Redes neuronales profundas

Ernesto Reynoso Lizárraga A01639915

Importamos librerias

```
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt
```

Preparamos el dataset

Exportamos los datos desde keras y los normalizamos

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

#Normalizar
train_images, test_images = train_images/255.0, test_images/255.0
```

Validadción de datos

Establecemos los nombres de las clases y los relacionamos con las imagenes de la base de datos

```
class_names=['t-shirt/top', 'trouser', 'pullover','dress','coat','sandal','shirt','sneaker','bag','anckle boot']
plt.figure(figsize=(10,10))
for i in range(25):
  plt.subplot(5,5,i+1)
  plt.xticks([])
  plt.yticks([])
  plt.grid(False)
  plt.imshow(train_images[i])
  plt.xlabel(class_names[train_labels[i]])
plt.show()
```

Creamos las capas de convolución

```
model = models.Sequential()
model.add(layers.Conv2D(32,(3,3), activation='relu', input_shape=(28,28,1)))
model.add(layers.MaxPooling2D(2,2))
model.add(layers.Conv2D(128,(3,3), activation='relu'))
model.add(layers.MaxPooling2D(2,2))
model.add(layers.Conv2D(128,(3,3), activation='relu'))
```

```
model.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_13 (Conv2D)          (None, 26, 26, 32)        320

 max_pooling2d_9 (MaxPoolin  (None, 13, 13, 32)        0
 g2D)

 conv2d_14 (Conv2D)          (None, 11, 11, 128)       36992

 max_pooling2d_10 (MaxPooli  (None, 5, 5, 128)         0
 ng2D)

 conv2d_15 (Conv2D)          (None, 3, 3, 128)         147584

=================================================================
Total params: 184896 (722.25 KB)
Trainable params: 184896 (722.25 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Capas densas

```
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='sigmoid'))
```

```
model.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
```

```
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_13 (Conv2D)          (None, 26, 26, 32)        320

 max_pooling2d_9 (MaxPoolin  (None, 13, 13, 32)        0
 g2D)

 conv2d_14 (Conv2D)          (None, 11, 11, 128)       36992

 max_pooling2d_10 (MaxPooli  (None, 5, 5, 128)         0
 ng2D)

 conv2d_15 (Conv2D)          (None, 3, 3, 128)         147584

 flatten_4 (Flatten)         (None, 1152)              0

 dense_8 (Dense)             (None, 64)                73792

 dense_9 (Dense)             (None, 10)                650

=================================================================
Total params: 259338 (1013.04 KB)
Trainable params: 259338 (1013.04 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

entrenamos el modelo

entrenamos el modelo midiendo la precision del mismo y establecemos 10 epocas

```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

history = model.fit(train_images,train_labels, epochs=10, validation_data=(test_images, test_labels))
```

```
Epoch 1/10
/usr/local/lib/python3.10/dist-packages/keras/src/backend.py:5714: UserWarning: "`sparse_categorical_crossentropy` received `from_logits=True`, but the `output` argument was produced by a Softmax activati
  output, from_logits = _get_logits(
1875/1875 [==============================] - 102s 54ms/step - loss: 0.4766 - accuracy: 0.8253 - val_loss: 0.3632 - val_accuracy: 0.8681
Epoch 2/10
1875/1875 [==============================] - 98s 52ms/step - loss: 0.3046 - accuracy: 0.8884 - val_loss: 0.3009 - val_accuracy: 0.8915
Epoch 3/10
1875/1875 [==============================] - 99s 53ms/step - loss: 0.2592 - accuracy: 0.9051 - val_loss: 0.2756 - val_accuracy: 0.8965
Epoch 4/10
1875/1875 [==============================] - 96s 51ms/step - loss: 0.2227 - accuracy: 0.9177 - val_loss: 0.2622 - val_accuracy: 0.9054
Epoch 5/10
1875/1875 [==============================] - 97s 52ms/step - loss: 0.1988 - accuracy: 0.9257 - val_loss: 0.2682 - val_accuracy: 0.9065
Epoch 6/10
1875/1875 [==============================] - 97s 52ms/step - loss: 0.1729 - accuracy: 0.9358 - val_loss: 0.2828 - val_accuracy: 0.9038
Epoch 7/10
1875/1875 [==============================] - 99s 53ms/step - loss: 0.1547 - accuracy: 0.9425 - val_loss: 0.2635 - val_accuracy: 0.9081
Epoch 8/10
1875/1875 [==============================] - 98s 52ms/step - loss: 0.1377 - accuracy: 0.9483 - val_loss: 0.2887 - val_accuracy: 0.9088
Epoch 9/10
1875/1875 [==============================] - 98s 52ms/step - loss: 0.1217 - accuracy: 0.9536 - val_loss: 0.2792 - val_accuracy: 0.9158
Epoch 10/10
1875/1875 [==============================] - 97s 52ms/step - loss: 0.1088 - accuracy: 0.9602 - val_loss: 0.2827 - val_accuracy: 0.9117
```

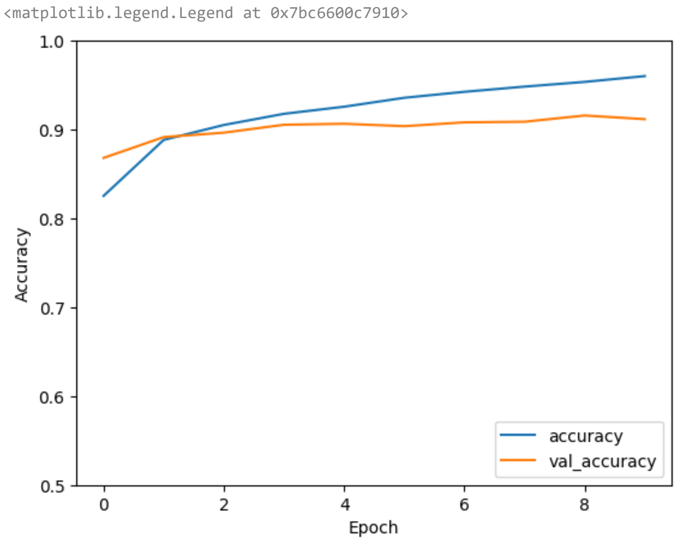
Evaluamos el modelo

```
Visualizamos los resultados de las epocas y ponemos a prueba el modelo
intentando predecir el nombre de una imagen en concreto
```

Visualizamos los resultados de las epocas y ponemos a prueba el modelo intentando predecir el nombre de una imagen en concreto

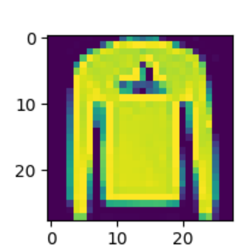
```
plt.plot(history.history['accuracy'],label='accuracy')
plt.plot(history.history['val_accuracy'],label='val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.5,1])
plt.legend(loc='lower right')
```

```
<matplotlib.legend.Legend at 0x7bc6600c7910>
```




```
n = 115

plt.figure(figsize=(2,2))
plt.imshow(test_images[n])
plt.xlabel(class_names[test_labels[n]])
plt.show()
```




```
predictions = model.predict(test_images)
print(predictions[n])

import numpy as np

print("La imagen pertenece al grupo {} con una probabilidad de {:2f}%". format(class_names[np.argmax(predictions[n])], 100 * np.max(predictions[n])))
```

```
313/313 [==============================] - 5s 15ms/step
[5.3903468e-02 5.3081413e-07 9.9999994e-01 4.3123856e-04 9.9437952e-01
 3.4240064e-09 9.9965751e-01 2.7647396e-12 6.2894074e-08 4.5285420e-09]
La imagen pertenece al grupo pullover con una probabilidad de 99.999994%
```