8) Aim: Train the Neural Network on a given datatset

In [3]:
```
import tensorflow
from tensorflow import keras
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense, Flatten
```

In [4]:
```
(X_train,y_train),(X_test,y_test) = keras.datasets.mnist.load_data()
```

In [5]:
```
X_train.shape
```

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

In [6]:
```
X_test.shape
```

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

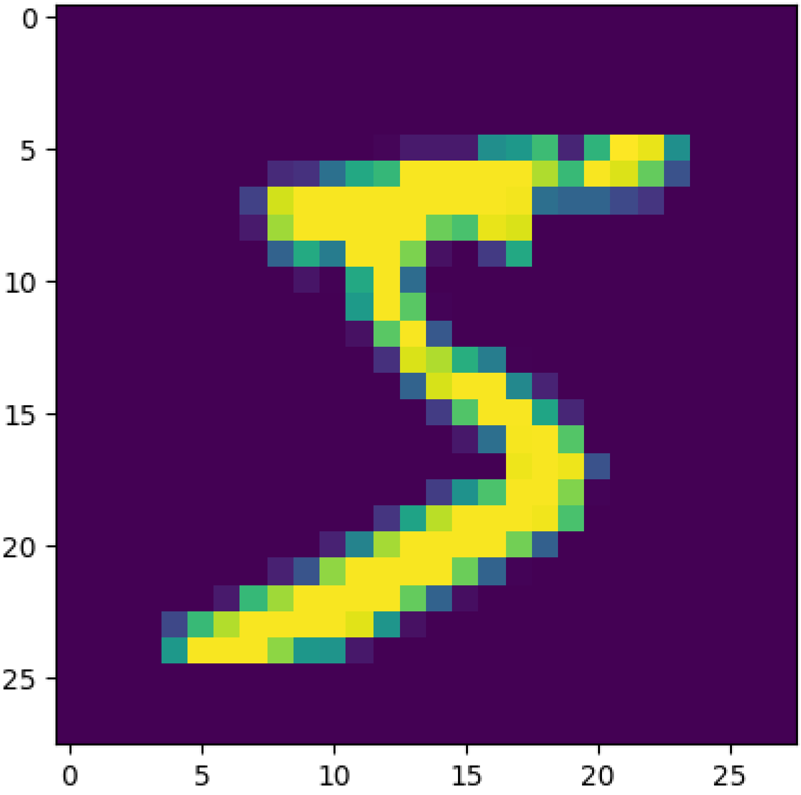
In [7]:
```
y_train
```

Out[7]:
```
array([5, 0, 4, ..., 5, 6, 8], dtype=uint8)
```

In [8]:
```
import matplotlib.pyplot as plt
```

In [9]:
```
plt.imshow(X_train[0])
```

Out[9]:
```
<matplotlib.image.AxesImage at 0x2531a483450>
```



In [10]:
```
X_train = X_train/255
X_test = X_test/255
```

In [138…
```
model = Sequential()
model.add(Flatten(input_shape=(28,28)))
model.add(Dense(128, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense (10, activation='softmax'))
```

In [12]:
```
model.summary()
```

```
Model: "sequential"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━
┃ Layer (type)                           ┃ Output Shape                 ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━
│ flatten (Flatten)                      │ (None, 784)                  │
├────────────────────────────────────────┼──────────────────────────────┼─────
│ dense (Dense)                          │ (None, 128)                  │
├────────────────────────────────────────┼──────────────────────────────┼─────
│ dense_1 (Dense)                        │ (None, 32)                   │
├────────────────────────────────────────┼──────────────────────────────┼─────
│ dense_2 (Dense)                        │ (None, 10)                   │
└────────────────────────────────────────┴──────────────────────────────┴─────

 Total params: 104,938 (409.91 KB)

 Trainable params: 104,938 (409.91 KB)

 Non-trainable params: 0 (0.00 B)
```

In [13]:
```
model.compile(loss='sparse_categorical_crossentropy', optimizer='Adam', metrics=
history = model.fit(X_train,y_train, epochs=25, validation_split=0.2)  #train te
```

```
Epoch 1/25
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - accuracy: 0.8476 - loss: 0.5139 - va
l_accuracy: 0.9610 - val_loss: 0.1377
Epoch 2/25
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - accuracy: 0.9635 - loss: 0.1259 - va
l_accuracy: 0.9651 - val_loss: 0.1148
Epoch 3/25
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - accuracy: 0.9740 - loss: 0.0867 - va
l_accuracy: 0.9678 - val_loss: 0.1057
Epoch 4/25
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.9817 - loss: 0.0592 - va
l_accuracy: 0.9725 - val_loss: 0.0938
Epoch 5/25
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.9848 - loss: 0.0467 - va
l_accuracy: 0.9723 - val_loss: 0.0935
Epoch 6/25
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 10s 5ms/step - accuracy: 0.9894 - loss: 0.0354 - v
al_accuracy: 0.9739 - val_loss: 0.0894
Epoch 7/25
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.9910 - loss: 0.0284 - va
l_accuracy: 0.9700 - val_loss: 0.1090
Epoch 8/25
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.9915 - loss: 0.0251 - va
l_accuracy: 0.9747 - val_loss: 0.0972
Epoch 9/25
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.9936 - loss: 0.0204 - va
l_accuracy: 0.9767 - val_loss: 0.0979
Epoch 10/25
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 10s 4ms/step - accuracy: 0.9940 - loss: 0.0175 - v
al_accuracy: 0.9714 - val_loss: 0.1240
Epoch 11/25
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.9944 - loss: 0.0170 - va
l_accuracy: 0.9764 - val_loss: 0.1047
Epoch 12/25
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.9954 - loss: 0.0141 - va
l_accuracy: 0.9721 - val_loss: 0.1240
Epoch 13/25
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.9959 - loss: 0.0122 - va
l_accuracy: 0.9711 - val_loss: 0.1425
Epoch 14/25
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.9959 - loss: 0.0146 - va
l_accuracy: 0.9701 - val_loss: 0.1377
Epoch 15/25
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - accuracy: 0.9968 - loss: 0.0097 - va
l_accuracy: 0.9722 - val_loss: 0.1397
Epoch 16/25
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.9955 - loss: 0.0126 - va
l_accuracy: 0.9757 - val_loss: 0.1260
Epoch 17/25
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.9980 - loss: 0.0066 - va
l_accuracy: 0.9722 - val_loss: 0.1528
Epoch 18/25
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 8s 5ms/step - accuracy: 0.9951 - loss: 0.0149 - va
l_accuracy: 0.9744 - val_loss: 0.1416
Epoch 19/25
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.9968 - loss: 0.0091 - va
l_accuracy: 0.9758 - val_loss: 0.1345
Epoch 20/25
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.9986 - loss: 0.0045 - va
l_accuracy: 0.9732 - val_loss: 0.1579
```

```
Epoch 21/25
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.9969 - loss: 0.0092 - va
l_accuracy: 0.9764 - val_loss: 0.1541
Epoch 22/25
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 8s 5ms/step - accuracy: 0.9973 - loss: 0.0078 - va
l_accuracy: 0.9756 - val_loss: 0.1584
Epoch 23/25
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.9981 - loss: 0.0064 - va
l_accuracy: 0.9768 - val_loss: 0.1439
Epoch 24/25
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.9983 - loss: 0.0053 - va
l_accuracy: 0.9770 - val_loss: 0.1555
Epoch 25/25
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.9978 - loss: 0.0073 - va
l_accuracy: 0.9761 - val_loss: 0.1673
```

In [14]:
```
y_prob= model.predict(X_test)
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
```

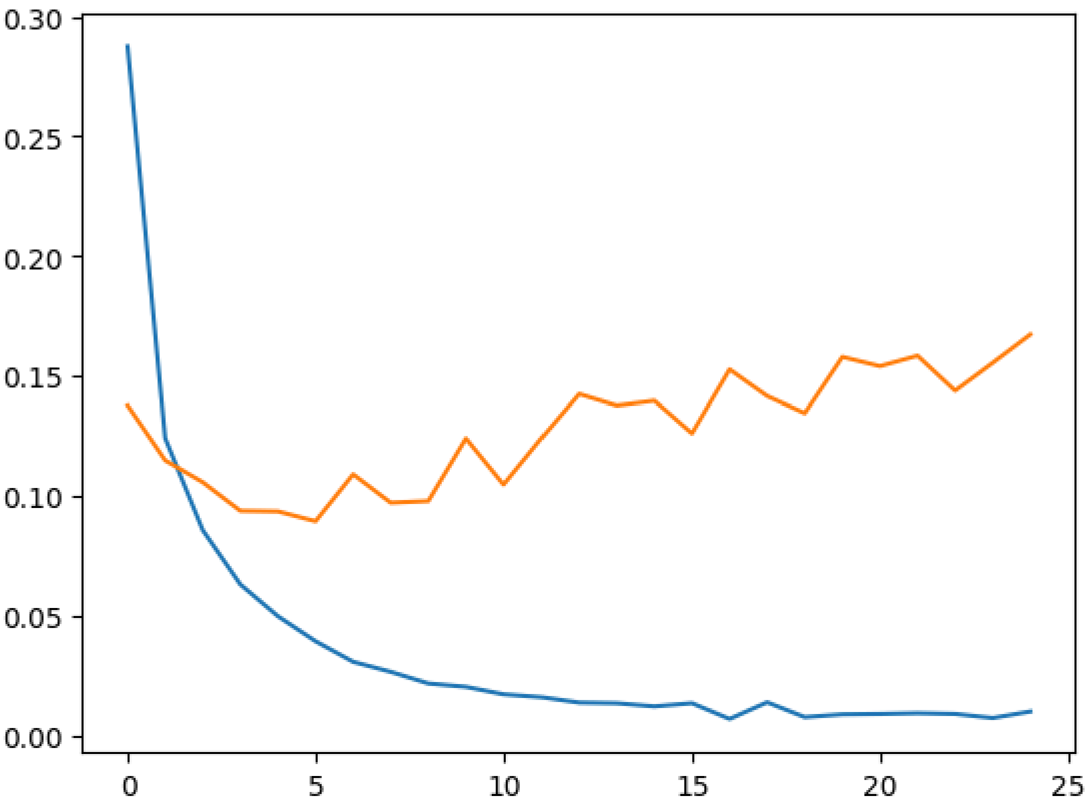
In [15]:
```
y_pred= y_prob.argmax(axis=1)
```

