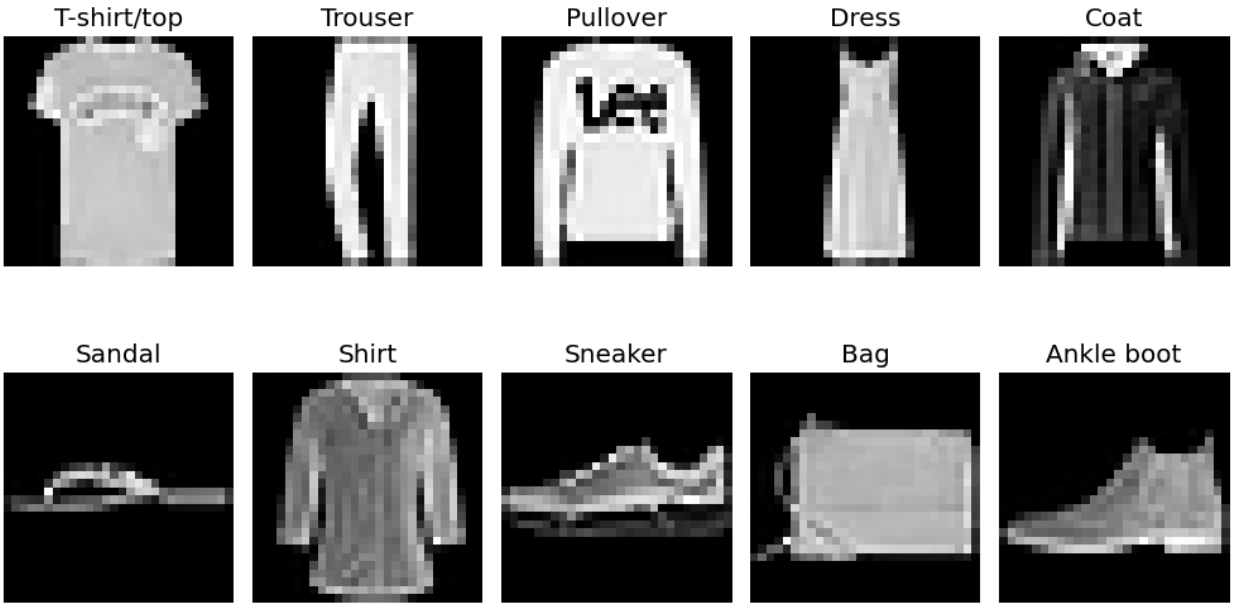


Figura 1.2 (Desplegado de los 10 tipos de clases en la database de Fashion MNIST)

La base de datos Fashion MNIST consta de 10 clases distintas de prendas de vestir. Cada clase representa un tipo específico de prenda, y las imágenes asociadas muestran ejemplos variados de cada tipo. Las clases en esta base de datos son las siguientes: T-shirt/top, Trouser, Pullover, Dress, Coat, Sandal, Shirt: Camisas de manga corta o larga, Sneaker, Bag, Ankle boot.

Modelo Visto en clase: LeNet5

¿Por qué escogiste este modelo?

El modelo de **LeNet5** para la base de datos **Fashion MNIST,** en el cual me base en el tutorial **"LeNet_5_Digits_MNIST.ipynb"**, puede ser una elección sensata en esta situación por varias razones específicas del conjunto de datos y la arquitectura del modelo:

Simplicidad y eficiencia: Fashion MNIST es un conjunto de datos relativamente simple en comparación con otros conjuntos de imágenes más complejos. LeNet, por mi experiencia en clase en relación a las actividades relacionadas con ello, es una arquitectura de red neuronal convolucional (CNN) simple y eficiente, puede ser adecuada para este tipo de conjunto de datos. Su estructura simple puede ser suficiente para capturar las características distintivas de las prendas de vestir en Fashion MNIST.[2]

Tamaño y complejidad de las imágenes: Las imágenes en Fashion MNIST son de baja resolución (28 x 28 píxeles) y en escala de grises. LeNet fue diseñado originalmente para trabajar con imágenes de este tamaño, lo que lo hace apropiado para este conjunto de datos específico.

Parámetros compartidos: El concepto de parámetros compartidos en las capas convolucionales de LeNet puede ser beneficioso para Fashion MNIST, ya que reduce el número total de parámetros entrenables. Dado que Fashion MNIST es un conjunto de datos más pequeño en comparación con otros conjuntos de imágenes, esto puede ayudar a evitar el sobreajuste y mejorar la generalización del modelo.

Rendimiento histórico: LeNet ha demostrado ser efectivo en tareas de clasificación de patrones, especialmente en conjuntos de datos de dígitos manuscritos como MNIST. Dado que Fashion MNIST comparte similitudes en términos de tamaño de imagen y complejidad con MNIST, la elección de LeNet se basa en su rendimiento histórico probado en conjuntos de datos similares.[2]

Pre-procesamiento realizado sobre los datos

- **Carga de datos:** Se utilizan los conjuntos de entrenamiento y prueba de Fashion MNIST provistos por Keras.
- **Normalización de datos:** Las imágenes de Fashion MNIST se escalan para tener valores de píxeles en el rango entre 0 y 1. Este proceso se realiza dividiendo cada valor de píxel (originalmente entre 0 y 255) por 255. Esto ayuda a que el modelo converja más rápido durante el entrenamiento y a reducir el impacto de las diferencias en la escala de los píxeles.
- **Redimensión de imágenes:** LeNet originalmente trabaja con imágenes de tamaño 32x32 píxeles. Por lo tanto, las imágenes de Fashion MNIST, que son de 28x28 píxeles, se redimensionan a 32x32 píxeles mediante relleno con ceros o mediante métodos de interpolación para ajustarlas al tamaño de entrada de LeNet.
- **Ajuste del formato de entrada:** Las imágenes en la base de datos de Fashion MNIST originalmente vienen en un arreglo tridimensional (número de imágenes, altura, ancho), donde el número de imágenes es la cantidad total de muestras. Para LeNet, se ajusta el formato de entrada para tener (número de imágenes, altura, ancho, canales), siendo los canales igual a 1 en caso de imágenes en escala de grises o 3 en caso de imágenes a color (RGB).
- **Codificación one-hot:** Las etiquetas de las clases se convierten a un formato one-hot encoding. Esto convierte las etiquetas enteras (0-9) en vectores binarios donde un valor es 1 en la posición correspondiente a la clase y el resto son 0. Esta codificación es común para problemas de clasificación multi-clase y ayuda al modelo a interpretar las clases como categorías distintas y no ordinales.

Implementación y entrenamiento. Especifica claramente que modificaste del modelo seleccionado para que funcionara de la mejor manera para tus datos.

Este código implementa el modelo LeNet-5 adaptado para la base de datos Fashion MNIST. Con la siguiente arquitectura modificada e ilustrada:

```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_12 (Conv2D)          (None, 28, 28, 6)         156

 max_pooling2d_12 (MaxPooli  (None, 14, 14, 6)         0
 ng2D)

 conv2d_13 (Conv2D)          (None, 10, 10, 16)        2416

 max_pooling2d_13 (MaxPooli  (None, 5, 5, 16)          0
 ng2D)

 flatten_6 (Flatten)         (None, 400)               0

 dense_17 (Dense)            (None, 120)               48120

 dense_18 (Dense)            (None, 84)                10164

 dense_19 (Dense)            (None, 10)                850

=================================================================
Total params: 61706 (241.04 KB)
Trainable params: 61706 (241.04 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Figura 1.3 (Arquitectura del modelo leNet implementado con los datos de la base de fashion MNIST)

- **Capas Convolucionales y de Submuestreo:** El modelo comienza con una capa convolucional (C1) que utiliza filtros de tamaño 5x5 y función de activación tanh. Luego sigue con una capa de submuestreo (S2) utilizando max pooling para reducir las dimensiones de las características.
- **Más capas convolucionales y de submuestreo:** Posteriormente, se agrega otra capa convolucional (C3) y otra capa de submuestreo (S4) con configuraciones similares a las anteriores.
- **Capas Densas:** Después de la capa de submuestreo final, se aplana (Flatten) la salida y se agregan dos capas densas (fully connected) con funciones de activación tanh.
- **Capa de Salida:** La última capa utiliza una función de activación softmax para la clasificación de 10 clases (correspondientes a las categorías de Fashion MNIST).

Compilación, Entrenamiento e Hiperparámetros utilizados:

- **Número de épocas:** Definido como n_epochs = 10, indica la cantidad de veces que el modelo recorrerá todo el conjunto de datos de entrenamiento durante el proceso de entrenamiento.
- **Tamaño del lote:** Establecido como n_batch = 128, indica la cantidad de muestras que se utilizan en cada iteración para actualizar los pesos del modelo durante el entrenamiento.
- **Tasa de aprendizaje (Learning Rate):** Se asume un valor de learning_rate = 0.01, aunque en este caso no se está utilizando explícitamente en el código proporcionado.
- **Optimizador**: Se usa el optimizador Adam para ajustar los pesos del modelo durante el entrenamiento. La línea opt = tf.keras.optimizers.Adam() crea un objeto de optimizador Adam que luego se utiliza en la compilación del modelo.
- **Compilación del modelo:** Se configura el modelo para el entrenamiento con el optimizador Adam, la función de pérdida categorical_crossentropy y se monitorea la métrica de precisión (accuracy).
- **Entrenamiento del modelo:** Se ejecuta el entrenamiento del modelo utilizando los datos de entrenamiento (x_train e y_train) y se valida con los datos de validación (x_test e y_test). El modelo se entrena durante n_epochs épocas, utilizando lotes de tamaño n_batch. Los resultados del entrenamiento se almacenan en el objeto Historia para su análisis posterior.

Evaluación del modelo entrenado. De preferencia, además de las métricas de desempeño, incluye gráficas y tablas de experimentos.

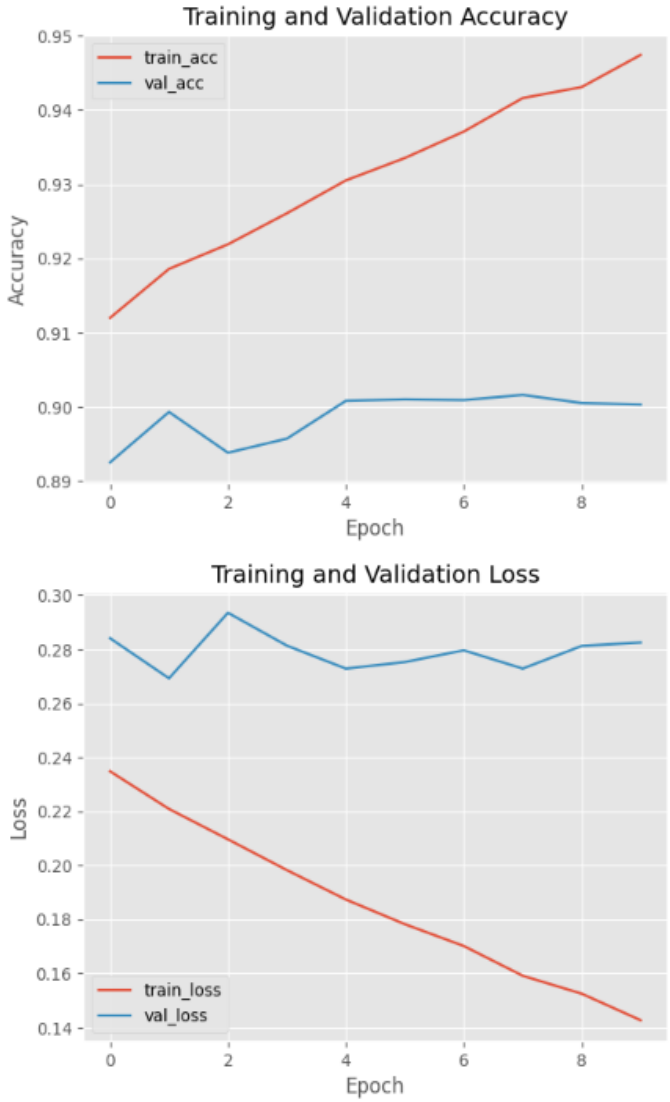
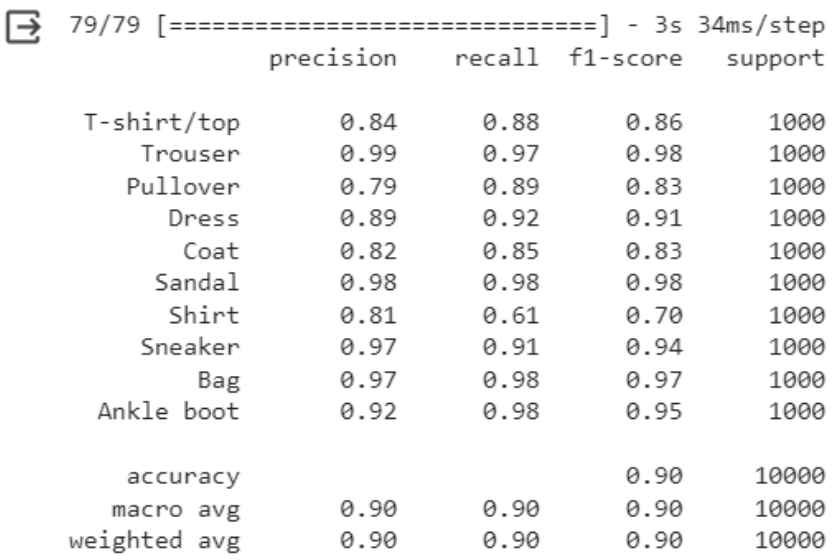


Figura 1.4 (Gráfica de Accuracy y Lost vs Épocas LeNet 10 épocas)

Puede observarse cómo a pesar del transcurrir de las épocas no mejora mucho más el validation accuracy del modelo en comparativa con el training accuracy, pues parece llegar a un tope y estancarse, y la misma situación pasa con la "pérdida", debido a que se estanca y en ese punto no mejora mucho más, sin embargo, sigue siendo muy bueno llegando a aproximadamente un 90%.



```
79/79 [==============================] - 3s 34ms/step
              precision    recall  f1-score   support

 T-shirt/top       0.84      0.88      0.86      1000
     Trouser       0.99      0.97      0.98      1000
    Pullover       0.79      0.89      0.83      1000
       Dress       0.89      0.92      0.91      1000
        Coat       0.82      0.85      0.83      1000
      Sandal       0.98      0.98      0.98      1000
       Shirt       0.81      0.61      0.70      1000
     Sneaker       0.97      0.91      0.94      1000
         Bag       0.97      0.98      0.97      1000
  Ankle boot       0.92      0.98      0.95      1000

    accuracy                           0.90     10000
   macro avg       0.90      0.90      0.90     10000
weighted avg       0.90      0.90      0.90     10000
```

Figura 1.5 (Métricas obtenidas después de la compilación, entrenamiento y evaluación de LeNet)

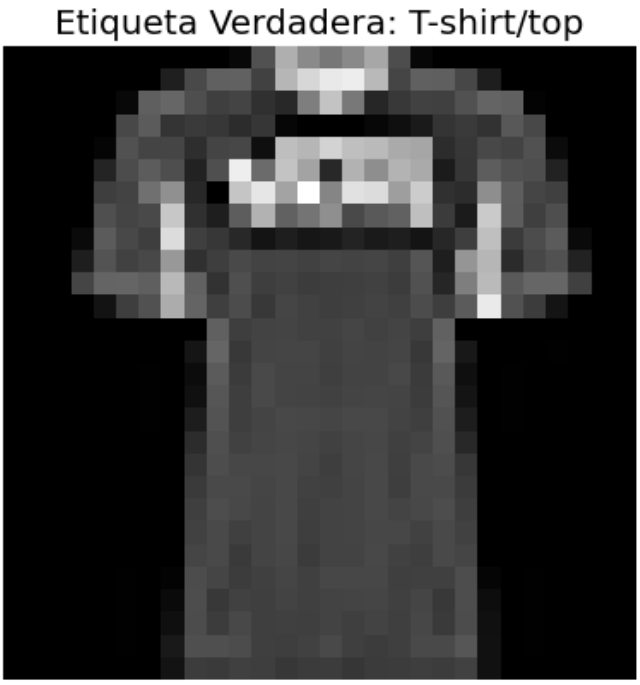
Precisión (Precisión): Representa la proporción de elementos identificados correctamente como pertenecientes a una determinada clase con respecto a todos los elementos identificados en esa clase. Por ejemplo, para la clase 'T-shirt/top', el modelo clasifica correctamente el 84% de los elementos que predijo como 'T-shirt/top'.

Recall (Sensibilidad o Exhaustividad): Es la proporción de elementos de una clase que fueron identificados correctamente por el modelo con respecto a todos los elementos que realmente pertenecen a esa clase. Para 'T-shirt/top', el modelo identifica correctamente el 88% de todas las instancias de 'T-shirt/top' en el conjunto de datos.

F1-Score: Es una métrica que combina precisión y recall en una sola puntuación, proporcionando un equilibrio entre ambas. Es útil cuando las clases están desbalanceadas. Para 'T-shirt/top', el F1-Score es 0.86.

Soporte (Support): Es la cantidad de ocurrencias reales de cada clase en el conjunto de datos. En este caso, todas las clases tienen 1000 muestras cada una.

Exactitud (Accuracy): Representa la proporción de muestras clasificadas correctamente sobre el total de muestras. En el conjunto de prueba, el modelo tiene una precisión general del 90%, lo que significa que clasifica correctamente el 90% de todas las muestras.




```
1/1 [==============================] - 0s 20ms/step
Etiqueta Verdadera: T-shirt/top
T-shirt/top: 0.99919
Trouser: 0.00000
Pullover: 0.00006
Dress: 0.00012
Coat: 0.00000
Sandal: 0.00000
Shirt: 0.00063
Sneaker: 0.00000
Bag: 0.00000
Ankle boot: 0.00000
Predicción del modelo: T-shirt/top
El modelo acertó en su predicción.
```

Figura 1.6 (Evaluación de imagen aleatoria con el modelo de predicción Lenet de fashion MNIST)

Se carga un modelo previamente entrenado en **Fashion MNIST**, selecciona aleatoriamente una imagen de prueba, realiza una predicción con el modelo y muestra la imagen junto con su etiqueta verdadera. Luego, verifica si la predicción del modelo coincide con la etiqueta verdadera y proporciona esta información, como podemos observar, predice de una manera correcta el artículo o prenda de ropa en la mayoría de los casos en torno al accuracy anteriormente obtenido,

donde se evalua en esta caso con **99.919%** para una T-shirt/top y siendo efectivamente la etiqueta correcta una T-shirt/top.

```
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 19ms/step
Total de aciertos en 100 experimentos: 92
```

Figura 1.7 (Evaluación y Predicción correcta con respecto a 100 eventos aleatorios)

Conclusiones del modelo.

- **Particularidades del modelo con el dataset utilizado.**

Me pareció destacable como parece aumentar de forma gradual y unitariamente significativa unas cuantas unidades con el transcurso del tiempo a lo largo de las épocas desde el comienzo, sin embargo, si se instancia con un número de épocas mayor, no parece aumentar de forma tan significativa el accuracy llegando a un punto de estancamiento, y no se compensa en mi opinión en cuestión de eficiencia el aumento en costo computacional con respecto del tiempo destinado al accuracy obtenido.

Ademas me parecio de significancia que LeNet representa un modelo significativamente simple en comparativa con otros modelos, y en cuestión de la selección del mismo dentro de las **redes convolucionales** me parece una opción sumamente acertada debido a sus ventajas en simplicidad y balance entre **precisión y eficiencia.**

- **Comportamiento para predecir la categoría. ¿Funcionó como lo esperabas?**

Obviamente en torno a los valores obtenidos en las métricas anteriores con un **accuracy** cercano a **90%** después del entrenamiento, esperaría que la mayoría de ocasiones, predijera la prenda de forma correcta, y en el ejemplo del reconocimiento de imagen, tuvo un rendimiento muy bueno, realizando varias pruebas aleatorias por mi cuenta y funcionando de una muy buena manera.

Asimismo, en la serie aleatoria de 100 ejemplos creados para observar el rendimiento del modelo, los resultados convergen a un valor cercano del **accuracy** obtenido prediciendo aproximadamente 90 prendas de forma correcta.

Sin duda, en torno a toda la implementación y entrenamiento, así como la evaluación del modelo, se obtuvieron unos excelentes resultados en los que supongo que se debe al buen rendimiento de las **redes neuronales convolucionales**.

Modelo Investigado: ResNet y tutorial de ResNet 50

¿Por qué escogiste este modelo?

Profundidad moderada y eficiencia: La versión simplificada de ResNet que se ha empleado tiene una profundidad óptima, no siendo ni excesivamente profunda ni superficial. Esta característica permite capturar características complejas de las imágenes sin generar una carga computacional excesiva, lo que la hace eficiente para tareas de reconocimiento de moda.[3]

Capacidad para capturar detalles: Con sus capas intermedias, esta versión reducida de ResNet puede capturar y representar detalles significativos en las imágenes de moda, tales como texturas, patrones, colores y estilos, facilitando una comprensión más profunda de los elementos clave en la ropa y accesorios.[3]

Transferencia de aprendizaje: Al ser una variante simplificada de arquitecturas más grandes como ResNet-50 o ResNet-101, esta versión ha sido entrenada en conjuntos de datos extensos y diversos, lo que la hace ideal para la transferencia de aprendizaje. Puede adaptarse bien a

conjuntos de datos específicos de moda con una cantidad moderada de datos de entrenamiento, lo que mejora la precisión del modelo.[3]

Rendimiento general: Esta versión simplificada ofrece un buen equilibrio entre rendimiento y uso de recursos. Proporciona resultados sólidos en tareas de clasificación y detección de objetos, convirtiéndola en una opción confiable para aplicaciones de moda donde la precisión y la capacidad de generalización son fundamentales.[3]

Disponibilidad y soporte: Dado que las arquitecturas ResNet son populares y ampliamente estudiadas en la comunidad de aprendizaje profundo, existe una gran cantidad de recursos, implementaciones pre-entrenadas y documentación disponible. Esto facilita su implementación y ajuste para conjuntos de datos específicos de moda, debido al amplio respaldo y disponibilidad de información.[3]

Pre-procesamiento realizado sobre los datos

- **Cargar los datos:** El conjunto de datos Fashion MNIST se carga en cuatro variables: train_images, train_labels, test_images, y test_labels. Los datos de entrenamiento consisten en imágenes y sus etiquetas correspondientes, mientras que los datos de prueba contienen imágenes separadas para evaluar el rendimiento del modelo.
- **Reformatear las imágenes:** Las imágenes en el conjunto de datos están inicialmente en una forma de matriz bidimensional (28x28 píxeles). Para ser compatibles con modelos de redes neuronales convolucionales (CNN), se reformatean para tener una dimensión adicional que represente los canales de color (en este caso, 1 canal para imágenes en escala de grises). Por lo tanto, las imágenes se reformatean a una forma de (28, 28, 1).
- **Normalización de píxeles:** Los valores de píxeles de las imágenes se escalan para estar en el rango de 0 a 1 dividiendo cada valor por 255. Esto ayuda a que el modelo converja más rápido durante el entrenamiento y a reducir la posibilidad de problemas relacionados con gradientes durante el proceso de optimización.
- **Codificación one-hot de etiquetas**: Las etiquetas originales (números enteros que representan las clases) se transforman en un formato one-hot encoding utilizando to_categorical de Keras. Este paso convierte las etiquetas en vectores binarios donde un

solo elemento tiene valor 1 y el resto es 0, lo que facilita el procesamiento y la interpretación de las etiquetas por parte del modelo durante el entrenamiento.

Implementación y entrenamiento. Especifica claramente que modificaste del modelo seleccionado para que funcionara de la mejor manera para tus datos.

```
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                Output Shape              Param #   Connected to
==================================================================================================
 input_1 (InputLayer)        [(None, 28, 28, 1)]       0         []

 conv2d (Conv2D)             (None, 14, 14, 64)        3200      ['input_1[0][0]']

 batch_normalization (Batch  (None, 14, 14, 64)        256       ['conv2d[0][0]']
 Normalization)

 activation (Activation)     (None, 14, 14, 64)        0         ['batch_normalization[0][0]']

 max_pooling2d (MaxPooling2  (None, 7, 7, 64)          0         ['activation[0][0]']
 D)

 conv2d_1 (Conv2D)           (None, 7, 7, 64)          36928     ['max_pooling2d[0][0]']

 batch_normalization_1 (Bat  (None, 7, 7, 64)          256       ['conv2d_1[0][0]']
 chNormalization)

 activation_1 (Activation)   (None, 7, 7, 64)          0         ['batch_normalization_1[0][0]'
                                                                 ]

 conv2d_2 (Conv2D)           (None, 7, 7, 64)          36928     ['activation_1[0][0]']

 batch_normalization_2 (Bat  (None, 7, 7, 64)          256       ['conv2d_2[0][0]']
 chNormalization)

 add (Add)                   (None, 7, 7, 64)          0         ['batch_normalization_2[0][0]'
                                                                 , 'max_pooling2d[0][0]']

