DEPARTMENT OF INFORMATION TECHNOLOGY

SMT. PARMESHWARI DEVI DURGADUTT TIBREWALA

LIONS JUHU COLLEGE

OF ARTS, COMMERE AND SCIENCE

Affiliated to University of Mumbai

J.B. NAGAR, ANDHERI (E), MUMBAI-400059

Academic Year 2022-2023

For

Semester IV

<u>Submitted By:</u>

Tufail Shaikh

Msc.IT (Sem IV)

DEEP LEARNING

DEPARTMENT OF INFORMATION TECHNOLOGY

SMT. PARMESHWARIDEVI DURGADUTT TIBREWALA

LIONS JUHU COLLEGE

OF ARTS, COMMERE AND SCIENCE

Affiliated to University of Mumbai

J.B. NAGAR, ANDHERI (E), MUMBAI-400059

Academic Year 2022-2023

DEEP LEARNING

For

Semester IV

<u>Submitted By:</u>

Tufail Shaikh

Msc.IT (Sem IV)

SMT. PARMESHWARIDEVI DURGADUTT TIBREWALA

LIONS JUHU COLLEGE

OF ARTS, COMMERE AND SCIENCE

Affiliated to University of Mumbai

J.B. NAGAR, ANDHERI (E), MUMBAI-400059

DEPARTMENT OF INFORMATION TECHNOLOGY

Certificate of Approval

This is to certify that practical entitled **"Deep Learning"** Undertaken at **SMT.PARMESHWARIDEVI DURGADUTT TIBREWALA LIONS JUHU COLLEGEOF ARTS, COMMERECE & SCIENCE.** By **Tufail Shaikh** **Seat No.** ____________ in partial fulfilment of **M.Sc. (IT) (Semester IV)** Examination had not been submitted for any other examination and does not form of any other course undergone by the candidate. It is further certified that she has completed all required phases of the practical.

__________ __________

Internal Examiner **External Examiner**

__________ __________

HOD / In-Charge / Coordinator **Signature/**

Principal/Stamp

INDEX

Sr No.	Practical Name	Date	Sign
1. [A]	Perform basic mathematics operations in python. 1a) Add matrix using dot() 1b) Add matrix using add() 2a) Multiply using dot() 3a) Linear Combination 4a) Linear Equation 5a) Norm one-dimensional 5b) Norm two-dimensional 5c) Norm three-dimensional 5d) Norm 6a) Symmetric Matrix		
[B]	Performing matrix multiplication and finding eigen vectors and eigen values using TensorFlow.		
2.	Solving XOR problem using deep feed forward network.		
3.	Implementing deep neural network for performing binary classification task.		
4.[A]	Using a deep field forward network with two hidden layers for performing classification and predicting the probability of class.		

B]	Using a deep field forward network with two hidden layers for performing classification and predicting the probability of class.		
C]	Using a deep field forward network with two hidden layers for performing linear regression and predicting values.		
5.[A]	Evaluating feed forward deep network for regression using K Fold cross validation.		
B]	Evaluating feed forward deep network for multiclass Classification using K Fold cross validation.		
6.[A]	Implementing regularization to avoid overfitting in binary classification.		
[B]	Implementing L2 regularization		
[C]	Replacing L2 regularizer with L1 regularizer.		
7.	Demonstrate recurrent neural network that learns to perform sequence analysis for stock price.		
8.	Performing encoding and decoding of images using deep autoencoder.		
9.	Implementation of convolutional neural network to predict numbers from number images		
10.	Denoising of images using autoencoder.		

PRACTICAL NO.:1

[A] Aim: Perform basic mathematics operations in python

[1a] Add matrix using dot()

Source Code:

```
import numpy as np
A = np.array([[1, 2], [3, 4], [5, 6]])
A
B = np.array([[2, 5], [7, 4], [4, 3]])
B
# Add matrices A and B
C = A + B
print(C)
```

Output

```
============= RESTART: C:/Users/COMP/Desktop/Uzma Khan/DL prct1.py =============
[[ 3  7]
 [10  8]
 [ 9  9]]
>>>
```

[1b] Add matrix using add()

Source Code:

```
import numpy as np
A = np.array([[1, 2], [3, 4], [5, 6]])
B = np.array([[2, 5], [7, 4], [4, 3]])
# Add matrices A and B C
= np.add(A, B) print(C)
```

Output

```
============= RESTART: C:/Users/COMP/Desktop/Uzma Khan/DL prct1.py ===
[[ 3  7]
 [10  8]
 [ 9  9]]
>>>
```

[2a] Multiply using dot()

Source Code:

```
# importing the module
import numpy as np

# creating two matrices
p = [[1, 2], [2, 3], [4, 5]]
q = [[4, 5, 1], [6, 7, 2]]
print("Matrix p :").
print(p) print("Matrix q
:") print(q)

# computing product result
= np.dot(p, q)

# printing the result
print("The matrix multiplication is :")
print(result)
```

