Srushti Ramesh Bhamre BC56

Use MNIST Fashion Dataset and create a classifier to classify fashion clothing into categories.

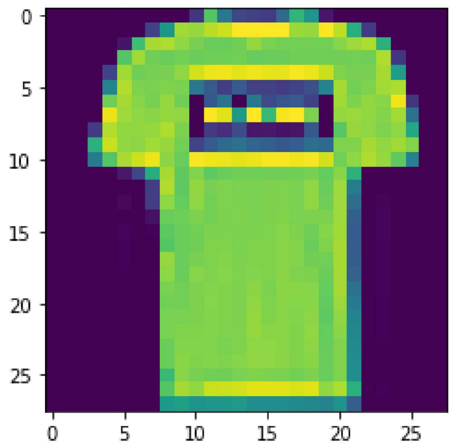
In [1]:

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

(x_train, y_train), (x_test, y_test) = keras.datasets.fashion_mnist.load_data()

# There are 10 image classes in this dataset and each class has a mapping correspond

#0 T-shirt/top
#1 Trouser
#2 pullover
#3 Dress
#4 Coat
#5 sandals
#6 shirt
#7 sneaker
#8 bag
#9 ankle boot
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-idx1-ubyte.gz (https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-idx1-ubyte.gz)
29515/29515 [==============================] - 0s 4us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-idx3-ubyte.gz (https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-idx3-ubyte.gz)
26421880/26421880 [==============================] - 20s 1us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-labels-idx1-ubyte.gz (https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-labels-idx1-ubyte.gz)
5148/5148 [==============================] - 0s 0s/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-images-idx3-ubyte.gz (https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-images-idx3-ubyte.gz)
4422102/4422102 [==============================] - 2s 0us/step
```

In [2]:
```
plt.imshow(x_train[1])
```

Out[2]: <matplotlib.image.AxesImage at 0x16ebe6f2fa0>



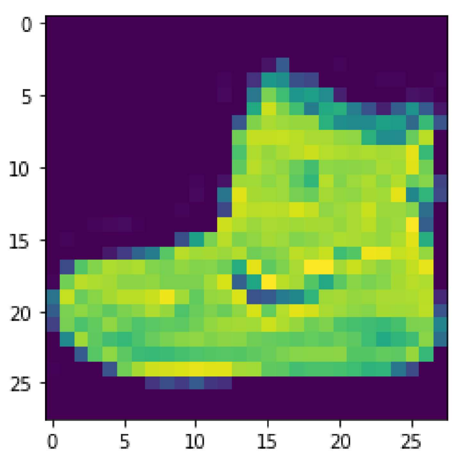
plt.imshow(): This is a function from the matplotlib library, specifically used to display images. It takes an array-like object as input and renders it as an image.

x_train[1]: This accesses the second element (index 1) of the x_train array, which contains the input images from the Fashion MNIST dataset. Each element of x_train is a 2D array representing a grayscale image.

So, plt.imshow(x_train[1]) displays the second image from the training set (x_train[1]) as a grayscale image using matplotlib. Since the Fashion MNIST dataset contains grayscale images, imshow() will render the image in grayscale by default.

In [3]:
```
plt.imshow(x_train[0])
```

Out[3]: <matplotlib.image.AxesImage at 0x16ebe720f70>



plt.imshow(): This function, as before, is used to display images using matplotlib.

x_train[0]: This retrieves the first element (index 0) of the x_train array, which represents the input image data from the Fashion MNIST dataset. Each element of x_train is a 2D array representing a grayscale image.

So, plt.imshow(x_train[0]) displays the first image from the training set (x_train[0]) as a grayscale image using matplotlib. Since the Fashion MNIST dataset contains grayscale images, imshow() will render the image in grayscale by default.

Type *Markdown* and LaTeX: α^2

In [4]:

```
# Next, we will preprocess the data by scaling the pixel values to be between 0 and 

x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0

x_train = x_train.reshape(-1, 28, 28, 1)
x_test = x_test.reshape(-1, 28, 28, 1)

# 28, 28 comes from width, height, 1 comes from the number of channels
# -1 means that the length in that dimension is inferred.
# This is done based on the constraint that the number of elements in an ndarray or 

# each image is a row vector (784 elements) and there are lots of such rows (let it 

```

In [5]:

```
# converting the training_images array to 4 dimensional array with sizes 60000, 28, 

x_train.shape
```

Out[5]: (60000, 28, 28, 1)

In [6]:

```
x_test.shape
```

Out[6]: (10000, 28, 28, 1)

In [7]:

```
y_train.shape
```

Out[7]: (60000,)

In [8]:

```
y_test.shape
```

Out[8]: (10000,)

In [9]:

```python
# We will use a convolutional neural network (CNN) to classify the fashion items.
# The CNN will consist of multiple convolutional layers followed by max pooling,
# dropout, and dense layers. Here is the code for the model:

model = keras.Sequential([
    keras.layers.Conv2D(32, (3,3), activation='relu', input_shape=(28,28,1)),
    # 32 filters (default), randomly initialized
    # 3*3 is Size of Filter
    # 28,28,1 size of Input Image
    # No zero-padding: every output 2 pixels less in every dimension
    # in Paramter shwon 320 is value of weights: (3x3 filter weights + 32 bias) * 32
    # 32*3*3=288(Total)+32(bias)= 320


    keras.layers.MaxPooling2D((2,2)),
    # It shown 13 * 13 size image with 32 channel or filter or depth.

    keras.layers.Dropout(0.25),
    # Reduce Overfitting of Training sample drop out 25% Neuron

    keras.layers.Conv2D(64, (3,3), activation='relu'),
    # Deeper layers use 64 filters
    # 3*3 is Size of Filter
    # Observe how the input image on 28x28x1 is transformed to a 3x3x64 feature map
    # 13(Size)-3(Filter Size )+1(bias)=11 Size for Width and Height with 64 Depth or
    # in Paramter shwon 18496 is value of weights: (3x3 filter weights + 64 bias) * 
    # 64*3*3=576+1=577*32 + 32(bias)=18496

    keras.layers.MaxPooling2D((2,2)),
    # It shown 5 * 5 size image with 64 channel or filter or depth.

    keras.layers.Dropout(0.25),

    keras.layers.Conv2D(128, (3,3), activation='relu'),
    # Deeper layers use 128 filters
    # 3*3 is Size of Filter
    # Observe how the input image on 28x28x1 is transformed to a 3x3x128 feature map
    # It show 5(Size)-3(Filter Size )+1(bias)=3 Size for Width and Height with 64 De
    # 128*3*3=1152+1=1153*64 + 64(bias)= 73856

    # To classify the images, we still need a Dense and Softmax layer.
    # We need to flatten the 3x3x128 feature map to a vector of size 1152
    # https://medium.com/@iamvarman/how-to-calculate-the-number-of-parameters-in-the

    keras.layers.Flatten(),
    keras.layers.Dense(128, activation='relu'),
    # 128 Size of Node in Dense Layer
    # 1152*128 = 147584

    keras.layers.Dropout(0.25),
    keras.layers.Dense(10, activation='softmax')
    # 10 Size of Node another Dense Layer
    # 128*10+10 bias= 1290
])
```

In [10]:

```python
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 32)        320       
                                                                 
 max_pooling2d (MaxPooling2D  (None, 13, 13, 32)       0         
 )                                                               
                                                                 
 dropout (Dropout)           (None, 13, 13, 32)        0         
                                                                 
 conv2d_1 (Conv2D)           (None, 11, 11, 64)        18496     
                                                                 
 max_pooling2d_1 (MaxPooling  (None, 5, 5, 64)         0         
 2D)                                                             
                                                                 
 dropout_1 (Dropout)         (None, 5, 5, 64)          0         
                                                                 
 conv2d_2 (Conv2D)           (None, 3, 3, 128)         73856     
                                                                 
 flatten (Flatten)           (None, 1152)              0         
                                                                 
 dense (Dense)               (None, 128)               147584    
                                                                 
 dropout_2 (Dropout)         (None, 128)               0         
                                                                 
 dense_1 (Dense)             (None, 10)                1290      
                                                                 
=================================================================
Total params: 241,546
Trainable params: 241,546
Non-trainable params: 0
_________________________________________________________________
```

In [11]:

```python
# Compile and Train the Model
# After defining the model, we will compile it and train it on the training data.

model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['ac

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

# 1875 is a number of batches. By default batches contain 32 samles.60000 / 32 = 187
```

```
Epoch 1/10
1875/1875 [==============================] - 54s 27ms/step - loss: 0.5724 - accuracy:
0.7858 - val_loss: 0.3757 - val_accuracy: 0.8620
Epoch 2/10
1875/1875 [==============================] - 51s 27ms/step - loss: 0.3712 - accuracy:
0.8638 - val_loss: 0.3395 - val_accuracy: 0.8729
Epoch 3/10
1875/1875 [==============================] - 51s 27ms/step - loss: 0.3210 - accuracy:
0.8822 - val_loss: 0.2969 - val_accuracy: 0.8942
Epoch 4/10
1875/1875 [==============================] - 51s 27ms/step - loss: 0.2952 - accuracy:
0.8912 - val_loss: 0.2805 - val_accuracy: 0.8947
Epoch 5/10
1875/1875 [==============================] - 52s 28ms/step - loss: 0.2742 - accuracy:
0.8981 - val_loss: 0.2626 - val_accuracy: 0.9029
Epoch 6/10
1875/1875 [==============================] - 52s 28ms/step - loss: 0.2619 - accuracy:
0.9021 - val_loss: 0.2720 - val_accuracy: 0.9000
Epoch 7/10
1875/1875 [==============================] - 53s 28ms/step - loss: 0.2519 - accuracy:
0.9057 - val_loss: 0.2544 - val_accuracy: 0.9089
Epoch 8/10
1875/1875 [==============================] - 53s 28ms/step - loss: 0.2420 - accuracy:
0.9088 - val_loss: 0.2545 - val_accuracy: 0.9087
Epoch 9/10
1875/1875 [==============================] - 53s 28ms/step - loss: 0.2333 - accuracy:
0.9123 - val_loss: 0.2581 - val_accuracy: 0.9080
Epoch 10/10
1875/1875 [==============================] - 54s 29ms/step - loss: 0.2259 - accuracy:
0.9159 - val_loss: 0.2636 - val_accuracy: 0.9076
```

In [12]:

```python
# Finally, we will evaluate the performance of the model on the test data.

test_loss, test_acc = model.evaluate(x_test, y_test)

print('Test accuracy:', test_acc)
```

```
313/313 [==============================] - 3s 11ms/step - loss: 0.2636 - accuracy: 0.90
76
Test accuracy: 0.9075999855995178
```