 activation_2 (Activation)   (None, 7, 7, 64)          0         ['add[0][0]']

 conv2d_3 (Conv2D)           (None, 7, 7, 64)          36928     ['activation_2[0][0]']

 batch_normalization_3 (Bat  (None, 7, 7, 64)          256       ['conv2d_3[0][0]']
 chNormalization)

 activation_3 (Activation)   (None, 7, 7, 64)          0         ['batch_normalization_3[0][0]'
                                                                 ]

 conv2d_4 (Conv2D)           (None, 7, 7, 64)          36928     ['activation_3[0][0]']

 batch_normalization_4 (Bat  (None, 7, 7, 64)          256       ['conv2d_4[0][0]']
 chNormalization)

 add_1 (Add)                 (None, 7, 7, 64)          0         ['batch_normalization_4[0][0]'
                                                                 , 'activation_2[0][0]']

 activation_4 (Activation)   (None, 7, 7, 64)          0         ['add_1[0][0]']

 conv2d_5 (Conv2D)           (None, 7, 7, 64)          36928     ['activation_4[0][0]']
```

Figura 1.8 (Arquitectura del modelo ResNet simplificado implementado con los datos de la base de fashion MNIST)

En primera instancia cabe mencionar que este modelo de ResNet de arquitectura más ligera que ResNet 50 o ResNet 100 fue creado para la implementación de este dataset por un servidor a través de documentación encontrada en específico por lo que no fue obtenido específicamente de algún tutorial, sin embargo el código ResNet50 si fue modificado en torno a una serie de hiperparametros distintos y una arquitectura algo diferente de un tutorial consultado pero se verá la comparativa con este modelo actual de ResNet, por el momento el modelo actual cuenta con la siguiente arquitectura:

Capas de Convolución y Normalización:

- Input Layer: Toma imágenes de tamaño (28, 28, 1) como entrada (imágenes en escala de grises).
- Conv2D (64 filtros, kernel 7x7, stride 2, padding same): Realiza una convolución inicial con 64 filtros, un kernel de 7x7, stride de 2 y relleno para mantener el mismo tamaño.
- Batch Normalization y Activación ReLU: Aplicados después de cada capa convolucional para normalizar y activar los mapas de características.

Bloques Residuales:

- La función resnet_block define un bloque residual con dos capas convolucionales, Batch Normalization y activación ReLU, seguido por la suma de la salida de la segunda capa convolucional y la entrada original.
- En el bucle for, se añaden tres bloques residuales al modelo, cada uno aplicando la función resnet_block con 64 filtros.

Capas de Reducción y Clasificación:

- GlobalAveragePooling2D: Reduce el tamaño de los mapas de características a un solo vector promediando todos los valores.
- Dense (10 neuronas, activación softmax): Capa densa de salida con 10 neuronas y activación softmax para clasificar las imágenes en 10 categorías diferentes (en este caso, las clases de Fashion MNIST).

Compilación, Entrenamiento e Hiperparámetros utilizados:

- **Learning Rate:** No se especifica directamente en el código proporcionado. El optimizador Adam predeterminado usa una tasa de aprendizaje predeterminada de 0.001, a menos que se modifique explícitamente.
- **Epochs:** Se configura en 10. Representa el número de veces que el modelo recorrerá todo el conjunto de datos de entrenamiento.
- **Batch Size:** Fijado en 64. Indica el número de muestras que se utilizan en cada iteración de entrenamiento.
- **Optimizer:** Se utiliza 'Adam', un algoritmo de optimización que se adapta automáticamente las tasas de aprendizaje individuales para cada parámetro.
- **Loss Function:** 'Categorical Crossentropy' se emplea como función de pérdida, ideal para problemas de clasificación multiclase.
- **Métricas:** La métrica de evaluación se establece en 'accuracy' para evaluar la precisión del modelo durante el entrenamiento.
- **Compilación:** Se utiliza el optimizador 'adam' con una función de pérdida 'categorical_crossentropy', comúnmente usada para problemas de clasificación multiclase. La métrica utilizada para evaluar el rendimiento durante el entrenamiento es la precisión ('accuracy').
- **Fit:** Entrena el modelo utilizando los datos de entrenamiento (train_images y train_labels) durante 10 épocas, con lotes de 64 imágenes.
- **Validation Data:** Utiliza datos de validación (test_images y test_labels) para evaluar el modelo después de cada época y monitorear su rendimiento en un conjunto de datos no visto. Los resultados se almacenan en la historia.

Variante Tutorial ResNet50 (Variante con tutorial):

¿Por qué escogiste este modelo?

Solamente añadiendo como una comparativa del poder que podría tener los niveles superiores de ResNet en comparativa con la versión que utilizaremos para este reporte pero que no ejerce diferente a un nivel de épocas bajas, ya que en 400 este llega a tener una accuracy cercano a 95%

pero observaremos lo que pasa en esta prueba, justificando porque para este reporte hemos utilizado la otra variante de ResNet y no ResNet50 o superior. Por lo que a diferencia del código anterior, solo se ilustran las diferencias destacables en torno a ambos modelos, ya que el pre-procesamiento de datos sigue un sistema casi equivalente al anterior.

Implementación y entrenamiento. Especifica claramente que modificaste del modelo seleccionado para que funcionara de la mejor manera para tus datos.

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	multiple	3200
batch_normalization (BatchNormalization)	multiple	256
activation (Activation)	multiple	0
max_pooling2d (MaxPooling2D)	multiple	0
residual_block (ResidualBlock)	multiple	75904
residual_block_1 (ResidualBlock)	multiple	71552
conv2d_8 (Conv2D)	multiple	131584
residual_block_2 (ResidualBlock)	multiple	282368
residual_block_3 (ResidualBlock)	multiple	282368
conv2d_15 (Conv2D)	multiple	525312
residual_block_4 (ResidualBlock)	multiple	1121792
residual_block_5 (ResidualBlock)	multiple	1121792
residual_block_6 (ResidualBlock)	multiple	1121792
conv2d_25 (Conv2D)	multiple	2099200
residual_block_7 (ResidualBlock)	multiple	4471808
residual_block_8 (ResidualBlock)	multiple	4471808
global_average_pooling2d (GlobalAveragePooling2D)	multiple	0
dense (Dense)	multiple	2049000
dense_1 (Dense)	multiple	10010

Total params: 17839746 (68.05 MB)
Trainable params: 17813506 (67.95 MB)
Non-trainable params: 26240 (102.50 KB)

Figura 1.9 (Arquitectura del modelo ResNet50 simplificado implementado con los datos de la base de fashion MNIST)

Se realizaron modificaciones en todos los hiperparametros como épocas pasando de 400 a solo 10, batch_size de 128 a 64, así como modificando la arquitectura interna de la red ResNet50 para reducir el número de bloques residuales en varias etapas de la red (marcados con comentarios), manteniendo su estructura original pero disminuyendo la complejidad general del modelo. Esta modificación debería resultar en un modelo ResNet50 más ligero pero aún capaz de capturar características importantes en el conjunto de datos de moda.

Evaluación del modelo entrenado. De preferencia, además de las métricas de desempeño, incluye gráficas y tablas de experimentos.