Output

```
============= RESTART: C:/Users/COMP/Desktop/Uzma Khan/DL prct1.py =
Matrix p :
[[1, 2], [2, 3], [4, 5]]
Matrix q :
[[4, 5, 1], [6, 7, 2]]
The matrix multiplication is :
[[16 19  5]
 [26 31  8]
 [46 55 14]]
```

[2b] Multiply using dot()

Source Code:

```
import numpy as np
A = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12]]) print(A)
B = np.array([[2, 7], [1, 2], [3, 6]])
print(B) C =
A.dot(B)
print(C)
```

Output:

```
============= RESTART: C:/Users/COMP/Desktop/Uzma Khan/DL prct1.py ======
[[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 12]]
[[2 7]
 [1 2]
 [3 6]]
[[ 13  29]
 [ 31  74]
 [ 49 119]
 [ 67 164]]
>>
```

[3a] Linear Combination

Source Code:

```
import numpy as np
x = np.array([[0, 1, 1],
              [1, 1, 0],
              [1, 0, 1]])
y = ([3.65, 1.55, 3.42]) scalars = np.linalg.solve(x, y)
   print(scalars)
```

Output

```
============== RESTART: C:/Users/COMP/Desktop/Uzma Khan/DL prct1.py ==============
[0.66 0.89 2.76]
>>|
```

[3b] Linear Combination Source

Code:

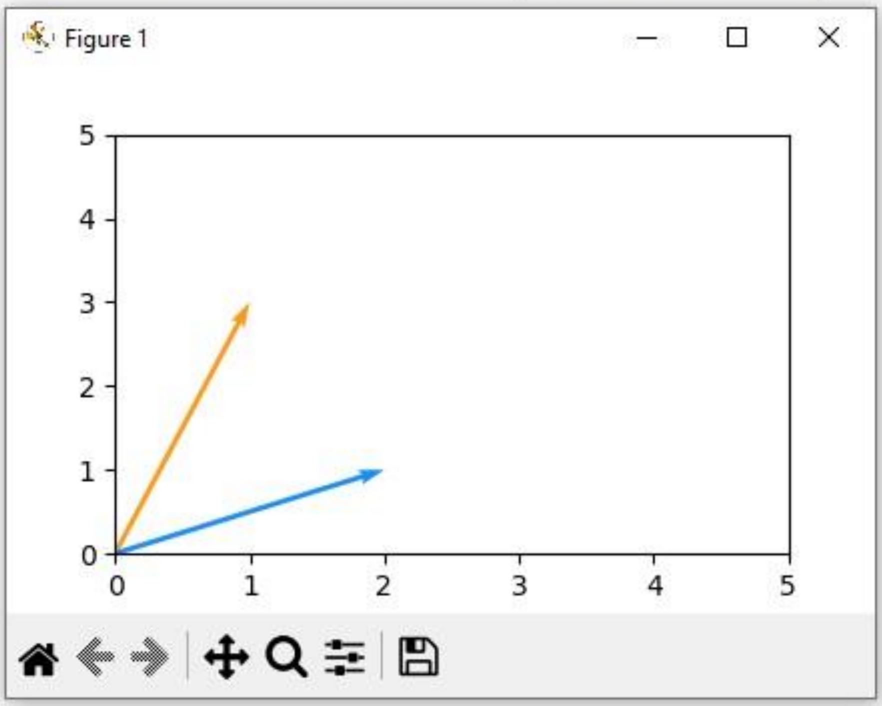
```
import numpy as np
import matplotlib.pyplot as plt

def plotVectors(vecs, cols, alpha =1 ):
    plt.figure()     plt.axvline(x=0, color =
'#A9A9A9', zorder = 0)     for i in
```

```
range(len(vecs)):        x =
np.concatenate([[0,0],vecs[i]])
        plt.quiver([x[0]],
               [x[1]],
               [x[2]],
               [x[3]],
               angles='xy', scale_units='xy', scale=1,color=cols[i], alpha=alpha)
orange= '#FF9A13' blue= '#1190FF'
plotVectors([[1,3],[2,1]],[orange,blue])
plt.xlim(0,5) plt.ylim(0,5)
plt.show()
```

Output

```
= RESTART: C:/Users/COMP/Desktop/Uzma Khan/DL prct1.py =
```



[4a] Linear Equation

Source Code:

```
import numpy as np
A = np.array([[20, 10], [17, 22]])
```

```
B = np.array([350, 500]) R = np.linalg.solve(A,B) x, y =
    np.linalg.solve(A,B) print(R) print("x =", x) print("y =", y)
```

Output

```
============= RESTART: C:/Users/COMP/Desktop/Uzma Khan/DL prct1.py ==
[10. 15.]
x = 10.000000000000002
y = 14.999999999999996
```

