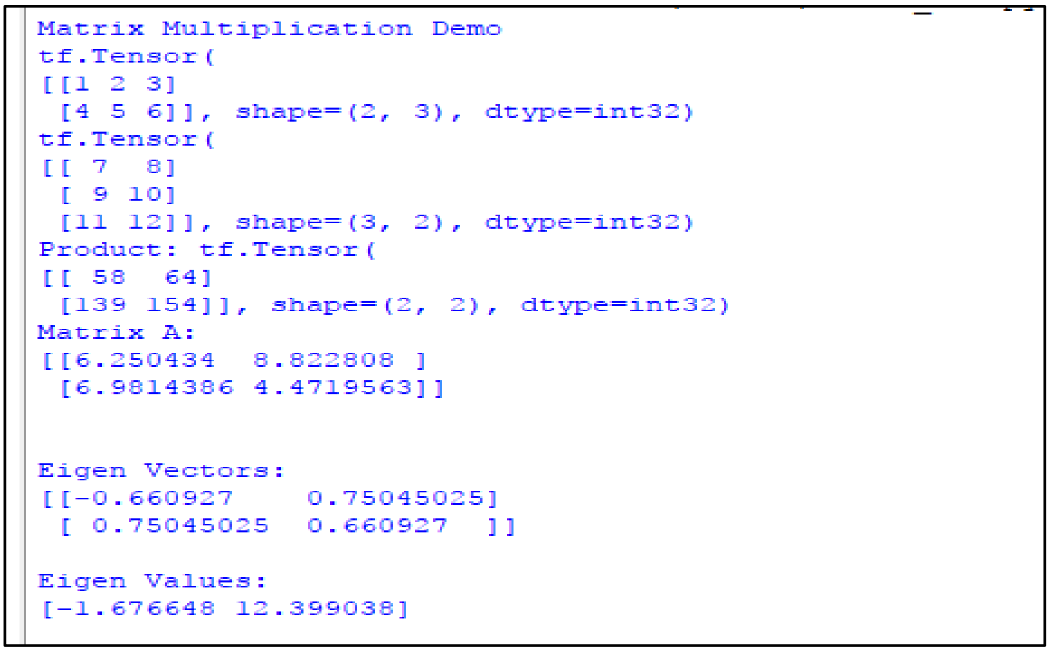
# PRACTICAL 1
# Performing Matrix Multiplication and Finding Eigenvectors and Eigenvalues Using Tensorflow

```
import tensorflow as tf
print("Matrix multiplication Demo")
x=tf.constant([1,2,3,4,5,6],shape=[2,3])
print(x)
y=tf.constant([7,8,9,10,11,12],shape=[3,2])
print(y)
z=tf.matmul(x,y)
print("Product",z)
mat_A=tf.random.uniform([2,2],minval=3,maxval=10,dtype=tf.float32,name="MatrixA")
print("Matrix A:\n{}\n\n".format(mat_A))
eigen_values_A,eigen_vectors_A=tf.linalg.eigh(mat_A)
print("Eigen Vectors:\n{}\n\nEigen Values:\n{}\n".format(eigen_vectors_A,eigen_values_A))
```

**OUTPUT:**

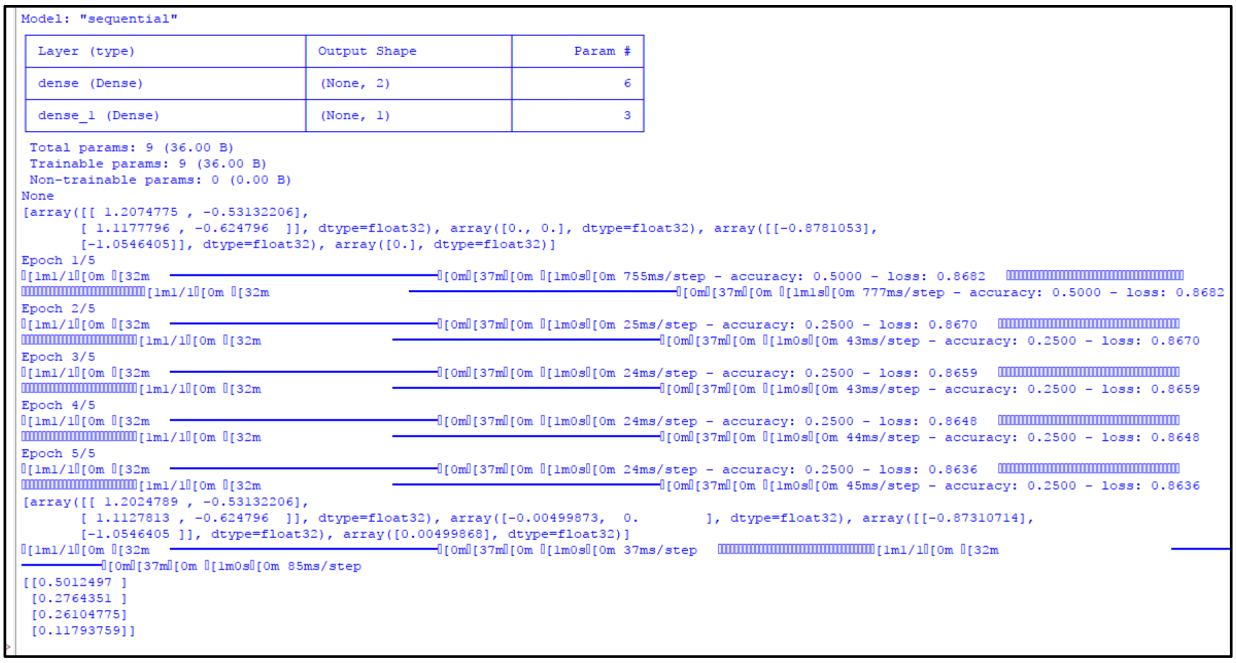


```
Matrix Multiplication Demo
tf.Tensor(
[[1 2 3]
 [4 5 6]], shape=(2, 3), dtype=int32)
tf.Tensor(
[[ 7  8]
 [ 9 10]
 [11 12]], shape=(3, 2), dtype=int32)
Product: tf.Tensor(
[[ 58  64]
 [139 154]], shape=(2, 2), dtype=int32)
Matrix A:
[[6.250434  8.822808 ]
 [6.9814386 4.4719563]]


Eigen Vectors:
[[-0.660927    0.75045025]
 [ 0.75045025  0.660927  ]]

Eigen Values:
[-1.676648 12.399038]
```

# PRACTICAL 2
## Solving Xor Problem Using Deep Feed Forward Network

```
import numpy as np
from keras.layers import Dense
from keras.models import Sequential
model=Sequential()
model.add(Dense(units=2,activation='relu',input_dim=2))
model.add(Dense(units=1,activation='sigmoid'))
model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
print(model.summary())
print(model.get_weights())
X=np.array([[0.,0.],[0.,1.],[1.,0.],[1.,1.]])
Y=np.array([0.,1.,1.,0.])
model.fit(X,Y,epochs=5,batch_size=4)
print(model.get_weights())
print(model.predict(X,batch_size=4))
```

**OUTPUT:**

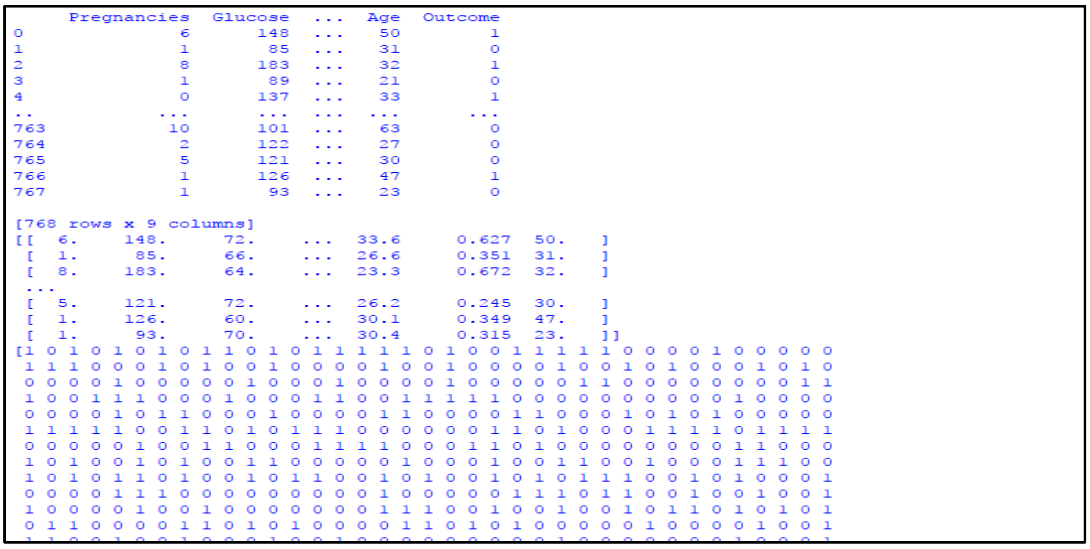


```
Model: "sequential"
┌──────────────────────────────────┬───────────────────────────┬────────────┐
│ Layer (type)                     │ Output Shape              │    Param # │
├──────────────────────────────────┼───────────────────────────┼────────────┤
│ dense (Dense)                    │ (None, 2)                 │          6 │
├──────────────────────────────────┼───────────────────────────┼────────────┤
│ dense_1 (Dense)                  │ (None, 1)                 │          3 │
└──────────────────────────────────┴───────────────────────────┴────────────┘
 Total params: 9 (36.00 B)
 Trainable params: 9 (36.00 B)
 Non-trainable params: 0 (0.00 B)
None
[array([[ 1.2074775 , -0.53132206],
       [ 1.1177796 , -0.624796  ]], dtype=float32), array([0., 0.], dtype=float32), array([[-0.8781053],
       [-1.0546405]], dtype=float32), array([0.], dtype=float32)]
Epoch 1/5
[1m1/1[0m [32m ━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m 755ms/step - accuracy: 0.5000 - loss: 0.8682  
[1m1/1[0m [32m ━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m 777ms/step - accuracy: 0.5000 - loss: 0.8682
Epoch 2/5
[1m1/1[0m [32m ━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m 25ms/step - accuracy: 0.2500 - loss: 0.8670  
[1m1/1[0m [32m ━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m 43ms/step - accuracy: 0.2500 - loss: 0.8670
Epoch 3/5
[1m1/1[0m [32m ━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m 24ms/step - accuracy: 0.2500 - loss: 0.8659  
[1m1/1[0m [32m ━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m 43ms/step - accuracy: 0.2500 - loss: 0.8659
Epoch 4/5
[1m1/1[0m [32m ━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m 24ms/step - accuracy: 0.2500 - loss: 0.8648  
[1m1/1[0m [32m ━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m 44ms/step - accuracy: 0.2500 - loss: 0.8648
Epoch 5/5
[1m1/1[0m [32m ━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m 24ms/step - accuracy: 0.2500 - loss: 0.8636  
[1m1/1[0m [32m ━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m 45ms/step - accuracy: 0.2500 - loss: 0.8636
[array([[ 1.2024789 , -0.53132206],
       [ 1.1127813 , -0.624796  ]], dtype=float32), array([-0.00499873,  0.        ], dtype=float32), array([[-0.87310714],
       [-1.0546405 ]], dtype=float32), array([0.00499868], dtype=float32)]
[1m1/1[0m [32m ━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m 37ms/step  [1m1/1[0m [32m ━━━
━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m 85ms/step
[[0.5012497 ]
 [0.2764351 ]
 [0.26104775]
 [0.11793759]]
```

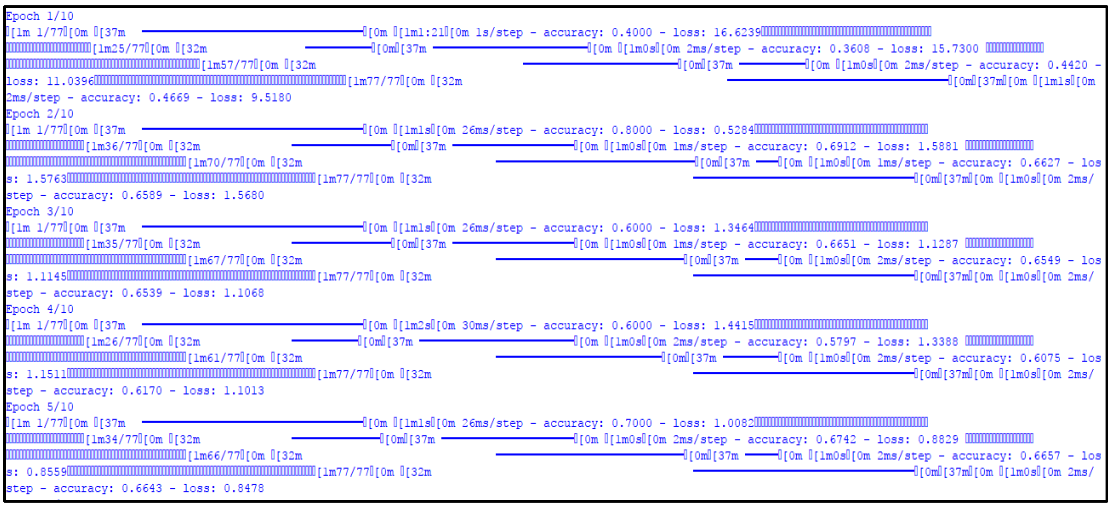
# PRACTICAL 3
## Implementing A Deep Neural Network For Performing Classification Tasks

```
from numpy import loadtxt
from keras.models import Sequential
from keras.layers import Dense
import pandas as pd
dataset=pd.read_csv('D:/Vinish/diabetes.csv',header=0)
print(dataset)
X=dataset.iloc[:,0:8].values
Y=dataset.iloc[:,8].values
print(X)
print(Y)
model=Sequential()
model.add(Dense(12,input_dim=8,activation='relu'))
model.add(Dense(8,activation='relu'))
model.add(Dense(1,activation='sigmoid'))
model.compile(loss="binary_crossentropy",optimizer='adam',metrics=['accuracy'])
model.fit(X,Y,epochs=10,batch_size=10)
_,accuracy=model.evaluate(X,Y)
print("Accuracy of Model is",(accuracy*100))
prediction=model.predict_step(X)
for i in range(5):
   print(X[i].tolist(),prediction[i],Y[i])
```

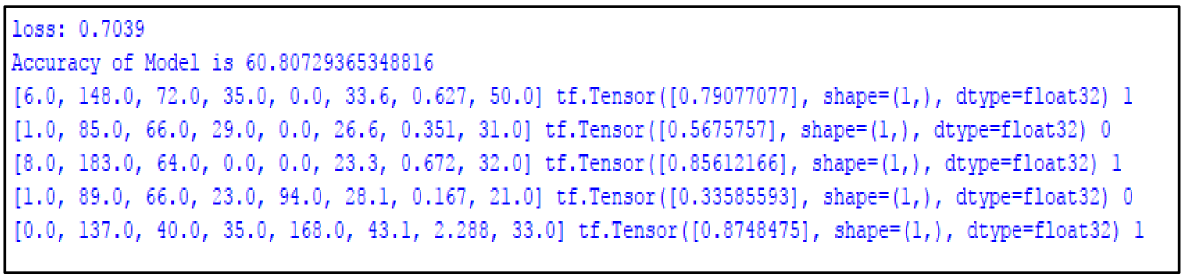
**OUTPUT:**



```
     Pregnancies  Glucose  ...  Age  Outcome
0              6      148  ...   50        1
1              1       85  ...   31        0
2              8      183  ...   32        1
3              1       89  ...   21        0
4              0      137  ...   33        1
..           ...      ...  ...  ...      ...
763           10      101  ...   63        0
764            2      122  ...   27        0
765            5      121  ...   30        0
766            1      126  ...   47        1
767            1       93  ...   23        0

[768 rows x 9 columns]
[[  6.    148.     72.    ...  33.6     0.627  50.   ]
 [  1.     85.     66.    ...  26.6     0.351  31.   ]
 [  8.    183.     64.    ...  23.3     0.672  32.   ]
 ...
 [  5.    121.     72.    ...  26.2     0.245  30.   ]
 [  1.    126.     60.    ...  30.1     0.349  47.   ]
 [  1.     93.     70.    ...  30.4     0.315  23.   ]]
[1 0 1 0 1 0 1 0 1 1 0 1 0 1 1 1 1 1 0 1 0 0 1 1 1 1 1 0 0 0 0 1 0 0 0 0 0
 1 1 1 0 0 0 1 0 1 0 0 1 0 0 0 0 1 0 0 1 0 0 0 0 1 0 0 1 0 1 0 0 0 1 0 1 0
 0 0 0 0 1 0 0 0 0 0 1 0 0 0 1 0 0 0 0 1 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 1 1
 1 0 0 1 1 1 0 0 0 1 0 0 0 1 1 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
 0 0 0 0 1 0 1 1 0 0 0 1 0 0 0 0 1 1 0 0 0 0 1 1 0 0 0 1 0 1 0 1 0 0 0 0 0
 1 1 1 1 1 0 0 1 1 0 1 0 1 1 1 0 0 0 0 0 0 1 1 0 1 0 0 0 1 1 1 1 0 1 1 1 1
 0 0 0 0 0 1 0 0 1 1 0 0 0 1 1 1 1 0 0 0 1 1 0 1 0 0 0 0 0 0 0 0 1 1 0 0 0
 1 0 1 0 0 1 0 1 0 0 1 1 0 0 0 0 0 1 0 0 0 1 0 0 1 1 0 0 1 0 0 0 1 1 1 0 0
 1 0 1 0 1 1 0 1 0 0 1 0 1 1 0 0 1 0 1 0 0 1 0 1 0 1 1 1 0 0 1 0 1 0 0 0 1
 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 1 1 0 1 1 0 0 1 0 0 1 0 0 1
 1 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 1 1 1 0 0 1 0 0 1 0 0 1 0 1 1 0 1 0 1 0 1
 0 1 1 0 0 0 0 1 1 0 1 0 1 0 0 0 0 1 1 0 1 0 1 0 0 0 0 0 1 0 0 0 0 1 0 0 1
```



```
Epoch 1/10
▯[1m 1/77▯[0m ▯[37m ━━━━━━━━━━━━━━━━━━━━▯[0m ▯[1m1:21▯[0m 1s/step - accuracy: 0.4000 - loss: 16.6239▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯
▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯[1m25/77▯[0m ▯[32m ━━━━━━▯[0m▯[37m ━━━━━━━━━━━━━━▯[0m ▯[1m0s▯[0m 2ms/step - accuracy: 0.3608 - loss: 15.7300 ▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯
▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯[1m57/77▯[0m ▯[32m ━━━━━━━━━━━━━━▯[0m▯[37m ━━━━━━▯[0m ▯[1m0s▯[0m 2ms/step - accuracy: 0.4420 -
loss: 11.0396▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯[1m77/77▯[0m ▯[32m ━━━━━━━━━━━━━━━━━━━━▯[0m▯[37m▯[0m ▯[1m1s▯[0m
2ms/step - accuracy: 0.4669 - loss: 9.5180
Epoch 2/10
▯[1m 1/77▯[0m ▯[37m ━━━━━━━━━━━━━━━━━━━━▯[0m ▯[1m1s▯[0m 26ms/step - accuracy: 0.8000 - loss: 0.5284▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯
▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯[1m36/77▯[0m ▯[32m ━━━━━━━━━▯[0m▯[37m ━━━━━━━━━━━▯[0m ▯[1m0s▯[0m 1ms/step - accuracy: 0.6912 - loss: 1.5881 ▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯
▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯[1m70/77▯[0m ▯[32m ━━━━━━━━━━━━━━━━━━▯[0m▯[37m ━━▯[0m ▯[1m0s▯[0m 1ms/step - accuracy: 0.6627 - los
s: 1.5763▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯[1m77/77▯[0m ▯[32m ━━━━━━━━━━━━━━━━━━━━▯[0m▯[37m▯[0m ▯[1m0s▯[0m 2ms/
step - accuracy: 0.6589 - loss: 1.5680
Epoch 3/10
▯[1m 1/77▯[0m ▯[37m ━━━━━━━━━━━━━━━━━━━━▯[0m ▯[1m1s▯[0m 26ms/step - accuracy: 0.6000 - loss: 1.3464▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯
▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯[1m35/77▯[0m ▯[32m ━━━━━━━━━▯[0m▯[37m ━━━━━━━━━━━▯[0m ▯[1m0s▯[0m 1ms/step - accuracy: 0.6651 - loss: 1.1287 ▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯
▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯[1m67/77▯[0m ▯[32m ━━━━━━━━━━━━━━━━━▯[0m▯[37m ━━━▯[0m ▯[1m0s▯[0m 2ms/step - accuracy: 0.6549 - los
s: 1.1145▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯[1m77/77▯[0m ▯[32m ━━━━━━━━━━━━━━━━━━━━▯[0m▯[37m▯[0m ▯[1m0s▯[0m 2ms/
step - accuracy: 0.6539 - loss: 1.1068
Epoch 4/10
▯[1m 1/77▯[0m ▯[37m ━━━━━━━━━━━━━━━━━━━━▯[0m ▯[1m2s▯[0m 30ms/step - accuracy: 0.6000 - loss: 1.4415▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯
▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯[1m26/77▯[0m ▯[32m ━━━━━━▯[0m▯[37m ━━━━━━━━━━━━━━▯[0m ▯[1m0s▯[0m 2ms/step - accuracy: 0.5797 - loss: 1.3388 ▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯
▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯[1m61/77▯[0m ▯[32m ━━━━━━━━━━━━━━━▯[0m▯[37m ━━━━━▯[0m ▯[1m0s▯[0m 2ms/step - accuracy: 0.6075 - los
s: 1.1511▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯[1m77/77▯[0m ▯[32m ━━━━━━━━━━━━━━━━━━━━▯[0m▯[37m▯[0m ▯[1m0s▯[0m 2ms/
step - accuracy: 0.6170 - loss: 1.1013
Epoch 5/10
▯[1m 1/77▯[0m ▯[37m ━━━━━━━━━━━━━━━━━━━━▯[0m ▯[1m1s▯[0m 26ms/step - accuracy: 0.7000 - loss: 1.0082▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯
▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯[1m34/77▯[0m ▯[32m ━━━━━━━━▯[0m▯[37m ━━━━━━━━━━━━▯[0m ▯[1m0s▯[0m 2ms/step - accuracy: 0.6742 - loss: 0.8829 ▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯
▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯[1m66/77▯[0m ▯[32m ━━━━━━━━━━━━━━━━━▯[0m▯[37m ━━━▯[0m ▯[1m0s▯[0m 2ms/step - accuracy: 0.6657 - los
s: 0.8559▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯[1m77/77▯[0m ▯[32m ━━━━━━━━━━━━━━━━━━━━▯[0m▯[37m▯[0m ▯[1m0s▯[0m 2ms/
step - accuracy: 0.6643 - loss: 0.8478
```



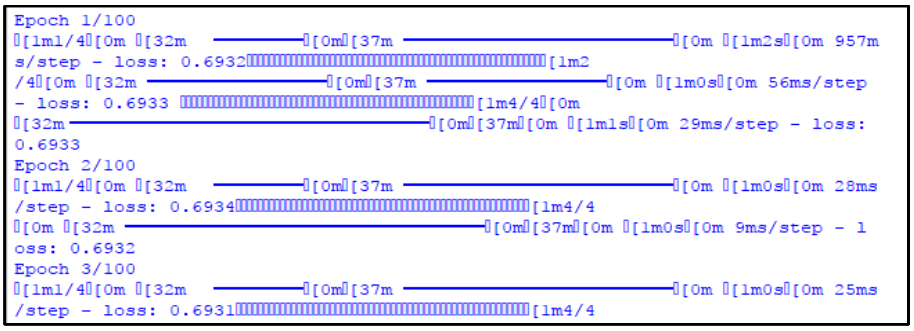
```
loss: 0.7039
Accuracy of Model is 60.80729365348816
[6.0, 148.0, 72.0, 35.0, 0.0, 33.6, 0.627, 50.0] tf.Tensor([0.79077077], shape=(1,), dtype=float32) 1
[1.0, 85.0, 66.0, 29.0, 0.0, 26.6, 0.351, 31.0] tf.Tensor([0.5675757], shape=(1,), dtype=float32) 0
[8.0, 183.0, 64.0, 0.0, 0.0, 23.3, 0.672, 32.0] tf.Tensor([0.85612166], shape=(1,), dtype=float32) 1
[1.0, 89.0, 66.0, 23.0, 94.0, 28.1, 0.167, 21.0] tf.Tensor([0.33585593], shape=(1,), dtype=float32) 0
[0.0, 137.0, 40.0, 35.0, 168.0, 43.1, 2.288, 33.0] tf.Tensor([0.8748475], shape=(1,), dtype=float32) 1
```

# PRACTICAL 4

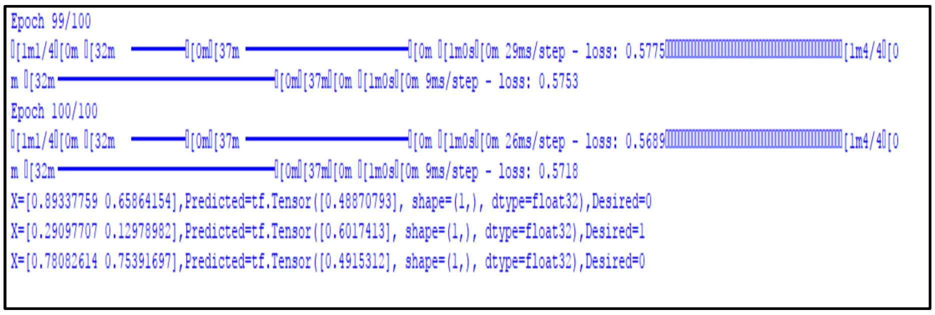
## A) Using A Deep Feed Forward Network With Two Hidden Layers For Performing Classification And Predicting The Class

```
from keras.models import Sequential
from keras.layers import Dense
from sklearn.datasets import make_blobs
from sklearn.preprocessing import MinMaxScaler
X,Y=make_blobs(n_samples=100,centers=2,n_features=2,random_state=1)
scalar=MinMaxScaler()
scalar.fit(X)
X=scalar.transform(X)
model=Sequential()
model.add(Dense(4,input_dim=2,activation='relu'))
model.add(Dense(4,activation='relu'))
model.add(Dense(1,activation='sigmoid'))
model.compile(loss='binary_crossentropy',optimizer='adam')
model.fit(X,Y,epochs=100)
Xnew,Yreal=make_blobs(n_samples=3,centers=2,n_features=2,random_state=1)
Xnew=scalar.transform(Xnew)
Ynew=model.predict_step(Xnew)
for i in range(len(Xnew)):
   print("X=%s,Predicted=%s,Desired=%s"%(Xnew[i],Ynew[i],Yreal[i]))
```

**OUTPUT:**



```
Epoch 1/100
▯[1m1/4▯[0m ▯[32m  ━━━━━━━▯[0m▯[37m ━━━━━━━━━━━━━━━━━━━━━▯[0m ▯[1m2s▯[0m 957m
s/step - loss: 0.6932▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯[1m2
/4▯[0m ▯[32m ━━━━━━━━━━━━━━▯[0m▯[37m ━━━━━━━━━━━━━━▯[0m ▯[1m0s▯[0m 56ms/step
- loss: 0.6933 ▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯[1m4/4▯[0m
▯[32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━▯[0m▯[37m▯[0m ▯[1m1s▯[0m 29ms/step - loss:
0.6933
Epoch 2/100
▯[1m1/4▯[0m ▯[32m  ━━━━━━━▯[0m▯[37m ━━━━━━━━━━━━━━━━━━━━━▯[0m ▯[1m0s▯[0m 28ms
/step - loss: 0.6934▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯[1m4/4
▯[0m ▯[32m ━━━━━━━━━━━━━━━━━━━━━━━━━━━━▯[0m▯[37m▯[0m ▯[1m0s▯[0m 9ms/step - l
oss: 0.6932
Epoch 3/100
▯[1m1/4▯[0m ▯[32m  ━━━━━━━▯[0m▯[37m ━━━━━━━━━━━━━━━━━━━━━▯[0m ▯[1m0s▯[0m 25ms
/step - loss: 0.6931▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯[1m4/4
```

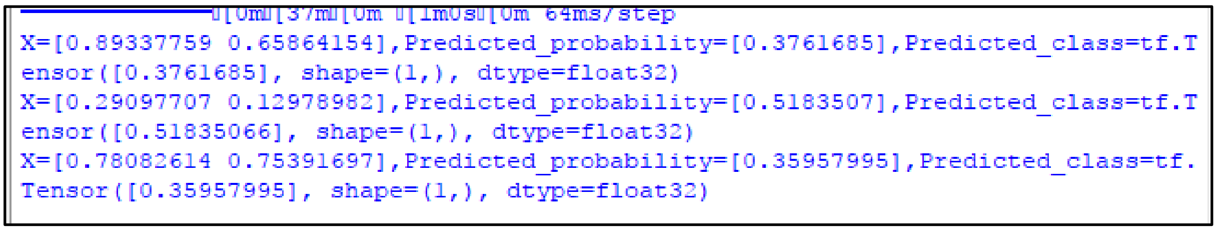


```
Epoch 99/100
▯[1m1/4▯[0m ▯[32m  ━━━━━▯[0m▯[37m ━━━━━━━━━━━━━━━━▯[0m ▯[1m0s▯[0m 29ms/step - loss: 0.5775▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯[1m4/4▯[0
m ▯[32m━━━━━━━━━━━━━━━━━━━━━━▯[0m▯[37m▯[0m ▯[1m0s▯[0m 9ms/step - loss: 0.5753
Epoch 100/100
▯[1m1/4▯[0m ▯[32m  ━━━━━▯[0m▯[37m ━━━━━━━━━━━━━━━━▯[0m ▯[1m0s▯[0m 26ms/step - loss: 0.5689▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯[1m4/4▯[0
m ▯[32m━━━━━━━━━━━━━━━━━━━━━━▯[0m▯[37m▯[0m ▯[1m0s▯[0m 9ms/step - loss: 0.5718
X=[0.89337759 0.65864154],Predicted=tf.Tensor([0.48870793], shape=(1,), dtype=float32),Desired=0
X=[0.29097707 0.12978982],Predicted=tf.Tensor([0.6017413], shape=(1,), dtype=float32),Desired=1
X=[0.78082614 0.75391697],Predicted=tf.Tensor([0.4915312], shape=(1,), dtype=float32),Desired=0
```

## B) Using a deep field forward network with two hidden layers for performing classification and predicting the probability of class.

```
from keras.models import Sequential
from keras.layers import Dense
from sklearn.datasets import make_blobs
from sklearn.preprocessing import MinMaxScaler
X,Y=make_blobs(n_samples=100,centers=2,n_features=2,random_state=1)
scalar=MinMaxScaler()
scalar.fit(X)
X=scalar.transform(X)
model=Sequential()
model.add(Dense(4,input_dim=2,activation='relu'))
model.add(Dense(4,activation='relu'))
model.add(Dense(1,activation='sigmoid'))
model.compile(loss='binary_crossentropy',optimizer='adam')
model.fit(X,Y,epochs=100)
Xnew,Yreal=make_blobs(n_samples=3,centers=2,n_features=2,random_state=1)
Xnew=scalar.transform(Xnew)
Yclass=model.predict_step(Xnew)
Ynew=model.predict(Xnew)
for i in range(len(Xnew)):
   print("X=%s,Predicted_probability=%s,Predicted_class=%s"%(Xnew[i],Ynew[i],Yclass[i]))
```

**OUTPUT:**

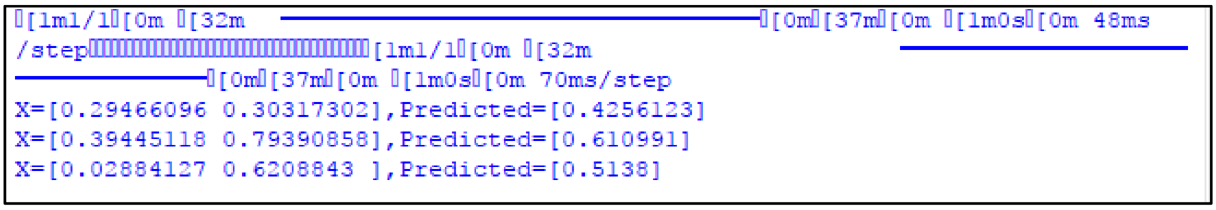


```
━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m 64ms/step
X=[0.89337759 0.65864154],Predicted_probability=[0.3761685],Predicted_class=tf.T
ensor([0.3761685], shape=(1,), dtype=float32)
X=[0.29097707 0.12978982],Predicted_probability=[0.5183507],Predicted_class=tf.T
ensor([0.51835066], shape=(1,), dtype=float32)
X=[0.78082614 0.75391697],Predicted_probability=[0.35957995],Predicted_class=tf.
Tensor([0.35957995], shape=(1,), dtype=float32)
```

## C) Using A Deep Field Forward Network With Two Hidden Layers For Performing Linear Regression And Predicting Values

```
from keras.models import Sequential
from keras.layers import Dense
from sklearn.datasets import make_regression
from sklearn.preprocessing import MinMaxScaler
X,Y=make_regression(n_samples=100,n_features=2,noise=0.1,random_state=1)
scalarX,scalarY=MinMaxScaler(),MinMaxScaler()
scalarX.fit(X)
scalarY.fit(Y.reshape(100,1))
X=scalarX.transform(X)
Y=scalarY.transform(Y.reshape(100,1))
model=Sequential()
model.add(Dense(4,input_dim=2,activation='relu'))
model.add(Dense(4,activation='relu'))
model.add(Dense(1,activation='sigmoid'))
model.compile(loss='mse',optimizer='adam')
model.fit(X,Y,epochs=100,verbose=0)
Xnew,a=make_regression(n_samples=3,n_features=2,noise=0.1,random_state=1)
Xnew=scalarX.transform(Xnew)
Ynew=model.predict(Xnew)
for i in range(len(Xnew)):
   print("X=%s,Predicted=%s"%(Xnew[i],Ynew[i]))
```

**OUTPUT:**



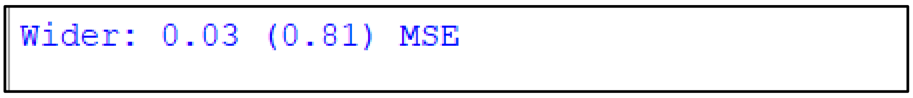
```
▯[1m1/1▯[0m ▯[32m ━━━━━━━━━━━━━━━━━━━━▯[0m▯[37m▯[0m ▯[1m0s▯[0m 48ms
/step▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯[1m1/1▯[0m ▯[32m ━━━━━━━━━━━━
━━━━━━━━▯[0m▯[37m▯[0m ▯[1m0s▯[0m 70ms/step
X=[0.29466096 0.30317302],Predicted=[0.4256123]
X=[0.39445118 0.79390858],Predicted=[0.610991]
X=[0.02884127 0.6208843 ],Predicted=[0.5138]
```

## PRACTICAL 5

## A) Evaluating feed forward deep network for regression using KFold cross validation.

```
import pandas as pd
from keras.models import Model
from keras.layers import Dense, Input
from scikeras.wrappers import KerasRegressor
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
dataframe = pd.read_csv("C:/Users/Vinish/OneDrive/Desktop/DL/housing.csv",sep='\s+',
header=None)
dataset = dataframe.values
X = dataset[:, 0:13]
Y = dataset[:, 13]
def wider_model():
   inputs = Input(shape=(13,))
   x = Dense(15, kernel_initializer='normal', activation='relu')(inputs)
   #x = Dense(20, kernel_initializer='normal', activation='relu')(inputs) Modifying Neurons
   x = Dense(13, kernel_initializer='normal', activation='relu')(x)
   outputs = Dense(1, kernel_initializer='normal')(x)
   model = Model(inputs=inputs, outputs=outputs)
   model.compile(loss='mean_squared_error', optimizer='adam')
   return model
estimators = []
estimators.append(('standardize', StandardScaler()))
estimators.append(('mlp', KerasRegressor(model=wider_model, epochs=10, batch_size=5,
verbose=0)))
pipeline = Pipeline(estimators)
kfold = KFold(n_splits=10)
results = cross_val_score(pipeline, X, Y, cv=kfold)
print("Wider: %.2f (%.2f) MSE" % (results.mean(), results.std()))
```

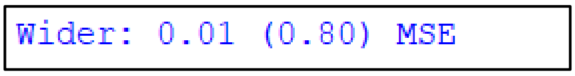
**OUTPUT:**



```
Wider: 0.03 (0.81) MSE
```

x = Dense(20, kernel_initializer='normal', activation='relu')(inputs) #Modifying Neurons

**OUTPUT:**

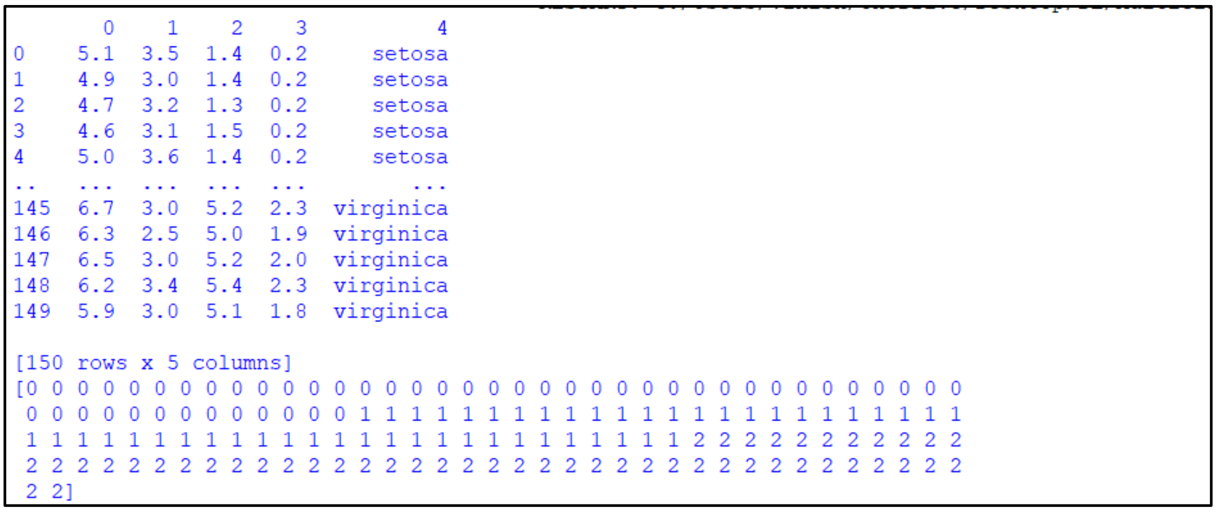


```
Wider: 0.01 (0.80) MSE
```

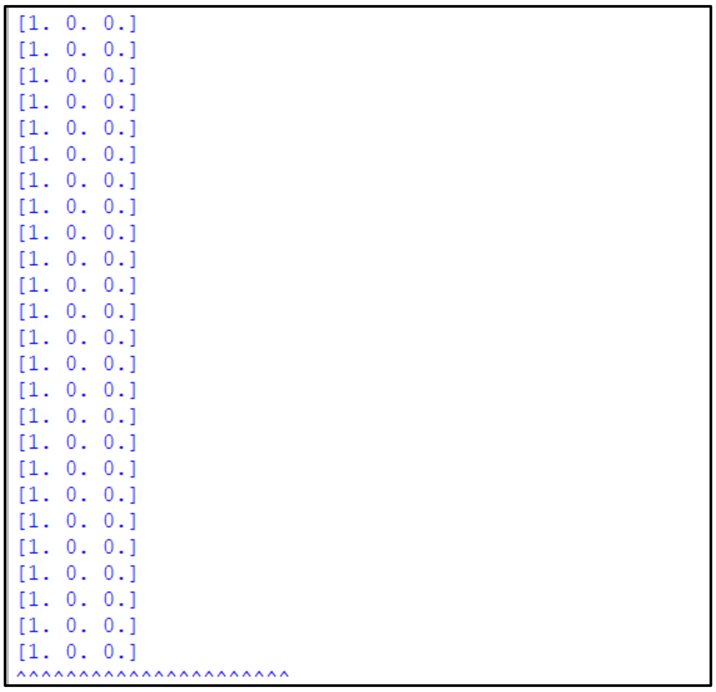
**B) Evaluating Feed Forward Deep Network For Multiclass Classification Using Kfold Cross-Validation.**

```
import pandas
from keras.models import Sequential
from keras.layers import Dense,Input
from scikeras.wrappers import KerasClassifier
from keras import utils
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.preprocessing import LabelEncoder
df=pandas.read_csv("C:/Users/Vinish/OneDrive/Desktop/DL/flowers.csv",header=None)
print(df)
X = df.iloc[:,0:4].astype(float)
y=df.iloc[:,4]
encoder=LabelEncoder()
encoder.fit(y)
encoded_y=encoder.transform(y)
print(encoded_y)
dummy_Y=utils.to_categorical(encoded_y)
print(dummy_Y)
def baseline_model():
    model = Sequential()
    model.add(Input(shape=(4,)))
    model.add(Dense(8, activation='relu'))
    model.add(Dense(3, activation='softmax'))
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model
estimator=baseline_model()
estimator.fit(X,dummy_Y,epochs=100,shuffle=True)
action=estimator.predict(X)
for i in range(25):
    print(dummy_Y[i])
print('^^^^^^^^^^^^^^^^^^^^^^')
for i in range(25):
    print(action[i])
```

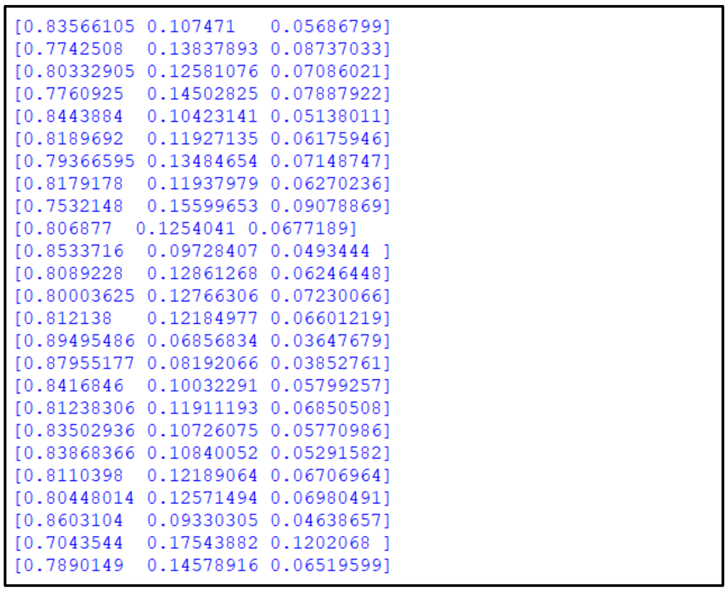
**OUTPUT:**



```
       0    1    2    3          4
0    5.1  3.5  1.4  0.2     setosa
1    4.9  3.0  1.4  0.2     setosa
2    4.7  3.2  1.3  0.2     setosa
3    4.6  3.1  1.5  0.2     setosa
4    5.0  3.6  1.4  0.2     setosa
..   ...  ...  ...  ...        ...
145  6.7  3.0  5.2  2.3  virginica
146  6.3  2.5  5.0  1.9  virginica
147  6.5  3.0  5.2  2.0  virginica
148  6.2  3.4  5.4  2.3  virginica
149  5.9  3.0  5.1  1.8  virginica

[150 rows x 5 columns]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 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 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
```



```
[1. 0. 0.]
[1. 0. 0.]
[1. 0. 0.]
[1. 0. 0.]
[1. 0. 0.]
[1. 0. 0.]
[1. 0. 0.]
[1. 0. 0.]
[1. 0. 0.]
[1. 0. 0.]
[1. 0. 0.]
[1. 0. 0.]
[1. 0. 0.]
[1. 0. 0.]
[1. 0. 0.]
[1. 0. 0.]
[1. 0. 0.]
[1. 0. 0.]
[1. 0. 0.]
[1. 0. 0.]
[1. 0. 0.]
[1. 0. 0.]
[1. 0. 0.]
[1. 0. 0.]
[1. 0. 0.]
^^^^^^^^^^^^^^^^^^^^^^
```



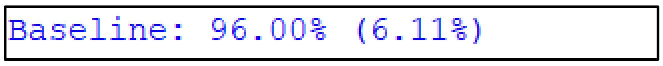
```
[0.83566105 0.107471   0.05686799]
[0.7742508  0.13837893 0.08737033]
[0.80332905 0.12581076 0.07086021]
[0.7760925  0.14502825 0.07887922]
[0.8443884  0.10423141 0.05138011]
[0.8189692  0.11927135 0.06175946]
[0.79366595 0.13484654 0.07148747]
[0.8179178  0.11937979 0.06270236]
[0.7532148  0.15599653 0.09078869]
[0.806877  0.1254041 0.0677189]
[0.8533716  0.09728407 0.0493444 ]
[0.8089228  0.12861268 0.06246448]
[0.80003625 0.12766306 0.07230066]
[0.812138   0.12184977 0.06601219]
[0.89495486 0.06856834 0.03647679]
[0.87955177 0.08192066 0.03852761]
[0.8416846  0.10032291 0.05799257]
[0.81238306 0.11911193 0.06850508]
[0.83502936 0.10726075 0.05770986]
[0.83868366 0.10840052 0.05291582]
[0.8110398  0.12189064 0.06706964]
[0.80448014 0.12571494 0.06980491]
[0.8603104  0.09330305 0.04638657]
[0.7043544  0.17543882 0.1202068 ]
[0.7890149  0.14578916 0.06519599]
```

## CODE 2:

```
import pandas as pd
from keras.models import Sequential
from keras.layers import Dense, Input
from scikeras.wrappers import KerasClassifier
from keras import utils
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.preprocessing import LabelEncoder
dataset = pd.read_csv("C:/Users/Vinish/OneDrive/Desktop/DL/flowers.csv", header=None)
dataset1 = dataset.values
X = dataset1[:, 0:4].astype(float)
Y = dataset1[:, 4]
```

```
encoder = LabelEncoder()
encoder.fit(Y)
encoder_Y = encoder.transform(Y)
print(encoder_Y)
dummy_Y = utils.to_categorical(encoder_Y)
print(dummy_Y)
def baseline_model():
    model = Sequential()
    model.add(Input(shape=(4,)))
    model.add(Dense(8, activation='relu'))
     model.add(Dense(3, activation='softmax'))
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model
estimator = KerasClassifier(
    model=baseline_model,
    epochs=100,
    batch_size=5,
    verbose=0
)
kfold = KFold(n_splits=10, shuffle=True)
results = cross_val_score(estimator, X, dummy_Y, cv=kfold)
print("Baseline: %.2f%% (%.2f%%)" % (results.mean() * 100, results.std() * 100))
```

**OUTPUT:**

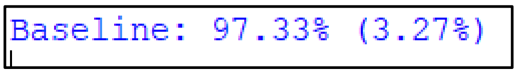


```
Baseline: 96.00% (6.11%)
```

**Changing Neuron:**

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

**OUTPUT:**

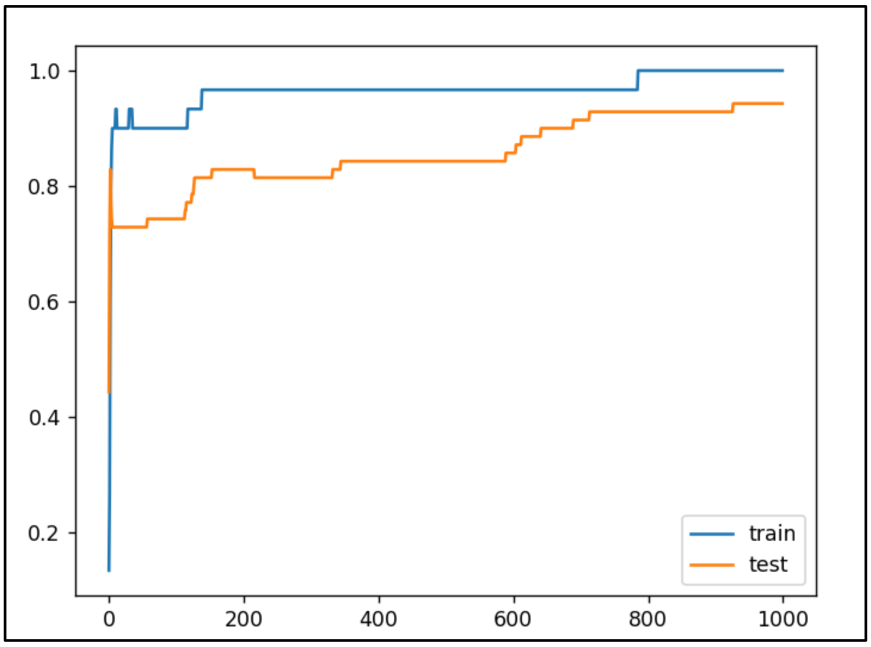


```
Baseline: 97.33% (3.27%)
```

# PRACTICAL 6
## Implementing Regularization To Avoid Overfitting In Binary Classification

```
from matplotlib import pyplot
from sklearn.datasets import make_moons
from keras.models import Sequential
from keras.layers import Dense,Input
X,Y=make_moons(n_samples=100,noise=0.2,random_state=1)
n_train=30
trainX,testX=X[:n_train,:],X[n_train:]
trainY,testY=Y[:n_train],Y[n_train:]
model = Sequential()
model.add(Input(shape=(2,)))
model.add(Dense(500, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
history=model.fit(trainX,trainY,validation_data=(testX,testY),epochs=1000)
pyplot.plot(history.history['accuracy'],label='train')
pyplot.plot(history.history['val_accuracy'],label='test')
pyplot.legend()
pyplot.show()
```

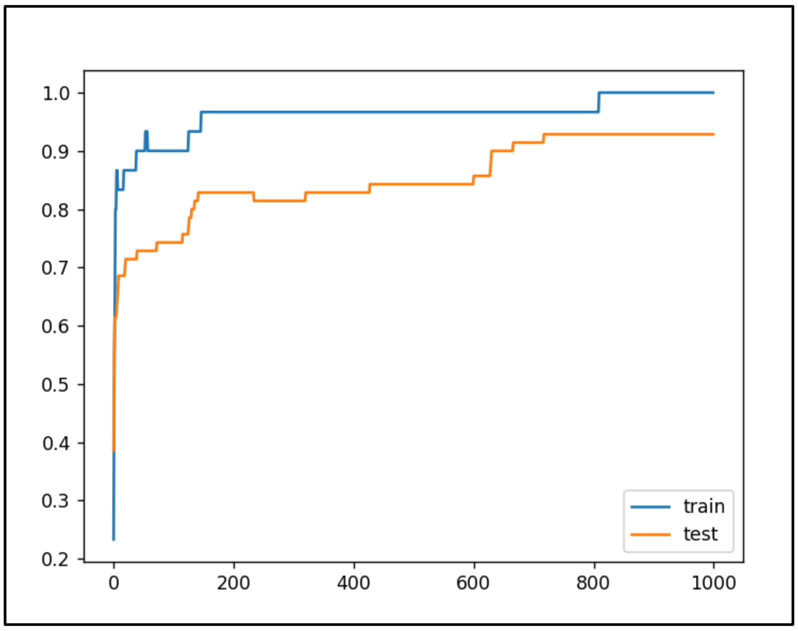
**OUTPUT:**



**Implement L2 Regularization with Alpha=0.001**

```
from matplotlib import pyplot
from sklearn.datasets import make_moons
from keras.models import Sequential
from keras.layers import Dense, Input
from keras.regularizers import l2
X, Y = make_moons(n_samples=100, noise=0.2, random_state=1)
n_train = 30
trainX, testX = X[:n_train, :], X[n_train:]
trainY, testY = Y[:n_train], Y[n_train:]
model = Sequential()
model.add(Input(shape=(2,)))
model.add(Dense(500, activation='relu', kernel_regularizer=l2(0.001)))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
history = model.fit(trainX, trainY, validation_data=(testX, testY), epochs=1000)
pyplot.plot(history.history['accuracy'], label='train')
pyplot.plot(history.history['val_accuracy'], label='test')
pyplot.legend()
pyplot.show()
```

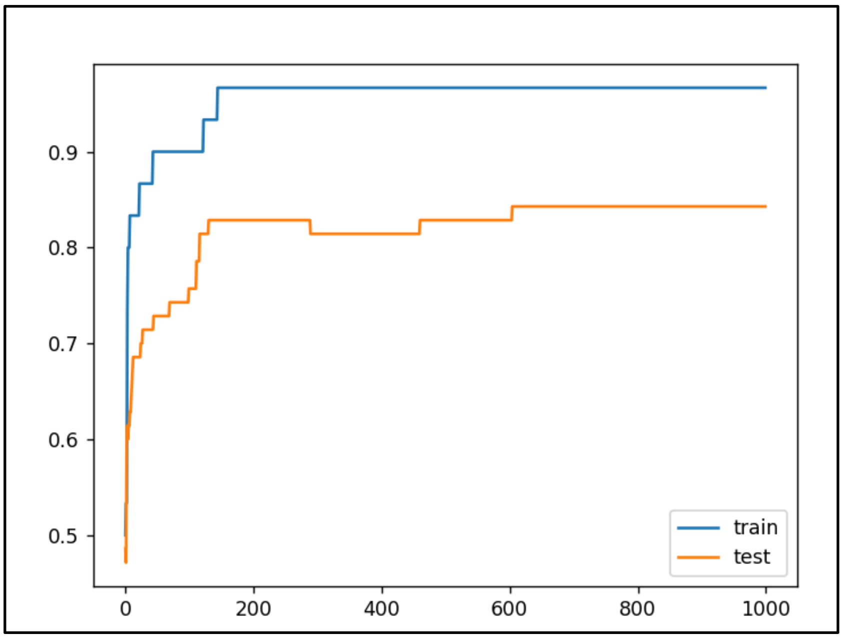
**OUTPUT:**



**Applying L1 And L2 Regularizer**

```
from matplotlib import pyplot
from sklearn.datasets import make_moons
from keras.models import Sequential
from keras.layers import Dense, Input
from keras.regularizers import l1_l2
X, Y = make_moons(n_samples=100, noise=0.2, random_state=1)
n_train = 30
trainX, testX = X[:n_train, :], X[n_train:]
trainY, testY = Y[:n_train], Y[n_train:]
model = Sequential()
model.add(Input(shape=(2,)))
model.add(Dense(500, activation='relu',kernel_regularizer=l1_l2(l1=0.001,l2=0.001)))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
history = model.fit(trainX, trainY, validation_data=(testX, testY), epochs=1000)
pyplot.plot(history.history['accuracy'], label='train')
pyplot.plot(history.history['val_accuracy'], label='test')
pyplot.legend()
pyplot.show()
```

**OUTPUT:**

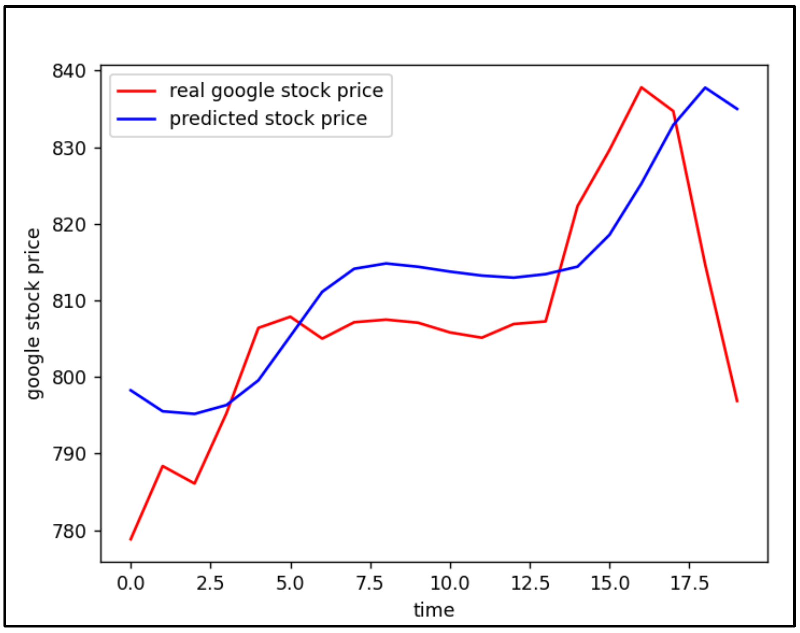


## PRACTICAL 7
## Demonstrate Recurrent Neural Network That Learns To Perform Sequence Analysis For Stock Price

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Dropout
from sklearn.preprocessing import MinMaxScaler
dataset_train=pd.read_csv('C:/Users/Vinish/OneDrive/Desktop/DL/Google_Stock_price_train.csv')
training_set=dataset_train.iloc[:,1:2].values
sc=MinMaxScaler(feature_range=(0,1))
training_set_scaled=sc.fit_transform(training_set)
X_train=[]
Y_train=[]
for i in range(60,1258):
    X_train.append(training_set_scaled[i-60:i,0])
    Y_train.append(training_set_scaled[i,0])
X_train,Y_train=np.array(X_train),np.array(Y_train)
print(X_train)
print('********************************************')
print(Y_train)
X_train=np.reshape(X_train,(X_train.shape[0],X_train.shape[1],1))
print('********************************************')
print(X_train)
regressor=Sequential()
regressor.add(LSTM(units=50,return_sequences=True,input_shape=(X_train.shape[1],1)))
regressor.add(Dropout(0.2))
regressor.add(LSTM(units=50,return_sequences=True))
regressor.add(Dropout(0.2))
regressor.add(LSTM(units=50,return_sequences=True))
regressor.add(Dropout(0.2))
regressor.add(LSTM(units=50))
regressor.add(Dropout(0.2))
regressor.add(Dense(units=1))
regressor.compile(optimizer='adam',loss='mean_squared_error')
regressor.fit(X_train,Y_train,epochs=100,batch_size=32)
dataset_test=pd.read_csv('C:/Users/Vinish/OneDrive/Desktop/DL/Google_Stock_Price_Test.csv')
real_stock_price=dataset_test.iloc[:,1:2].values
dataset_total=pd.concat((dataset_train['Open'],dataset_test['Open']),axis=0)
inputs=dataset_total[len(dataset_total)-len(dataset_test)-60:].values
inputs=inputs.reshape(-1,1)
inputs=sc.transform(inputs)
```

```
X_test=[]
for i in range(60,80):
    X_test.append(inputs[i-60:i,0])
X_test=np.array(X_test)
X_test=np.reshape(X_test,(X_test.shape[0],X_test.shape[1],1))
predicted_stock_price=regressor.predict(X_test)
predicted_stock_price=sc.inverse_transform(predicted_stock_price)
plt.plot(real_stock_price,color='red',label='real google stock price')
plt.plot(predicted_stock_price,color='blue',label='predicted stock price')
plt.xlabel('time')
plt.ylabel('google stock price')
plt.legend()
plt.show()
```

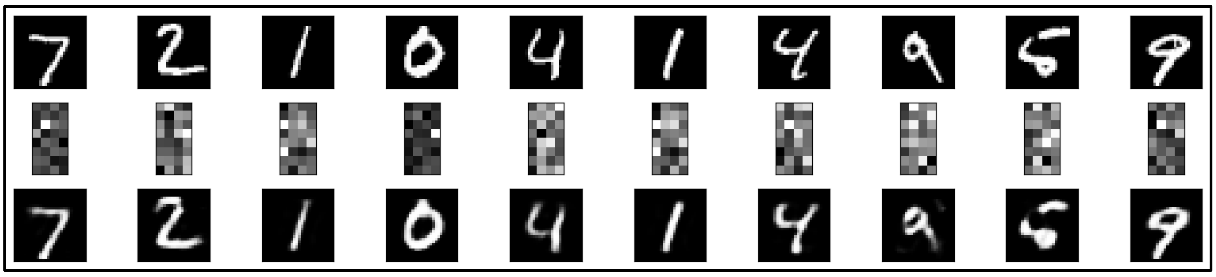
**OUTPUT:**



# PRACTICAL 8
## Performing Encoding And Decoding Of Images Using Deep Autoencoder

```
import keras
from keras import layers
from keras.datasets import mnist
import numpy as np
encoding_dim=32
input_img=keras.Input(shape=(784,))
encoded=layers.Dense(encoding_dim, activation='relu')(input_img)
decoded=layers.Dense(784, activation='sigmoid')(encoded)
autoencoder=keras.Model(input_img,decoded)
encoder=keras.Model(input_img,encoded)
encoded_input=keras.Input(shape=(encoding_dim,))
decoder_layer=autoencoder.layers[-1]
decoder=keras.Model(encoded_input,decoder_layer(encoded_input))
autoencoder.compile(optimizer='adam',loss='binary_crossentropy')
(X_train,_),(X_test,_)=mnist.load_data()
X_train=X_train.astype('float32')/255.
X_test=X_test.astype('float32')/255.
X_train=X_train.reshape((len(X_train),np.prod(X_train.shape[1:])))
X_test=X_test.reshape((len(X_test),np.prod(X_test.shape[1:])))
print(X_train.shape)
print(X_test.shape)
autoencoder.fit(X_train,X_train,
epochs=50,
batch_size=256,
shuffle=True,
validation_data=(X_test,X_test))
encoded_imgs=encoder.predict(X_test)
decoded_imgs=decoder.predict(encoded_imgs)
import matplotlib.pyplot as plt
n = 10
plt.figure(figsize=(40, 4))
for i in range(10):
    ax = plt.subplot(3, 20, i + 1)
    plt.imshow(X_test[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
    ax = plt.subplot(3, 20, i + 1 + 20)
    plt.imshow(encoded_imgs[i].reshape(8,4))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
    ax = plt.subplot(3, 20, 2*20 +i+ 1)
    plt.imshow(decoded_imgs[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
plt.show()
```

**OUTPUT:**

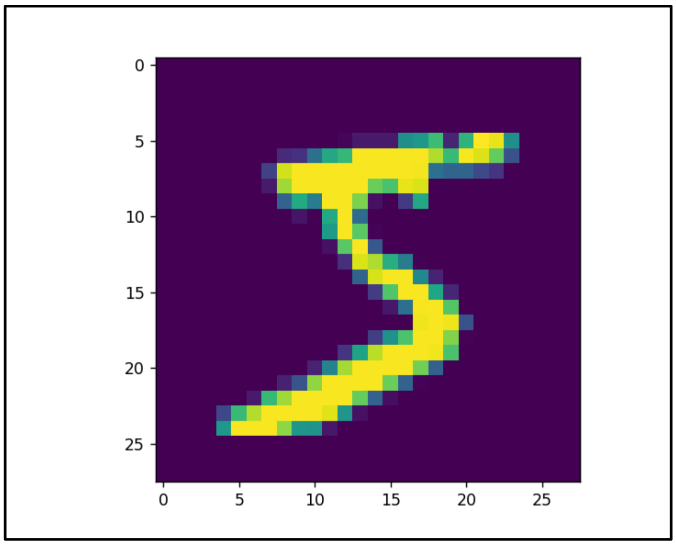


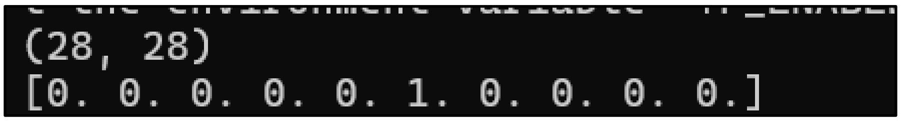
# PRACTICAL 9

## Implementation Of Convolutional Neural Network To Predict Numbers From Number Images

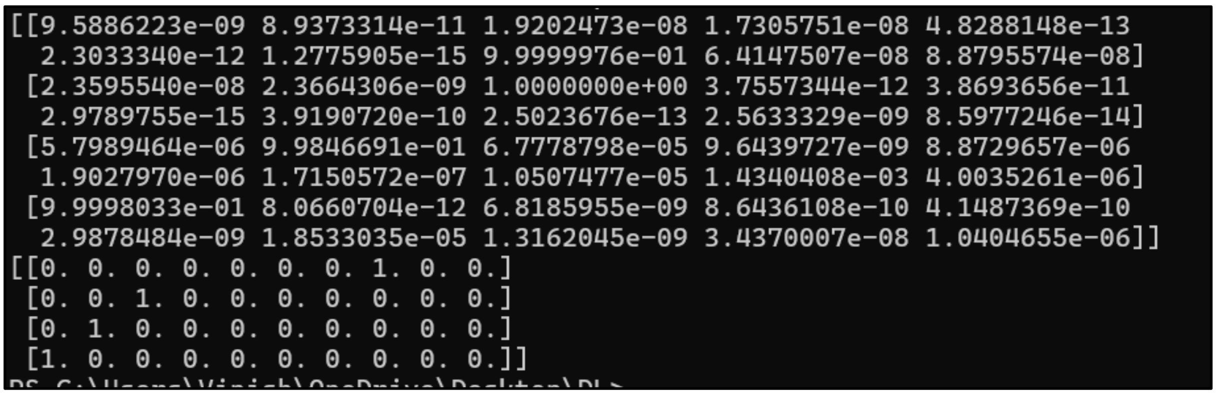
```
from keras.datasets import mnist
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Dense,Conv2D,Flatten
import matplotlib.pyplot as plt
(X_train,Y_train),(X_test,Y_test)=mnist.load_data()
plt.imshow(X_train[0])
plt.show()
print(X_train[0].shape)
X_train=X_train.reshape(60000,28,28,1)
X_test=X_test.reshape(10000,28,28,1)
Y_train=to_categorical(Y_train)
Y_test=to_categorical(Y_test)
Y_train[0]
print(Y_train[0])
model=Sequential()
model.add(Conv2D(64,kernel_size=3,activation='relu',input_shape=(28,28,1)))
model.add(Conv2D(32,kernel_size=3,activation='relu'))
model.add(Flatten())
model.add(Dense(10,activation='softmax'))
model.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accuracy'])
model.fit(X_train,Y_train,validation_data=(X_test,Y_test),epochs=3)
print(model.predict(X_test[:4]))
print(Y_test[:4])
```

**OUTPUT:**





```
(28, 28)
[0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
```



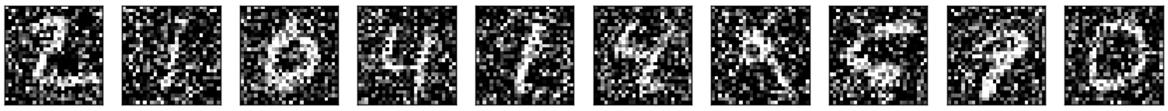
```
[[9.5886223e-09 8.9373314e-11 1.9202473e-08 1.7305751e-08 4.8288148e-13
  2.3033340e-12 1.2775905e-15 9.9999976e-01 6.4147507e-08 8.8795574e-08]
 [2.3595540e-08 2.3664306e-09 1.0000000e+00 3.7557344e-12 3.8693656e-11
  2.9789755e-15 3.9190720e-10 2.5023676e-13 2.5633329e-09 8.5977246e-14]
 [5.7989464e-06 9.9846691e-01 6.7778798e-05 9.6439727e-09 8.8729657e-06
  1.9027970e-06 1.7150572e-07 1.0507477e-05 1.4340408e-03 4.0035261e-06]
 [9.9998033e-01 8.0660704e-12 6.8185955e-09 8.6436108e-10 4.1487369e-10
  2.9878484e-09 1.8533035e-05 1.3162045e-09 3.4370007e-08 1.0404655e-06]]
[[0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]
 [0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
```

# PRACTICAL 10
# Denoising Of Images Using Autoencoder

```
import keras
from keras.datasets import mnist
from keras import layers
import numpy as np
from keras.callbacks import TensorBoard
import matplotlib.pyplot as plt
(X_train,_),(X_test,_)=mnist.load_data()
X_train=X_train.astype('float32')/255.
X_test=X_test.astype('float32')/255.
X_train=np.reshape(X_train,(len(X_train),28,28,1))
X_test=np.reshape(X_test,(len(X_test),28,28,1))
noise_factor=0.5
X_train_noisy=X_train+noise_factor*np.random.normal(loc=0.0,scale=1.0,size=X_train.shape)
X_test_noisy=X_test+noise_factor*np.random.normal(loc=0.0,scale=1.0,size=X_test.shape)
X_train_noisy=np.clip(X_train_noisy,0.,1.)
X_test_noisy=np.clip(X_test_noisy,0.,1.)
n=10
plt.figure(figsize=(20,2))
for i in range(1,n+1):
    ax=plt.subplot(1,n,i)
    plt.imshow(X_test_noisy[i].reshape(28,28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
plt.show()
input_img=keras.Input(shape=(28,28,1))
x=layers.Conv2D(32,(3,3),activation='relu',padding='same')(input_img)
x=layers.MaxPooling2D((2,2),padding='same')(x)
x=layers.Conv2D(32,(3,3),activation='relu',padding='same')(x)
encoded=layers.MaxPooling2D((2,2),padding='same')(x)
x=layers.Conv2D(32,(3,3),activation='relu',padding='same')(encoded)
x=layers.UpSampling2D((2,2))(x)
x=layers.Conv2D(32,(3,3),activation='relu',padding='same')(x)
x=layers.UpSampling2D((2,2))(x)
decoded=layers.Conv2D(1,(3,3),activation='sigmoid',padding='same')(x)
autoencoder=keras.Model(input_img,decoded)
autoencoder.compile(optimizer='adam',loss='binary_crossentropy')
autoencoder.fit(X_train_noisy,X_train,
epochs=3,
batch_size=128,
shuffle=True,
validation_data=(X_test_noisy,X_test),
callbacks=[TensorBoard(log_dir='/tmo/tb',histogram_freq=0,write_graph=False)])
```

```
predictions=autoencoder.predict(X_test_noisy)
m=10
plt.figure(figsize=(20,2))
for i in range(1,m+1):
    ax=plt.subplot(1,m,i)
    plt.imshow(predictions[i].reshape(28,28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
plt.show()
```

**OUTPUT:**



**After Epochs:**