```
#VAISHNAVI SOLANKAR
#MY PROGRAM
import numpy as np
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense
```

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

```
x_train.shape
```

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

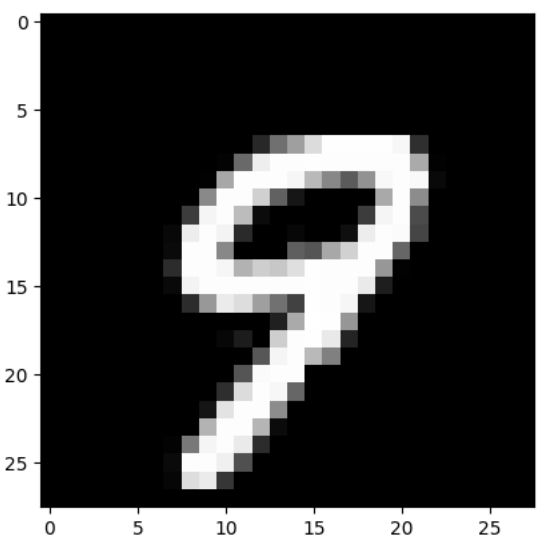
```
x_test.shape
```

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

```
x_train[200]
```

```
ndarray (28, 28) show data
```

```
import matplotlib.pyplot as plt
plt.imshow(x_train[2453],cmap='gray');
```




```
y_train[2453]
```

```
9
```

```
img = np.array([[0.78,0.34],[0.20,0.80]])
```

```
img
```

```
array([[0.78, 0.34],
       [0.2 , 0.8 ]])
```

```
img.flatten()
```

```
array([0.78, 0.34, 0.2 , 0.8 ])
```

```
Start coding or generate with AI.
```

```
x_train[200].shape
```

```
(28, 28)
```

```
x_train[200].flatten()
```

```
array([  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
```

```
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,  29, 197, 255,  84,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,  85, 251, 253,  83,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  86, 253, 254,
253,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  85,
251, 253, 251,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,  86, 253, 254, 253, 169,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,  85, 251, 253, 251, 168,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,  86, 253, 254, 253, 169,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,  28, 196, 253, 251, 168,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0, 169, 254, 253,
169,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0, 168,
253, 251, 168,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0, 169, 254, 253, 169,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0, 168, 253, 251, 168,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0, 254, 253, 254, 139,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0, 253, 251, 253, 251,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0, 254, 253, 254,
253,  57,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0, 253,
251, 253, 251, 168,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0, 198, 253, 254, 253, 114,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,  85, 251, 253, 251,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,  85, 253, 254, 253,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,  28,  83, 196,  83,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
```

```
x_train[200].flatten().shape
```

```
(784,)
```

```
#reshape
arr=np.array([[1,2,3],[4,5,6],[7,8,9],[10,11,12]])
arr
```

```
array([[ 1,  2,  3],
       [ 4,  5,  6],
       [ 7,  8,  9],
       [10, 11, 12]])
```

```
arr.shape
```

```
(4, 3)
```

```
arr.reshape(2,6)
```

```
array([[ 1,  2,  3,  4,  5,  6],
       [ 7,  8,  9, 10, 11, 12]])
```

```
x_train=x_train.reshape(60000,784)
x_test=x_test.reshape(10000,784)
```

```
x_train.shape
```

```
(60000, 784)
```

```
x_train[200]
```

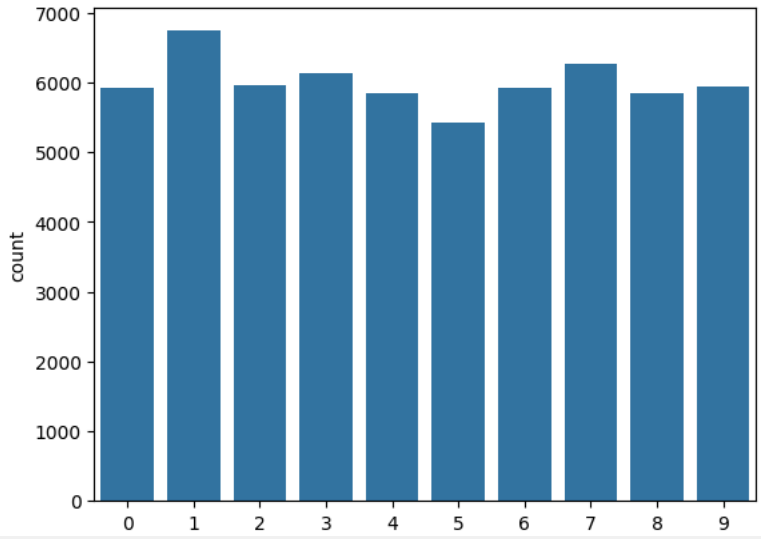
```
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,  29, 197, 255,  84,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,  85, 251, 253,  83,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  86, 253, 254,
       253,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  85,
       251, 253, 251,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,  86, 253, 254, 253, 169,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,  85, 251, 253, 251, 168,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,  86, 253, 254, 253, 169,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,  28, 196, 253, 251, 168,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0, 169, 254, 253,
       169,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0, 168,
       253, 251, 168,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0, 169, 254, 253, 169,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0, 168, 253, 251, 168,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0, 254, 253, 254, 139,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0, 253, 251, 253, 251,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0, 254, 253, 254,
       253,  57,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0, 253,
       251, 253, 251, 168,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0, 198, 253, 254, 253, 114,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,  85, 251, 253, 251,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,  85, 253, 254, 253,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,  28,  83, 196,  83,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0], dtype=uint8)
```

```
x_train=x_train/255
x_test=x_test/255
```

```
set(y_train)
```

```
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
```

```
import seaborn as sns
sns.countplot(x=y_train);
#
```




```
#one hot encoding

from keras.utils import to_categorical
y_train=to_categorical(y_train)
y_test=to_categorical(y_test)


#define the network architecture

from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import SGD


model=Sequential()
model.add(Dense(784,activation='relu',input_shape=(784,)))

model.add(Dense(256,activation='relu'))

model.add(Dense(10,activation='softmax'))
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` arg
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
model.summary()
```

```
Model: "sequential_3"
```

Layer (type)	Output Shape	Param #
dense_9 (Dense)	(None, 784)	615,440
dense_10 (Dense)	(None, 256)	200,960
dense_11 (Dense)	(None, 10)	2,570

```
Total params: 818,970 (3.12 MB)
Trainable params: 818,970 (3.12 MB)
Non-trainable params: 0 (0.00 B)
```

```
model.compile(loss='categorical_crossentropy', metrics=['accuracy'],optimizer=SGD(learning_rate=0.001,momentum=0.7))


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

```
Epoch 1/10
6000/6000 ━━━━━━━━━━━━━━━━━━━━ 56s 9ms/step - accuracy: 0.9782 - loss: 0.0794
Epoch 2/10
6000/6000 ━━━━━━━━━━━━━━━━━━━━ 40s 7ms/step - accuracy: 0.9812 - loss: 0.0702
Epoch 3/10
6000/6000 ━━━━━━━━━━━━━━━━━━━━ 41s 7ms/step - accuracy: 0.9825 - loss: 0.0625
Epoch 4/10
6000/6000 ━━━━━━━━━━━━━━━━━━━━ 84s 7ms/step - accuracy: 0.9838 - loss: 0.0597
Epoch 5/10
6000/6000 ━━━━━━━━━━━━━━━━━━━━ 43s 7ms/step - accuracy: 0.9856 - loss: 0.0546
Epoch 6/10
6000/6000 ━━━━━━━━━━━━━━━━━━━━ 43s 7ms/step - accuracy: 0.9867 - loss: 0.0510
Epoch 7/10
6000/6000 ━━━━━━━━━━━━━━━━━━━━ 41s 7ms/step - accuracy: 0.9880 - loss: 0.0468
Epoch 8/10
```

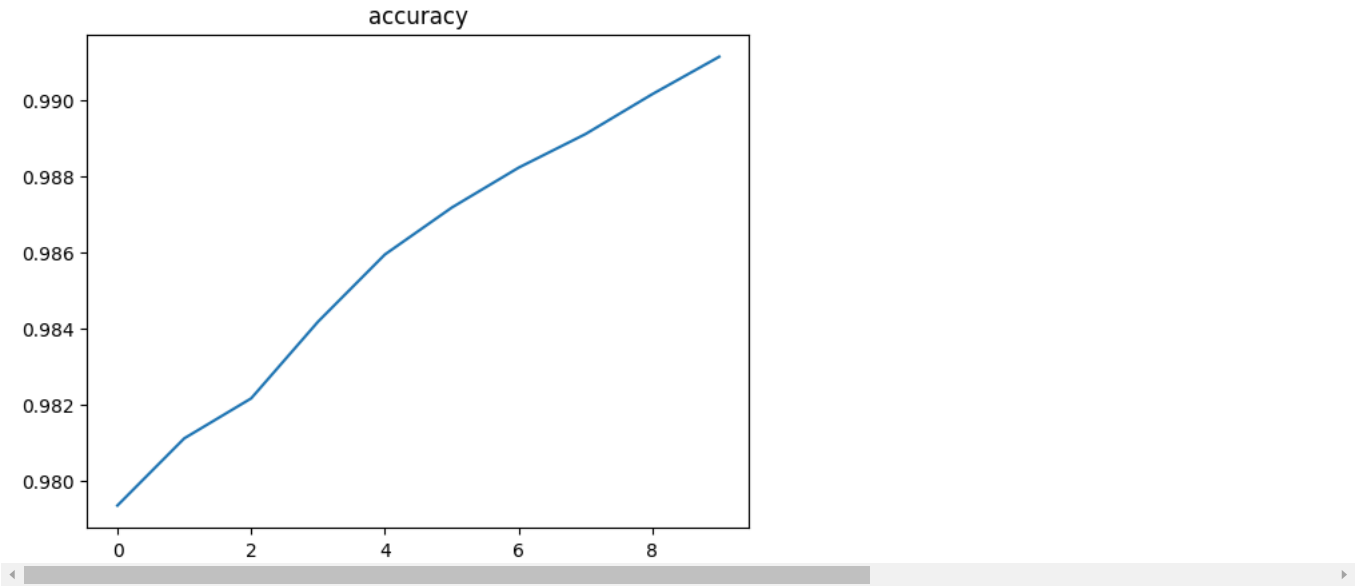
```
6000/6000 ━━━━━━━━━━━━━━━━━━━━ 42s 7ms/step - accuracy: 0.9896 - loss: 0.0418
Epoch 9/10
6000/6000 ━━━━━━━━━━━━━━━━━━━━ 80s 7ms/step - accuracy: 0.9903 - loss: 0.0387
Epoch 10/10
6000/6000 ━━━━━━━━━━━━━━━━━━━━ 41s 7ms/step - accuracy: 0.9910 - loss: 0.0363
```

