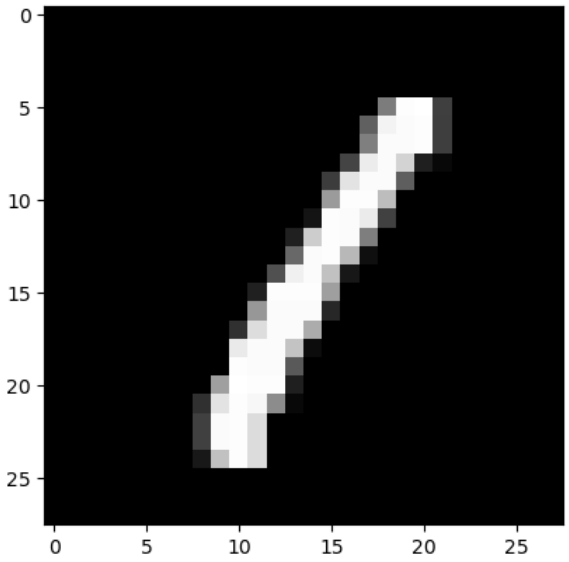
Name:	Tufan Kundu
Registration no:	24MDT0184
Course Name:	Deep Learning Lab
Course Code:	PMDS603P
Experiment:	3
Date:	31 July,2025

Question1. Today, we will try to recall the work done in the previous lab first. The second problem attempted in the last lab was to use MNIST dataset which contains handwritten numbers (their images) from 0 to 9 digits. First try to fit a simple neural network model. Let us import the necessary modules required for this along with the dataset. It contains 70000 handwritten images of digits from 0 to 9. So its a 10 class classification problem. Lets try to create a model that can do the classification task.

In [1]:
```
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense,Dropout,Flatten
from keras.optimizers import SGD
import matplotlib.pyplot as plt

import warnings
warnings.filterwarnings('ignore')
batch_size = 128
num_classes = 10
epochs = 50
(x_train,y_train), (x_test,y_test) = mnist.load_data()
plt.imshow(x_train[3],cmap='gray')
plt.show()
```



In [2]:
```
x_train = x_train.reshape(60000,784)
x_test = x_test.reshape(10000,784)
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train/=255
x_test/=255
print(x_train.shape[0],'train samples')
print(x_test.shape[0],'test samples')
y_train = keras.utils.to_categorical(y_train,num_classes)
y_test_ = keras.utils.to_categorical(y_test,num_classes)
```

```
60000 train samples
10000 test samples
```

Without dropout with relu activation

In [3]:
```
model = Sequential()
model.add(Dense(512, activation = 'relu',input_shape = (784,)))
model.add(Dense(512, activation = 'relu'))
```

```
model.add(Dense(10, activation = 'softmax'))
model.summary()
sgd1 = SGD(learning_rate=0.01)
model.compile(loss = 'categorical_crossentropy', optimizer = sgd1, metrics = ['accuracy'])
history = model.fit(x_train,y_train,batch_size=batch_size,epochs=epochs,verbose=1,validation_data=(x_test,y_test_))
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 512)	401,920
dense_1 (Dense)	(None, 512)	262,656
dense_2 (Dense)	(None, 10)	5,130

Total params: 669,706 (2.55 MB)

Trainable params: 669,706 (2.55 MB)

Non-trainable params: 0 (0.00 B)

