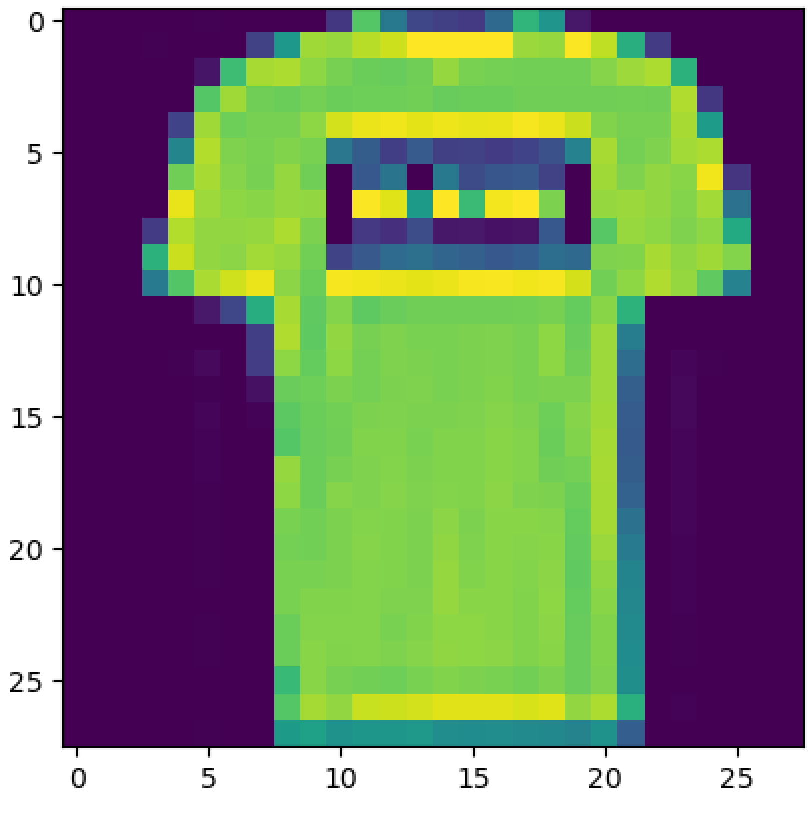
In [5]:
```
#Practical No:03
#Use MNIST Fashion Dataset and create a classifier to classify fashion c
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow import keras
import numpy as np
```

```
WARNING:tensorflow:From C:\Users\Ekata\AppData\Roaming\Python\Python310
\site-packages\keras\src\losses.py:2976: The name tf.losses.sparse_soft
max_cross_entropy is deprecated. Please use tf.compat.v1.losses.sparse_
softmax_cross_entropy instead.
```

In [6]:
```
(x_train, y_train), (x_test, y_test) = keras.datasets.fashion_mnist.load
```

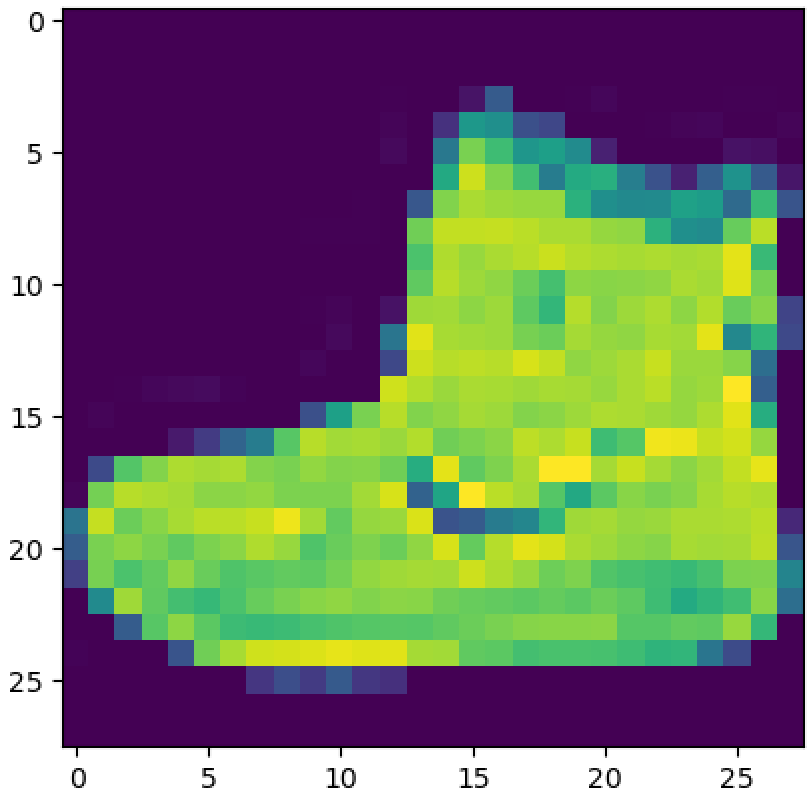
In [7]:
```
# There are 10 image classes in this dataset and each class has a mappin
#0 T-shirt/top
#1 Trouser
#2 pullover
#3 Dress
#4 Coat
#5 sandals
#6 shirt
#7 sneaker
#8 bag
#9 ankle boot
plt.imshow(x_train[1])
```