```
11] Epoch 1, Loss: 0.4720892310142517, Accuracy: 82.25167083740234, Test Loss: (
    Epoch 2, Loss: 0.45608431100845337, Accuracy: 82.87309265136719, Test Loss:
    Epoch 3, Loss: 0.4425831735134125, Accuracy: 83.38812255859375, Test Loss: (
    Epoch 4, Loss: 0.43132296204566956, Accuracy: 83.81555938720703, Test Loss:
    Epoch 5, Loss: 0.42087769508361816, Accuracy: 84.22383117675781, Test Loss:
    Epoch 6, Loss: 0.41194865107536316, Accuracy: 84.56121063232422, Test Loss:
    Epoch 7, Loss: 0.40395256876945496, Accuracy: 84.86499786376953, Test Loss:
    Epoch 8, Loss: 0.39682677388191223, Accuracy: 85.12628173828125, Test Loss:
    Epoch 9, Loss: 0.3901771605014801, Accuracy: 85.38143157958984, Test Loss: (
    Epoch 10, Loss: 0.38419923186302185, Accuracy: 85.60100555419922, Test Loss
```

Train y Test Accurary

```
import matplotlib.pyplot as plt

plt.plot(train_accuracies, label = 'Train Accuracy')
plt.plot(test_accuracies, linestyle = 'dashed', label = 'Test Accuracy')
plt.legend()
plt.show()
```

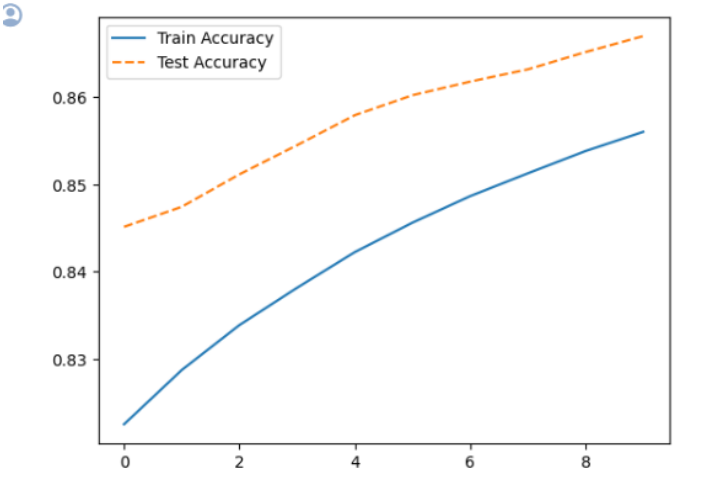


Figura 1.10 (Gráfica de Accuracy y Lost vs Épocas ResNet50 con 10 épocas)

Figura 1.11(Gráfica de Accuracy y Lost vs Épocas ResNet con 10 épocas)

En primera instancia podemos confirmar lo que afirmamos hace un momento y es que con respecto de nuestro modelo de ResNet y ResNet50 a épocas bajas no notamos una diferencia significativa en cuestión de las métricas obtenidas, es mas, nos es inclusive mas conveniente

utilizar ResNet ya que converge a un accuracy promedio de 90%, mientras que el de ResNet50 es de 86% aproximadamente, esa es la justificación por la cual en este reporte continuaremos únicamente en torno al modelo ResNet basico.

Puede observarse a diferencia de **LeNet** como los **training** con los **validation** tanto en **accuracy** como en **loss**, parecen ir acorde y de forma proporcional a como se avanza en el número de épocas cuando estas son 5, sin embargo, puede observarse en comparativa como parece llegar un estancamiento, y no solo eso, sino que parece empeorar conforme al aumento de épocas, compensado y descompensado tanto en el Accuracy así como en el loss conforme avanza a la época 10, llegando a un **90%** en caso de buscar eficiencia, no compensa, sin embargo, debido a que no es un tiempo tan irrazonable podemos dejarlo con 10 épocas, solo siendo conscientes de que el modelo no podrá mejorar mucho más, salvo algunos puntos aumentando considerablemente el número de épocas.

```
313/313 [==============================] - 12s 39ms/step
              precision    recall  f1-score   support

           0       0.87      0.84      0.86      1000
           1       0.95      0.99      0.97      1000
           2       0.82      0.87      0.84      1000
           3       0.93      0.88      0.90      1000
           4       0.91      0.71      0.80      1000
           5       0.97      0.99      0.98      1000
           6       0.68      0.79      0.73      1000
           7       0.98      0.94      0.96      1000
           8       0.97      0.99      0.98      1000
           9       0.96      0.97      0.96      1000

    accuracy                           0.90     10000
   macro avg       0.90      0.90      0.90     10000
weighted avg       0.90      0.90      0.90     10000
```

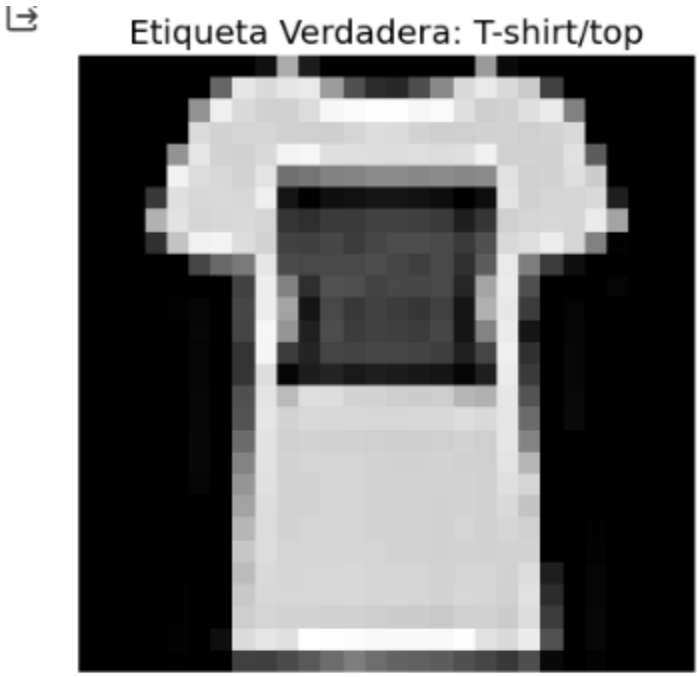
```
# Evaluar el modelo en el conjunto de pruebas
test_loss, test_accuracy = model.evaluate(test_images, test_labels)

print(f"Pérdida en el conjunto de pruebas: {test_loss}")
print(f"Precisión en el conjunto de pruebas: {test_accuracy}")
```