In [16]:
```
from sklearn.metrics import accuracy_score
accuracy_score(y_test,y_pred)
```

Out[16]:
```
0.9762
```

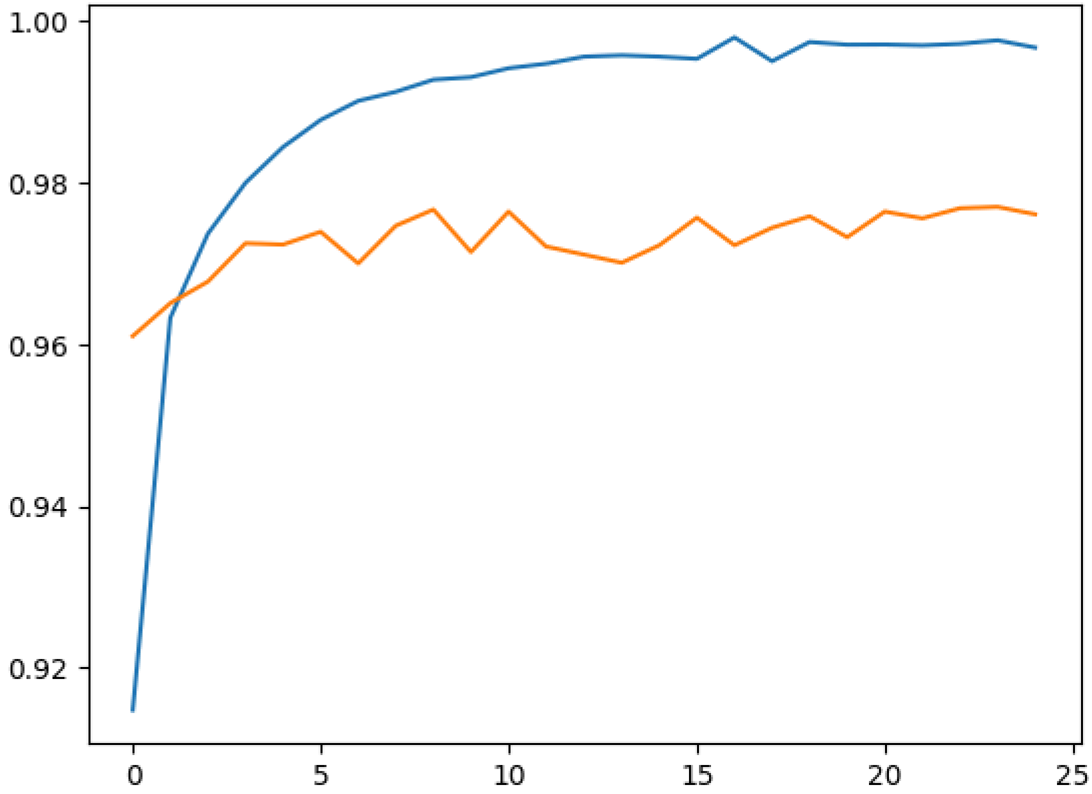
In [17]:
```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
```

Out[17]:
```
[<matplotlib.lines.Line2D at 0x2533864f410>]
```



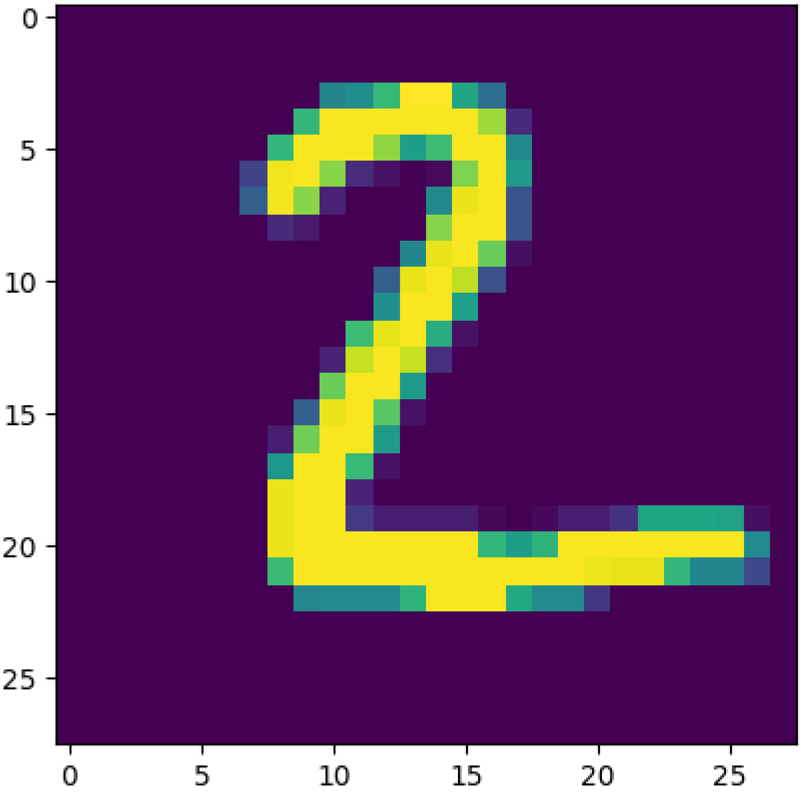
In [18]:
```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
```

```
Out[18]: [<matplotlib.lines.Line2D at 0x253387229d0>]
```



```
In [19]: plt.imshow(X_test[1])
```

```
Out[19]: <matplotlib.image.AxesImage at 0x2533873dd90>
```



```
In [20]: model.predict(X_test[1].reshape(1,28,28)).argmax(axis=1)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step
```

```
Out[20]:  array([2], dtype=int64)
```

