# Actividad Redes Neuronales Profundas Ejercicio 2

## Mario Alberto Castañeda Martínez - A01640152

**En este segundo ejercicio, se creará un modelo de redes neuronales profundas para el dataset Fashion_nmist de Tensorflow. Se probarán diferentes modificaciones del modelo para encontrar el mejor accuracy posible.**

Primero se importan las librerías necesarias:

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

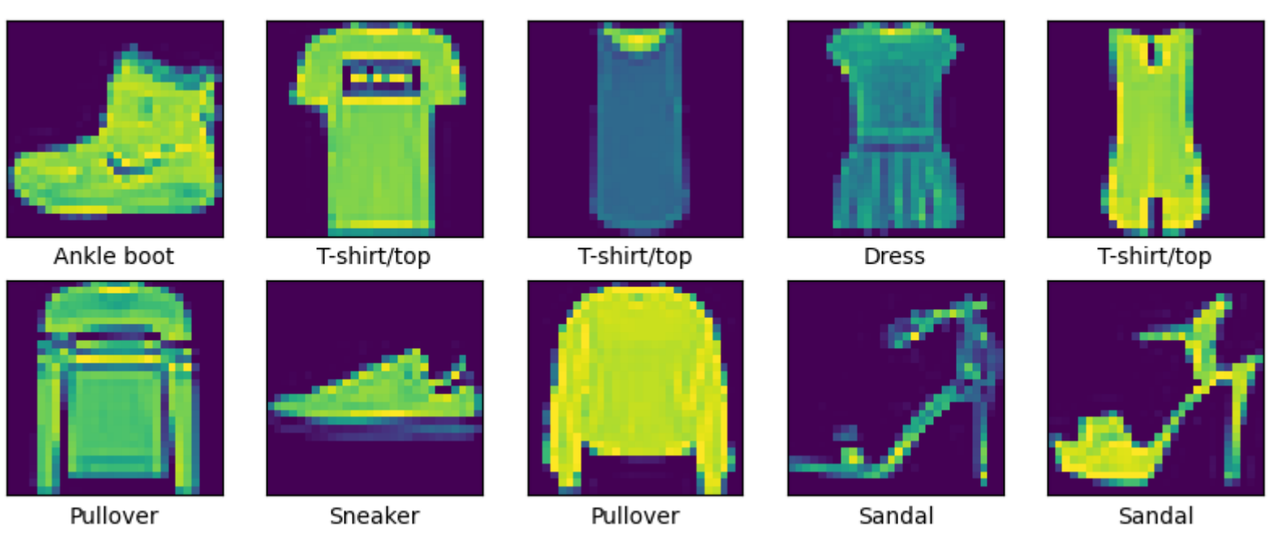
Se carga el dataset y se definen los conjuntos de entrenamiento y de prueba:

```
#prepara dataset Fashion_mnist
(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
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-idx1-ubyte.gz
29515/29515 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-idx3-ubyte.gz
26421880/26421880 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-labels-idx1-ubyte.gz
5148/5148 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-images-idx3-ubyte.gz
4422102/4422102 [==============================] - 0s 0us/step
```

Se definen las clases del dataset y se muestran en sus respectivas imágenes:

```
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat', 'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
plt.figure(figsize=(10,10))
for i in range (10):
  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()
```




Se agregan capas de convolución, filtros y la función de activación:

En este caso, solo se agregaron dos capas porque con dos capas se consigue un accuracy alto y al haber realizado pruebas con 3 capas, se sigue consiguiendo el mismo accuracy e incluso en ocasiones, se obtenía un accuracy un poco menor.

```
#Capas de convolución
model = models.Sequential()
model.add(layers.Conv2D(32, (3,3), activation='relu', input_shape=(28, 28, 1))) # el input_shape se refiere a la entrada de las imagenes, 28x28 pixeles, un canal de color
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.summary()
```

```
Model: "sequential_10"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_28 (Conv2D)          (None, 26, 26, 32)        320

 max_pooling2d_19 (MaxPooli  (None, 13, 13, 32)        0
 ng2D)

 conv2d_29 (Conv2D)          (None, 11, 11, 64)        18496

 max_pooling2d_20 (MaxPooli  (None, 5, 5, 64)          0
 ng2D)

=================================================================
Total params: 18816 (73.50 KB)
Trainable params: 18816 (73.50 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Se agrega la parte de aplanamiento de la salida de las capas convolucionales, para después agregar uan capa densa con 128 neuronas con activación Relu y una capa de salida con activación sigmoide:

```
model.add(layers.Flatten())
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dense(10, activation ='sigmoid'))
model.summary()
```

```
Model: "sequential_10"
_________________________________________________________________
```

```
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_28 (Conv2D)          (None, 26, 26, 32)        320

 max_pooling2d_19 (MaxPooli  (None, 13, 13, 32)        0
 ng2D)

 conv2d_29 (Conv2D)          (None, 11, 11, 64)        18496

 max_pooling2d_20 (MaxPooli  (None, 5, 5, 64)          0
 ng2D)

 flatten_9 (Flatten)         (None, 1600)              0

 dense_18 (Dense)            (None, 128)               204928

 dense_19 (Dense)            (None, 10)                1290

=================================================================
Total params: 225034 (879.04 KB)
Trainable params: 225034 (879.04 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Se compila el modelo, en este caso se entrena en 20 épocas y se agrega el conjunto de validación.

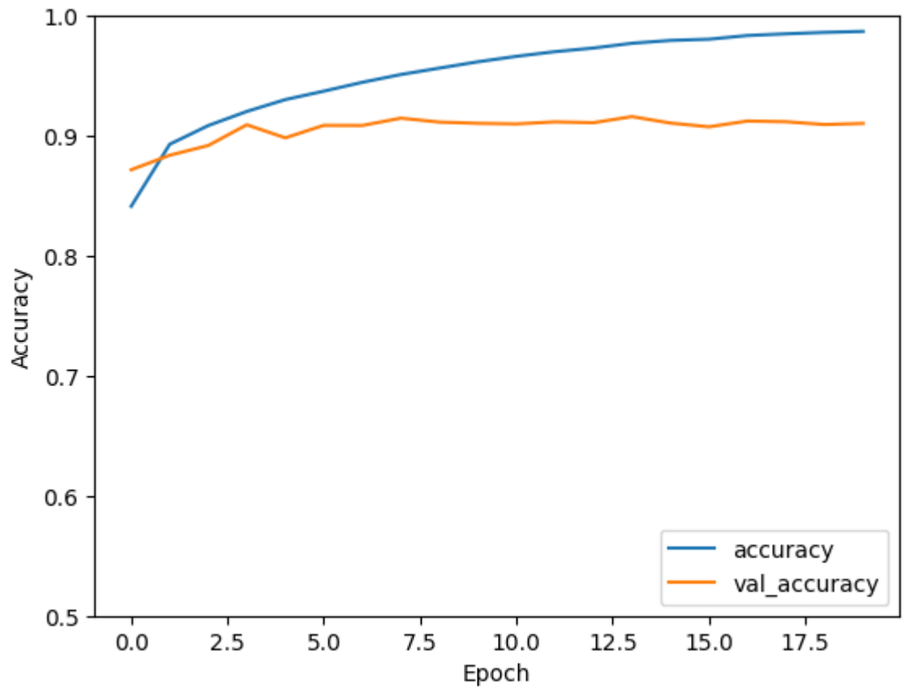
```
model.compile(optimizer='adam',loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),metrics=['accuracy'])
history = model.fit(train_images, train_labels, epochs=20, validation_data=(test_images, test_labels))
```

```
Epoch 1/20
1875/1875 [==============================] - 36s 19ms/step - loss: 0.4400 - accuracy: 0.8413 - val_loss: 0.3610 - val_accuracy: 0.8717
Epoch 2/20
1875/1875 [==============================] - 35s 18ms/step - loss: 0.2917 - accuracy: 0.8929 - val_loss: 0.3234 - val_accuracy: 0.8838
Epoch 3/20
1875/1875 [==============================] - 36s 19ms/step - loss: 0.2473 - accuracy: 0.9085 - val_loss: 0.2856 - val_accuracy: 0.8919
Epoch 4/20
1875/1875 [==============================] - 35s 19ms/step - loss: 0.2170 - accuracy: 0.9203 - val_loss: 0.2547 - val_accuracy: 0.9092
Epoch 5/20
1875/1875 [==============================] - 34s 18ms/step - loss: 0.1902 - accuracy: 0.9301 - val_loss: 0.2791 - val_accuracy: 0.8983
Epoch 6/20
1875/1875 [==============================] - 34s 18ms/step - loss: 0.1684 - accuracy: 0.9372 - val_loss: 0.2577 - val_accuracy: 0.9087
Epoch 7/20
1875/1875 [==============================] - 35s 19ms/step - loss: 0.1488 - accuracy: 0.9446 - val_loss: 0.2677 - val_accuracy: 0.9086
Epoch 8/20
1875/1875 [==============================] - 35s 18ms/step - loss: 0.1318 - accuracy: 0.9511 - val_loss: 0.2587 - val_accuracy: 0.9147
Epoch 9/20
1875/1875 [==============================] - 36s 19ms/step - loss: 0.1161 - accuracy: 0.9564 - val_loss: 0.2785 - val_accuracy: 0.9114
Epoch 10/20
1875/1875 [==============================] - 35s 18ms/step - loss: 0.1022 - accuracy: 0.9617 - val_loss: 0.3126 - val_accuracy: 0.9104
Epoch 11/20
1875/1875 [==============================] - 40s 21ms/step - loss: 0.0898 - accuracy: 0.9662 - val_loss: 0.3262 - val_accuracy: 0.9099
Epoch 12/20
1875/1875 [==============================] - 35s 19ms/step - loss: 0.0785 - accuracy: 0.9701 - val_loss: 0.3231 - val_accuracy: 0.9116
Epoch 13/20
1875/1875 [==============================] - 34s 18ms/step - loss: 0.0709 - accuracy: 0.9731 - val_loss: 0.3504 - val_accuracy: 0.9109
Epoch 14/20
1875/1875 [==============================] - 35s 19ms/step - loss: 0.0602 - accuracy: 0.9771 - val_loss: 0.3713 - val_accuracy: 0.9160
Epoch 15/20
1875/1875 [==============================] - 35s 19ms/step - loss: 0.0540 - accuracy: 0.9795 - val_loss: 0.3926 - val_accuracy: 0.9106
Epoch 16/20
1875/1875 [==============================] - 35s 19ms/step - loss: 0.0514 - accuracy: 0.9805 - val_loss: 0.3991 - val_accuracy: 0.9075
Epoch 17/20
1875/1875 [==============================] - 36s 19ms/step - loss: 0.0429 - accuracy: 0.9836 - val_loss: 0.4513 - val_accuracy: 0.9123
Epoch 18/20
1875/1875 [==============================] - 34s 18ms/step - loss: 0.0425 - accuracy: 0.9850 - val_loss: 0.4505 - val_accuracy: 0.9117
Epoch 19/20
1875/1875 [==============================] - 35s 19ms/step - loss: 0.0377 - accuracy: 0.9862 - val_loss: 0.5220 - val_accuracy: 0.9094
Epoch 20/20
1875/1875 [==============================] - 33s 18ms/step - loss: 0.0349 - accuracy: 0.9870 - val_loss: 0.5242 - val_accuracy: 0.9103
```