```
history.history
```

```
{'accuracy': [0.9793499708175659,
  0.9811166524887085,
  0.9821666479110718,
  0.9841833114624023,
  0.9859499931335449,
  0.9871833324432373,
  0.9882333278656006,
  0.9891166687011719,
  0.9901666641235352,
  0.9911500215530396],
 'loss': [0.07642027735710144,
  0.06970127671957016,
  0.0635879784822464,
  0.058508165180683136,
  0.054063573479652405,
  0.0496211051940918,
  0.045694269239902496,
  0.04225872829556465,
  0.03927082568407059,
  0.036357633769512177]}
```

```
plt.title('accuracy')
plt.plot(history.history['accuracy'])
```

```
[<matplotlib.lines.Line2D at 0x7ad9f6079e40>]
```




```
plt.title('loss')
plt.plot(history.history['loss'])
```

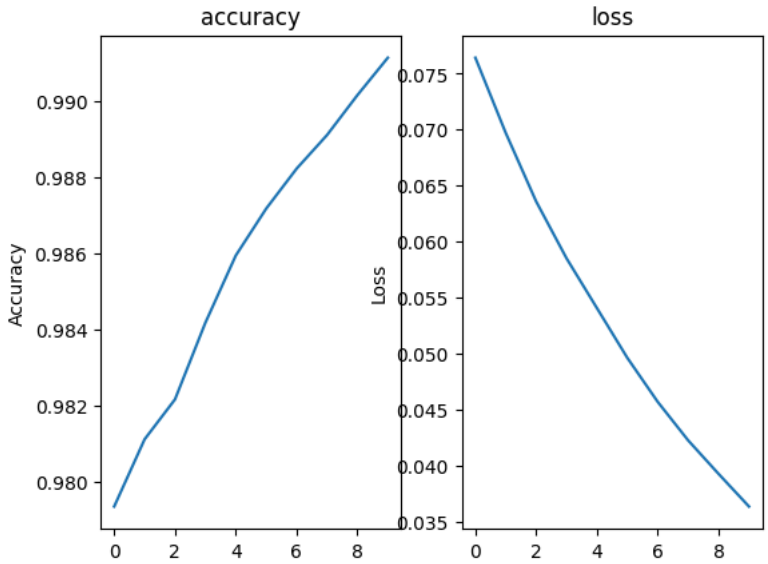
```
[<matplotlib.lines.Line2D at 0x7ad9fd78fac0>]
```



```
plt.subplot(1,2,1)
plt.title('accuracy')
plt.ylabel('Accuracy')
plt.plot(history.history['accuracy'])

plt.subplot(1,2,2)
plt.title('loss')
plt.ylabel('Loss')
plt.plot(history.history['loss'])
```

```
[<matplotlib.lines.Line2D at 0x7ada516b7790>]
```




```
model.evaluate(x_test,y_test,batch_size=1)
```

```
10000/10000 ━━━━━━━━━━━━━━━━━━━━ 20s 2ms/step - accuracy: 0.9756 - loss: 0.0791
[0.06779246777296066, 0.9790999889373779]
```

Start coding or generate with AI.