"""  
En este cuaderno haremos una breve incursión a las funcionalidades que nos ofrece Tensorboard   
para realizar la monitorización del aprendizaje de nuestro modelo  
"""  
# Empezamos cargando la extesión de esta librería para trabajar con Notas  
%load\_ext tensorboard

# Importamos tensorflow y el módulo temporal de Python  
import tensorflow as tf  
  
# Cargamos los datos de conjuno MNIST  
mnist = tf.keras.datasets.mnist

# Generamos los conjuntos para entrenamiento y validación  
(x\_train, y\_train),(x\_test, y\_test) = mnist.load\_data()  
print("X train:", x\_train.shape, ", Y train:\n", y\_train.shape)  
print("X test:", x\_test.shape, ", Y test:\n", y\_test.shape)

X train: (60000, 28, 28) , Y train:  
 (60000,)  
X test: (10000, 28, 28) , Y test:  
 (10000,)

# Normalizamos los valores al intervalo 0-255  
import numpy as np  
print("A:", np.min(x\_train), np.max(x\_train))  
  
x\_train, x\_test = x\_train / 255.0, x\_test / 255.0  
  
print("B:", np.min(x\_train), np.max(x\_train))  
  
# TODO Ver necesidad de esta normalización

A: 0.0 6.030862941101084e-08  
B: 0.0 2.365044290627876e-10

# Con la siguiente función construimos el modelo que emplearemos para clasificar   
def create\_model():  
 return tf.keras.models.Sequential([  
 tf.keras.layers.Flatten(input\_shape=(28, 28)),  
 tf.keras.layers.Dense(512, activation='relu'),  
 tf.keras.layers.Dropout(0.2),  
 tf.keras.layers.Dense(10, activation='softmax')  
 ])  
  
# Revisemos la estructura de este modelo:  
# TODO Explicar

# Creamos una instancia del modelo que acabamos de definir  
model = create\_model()  
# Llamamos a la función compilador para TODO  
model.compile(optimizer='adam',  
 loss='sparse\_categorical\_crossentropy',  
 metrics=['accuracy'])

# La función summary nos da alguna información sobre la arquitectura y número de parámetros de nuestra red  
model.summary()

Model: "sequential"  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
Layer (type) Output Shape Param #   
=================================================================  
flatten (Flatten) (None, 784) 0   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
dense (Dense) (None, 512) 401920   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
dropout (Dropout) (None, 512) 0   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
dense\_1 (Dense) (None, 10) 5130   
=================================================================  
Total params: 407,050  
Trainable params: 407,050  
Non-trainable params: 0  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# En una capa densa, los parámetros que se ajustan obecen la sigiente regla  
# ==> num\_params = (input\_size + bias) \* output\_size, bias = 1  
# Como vemos, la primera capa lee un array de 784 valores = 28 \* 28 que son las dimensiones de nuestras imágenes de entrada  
print("Valores de entrada:", 28 \* 28)   
# La segunda capa de la red es una capa de activación en la que introducimos los 784 valores cargados para entrenamiento con ReLu  
print("Parámetros de entrenamiento:", (784 + 1) \* 512)  
# El paso Dropout elimina elementos de la red con una probabilidad del 20%  
# Finalmente tenemos una capa Dense en la que entran 512 + 1 parámetros  
print("Parámetros de entrenamiento:", (784 + 1) \* 512)

# Configuramos el entorno para nuestra pizarra   
import datetime  
log\_dir = "logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")  
tensorboard\_callback = tf.keras.callbacks.TensorBoard(log\_dir=log\_dir, histogram\_freq=1)

# Entrenamos nuestro modelo sobre los datos de entrenamiento y pasamos la pizarra TODO  
model.fit(x=x\_train,   
 y=y\_train,   
 epochs=5,   
 validation\_data=(x\_test, y\_test),   
 callbacks=[tensorboard\_callback])

Epoch 1/5  
1875/1875 [==============================] - 14s 7ms/step - loss: 2.3015 - accuracy: 0.1117 - val\_loss: 2.3014 - val\_accuracy: 0.1135  
Epoch 2/5  
1875/1875 [==============================] - 13s 7ms/step - loss: 2.3014 - accuracy: 0.1124 - val\_loss: 2.3011 - val\_accuracy: 0.1135  
Epoch 3/5  
1875/1875 [==============================] - 13s 7ms/step - loss: 2.3014 - accuracy: 0.1124 - val\_loss: 2.3011 - val\_accuracy: 0.1135  
Epoch 4/5  
1875/1875 [==============================] - 13s 7ms/step - loss: 2.3014 - accuracy: 0.1124 - val\_loss: 2.3011 - val\_accuracy: 0.1135  
Epoch 5/5  
1875/1875 [==============================] - 13s 7ms/step - loss: 2.3014 - accuracy: 0.1124 - val\_loss: 2.3010 - val\_accuracy: 0.1135  
  
  
  
  
  
<keras.callbacks.History at 0x7f8c2e0ecad0>

# Terminado el proceso de entrenamiento de nuestro modelo resisaremos como han evolucionado las métricas y paraámetros XXX  
%tensorboard --logdir logs/fit

<IPython.core.display.Javascript object>