Después de generar el modelo, se visualiza el accuracy con los datos de entrenamiento y el accuracy para nuevas imágenes.

```
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')
test_loss, test_acc = model.evaluate(test_images, test_labels, verbose=2)
```

```
313/313 - 2s - loss: 0.5242 - accuracy: 0.9103 - 2s/epoch - 5ms/step
```



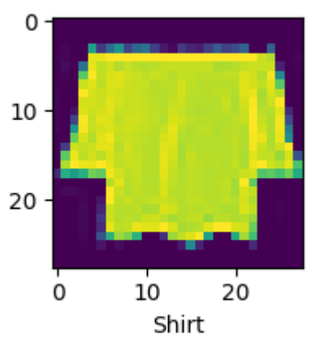

```
print(test_acc)
```

```
0.9103000164031982
```

**Este accuracy obtenido indica que el modelo es muy bueno para pronosticar nuevas imágenes, cabe destacar que durante las diferentes pruebas de modelos, el accuracy siempre se mantenía entre el 90 y el 91% de accuracy. Se debe igualmente señalar que los modelos en general indicaban un cierto sobreajuste para los datos o imágenes de entrenamiento, a pesar de esto, se logró encontrar un modelo con un accuracy para nuevas imágenes muy bueno. Es importante recalcar que se probaron distintas modificaciones del modelo (más filtros, menos neuronas, menos épocas) y en general los resultados siempre eran del 90%, con el modelo actual, se logró alcanzar un accuracy del 91%.**

Se prueba el modelo con una imagen:

```
n=5002 ##Number of image
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])
print("This image most likely belongs to {} with a {:.2f} percent confidence.".format(class_names[np.argmax(predictions[n])], 100 * np.max(predictions[n])))
```

```
313/313 [==============================] - 2s 6ms/step
[2.4693267e-02 5.4905080e-10 1.3979074e-07 1.2409546e-06 3.9001578e-03
 1.0414582e-13 9.9994624e-01 6.5085964e-10 1.1575962e-04 7.3319905e-05]
This image most likely belongs to Shirt with a 99.99 percent confidence.
```

El modelo es capaz de identificar la imagen correctamente.