on fashion_mnist dataset

```
In [21]:  (X_train,y_train),(X_test,y_test) = keras.datasets.fashion_mnist.load_data()
```

```
In [22]:  X_test.shape
```

```
Out[22]:  (10000, 28, 28)
```

```
In [23]:  X_train.shape
```

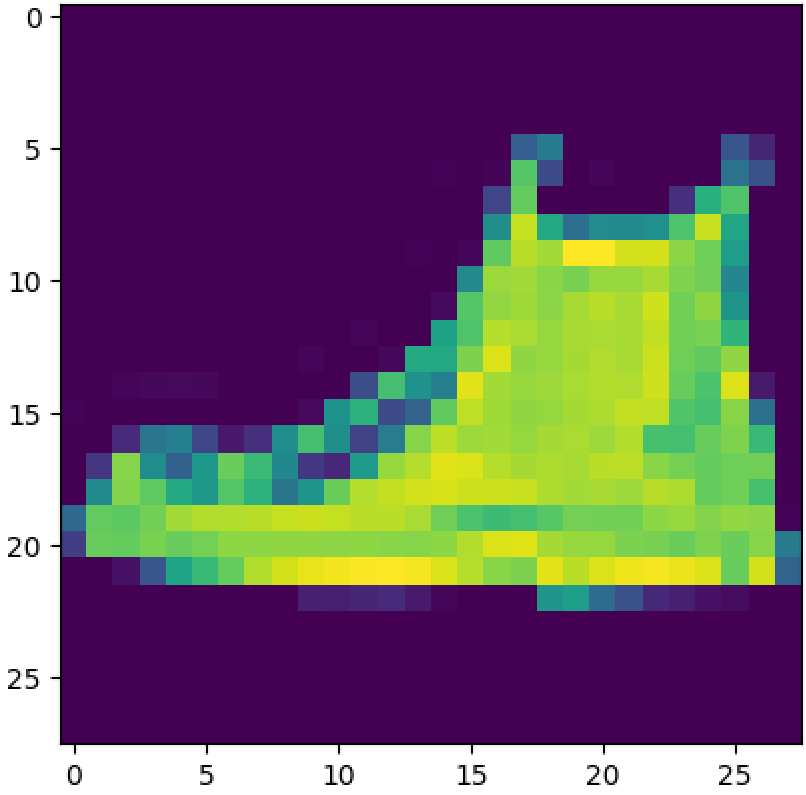
```
Out[23]:  (60000, 28, 28)
```

```
In [24]:  y_train
```

```
Out[24]:  array([9, 0, 0, ..., 3, 0, 5], dtype=uint8)
```

```
In [156…  plt.imshow(X_train[450])
```

```
Out[156…  <matplotlib.image.AxesImage at 0x25344a15d90>
```



X_train = X_train/255

X_test = X_test/255

In [137…

```
model = Sequential()
model.add(Flatten(input_shape=(28,28)))
model.add(Dense(128, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense (10, activation='softmax'))
```

In [28]:

```
model.summary()
```

```
Model: "sequential_1"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┓
┃ Layer (type)                           ┃ Output Shape                 ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┩
│ flatten_1 (Flatten)                    │ (None, 784)                  │
├────────────────────────────────────────┼──────────────────────────────┤
│ dense_3 (Dense)                        │ (None, 128)                  │
├────────────────────────────────────────┼──────────────────────────────┤
│ dense_4 (Dense)                        │ (None, 32)                   │
├────────────────────────────────────────┼──────────────────────────────┤
│ dense_5 (Dense)                        │ (None, 10)                   │
└────────────────────────────────────────┴──────────────────────────────┘

 Total params: 104,938 (409.91 KB)

 Trainable params: 104,938 (409.91 KB)

 Non-trainable params: 0 (0.00 B)
```

