Proyecto Final

Descripción de la Evidencia: El proyecto de aprendizaje profundo consiste en la solución del problema de clasificación de prendas de vestir utilizando el conjunto de datos Fashion MNIST. Éste código implementa un modelo de red neuronal (*visto en clase*) que resuelva el problema de clasificación.

A01285158 | Grace Aviance Silva Aróstegui

A00000000 | Christian Jaffé Alarcón Acosta

A00000000 | David Vázquez Moreno

```
pip install visualkeras #Instalamos visualkeras para visualizar la arquitectura de la red neuronal
```

```
Requirement already satisfied: visualkeras in /usr/local/lib/python3.10/dist-packages (0.0.2)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from visualkeras) (9.4.0)
Requirement already satisfied: numpy>=1.18.1 in /usr/local/lib/python3.10/dist-packages (from visualkeras) (1.23.5)
Requirement already satisfied: aggdraw>=1.3.11 in /usr/local/lib/python3.10/dist-packages (from visualkeras) (1.3.16)
```

```
import numpy as np
import pandas as pd
import seaborn as sns
import tensorflow as tf
import matplotlib.pyplot as plt
import math

from tensorflow import keras
from keras.utils import to_categorical
from keras.optimizers import RMSprop,Adam
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report

# Para visualizar la arquitectura de la red neuronal
# pip install visualkeras
import visualkeras
```

```
# Model / data parameters
num_classes = 10
input_shape = (28, 28, 1) #Las imágenes de MNIST tienen una forma 28x28

#Cargamos los datos dividiendo en train y test set
(x_train, y_train), (x_test, y_test) = keras.datasets.fashion_mnist.load_data()
```

```
#Visualizamos las imágenes a clasificar
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle 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(x_train[i], cmap=plt.cm.binary)
    plt.xlabel(class_names[y_train[i]])
plt.show()
```

Se trata de un conjunto de datos de 60,000 imágenes en escala de grises de 28x28 de 10 categorías de moda, con valores de pixel que varian de 0 a 255. Junto con un conjunto de prueba de 10,000 imágenes. Los *labels* son un arreglo de integros, que van del 0 al 9. Estos corresponden a la *class* de ropa que la imagen representa.

Label	Class
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot

```
#Forma de los datos de entrenamiento
x_train.shape
y_train.shape

#Forma de los datos de entrenamiento
x_test.shape
y_test.shape
```

```
(10000,)
```

Preparamos los datos para crear el modelo

```
# Scale images to the [0, 1] range
x_train = x_train.astype("float32") / 255
x_test = x_test.astype("float32") / 255

# Make sure images have shape (28, 28, 1)
x_train = np.expand_dims(x_train, -1)
x_test = np.expand_dims(x_test, -1)
print("x_train shape:", x_train.shape)
print(x_train.shape[0], "train samples")
print(x_test.shape[0], "test samples")
```

```
# convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
```

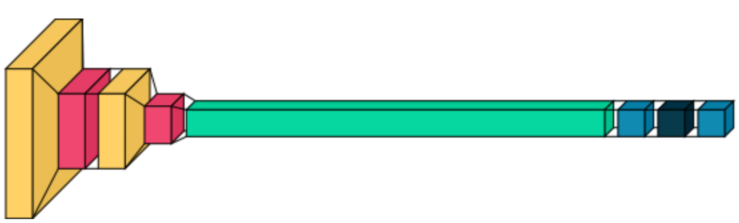
```
x_train shape: (60000, 28, 28, 1)
60000 train samples
10000 test samples
```

Construimos el modelo

```
# Usamos un modelo secuencial, que es una pila lineal de capas
model = tf.keras.models.Sequential([
    # Primera capa. Tiene una capa Convolucional 2D con un kernel de tamaño 3x3 y operación Max pooling
    tf.keras.layers.Conv2D(32, (3,3), padding='same', input_shape=(28,28,1)),
    tf.keras.layers.MaxPooling2D(pool_size=(2,2)),

    # Segunda capa. Tiene una capa Convolucional 2D con un kernel de tamaño 3x3, función de activación ReLu y operación Max pooling
    tf.keras.layers.Conv2D(64, (3,3), padding='same' ,activation='relu'),
    tf.keras.layers.MaxPooling2D(pool_size=(2,2)),

    # Capa completamente conectada con función de activación ReLu
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'),

    # Capa dropout para evitar overfitting
    tf.keras.layers.Dropout(0.5),

    # Capa Output con función de activación Softmax
    tf.keras.layers.Dense(10, activation='softmax')

])
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 28, 28, 32)        320

 max_pooling2d (MaxPooling2  (None, 14, 14, 32)        0
 D)

 conv2d_1 (Conv2D)           (None, 14, 14, 64)        18496

 max_pooling2d_1 (MaxPoolin  (None, 7, 7, 64)          0
 g2D)

 flatten (Flatten)           (None, 3136)              0

 dense (Dense)               (None, 128)               401536

 dropout (Dropout)           (None, 128)               0

 dense_1 (Dense)             (None, 10)                1290

=================================================================
Total params: 421642 (1.61 MB)
Trainable params: 421642 (1.61 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
# Visualización 3D de la arquitectura de nuestro CNN
visualkeras.layered_view(model)
```



```
# Compilamos el modelo
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics='accuracy')


# Entrenamos el modelo
model.fit(x_train, y_train, epochs=50, batch_size=128,verbose=1)
```

```
Epoch 1/50
469/469 [==============================] - 32s 66ms/step - loss: 0.5744 - accuracy: 0.7957
Epoch 2/50
469/469 [==============================] - 27s 58ms/step - loss: 0.3828 - accuracy: 0.8655
Epoch 3/50
469/469 [==============================] - 30s 64ms/step - loss: 0.3347 - accuracy: 0.8796
Epoch 4/50
469/469 [==============================] - 31s 65ms/step - loss: 0.3043 - accuracy: 0.8887
Epoch 5/50
469/469 [==============================] - 38s 80ms/step - loss: 0.2846 - accuracy: 0.8966
Epoch 6/50
469/469 [==============================] - 36s 76ms/step - loss: 0.2678 - accuracy: 0.9028
Epoch 7/50
469/469 [==============================] - 30s 65ms/step - loss: 0.2495 - accuracy: 0.9087
Epoch 8/50
469/469 [==============================] - 35s 74ms/step - loss: 0.2397 - accuracy: 0.9119
Epoch 9/50
469/469 [==============================] - 26s 55ms/step - loss: 0.2266 - accuracy: 0.9169
Epoch 10/50
469/469 [==============================] - 29s 62ms/step - loss: 0.2155 - accuracy: 0.9209
Epoch 11/50
469/469 [==============================] - 31s 66ms/step - loss: 0.2049 - accuracy: 0.9248
Epoch 12/50
469/469 [==============================] - 32s 68ms/step - loss: 0.1945 - accuracy: 0.9262
Epoch 13/50
469/469 [==============================] - 28s 59ms/step - loss: 0.1875 - accuracy: 0.9298
Epoch 14/50
469/469 [==============================] - 29s 61ms/step - loss: 0.1797 - accuracy: 0.9319
Epoch 15/50
469/469 [==============================] - 31s 66ms/step - loss: 0.1710 - accuracy: 0.9359
Epoch 16/50
469/469 [==============================] - 34s 73ms/step - loss: 0.1651 - accuracy: 0.9372
Epoch 17/50
469/469 [==============================] - 28s 61ms/step - loss: 0.1603 - accuracy: 0.9396
Epoch 18/50
469/469 [==============================] - 30s 65ms/step - loss: 0.1521 - accuracy: 0.9412
Epoch 19/50
469/469 [==============================] - 32s 69ms/step - loss: 0.1467 - accuracy: 0.9434
Epoch 20/50
469/469 [==============================] - 32s 68ms/step - loss: 0.1389 - accuracy: 0.9454
Epoch 21/50
469/469 [==============================] - 31s 67ms/step - loss: 0.1370 - accuracy: 0.9471
Epoch 22/50
469/469 [==============================] - 30s 63ms/step - loss: 0.1321 - accuracy: 0.9486
Epoch 23/50
469/469 [==============================] - 32s 67ms/step - loss: 0.1264 - accuracy: 0.9508
Epoch 24/50
469/469 [==============================] - 30s 63ms/step - loss: 0.1210 - accuracy: 0.9525
Epoch 25/50
469/469 [==============================] - 32s 68ms/step - loss: 0.1171 - accuracy: 0.9536
Epoch 26/50
469/469 [==============================] - 27s 57ms/step - loss: 0.1139 - accuracy: 0.9554
Epoch 27/50
469/469 [==============================] - 27s 57ms/step - loss: 0.1092 - accuracy: 0.9570
Epoch 28/50
469/469 [==============================] - 28s 59ms/step - loss: 0.1070 - accuracy: 0.9572
Epoch 29/50
469/469 [==============================] - 28s 59ms/step - loss: 0.1025 - accuracy: 0.9596
```

```
# Evaluamos el modelo con el conjunto de datos de prueba
score = model.evaluate(x_test,  y_test, steps=math.ceil(10000/32))

# Pérdida en la prueba y Exactitud en prueba
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
313/313 [==============================] - 1s 4ms/step - loss: 0.3883 - accuracy: 0.9225
Test loss: 0.3882904648780823
Test accuracy: 0.9225000143051147
```

RESULTADOS:

Obtuvimos una exactitud del 92.25%

Predicción

```
# Imprimimos 16 imágenes aleatorias con su clase de ropa tanto real como con predicción

labels = {0 : "T-shirt/top", 1: "Trouser", 2: "Pullover", 3: "Dress", 4: "Coat",
          5: "Sandal", 6: "Shirt", 7: "Sneaker", 8: "Bag", 9: "Ankle Boot"}

y_pred = model.predict(x_test)
x_test__ = x_test.reshape(x_test.shape[0], 28, 28)

fig, axis = plt.subplots(4, 4, figsize=(12, 14))
for i, ax in enumerate(axis.flat):
    ax.imshow(x_test__[i], cmap='binary')
    ax.set(title = f"Real Class is {labels[y_test[i].argmax()]}\nPredict Class is {labels[y_pred[i].argmax()]}");
```

```
313/313 [==============================] - 1s 4ms/step
```