```
313/313 [==============================] - 9s 28ms/step - loss: 0.2949 - accuracy: 0.8981
Pérdida en el conjunto de pruebas: 0.29488739371299744
Precisión en el conjunto de pruebas: 0.8981000185012817
```

Figura 1.12(Métricas obtenidas después de la compilación, entrenamiento y evaluación de ResNet)

De manera similar al caso anterior con LeNet5 se pueden visualizar las métricas correspondientes al modelo ResNet en su accuracy. A continuación aunque la mayoría de ResNet es por medio de colores RGB puede ser adaptada a escala de grises donde podremos realizar una prueba similar a la anterior en comparativa con una prueba de una imagen aleatoriamente tomada de los datos de prueba:




```
1/1 [==============================] - 0s 23ms/step
Etiqueta Verdadera: T-shirt/top
T-shirt/top: 0.99604
Trouser: 0.00000
Pullover: 0.00012
Dress: 0.00014
Coat: 0.00000
Sandal: 0.00000
Shirt: 0.00370
Sneaker: 0.00000
Bag: 0.00000
Ankle boot: 0.00000
Predicción del modelo: T-shirt/top
El modelo acertó en su predicción.
```

Figura 1.13 (Evaluación de imagen aleatoria con el modelo de predicción ResNet de fashion MNIST)

Al igual que el modelo anterior de LeNet se realiza la misma prueba, con el conjunto de datos previamente cargado, de forma aleatoria se realiza una prueba para evaluar el modelo. En este

caso podemos ver como con un **99.604%** se inclina por decir que la imagen detectada es un **“T-shirt”** y efectivamente es correcta la predicción .