In [224…

```
model.compile(loss='sparse_categorical_crossentropy', optimizer='Adam', metrics=
history = model.fit(X_train,y_train, epochs=40, validation_split=0.2)  #train te
```

```
Epoch 1/40
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 10s 5ms/step - accuracy: 0.7643 - loss: 0.6755 - v
al_accuracy: 0.8438 - val_loss: 0.4228
Epoch 2/40
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.8584 - loss: 0.3920 - va
l_accuracy: 0.8572 - val_loss: 0.4002
Epoch 3/40
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.8708 - loss: 0.3512 - va
l_accuracy: 0.8701 - val_loss: 0.3642
Epoch 4/40
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.8804 - loss: 0.3206 - va
l_accuracy: 0.8742 - val_loss: 0.3571
Epoch 5/40
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.8877 - loss: 0.3044 - va
l_accuracy: 0.8798 - val_loss: 0.3382
Epoch 6/40
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 12s 8ms/step - accuracy: 0.8928 - loss: 0.2908 - v
al_accuracy: 0.8770 - val_loss: 0.3548
Epoch 7/40
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 19s 13ms/step - accuracy: 0.8982 - loss: 0.2763 -
val_accuracy: 0.8802 - val_loss: 0.3328
Epoch 8/40
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.9000 - loss: 0.2715 - va
l_accuracy: 0.8769 - val_loss: 0.3432
Epoch 9/40
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.9065 - loss: 0.2528 - va
l_accuracy: 0.8847 - val_loss: 0.3297
Epoch 10/40
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 9s 6ms/step - accuracy: 0.9061 - loss: 0.2524 - va
l_accuracy: 0.8899 - val_loss: 0.3132
Epoch 11/40
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.9144 - loss: 0.2336 - va
l_accuracy: 0.8835 - val_loss: 0.3342
Epoch 12/40
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.9134 - loss: 0.2292 - va
l_accuracy: 0.8813 - val_loss: 0.3442
Epoch 13/40
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.9165 - loss: 0.2234 - va
l_accuracy: 0.8878 - val_loss: 0.3265
Epoch 14/40
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.9209 - loss: 0.2101 - va
l_accuracy: 0.8840 - val_loss: 0.3553
Epoch 15/40
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.9234 - loss: 0.2042 - va
l_accuracy: 0.8789 - val_loss: 0.3529
Epoch 16/40
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.9235 - loss: 0.2055 - va
l_accuracy: 0.8914 - val_loss: 0.3353
Epoch 17/40
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.9259 - loss: 0.1968 - va
l_accuracy: 0.8877 - val_loss: 0.3478
Epoch 18/40
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.9240 - loss: 0.1995 - va
l_accuracy: 0.8867 - val_loss: 0.3488
Epoch 19/40
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.9323 - loss: 0.1804 - va
l_accuracy: 0.8928 - val_loss: 0.3320
Epoch 20/40
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.9351 - loss: 0.1765 - va
l_accuracy: 0.8896 - val_loss: 0.3420
```

```
Epoch 21/40
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.9328 - loss: 0.1793 - va
l_accuracy: 0.8917 - val_loss: 0.3503
Epoch 22/40
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.9376 - loss: 0.1691 - va
l_accuracy: 0.8908 - val_loss: 0.3569
Epoch 23/40
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.9393 - loss: 0.1632 - va
l_accuracy: 0.8878 - val_loss: 0.3542
Epoch 24/40
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.9417 - loss: 0.1564 - va
l_accuracy: 0.8851 - val_loss: 0.3732
Epoch 25/40
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.9382 - loss: 0.1610 - va
l_accuracy: 0.8882 - val_loss: 0.3876
Epoch 26/40
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.9428 - loss: 0.1511 - va
l_accuracy: 0.8871 - val_loss: 0.3721
Epoch 27/40
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.9440 - loss: 0.1472 - va
l_accuracy: 0.8880 - val_loss: 0.3808
Epoch 28/40
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.9451 - loss: 0.1469 - va
l_accuracy: 0.8828 - val_loss: 0.4405
Epoch 29/40
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.9459 - loss: 0.1429 - va
l_accuracy: 0.8913 - val_loss: 0.3735
Epoch 30/40
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.9481 - loss: 0.1384 - va
l_accuracy: 0.8841 - val_loss: 0.4040
Epoch 31/40
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.9479 - loss: 0.1377 - va
l_accuracy: 0.8860 - val_loss: 0.4036
Epoch 32/40
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.9489 - loss: 0.1375 - va
l_accuracy: 0.8941 - val_loss: 0.3859
Epoch 33/40
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.9501 - loss: 0.1308 - va
l_accuracy: 0.8836 - val_loss: 0.4580
Epoch 34/40
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.9506 - loss: 0.1333 - va
l_accuracy: 0.8916 - val_loss: 0.4048
Epoch 35/40
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.9515 - loss: 0.1268 - va
l_accuracy: 0.8847 - val_loss: 0.4348
Epoch 36/40
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.9531 - loss: 0.1231 - va
l_accuracy: 0.8935 - val_loss: 0.4283
Epoch 37/40
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.9562 - loss: 0.1151 - va
l_accuracy: 0.8896 - val_loss: 0.4330
Epoch 38/40
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.9551 - loss: 0.1216 - va
l_accuracy: 0.8888 - val_loss: 0.4439
Epoch 39/40
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.9578 - loss: 0.1106 - va
l_accuracy: 0.8888 - val_loss: 0.4647
Epoch 40/40
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.9557 - loss: 0.1181 - va
l_accuracy: 0.8918 - val_loss: 0.4173
```

In [225…

```
y_prob= model.predict(X_test)
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
```

In [226…

```
y_pred= y_prob.argmax(axis=1)
```

In [227…

```
from sklearn.metrics import accuracy_score
accuracy_score(y_test,y_pred)
```

Out[227…

```
0.8871
```

In [228…

```
loss, accuracy = model.evaluate(X_test, y_test)
print("Test Accuracy:", accuracy)
print("Test loss:", loss)
```

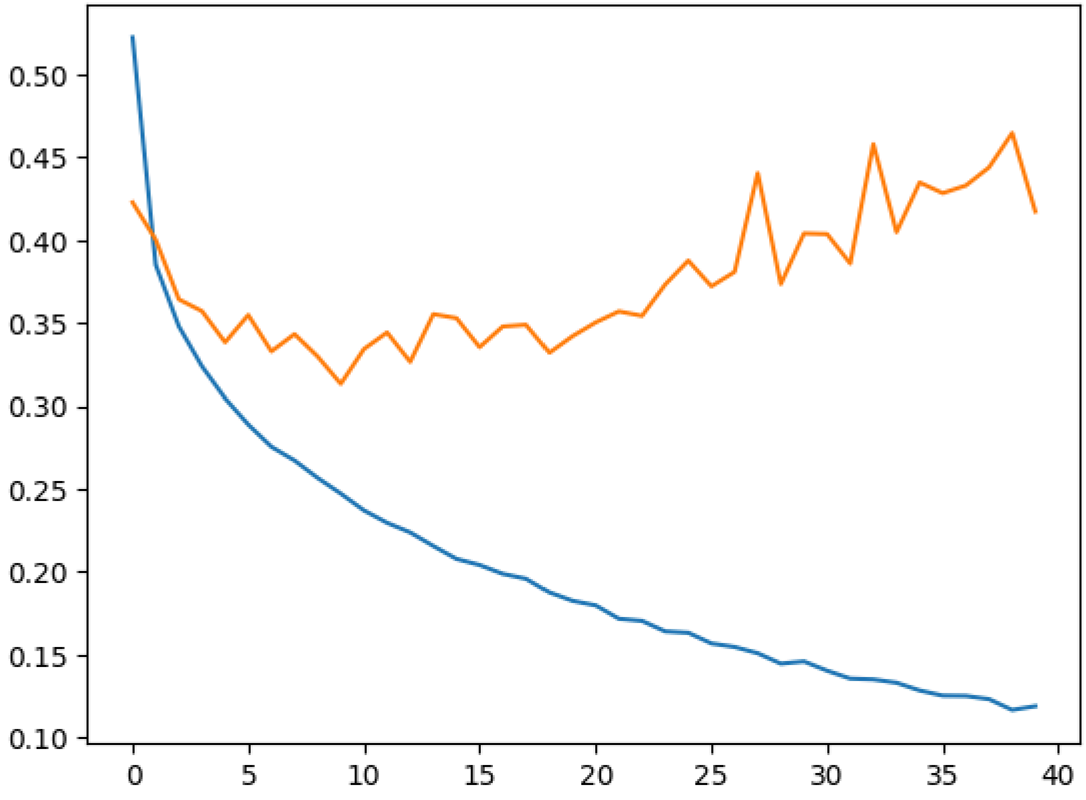
```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8850 - loss: 0.4613
Test Accuracy: 0.8870999813079834
Test loss: 0.46158188581466675
```

In [229…

```
plt.plot(history.history['loss'])    #blue
plt.plot(history.history['val_loss'])   #orange
```

