Red neuronal con Tensorflow

INTELIGENCIA ARTIFICIAL

Litzy Yulissa Nevarez García

La librería por excelencia para trabajar con redes neuronales es Tensorflow, de Google. Esta es de código abierto y permite desarrollar nuestra red neuronal en lenguajes como Python, Go, Java y C. En este ejemplo usaremos python. Desarrollar y entrenar una red neuronal usando el siguiente Dataset Fassion MNIST, que nos servira para que nuestra red neural pueda clasificar los tipos de prenda que le daremos con nuestro Dataset. Este dataset está formado por un conjunto de imagenes de prendas de ropa categorizadas según el tipo de prenda, jersey, bolso, camiseta. En el dataset hay un total de 10.000 imágenes con su correspondiente etiqueta y un total de 10 etiquetas de ropa diferentes

In [1]:
```
#dataset Fashion MNIST
from tensorflow.keras.datasets
import fashion_mnist
(X, y), (X_test, y_test) = fashion_mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/tra
in-labels-idx1-ubyte.gz
29515/29515 [==============================] - 0s 4us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/tra
in-images-idx3-ubyte.gz
26421880/26421880 [==============================] - 86s 3us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10
k-labels-idx1-ubyte.gz
5148/5148 [==============================] - 0s 0s/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10
k-images-idx3-ubyte.gz
4422102/4422102 [==============================] - 17s 4us/step
```

In [4]:
```
# Cargamos los labels del dataset.
import matplotlib.pyplot as plt
import numpy as np
labels = ["T-shirt/top",
          "Trouser",
          "Pullover",
          "Dress",
          "Coat",
          "Sandal",
          "Shirt",
          "Sneaker",
          "Bag",
          "Ankle boot"]

# Mostramos una tabla con las algunas imagenes del dataset
plt.figure(figsize=(14,8))
ind = np.random.choice(X.shape[0],20)
for i,img in enumerate(ind):
    plt.subplot(5,10,i+1)
    plt.title(labels[y[img]])
    plt.imshow(X[img], cmap="binary")
    plt.axis("off")
```

Preparando el dataset

Ahora que ya conocemos los datos con los que vamos a trabajar para crear nuestra primera red neuronal. Importemos entonces las bibliotecas necesarias para empezar el diseño de la topología y el entrenamiento de nuestra primera red neuronal.

In [17]:
```
# TensorFlow y tf.keras
import tensorflow as tf
from tensorflow import keras
```

La primera tares que realizaremos será la de separar los datos de nuestro dataset en dos conjuntos, uno para que pueda entrenarse y otro para testear luego su rendimiento. X_train son las imágenes de 28x28 píxeles de entrenamiento y X_test las de test.

In [18]:
```
# Dividimos el dataset en dos partes 10% de las imagenes totales serán para testeo y e

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.1)

print("Imágenes de entrenamiento", X_train.shape)
print("Imágenes de test", X_test.shape)
```

```
Imágenes de entrenamiento (54000, 28, 28)
Imágenes de test (6000, 28, 28)
```

La red

A continuación vamos a definir como serán las neuronas que compondrán la topología de nuestra red. Empecemos por el principio, la red que hemos diseñado consta de 3 capas. Este sería el código que define nuestro modelo de red neuronal.

In [19]:
```
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(128, activation='relu'),
    keras.layers.Dense(10, activation='softmax')
])
```

La primera capa Flatten realiza un "aplanado" de la imagen en 2D en de 28x28 píxeles de nuestro dataset y crea un vector de 1 dimensión de 784 parámetros. Las dos capas Dense que tenemos a continuación. En ella se definen un total de 128 neuronas cuya salida está conectada con la siguiente capa de 10 neuronas. Es decir, cada una de las 128 neuronas conecta su salida con cada una de las 10 de la siguiente capa formando un total de 128·10=1280 conexiones.

Imagina que esas 10 neuronas de la capa final son 10 bombillas y que cada bombilla representa una etiqueta de prenda de ropa de nuestro dataset. En este caso, la red encenderá la bombilla "Camiseta" cuando a la entrada le enseñemos una imagen de una camiseta.

Pero para que eso sea posible, antes tenemos que entrenar la red neuronal.

