```
# Import the necessary packages
```

```
# Importing necessary Libraries
import tensorflow as tf
from tensorflow import keras
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import random
```

```
# Load the training and testing data MNIST
```

```
# Import dataset & split into train and test data
fmnist=tf.keras.datasets.fashion_mnist
(x_train,y_train),(x_test,y_test)=fmnist.load_data()
```

```
# Length of the training dataset
len(x_train)
len(y_train)
```

```
60000
```

```
# Length of the testing dataset
len(x_test)
len(y_test)
```

```
10000
```

```
# Shape of the training dataset
x_train.shape
```

```
(60000, 28, 28)
```

```
# Shape of the testing dataset
x_test.shape
```

```
(10000, 28, 28)
```

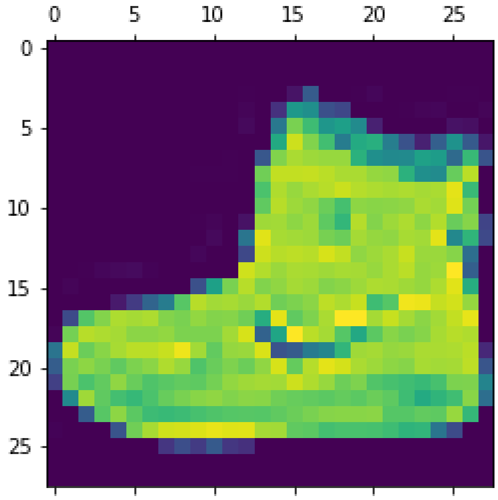
```
# See first Image Matrix
x_train[0]
```

```
             200, 232, 232, 233, 229, 223, 223, 215, 213, 164, 127, 123, 196,
             229,   0],
```

```
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
 183, 225, 216, 223, 228, 235, 227, 224, 222, 224, 221, 223, 245,
 173,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
 193, 228, 218, 213, 198, 180, 212, 210, 211, 213, 223, 220, 243,
 202,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   1,   3,   0,  12,
 219, 220, 212, 218, 192, 169, 227, 208, 218, 224, 212, 226, 197,
 209,  52],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   6,   0,  99,
 244, 222, 220, 218, 203, 198, 221, 215, 213, 222, 220, 245, 119,
 167,  56],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   4,   0,   0,  55,
 236, 228, 230, 228, 240, 232, 213, 218, 223, 234, 217, 217, 209,
  92,   0],
[  0,   0,   1,   4,   6,   7,   2,   0,   0,   0,   0,   0, 237,
 226, 217, 223, 222, 219, 222, 221, 216, 223, 229, 215, 218, 255,
  77,   0],
[  0,   3,   0,   0,   0,   0,   0,   0,  62, 145, 204, 228,
 207, 213, 221, 218, 208, 211, 218, 224, 223, 219, 215, 224, 244,
 159,   0],
[  0,   0,   0,   0,  18,  44,  82, 107, 189, 228, 220, 222, 217,
 226, 200, 205, 211, 230, 224, 234, 176, 188, 250, 248, 233, 238,
 215,   0],
[  0,  57, 187, 208, 224, 221, 224, 208, 204, 214, 208, 209, 200,
 159, 245, 193, 206, 223, 255, 255, 221, 234, 221, 211, 220, 232,
 246,   0],
[  3, 202, 228, 224, 221, 211, 211, 214, 205, 205, 205, 220, 240,
  80, 150, 255, 229, 221, 188, 154, 191, 210, 204, 209, 222, 228,
 225,   0],
[ 98, 233, 198, 210, 222, 229, 229, 234, 249, 220, 194, 215, 217,
 241,  65,  73, 106, 117, 168, 219, 221, 215, 217, 223, 223, 224,
 229,  29],
[ 75, 204, 212, 204, 193, 205, 211, 225, 216, 185, 197, 206, 198,
 213, 240, 195, 227, 245, 239, 223, 218, 212, 209, 222, 220, 221,
 230,  67],
[ 48, 203, 183, 194, 213, 197, 185, 190, 194, 192, 202, 214, 219,
 221, 220, 236, 225, 216, 199, 206, 186, 181, 177, 172, 181, 205,
 206, 115],
[  0, 122, 219, 193, 179, 171, 183, 196, 204, 210, 213, 207, 211,
 210, 200, 196, 194, 191, 195, 191, 198, 192, 176, 156, 167, 177,
 210,  92],
[  0,   0,  74, 189, 212, 191, 175, 172, 175, 181, 185, 188, 189,
 188, 193, 198, 204, 209, 210, 210, 211, 188, 188, 194, 192, 216,
 170,   0],
[  2,   0,   0,   0,  66, 200, 222, 237, 239, 242, 246, 243, 244,
 221, 220, 193, 191, 179, 182, 182, 181, 176, 166, 168,  99,  58,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,  40,  61,  44,  72,  41,  35,
   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0]], dtype=uint8)
```

```
# See first image
plt.matshow(x_train[0])
```

```
<matplotlib.image.AxesImage at 0x7f9288264f90>
```




```
# Normalize the iamges by scaling pixel intensities to the range 0,1
x_train=x_train/255
x_test=x_test/255
```

```
# See first Naormalize Image Matrix
x_train[0]
```

```
         0.82745098, 0.82745098, 0.83921569, 0.80392157, 0.80392157,
         0.80392157, 0.8627451 , 0.94117647, 0.31372549, 0.58823529,
         1.        , 0.89803922, 0.86666667, 0.7372549 , 0.60392157,
         0.74901961, 0.82352941, 0.8       , 0.81960784, 0.87058824,
         0.89411765, 0.88235294, 0.        ],
        [0.38431373, 0.91372549, 0.77647059, 0.82352941, 0.87058824,
         0.89803922, 0.89803922, 0.91764706, 0.97647059, 0.8627451 ,
         0.76078431, 0.84313725, 0.85098039, 0.94509804, 0.25490196,
         0.28627451, 0.41568627, 0.45882353, 0.65882353, 0.85882353,
         0.86666667, 0.84313725, 0.85098039, 0.8745098 , 0.8745098 ,
         0.87843137, 0.89803922, 0.11372549],
        [0.29411765, 0.8       , 0.83137255, 0.8       , 0.75686275,
         0.80392157, 0.82745098, 0.88235294, 0.84705882, 0.7254902 ,
         0.77254902, 0.80784314, 0.77647059, 0.83529412, 0.94117647,
         0.76470588, 0.89019608, 0.96078431, 0.9372549 , 0.8745098 ,
         0.85490196, 0.83137255, 0.81960784, 0.87058824, 0.8627451 ,
         0.86666667, 0.90196078, 0.2627451 ],
        [0.18823529, 0.79607843, 0.71764706, 0.76078431, 0.83529412,
         0.77254902, 0.7254902 , 0.74509804, 0.76078431, 0.75294118,
         0.79215686, 0.83921569, 0.85882353, 0.86666667, 0.8627451 ,
         0.9254902 , 0.88235294, 0.84705882, 0.78039216, 0.80784314,
         0.72941176, 0.70980392, 0.69411765, 0.6745098 , 0.70980392,
         0.80392157, 0.80784314, 0.45098039],
        [0.        , 0.47843137, 0.85882353, 0.75686275, 0.70196078,
         0.67058824, 0.71764706, 0.76862745, 0.8       , 0.82352941,
         0.83529412, 0.81176471, 0.82745098, 0.82352941, 0.78431373,
```

```
         0.76862745, 0.76078431, 0.74901961, 0.76470588, 0.74901961,
         0.77647059, 0.75294118, 0.69019608, 0.61176471, 0.65490196,
         0.69411765, 0.82352941, 0.36078431],
        [0.        , 0.        , 0.29019608, 0.74117647, 0.83137255,
         0.74901961, 0.68627451, 0.6745098 , 0.68627451, 0.70980392,
         0.7254902 , 0.7372549 , 0.74117647, 0.7372549 , 0.75686275,
         0.77647059, 0.8       , 0.81960784, 0.82352941, 0.82352941,
         0.82745098, 0.7372549 , 0.7372549 , 0.76078431, 0.75294118,
         0.84705882, 0.66666667, 0.        ],
        [0.00784314, 0.        , 0.        , 0.        , 0.25882353,
         0.78431373, 0.87058824, 0.92941176, 0.9372549 , 0.94901961,
         0.96470588, 0.95294118, 0.95686275, 0.86666667, 0.8627451 ,
         0.75686275, 0.74901961, 0.70196078, 0.71372549, 0.71372549,
         0.70980392, 0.69019608, 0.65098039, 0.65882353, 0.38823529,
         0.22745098, 0.        , 0.        ],
        [0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.15686275, 0.23921569, 0.17254902,
         0.28235294, 0.16078431, 0.1372549 , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        ],
        [0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        ],
        [0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
```

```
# Define the network architecture using Keras


model=keras.Sequential([
  # Input Layer
  keras.layers.Flatten(input_shape = (28,28)),
  # Hidden Layer
  keras.layers.Dense(128,activation ='relu'),
  # Output Layer
  keras.layers.Dense(20,activation = 'softmax')
])


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, 20)                2580

=================================================================
Total params: 103,060
Trainable params: 103,060
Non-trainable params: 0
_________________________________________________________________
```

```
# Compile the Model
model.compile(loss='sparse_categorical_crossentropy', optimizer='sgd', metrics=['accuracy'])
```

```
#Train the model using SGD
```

```
history=model.fit(x_train,y_train,validation_data=(x_test,y_test),epochs=10)
```

```
Epoch 1/10
1875/1875 [==============================] - 9s 4ms/step - loss: 0.7605 - accuracy: 0.76
Epoch 2/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.5169 - accuracy: 0.82
Epoch 3/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.4731 - accuracy: 0.83
Epoch 4/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.4474 - accuracy: 0.84
Epoch 5/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.4289 - accuracy: 0.85
Epoch 6/10
1875/1875 [==============================] - 7s 4ms/step - loss: 0.4142 - accuracy: 0.85
Epoch 7/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.4036 - accuracy: 0.86
Epoch 8/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.3938 - accuracy: 0.86
Epoch 9/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.3836 - accuracy: 0.86
Epoch 10/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.3765 - accuracy: 0.86
```

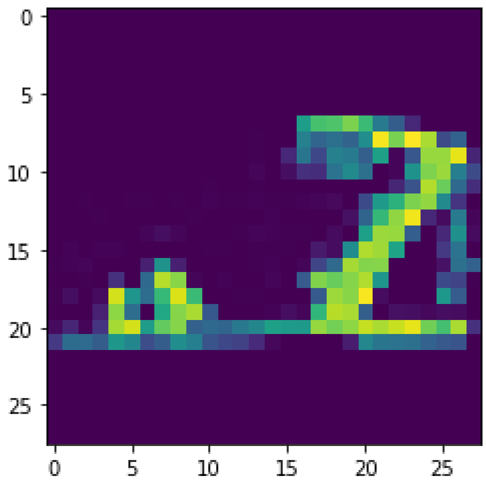
```
#Evaluate the network
```

```
test_loss,test_acc=model.evaluate(x_test,y_test)
print("Loss=%.3f" %test_loss)
print("Accuracy=%.3f" %test_acc)
```

```
313/313 [==============================] - 1s 2ms/step - loss: 0.4110 - accuracy: 0.8554
Loss=0.411
Accuracy=0.855
```

```
# Making Prediction on New Data

n=random.randint(0,9999)
plt.imshow(x_test[n])
plt.show()
```



```
predicted_value=model.predict(x_test)
print("Image is = %d" %np.argmax(predicted_value[0]))
```

```
313/313 [==============================] - 1s 2ms/step
Image is = 9
```

```
class_labels=["T - shirt / top","Trouser","Pullover","Dress","Coat","Sandal","Shirt","Sneaker
class_labels[np.argmax(predicted_value[0])]
```

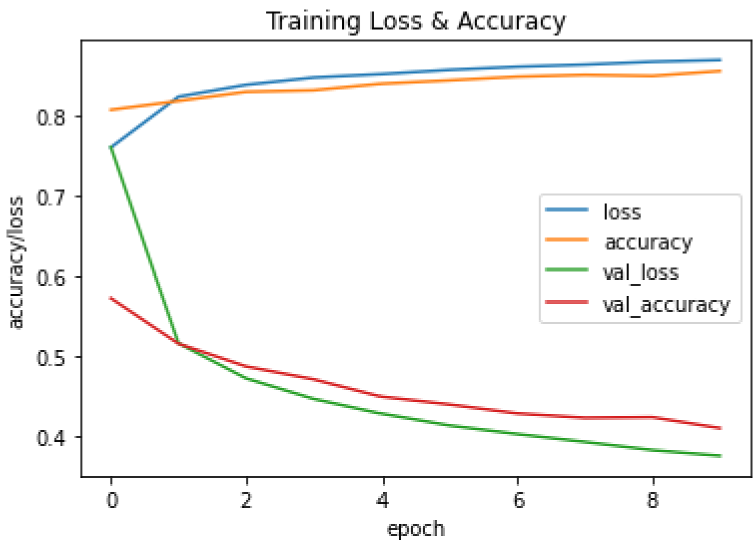
```
'Ankle boot'
```

```
# Plot the training loss and accuracy
```

```
history.history.keys()
```

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Training Loss & Accuracy')
plt.ylabel('accuracy/loss')
plt.xlabel('epoch')
plt.legend(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
plt.show()
```



✓ 0s completed at 12:15 PM