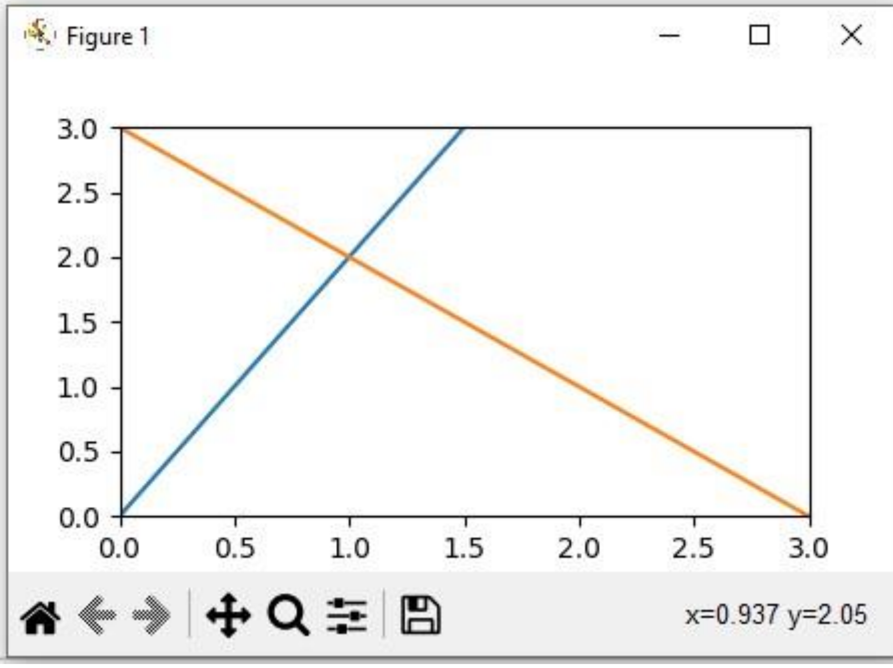
[4b] Linear Equation

Source Code:

```
import numpy as np import
matplotlib.pyplot as plt x =
np.arange(-10,10) y = 2*x y1
= -x + 3 plt.figure()
plt.plot(x,y) plt.plot(x,y1)
plt.xlim(0,3) plt.ylim(0,3)
plt.axvline(x=0, color='grey') plt.show()
```

Output

```
= RESTART: C:/Users/COMP/Desktop/Uzma Khan/DL prct1.py =
```



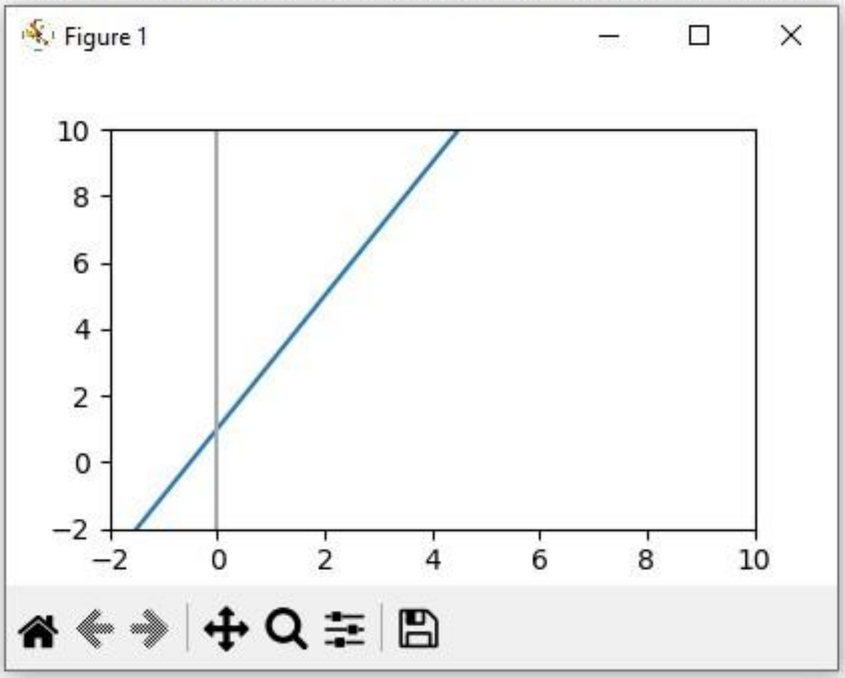
[4c] Linear Equation

Source Code:

import numpy as np import
matplotlib.pyplot as plt x =
np.arange(-10,10) y = 2*x + 1
plt.figure() plt.plot(x,y)
plt.xlim(-2,10) plt.ylim(-2,10)
plt.axvline(x=0, color = '#A9A9A9') plt.show()

Output

```
= RESTART: C:/Users/COMP/Desktop/Uzma Khan/DL prct1.py
```



[5a] Norm one-dimensional

Source Code:

 # import library import numpy as
np *# initialize vector* oned =
np.arange(10) *# compute norm of*
vector manh_norm =
np.linalg.norm(oned)
print("Manhattan norm:")
print(manh_norm)

Output

```
============= RESTART: C:\Users\COMP\Desktop\Uzma Khan\DL prct1.py =
Manhattan norm:
16.881943016134134
>>> |
```

[5b] Norm one-dimensional

Source Code:

numpy as np
import matplotlib.pyplot as plt

u = np.array([1,6])
print(u) v =
np.array([4,2])
print(v)

manhatan_norm = np.linalg.norm(u+v)
print("Manhattan Norm") print(manhatan_norm)import

Output

```
============= RESTART: C:\Users\COMP\Desktop\Uzma Khan\DL prct1.py ======
[1 6]
[4 2]
Manhattan Norm
9.433981132056603
```

[5c] Norm two-dimensional

Source Code:

import library import
numpy as np # *initialize*
matrix twod = np.array([[
1, 2, 3],
        [4, 5, 6]])
compute norm of matrix eucl_norm
= np.linalg.norm(twod)
print("Euclidean norm:")
print(eucl_norm)

Output

```
============= RESTART: C:\Users\COMP\Desktop\Uzma Khan\DL prct1.py ===
Euclidean norm:
9.539392014169456
>
```

[5d] Norm three-dimensional

Source Code:

```
# import library import
numpy as np # initialize
matrix threed = np.array([[[
1, 2, 3],
        [4, 5, 6]],[[ 11, 12, 13],
        [14, 15, 16]]])
# compute norm of matrix mink_norm
= np.linalg.norm(threed)
print("Minkowski norm:")
print(mink_norm)
```

Output

```
============= RESTART: C:\Users\COMP\Desktop\Uzma Khan\DL prct1.py ==
Minkowski norm:
34.66987164671943
>>>
```

[5e] Norm

Source Code:

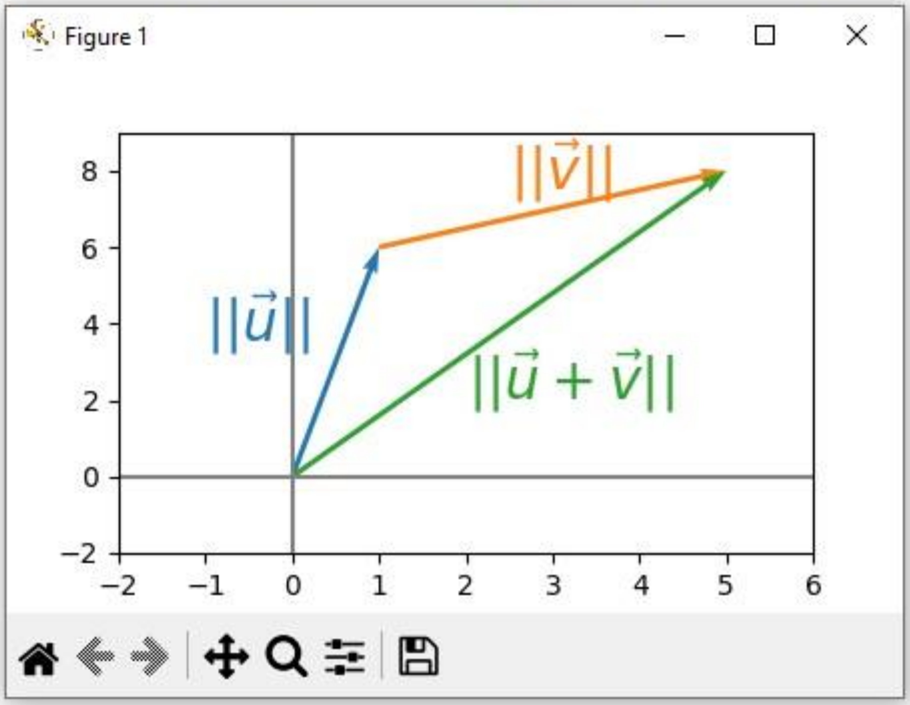
```
import numpy as np import
matplotlib.pyplot as plt import
seaborn as sns

u = [0,0,1,6] v = [0,0,4,2] u_bis
= [1,6,v[2],v[3]] w = [0,0,5,8]
plt.quiver([u[0], u_bis[0], w[0]],
[u[1], u_bis[1], w[1]],
         [u[2], u_bis[2], w[2]],
         [u[3], u_bis[3], w[3]],
         angles = 'xy', scale_units = 'xy', scale = 1, color = sns.color_palette())

plt.xlim(-2,6) plt.ylim(-2,9) plt.axvline(x=0, color='grey') plt.axhline(y=0,
color='grey') plt.text(-1, 3.5, r'$||\vec{u}||$', color = sns.color_palette()[0],
size =20) plt.text(2.5, 7.5, r'$||\vec{v}||$', color = sns.color_palette()[1], size
=20) plt.text(2, 2, r'$||\vec{u}+\vec{v}||$', color = sns.color_palette()[2],
size =20) plt.show()
```

