Actividad 1: Conceptos generales de redes neuronales

En esta actividad vamos a revisar algunos de los conceptos basicos de las redes neuronales, pero no por ello menos importantes.

El dataset a utilizar es Fashion MNIST, un problema sencillo con imágenes pequeñas de ropa, pero más interesante que el dataset de MNIST. Puedes consultar más información sobre el dataset en este enlace.

El código utilizado para contestar tiene que quedar claramente reflejado en el Notebook. Puedes crear nuevas cells si así lo deseas para estructurar tu código y sus salidas. A la hora de entregar el notebook, asegúrate de que los resultados de ejecutar tu código han quedado guardados (por ejemplo, a la hora de entrenar una red neuronal tiene que verse claramente un log de los resultados de cada epoch).

In [1]:
```
import tensorflow as tf
print(tf.__version__)
```

```
2.8.0
```

En primer lugar vamos a importar el dataset Fashion MNIST (recordad que este es uno de los dataset de entranamiento que estan guardados en keras) que es el que vamos a utilizar en esta actividad:

In [2]:
```
mnist = tf.keras.datasets.fashion_mnist
```

Llamar a **load_data** en este dataset nos dará dos conjuntos de dos listas, estos serán los valores de entrenamiento y prueba para los gráficos que contienen las prendas de vestir y sus etiquetas.

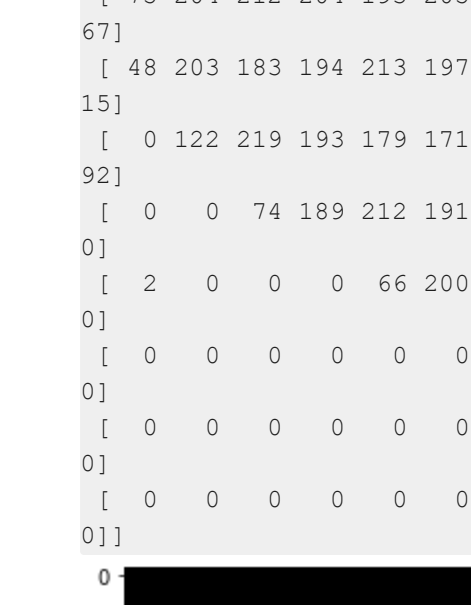
Nota: Aunque en esta actividad lo veis de esta forma, también lo vais a poder encontrar como 4 variables de esta forma: training_images, training_labels, test_images, test_labels = mnist.load_data()

In [3]:
```
(training_images, training_labels), (test_images, test_labels) = mnist.load_data()
```

Antes de continuar vamos a dar un vistazo a nuestro dataset, para ello vamos a ver una imagen de entrenamiento y su etiqueta o clase.

In [4]:
```
import numpy as np
np.set_printoptions(linewidth=200)
import matplotlib.pyplot as plt
plt.imshow(training_images[0], cmap="gray") # recordad que siempre es preferible trabajar en blanco y negro
#
print(training_labels[0])
print(training_images[0])
```

```
9
[[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   1   0   0  13  73   0   0   1   4   0   0   0   0   1   1   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   3   0  36 136 127  62  54   0   0   0   1   3   4   0   0   3]
 [  0   0   0   0   0   0   0   0   0   0   0   0   6   0 102 204 176 134 144 123  23   0   0   0   0  12  10   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0 155 236 207 178 107 156 161 109  64  23  77 130  72  15]
 [  0   0   0   0   0   0   0   0   0   0   0   1   0  69 207 223 218 216 216 163 127 121 122 146 141  88 172  66]
 [  0   0   0   0   0   0   0   0   0   1   1   1   0 200 232 232 233 229 223 223 215 213 164 127 123 196 229   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0 183 225 216 223 228 235 227 224 222 224 221 223 245 173   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0 193 228 218 213 198 180 212 210 211 213 223 220 243 202   0]
 [  0   0   0   0   0   0   0   0   0   1   3   0  12 219 220 212 218 192 169 227 208 218 224 212 226 197 209  52]
 [  0   0   0   0   0   0   0   0   0   0   6   0  99 244 222 220 218 203 198 221 215 213 222 220 245 119 167  56]
 [  0   0   0   0   0   0   0   0   0   4   0   0  55 236 228 230 228 240 232 213 218 223 234 217 217 209  92   0]
 [  0   0   1   4   6   7   2   0   0   0   0   0 237 226 217 223 222 219 222 221 216 223 229 215 218 255  77   0]
 [  0   3   0   0   0   0   0   0   0  62 145 204 228 207 213 221 218 208 211 218 224 223 219 215 224 244 159   0]
 [  0   0   0   0  18  44  82 107 189 228 220 222 217 226 200 205 211 230 224 234 176 188 250 248 233 238 215   0]
 [  0  57 187 208 224 221 224 208 204 214 208 209 200 159 245 193 206 223 255 255 221 234 221 211 220 232 246   0]
 [  3 202 228 224 221 211 211 214 205 205 205 220 240  80 150 255 229 221 188 154 191 210 204 209 222 228 225   0]
 [ 98 233 198 210 222 229 229 234 249 220 194 215 217 241  65  73 106 117 168 219 221 215 217 223 223 224 229  29]
 [ 75 204 212 204 193 205 211 225 216 185 197 206 198 213 240 195 227 245 239 223 218 212 209 222 220 221 230  67]
 [ 48 203 183 194 213 197 185 190 194 192 202 214 219 221 220 236 225 216 199 206 186 181 177 172 181 205 206 115]
 [  0 122 219 193 179 171 183 196 204 210 213 207 211 210 200 196 194 191 195 191 198 192 176 156 167 177 210  92]
 [  0   0  74 189 212 191 175 172 175 181 185 188 189 188 193 198 204 209 210 210 211 188 188 194 192 216 170   0]
 [  2   0   0   0  66 200 222 237 239 242 246 243 244 221 220 193 191 179 182 182 181 176 166 168  99  58   0   0]
 [  0   0   0   0   0   0   0  40  61  44  72  41  35   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]]
```



Habreis notado que todos los valores numericos están entre 0 y 255. Si estamos entrenando una red neuronal, una buena practica es transformar todos los valores entre 0 y 1, un proceso llamado "normalización" y afortunadamente en Python es fácil normalizar una lista. Lo puedes hacer de esta manera:

In [5]:
```
training_images  = training_images / 255.0
test_images = test_images / 255.0
```

Ahora vamos a definir el modelo, pero antes vamos a repasar algunos comandos y conceptos muy utiles:

- **Sequential**: Eso define una SECUENCIA de capas en la red neuronal
- **Dense**: Añade una capa de neuronas
- **Flatten**: ¿Recuerdas que las imágenes cómo eran las imágenes cuando las imprimiste para poder verlas? Un cuadrado, Flatten sólo toma ese cuadrado y lo convierte en un vector de una dimensión.

Cada capa de neuronas necesita una función de activación. Normalmente se usa la función relu en las capas intermedias y softmax en la ultima capa

- **Relu** significa que "Si X>0 devuelve X, si no, devuelve 0", así que lo que hace es pasar sólo valores 0 o mayores a la siguiente capa de la red.
- **Softmax** toma un conjunto de valores, y escoge el más grande.

Pregunta 1 (3.5 puntos): Utilizando Keras, y preparando los datos de X e y como fuera necesario, define y entrena una red neuronal que sea capaz de clasificar imágenes de Fashion MNIST con las siguientes características:

- Una hidden layer de tamaños 128, utilizando unidades sigmoid Optimizador Adam.
- Durante el entrenamiento, la red tiene que mostrar resultados de loss y accuracy por cada epoch.
- La red debe entrenar durante 10 epochs y batch size de 64.
- La última capa debe de ser una capa softmax.
- Tu red tendría que ser capaz de superar fácilmente 80% de accuracy.

In [6]:
```
### Tu código para la red neuronal de la pregunta 1 aquí ###
from tensorflow import keras
from tensorflow.keras import layers

model = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(128, activation='sigmoid'),
    keras.layers.Dense(10, activation='softmax')
])
model.compile(loss = "sparse_categorical_crossentropy" , optimizer="adam",metrics=["accuracy"])
model.fit(training_images,training_labels,batch_size = 64 , epochs = 10)
```

```
Epoch 1/10
938/938 [==============================] - 2s 2ms/step - loss: 0.5959 - accuracy: 0.8045
Epoch 2/10
938/938 [==============================] - 2s 2ms/step - loss: 0.4104 - accuracy: 0.8537
Epoch 3/10
938/938 [==============================] - 2s 2ms/step - loss: 0.3727 - accuracy: 0.8679
Epoch 4/10
938/938 [==============================] - 2s 2ms/step - loss: 0.3479 - accuracy: 0.8751
Epoch 5/10
938/938 [==============================] - 2s 2ms/step - loss: 0.3310 - accuracy: 0.8805
Epoch 6/10
938/938 [==============================] - 2s 2ms/step - loss: 0.3154 - accuracy: 0.8851
Epoch 7/10
938/938 [==============================] - 2s 2ms/step - loss: 0.3021 - accuracy: 0.8904
Epoch 8/10
938/938 [==============================] - 2s 2ms/step - loss: 0.2917 - accuracy: 0.8949
Epoch 9/10
938/938 [==============================] - 2s 2ms/step - loss: 0.2821 - accuracy: 0.8973
Epoch 10/10
938/938 [==============================] - 2s 2ms/step - loss: 0.2725 - accuracy: 0.8997
```

Out[6]:
```
<keras.callbacks.History at 0x1b90e18e910>
```

Para concluir el entrenamiento de la red neuronal, una buena practica es evaluar el modelo para ver si la precisión de entrenamiento es real

pregunta 2 (0.5 puntos): evalua el modelo con las imagenes y etiquetas test.

In [7]:
```
### Tu código para la evaluación de la red neuronal de la pregunta 2 aquí ###

score = model.evaluate (test_images, test_labels)
```

```
313/313 [==============================] - 1s 1ms/step - loss: 0.3432 - accuracy: 0.8729
```

Ahora vamos a explorar el código con una serie de ejercicios para alcanzar un grado de comprensión mayor sobre las redes neuronales y su entrenamiento.

Ejercicio 1: Funcionamiento de las predicción de la red neuronal

Para este primer ejercicio sigue los siguientes pasos:

- Crea una variable llamada **classifications** para construir un clasificador para las imágenes de prueba, para ello puedes utilizar la función predict sobre el conjunto de test
- Imprime con la función print la primera entrada en las clasificaciones.

pregunta 3.1 (0.25 puntos), el resultado al imprimirlo es un vector de números,

- ¿Por qué crees que ocurre esto, y qué representa este vector de números?

pregunta 3.2 (0.25 puntos)

- ¿Cuál es la clase de la primera entrada# de la variable **classifications**? La respuesta puede ser un número o su etiqueta/clase equivalente.

In [11]:
```
### Tu código del clasificador de la pregunta 3 aquí ###
classifications = model.predict(test_images)
classifications[0]
```

Out[11]:
```
array([2.5140305e-05, 1.0595375e-07, 1.1928208e-05, 8.5256252e-06, 3.7316753e-05, 1.8949447e-02, 9.5701122e-05,
       2.8292815e-02, 8.9763629e-04, 9.5168161e-01], dtype=float32)
```

Tu respuesta a la pregunta 3.1 aquí: es un vector que guarda las probabilidades de las posibles categorias.

Tu respuesta a la pregunta 3.2 aquí: 2.5140305e-05

Ejercicio 2: Impacto variar el número de neuronas en las capas ocultas

En este ejercicio vamos a experimentar con nuestra red neuronal cambiando el numero de neuronas por 512 y por 1024. Para ello, utiliza la red neuronal de la pregunta 1, y su capa oculta cambia las 128 neuronas:

- **pregunta 4.1 (0.25 puntos)**: 512 neuronas en la capa oculta
- **pregunta 4.2 (0.25 puntos)**:1024 neuronas en la capa oculta

y entrena la red en ambos casos.

pregunta 4.3 (0.5 puntos): ¿Cual es el impacto que tiene la red neuronal?

In [12]:
```
### Tu código para 512 neuronas aquí ###

model_2 = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(512, activation='sigmoid'),
    keras.layers.Dense(10, activation='softmax')
])
model_2.compile(loss = "sparse_categorical_crossentropy" , optimizer="adam",metrics=
["accuracy"])
model_2.fit(training_images,training_labels,batch_size = 64 , epochs = 10)
```

```
Epoch 1/10
938/938 [==============================] - 4s 4ms/step - loss: 0.5297 - accuracy: 0.8115
Epoch 2/10
938/938 [==============================] - 4s 4ms/step - loss: 0.4013 - accuracy: 0.8555
Epoch 3/10
938/938 [==============================] - 4s 4ms/step - loss: 0.3626 - accuracy: 0.8690
Epoch 4/10
938/938 [==============================] - 4s 4ms/step - loss: 0.3358 - accuracy: 0.8785
Epoch 5/10
938/938 [==============================] - 4s 4ms/step - loss: 0.3163 - accuracy: 0.8838
Epoch 6/10
938/938 [==============================] - 4s 4ms/step - loss: 0.2978 - accuracy: 0.8910
Epoch 7/10
938/938 [==============================] - 4s 4ms/step - loss: 0.2830 - accuracy: 0.8954
Epoch 8/10
938/938 [==============================] - 4s 4ms/step - loss: 0.2706 - accuracy: 0.8993
Epoch 9/10
938/938 [==============================] - 4s 4ms/step - loss: 0.2558 - accuracy: 0.9043
Epoch 10/10
938/938 [==============================] - 4s 4ms/step - loss: 0.2462 - accuracy: 0.9089
```

Out[12]:
```
<keras.callbacks.History at 0x1b90d00aa60>
```

In [14]:
```
### Tu código para 1024 neuronas aquí ###

model_3 = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(1024, activation='sigmoid'),
    keras.layers.Dense(10, activation='softmax')
])
model_3.compile(loss = "sparse_categorical_crossentropy" , optimizer="adam",metrics=
["accuracy"])
model_3.fit(training_images,training_labels,batch_size = 64 , epochs = 10)
```

```
Epoch 1/10
938/938 [==============================] - 4s 4ms/step - loss: 0.5192 - accuracy: 0.8133
Epoch 2/10
938/938 [==============================] - 4s 4ms/step - loss: 0.4008 - accuracy: 0.8554
Epoch 3/10
938/938 [==============================] - 4s 4ms/step - loss: 0.3619 - accuracy: 0.8681
Epoch 4/10
938/938 [==============================] - 4s 4ms/step - loss: 0.3351 - accuracy: 0.8771
Epoch 5/10
938/938 [==============================] - 4s 4ms/step - loss: 0.3117 - accuracy: 0.8852
Epoch 6/10
938/938 [==============================] - 4s 4ms/step - loss: 0.2939 - accuracy: 0.8910
Epoch 7/10
938/938 [==============================] - 4s 4ms/step - loss: 0.2780 - accuracy: 0.8971
Epoch 8/10
938/938 [==============================] - 4s 4ms/step - loss: 0.2620 - accuracy: 0.9022
Epoch 9/10
938/938 [==============================] - 4s 4ms/step - loss: 0.2491 - accuracy: 0.9063
Epoch 10/10
938/938 [==============================] - 4s 4ms/step - loss: 0.2373 - accuracy: 0.9114
```

Out[14]:
```
<keras.callbacks.History at 0x1b90b53a150>
```

Tu respuesta a la pregunta 4.3 aquí: el accuracy con mas neuronas incremento, aunque la velocidad de ejecucion disminuyo. Con 1024 neuronas el accuracy subio de .8997 a .09114

Si ahora entrenais el modelo de esta forma (con 512 y 1024 neuronas en la capa oculta) y volveis a ejecutar el predictor guardado en la variable **classifications**, escribir el código del clasificador del ejercicio 1 de nuevo e imprimid el primer objeto guardado en la variable classifications.

pregunta 5.1 (0.25 puntos):

- ¿En que clase esta clasificado ahora la primera prenda de vestir de la variable classifications?

pregunta 5.1 (0.25 puntos):

- ¿Porque crees que ha ocurrido esto?

In [15]:
```
### Tu código del clasificador de la pregunta 5 aquí ###

classifications_2 = model.predict(test_images)
print(classifications_2[0])
classifications_3 = model.predict(test_images)
print(classifications_3[0])
```

```
[4.2243528e-07 6.2569725e-08 1.2545759e-07 2.2208357e-07 6.9335954e-08 1.0819657e-02 5.0021135e-06 3.4473564e-02
 2.3.7395185e-07 9.5470027e-01]
[4.2243528e-07 6.2569725e-08 1.2545759e-07 2.2208357e-07 6.9335954e-08 1.0819657e-02 5.0021135e-06 3.4473564e-02
 2.3.7395185e-07 9.5470027e-01]
```

Tu respuesta a la pregunta 5.1 aquí: ahora esta en 4.2243528e-07 tanto para 512 como 1024 neuronas

Tu respuesta a la pregunta 5.2 aquí: por que el accuracy ha sido mejor con mas neuronas, la probabilidad de la prediccion tambien puede cambiar

Ejercicio 3: ¿por qué es tan importante la capa Flatten?

En este ejercicio vamos a ver que ocurre cuando quitamos la capa flatten, para ello, escribe la red neuronal de la pregunta 1 y no pongas la capa Flatten.

pregunta 6 (0.5 puntos): ¿puedes explicar porque da el error que da?