In [21]:

```
# Configuramos como se entrenará la red
model.compile(
    loss="sparse_categorical_crossentropy",
    optimizer=keras.optimizers.Adam(learning_rate=0.001),
    metrics=["accuracy"]
)

# Definimos los parametros de entrenamiento
params = {
    "validation_data": (X_val, y_val),
    "epochs": 100,
    "verbose": 2,
    "batch_size": 256,
}

# Iniciamos el entrenamiento
model.fit(X_train, y_train, **params)
```

```
Epoch 1/100
211/211 - 2s - loss: 13.4051 - accuracy: 0.7281 - val_loss: 4.3516 - val_accuracy: 0.
7592 - 2s/epoch - 11ms/step
Epoch 2/100
211/211 - 1s - loss: 3.1594 - accuracy: 0.7787 - val_loss: 1.9397 - val_accuracy: 0.7
680 - 1s/epoch - 5ms/step
Epoch 3/100
211/211 - 1s - loss: 1.1513 - accuracy: 0.7388 - val_loss: 0.8028 - val_accuracy: 0.7
353 - 1s/epoch - 5ms/step
Epoch 4/100
211/211 - 1s - loss: 0.7460 - accuracy: 0.7511 - val_loss: 0.6507 - val_accuracy: 0.7
667 - 1s/epoch - 5ms/step
Epoch 5/100
211/211 - 1s - loss: 0.6285 - accuracy: 0.7748 - val_loss: 0.5734 - val_accuracy: 0.7
893 - 1s/epoch - 5ms/step
Epoch 6/100
211/211 - 1s - loss: 0.5568 - accuracy: 0.7947 - val_loss: 0.5181 - val_accuracy: 0.8
070 - 1s/epoch - 5ms/step
Epoch 7/100
211/211 - 1s - loss: 0.5092 - accuracy: 0.8084 - val_loss: 0.4935 - val_accuracy: 0.8
112 - 1s/epoch - 6ms/step
Epoch 8/100
211/211 - 1s - loss: 0.4800 - accuracy: 0.8188 - val_loss: 0.4611 - val_accuracy: 0.8
238 - 1s/epoch - 5ms/step
Epoch 9/100
211/211 - 1s - loss: 0.4529 - accuracy: 0.8266 - val_loss: 0.4455 - val_accuracy: 0.8
323 - 1s/epoch - 5ms/step
Epoch 10/100
211/211 - 1s - loss: 0.4255 - accuracy: 0.8453 - val_loss: 0.4138 - val_accuracy: 0.8
437 - 1s/epoch - 5ms/step
Epoch 11/100
211/211 - 1s - loss: 0.4065 - accuracy: 0.8517 - val_loss: 0.4300 - val_accuracy: 0.8
437 - 1s/epoch - 5ms/step
Epoch 12/100
211/211 - 1s - loss: 0.3845 - accuracy: 0.8597 - val_loss: 0.3852 - val_accuracy: 0.8
597 - 1s/epoch - 5ms/step
Epoch 13/100
211/211 - 1s - loss: 0.3827 - accuracy: 0.8617 - val_loss: 0.3703 - val_accuracy: 0.8
650 - 1s/epoch - 5ms/step
Epoch 14/100
211/211 - 1s - loss: 0.3736 - accuracy: 0.8634 - val_loss: 0.3682 - val_accuracy: 0.8
678 - 1s/epoch - 6ms/step
Epoch 15/100
211/211 - 1s - loss: 0.3633 - accuracy: 0.8669 - val_loss: 0.3723 - val_accuracy: 0.8
645 - 1s/epoch - 6ms/step
Epoch 16/100
211/211 - 1s - loss: 0.3563 - accuracy: 0.8693 - val_loss: 0.3540 - val_accuracy: 0.8
732 - 1s/epoch - 6ms/step
Epoch 17/100
211/211 - 1s - loss: 0.3537 - accuracy: 0.8704 - val_loss: 0.3533 - val_accuracy: 0.8
743 - 1s/epoch - 6ms/step
Epoch 18/100
211/211 - 1s - loss: 0.3496 - accuracy: 0.8726 - val_loss: 0.3793 - val_accuracy: 0.8
618 - 1s/epoch - 6ms/step
Epoch 19/100
211/211 - 1s - loss: 0.3523 - accuracy: 0.8710 - val_loss: 0.3525 - val_accuracy: 0.8