Output

```
RESTART: C:\Users\COMP\Desktop\Uzma Khan\DL prct1.py =
```



[6a] Symmetric Matrix

Source Code:

 # Linear Algebra Learning Sequence
 # Transpose using different Method

import numpy as np

M = np.array([[2,3,4], [3,45,8], [4,8,78]]) print("---Matrix M---\n", M)

 # Transposing the Matrix M
print('\n\nTranspose as M.T----\n', M.T)

if M.T.all() == M.all(): print("--------> Transpose is equal to M!! -----> It is a Symmetric Matrix") else:
print("---------> Transpose is not equal o M!! ------> It is not a Symmetric Matrix")

Output

```
============= RESTART: C:\Users\COMP\Desktop\Uzma Khan\DL prct1.py ===========
---Matrix M---
 [[ 2  3  4]
 [ 3 45  8]
 [ 4  8 78]]


Transpose as M.T----
 [[ 2  3  4]
 [ 3 45  8]
 [ 4  8 78]]
--------> Transpose is equal to M!! -----> It is a Symmetric Matrix
```

[6b] Symmetric Matrix

Source Code:

```
import numpy as np
A = np.array([[2,4,-1],[4,-8,0],[-1,0,3]])
print(A) print(A.T)
```

Output

```
[[ 2  4 -1]
 [ 4 -8  0]
 [-1  0  3]]
[[ 2  4 -1]
 [ 4 -8  0]
 [-1  0  3]]
>>>
```

[B] Aim: Performing matrix multiplication and finding eigen vectors and eigen values using TensorFlow.

Source Code:

```
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)
e_matrix_A=tf.random.uniform([2,2],minval=3,maxval=10,dtype=tf.float32,name="matrixA")
print("Matrix A:\n{}\n\n".format(e_matrix_A))
eigen_values_A,eigen_vectors_A=tf.linalg.eigh(e_matrix_A)
print("Eigen Vectors:\n{}\n\nEigen Values:\n{}\n".format(eigen_vectors_A,eigen_values_A))
```

Output

```
>>>
========= RESTART: C:/Users/COMP/Desktop/DL/6 bbbb usir

Warning (from warnings module):
  File "C:\Users\COMP\AppData\Local\Programs\Python\Pyt
    warnings.warn("loaded more than 1 DLL from .libs:"
UserWarning: loaded more than 1 DLL from .libs:
C:\Users\COMP\AppData\Local\Programs\Python\Python310\l
gfortran-win_amd64.dll
C:\Users\COMP\AppData\Local\Programs\Python\Python310\l
gfortran-win_amd64.dll
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:
[[3.0645185 3.9833846]
 [5.2984214 7.856101 ]]


Eigen Vectors:
[[-0.84024066 -0.5422136 ]
 [ 0.5422136  -0.84024066]]

Eigen Values:
[-0.35459203 11.275211  ]
>>> |
```

PRACTICAL NO.:2

Aim: Solving XOR problem using deep feed forward network.

Source Code 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=1000, batch_size=4)
print(model.get_weights()) print(model.predict(X, batch_size=4))

Output

```
====== RESTART: C:/Users/COMP/Desktop/DL/XOR Deep feed forward network.py ======

Warning (from warnings module):
  File "C:\Users\COMP\AppData\Local\Programs\Python\Python310\lib\site-packages\numpy\_distributor_init.p
    warnings.warn("loaded more than 1 DLL from .libs:"
UserWarning: loaded more than 1 DLL from .libs:
C:\Users\COMP\AppData\Local\Programs\Python\Python310\lib\site-packages\numpy\.libs\libopenblas.EL2C6PLE4
gfortran-win_amd64.dll
C:\Users\COMP\AppData\Local\Programs\Python\Python310\lib\site-packages\numpy\.libs\libopenblas.FB5AE2TYX
gfortran-win_amd64.dll
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 2)                 6

 dense_1 (Dense)             (None, 1)                 3

=================================================================
Total params: 9
Trainable params: 9
Non-trainable params: 0
_________________________________________________________________
None
[array([[-0.56529707,  0.02641189],
       [ 0.6831105 ,  0.35668623]], dtype=float32), array([0., 0.], dtype=float32), array([[0.6233808],
       [0.4972006]], dtype=float32), array([0.], dtype=float32)]
Epoch 1/1000
1/1 [==============================] - ETA: 0s - loss: 0.6625 - accuracy: 0.7500
1/1 [==============================] - 2s 2s/step - loss: 0.6625 - accuracy: 0.7500
Epoch 2/1000
1/1 [==============================] - ETA: 0s - loss: 0.6622 - accuracy: 0.7500
1/1 [==============================] - 0s 18ms/step - loss: 0.6622 - accuracy: 0.7500
Epoch 3/1000
Epoch 992/1000
1/1 [==============================] - ETA: 0s - loss: 0.5084 - accuracy: 0.7500
1/1 [==============================] - 0s 9ms/step - loss: 0.5084 - accuracy: 0.7500
Epoch 993/1000
1/1 [==============================] - ETA: 0s - loss: 0.5084 - accuracy: 0.7500
1/1 [==============================] - 0s 10ms/step - loss: 0.5084 - accuracy: 0.7500
Epoch 994/1000
1/1 [==============================] - ETA: 0s - loss: 0.5083 - accuracy: 0.7500
1/1 [==============================] - 0s 8ms/step - loss: 0.5083 - accuracy: 0.7500
Epoch 995/1000
1/1 [==============================] - ETA: 0s - loss: 0.5082 - accuracy: 0.7500
1/1 [==============================] - 0s 8ms/step - loss: 0.5082 - accuracy: 0.7500
Epoch 996/1000
1/1 [==============================] - ETA: 0s - loss: 0.5082 - accuracy: 0.7500
1/1 [==============================] - 0s 8ms/step - loss: 0.5082 - accuracy: 0.7500
Epoch 997/1000
1/1 [==============================] - ETA: 0s - loss: 0.5081 - accuracy: 0.7500
1/1 [==============================] - 0s 7ms/step - loss: 0.5081 - accuracy: 0.7500
Epoch 998/1000
1/1 [==============================] - ETA: 0s - loss: 0.5080 - accuracy: 0.7500
1/1 [==============================] - 0s 7ms/step - loss: 0.5080 - accuracy: 0.7500
Epoch 999/1000
1/1 [==============================] - ETA: 0s - loss: 0.5080 - accuracy: 0.7500
1/1 [==============================] - 0s 7ms/step - loss: 0.5080 - accuracy: 0.7500
Epoch 1000/1000
1/1 [==============================] - ETA: 0s - loss: 0.5079 - accuracy: 0.7500
1/1 [==============================] - 0s 6ms/step - loss: 0.5079 - accuracy: 0.7500
[array([[-1.114009 , -0.6541138],
       [ 1.1138123,  0.6546648]], dtype=float32), array([ 3.3032193e-06, -5.9689838e-04], dtype=float32), array([[1.5167154],
       [1.4778702]], dtype=float32), array([-0.5182931], dtype=float32)]
1/1 [==============================] - ETA: 0s1/1 [==============================
] - 0s 70ms/step
[[0.37325263]
 [0.8945115 ]
 [0.37325144]
 [0.37325144]]
>>>
```

PRACTICAL NO.:3

Aim: Implementing deep neural network for performing binary classification task.

Source Code from numpy import
loadtxt from keras.models import
Sequential from keras.layers import
Dense

```
dataset = loadtxt("C:/Users/Hajra Khan/Desktop/HAJRA/DL Practical/csv files/pima-
indiansdiabetes.csv" , delimiter = ',')
X=dataset[:,0:8] Y=dataset[:,8]
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 =150,batch_size=10) accuracy=model.evaluate(X,Y)
print('Accuracy of model is', (accuracy*100)) prediction=model.predict(X)
exec("for i in range(5):print(X[i].tolist(),prediction[i],Y[i])")
```

Output

```
Epoch 1/150
77/77 [==============================] - 2s 3ms/step - loss: 2.5263 - accuracy: 0.5586
Epoch 2/150
77/77 [==============================] - 0s 3ms/step - loss: 1.1873 - accuracy: 0.5885
Epoch 3/150
77/77 [==============================] - 0s 3ms/step - loss: 1.0318 - accuracy: 0.5443
Epoch 4/150
77/77 [==============================] - 0s 3ms/step - loss: 0.9486 - accuracy: 0.5404
Epoch 5/150
77/77 [==============================] - 0s 3ms/step - loss: 0.8373 - accuracy: 0.5651
Epoch 6/150
77/77 [==============================] - 0s 3ms/step - loss: 0.7979 - accuracy: 0.5794
Epoch 7/150
77/77 [==============================] - 0s 3ms/step - loss: 0.7693 - accuracy: 0.5951
Epoch 8/150
77/77 [==============================] - 0s 3ms/step - loss: 0.7384 - accuracy: 0.6094
Epoch 9/150
77/77 [==============================] - 0s 3ms/step - loss: 0.7010 - accuracy: 0.6289
Epoch 10/150
77/77 [==============================] - 0s 4ms/step - loss: 0.6959 - accuracy: 0.6328
Epoch 11/150
77/77 [==============================] - 0s 5ms/step - loss: 0.6670 - accuracy: 0.6523
```

```
Accuracy of model is [0.479278564453125, 0.7734375, 0.479278564453125, 0.7734375, 0.479278564453125, 0.7734375, 0.479278564453125, 0.7734375,
0.479278564453125, 0.7734375, 0.479278564453125, 0.7734375, 0.479278564453125, 0.7734375, 0.479278564453125, 0.7734375, 0.479278564453125,
0.7734375, 0.479278564453125, 0.7734375, 0.479278564453125, 0.7734375, 0.479278564453125, 0.7734375, 0.479278564453125, 0.7734375, 0
.479278564453125, 0.7734375, 0.479278564453125, 0.7734375, 0.479278564453125, 0.7734375, 0.479278564453125, 0.7734375, 0.479278564453125,
0.7734375, 0.479278564453125, 0.7734375, 0.479278564453125, 0.7734375, 0.479278564453125, 0.7734375, 0.479278564453125, 0.7734375, 0
.479278564453125, 0.7734375, 0.479278564453125, 0.7734375, 0.479278564453125, 0.7734375, 0.479278564453125, 0.7734375, 0.479278564453125,
0.7734375, 0.479278564453125, 0.7734375, 0.479278564453125, 0.7734375, 0.479278564453125, 0.7734375, 0.479278564453125, 0.7734375, 0
.479278564453125, 0.7734375, 0.479278564453125, 0.7734375, 0.479278564453125, 0.7734375, 0.479278564453125, 0.7734375, 0.479278564453125,
0.7734375, 0.479278564453125, 0.7734375, 0.479278564453125, 0.7734375, 0.479278564453125, 0.7734375, 0.479278564453125, 0.7734375, 0
.479278564453125, 0.7734375, 0.479278564453125, 0.7734375, 0.479278564453125, 0.7734375, 0.479278564453125, 0.7734375, 0.479278564453125,
0.7734375, 0.479278564453125, 0.7734375, 0.479278564453125, 0.7734375, 0.479278564453125, 0.7734375, 0.479278564453125, 0.7734375, 0
.479278564453125, 0.7734375, 0.479278564453125, 0.7734375, 0.479278564453125, 0.7734375, 0.479278564453125, 0.7734375, 0.479278564453125,
0.7734375, 0.479278564453125, 0.7734375, 0.479278564453125, 0.7734375, 0.479278564453125, 0.7734375, 0.479278564453125, 0.7734375, 0
.479278564453125, 0.7734375, 0.479278564453125, 0.7734375, 0.479278564453125, 0.7734375, 0.479278564453125, 0.7734375, 0.479278564453125,
0.7734375, 0.479278564453125, 0.7734375, 0.479278564453125, 0.7734375, 0.479278564453125, 0.7734375, 0.479278564453125, 0.7734375, 0
.479278564453125, 0.7734375, 0.479278564453125, 0.7734375, 0.479278564453125, 0.7734375, 0.479278564453125, 0.7734375, 0.479278564453125,
0.7734375, 0.479278564453125, 0.7734375, 0.479278564453125, 0.7734375, 0.479278564453125, 0.7734375, 0.479278564453125, 0.7734375, 0
.479278564453125, 0.7734375, 0.479278564453125, 0.7734375, 0.479278564453125, 0.7734375, 0.479278564453125, 0.7734375, 0.479278564453125,
0.7734375, 0.479278564453125, 0.7734375, 0.479278564453125, 0.7734375, 0.479278564453125, 0.7734375, 0.479278564453125, 0.7734375, 0
.479278564453125, 0.7734375, 0.479278564453125, 0.7734375, 0.479278564453125, 0.7734375, 0.479278564453125, 0.7734375, 0.479278564453125,
0.7734375, 0.479278564453125, 0.7734375, 0.479278564453125, 0.7734375, 0.479278564453125, 0.7734375, 0.479278564453125, 0.7734375, 0
.479278564453125, 0.7734375, 0.479278564453125, 0.7734375, 0.479278564453125, 0.7734375, 0.479278564453125, 0.7734375, 0.479278564453125,
```

```
24/24 [==============================] - 0s 2ms/step
[6.0, 148.0, 72.0, 35.0, 0.0, 33.6, 0.627, 50.0] [0.73097056] 1.0
[1.0, 85.0, 66.0, 29.0, 0.0, 26.6, 0.351, 31.0] [0.11377989] 0.0
[8.0, 183.0, 64.0, 0.0, 0.0, 23.3, 0.672, 32.0] [0.87028414] 1.0
[1.0, 89.0, 66.0, 23.0, 94.0, 28.1, 0.167, 21.0] [0.08588263] 0.0
[0.0, 137.0, 40.0, 35.0, 168.0, 43.1, 2.288, 33.0] [0.66002005] 1.0
```

PRACTICAL NO.:4

[A] Aim: Using a deep field forward network with two hidden layers for performing classification and predicting the probability of class.