Out[229…

```
[<matplotlib.lines.Line2D at 0x2534ce8f750>]
```

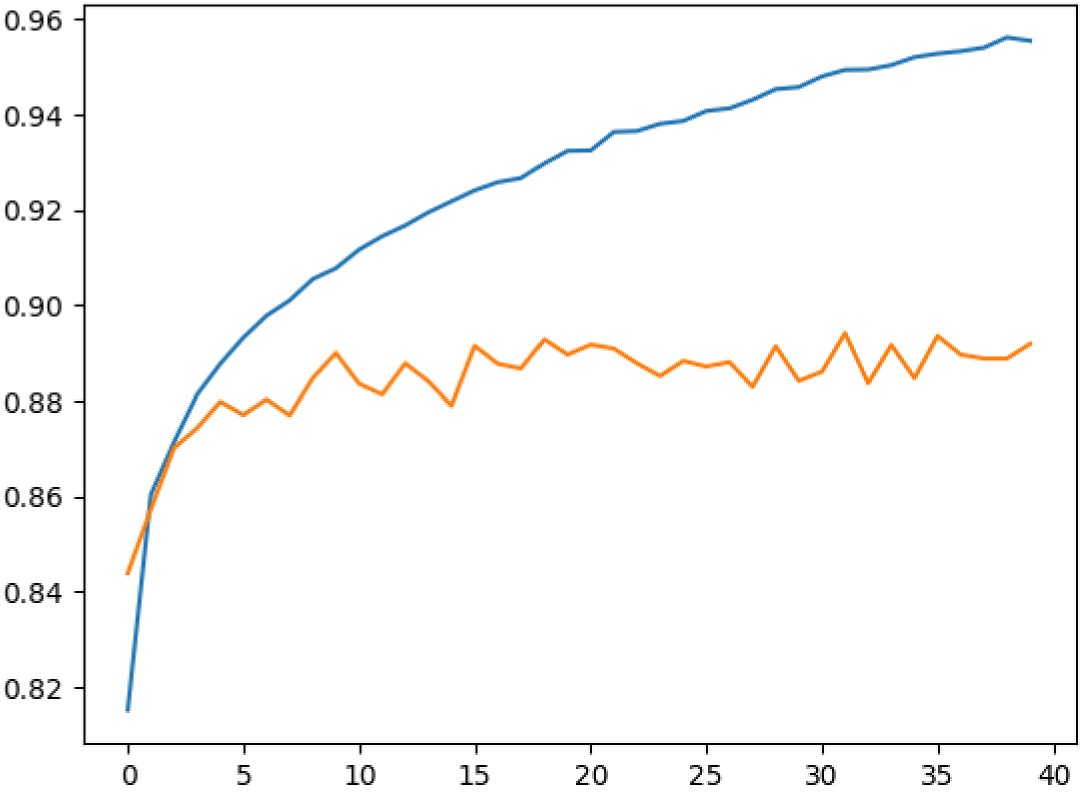


In [230…

```
plt.plot(history.history['accuracy'])  #blue
plt.plot(history.history['val_accuracy'])   #orange
```

Out[230…

```
[<matplotlib.lines.Line2D at 0x2534cf4b1d0>]
```

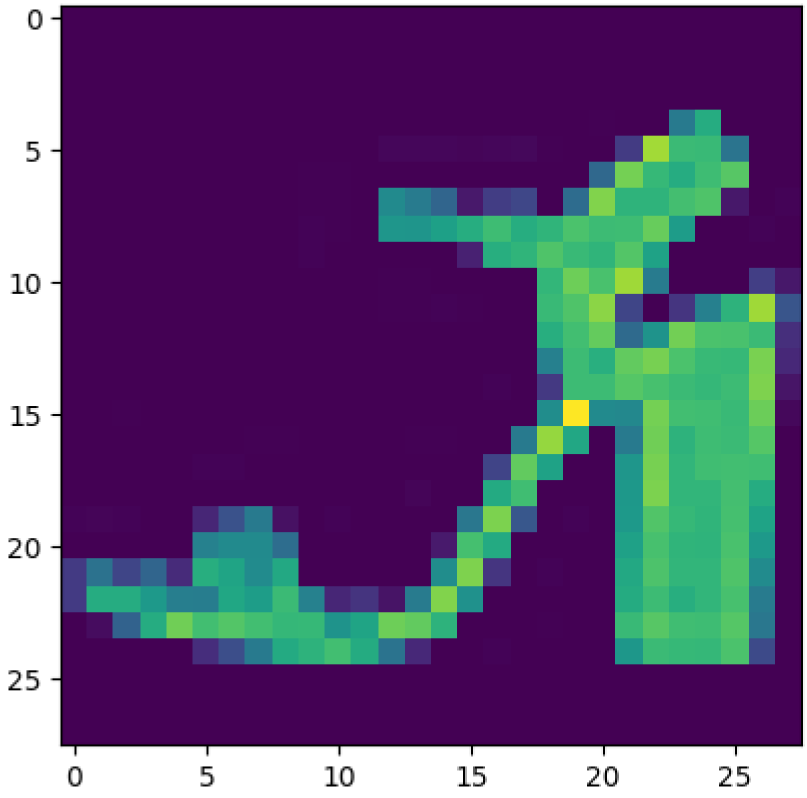


In [231…

```
n=8170
plt.imshow(X_test[n])
```

Out[231…

```
<matplotlib.image.AxesImage at 0x2534cf9dd90>
```



model.predict(X_test[n].reshape(1,28,28)).argmax(axis=1)

In [232…

```
class_labels = {
    0: 'T-shirt/top',
    1: 'Trouser',
    2: 'Pullover',
    3: 'Dress',
    4: 'Coat',
    5: 'Sandal',
    6: 'Shirt',
    7: 'Sneaker',
    8: 'Bag',
    9: 'Ankle boot'
}
output_label= model.predict(X_test[n].reshape(1, 28, 28))[0].argmax()
print(class_labels[output_label])
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
Sandal
```

In [233…

```
model.save("my_model.keras")
```

GUI Model for image Classification

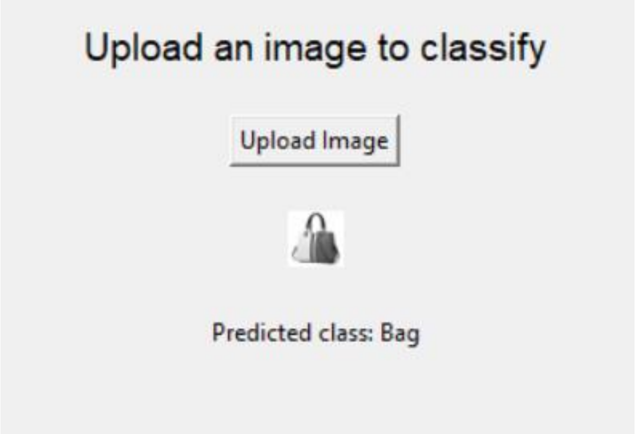
In [234…

```
import tkinter as tk
from tkinter import filedialog, messagebox
from PIL import Image, ImageTk
import numpy as np
from keras.models import load_model

model = load_model("my_model.keras")

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

def classify_image():
    file_path = filedialog.askopenfilename()
    if file_path:

        image = Image.open(file_path).resize((28, 28)).convert('L')
        image_array = np.array(image) / 255.0
        image_array = image_array.reshape(1, 28, 28, 1)

        # uploaded image
        resized_image = image.resize((50, 50))
        image_display = ImageTk.PhotoImage(resized_image)
        image_label.config(image=image_display)
        image_label.image = image_display

        predicted_class = model.predict(image_array)[0].argmax()

        result_label.config(text=f"Predicted class: {class_labels[predicted_clas
root = tk.Tk()
root.title("Image Classifier")

window_width = 800     #centering content
window_height = 700
screen_width = root.winfo_screenwidth()
screen_height = root.winfo_screenheight()
x_coordinate = (screen_width / 2) - (window_width / 2)
y_coordinate = (screen_height / 2) - (window_height / 2)
root.geometry(f"{window_width}x{window_height}+{int(x_coordinate)}+{int(y_coordi

explanation_label = tk.Label(root, text="Upload an image to classify", font=("He
explanation_label.pack(pady=10)

upload_button = tk.Button(root, text="Upload Image", command=classify_image)
upload_button.pack(pady=10)

image_label = tk.Label(root)
image_label.pack(pady=10)

result_label = tk.Label(root, text="")
result_label.pack(pady=10)

root.mainloop()
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 104ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 190ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
```

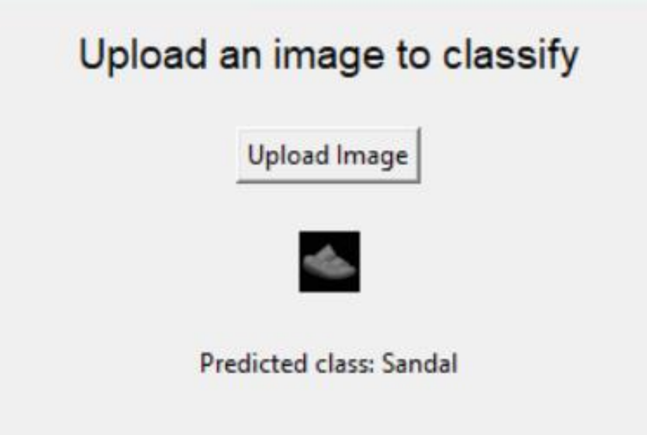
In [ ]:

Gui Model Output



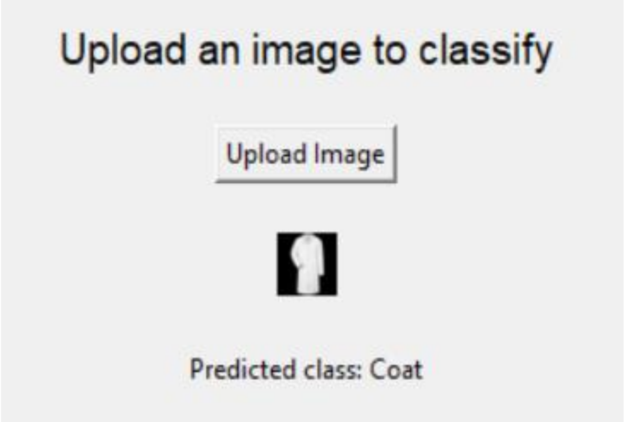
Upload an image to classify

Predicted class: Bag



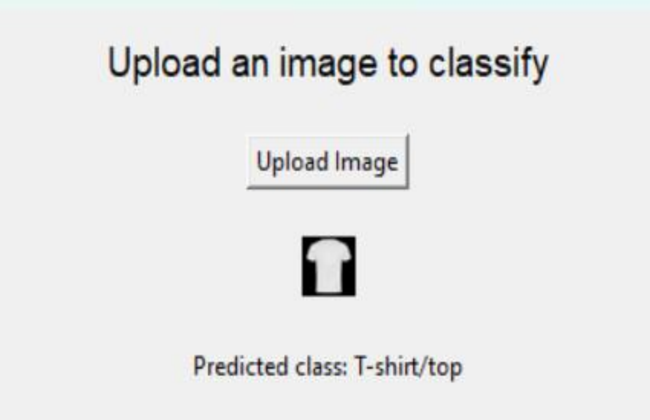
Upload an image to classify

Predicted class: Sandal



Upload an image to classify

Predicted class: Coat



Upload an image to classify

Predicted class: T-shirt/top