```
1/1 [==============================] - 0s 36ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 24ms/step
Total de aciertos en 100 experimentos: 90
```

Figura 1.14 (Evaluación y Predicción correcta con respecto a 100 eventos aleatorios con ResNet)

Conclusiones del modelo.

- **Particularidades del modelo con el dataset utilizado.**

A través de la investigación del modelo de ResNet, pude ver que es ideal para entornos computacionales con un alto poder cuando se trata de versiones mayores como ResNet50 y ResNet100 acostumbradas a un número alto de épocas, y ofrece excelentes resultados en dichos dispositivos [4], sin embargo de forma convencional o en el trabajo gratuito en la nube, algunos de estos pueden suponer un costo computacional demasiado elevado, pero me parece increible que puedan plantearse adaptaciones de modelos como **ResNet** que se basen en otros un poco más complejos como el **ResNet50** o **ResNet100**, en el que mi objetivo personal durante la realización de esta problemática era la eficiencia en ambos modelos, **LeNet5** y **ResNet**, con un balance entre **accuracy** y **costo computacional** en tiempo (al menos siendo algo razonables en ResNet en comparativa con sus modelos superiores), que a través del entrenamiento por épocas pudieran modificarse para experimentación sin problemas y sin comprometer demasiado el factor tiempo, por ello también encontrar el punto óptimo del número de épocas y la configuración de los hiperparametros.

Me pareció sorprendente que a pesar de tratarse de una red ResNet, el desenvolvimiento a través de esta base de datos fue sumamente agradable, teniendo un gran accuracy desde un inicio y pudieron llegar cercanamente a un **accuracy** de **90%** en tan solo **10 épocas** aunque con un costo computacional en tiempo equivalente a casi 30 min de procesamiento computacional en el entrenamiento y posterior evaluación del modelo, lo que considero algo ineficiente, debido a que obtuvimos un resultado equivalente con lennet en un menor tiempo.

- **Comportamiento para predecir la categoría. ¿Funcionó como lo esperabas?**

Al igual que el modelo LeNet, el modelo planteado de ResNet tuvo un resultado espectacular en mi parecer para tratarse de una red convolucional con ese nivel de accuracy, en el cual después de 1**0 épocas** converge a uno cercano a **90%,** y que a través de otra prueba adicional donde pongo a prueba 100 iteraciones de prueba para el modelo, podemos ver que acierta en 90 de ellas lo que es un muy buen resultado y que por consiguiente es lógico debido al nivel de accuracy obtenido.

Comparación general de los 2 modelos

¿Cuál fue el más latoso? ¿por qué?

Aunque no lo definiría como latoso, al momento de la implementación y experimentación considero que la red ResNet fue un poco más complicada en mi opinión, debido al costo computacional que implica ya que va destinado más a dispositivos de alto poder donde se pueden obtener resultados interesantes al elevarlos a un número alto de épocas, sin embargo considero que sin duda fue una opción interesante experimentar con ella en este entorno, obteniendo resultados favorables aunque sin duda debo decir que tardados y que tal vez a pesar del accuracy mayor, pierde eficiencia en entornos de la nube o dispositivos de bajo poder computacional.

¿Cuál se entrenó mejor? ¿por qué?

Aunque en mi opinión por la simplicidad, LeNet tiene una clara superioridad ante ResNet aun en su versión simplificada, ResNet tiene mejor potencial con un mayor poder de cómputo conforme se ve que va evolucionando en su accuracy mientras que LeNet lo hace de forma muy pasiva y con estancamiento aparente a lo largo de las epocas, sin embargo por el entorno trabajado en nube por medio de Colab y considerando de forma arbitraria los factores me gusta la forma en que entrena LeNet por su futuro desempeño.

¿Cuál tuvo mejor desempeño de acuerdo con las métricas seleccionadas?

En torno a los parámetros seleccionados siendo el más destacable sobre todo el accuracy, aunque de forma decimal en porcentaje, LeNet obtuvo un resultado insignificantemente mayor que ResNet, pero ambos convergen cercanamente a 90% lo que sigue siendo un buen resultado. Y como se pueden visualizar de forma porcentual en las demás métricas como precisión, recall, puntuación F1 y soporte etc. (las cuales en LeNet convergen cercanamente al valor del accuracy), sigue siendo ligeramente mejor en cada una de ellas.

Observaciones sobre tiempos y recursos requeridos.

Uno de los factores que más aprecie en el desempeño de ambos modelos ilustrado a lo largo del reporte, se sobrepone muchísimo en cuestión de eficiencia para un buen resultado funcional, siendo LeNet quien nuevamente toma ventaja en este sentido sobre esta versión de ResNet, debido a que consumió solo cerca de 7 min aprox en comparativa con los 30 que se demoró ResNet debido al poder computacional requerido para esta clase de modelos.

Conclusiones

Posibles mejoras de cada modelo y en general.

Para LeNet:

- Aumentar su profundidad con más capas convolucionales y neuronas podría ayudar a capturar características más abstractas y complejas de las imágenes.
- Agregar capas de regularización como Dropout o técnicas de regularización L1/L2 podría ayudar a prevenir el sobreajuste y mejorar la generalización del modelo.
- La exploración hacia arquitecturas un poco más avanzadas como la explorada en su comparativa, ResNet, Inception o algunos Transformers.
- Debido a que disponemos de una cantidad de datos finita de datos y si se exploraron múltiples combinaciones de hiperparametros y estructuras, en ese sentido concluimos que en ese aspecto estamos cubiertos

Para ResNet:

- Aumentar la profundidad: Agregar más capas residuales o bloques residuales puede ayudar a capturar características más complejas y abstractas en los datos.[3]
- Variantes de ResNet: Explorar variantes como ResNet-50 (Como la explorada en comparativa), ResNet-101 o ResNet-152, que son más profundas y pueden capturar representaciones más ricas, obviamente con una disposición de tiempo, y recursos mayores.
- Dropout: Agregar capas de dropout para evitar el sobreajuste, debido a que observamos ese problema en esta situación)
- Regularización L1/L2: Introducir términos de regularización para penalizar pesos grandes en lFunciones de activación: Experimentar con otras funciones de activación como Leaky ReLU, PReLU, o ReLU modificado (como Swish) para ver si mejoran el rendimiento.a función de pérdida. En cuestión de los optimizadores la realidad es que podríamos probar algunos pero Adam se comporta de manera eficiente en estos modelos.

- Uso de GPU/TPU: Aprovechar el paralelismo masivo de GPU o TPU para acelerar el entrenamiento y la inferencia. Debido a las características de alta necesidad de recursos, destinarlos desde la GPU podría ser una manera de eficientar procesos.

En General:

- A pesar de utilizar una experimentación con distintos hiperparametros e ir probando, podría ser interesante darle un vistazo a los buscadores de hiperparámetros, como la optimización bayesiana, la búsqueda aleatoria o la optimización basada en gradiente, son herramientas poderosas para encontrar los conjuntos óptimos de hiperparámetros para una red neuronal.
- Fuera de ello, yo creo que la implementación de otros modelos que están tomando relevancia en la actualidad como los transformers serán una opción interesante a explorar durante un problema similar a este.

¿Qué aprendiste del proyecto?

Al comparar modelos como ResNet y LeNet para resolver un problema de clasificación de conjuntos de moda, he aprendido valiosas lecciones sobre la importancia de la arquitectura de las redes neuronales convolucionales. La profundidad y complejidad de los modelos juegan un papel crucial en su capacidad para capturar características complejas y abstractas en los datos.

Observé cómo ResNet, con su estructura más profunda y conexiones residuales, supera a LeNet en términos de capacidad de generalización y precisión en problemas de clasificación desafiantes, como el reconocimiento de moda. Sin embargo, esta mejora en el rendimiento conlleva un costo computacional mayor, la cual por ejemplo en este escenario no se pudo explotar todo el potencial debido a los recursos disponibles para ResNet. Esta comparativa me ha enseñado que la elección entre modelos como ResNet y LeNet depende del contexto específico, los recursos disponibles y la complejidad de los datos del problema en cuestión. En última instancia, la comprensión de las fortalezas y debilidades de cada arquitectura es crucial para tomar decisiones informadas al abordar problemas de clasificación visual complejos.

¿Qué fue lo que más te gustó?

Siento que la comprensión y trasfondo en la comprensión de cómo realizar y mejorar la implementación modelos de redes neuronales, así como la distinción del proceso y el porqué unas pueden adaptarse más a otros escenarios que otras, como por ejemplo en este caso, saber que las CNN pueden ser las indicadas para esta clase de cuestiones, y observar y realizar esto por primera vez fue una experiencia unica, ademas que me llevo la experiencia y curiosidad de seguirme nutriendo y diversificando mis conocimientos y la visualización de aplicaciones de estas poderosas herramientas.

¿Qué fue lo que menos te gustó?

Debo admitir que por diversas cuestiones fue algo frustrante en algunas ocasiones el entendimiento desde 0 de un nuevo modelo de redes nunca antes visto, no por el hecho de descubrirlo sino por los tecnicismos o las syntaxis en la programación que me impedían comprender el procesos observable o la interpretación de algunas acciones básicas de las redes neuronales, sin embargo, considero que fue una actividad desafiante y que me puso a prueba en un par de ocasiones, en especial con ResNet y las exigencias de recursos computacionales y tiempo requeridas así como ciertos problemas en cuestión de la configuración de las capas, pero me siento satisfecho de haber conseguido librar las dificultades encontradas.

Referencias

[1] ¿Qué es una red neuronal convolucional? | 3 cosas que debe saber. (2023). Mathworks.com. https://es.mathworks.com/discovery/convolutional-neural-network-matlab.html

[2] López, F | (2019). Clasificación de imágenes usando redes neuronales convolucionales. http://zaloamati.azc.uam.mx/bitstream/handle/11191/6123/Clasificacion_de_imagenes_Lopez_Saca_F_2019.pdf?sequence=1

[3] Rubén Rodríguez Abril. (2022, January 15). Redes Residuales: ResNet • Un artículo de LMO. La Máquina Oráculo. https://lamaquinaoraculo.com/deep-learning/redes-residuales/#:~:text=Las%20redes%20residuales%20(ResNets)%20permiten,que%20conectan%20capas%20no%20contiguas.

[4] VisibleBreadcrumbs. (2023). Mathworks.com. https://es.mathworks.com/help/deeplearning/ref/resnet50.html