```
Epoch 1/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.6002 - loss: 1.6090 - val_accuracy: 0.8730 - val_loss: 0.5244
Epoch 2/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8790 - loss: 0.4862 - val_accuracy: 0.9014 - val_loss: 0.3686
Epoch 3/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8980 - loss: 0.3731 - val_accuracy: 0.9117 - val_loss: 0.3183
Epoch 4/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9079 - loss: 0.3255 - val_accuracy: 0.9185 - val_loss: 0.2891
Epoch 5/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9167 - loss: 0.2995 - val_accuracy: 0.9244 - val_loss: 0.2702
Epoch 6/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9215 - loss: 0.2764 - val_accuracy: 0.9267 - val_loss: 0.2539
Epoch 7/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9277 - loss: 0.2582 - val_accuracy: 0.9320 - val_loss: 0.2401
Epoch 8/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9337 - loss: 0.2403 - val_accuracy: 0.9344 - val_loss: 0.2297
Epoch 9/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9353 - loss: 0.2265 - val_accuracy: 0.9387 - val_loss: 0.2193
Epoch 10/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9371 - loss: 0.2223 - val_accuracy: 0.9406 - val_loss: 0.2085
Epoch 11/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9402 - loss: 0.2089 - val_accuracy: 0.9426 - val_loss: 0.2010
Epoch 12/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9436 - loss: 0.2028 - val_accuracy: 0.9454 - val_loss: 0.1931
Epoch 13/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9456 - loss: 0.1934 - val_accuracy: 0.9459 - val_loss: 0.1884
Epoch 14/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9469 - loss: 0.1909 - val_accuracy: 0.9480 - val_loss: 0.1795
Epoch 15/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9503 - loss: 0.1800 - val_accuracy: 0.9504 - val_loss: 0.1750
Epoch 16/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9528 - loss: 0.1709 - val_accuracy: 0.9507 - val_loss: 0.1692
Epoch 17/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9532 - loss: 0.1643 - val_accuracy: 0.9525 - val_loss: 0.1653
Epoch 18/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9539 - loss: 0.1617 - val_accuracy: 0.9554 - val_loss: 0.1599
Epoch 19/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9575 - loss: 0.1556 - val_accuracy: 0.9555 - val_loss: 0.1544
Epoch 20/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9570 - loss: 0.1519 - val_accuracy: 0.9563 - val_loss: 0.1502
Epoch 21/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9601 - loss: 0.1456 - val_accuracy: 0.9566 - val_loss: 0.1495
Epoch 22/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9608 - loss: 0.1404 - val_accuracy: 0.9590 - val_loss: 0.1433
Epoch 23/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9634 - loss: 0.1343 - val_accuracy: 0.9592 - val_loss: 0.1394
Epoch 24/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9629 - loss: 0.1343 - val_accuracy: 0.9588 - val_loss: 0.1374
Epoch 25/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9645 - loss: 0.1277 - val_accuracy: 0.9607 - val_loss: 0.1341
Epoch 26/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9654 - loss: 0.1244 - val_accuracy: 0.9610 - val_loss: 0.1301
Epoch 27/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9661 - loss: 0.1199 - val_accuracy: 0.9625 - val_loss: 0.1289
Epoch 28/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9672 - loss: 0.1170 - val_accuracy: 0.9617 - val_loss: 0.1262
Epoch 29/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9689 - loss: 0.1140 - val_accuracy: 0.9636 - val_loss: 0.1239
Epoch 30/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9685 - loss: 0.1148 - val_accuracy: 0.9638 - val_loss: 0.1207
Epoch 31/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9708 - loss: 0.1069 - val_accuracy: 0.9641 - val_loss: 0.1195
Epoch 32/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9717 - loss: 0.1042 - val_accuracy: 0.9650 - val_loss: 0.1164
Epoch 33/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9725 - loss: 0.1000 - val_accuracy: 0.9664 - val_loss: 0.1135
Epoch 34/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9715 - loss: 0.1018 - val_accuracy: 0.9662 - val_loss: 0.1129
Epoch 35/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9734 - loss: 0.0983 - val_accuracy: 0.9667 - val_loss: 0.1102
Epoch 36/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9731 - loss: 0.0988 - val_accuracy: 0.9671 - val_loss: 0.1090
Epoch 37/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9729 - loss: 0.0969 - val_accuracy: 0.9667 - val_loss: 0.1085
Epoch 38/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9748 - loss: 0.0924 - val_accuracy: 0.9679 - val_loss: 0.1065
Epoch 39/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9748 - loss: 0.0902 - val_accuracy: 0.9694 - val_loss: 0.1048
Epoch 40/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9766 - loss: 0.0863 - val_accuracy: 0.9691 - val_loss: 0.1043
Epoch 41/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9767 - loss: 0.0853 - val_accuracy: 0.9695 - val_loss: 0.1031
Epoch 42/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9776 - loss: 0.0841 - val_accuracy: 0.9696 - val_loss: 0.1002
Epoch 43/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9774 - loss: 0.0822 - val_accuracy: 0.9701 - val_loss: 0.0997
```

```
Epoch 44/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9778 - loss: 0.0818 - val_accuracy: 0.9705 - val_loss: 0.0974
Epoch 45/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9788 - loss: 0.0764 - val_accuracy: 0.9717 - val_loss: 0.0967
Epoch 46/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9797 - loss: 0.0754 - val_accuracy: 0.9715 - val_loss: 0.0957
Epoch 47/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9799 - loss: 0.0731 - val_accuracy: 0.9710 - val_loss: 0.0944
Epoch 48/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9794 - loss: 0.0758 - val_accuracy: 0.9724 - val_loss: 0.0925
Epoch 49/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9797 - loss: 0.0748 - val_accuracy: 0.9719 - val_loss: 0.0932
Epoch 50/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9817 - loss: 0.0674 - val_accuracy: 0.9723 - val_loss: 0.0914
```

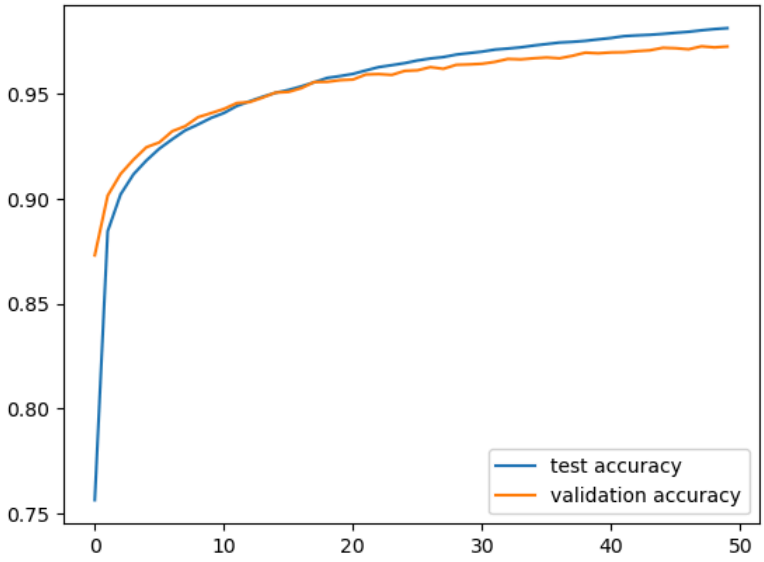
In [4]:
```python
y_prob = model.predict(x_test)
y_pred = y_prob.argmax(axis = 1)
from sklearn.metrics import accuracy_score
print(f"Accuracy:{accuracy_score(y_test,y_pred)*100:.2f}%")
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
Accuracy:97.23%
```

In [5]:
```python
score = model.evaluate(x_test,y_test_, verbose = 1)
print("Test loss:", score[0])
print(f"Test Accuracy:{score[1]*100:.2f}%")
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9669 - loss: 0.1094
Test loss: 0.09142811596393585
Test Accuracy:97.23%
```

In [6]:
```python
plt.plot(history.history['accuracy'],label='test accuracy')
plt.plot(history.history['val_accuracy'],label='validation accuracy')
plt.legend()
plt.show()
```



Without dropout using sigmoid activation

In [7]:
```python
model = Sequential()
model.add(Dense(512, activation = 'sigmoid',input_shape = (784,)))
model.add(Dense(512, activation = 'sigmoid'))
model.add(Dense(10, activation = 'softmax'))
model.summary()
sgd1 = SGD(learning_rate=0.01)
model.compile(loss = 'categorical_crossentropy', optimizer = sgd1, metrics = ['accuracy'])
history = model.fit(x_train,y_train,batch_size=batch_size,epochs=epochs,verbose=1,validation_data=(x_test,y_test_))
score = model.evaluate(x_test,y_test_, verbose = 1)
print("Test loss:", score[0])
print(f"Test Accuracy:{score[1]*100:.2f}%")
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
dense_3 (Dense)	(None, 512)	401,920
dense_4 (Dense)	(None, 512)	262,656
dense_5 (Dense)	(None, 10)	5,130

Total params: 669,706 (2.55 MB)

Trainable params: 669,706 (2.55 MB)

Non-trainable params: 0 (0.00 B)

```
Epoch 1/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.1514 - loss: 2.3069 - val_accuracy: 0.2339 - val_loss: 2.2321
Epoch 2/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.3451 - loss: 2.2118 - val_accuracy: 0.4213 - val_loss: 2.1418
Epoch 3/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4897 - loss: 2.1128 - val_accuracy: 0.4130 - val_loss: 2.0063
Epoch 4/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.5633 - loss: 1.9659 - val_accuracy: 0.5803 - val_loss: 1.8083
Epoch 5/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.6322 - loss: 1.7591 - val_accuracy: 0.6437 - val_loss: 1.5656
Epoch 6/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.6756 - loss: 1.5155 - val_accuracy: 0.6589 - val_loss: 1.3284
Epoch 7/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.7187 - loss: 1.2881 - val_accuracy: 0.7564 - val_loss: 1.1290
Epoch 8/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.7550 - loss: 1.1044 - val_accuracy: 0.7752 - val_loss: 0.9824
Epoch 9/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.7742 - loss: 0.9683 - val_accuracy: 0.8056 - val_loss: 0.8730
Epoch 10/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.7956 - loss: 0.8654 - val_accuracy: 0.8137 - val_loss: 0.7864
Epoch 11/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8099 - loss: 0.7868 - val_accuracy: 0.8183 - val_loss: 0.7203
Epoch 12/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.8194 - loss: 0.7253 - val_accuracy: 0.8313 - val_loss: 0.6665
Epoch 13/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8328 - loss: 0.6662 - val_accuracy: 0.8413 - val_loss: 0.6221
Epoch 14/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8393 - loss: 0.6276 - val_accuracy: 0.8474 - val_loss: 0.5851
Epoch 15/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.8457 - loss: 0.5921 - val_accuracy: 0.8542 - val_loss: 0.5548
Epoch 16/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8512 - loss: 0.5665 - val_accuracy: 0.8595 - val_loss: 0.5290
Epoch 17/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8597 - loss: 0.5322 - val_accuracy: 0.8650 - val_loss: 0.5060
Epoch 18/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8651 - loss: 0.5139 - val_accuracy: 0.8680 - val_loss: 0.4884
Epoch 19/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8695 - loss: 0.4909 - val_accuracy: 0.8726 - val_loss: 0.4701
Epoch 20/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8703 - loss: 0.4848 - val_accuracy: 0.8771 - val_loss: 0.4561
Epoch 21/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8742 - loss: 0.4635 - val_accuracy: 0.8792 - val_loss: 0.4429
Epoch 22/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8756 - loss: 0.4547 - val_accuracy: 0.8814 - val_loss: 0.4321
Epoch 23/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8795 - loss: 0.4461 - val_accuracy: 0.8842 - val_loss: 0.4213
Epoch 24/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8800 - loss: 0.4385 - val_accuracy: 0.8853 - val_loss: 0.4126
Epoch 25/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8838 - loss: 0.4228 - val_accuracy: 0.8874 - val_loss: 0.4051
Epoch 26/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8880 - loss: 0.4116 - val_accuracy: 0.8891 - val_loss: 0.3973
Epoch 27/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.8880 - loss: 0.4082 - val_accuracy: 0.8923 - val_loss: 0.3904
Epoch 28/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.8882 - loss: 0.4033 - val_accuracy: 0.8950 - val_loss: 0.3835
Epoch 29/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.8911 - loss: 0.3952 - val_accuracy: 0.8946 - val_loss: 0.3789
Epoch 30/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.8910 - loss: 0.3904 - val_accuracy: 0.8954 - val_loss: 0.3732
Epoch 31/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.8928 - loss: 0.3825 - val_accuracy: 0.8967 - val_loss: 0.3691
Epoch 32/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.8932 - loss: 0.3835 - val_accuracy: 0.8969 - val_loss: 0.3647
Epoch 33/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8961 - loss: 0.3772 - val_accuracy: 0.9002 - val_loss: 0.3610
Epoch 34/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.8963 - loss: 0.3767 - val_accuracy: 0.8990 - val_loss: 0.3560
Epoch 35/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8957 - loss: 0.3730 - val_accuracy: 0.9001 - val_loss: 0.3533
Epoch 36/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8957 - loss: 0.3672 - val_accuracy: 0.9004 - val_loss: 0.3501
Epoch 37/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8993 - loss: 0.3581 - val_accuracy: 0.8995 - val_loss: 0.3481
Epoch 38/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8993 - loss: 0.3598 - val_accuracy: 0.9015 - val_loss: 0.3437
Epoch 39/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8979 - loss: 0.3590 - val_accuracy: 0.9019 - val_loss: 0.3415
Epoch 40/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9009 - loss: 0.3501 - val_accuracy: 0.9016 - val_loss: 0.3387
Epoch 41/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8989 - loss: 0.3580 - val_accuracy: 0.9028 - val_loss: 0.3360
Epoch 42/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9012 - loss: 0.3505 - val_accuracy: 0.9043 - val_loss: 0.3331
Epoch 43/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9002 - loss: 0.3478 - val_accuracy: 0.9042 - val_loss: 0.3321
```

```
Epoch 44/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9023 - loss: 0.3449 - val_accuracy: 0.9041 - val_loss: 0.3292
Epoch 45/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9025 - loss: 0.3400 - val_accuracy: 0.9053 - val_loss: 0.3275
Epoch 46/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9044 - loss: 0.3356 - val_accuracy: 0.9059 - val_loss: 0.3253
Epoch 47/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9032 - loss: 0.3348 - val_accuracy: 0.9059 - val_loss: 0.3228
Epoch 48/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9061 - loss: 0.3352 - val_accuracy: 0.9070 - val_loss: 0.3213
Epoch 49/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9042 - loss: 0.3342 - val_accuracy: 0.9069 - val_loss: 0.3201
Epoch 50/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9038 - loss: 0.3339 - val_accuracy: 0.9077 - val_loss: 0.3187
313/313 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8962 - loss: 0.3621
Test loss: 0.3186941146850586
Test Accuracy:90.77%
```

Q2- Regularization Techniques

Using dropout(0.2)

In [8]:
```
model = Sequential()
model.add(Dense(512, activation = 'relu',input_shape = (784,)))
model.add(Dropout(0.2))
model.add(Dense(512, activation = 'relu'))
model.add(Dropout(0.2))
model.add(Dense(10, activation = 'softmax'))
model.summary()
sgd1 = SGD(learning_rate=0.01)
model.compile(loss = 'categorical_crossentropy', optimizer = sgd1, metrics = ['accuracy'])
history = model.fit(x_train,y_train,batch_size=batch_size,epochs=epochs,verbose=1,validation_data=(x_test,y_test_))
score = model.evaluate(x_test,y_test_, verbose = 1)
print("Test loss:", score[0])
print(f"Test Accuracy:{score[1]*100:.2f}%")
```

Model: "sequential_2"

Layer (type)	Output Shape	Param #
dense_6 (Dense)	(None, 512)	401,920
dropout (Dropout)	(None, 512)	0
dense_7 (Dense)	(None, 512)	262,656
dropout_1 (Dropout)	(None, 512)	0
dense_8 (Dense)	(None, 10)	5,130

Total params: 669,706 (2.55 MB)

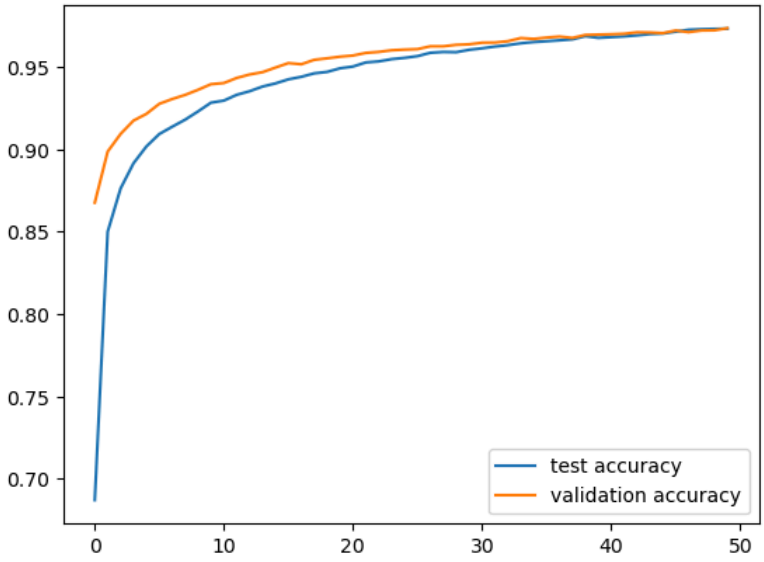
Trainable params: 669,706 (2.55 MB)

Non-trainable params: 0 (0.00 B)

```
Epoch 1/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.5206 - loss: 1.6797 - val_accuracy: 0.8676 - val_loss: 0.5445
Epoch 2/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.8382 - loss: 0.5817 - val_accuracy: 0.8986 - val_loss: 0.3766
Epoch 3/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.8708 - loss: 0.4441 - val_accuracy: 0.9093 - val_loss: 0.3227
Epoch 4/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.8873 - loss: 0.3887 - val_accuracy: 0.9174 - val_loss: 0.2939
Epoch 5/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.8993 - loss: 0.3489 - val_accuracy: 0.9215 - val_loss: 0.2720
Epoch 6/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9082 - loss: 0.3207 - val_accuracy: 0.9276 - val_loss: 0.2553
Epoch 7/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9113 - loss: 0.3112 - val_accuracy: 0.9305 - val_loss: 0.2415
Epoch 8/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9156 - loss: 0.2882 - val_accuracy: 0.9330 - val_loss: 0.2300
Epoch 9/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9206 - loss: 0.2746 - val_accuracy: 0.9361 - val_loss: 0.2184
Epoch 10/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9277 - loss: 0.2577 - val_accuracy: 0.9395 - val_loss: 0.2090
Epoch 11/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9286 - loss: 0.2444 - val_accuracy: 0.9402 - val_loss: 0.2000
Epoch 12/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9318 - loss: 0.2353 - val_accuracy: 0.9433 - val_loss: 0.1915
Epoch 13/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9356 - loss: 0.2217 - val_accuracy: 0.9454 - val_loss: 0.1854
Epoch 14/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9381 - loss: 0.2136 - val_accuracy: 0.9468 - val_loss: 0.1782
Epoch 15/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9403 - loss: 0.2064 - val_accuracy: 0.9496 - val_loss: 0.1711
Epoch 16/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9431 - loss: 0.1969 - val_accuracy: 0.9523 - val_loss: 0.1658
Epoch 17/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9441 - loss: 0.1914 - val_accuracy: 0.9516 - val_loss: 0.1613
Epoch 18/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9457 - loss: 0.1857 - val_accuracy: 0.9543 - val_loss: 0.1556
Epoch 19/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9468 - loss: 0.1805 - val_accuracy: 0.9552 - val_loss: 0.1508
Epoch 20/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9490 - loss: 0.1763 - val_accuracy: 0.9562 - val_loss: 0.1466
Epoch 21/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9490 - loss: 0.1746 - val_accuracy: 0.9569 - val_loss: 0.1432
Epoch 22/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9535 - loss: 0.1637 - val_accuracy: 0.9585 - val_loss: 0.1391
Epoch 23/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9519 - loss: 0.1635 - val_accuracy: 0.9591 - val_loss: 0.1363
Epoch 24/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9528 - loss: 0.1609 - val_accuracy: 0.9601 - val_loss: 0.1324
Epoch 25/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9549 - loss: 0.1533 - val_accuracy: 0.9605 - val_loss: 0.1298
Epoch 26/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9557 - loss: 0.1536 - val_accuracy: 0.9608 - val_loss: 0.1271
Epoch 27/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9589 - loss: 0.1423 - val_accuracy: 0.9625 - val_loss: 0.1235
Epoch 28/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9596 - loss: 0.1388 - val_accuracy: 0.9625 - val_loss: 0.1213
Epoch 29/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9588 - loss: 0.1374 - val_accuracy: 0.9634 - val_loss: 0.1188
Epoch 30/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9596 - loss: 0.1365 - val_accuracy: 0.9637 - val_loss: 0.1164
Epoch 31/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9611 - loss: 0.1331 - val_accuracy: 0.9647 - val_loss: 0.1141
Epoch 32/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9634 - loss: 0.1264 - val_accuracy: 0.9648 - val_loss: 0.1125
Epoch 33/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9643 - loss: 0.1255 - val_accuracy: 0.9656 - val_loss: 0.1101
Epoch 34/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9635 - loss: 0.1249 - val_accuracy: 0.9675 - val_loss: 0.1084
Epoch 35/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9657 - loss: 0.1194 - val_accuracy: 0.9670 - val_loss: 0.1068
Epoch 36/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9653 - loss: 0.1209 - val_accuracy: 0.9678 - val_loss: 0.1045
Epoch 37/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9650 - loss: 0.1174 - val_accuracy: 0.9684 - val_loss: 0.1029
Epoch 38/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9675 - loss: 0.1122 - val_accuracy: 0.9676 - val_loss: 0.1015
Epoch 39/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9689 - loss: 0.1106 - val_accuracy: 0.9694 - val_loss: 0.0999
Epoch 40/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9684 - loss: 0.1090 - val_accuracy: 0.9696 - val_loss: 0.0990
Epoch 41/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9676 - loss: 0.1087 - val_accuracy: 0.9698 - val_loss: 0.0976
Epoch 42/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9673 - loss: 0.1103 - val_accuracy: 0.9701 - val_loss: 0.0963
Epoch 43/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9692 - loss: 0.1053 - val_accuracy: 0.9710 - val_loss: 0.0945
```

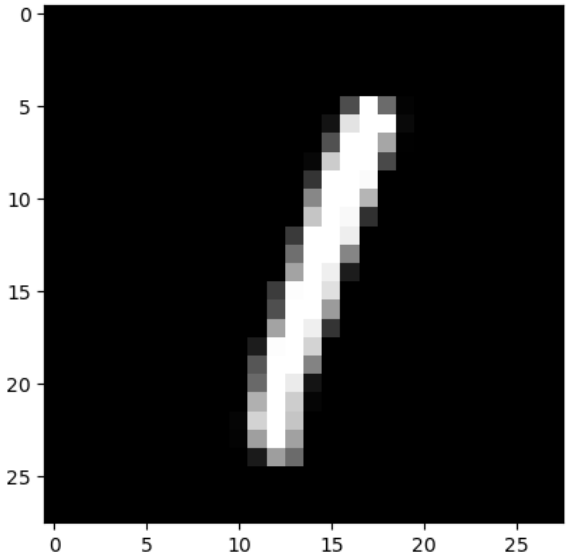
```
Epoch 44/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9709 - loss: 0.1008 - val_accuracy: 0.9709 - val_loss: 0.0937
Epoch 45/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9696 - loss: 0.1059 - val_accuracy: 0.9705 - val_loss: 0.0928
Epoch 46/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9713 - loss: 0.0994 - val_accuracy: 0.9721 - val_loss: 0.0917
Epoch 47/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9720 - loss: 0.0974 - val_accuracy: 0.9711 - val_loss: 0.0904
Epoch 48/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9723 - loss: 0.0933 - val_accuracy: 0.9721 - val_loss: 0.0894
Epoch 49/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9732 - loss: 0.0949 - val_accuracy: 0.9722 - val_loss: 0.0885
Epoch 50/50
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9724 - loss: 0.0956 - val_accuracy: 0.9735 - val_loss: 0.0870
313/313 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9687 - loss: 0.1040
Test loss: 0.08701445907354355
Test Accuracy:97.35%
```

In [9]:
```
plt.plot(history.history['accuracy'],label='test accuracy')
plt.plot(history.history['val_accuracy'],label='validation accuracy')
plt.legend()
plt.show()
```



Checking the image and output

In [10]:
```
plt.imshow(x_test[5].reshape(28, 28),cmap = 'gray')
plt.show()
print(f"Label:{model.predict(x_test[5].reshape(-1,784),verbose=0).argmax(axis = 1)}")
```




```
Label:[1]
```

Early stopping

In [11]:
```
from sklearn.model_selection import train_test_split
(x_train,y_train),(x_test,y_test) = mnist.load_data()
x_subtrain,x_valid,y_subtrain,y_valid = train_test_split(x_train,y_train,test_size = 0.10, random_state = 1)
```

In [12]:
```
x_train = x_train/255
x_test = x_test/255
x_subtrain = x_subtrain/255
x_valid=x_valid/255
```

making the ANN

In [13]:
```
model = Sequential()
model.add(Flatten(input_shape = (28,28)))
model.add(Dense(512, activation = 'relu'))
model.add(Dense(512, activation = 'relu'))
model.add(Dense(10, activation = 'softmax'))
model.summary()
sgd1 = SGD(learning_rate=0.01)
```

Model: "sequential_3"

Layer (type)	Output Shape	Param #
flatten (Flatten)	(None, 784)	0
dense_9 (Dense)	(None, 512)	401,920
dense_10 (Dense)	(None, 512)	262,656
dense_11 (Dense)	(None, 10)	5,130

Total params: 669,706 (2.55 MB)

Trainable params: 669,706 (2.55 MB)

Non-trainable params: 0 (0.00 B)

In [14]:
```
from keras.callbacks import EarlyStopping
model.compile(loss= 'sparse_categorical_crossentropy',optimizer = sgd1,metrics = ['accuracy'])
estop = EarlyStopping(monitor = 'val_loss', min_delta = 1e-3, mode = 'min', patience = 4, verbose = 1, restore_best_weight
history = model.fit(x_subtrain,y_subtrain, batch_size=batch_size, epochs = 50, verbose = 1, validation_data=(x_valid,y_val
```

```
Epoch 1/50
422/422 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.6003 - loss: 1.6514 - val_accuracy: 0.8655 - val_loss: 0.5795
Epoch 2/50
422/422 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8713 - loss: 0.5177 - val_accuracy: 0.8895 - val_loss: 0.4065
Epoch 3/50
422/422 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8961 - loss: 0.3824 - val_accuracy: 0.9010 - val_loss: 0.3513
Epoch 4/50
422/422 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9060 - loss: 0.3383 - val_accuracy: 0.9092 - val_loss: 0.3187
Epoch 5/50
422/422 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9122 - loss: 0.3079 - val_accuracy: 0.9167 - val_loss: 0.2962
Epoch 6/50
422/422 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9177 - loss: 0.2860 - val_accuracy: 0.9205 - val_loss: 0.2813
Epoch 7/50
422/422 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9239 - loss: 0.2685 - val_accuracy: 0.9252 - val_loss: 0.2681
Epoch 8/50
422/422 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9296 - loss: 0.2497 - val_accuracy: 0.9263 - val_loss: 0.2562
Epoch 9/50
422/422 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9323 - loss: 0.2398 - val_accuracy: 0.9297 - val_loss: 0.2469
Epoch 10/50
422/422 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9340 - loss: 0.2308 - val_accuracy: 0.9325 - val_loss: 0.2388
Epoch 11/50
422/422 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9382 - loss: 0.2194 - val_accuracy: 0.9350 - val_loss: 0.2292
Epoch 12/50
422/422 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9410 - loss: 0.2101 - val_accuracy: 0.9390 - val_loss: 0.2204
Epoch 13/50
422/422 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9420 - loss: 0.2071 - val_accuracy: 0.9412 - val_loss: 0.2138
Epoch 14/50
422/422 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9462 - loss: 0.1918 - val_accuracy: 0.9418 - val_loss: 0.2087
Epoch 15/50
422/422 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9470 - loss: 0.1845 - val_accuracy: 0.9447 - val_loss: 0.2025
Epoch 16/50
422/422 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9483 - loss: 0.1800 - val_accuracy: 0.9450 - val_loss: 0.1982
Epoch 17/50
422/422 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9516 - loss: 0.1732 - val_accuracy: 0.9457 - val_loss: 0.1909
Epoch 18/50
422/422 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9526 - loss: 0.1666 - val_accuracy: 0.9473 - val_loss: 0.1876
Epoch 19/50
422/422 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9540 - loss: 0.1627 - val_accuracy: 0.9470 - val_loss: 0.1852
Epoch 20/50
422/422 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9562 - loss: 0.1584 - val_accuracy: 0.9497 - val_loss: 0.1784
Epoch 21/50
422/422 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9566 - loss: 0.1539 - val_accuracy: 0.9498 - val_loss: 0.1737
Epoch 22/50
422/422 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9580 - loss: 0.1478 - val_accuracy: 0.9500 - val_loss: 0.1733
Epoch 23/50
422/422 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9604 - loss: 0.1444 - val_accuracy: 0.9518 - val_loss: 0.1676
Epoch 24/50
422/422 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9593 - loss: 0.1439 - val_accuracy: 0.9532 - val_loss: 0.1661
Epoch 25/50
422/422 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9632 - loss: 0.1323 - val_accuracy: 0.9542 - val_loss: 0.1621
Epoch 26/50
422/422 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9641 - loss: 0.1322 - val_accuracy: 0.9560 - val_loss: 0.1579
Epoch 27/50
422/422 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9646 - loss: 0.1268 - val_accuracy: 0.9555 - val_loss: 0.1562
Epoch 28/50
422/422 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9646 - loss: 0.1224 - val_accuracy: 0.9575 - val_loss: 0.1519
Epoch 29/50
422/422 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9664 - loss: 0.1233 - val_accuracy: 0.9573 - val_loss: 0.1506
Epoch 30/50
422/422 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9663 - loss: 0.1198 - val_accuracy: 0.9587 - val_loss: 0.1487
Epoch 31/50
422/422 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9682 - loss: 0.1165 - val_accuracy: 0.9597 - val_loss: 0.1453
Epoch 32/50
422/422 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9691 - loss: 0.1111 - val_accuracy: 0.9603 - val_loss: 0.1436
Epoch 33/50
422/422 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9707 - loss: 0.1085 - val_accuracy: 0.9612 - val_loss: 0.1413
Epoch 34/50
422/422 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9702 - loss: 0.1091 - val_accuracy: 0.9610 - val_loss: 0.1397
Epoch 35/50
422/422 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9713 - loss: 0.1033 - val_accuracy: 0.9617 - val_loss: 0.1375
Epoch 36/50
422/422 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9729 - loss: 0.1000 - val_accuracy: 0.9625 - val_loss: 0.1363
Epoch 37/50
422/422 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9714 - loss: 0.1013 - val_accuracy: 0.9622 - val_loss: 0.1363
Epoch 38/50
422/422 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9733 - loss: 0.0973 - val_accuracy: 0.9635 - val_loss: 0.1316
Epoch 39/50
422/422 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9741 - loss: 0.0950 - val_accuracy: 0.9637 - val_loss: 0.1311
Epoch 40/50
422/422 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9743 - loss: 0.0938 - val_accuracy: 0.9642 - val_loss: 0.1282
Epoch 41/50
422/422 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9757 - loss: 0.0895 - val_accuracy: 0.9640 - val_loss: 0.1270
Epoch 42/50
422/422 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9751 - loss: 0.0902 - val_accuracy: 0.9638 - val_loss: 0.1263
Epoch 43/50
422/422 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9756 - loss: 0.0883 - val_accuracy: 0.9652 - val_loss: 0.1239
```

```
Epoch 44/50
422/422 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9768 - loss: 0.0870 - val_accuracy: 0.9655 - val_loss: 0.1237
Epoch 45/50
422/422 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9776 - loss: 0.0832 - val_accuracy: 0.9637 - val_loss: 0.1224
Epoch 46/50
422/422 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9781 - loss: 0.0803 - val_accuracy: 0.9660 - val_loss: 0.1212
Epoch 47/50
422/422 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9788 - loss: 0.0792 - val_accuracy: 0.9658 - val_loss: 0.1189
Epoch 48/50
422/422 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9791 - loss: 0.0784 - val_accuracy: 0.9657 - val_loss: 0.1188
Epoch 49/50
422/422 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9797 - loss: 0.0757 - val_accuracy: 0.9670 - val_loss: 0.1160
Epoch 50/50
422/422 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9794 - loss: 0.0759 - val_accuracy: 0.9670 - val_loss: 0.1162
Restoring model weights from the end of the best epoch: 49.
```

In [15]:
```python
score = model.evaluate(x_test,y_test, verbose = 1)
print("Test loss:", score[0])
print(f"Test Accuracy:{score[1]*100:.2f}%")
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9654 - loss: 0.1168
Test loss: 0.09891818463802338
Test Accuracy:97.10%
```

Challenging Question: Try for a scratch code for this case where you can create a custom neural network without using any inbuilt classes like sequential etc. Where you need to define a class neural network which has methods like forwardpass, backwardpass, and train. Figure out how we can do this. This model has inputs as [0, 0, 1], [0, 1, 1], [1, 0, 1], [1, 1, 1] and the expected output as [0], [1], [1], [0] in each case. So there are three features in our dataset as you see above. The activation function is to be taken as sigmoid. The architecture is like we have only one hidden layer and an output layer with one neuron. Take the error function as (1/2)(y − ŷ)^2

.

In [17]:
```python
import numpy as np

# sigmoid activation function
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

# derivative of sigmoid function
def sigmoid_derivative(x):
    return x*(1-x)

# mean squared error loss
def mse_loss(y_true,y_pred):
    return 0.5*np.mean((y_true-y_pred)**2)

# Input dataset (XOR gate inputs with bias term)
x = np.array([[0,0,1],
[0,1,1],
[1,0,1],
[1,1,1]])

# output labels
y = np.array([[0],
[1],
[1],
[0]])

# seed for reproducibility
np.random.seed(1)

# Initialize weights randomly with mean 0
input_size = 3 # 3 input features
hidden_size = 2 # 2 hidden layers
output_size = 1 # 1 output neuron

# Weights
w1 = 2 * np.random.random((input_size, hidden_size))-1
w2 = 2 * np.random.random((hidden_size, output_size))-1

# Biases
b1 = np.zeros((1, hidden_size))
b2 = np.zeros((1, output_size))

# Learning rate
lr = 0.1
# Training loop
```

```
for epoch in range(10000):

    ##------------- Forward pass -------------------
    a1 = np.dot(x,w1) + b1
    h1 = sigmoid(a1) # activation of hidden layer
    a2 = np.dot(h1,w2) + b2
    output = sigmoid(a2) # final prediction
    # loss calculation
    loss = mse_loss(y,output)
    ##-------------- Back propagation ---------------
    # output layer error
    output_error = output - y
    output_delta = output_error * sigmoid_derivative(output)
    ## hidden layer error
    hidden_error = np.dot(output_delta, w2.T)
    hidden_delta = hidden_error * sigmoid_derivative(h1)
    ##------------Updating weights and biases -----------
    w2 -= lr * np.dot(h1.T,output_delta)
    b2 -= lr * np.sum(output_delta, axis = 0, keepdims = True)
    w1 -= lr * np.dot(x.T, hidden_delta)
    b1 -= lr * np.sum(hidden_delta, axis = 0, keepdims = True)
    # Print loss every 1000 epochs
    if epoch % 1000 == 0:
        print(f"Epoch {epoch}, Loss: {loss:.4f}")

# --------- Final Output ---------
print("\nFinal predictions after training:")
print(output.round(3))
```

```
Epoch 0, Loss: 0.1267
Epoch 1000, Loss: 0.1215
Epoch 2000, Loss: 0.1029
Epoch 3000, Loss: 0.0905
Epoch 4000, Loss: 0.0828
Epoch 5000, Loss: 0.0433
Epoch 6000, Loss: 0.0105
Epoch 7000, Loss: 0.0049
Epoch 8000, Loss: 0.0031
Epoch 9000, Loss: 0.0022

Final predictions after training:
[[0.049]
 [0.945]
 [0.945]
 [0.071]]
```

In [18]:
```
y_pred_binary = (output > 0.5).astype(int)
print("Predicted labels:", y_pred_binary.ravel())
print("True labels: ", y.ravel())
```

```
Predicted labels: [0 1 1 0]
True labels:  [0 1 1 0]
```