br6ihc4ns

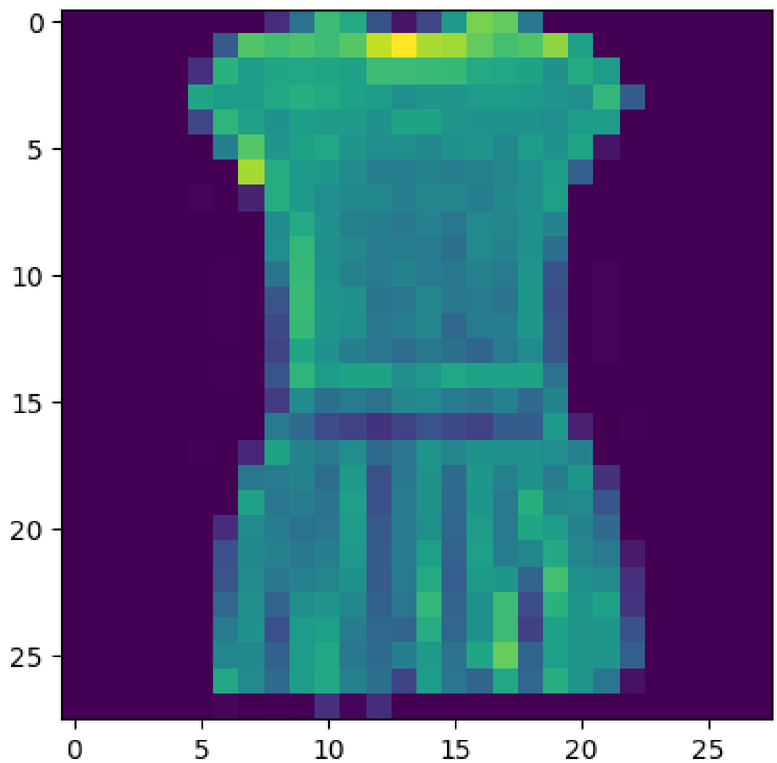
March 20, 2025

[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()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/train-labels-idx1-ubyte.gz
29515/29515            0s 2us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/train-images-idx3-ubyte.gz
26421880/26421880            7s
0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/t10k-labels-idx1-ubyte.gz
5148/5148            0s 1us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/t10k-images-idx3-ubyte.gz
4422102/4422102            3s
1us/step
```

[2]:
```
plt.imshow(x_train[3])
```

[2]:
```
<matplotlib.image.AxesImage at 0x1e14da76d50>
```



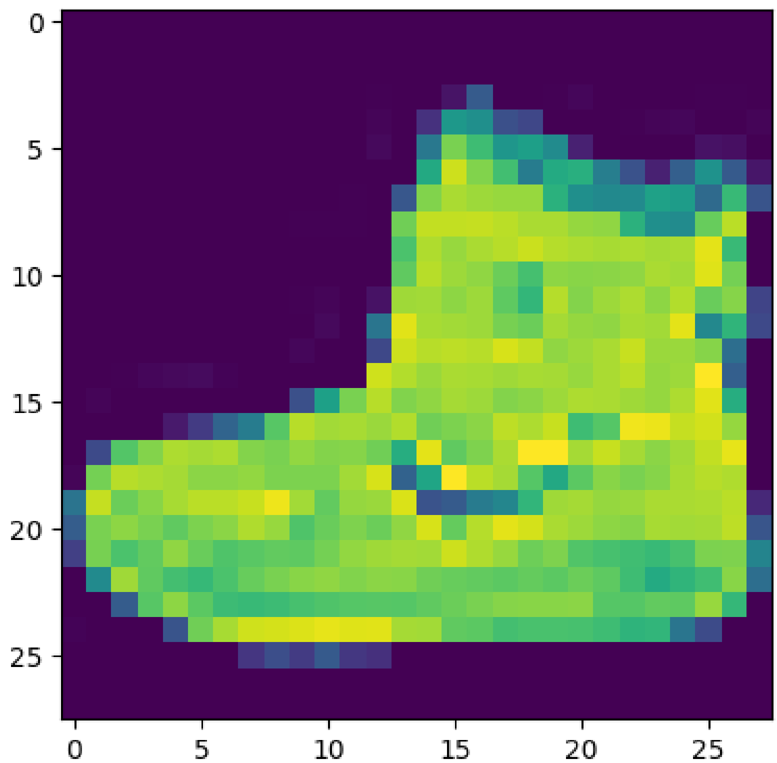
plt.imshow(): This is a function from the matplotlib library, specifically used to display images. It takes an array-like object as input and renders it as an image.

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.

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

[3]: <matplotlib.image.AxesImage at 0x1e15613b210>



[4]:
```
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)
```

[5]:
```
x_train.shape
```

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

[6]:
```
x_test.shape
```

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

[7]:
```
y_train.shape
```

[7]:
```
(60000,)
```

[8]:
```
y_test.shape
```

[8]:
```
(10000,)
```

[9]:
```
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
    # https://medium.com/@iamvarman/
↪how-to-calculate-the-number-of-parameters-in-the-cnn-5bd55364d7ca
```

```
    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
])
```

```
C:\Users\arana\AppData\Roaming\Python\Python311\site-
packages\keras\src\layers\convolutional\base_conv.py:107: UserWarning: Do not
pass an `input_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in the model
instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

[10]:
```
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)                   ␣
↪18,496

 max_pooling2d_1 (MaxPooling2D)      (None, 5, 5, 64)                        ␣
↪  0

 dropout_1 (Dropout)                 (None, 5, 5, 64)                        ␣
↪  0

 conv2d_2 (Conv2D)                   (None, 3, 3, 128)                    ␣
↪73,856
```

```
 flatten (Flatten)                    (None, 1152)                                  ␣
↪  0

 dense (Dense)                        (None, 128)                                ␣
↪147,584

 dropout_2 (Dropout)                  (None, 128)                                   ␣
↪  0

 dense_1 (Dense)                      (None, 10)                                  ␣
↪1,290
```

Total params: 241,546 (943.54 KB)

Trainable params: 241,546 (943.54 KB)

Non-trainable params: 0 (0.00 B)

[11]:

```
# 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
```

```
Epoch 1/10
1875/1875            119s 58ms/step
- accuracy: 0.7140 - loss: 0.7703 - val_accuracy: 0.8585 - val_loss: 0.3809
Epoch 2/10
1875/1875            108s 57ms/step
- accuracy: 0.8559 - loss: 0.3882 - val_accuracy: 0.8702 - val_loss: 0.3459
Epoch 3/10
1875/1875            98s 52ms/step -
accuracy: 0.8749 - loss: 0.3348 - val_accuracy: 0.8895 - val_loss: 0.3014
Epoch 4/10
1875/1875            80s 43ms/step -
accuracy: 0.8892 - loss: 0.3010 - val_accuracy: 0.8963 - val_loss: 0.2866
Epoch 5/10
1875/1875            74s 39ms/step -
accuracy: 0.8969 - loss: 0.2774 - val_accuracy: 0.8987 - val_loss: 0.2761
```

```
Epoch 6/10
1875/1875            105s 56ms/step
- accuracy: 0.8998 - loss: 0.2693 - val_accuracy: 0.8955 - val_loss: 0.2798
Epoch 7/10
1875/1875            109s 58ms/step
- accuracy: 0.9029 - loss: 0.2574 - val_accuracy: 0.9055 - val_loss: 0.2542
Epoch 8/10
1875/1875            120s 64ms/step
- accuracy: 0.9111 - loss: 0.2385 - val_accuracy: 0.9030 - val_loss: 0.2617
Epoch 9/10
1875/1875            143s 64ms/step
- accuracy: 0.9108 - loss: 0.2376 - val_accuracy: 0.9074 - val_loss: 0.2482
Epoch 10/10
1875/1875            72s 38ms/step -
accuracy: 0.9149 - loss: 0.2260 - val_accuracy: 0.9073 - val_loss: 0.2477
```

[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 10ms/step -
accuracy: 0.9042 - loss: 0.2548
Test accuracy: 0.9072999954223633
```