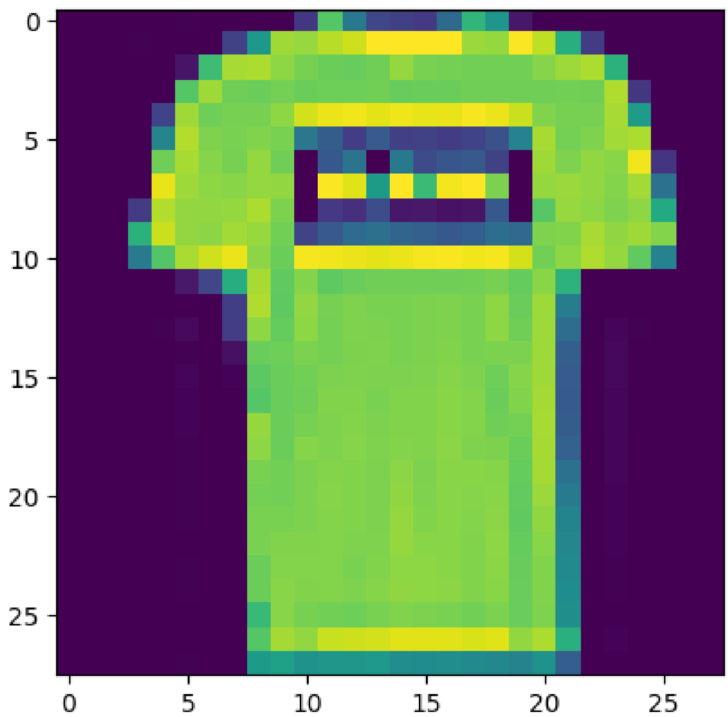
In [1]:
```
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow import keras
import numpy as np
```

In [2]:
```
(x_train, y_train), (x_test, y_test) = keras.datasets.fashion_mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labe
ls-idx1-ubyte.gz
29515/29515 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-imag
es-idx3-ubyte.gz
26421880/26421880 [==============================] - 6s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-label
s-idx1-ubyte.gz
5148/5148 [==============================] - 0s 0s/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-image
s-idx3-ubyte.gz
4422102/4422102 [==============================] - 1s 0us/step
```

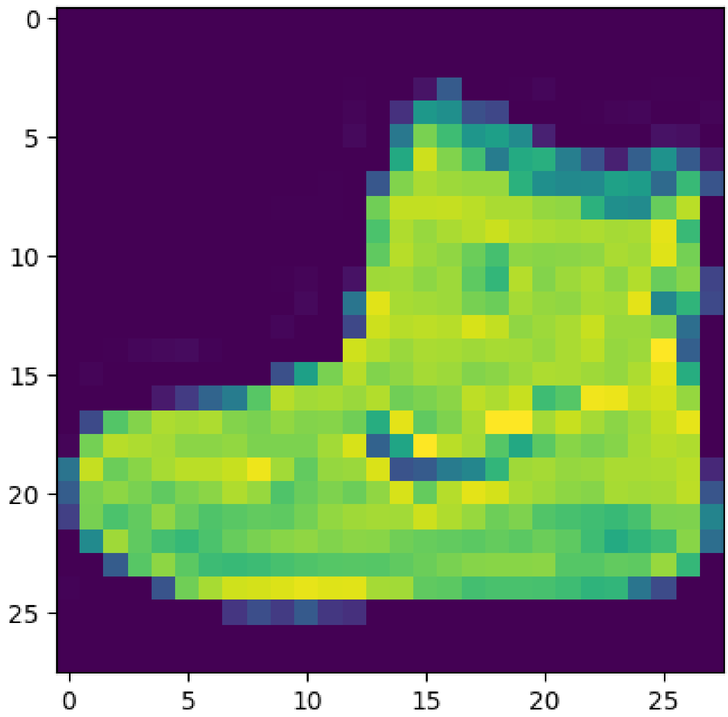
In [3]:
```
plt.imshow(x_train[1])
```

Out[3]: `<matplotlib.image.AxesImage at 0x275423e9e80>`



In [4]:
```
plt.imshow(x_train[0])
```

Out[4]: `<matplotlib.image.AxesImage at 0x2754aa3faf0>`



In [ ]:
```
# Next, we will preprocess the data by scaling the pixel values to be between 0 and 1,
# and then reshaping the images to be 28x28 pixels.
```

In [7]:
```
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)
```

In [ ]:
```
# 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
# Tensor when reshaped must remain the same.

# Each image is a row vector (784 elements) and there are lots of such rows (let it be n,
# so there are 784n elements). So TensorFlow can infer that -1 is n.

# converting the training_images array to 4 dimensional array with sizes
# 60000, 28, 28, 1 for 0th to 3rd dimension.
```

In [11]:
```
x_train.shape
(60000, 28, 28)
x_test.shape
(10000, 28, 28, 1)
y_train.shape
(60000,)
y_test.shape
(10000,)
```

Out[11]: (10000,)

In [12]:
```
# 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
    # filters
    # 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
    # filtter or channel
    # in Paramter shwon 18496 is value of weights: (3x3 filter weights + 64 bias) * 64
    # filters
    # 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 Depth
    # or filtter or channel
    # 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

    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
])
model.summary()
Model: "sequential"
```

```
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 [13]:

```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=['accuracy'
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 = 1875
# Finally, we will evaluate the performance of the model on the test data.
```

```
Epoch 1/10
1875/1875 [==============================] - 33s 17ms/step - loss: 0.5614 - accuracy: 0.7907
- val_loss: 0.3702 - val_accuracy: 0.8652
Epoch 2/10
1875/1875 [==============================] - 33s 17ms/step - loss: 0.3671 - accuracy: 0.8656
- val_loss: 0.3259 - val_accuracy: 0.8840
Epoch 3/10
1875/1875 [==============================] - 33s 17ms/step - loss: 0.3213 - accuracy: 0.8812
- val_loss: 0.2949 - val_accuracy: 0.8940
Epoch 4/10
1875/1875 [==============================] - 35s 18ms/step - loss: 0.2932 - accuracy: 0.8930
- val_loss: 0.2802 - val_accuracy: 0.8994
Epoch 5/10
1875/1875 [==============================] - 38s 20ms/step - loss: 0.2767 - accuracy: 0.8969
- val_loss: 0.2698 - val_accuracy: 0.8979
Epoch 6/10
1875/1875 [==============================] - 32s 17ms/step - loss: 0.2611 - accuracy: 0.9022
- val_loss: 0.2700 - val_accuracy: 0.9001
Epoch 7/10
1875/1875 [==============================] - 34s 18ms/step - loss: 0.2481 - accuracy: 0.9077
- val_loss: 0.2500 - val_accuracy: 0.9081
Epoch 8/10
1875/1875 [==============================] - 34s 18ms/step - loss: 0.2391 - accuracy: 0.9102
- val_loss: 0.2560 - val_accuracy: 0.9048
Epoch 9/10
1875/1875 [==============================] - 34s 18ms/step - loss: 0.2318 - accuracy: 0.9137
- val_loss: 0.2569 - val_accuracy: 0.9067
Epoch 10/10
1875/1875 [==============================] - 32s 17ms/step - loss: 0.2258 - accuracy: 0.9143
- val_loss: 0.2507 - val_accuracy: 0.9079
```

In [14]:

```
test_loss, test_acc = model.evaluate(x_test, y_test)
print('Test accuracy:', test_acc)
```

```
313/313 [==============================] - 2s 5ms/step - loss: 0.2507 - accuracy: 0.9079
Test accuracy: 0.9078999757766724
```

In [ ]: