```
import tensorflow as tf
```

```
fashion_mnist = tf.keras.datasets.fashion_mnist
# mnist = tf.keras.datasets.mnist
```

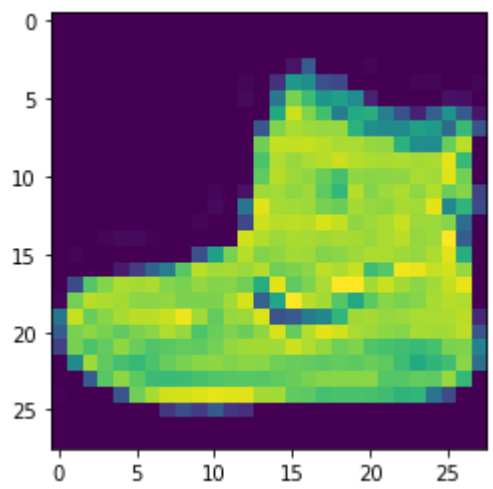
```
(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()
# (x_train, y_train), (x_test, y_test) = mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-idx1-ubyte.gz
32768/29515 [================================] - 0s 0us/step
40960/29515 [========================================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-idx3-ubyte.gz
26427392/26421880 [==============================] - 0s 0us/step
26435584/26421880 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-labels-idx1-ubyte.gz
16384/5148 [================================================================================] - 0s 0us/s
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-images-idx3-ubyte.gz
4423680/4422102 [==============================] - 0s 0us/step
4431872/4422102 [==============================] - 0s 0us/step
```

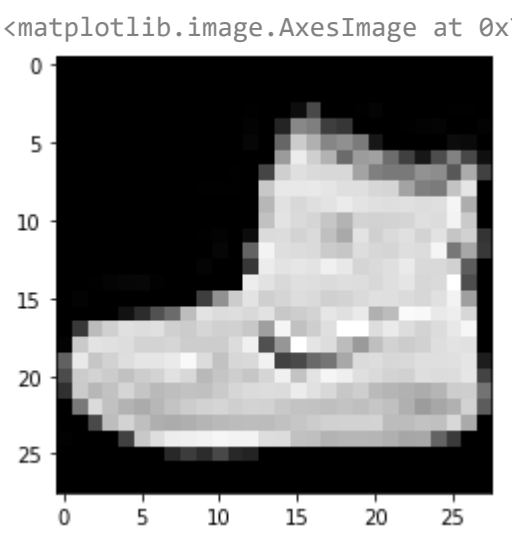
```
print(x_train.shape)
print(y_train.shape)
print(x_test.shape)
print(y_test.shape)
```

```
(60000, 28, 28)
(60000,)
(10000, 28, 28)
(10000,)
```

```
import matplotlib.pyplot as plt

plt.imshow(x_train[0])
plt.show()

plt.imshow(x_train[0], cmap=plt.get_cmap('gray'))
```




```
<matplotlib.image.AxesImage at 0x7f9bc720c590>
```



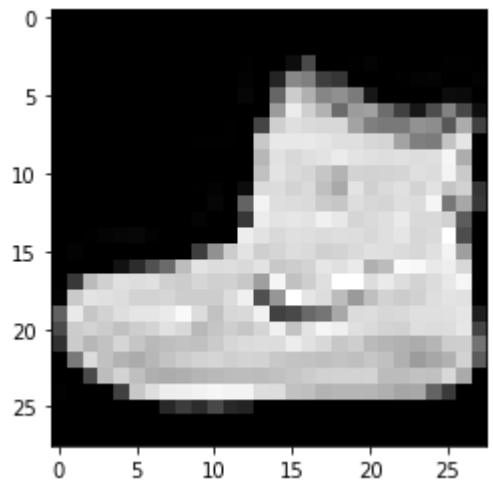

```
print(x_train[0])
```

```
[[  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   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   1   0   0  13  73   0
    0   1   4   0   0   0   0   1   1   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   3   0  36 136 127  62
   54   0   0   0   1   3   4   0   0   3]
```

```
[  0   0   0   0   0   0   0   0   0   0   0   0   6   0 102 204 176 134
 144 123  23   0   0   0   0  12  10   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0 155 236 207 178
 107 156 161 109  64  23  77 130  72  15]
[  0   0   0   0   0   0   0   0   0   0   0   1   0  69 207 223 218 216
 216 163 127 121 122 146 141  88 172  66]
[  0   0   0   0   0   0   0   0   0   1   1   1   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   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]]
```

```
1 x_train = x_train/255
2 x_test = x_test/255
3 plt.imshow(x_train[0], cmap=plt.get_cmap('gray'))
```

```
<matplotlib.image.AxesImage at 0x7f9bc7195610>
```




```
1 print(x_test[0])
```

```
 0.64705882 0.66666667 0.60392157 0.59215686 0.60392157 0.56078431
 0.54117647 0.58823529 0.64705882 0.16862745]
[0.         0.         0.09019608 0.21176471 0.25490196 0.29803922
 0.33333333 0.4627451  0.50196078 0.48235294 0.43529412 0.44313725
 0.4627451  0.49803922 0.49019608 0.54509804 0.52156863 0.53333333
 0.62745098 0.54901961 0.60784314 0.63137255 0.56470588 0.60784314
 0.6745098  0.63137255 0.74117647 0.24313725]
[0.         0.26666667 0.36862745 0.35294118 0.43529412 0.44705882
 0.43529412 0.44705882 0.45098039 0.49803922 0.52941176 0.53333333
 0.56078431 0.49411765 0.49803922 0.59215686 0.60392157 0.56078431
 0.58039216 0.49019608 0.63529412 0.63529412 0.56470588 0.54117647
 0.6        0.63529412 0.76862745 0.22745098]
[0.2745098  0.6627451  0.50588235 0.40784314 0.38431373 0.39215686
 0.36862745 0.38039216 0.38431373 0.4        0.42352941 0.41568627
 0.46666667 0.47058824 0.50588235 0.58431373 0.61176471 0.65490196
 0.74509804 0.74509804 0.76862745 0.77647059 0.77647059 0.73333333
 0.77254902 0.74117647 0.72156863 0.14117647]
[0.0627451  0.49411765 0.67058824 0.7372549  0.7372549  0.72156863
```

```
  0.67058824 0.6        0.52941176 0.47058824 0.49411765 0.49803922
  0.57254902 0.7254902  0.76470588 0.81960784 0.81568627 1.
  0.81960784 0.69411765 0.96078431 0.98823529 0.98431373 0.98431373
  0.96862745 0.8627451  0.80784314 0.19215686]
 [0.         0.         0.         0.04705882 0.2627451  0.41568627
  0.64313725 0.7254902  0.78039216 0.82352941 0.82745098 0.82352941
  0.81568627 0.74509804 0.58823529 0.32156863 0.03137255 0.
  0.         0.         0.69803922 0.81568627 0.7372549  0.68627451
  0.63529412 0.61960784 0.59215686 0.04313725]
 [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.         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.         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.        ]]
```

```python
print(y_train[1], y_test[1])
```

```
0 2
```

```python
import numpy as np

x_trainr = np.array(x_train).reshape(60000, 28, 28, 1)
x_testr = np.array(x_test).reshape(10000, 28, 28, 1)

print("Training sample dimention: ", x_trainr.shape)
print("Testing sample dimention: ", x_testr.shape)
```

```
Training sample dimention:  (60000, 28, 28, 1)
Testing sample dimention:  (10000, 28, 28, 1)
```

```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation, Flatten, Conv2D, MaxPooling2D, Dropout
```

```python

```

Model Create

```python
from tensorflow.keras.callbacks import ModelCheckpoint
fmnist_model_checkpoint = ModelCheckpoint('FMNIST_Weight.h5',
                                   save_best_only=True,
                                   monitor='loss',
                                   verbose=1
                                   )
```

```python
model = Sequential()
```

