Topic : Neural Network for Image Classification

Objective for this template:

1. To introduce participants to the basic pipeline for Image classification.
2. Use tensorflow to build a simple sequential neural network image classifier
3. Demonstrate the process of training the model and evaluating its performance
4. Allow participants to experiment on testing the prediction model using single images.

Designed By: *Rodolfo C. Raga Jr.*

Permission granted to use template for educational purposes so long as this heading is not removed.

Step 1: load the tensorflow library and other helper libaries using the import keyword

```
import tensorflow as tf
from tensorflow import keras
import numpy as np
import matplotlib.pyplot as plt

print("Done with library declaration. Current version of Tensorflow is
:", tf.__version__)
```

```
Done with library declaration. Current version of Tensorflow is :
2.7.0
```

Step 2: Extract data from the MNIST dataset. Fashion-MNIST is an online dataset of various clothes with 10 categories (Tshirt/top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag, Ankle boot). The images have size (28x28) with (60,000) training and (10,000) testing instances.

```
fashion_mnist = keras.datasets.fashion_mnist
```

Step 3 : Separate the data into distinct training and testing datasets and Define the class categories

```
(train_images, train_labels), (test_images, test_labels) =
fashion_mnist.load_data()
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-
keras-datasets/train-labels-idx1-ubyte.gz
32768/29515 [==============================] - 0s 0us/step
40960/29515 [==========================================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-
keras-datasets/train-images-idx3-ubyte.gz
```

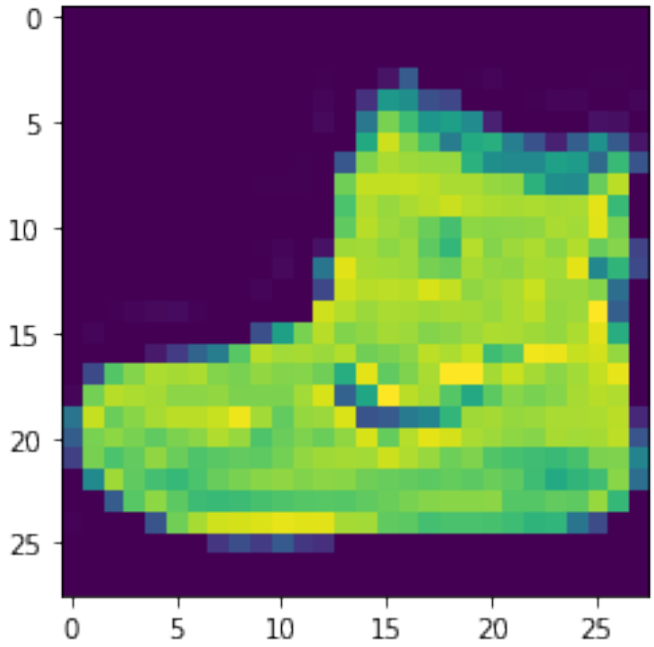
```
26427392/26421880 [==============================] - 0s 0us/step
26435584/26421880 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-
keras-datasets/t10k-labels-idx1-ubyte.gz
16384/5148
[==============================================================================
==========================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-
keras-datasets/t10k-images-idx3-ubyte.gz
4423680/4422102 [==============================] - 0s 0us/step
4431872/4422102 [==============================] - 0s 0us/step
```

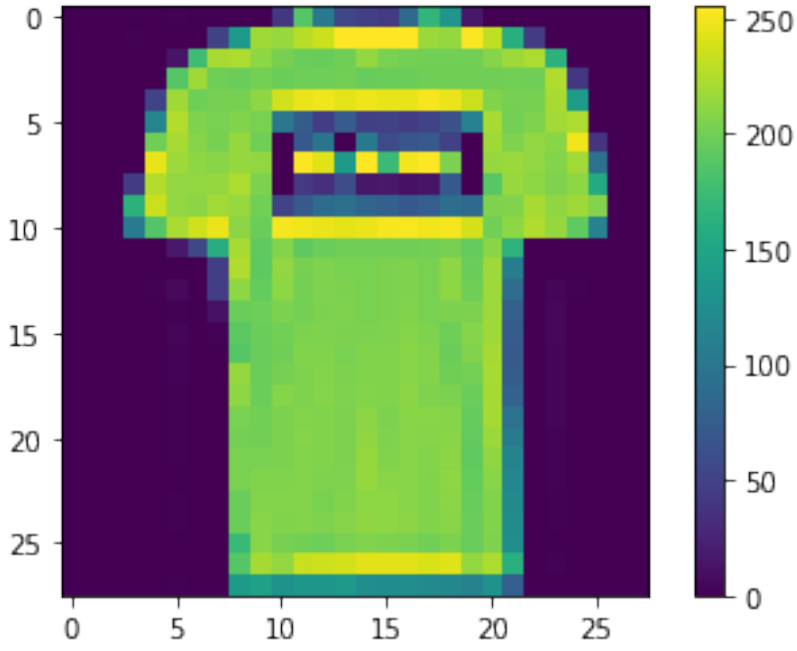
Step 4: Display and inspect sample images from the dataset

```
plt.figure()
plt.imshow(train_images[0])
plt.show()

plt.imshow(train_images[1])
plt.colorbar()
plt.grid(False)
plt.show()

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()
```


0
5
10
15
20
25
0
5
10
15
20
25


0
5
10
15
20
25
0
5
10
15
20
25
250
200
150
100
50
0

Step 5 : Preprocess the image by normalizing the pixel values. Display the first 25 images from the training set and display the class name below each image. Verify that the data is in the correct format.

```
train_images = train_images / 255.0
test_images = test_images / 255.0

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], cmap=plt.cm.binary)
    plt.xlabel("image#"+str(i)+" "+class_names[train_labels[i]])
plt.show()
```

Step 6: Start building the neural network by using the Sequential API.

Then, define the layers using the Dense class. Note that the activation function for the hidden layer is relu while for the output layer it is softmax.

Add the layers to the NN model.

```
model = tf.keras.Sequential()

layer_0 = tf.keras.layers.Flatten(input_shape=(28,28))
layer_1 = tf.keras.layers.Dense(units=128, activation="relu")
#layer_2 = tf.keras.layers.Dense(units=128, activation="relu")
#layer_3 = tf.keras.layers.Dense(units=128, activation="relu")
layer_4 = tf.keras.layers.Dense(units=10, activation="softmax")

model.add(layer_0)
model.add(layer_1)
```

```
#model.add(layer_2)
#model.add(layer_3)
model.add(layer_4)
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 flatten_1 (Flatten)          (None, 784)               0

 dense_2 (Dense)              (None, 128)               100480

 dense_3 (Dense)              (None, 10)                1290

=================================================================
Total params: 101,770
Trainable params: 101,770
Non-trainable params: 0
_________________________________________________________________
```

Step 7: Compile the built model architecture. The compilation configures the learning process by defining an optimizer, a loss function, and other useful training parameters.

```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

Step 8: Train the model by using the set training data

```
trained_model = model.fit(train_images, train_labels, epochs=5)
```

```
Epoch 1/5
1875/1875 [==============================] - 4s 2ms/step - loss:
0.4960 - accuracy: 0.8262
Epoch 2/5
1875/1875 [==============================] - 4s 2ms/step - loss:
0.3732 - accuracy: 0.8660
Epoch 3/5
1875/1875 [==============================] - 4s 2ms/step - loss:
0.3339 - accuracy: 0.8785
Epoch 4/5
1875/1875 [==============================] - 4s 2ms/step - loss:
0.3087 - accuracy: 0.8874
Epoch 5/5
1875/1875 [==============================] - 4s 2ms/step - loss:
0.2925 - accuracy: 0.8924
```

Step 9: Test the performance of the model using the testing dataset

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

```
print('Test accuracy: {} Test Loss: {}'.format(test_acc*100,
test_loss))
```

```
313/313 [==============================] - 1s 2ms/step - loss: 0.3468
- accuracy: 0.8756
Test accuracy: 87.55999803543091 Test Loss: 0.34677854180336
```

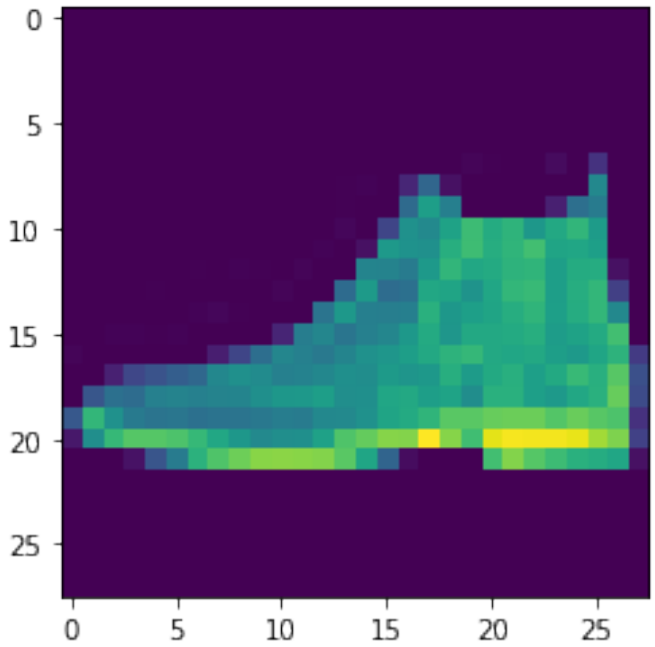
Step 10: Test the performance of the model using individual images from the dataset

```
imageIndex=0

img = test_images[imageIndex]
print(img.shape)
img = (np.expand_dims(img,0))
predictions_single = model.predict(img)
print("That is an "+class_names[np.argmax(predictions_single)])


plt.figure()
plt.imshow( test_images[imageIndex])
plt.show()
```

```
(28, 28)
That is an Ankle boot
```



Other things we can do:

Analyze training statistics

```
plt.xlabel('Epoch Number')
plt.ylabel("Loss Magnitude")
plt.plot(trained_model.history['loss'])
```

```
[<matplotlib.lines.Line2D at 0x7fb828138d10>]
```