In [17]:
```
### Tu código de la red neuronal sin capa flatten de la pregunta 6 aquí ###

model_4 = keras.Sequential([
    #keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(1024, activation='sigmoid'),
    keras.layers.Dense(10, activation='softmax')
])
model_4.compile(loss = "sparse_categorical_crossentropy" , optimizer="adam",metrics=
["accuracy"])
model_4.fit(training_images,training_labels,batch_size = 64 , epochs = 10)
```

```
Epoch 1/10
---------------------------------------------------------------------------
InvalidArgumentError                      Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_7016/1215204643.py in <module>
      7 ])
      8 model_4.compile(loss = "sparse_categorical_crossentropy" , optimizer="adam",metrics=["accuracy"])
----> 9 model_4.fit(training_images,training_labels,batch_size = 64 , epochs = 10)

C:\ProgramData\Anaconda3\lib\site-packages\keras\utils\traceback_utils.py in error_handler(*args, **kwargs)
     65     except Exception as e:  # pylint: disable=broad-except
     66       filtered_tb = _process_traceback_frames(e.__traceback__)
---> 67       raise e.with_traceback(filtered_tb) from None
     68     finally:
     69       del filtered_tb

C:\ProgramData\Anaconda3\lib\site-packages\tensorflow\python\eager\execute.py in quick_execute(op_name, num_out
puts, inputs, attrs, ctx, name)
     52   try:
     53     ctx.ensure_initialized()
---> 54     tensors = pywrap_tfe.TFE_Py_Execute(ctx._handle, device_name, op_name,
     55                                         inputs, attrs, num_outputs)
     56   except core._NotOkStatusException as e:

InvalidArgumentError: Graph execution error:

Detected at node 'sparse_categorical_crossentropy/SparseSoftmaxCrossEntropyWithLogits/SparseSoftmaxCrossEntropy
WithLogits' defined at (most recent call last):
    File "C:\ProgramData\Anaconda3\lib\runpy.py", line 197, in _run_module_as_main
      return _run_code(code, main_globals, None,
    File "C:\ProgramData\Anaconda3\lib\runpy.py", line 87, in _run_code
      exec(code, run_globals)
    File "C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py", line 16, in <module>
      app.launch_new_instance()
    File "C:\ProgramData\Anaconda3\lib\site-packages\traitlets\config\application.py", line 846, in launch_inst
ance
      app.start()
    File "C:\ProgramData\Anaconda3\lib\site-packages\ipykernel\kernelapp.py", line 677, in start
      self.io_loop.start()
    File "C:\ProgramData\Anaconda3\lib\site-packages\tornado\platform\asyncio.py", line 199, in start
      self.asyncio_loop.run_forever()
    File "C:\ProgramData\Anaconda3\lib\asyncio\base_events.py", line 596, in run_forever
      self._run_once()
    File "C:\ProgramData\Anaconda3\lib\asyncio\base_events.py", line 1890, in _run_once
      handle._run()
    File "C:\ProgramData\Anaconda3\lib\asyncio\events.py", line 80, in _run
      self._context.run(self._callback, *self._args)
    File "C:\ProgramData\Anaconda3\lib\site-packages\ipykernel\kernelbase.py", line 457, in dispatch_queue
      await self.process_one()
    File "C:\ProgramData\Anaconda3\lib\site-packages\ipykernel\kernelbase.py", line 446, in process_one
      await dispatch(*args)
    File "C:\ProgramData\Anaconda3\lib\site-packages\ipykernel\kernelbase.py", line 353, in dispatch_shell
      await result
    File "C:\ProgramData\Anaconda3\lib\site-packages\ipykernel\kernelbase.py", line 648, in execute_request
      reply_content = await reply_content
    File "C:\ProgramData\Anaconda3\lib\site-packages\ipykernel\ipkernel.py", line 353, in do_execute
      res = shell.run_cell(code, store_history=store_history, silent=silent)
    File "C:\ProgramData\Anaconda3\lib\site-packages\ipykernel\zmqshell.py", line 533, in run_cell
      return super(ZMQInteractiveShell, self).run_cell(*args, **kwargs)
    File "C:\ProgramData\Anaconda3\lib\site-packages\IPython\core\interactiveshell.py", line 2901, in run_cell
      result = self._run_cell(
    File "C:\ProgramData\Anaconda3\lib\site-packages\IPython\core\interactiveshell.py", line 2947, in _run_cell
      return runner(coro)
    File "C:\ProgramData\Anaconda3\lib\site-packages\IPython\core\async_helpers.py", line 68, in _pseudo_sync_r
unner
      coro.send(None)
    File "C:\ProgramData\Anaconda3\lib\site-packages\IPython\core\interactiveshell.py", line 3172, in run_cell_
async
      has_raised = await self.run_ast_nodes(code_ast.body, cell_name,
    File "C:\ProgramData\Anaconda3\lib\site-packages\IPython\core\interactiveshell.py", line 3364, in run_ast_n
odes
      if (await self.run_code(code, result,  async_=asy)):
    File "C:\ProgramData\Anaconda3\lib\site-packages\IPython\core\interactiveshell.py", line 3444, in run_code
      exec(code_obj, self.user_global_ns, self.user_ns)
    File "C:\Users\ASUS\AppData\Local\Temp/ipykernel_7016/1215204643.py", line 9, in <module>
      model_4.fit(training_images,training_labels,batch_size = 64 , epochs = 10)
    File "C:\ProgramData\Anaconda3\lib\site-packages\keras\utils\traceback_utils.py", line 64, in error_handler
      return fn(*args, **kwargs)
    File "C:\ProgramData\Anaconda3\lib\site-packages\keras\engine\training.py", line 1384, in fit
      tmp_logs = self.train_function(iterator)
    File "C:\ProgramData\Anaconda3\lib\site-packages\keras\engine\training.py", line 1021, in train_function
      return step_function(self, iterator)
    File "C:\ProgramData\Anaconda3\lib\site-packages\keras\engine\training.py", line 1010, in step_function
      outputs = model.distribute_strategy.run(run_step, args=(data,))
    File "C:\ProgramData\Anaconda3\lib\site-packages\keras\engine\training.py", line 1000, in run_step
      outputs = model.train_step(data)
    File "C:\ProgramData\Anaconda3\lib\site-packages\keras\engine\training.py", line 860, in train_step
      loss = self.compute_loss(x, y, y_pred, sample_weight)
    File "C:\ProgramData\Anaconda3\lib\site-packages\keras\engine\training.py", line 918, in compute_loss
      return self.compiled_loss(
    File "C:\ProgramData\Anaconda3\lib\site-packages\keras\engine\compile_utils.py", line 201, in __call__
      loss_value = loss_obj(y_t, y_p, sample_weight=sw)
    File "C:\ProgramData\Anaconda3\lib\site-packages\keras\losses.py", line 141, in __call__
      losses = call_fn(y_true, y_pred)
    File "C:\ProgramData\Anaconda3\lib\site-packages\keras\losses.py", line 245, in call
      return ag_fn(y_true, y_pred, **self._fn_kwargs)
    File "C:\ProgramData\Anaconda3\lib\site-packages\keras\losses.py", line 1862, in sparse_categorical_crossen
tropy
      return backend.sparse_categorical_crossentropy(
    File "C:\ProgramData\Anaconda3\lib\site-packages\keras\backend.py", line 5202, in sparse_categorical_crosse
ntropy
      res = tf.nn.sparse_softmax_cross_entropy_with_logits(
Node: 'sparse_categorical_crossentropy/SparseSoftmaxCrossEntropyWithLogits/SparseSoftmaxCrossEntropyWithLogits'
logits and labels must have the same first dimension, got logits shape [1792,10] and labels shape [64]
         [[{{node sparse_categorical_crossentropy/SparseSoftmaxCrossEntropyWithLogits/SparseSoftmaxCrossEntropy
WithLogits}}]] [Op:__inference_train_function_198800]
```

Tu respuesta a la pregunta 6 aquí: la dimension de los datos , osea [28,28] superan los datos esperados por la red, por eso se usa Flatten , para hacerlos de un arreglo a un vector.

Ejercicio 4: Número de neuronas de la capa de salida

Considerad la capa final, la de salida de la red neuronal de la pregunta 1.

pregunta 7.1 (0.25 puntos): ¿Por qué son 10 las neuronas de la última capa?

pregunta 7.2 (0.25 puntos): ¿Qué pasaría si tuvieras una cantidad diferente a 10?

Por ejemplo, intenta entrenar la red con 5, para ello utiliza la red neuronal de la pregunta 1 y cambia a 5 el número de neuronas en la última capa.

In [18]:
```
### Tu código de la red neuronal con 5 neuronas en la capa de salida de la pregunta 7 aquí ###

model_5 = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(1024, activation='sigmoid'),
    keras.layers.Dense(5, activation='softmax')
])
model_5.compile(loss = "sparse_categorical_crossentropy" , optimizer="adam",metrics=
["accuracy"])
model_5.fit(training_images,training_labels,batch_size = 64 , epochs = 10)
```

```
Epoch 1/10
---------------------------------------------------------------------------
InvalidArgumentError                      Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_7016/895817041.py in <module>
      7 ])
      8 model_5.compile(loss = "sparse_categorical_crossentropy" , optimizer="adam",metrics=["accuracy"])
----> 9 model_5.fit(training_images,training_labels,batch_size = 64 , epochs = 10)
     10

C:\ProgramData\Anaconda3\lib\site-packages\keras\utils\traceback_utils.py in error_handler(*args, **kwargs)
     65     except Exception as e:  # pylint: disable=broad-except
     66       filtered_tb = _process_traceback_frames(e.__traceback__)
---> 67       raise e.with_traceback(filtered_tb) from None
     68     finally:
     69       del filtered_tb

C:\ProgramData\Anaconda3\lib\site-packages\tensorflow\python\eager\execute.py in quick_execute(op_name, num_out
puts, inputs, attrs, ctx, name)
     52   try:
     53     ctx.ensure_initialized()
---> 54     tensors = pywrap_tfe.TFE_Py_Execute(ctx._handle, device_name, op_name,
     55                                         inputs, attrs, num_outputs)
     56   except core._NotOkStatusException as e:

InvalidArgumentError: Graph execution error:

Detected at node 'sparse_categorical_crossentropy/SparseSoftmaxCrossEntropyWithLogits/SparseSoftmaxCrossEntropy
WithLogits' defined at (most recent call last):
    File "C:\ProgramData\Anaconda3\lib\runpy.py", line 197, in _run_module_as_main
      return _run_code(code, main_globals, None,
    File "C:\ProgramData\Anaconda3\lib\runpy.py", line 87, in _run_code
      exec(code, run_globals)
    File "C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py", line 16, in <module>
      app.launch_new_instance()
    File "C:\ProgramData\Anaconda3\lib\site-packages\traitlets\config\application.py", line 846, in launch_inst
ance
      app.start()
    File "C:\ProgramData\Anaconda3\lib\site-packages\ipykernel\kernelapp.py", line 677, in start
      self.io_loop.start()
    File "C:\ProgramData\Anaconda3\lib\site-packages\tornado\platform\asyncio.py", line 199, in start
      self.asyncio_loop.run_forever()
    File "C:\ProgramData\Anaconda3\lib\asyncio\base_events.py", line 596, in run_forever
      self._run_once()
    File "C:\ProgramData\Anaconda3\lib\asyncio\base_events.py", line 1890, in _run_once
      handle._run()
    File "C:\ProgramData\Anaconda3\lib\asyncio\events.py", line 80, in _run
      self._context.run(self._callback, *self._args)
    File "C:\ProgramData\Anaconda3\lib\site-packages\ipykernel\kernelbase.py", line 457, in dispatch_queue
      await self.process_one()
    File "C:\ProgramData\Anaconda3\lib\site-packages\ipykernel\kernelbase.py", line 446, in process_one
      await dispatch(*args)
    File "C:\ProgramData\Anaconda3\lib\site-packages\ipykernel\kernelbase.py", line 353, in dispatch_shell
      await result
    File "C:\ProgramData\Anaconda3\lib\site-packages\ipykernel\kernelbase.py", line 648, in execute_request
      reply_content = await reply_content
    File "C:\ProgramData\Anaconda3\lib\site-packages\ipykernel\ipkernel.py", line 353, in do_execute
      res = shell.run_cell(code, store_history=store_history, silent=silent)
    File "C:\ProgramData\Anaconda3\lib\site-packages\ipykernel\zmqshell.py", line 533, in run_cell
      return super(ZMQInteractiveShell, self).run_cell(*args, **kwargs)
    File "C:\ProgramData\Anaconda3\lib\site-packages\IPython\core\interactiveshell.py", line 2901, in run_cell
      result = self._run_cell(
    File "C:\ProgramData\Anaconda3\lib\site-packages\IPython\core\interactiveshell.py", line 2947, in _run_cell
      return runner(coro)
    File "C:\ProgramData\Anaconda3\lib\site-packages\IPython\core\async_helpers.py", line 68, in _pseudo_sync_r
unner
      coro.send(None)
    File "C:\ProgramData\Anaconda3\lib\site-packages\IPython\core\interactiveshell.py", line 3172, in run_cell_
async
      has_raised = await self.run_ast_nodes(code_ast.body, cell_name,
    File "C:\ProgramData\Anaconda3\lib\site-packages\IPython\core\interactiveshell.py", line 3364, in run_ast_n
odes
      if (await self.run_code(code, result,  async_=asy)):
    File "C:\ProgramData\Anaconda3\lib\site-packages\IPython\core\interactiveshell.py", line 3444, in run_code
      exec(code_obj, self.user_global_ns, self.user_ns)
    File "C:\Users\ASUS\AppData\Local\Temp/ipykernel_7016/895817041.py", line 9, in <module>
      model_5.fit(training_images,training_labels,batch_size = 64 , epochs = 10)
    File "C:\ProgramData\Anaconda3\lib\site-packages\keras\utils\traceback_utils.py", line 64, in error_handler
      return fn(*args, **kwargs)
    File "C:\ProgramData\Anaconda3\lib\site-packages\keras\engine\training.py", line 1384, in fit
      tmp_logs = self.train_function(iterator)
    File "C:\ProgramData\Anaconda3\lib\site-packages\keras\engine\training.py", line 1021, in train_function
      return step_function(self, iterator)
    File "C:\ProgramData\Anaconda3\lib\site-packages\keras\engine\training.py", line 1010, in step_function
      outputs = model.distribute_strategy.run(run_step, args=(data,))
    File "C:\ProgramData\Anaconda3\lib\site-packages\keras\engine\training.py", line 1000, in run_step
      outputs = model.train_step(data)
    File "C:\ProgramData\Anaconda3\lib\site-packages\keras\engine\training.py", line 860, in train_step
      loss = self.compute_loss(x, y, y_pred, sample_weight)
    File "C:\ProgramData\Anaconda3\lib\site-packages\keras\engine\training.py", line 918, in compute_loss
      return self.compiled_loss(
    File "C:\ProgramData\Anaconda3\lib\site-packages\keras\engine\compile_utils.py", line 201, in __call__
      loss_value = loss_obj(y_t, y_p, sample_weight=sw)
    File "C:\ProgramData\Anaconda3\lib\site-packages\keras\losses.py", line 141, in __call__
      losses = call_fn(y_true, y_pred)
    File "C:\ProgramData\Anaconda3\lib\site-packages\keras\losses.py", line 245, in call
      return ag_fn(y_true, y_pred, **self._fn_kwargs)
    File "C:\ProgramData\Anaconda3\lib\site-packages\keras\losses.py", line 1862, in sparse_categorical_crossen
tropy
      return backend.sparse_categorical_crossentropy(
    File "C:\ProgramData\Anaconda3\lib\site-packages\keras\backend.py", line 5202, in sparse_categorical_crosse
ntropy
      res = tf.nn.sparse_softmax_cross_entropy_with_logits(
Node: 'sparse_categorical_crossentropy/SparseSoftmaxCrossEntropyWithLogits/SparseSoftmaxCrossEntropyWithLogits'
Received a label value of 9 which is outside the valid range of [0, 5).  Label values: 9 5 3 4 0 6 0 1 3 5 6 4
 9 1 3 6 0 2 6 8 5 1 7 5 8 9 9 5 6 3 4 3 6 9 2 3 4 6 9 2 0 7 8 2 7 6 2 1 7 3 0 2 9 9 4 4 6
         [[{{node sparse_categorical_crossentropy/SparseSoftmaxCrossEntropyWithLogits/SparseSoftmaxCrossEntropy
WithLogits}}]] [Op:__inference_train_function_199379]
```

Tu respuestas a la pregunta 7.1 aquí: por que es el numero de categorias existentes

Tu respuestas a la pregunta 7.2 aquí: el error nos indica que es un error de dimensionalidad , ya que el numero esperado es igual al numero de categorias existentes

Ejercicio 5: Aumento de epoch y su efecto en la red neuronal

En este ejercicio vamos a ver el impacto de aumentar los epoch en el entrenamiento. Usando la red neuronal de la pregunta 1:

pregunta 8.1 (0.20 puntos)

- Intentad 15 epoch para su entrenamiento, probablemente obtendras un modelo con una pérdida mucho mejor que el que tiene 5.

pregunta 8.2 (0.20 puntos)

- Intenta ahora con 30 epoch para su entrenamiento, podrás ver que el valor de la pérdida deja de disminuir, y a veces aumenta.

pregunta 8.3 (0.60 puntos)

- ¿Porque que piensas que ocurre esto? Explica tu respuesta y da el nombre de este efecto si lo conoces.

In [19]:
```
### Tu código para 15 epoch aquí ###

model = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(128, activation='sigmoid'),
    keras.layers.Dense(10, activation='softmax')
])
model.compile(loss = "sparse_categorical_crossentropy" , optimizer="adam",metrics=["accuracy"])
model.fit(training_images,training_labels,batch_size = 64 , epochs = 15)
```

```
Epoch 1/15
938/938 [==============================] - 2s 2ms/step - loss: 0.6015 - accuracy: 0.7990
Epoch 2/15
938/938 [==============================] - 2s 2ms/step - loss: 0.4112 - accuracy: 0.8540
Epoch 3/15
938/938 [==============================] - 2s 2ms/step - loss: 0.3736 - accuracy: 0.8666
Epoch 4/15
938/938 [==============================] - 2s 2ms/step - loss: 0.3498 - accuracy: 0.8741
Epoch 5/15
938/938 [==============================] - 2s 2ms/step - loss: 0.3303 - accuracy: 0.8801
Epoch 6/15
938/938 [==============================] - 2s 2ms/step - loss: 0.3138 - accuracy: 0.8872
Epoch 7/15
938/938 [==============================] - 2s 2ms/step - loss: 0.3014 - accuracy: 0.8901
Epoch 8/15
938/938 [==============================] - 2s 2ms/step - loss: 0.2896 - accuracy: 0.8954
Epoch 9/15
938/938 [==============================] - 2s 2ms/step - loss: 0.2805 - accuracy: 0.8974
Epoch 10/15
938/938 [==============================] - 2s 2ms/step - loss: 0.2700 - accuracy: 0.9014
Epoch 11/15
938/938 [==============================] - 2s 2ms/step - loss: 0.2606 - accuracy: 0.9040
Epoch 12/15
938/938 [==============================] - 2s 2ms/step - loss: 0.2532 - accuracy: 0.9070
Epoch 13/15
938/938 [==============================] - 2s 2ms/step - loss: 0.2468 - accuracy: 0.9080
Epoch 14/15
938/938 [==============================] - 2s 2ms/step - loss: 0.2394 - accuracy: 0.9120
Epoch 15/15
938/938 [==============================] - 2s 2ms/step - loss: 0.2329 - accuracy: 0.9148
```

Out[19]:
```
<keras.callbacks.History at 0x1b93ef56d90>
```

In [20]:
```
### Tu código para 30 epoch aquí ###
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(128, activation='sigmoid'),
    keras.layers.Dense(10, activation='softmax')
])
model.compile(loss = "sparse_categorical_crossentropy" , optimizer="adam",metrics=["accuracy"])
model.fit(training_images,training_labels,batch_size = 64 , epochs = 30)
```

```
Epoch 1/30
938/938 [==============================] - 2s 2ms/step - loss: 0.5963 - accuracy: 0.8017
Epoch 2/30
938/938 [==============================] - 2s 2ms/step - loss: 0.4117 - accuracy: 0.8527
Epoch 3/30
938/938 [==============================] - 2s 2ms/step - loss: 0.3718 - accuracy: 0.8662
Epoch 4/30
938/938 [==============================] - 2s 2ms/step - loss: 0.3474 - accuracy: 0.8759
Epoch 5/30
938/938 [==============================] - 2s 2ms/step - loss: 0.3284 - accuracy: 0.8819
Epoch 6/30
938/938 [==============================] - 2s 2ms/step - loss: 0.3132 - accuracy: 0.8863
Epoch 7/30
938/938 [==============================] - 2s 2ms/step - loss: 0.2990 - accuracy: 0.8918
Epoch 8/30
938/938 [==============================] - 2s 2ms/step - loss: 0.2894 - accuracy: 0.8944
Epoch 9/30
938/938 [==============================] - 2s 2ms/step - loss: 0.2795 - accuracy: 0.8969
Epoch 10/30
938/938 [==============================] - 2s 2ms/step - loss: 0.2697 - accuracy: 0.9015
Epoch 11/30
938/938 [==============================] - 2s 2ms/step - loss: 0.2607 - accuracy: 0.9035
Epoch 12/30
938/938 [==============================] - 2s 2ms/step - loss: 0.2535 - accuracy: 0.9082
Epoch 13/30
938/938 [==============================] - 2s 2ms/step - loss: 0.2474 - accuracy: 0.9095
Epoch 14/30
938/938 [==============================] - 2s 2ms/step - loss: 0.2394 - accuracy: 0.9118
Epoch 15/30
938/938 [==============================] - 2s 2ms/step - loss: 0.2350 - accuracy: 0.9136
Epoch 16/30
938/938 [==============================] - 2s 2ms/step - loss: 0.2284 - accuracy: 0.9166
Epoch 17/30
938/938 [==============================] - 2s 2ms/step - loss: 0.2230 - accuracy: 0.9187
Epoch 18/30
938/938 [==============================] - 2s 2ms/step - loss: 0.2160 - accuracy: 0.9208
Epoch 19/30
938/938 [==============================] - 2s 2ms/step - loss: 0.2114 - accuracy: 0.9222
Epoch 20/30
938/938 [==============================] - 2s 2ms/step - loss: 0.2057 - accuracy: 0.9249
Epoch 21/30
938/938 [==============================] - 2s 2ms/step - loss: 0.2034 - accuracy: 0.9255
Epoch 22/30
938/938 [==============================] - 2s 2ms/step - loss: 0.1972 - accuracy: 0.9289
Epoch 23/30
938/938 [==============================] - 2s 2ms/step - loss: 0.1919 - accuracy: 0.9301
Epoch 24/30
938/938 [==============================] - 2s 2ms/step - loss: 0.1900 - accuracy: 0.9308
Epoch 25/30
938/938 [==============================] - 2s 2ms/step - loss: 0.1835 - accuracy: 0.9342
Epoch 26/30
938/938 [==============================] - 2s 2ms/step - loss: 0.1805 - accuracy: 0.9337
Epoch 27/30
938/938 [==============================] - 2s 2ms/step - loss: 0.1759 - accuracy: 0.9354
Epoch 28/30
938/938 [==============================] - 2s 2ms/step - loss: 0.1728 - accuracy: 0.9381
Epoch 29/30
938/938 [==============================] - 2s 2ms/step - loss: 0.1682 - accuracy: 0.9399
Epoch 30/30
938/938 [==============================] - 2s 2ms/step - loss: 0.1653 - accuracy: 0.9400
```

Out[20]:
```
<keras.callbacks.History at 0x1b93b552d60>
```

Tu respuesta a la pregunta 8.3 aquí: se considera overfitting por que llega un momento en que el resultado de la evaluacion entre loss y accuracy no cambia en absoluto, por definicion la red neuronal memoriza pero no aprende. Una buena forma de arreglar esto seria BatchNormalization o vgg16, incluso ambos.

Ejercicio 6: Early stop

En el ejercicio anterior, cuando entrenabas con epoch extras, tenías un problema en el que tu pérdida podía cambiar. Puede que te haya llevado un poco de tiempo esperar a que el entrenamiento lo hiciera, y puede que hayas pensado "¿no estaría bien si pudiera parar el entrenamiento cuando alcance un valor deseado?", es decir, una precisión del 85% podría ser suficiente para ti, y si alcanzas eso después de 3 epoch, ¿por qué sentarte a esperar a que termine muchas más épocas? Como cualquier otro programa existen formas de parar la ejecución

A partir del ejemplo de código que

se da, hacer una nueva función que tenga en cuenta la pérdida (loss) y que pueda parar el código para evitar que ocurra el efeto secundario que vimos en el ejercicio 5.

In [ ]:
```
### Ejemplo de código

class myCallback(tf.keras.callbacks.Callback):
      def on_epoch_end(self, epoch, logs={}):
            if(logs.get('accuracy')> 0.85):
                  print("\nAlcanzado el 85% de precisión, se cancela el entrenamiento!!")
                  self.model.stop_training = True
```

Pregunta 9 (2 puntos): Completa el siguiente código con una clase callback que una vez alcanzado el 40% de perdida detenga el entrenamiento.

In [22]:

```python
import tensorflow as tf
print(tf.__version__)

### Tu código de la función callback para parar el entrenamiento de la red neuronal al 40% de loss aqui: ###

class myCallback (tf.keras.callbacks.Callback):
    def error_en_epoch (self,epoch,logs={}):
        if(log.get("loss") > 0.4):
            print("perdida del 40%")
            self.model.stop_training = True

callbacks = myCallback()
mnist = tf.keras.datasets.fashion_mnist
(training_images, training_labels) ,  (test_images, test_labels) = mnist.load_data()

training_images = training_images/255.0
test_images = test_images/255.0

model = tf.keras.models.Sequential([tf.keras.layers.Flatten(),
                                    tf.keras.layers.Dense(512, activation=tf.nn.relu),
                                    tf.keras.layers.Dense(10, activation=tf.nn.softmax)])

model.compile(optimizer = 'adam',
              loss = 'sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(training_images, training_labels, epochs=50, callbacks=[callbacks])
```

```
2.8.0
Epoch 1/50
1875/1875 [==============================] - 8s 4ms/step - loss: 0.4760 - accuracy: 0.8292
Epoch 2/50
1875/1875 [==============================] - 7s 4ms/step - loss: 0.3584 - accuracy: 0.8686
Epoch 3/50
1875/1875 [==============================] - 7s 4ms/step - loss: 0.3224 - accuracy: 0.8811
Epoch 4/50
1875/1875 [==============================] - 7s 4ms/step - loss: 0.2991 - accuracy: 0.8899
Epoch 5/50
1875/1875 [==============================] - 7s 4ms/step - loss: 0.2784 - accuracy: 0.8966
Epoch 6/50
1875/1875 [==============================] - 7s 4ms/step - loss: 0.2651 - accuracy: 0.9013
Epoch 7/50
1875/1875 [==============================] - 7s 4ms/step - loss: 0.2524 - accuracy: 0.9056
Epoch 8/50
1875/1875 [==============================] - 7s 4ms/step - loss: 0.2415 - accuracy: 0.9100
Epoch 9/50
1875/1875 [==============================] - 7s 4ms/step - loss: 0.2301 - accuracy: 0.9149
Epoch 10/50
1875/1875 [==============================] - 7s 4ms/step - loss: 0.2210 - accuracy: 0.9174
Epoch 11/50
1875/1875 [==============================] - 7s 4ms/step - loss: 0.2127 - accuracy: 0.9197
Epoch 12/50
1875/1875 [==============================] - 7s 4ms/step - loss: 0.2028 - accuracy: 0.9230
Epoch 13/50
1875/1875 [==============================] - 7s 4ms/step - loss: 0.1970 - accuracy: 0.9251
Epoch 14/50
1875/1875 [==============================] - 7s 4ms/step - loss: 0.1903 - accuracy: 0.9281
Epoch 15/50
1875/1875 [==============================] - 7s 4ms/step - loss: 0.1848 - accuracy: 0.9303
Epoch 16/50
1875/1875 [==============================] - 7s 4ms/step - loss: 0.1799 - accuracy: 0.9319
Epoch 17/50
1875/1875 [==============================] - 7s 4ms/step - loss: 0.1734 - accuracy: 0.9336
Epoch 18/50
1875/1875 [==============================] - 7s 4ms/step - loss: 0.1681 - accuracy: 0.9376
Epoch 19/50
1875/1875 [==============================] - 7s 4ms/step - loss: 0.1619 - accuracy: 0.9388
Epoch 20/50
1875/1875 [==============================] - 7s 4ms/step - loss: 0.1587 - accuracy: 0.9395
Epoch 21/50
1875/1875 [==============================] - 7s 4ms/step - loss: 0.1537 - accuracy: 0.9414
Epoch 22/50
1875/1875 [==============================] - 7s 4ms/step - loss: 0.1475 - accuracy: 0.9445
Epoch 23/50
1875/1875 [==============================] - 7s 4ms/step - loss: 0.1432 - accuracy: 0.9468
Epoch 24/50
1875/1875 [==============================] - 7s 4ms/step - loss: 0.1392 - accuracy: 0.9478
Epoch 25/50
1875/1875 [==============================] - 7s 4ms/step - loss: 0.1394 - accuracy: 0.9464
Epoch 26/50
1875/1875 [==============================] - 7s 4ms/step - loss: 0.1329 - accuracy: 0.9494
Epoch 27/50
1875/1875 [==============================] - 7s 4ms/step - loss: 0.1285 - accuracy: 0.9515
Epoch 28/50
1875/1875 [==============================] - 7s 4ms/step - loss: 0.1271 - accuracy: 0.9521
Epoch 29/50
1875/1875 [==============================] - 7s 4ms/step - loss: 0.1255 - accuracy: 0.9522
Epoch 30/50
1875/1875 [==============================] - 7s 4ms/step - loss: 0.1192 - accuracy: 0.9546
Epoch 31/50
1875/1875 [==============================] - 7s 4ms/step - loss: 0.1162 - accuracy: 0.9563
Epoch 32/50
1875/1875 [==============================] - 7s 4ms/step - loss: 0.1148 - accuracy: 0.9574
Epoch 33/50
1875/1875 [==============================] - 7s 4ms/step - loss: 0.1126 - accuracy: 0.9578
Epoch 34/50
1875/1875 [==============================] - 7s 4ms/step - loss: 0.1080 - accuracy: 0.9589
Epoch 35/50
1875/1875 [==============================] - 7s 4ms/step - loss: 0.1047 - accuracy: 0.9602
Epoch 36/50
1875/1875 [==============================] - 7s 4ms/step - loss: 0.1042 - accuracy: 0.9615
Epoch 37/50
1875/1875 [==============================] - 7s 4ms/step - loss: 0.1009 - accuracy: 0.9612
Epoch 38/50
1875/1875 [==============================] - 7s 4ms/step - loss: 0.0990 - accuracy: 0.9622
Epoch 39/50
1875/1875 [==============================] - 7s 4ms/step - loss: 0.0995 - accuracy: 0.9621
Epoch 40/50
1875/1875 [==============================] - 7s 4ms/step - loss: 0.0941 - accuracy: 0.9641
Epoch 41/50
1875/1875 [==============================] - 7s 4ms/step - loss: 0.0957 - accuracy: 0.9641
Epoch 42/50
1875/1875 [==============================] - 7s 4ms/step - loss: 0.0911 - accuracy: 0.9657
Epoch 43/50
1875/1875 [==============================] - 7s 4ms/step - loss: 0.0918 - accuracy: 0.9654
Epoch 44/50
1875/1875 [==============================] - 7s 4ms/step - loss: 0.0897 - accuracy: 0.9657
Epoch 45/50
1875/1875 [==============================] - 7s 4ms/step - loss: 0.0866 - accuracy: 0.9671
Epoch 46/50
1875/1875 [==============================] - 7s 4ms/step - loss: 0.0840 - accuracy: 0.9675
Epoch 47/50
1875/1875 [==============================] - 7s 4ms/step - loss: 0.0858 - accuracy: 0.9684
Epoch 48/50
1875/1875 [==============================] - 7s 4ms/step - loss: 0.0822 - accuracy: 0.9701
Epoch 49/50
1875/1875 [==============================] - 7s 4ms/step - loss: 0.0792 - accuracy: 0.9701
Epoch 50/50
1875/1875 [==============================] - 7s 4ms/step - loss: 0.0818 - accuracy: 0.9685
```

Out[22]:

```
<keras.callbacks.History at 0x1b90b628c10>
```