718 - 1s/epoch - 6ms/step
Epoch 20/100
211/211 - 1s - loss: 0.3459 - accuracy: 0.8724 - val_loss: 0.3407 - val_accuracy: 0.8
762 - 1s/epoch - 6ms/step
```

```
Epoch 21/100
211/211 - 1s - loss: 0.3447 - accuracy: 0.8731 - val_loss: 0.3393 - val_accuracy: 0.8
770 - 1s/epoch - 6ms/step
Epoch 22/100
211/211 - 1s - loss: 0.3357 - accuracy: 0.8755 - val_loss: 0.3617 - val_accuracy: 0.8
702 - 1s/epoch - 5ms/step
Epoch 23/100
211/211 - 1s - loss: 0.3365 - accuracy: 0.8754 - val_loss: 0.3270 - val_accuracy: 0.8
785 - 1s/epoch - 6ms/step
Epoch 24/100
211/211 - 1s - loss: 0.3363 - accuracy: 0.8755 - val_loss: 0.3572 - val_accuracy: 0.8
660 - 1s/epoch - 6ms/step
Epoch 25/100
211/211 - 1s - loss: 0.3345 - accuracy: 0.8771 - val_loss: 0.3272 - val_accuracy: 0.8
815 - 1s/epoch - 6ms/step
Epoch 26/100
211/211 - 1s - loss: 0.3357 - accuracy: 0.8770 - val_loss: 0.3284 - val_accuracy: 0.8
773 - 1s/epoch - 5ms/step
Epoch 27/100
211/211 - 1s - loss: 0.3381 - accuracy: 0.8757 - val_loss: 0.3322 - val_accuracy: 0.8
828 - 1s/epoch - 5ms/step
Epoch 28/100
211/211 - 1s - loss: 0.3297 - accuracy: 0.8788 - val_loss: 0.3331 - val_accuracy: 0.8
777 - 1s/epoch - 5ms/step
Epoch 29/100
211/211 - 1s - loss: 0.3225 - accuracy: 0.8793 - val_loss: 0.3226 - val_accuracy: 0.8
875 - 1s/epoch - 5ms/step
Epoch 30/100
211/211 - 1s - loss: 0.3213 - accuracy: 0.8804 - val_loss: 0.3224 - val_accuracy: 0.8
833 - 1s/epoch - 5ms/step
Epoch 31/100
211/211 - 1s - loss: 0.3229 - accuracy: 0.8799 - val_loss: 0.3171 - val_accuracy: 0.8
807 - 1s/epoch - 6ms/step
Epoch 32/100
211/211 - 1s - loss: 0.3343 - accuracy: 0.8780 - val_loss: 0.3295 - val_accuracy: 0.8
830 - 1s/epoch - 6ms/step
Epoch 33/100
211/211 - 1s - loss: 0.3191 - accuracy: 0.8817 - val_loss: 0.3445 - val_accuracy: 0.8
752 - 1s/epoch - 5ms/step
Epoch 34/100
211/211 - 1s - loss: 0.3117 - accuracy: 0.8847 - val_loss: 0.3343 - val_accuracy: 0.8
752 - 1s/epoch - 5ms/step
Epoch 35/100
211/211 - 1s - loss: 0.3175 - accuracy: 0.8814 - val_loss: 0.3205 - val_accuracy: 0.8
798 - 1s/epoch - 6ms/step
Epoch 36/100
211/211 - 1s - loss: 0.3104 - accuracy: 0.8851 - val_loss: 0.3370 - val_accuracy: 0.8
813 - 1s/epoch - 6ms/step
Epoch 37/100
211/211 - 1s - loss: 0.3323 - accuracy: 0.8776 - val_loss: 0.3345 - val_accuracy: 0.8
792 - 1s/epoch - 6ms/step
Epoch 38/100
211/211 - 1s - loss: 0.3234 - accuracy: 0.8802 - val_loss: 0.3301 - val_accuracy: 0.8
773 - 1s/epoch - 6ms/step
Epoch 39/100
211/211 - 1s - loss: 0.3125 - accuracy: 0.8839 - val_loss: 0.3310 - val_accuracy: 0.8
797 - 1s/epoch - 6ms/step
Epoch 40/100
211/211 - 1s - loss: 0.3081 - accuracy: 0.8853 - val_loss: 0.3139 - val_accuracy: 0.8
812 - 1s/epoch - 6ms/step
```

```
Epoch 41/100
211/211 - 1s - loss: 0.3021 - accuracy: 0.8880 - val_loss: 0.3340 - val_accuracy: 0.8
800 - 1s/epoch - 6ms/step
Epoch 42/100
211/211 - 1s - loss: 0.3127 - accuracy: 0.8846 - val_loss: 0.3026 - val_accuracy: 0.8
908 - 1s/epoch - 6ms/step
Epoch 43/100
211/211 - 1s - loss: 0.3061 - accuracy: 0.8853 - val_loss: 0.3241 - val_accuracy: 0.8
835 - 1s/epoch - 6ms/step
Epoch 44/100
211/211 - 1s - loss: 0.3093 - accuracy: 0.8860 - val_loss: 0.3079 - val_accuracy: 0.8
898 - 1s/epoch - 6ms/step
Epoch 45/100
211/211 - 1s - loss: 0.3021 - accuracy: 0.8880 - val_loss: 0.3250 - val_accuracy: 0.8
830 - 1s/epoch - 6ms/step
Epoch 46/100
211/211 - 1s - loss: 0.2988 - accuracy: 0.8892 - val_loss: 0.3465 - val_accuracy: 0.8
808 - 1s/epoch - 6ms/step
Epoch 47/100
211/211 - 1s - loss: 0.3018 - accuracy: 0.8886 - val_loss: 0.3235 - val_accuracy: 0.8
828 - 1s/epoch - 6ms/step
Epoch 48/100
211/211 - 1s - loss: 0.3088 - accuracy: 0.8859 - val_loss: 0.3097 - val_accuracy: 0.8
847 - 1s/epoch - 5ms/step
Epoch 49/100
211/211 - 1s - loss: 0.3073 - accuracy: 0.8858 - val_loss: 0.3399 - val_accuracy: 0.8
793 - 1s/epoch - 6ms/step
Epoch 50/100
211/211 - 1s - loss: 0.2993 - accuracy: 0.8888 - val_loss: 0.3374 - val_accuracy: 0.8
808 - 1s/epoch - 6ms/step
Epoch 51/100
211/211 - 1s - loss: 0.3005 - accuracy: 0.8890 - val_loss: 0.3465 - val_accuracy: 0.8
750 - 1s/epoch - 6ms/step
Epoch 52/100
211/211 - 1s - loss: 0.2986 - accuracy: 0.8884 - val_loss: 0.2921 - val_accuracy: 0.8
958 - 1s/epoch - 5ms/step
Epoch 53/100
211/211 - 1s - loss: 0.2935 - accuracy: 0.8909 - val_loss: 0.3028 - val_accuracy: 0.8
927 - 1s/epoch - 6ms/step
Epoch 54/100
211/211 - 1s - loss: 0.2921 - accuracy: 0.8916 - val_loss: 0.2986 - val_accuracy: 0.8
910 - 1s/epoch - 6ms/step
Epoch 55/100
211/211 - 1s - loss: 0.2997 - accuracy: 0.8893 - val_loss: 0.3037 - val_accuracy: 0.8
955 - 1s/epoch - 6ms/step
Epoch 56/100
211/211 - 1s - loss: 0.2933 - accuracy: 0.8910 - val_loss: 0.3034 - val_accuracy: 0.8
893 - 1s/epoch - 6ms/step
Epoch 57/100
211/211 - 1s - loss: 0.2911 - accuracy: 0.8897 - val_loss: 0.3217 - val_accuracy: 0.8
858 - 1s/epoch - 6ms/step
Epoch 58/100
211/211 - 1s - loss: 0.2938 - accuracy: 0.8916 - val_loss: 0.3145 - val_accuracy: 0.8
895 - 1s/epoch - 6ms/step
Epoch 59/100
211/211 - 1s - loss: 0.2870 - accuracy: 0.8929 - val_loss: 0.2905 - val_accuracy: 0.8
928 - 1s/epoch - 6ms/step
Epoch 60/100
211/211 - 1s - loss: 0.2913 - accuracy: 0.8916 - val_loss: 0.3059 - val_accuracy: 0.8
925 - 1s/epoch - 6ms/step
```

```
Epoch 61/100
211/211 - 1s - loss: 0.2948 - accuracy: 0.8911 - val_loss: 0.3187 - val_accuracy: 0.8
903 - 1s/epoch - 6ms/step
Epoch 62/100
211/211 - 1s - loss: 0.2892 - accuracy: 0.8929 - val_loss: 0.3183 - val_accuracy: 0.8
873 - 1s/epoch - 6ms/step
Epoch 63/100
211/211 - 1s - loss: 0.2895 - accuracy: 0.8939 - val_loss: 0.2971 - val_accuracy: 0.8
948 - 1s/epoch - 6ms/step
Epoch 64/100
211/211 - 1s - loss: 0.2866 - accuracy: 0.8950 - val_loss: 0.3219 - val_accuracy: 0.8
887 - 1s/epoch - 6ms/step
Epoch 65/100
211/211 - 1s - loss: 0.2850 - accuracy: 0.8936 - val_loss: 0.3066 - val_accuracy: 0.8
882 - 1s/epoch - 5ms/step
Epoch 66/100
211/211 - 1s - loss: 0.2775 - accuracy: 0.8962 - val_loss: 0.3003 - val_accuracy: 0.8
873 - 1s/epoch - 6ms/step
Epoch 67/100
211/211 - 1s - loss: 0.2804 - accuracy: 0.8961 - val_loss: 0.3039 - val_accuracy: 0.8
962 - 1s/epoch - 6ms/step
Epoch 68/100
211/211 - 1s - loss: 0.2863 - accuracy: 0.8932 - val_loss: 0.3016 - val_accuracy: 0.8
915 - 1s/epoch - 6ms/step
Epoch 69/100
211/211 - 1s - loss: 0.2785 - accuracy: 0.8973 - val_loss: 0.3075 - val_accuracy: 0.8
950 - 1s/epoch - 5ms/step
Epoch 70/100
211/211 - 1s - loss: 0.2785 - accuracy: 0.8965 - val_loss: 0.2887 - val_accuracy: 0.8
972 - 1s/epoch - 6ms/step
Epoch 71/100
211/211 - 1s - loss: 0.2742 - accuracy: 0.8979 - val_loss: 0.2927 - val_accuracy: 0.8
980 - 1s/epoch - 6ms/step
Epoch 72/100
211/211 - 1s - loss: 0.2783 - accuracy: 0.8958 - val_loss: 0.3058 - val_accuracy: 0.8
963 - 1s/epoch - 6ms/step
Epoch 73/100
211/211 - 1s - loss: 0.2853 - accuracy: 0.8936 - val_loss: 0.3209 - val_accuracy: 0.8
810 - 1s/epoch - 6ms/step
Epoch 74/100
211/211 - 1s - loss: 0.2824 - accuracy: 0.8946 - val_loss: 0.2933 - val_accuracy: 0.8
932 - 1s/epoch - 5ms/step
Epoch 75/100
211/211 - 1s - loss: 0.2707 - accuracy: 0.8986 - val_loss: 0.3091 - val_accuracy: 0.8
903 - 1s/epoch - 6ms/step
Epoch 76/100
211/211 - 1s - loss: 0.2759 - accuracy: 0.8966 - val_loss: 0.3311 - val_accuracy: 0.8
857 - 1s/epoch - 6ms/step
Epoch 77/100
211/211 - 1s - loss: 0.2806 - accuracy: 0.8954 - val_loss: 0.3119 - val_accuracy: 0.8
843 - 1s/epoch - 6ms/step
Epoch 78/100
211/211 - 1s - loss: 0.2720 - accuracy: 0.8972 - val_loss: 0.3266 - val_accuracy: 0.8
858 - 1s/epoch - 6ms/step
Epoch 79/100
211/211 - 1s - loss: 0.2777 - accuracy: 0.8962 - val_loss: 0.2884 - val_accuracy: 0.8
973 - 1s/epoch - 5ms/step
Epoch 80/100
211/211 - 1s - loss: 0.2670 - accuracy: 0.9004 - val_loss: 0.2900 - val_accuracy: 0.8
990 - 1s/epoch - 6ms/step
```

```
Epoch 81/100
211/211 - 1s - loss: 0.2648 - accuracy: 0.9005 - val_loss: 0.2948 - val_accuracy: 0.8
923 - 1s/epoch - 6ms/step
Epoch 82/100
211/211 - 1s - loss: 0.2666 - accuracy: 0.9000 - val_loss: 0.2793 - val_accuracy: 0.9
035 - 1s/epoch - 5ms/step
Epoch 83/100
211/211 - 1s - loss: 0.2719 - accuracy: 0.8984 - val_loss: 0.3021 - val_accuracy: 0.8
953 - 1s/epoch - 6ms/step
Epoch 84/100
211/211 - 1s - loss: 0.2730 - accuracy: 0.8999 - val_loss: 0.3375 - val_accuracy: 0.8
827 - 1s/epoch - 6ms/step
Epoch 85/100
211/211 - 1s - loss: 0.2801 - accuracy: 0.8969 - val_loss: 0.2964 - val_accuracy: 0.8
955 - 1s/epoch - 6ms/step
Epoch 86/100
211/211 - 1s - loss: 0.2732 - accuracy: 0.8988 - val_loss: 0.2699 - val_accuracy: 0.9
052 - 1s/epoch - 6ms/step
Epoch 87/100
211/211 - 1s - loss: 0.2701 - accuracy: 0.8999 - val_loss: 0.2997 - val_accuracy: 0.8
940 - 1s/epoch - 6ms/step
Epoch 88/100
211/211 - 1s - loss: 0.2718 - accuracy: 0.8995 - val_loss: 0.2838 - val_accuracy: 0.8
992 - 1s/epoch - 6ms/step
Epoch 89/100
211/211 - 1s - loss: 0.2648 - accuracy: 0.9008 - val_loss: 0.2846 - val_accuracy: 0.8
993 - 1s/epoch - 6ms/step
Epoch 90/100
211/211 - 1s - loss: 0.2680 - accuracy: 0.9013 - val_loss: 0.2955 - val_accuracy: 0.8
948 - 1s/epoch - 6ms/step
Epoch 91/100
211/211 - 1s - loss: 0.2647 - accuracy: 0.8997 - val_loss: 0.3058 - val_accuracy: 0.8
940 - 1s/epoch - 6ms/step
Epoch 92/100
211/211 - 1s - loss: 0.2644 - accuracy: 0.9019 - val_loss: 0.2870 - val_accuracy: 0.8
978 - 1s/epoch - 6ms/step
Epoch 93/100
211/211 - 1s - loss: 0.2664 - accuracy: 0.9005 - val_loss: 0.2793 - val_accuracy: 0.9
010 - 1s/epoch - 6ms/step
Epoch 94/100
211/211 - 1s - loss: 0.2672 - accuracy: 0.9020 - val_loss: 0.2929 - val_accuracy: 0.8
948 - 1s/epoch - 6ms/step
Epoch 95/100
211/211 - 1s - loss: 0.2646 - accuracy: 0.9013 - val_loss: 0.2899 - val_accuracy: 0.9
007 - 1s/epoch - 6ms/step
Epoch 96/100
211/211 - 1s - loss: 0.2635 - accuracy: 0.9018 - val_loss: 0.2757 - val_accuracy: 0.9
058 - 1s/epoch - 6ms/step
Epoch 97/100
211/211 - 1s - loss: 0.2597 - accuracy: 0.9029 - val_loss: 0.2855 - val_accuracy: 0.9
027 - 1s/epoch - 6ms/step
Epoch 98/100
211/211 - 1s - loss: 0.2546 - accuracy: 0.9046 - val_loss: 0.2759 - val_accuracy: 0.9
033 - 1s/epoch - 6ms/step
Epoch 99/100
211/211 - 1s - loss: 0.2480 - accuracy: 0.9061 - val_loss: 0.2906 - val_accuracy: 0.8
970 - 1s/epoch - 5ms/step
Epoch 100/100
211/211 - 1s - loss: 0.2555 - accuracy: 0.9038 - val_loss: 0.2662 - val_accuracy: 0.9
077 - 1s/epoch - 6ms/step
```

Out[21]:
```
<keras.callbacks.History at 0x17d611f2f50>
```

Resultados

Ya lo tenemos todo listo, con la siguiente linea vamos a poder evaluar la capacidad de nuestra red para predecir una prenda de ropa.

In [22]:
```
model.evaluate(X_test,y_test)
```

```
188/188 [==============================] - 0s 2ms/step - loss: 0.5084 - accuracy: 0.8
713
```

Out[22]:
```
[0.5084182620048523, 0.8713333606719971]
```

Al ejecutar el código debería dar el siguiente resultado o similar en la terminal. En el vemos el valor de accuracy calculado con los datos que habíamos apartado al inicio y con los que NO hemos entrenado. ¡Enhorabuena nuestra red acierta un **87.13%** de las veces la prenda de ropa correcta!