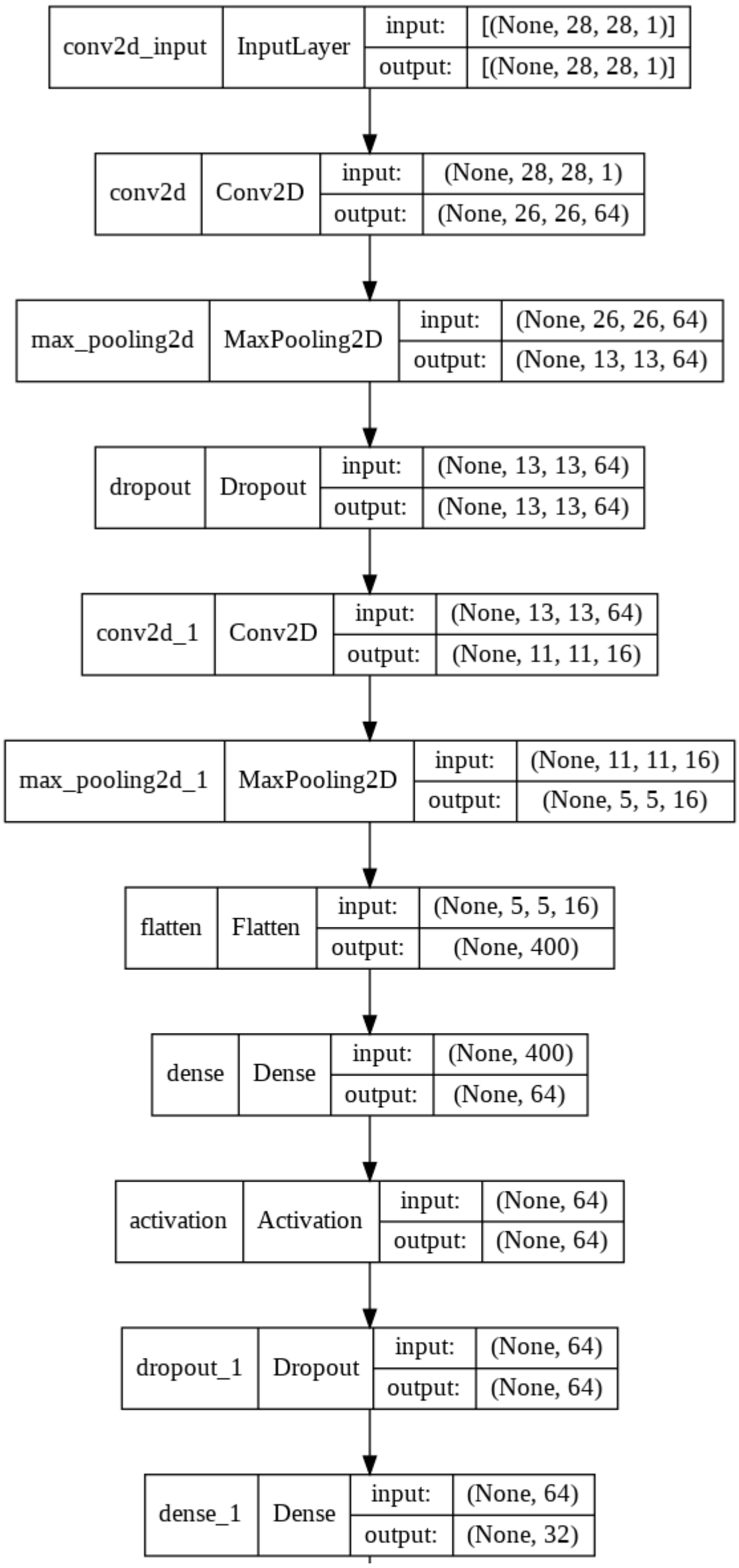
```python
# 1st Layer
model.add(Conv2D(64, (3,3), input_shape = (28,28,1), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))

# model.add(Dropout(rate=0.1))
```

```
# 2nd Layer
# model.add(Conv2D(32, (3,3), activation='relu'))
# model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Dropout(rate=0.1))
# 3rd Layer
model.add(Conv2D(16, (3,3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Flatten())

model.add(Dense(64))
model.add(Activation('relu'))

model.add(Dropout(rate=0.1))

model.add(Dense(32))
model.add(Activation('relu'))

model.add(Dense(10))
model.add(Activation('softmax'))
```

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 26, 26, 64)        640

max_pooling2d (MaxPooling2D  (None, 13, 13, 64)       0
)

dropout (Dropout)            (None, 13, 13, 64)        0

conv2d_1 (Conv2D)            (None, 11, 11, 16)        9232

max_pooling2d_1 (MaxPooling  (None, 5, 5, 16)         0
2D)

flatten (Flatten)            (None, 400)               0

dense (Dense)                (None, 64)                25664

activation (Activation)      (None, 64)                0

dropout_1 (Dropout)          (None, 64)                0

dense_1 (Dense)              (None, 32)                2080

activation_1 (Activation)    (None, 32)                0

dense_2 (Dense)              (None, 10)                330

activation_2 (Activation)    (None, 10)                0

=================================================================
Total params: 37,946
Trainable params: 37,946
Non-trainable params: 0
_________________________________________________________________
```

```
from keras.utils.vis_utils import plot_model
plot_model(model, to_file='model.png', show_shapes=True, show_layer_names=True)
```




```
1 print("Total training sample: ", len(x_trainr))
```

```
Total training sample:  60000
```

```
1 model.compile(optimizer="adam", loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

```
1 history = model.fit(x_trainr, y_train, batch_size=512, epochs=100, validation_split = 0.3, callbacks=[fmnist_mod
```

```
Epoch 1/100
83/83 [==============================] - ETA: 0s - loss: 1.1652 - accuracy: 0.5737
Epoch 00001: loss improved from inf to 1.16517, saving model to FMNIST_Weight.h5
83/83 [==============================] - 17s 21ms/step - loss: 1.1652 - accuracy: 0.5737 - val_loss: 0.6290 - val_acc
Epoch 2/100
82/83 [============================>.] - ETA: 0s - loss: 0.6091 - accuracy: 0.7697
Epoch 00002: loss improved from 1.16517 to 0.60909, saving model to FMNIST_Weight.h5
83/83 [==============================] - 1s 16ms/step - loss: 0.6091 - accuracy: 0.7697 - val_loss: 0.5407 - val_accu
Epoch 3/100
81/83 [============================>.] - ETA: 0s - loss: 0.5213 - accuracy: 0.8061
```

```
Epoch 00003: loss improved from 0.60909 to 0.52070, saving model to FMNIST_Weight.h5
83/83 [==============================] - 1s 18ms/step - loss: 0.5207 - accuracy: 0.8061 - val_loss: 0.4645 - val_accu
Epoch 4/100
81/83 [============================>.] - ETA: 0s - loss: 0.4697 - accuracy: 0.8275
Epoch 00004: loss improved from 0.52070 to 0.46931, saving model to FMNIST_Weight.h5
83/83 [==============================] - 1s 17ms/step - loss: 0.4693 - accuracy: 0.8276 - val_loss: 0.4281 - val_accu
Epoch 5/100
81/83 [============================>.] - ETA: 0s - loss: 0.4359 - accuracy: 0.8425
Epoch 00005: loss improved from 0.46931 to 0.43518, saving model to FMNIST_Weight.h5
83/83 [==============================] - 1s 17ms/step - loss: 0.4352 - accuracy: 0.8427 - val_loss: 0.4169 - val_accu
Epoch 6/100
81/83 [============================>.] - ETA: 0s - loss: 0.4129 - accuracy: 0.8490
Epoch 00006: loss improved from 0.43518 to 0.41253, saving model to FMNIST_Weight.h5
83/83 [==============================] - 1s 17ms/step - loss: 0.4125 - accuracy: 0.8491 - val_loss: 0.3827 - val_accu
Epoch 7/100
81/83 [============================>.] - ETA: 0s - loss: 0.3851 - accuracy: 0.8604
Epoch 00007: loss improved from 0.41253 to 0.38545, saving model to FMNIST_Weight.h5
83/83 [==============================] - 1s 18ms/step - loss: 0.3854 - accuracy: 0.8603 - val_loss: 0.3756 - val_accu
Epoch 8/100
81/83 [============================>.] - ETA: 0s - loss: 0.3780 - accuracy: 0.8613
Epoch 00008: loss improved from 0.38545 to 0.37751, saving model to FMNIST_Weight.h5
83/83 [==============================] - 1s 16ms/step - loss: 0.3775 - accuracy: 0.8614 - val_loss: 0.3611 - val_accu
Epoch 9/100
81/83 [============================>.] - ETA: 0s - loss: 0.3609 - accuracy: 0.8685
Epoch 00009: loss improved from 0.37751 to 0.36054, saving model to FMNIST_Weight.h5
83/83 [==============================] - 1s 16ms/step - loss: 0.3605 - accuracy: 0.8686 - val_loss: 0.3479 - val_accu
Epoch 10/100
81/83 [============================>.] - ETA: 0s - loss: 0.3475 - accuracy: 0.8733
Epoch 00010: loss improved from 0.36054 to 0.34818, saving model to FMNIST_Weight.h5
83/83 [==============================] - 1s 16ms/step - loss: 0.3482 - accuracy: 0.8729 - val_loss: 0.3444 - val_accu
Epoch 11/100
81/83 [============================>.] - ETA: 0s - loss: 0.3412 - accuracy: 0.8748
Epoch 00011: loss improved from 0.34818 to 0.34145, saving model to FMNIST_Weight.h5
83/83 [==============================] - 1s 16ms/step - loss: 0.3414 - accuracy: 0.8747 - val_loss: 0.3577 - val_accu
Epoch 12/100
81/83 [============================>.] - ETA: 0s - loss: 0.3336 - accuracy: 0.8784
Epoch 00012: loss improved from 0.34145 to 0.33376, saving model to FMNIST_Weight.h5
83/83 [==============================] - 1s 16ms/step - loss: 0.3338 - accuracy: 0.8785 - val_loss: 0.3336 - val_accu
Epoch 13/100
81/83 [============================>.] - ETA: 0s - loss: 0.3276 - accuracy: 0.8790
Epoch 00013: loss improved from 0.33376 to 0.32670, saving model to FMNIST_Weight.h5
83/83 [==============================] - 1s 16ms/step - loss: 0.3267 - accuracy: 0.8792 - val_loss: 0.3286 - val_accu
Epoch 14/100
81/83 [============================>.] - ETA: 0s - loss: 0.3155 - accuracy: 0.8845
Epoch 00014: loss improved from 0.32670 to 0.31606, saving model to FMNIST_Weight.h5
83/83 [==============================] - 1s 18ms/step - loss: 0.3161 - accuracy: 0.8845 - val_loss: 0.3220 - val_accu
Epoch 15/100
```

```
1 import os.path
2 if os.path.isfile("/content/FMNIST_Weight.h5") is False:
3     model.save_weights("/content/FMNIST_Weight.h5")
```

```
1 test_loss, test_acc = model.evaluate(x_testr, y_test)
2 print("Loss: ", test_loss)
3 print("Accuracy: ", test_acc)
```

```
313/313 [==============================] - 1s 2ms/step - loss: 0.3191 - accuracy: 0.9054
Loss:  0.3191052973270416
Accuracy:  0.9053999781608582
```

```
1 predictions = model.predict([x_testr])
```

```
1 print(predictions)
```

```
[[6.33567157e-11 5.41424327e-13 2.98562286e-10 ... 3.34859624e-06
  2.53699123e-12 9.99996662e-01]
 [7.25159308e-11 1.79561100e-22 9.99998808e-01 ... 2.37812415e-24
  3.71594243e-15 3.57003010e-23]
 [6.18690670e-19 1.00000000e+00 2.07815865e-25 ... 1.47642933e-28
  1.05599965e-25 2.94077177e-27]
 ...
 [4.85646289e-12 7.21230174e-15 5.09109729e-14 ... 3.17227364e-13
  1.00000000e+00 2.00905644e-17]
 [1.15037035e-10 1.00000000e+00 1.62806698e-13 ... 3.97043518e-15
  5.28763409e-15 2.22555620e-15]
 [3.71262239e-08 3.03794455e-11 1.25323723e-07 ... 1.22356942e-04
  1.66767973e-06 2.48970498e-08]]
```

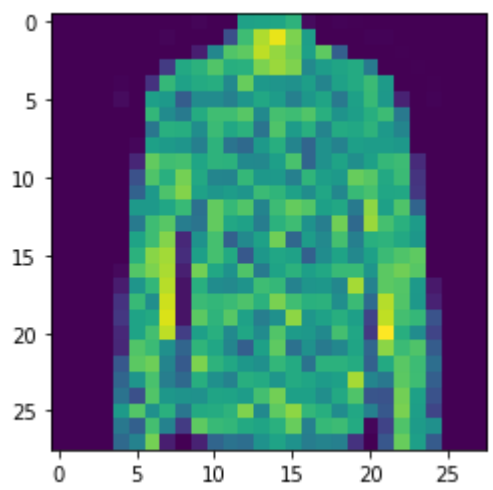
```
1 pred = np.argmax(predictions[1])
```

```
1 print(pred)
```

```
2
```

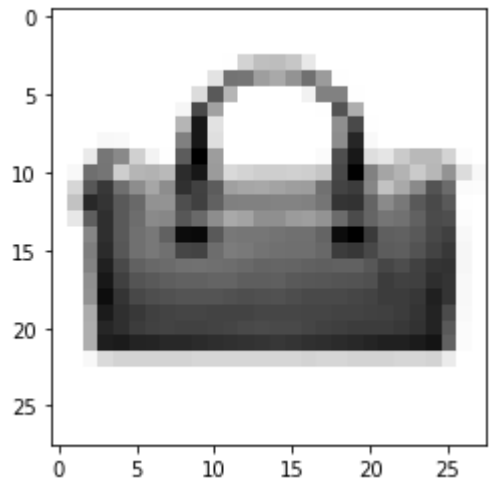
```
plt.imshow(x_test[101])
```

```
<matplotlib.image.AxesImage at 0x7f9b4a4a3f10>
```



```
image_class = {}
image_class[0] = 'T-shirt/top'
image_class[1] = 'Trouser'
image_class[2] = 'Pullover'
image_class[3] = 'Dress'
image_class[4] = 'Coat'
image_class[5] = 'Sandal'
image_class[6] = 'Shirt'
image_class[7] = 'Sneaker'
image_class[8] = 'Bag'
image_class[9] = 'Ankle boot'
```

```
from PIL import Image

# for i in range(10):
demo_image = "/content/drive/MyDrive/University/12th Semester/CSI 416 [Pattern Recognition Lab]/Project/raw_imag
img = Image.open(demo_image)


img = img.resize((28, 28))
imgGray = img.convert('L')
imgGray.save('test_gray.jpg')

image_array = np.array(imgGray)

plt.imshow(imgGray, cmap=plt.get_cmap('gray'))
plt.show()

image_array = image_array/255

new_img = np.array(image_array).reshape(1, 28, 28, 1)    # reshape for kerner operation

test_pred = model.predict(new_img)

predict_class = np.argmax(test_pred)
print("Prediction: ", image_class[predict_class])
```



```
Prediction:  Bag
```

```
0 T-shirt/top
1 Trouser
2 Pullover
3 Dress
4 Coat
```

```
5 Sandal
6 Shirt
7 Sneaker
8 Bag
9 Ankle boot
```

```
1 print(len(test_pred[0]))
2 for i in range(10):
3     print(f"Similarity with {image_class[i]} is --> [{round(test_pred[0][i]*100, 4)} %]")
```

```
10
Similarity with T-shirt/top is --> [0.5728 %]
Similarity with Trouser is --> [0.0 %]
Similarity with Pullover is --> [0.044 %]
Similarity with Dress is --> [0.0191 %]
Similarity with Coat is --> [0.0001 %]
Similarity with Sandal is --> [0.0211 %]
Similarity with Shirt is --> [0.0952 %]
Similarity with Sneaker is --> [0.007 %]
Similarity with Bag is --> [98.9009 %]
Similarity with Ankle boot is --> [0.3397 %]
```

```
1 import tensorflow as tf
2 import numpy as np
3 import matplotlib.pyplot as plt
4 fashion_mnist = tf.keras.datasets.fashion_mnist
5 (x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()
6 x_train = x_train/255
7 x_test = x_test/255
8 x_trainr = np.array(x_train).reshape(60000, 28, 28, 1)
9 x_testr = np.array(x_test).reshape(10000, 28, 28, 1)
```

```
1 from tensorflow.keras.models import Sequential
2 from tensorflow.keras.layers import Dense, Activation, Flatten, Conv2D, MaxPooling2D, Dropout
```