Out[7]:
```
<matplotlib.image.AxesImage at 0x116e4b3c280>
```



In [8]:
```
plt.imshow(x_train[0])
```

Out[8]:
```
<matplotlib.image.AxesImage at 0x116d0ecc370>
```



In [9]:
```
# Next, we will preprocess the data by scaling the pixel values to be be
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
# each image is a row vector (784 elements) and there are lots of such r
#elements). So TensorFlow can infer that -1 is n.
# converting the training_images array to 4 dimensional array with sizes
#dimension.
x_train.shape
(60000, 28, 28)
x_test.shape
(10000, 28, 28, 1)
y_train.shape
(60000,)
y_test.shape
(10000,)
```

Out[9]:
```
(10000,)
```

In [10]:
```
# We will use a convolutional neural network (CNN) to classify the fashi
# The CNN will consist of multiple convolutional layers followed by max
# 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)),keras.layers.MaxPooling2D((2,
    keras.layers.Dropout(0.25),keras.layers.Conv2D(64, (3,3), activation
    keras.layers.MaxPooling2D((2,2)),keras.layers.Dropout(0.25),
    keras.layers.Conv2D(128, (3,3), activation='relu'),keras.layers.Flat
keras.layers.Dense(128, activation='relu'),keras.layers.Dropout(0.25),
keras.layers.Dense(10, activation='softmax')])
model.summary()
```

```
WARNING:tensorflow:From C:\Users\Ekata\AppData\Roaming\Python\Python310
\site-packages\keras\src\backend.py:873: The name tf.get_default_graph
is deprecated. Please use tf.compat.v1.get_default_graph instead.

WARNING:tensorflow:From C:\Users\Ekata\AppData\Roaming\Python\Python310
\site-packages\keras\src\layers\pooling\max_pooling2d.py:161: The name
tf.nn.max_pool is deprecated. Please use tf.nn.max_pool2d instead.

Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 32)        320

 max_pooling2d (MaxPooling2  (None, 13, 13, 32)        0
 D)

 dropout (Dropout)           (None, 13, 13, 32)        0

 conv2d_1 (Conv2D)           (None, 11, 11, 64)        18496

 max_pooling2d_1 (MaxPoolin  (None, 5, 5, 64)          0
 g2D)

 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: 241546 (943.54 KB)
Trainable params: 241546 (943.54 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [12]:
```
# Compile and Train the Model
# After defining the model, we will compile it and train it on the train
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
history = model.fit(x_train, y_train, epochs=10, validation_data=(x_test
test_loss, test_acc = model.evaluate(x_test, y_test)
print('Test accuracy:', test_acc)
```

```
Epoch 1/10
1875/1875 [==============================] - 38s 20ms/step - loss: 0.22
59 - accuracy: 0.9143 - val_loss: 0.2517 - val_accuracy: 0.9114
Epoch 2/10
1875/1875 [==============================] - 45s 24ms/step - loss: 0.21
95 - accuracy: 0.9164 - val_loss: 0.2580 - val_accuracy: 0.9101
Epoch 3/10
1875/1875 [==============================] - 41s 22ms/step - loss: 0.21
54 - accuracy: 0.9189 - val_loss: 0.2590 - val_accuracy: 0.9061
Epoch 4/10
1875/1875 [==============================] - 34s 18ms/step - loss: 0.20
76 - accuracy: 0.9215 - val_loss: 0.2609 - val_accuracy: 0.9064
Epoch 5/10
1875/1875 [==============================] - 45s 24ms/step - loss: 0.20
56 - accuracy: 0.9209 - val_loss: 0.2532 - val_accuracy: 0.9122
Epoch 6/10
1875/1875 [==============================] - 34s 18ms/step - loss: 0.20
36 - accuracy: 0.9228 - val_loss: 0.2466 - val_accuracy: 0.9125
Epoch 7/10
1875/1875 [==============================] - 33s 18ms/step - loss: 0.19
81 - accuracy: 0.9243 - val_loss: 0.2514 - val_accuracy: 0.9118
Epoch 8/10
1875/1875 [==============================] - 30s 16ms/step - loss: 0.19
49 - accuracy: 0.9257 - val_loss: 0.2547 - val_accuracy: 0.9120
Epoch 9/10
1875/1875 [==============================] - 33s 17ms/step - loss: 0.19
32 - accuracy: 0.9254 - val_loss: 0.2580 - val_accuracy: 0.9157
Epoch 10/10
1875/1875 [==============================] - 32s 17ms/step - loss: 0.18
86 - accuracy: 0.9290 - val_loss: 0.2552 - val_accuracy: 0.9155
313/313 [==============================] - 2s 7ms/step - loss: 0.2552 -
accuracy: 0.9155
Test accuracy: 0.9154999852180481
```

In [ ]: