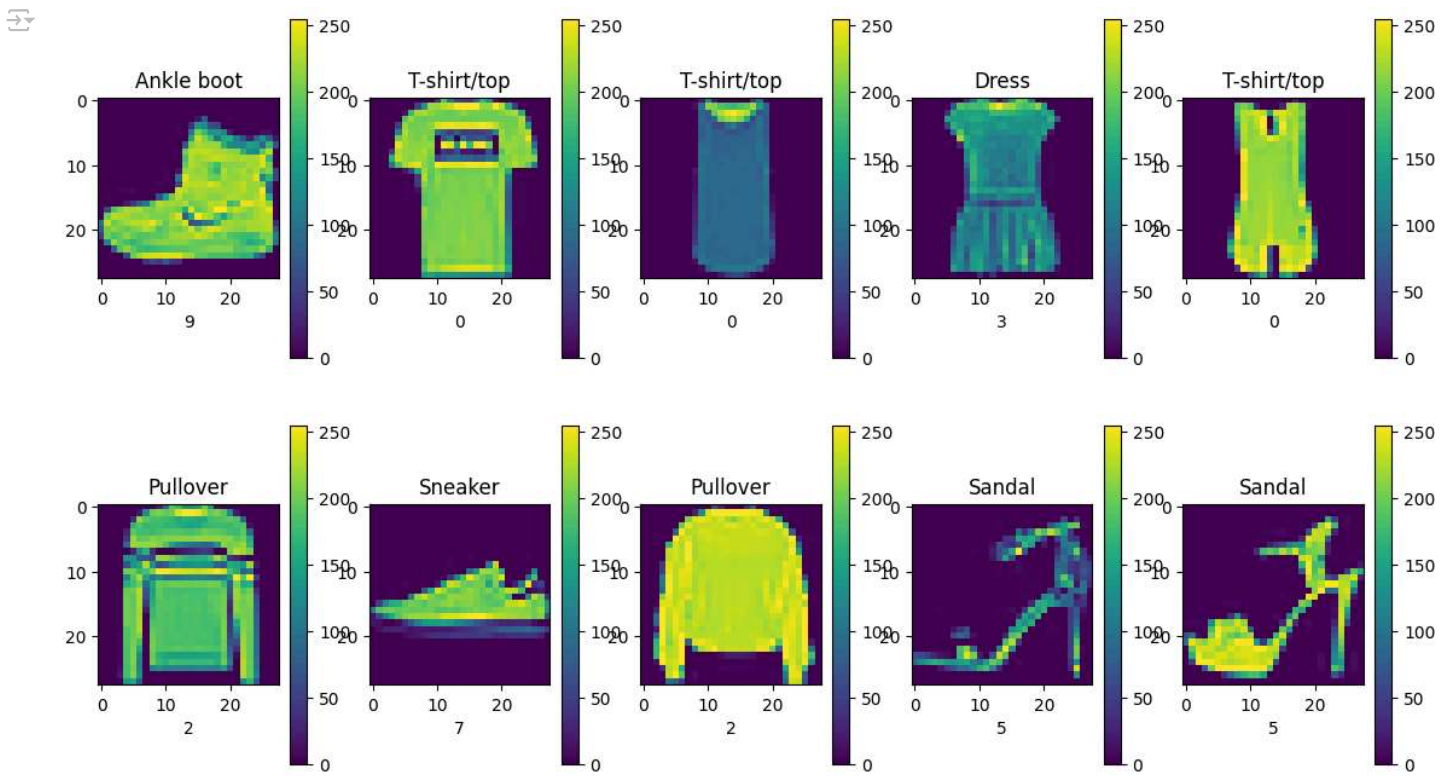
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt


from tensorflow.keras.datasets import fashion_mnist
load_data = fashion_mnist.load_data()

(X_train,y_train),(X_test,y_test)=load_data



X_train.shape,X_test.shape,y_train.shape,y_test.shape
```

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

```
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
plt.figure(figsize=(14,8))
for i in range(10):
  plt.subplot(2,5,i+1)
  plt.imshow(X_train[i])
  plt.xlabel(y_train[i])
  plt.title(class_names[y_train[i]])
  plt.colorbar()
plt.show()
```




```
#normalization
X_train=X_train/255.0
X_test=X_test/255.0
```

part 1 : we decide layer and numbers of neron in DL model

```
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense,Flatten

model=Sequential()
```

```
#input layer
model.add(Flatten(input_shape=(28,28)))

#hidden layer
model.add(Dense(128,activation='relu'))
model.add(Dense(64,activation='relu'))

#output layer
model.add(Dense(10,activation='softmax'))

model.summary()
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/reshaping/flatten.py:37: UserWarning: Do not pass an `input_shape`/`input_dim`
  super().__init__(**kwargs)
Model: "sequential"
```

Layer (type)	Output Shape	Param #
flatten (Flatten)	(None, 784)	0
dense (Dense)	(None, 128)	100,480
dense_1 (Dense)	(None, 64)	8,256
dense_2 (Dense)	(None, 10)	650

```
Total params: 109,386 (427.29 KB)
Trainable params: 109,386 (427.29 KB)
```

part 2: we hyper tuing for the model

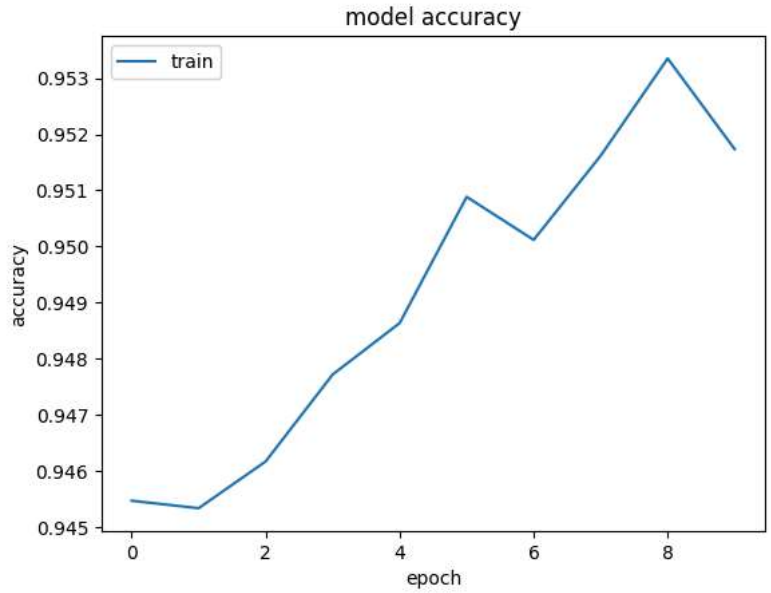
```
from re import VERBOSE

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

model.fit(X_train,
          y_train,
          epochs=10,
          batch_size=64,
          verbose=True)
```

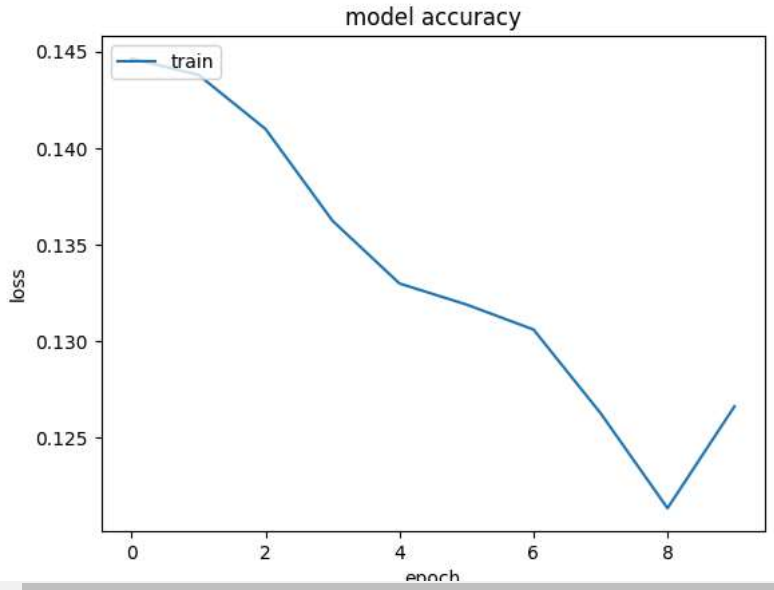
```
Epoch 1/10
938/938 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.7650 - loss: 0.6781
Epoch 2/10
938/938 ━━━━━━━━━━━━━━━━━━━━ 6s 6ms/step - accuracy: 0.8598 - loss: 0.3836
Epoch 3/10
938/938 ━━━━━━━━━━━━━━━━━━━━ 7s 3ms/step - accuracy: 0.8753 - loss: 0.3412
Epoch 4/10
938/938 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.8832 - loss: 0.3182
Epoch 5/10
938/938 ━━━━━━━━━━━━━━━━━━━━ 5s 5ms/step - accuracy: 0.8899 - loss: 0.2972
Epoch 6/10
938/938 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.8945 - loss: 0.2821
Epoch 7/10
938/938 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9003 - loss: 0.2683
Epoch 8/10
938/938 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - accuracy: 0.9024 - loss: 0.2614
Epoch 9/10
938/938 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.9039 - loss: 0.2526
Epoch 10/10
938/938 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.9121 - loss: 0.2368
<keras.src.callbacks.history.History at 0x7fc1667bf7c0>
```

```
history=model.fit(X_train,
          y_train,
          epochs=10)

history.history

plt.plot(history.history['accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train'],loc='upper left')
plt.show()
```

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

```
Epoch 1/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 8s 4ms/step - accuracy: 0.9461 - loss: 0.1428
Epoch 2/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 11s 4ms/step - accuracy: 0.9479 - loss: 0.1375
Epoch 3/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.9454 - loss: 0.1415
Epoch 4/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 13s 4ms/step - accuracy: 0.9488 - loss: 0.1344
Epoch 5/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 10s 4ms/step - accuracy: 0.9505 - loss: 0.1296
Epoch 6/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.9514 - loss: 0.1320
Epoch 7/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.9518 - loss: 0.1279
Epoch 8/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 3ms/step - accuracy: 0.9534 - loss: 0.1215
Epoch 9/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 8s 4ms/step - accuracy: 0.9543 - loss: 0.1196
Epoch 10/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 4ms/step - accuracy: 0.9538 - loss: 0.1198
```





```
test_loss,test_accuracy=model.evaluate(X_test,y_test)
print('test accuracy',test_accuracy)
print('test loss',test_loss)
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8931 - loss: 0.4447
test accuracy 0.8901000022888184
test loss 0.4628857374191284
```

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

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
array([[3.9055110e-11, 1.9484430e-18, 1.1507793e-12, ..., 7.5328321e-04,
        7.0992593e-13, 9.9924672e-01],
       [9.8721877e-08, 1.6307365e-27, 9.9997455e-01, ..., 5.3654022e-22,
        1.3944081e-17, 1.0977424e-22],
       [4.9173477e-23, 9.9999994e-01, 2.8237448e-28, ..., 0.0000000e+00,
        7.2079977e-26, 4.0286157e-34],
       ...,
       [1.6209696e-15, 9.6091236e-25, 3.3595519e-14, ..., 6.7997955e-12,
        9.9999994e-01, 1.7985019e-26],
       [2.7105251e-23, 9.9999994e-01, 1.5391924e-25, ..., 2.0408240e-27,
        5.4698360e-16, 1.8073486e-24],
       [8.0135220e-16, 1.1290169e-19, 1.9361081e-13, ..., 1.2065796e-08,
        1.7359673e-12, 1.0119332e-17]], dtype=float32)
```

```
amx=np.max(y_predict[0])
amax=np.argmax(y_predict[0])
print(amx,amax,sep=",")
```

```
0.9992467,9
```

```
max_prob=np.max(y_predict,axis=1)
index=np.argmax(y_predict,axis=1)
predict_class=class_names[index[0]]
Ground_truth_class=class_names[y_test[0]]
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-33-3030ca3a48d5> in <cell line: 3>()
      1 max_prob=np.max(y_predict,axis=1)
      2 index=np.argmax(y_predict,axis=1)
----> 3 predict_class=class_names[index]
      4 Ground_truth_class=class_names[y_test[0]]

TypeError: only integer scalar arrays can be converted to a scalar index
```

Start coding or generate with AI.