Use Model Explicitely

```
1 model2 = Sequential()
```

```
1 # 1st Layer
2 model2.add(Conv2D(64, (3,3), input_shape = (28,28,1), activation='relu'))
3 model2.add(MaxPooling2D(pool_size=(2,2)))
4 # 2nd Layer
5 # model2.add(Conv2D(32, (3,3), activation='relu'))
6 # model2.add(MaxPooling2D(pool_size=(2,2)))
7 model2.add(Dropout(rate=0.1))
8 # 3rd Layer
9 model2.add(Conv2D(16, (3,3), activation='relu'))
10 model2.add(MaxPooling2D(pool_size=(2,2)))
11
12 model2.add(Flatten())
13
14 model2.add(Dense(64))
15 model2.add(Activation('relu'))
16
17 model2.add(Dropout(rate=0.1))
18
19 model2.add(Dense(32))
20 model2.add(Activation('relu'))
21
22 model2.add(Dense(10))
23 model2.add(Activation('softmax'))
```

```
1 model2.load_weights("/content/FMNIST_Weight.h5")
2 # model2.get_weights()
```

```
1 model2.compile(optimizer="adam", loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

```
1 test2_loss, test2_acc = model2.evaluate(x_testr, y_test)
2 print("Loss: ", test2_loss)
3 print("Accuracy: ", test2_acc)
```

```
313/313 [==============================] - 1s 3ms/step - loss: 0.3191 - accuracy: 0.9054
Loss:  0.3191052973270416
Accuracy:  0.9053999781608582
```

```
predictions2 = model2.predict([x_testr])
```

```
print(predictions2)
```

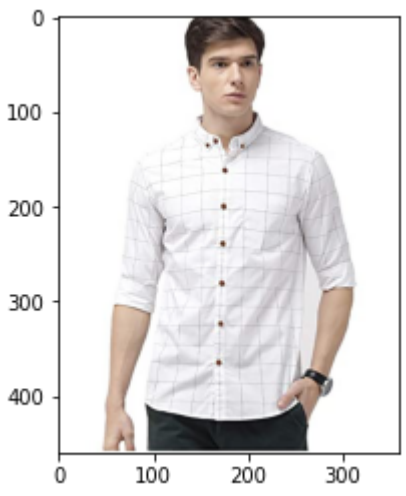
```
[[6.33567157e-11 5.41424327e-13 2.98562286e-10 ... 3.34859624e-06
  2.53699123e-12 9.99996662e-01]
 [7.25159308e-11 1.79561100e-22 9.99998808e-01 ... 2.37812415e-24
  3.71594243e-15 3.57003010e-23]
 [6.18690670e-19 1.00000000e+00 2.07815865e-25 ... 1.47642933e-28
  1.05599965e-25 2.94077177e-27]
 ...
 [4.85646289e-12 7.21230174e-15 5.09109729e-14 ... 3.17227364e-13
  1.00000000e+00 2.00905644e-17]
 [1.15037035e-10 1.00000000e+00 1.62806698e-13 ... 3.97043518e-15
  5.28763409e-15 2.22555620e-15]
 [3.71262239e-08 3.03794455e-11 1.25323723e-07 ... 1.22356942e-04
  1.66767973e-06 2.48970498e-08]]
```

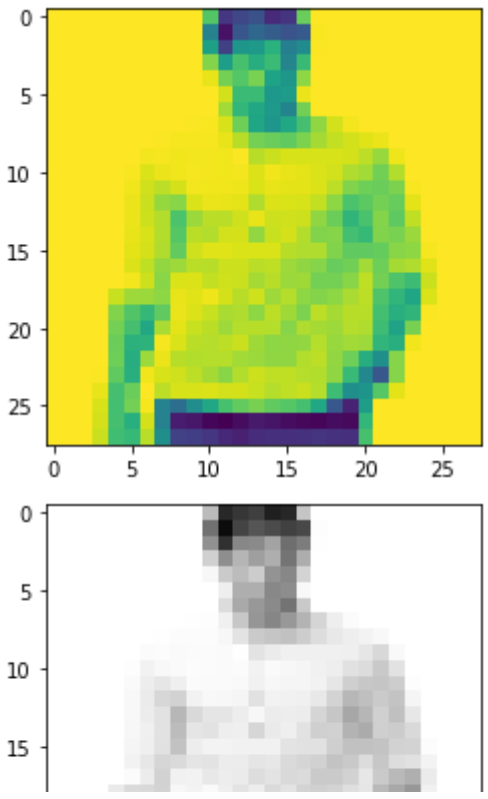
```
pred2 = np.argmax(predictions2[101])
print(pred2)
```

```
6
```

```
image_class = {}
image_class[0] = 'T-shirt/top'
image_class[1] = 'Trouser'
image_class[2] = 'Pullover'
image_class[3] = 'Dress'
image_class[4] = 'Coat'
image_class[5] = 'Sandal'
image_class[6] = 'Shirt'
image_class[7] = 'Sneaker'
image_class[8] = 'Bag'
image_class[9] = 'Ankle boot'
```

```
from PIL import Image

demo_image2 = "/content/drive/MyDrive/University/12th Semester/CSI 416 [Pattern Recognition Lab]/Project/raw_ima
img2 = Image.open(demo_image2)


plt.imshow(img2)
plt.show()

img2 = img2.resize((28, 28))
imgGray2 = img2.convert('L')
imgGray2.save('test_gray2.jpg')

plt.imshow(imgGray2)
plt.show()

image_array2 = np.array(imgGray2)

plt.imshow(imgGray2, cmap=plt.get_cmap('gray'))
plt.show()

image_array2 = image_array2/255

new_img2 = np.array(image_array2).reshape(1, 28, 28, 1)    # reshape for kerner operation

test_pred2 = model2.predict(new_img2)

label = np.argmax(test_pred2)

print("Predict: ", image_class[label])
```





Class Label

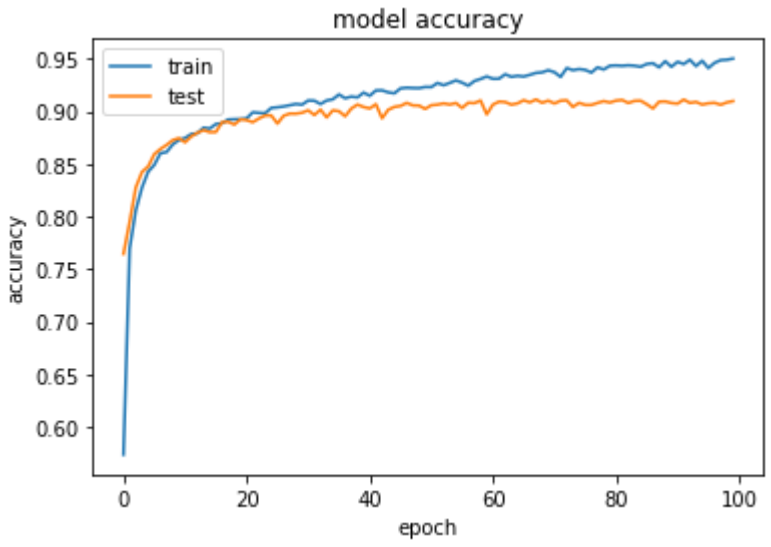
```
0 T-shirt/top
1 Trouser
2 Pullover
3 Dress
4 Coat
5 Sandal
6 Shirt
7 Sneaker
8 Bag
9 Ankle boot
```

```
1 print(len(test_pred2[0]))
2 for i in range(10):
3     print(f"Similarity with {image_class[i]} is --> [{round(test_pred2[0][i]*100, 4)} %]")
```

```
10
Similarity with T-shirt/top is --> [6.7574 %]
Similarity with Trouser is --> [0.0305 %]
Similarity with Pullover is --> [0.5813 %]
Similarity with Dress is --> [0.0573 %]
Similarity with Coat is --> [0.5569 %]
Similarity with Sandal is --> [0.0393 %]
Similarity with Shirt is --> [86.3899 %]
Similarity with Sneaker is --> [0.0031 %]
Similarity with Bag is --> [5.5588 %]
Similarity with Ankle boot is --> [0.0254 %]
```

```
1 # history.history.keys()
```

```
1 # summarize history for accuracy
2 plt.plot(history.history['accuracy'])
3 plt.plot(history.history['val_accuracy'])
4 plt.title('model accuracy')
5 plt.ylabel('accuracy')
6 plt.xlabel('epoch')
7 plt.legend(['train', 'test'], loc='upper left')
8 plt.show()
9
```




```
# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```