Source Code import numpy as np
from sklearn.datasets import
load_iris from keras.models import
Sequential from keras.layers import
Dense from keras.utils import
to_categorical
from sklearn.model_selection import train_test_split
 # Load the iris dataset
iris = load_iris() X =
iris.data y = iris.target
 # Convert target variable to one-hot encoded format y
= to_categorical(y)
 # Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
 # Define the model model
= Sequential()
model.add(Dense(64, input_dim=X_train.shape[1], activation='relu')) model.add(Dense(32, activation='relu'))
model.add(Dense(y_train.shape[1], activation='softmax'))
 # Compile the model
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
 # Train the model
model.fit(X_train, y_train, epochs=100, batch_size=32, validation_split=0.2)
 # Evaluate the model on the test set loss,
accuracy = model.evaluate(X_test, y_test)
print('Test accuracy:', accuracy)

Output

```
============= RESTART: C:/Users/COMP/Desktop/Uzma Khan/DL prct4A.py ============
Epoch 1/100
1/3 [=========>....................] - ETA: 1s - loss: 1.2028 - accuracy: 0.343
8
2/3 [===================>..........] - ETA: 0s - loss: 1.2237 - accuracy: 0.29
69
3/3 [==============================] - 1s 129ms/step - loss: 1.2193 - accurac
y: 0.2812 - val_loss: 0.9872 - val_accuracy: 0.5000
Epoch 2/100
1/3 [=========>....................] - ETA: 0s - loss: 1.1924 - accuracy: 0.187
5
3/3 [==============================] - 0s 13ms/step - loss: 1.1204 - accuracy:
 0.2812 - val_loss: 0.9641 - val_accuracy: 0.5000
Epoch 3/100
1/3 [=========>....................] - ETA: 0s - loss: 1.0407 - accuracy: 0.375
0
3/3 [==============================] - 0s 13ms/step - loss: 1.0354 - accuracy:
 0.3854 - val_loss: 0.9387 - val_accuracy: 0.7500
Epoch 4/100
1/3 [=========>....................] - ETA: 0s - loss: 0.9797 - accuracy: 0.656
2
3/3 [==============================] - 0s 13ms/step - loss: 0.9769 - accuracy:
 0.5312 - val_loss: 0.9154 - val_accuracy: 0.4167
Epoch 5/100
1/3 [=========>....................] - ETA: 0s - loss: 0.9415 - accuracy: 0.343
8
3/3 [==============================] - 0s 14ms/step - loss: 0.9235 - accuracy:
 0.4271 - val_loss: 0.8746 - val_accuracy: 0.5833
Epoch 6/100
1/3 [=========>....................] - ETA: 0s - loss: 0.8772 - accuracy: 0.656
2
3/3 [==============================] - 0s 14ms/step - loss: 0.8719 - accuracy:
```

```
Epoch 95/100
1/3 [=========>....................] - ETA: 0s - loss: 0.1519 - accuracy: 1.000
0
3/3 [==============================] - 0s 17ms/step - loss: 0.1340 - accuracy:
 0.9792 - val_loss: 0.1088 - val_accuracy: 1.0000
Epoch 96/100
1/3 [=========>....................] - ETA: 0s - loss: 0.1122 - accuracy: 1.000
0
3/3 [==============================] - 0s 17ms/step - loss: 0.1316 - accuracy:
 0.9792 - val_loss: 0.1092 - val_accuracy: 1.0000
Epoch 97/100
1/3 [=========>....................] - ETA: 0s - loss: 0.1562 - accuracy: 0.96 8
8
3/3 [==============================] - 0s 18ms/step - loss: 0.1328 - accuracy:
 0.9688 - val_loss: 0.1166 - val_accuracy: 1.0000
Epoch 98/100
1/3 [=========>....................] - ETA: 0s - loss: 0.1571 - accuracy: 0.937
5
3/3 [==============================] - 0s 18ms/step - loss: 0.1313 - accuracy:
 0.9583 - val_loss: 0.0986 - val_accuracy: 1.0000
Epoch 99/100
1/3 [=========>....................] - ETA: 0s - loss: 0.1047 - accuracy: 1.000
0
3/3 [==============================] - 0s 17ms/step - loss: 0.1315 - accuracy:
 0.9792 - val_loss: 0.0908 - val_accuracy: 1.0000
Epoch 100/100
1/3 [=========>....................] - ETA: 0s - loss: 0.1302 - accuracy: 1.000
0
3/3 [==============================] - 0s 16ms/step - loss: 0.1272 - accuracy:
 0.9792 - val_loss: 0.1034 - val_accuracy: 1.0000
1/1 [==============================] - ETA: 0s - loss: 0.1434 - accuracy: 1.000
0
1/1 [==============================] - 0s 73ms/step - loss: 0.1434 - accuracy:
 1.0000
Test accuracy: 1.0
>>
```

[B] Aim: Using a deep field forward network with two hidden layers for performing classification and predicting the probability of class.

Source Code

```
from tensorflow import keras 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) models=keras.Sequential()

models.add(Dense(4,input_dim=2,activation='relu'))

models.add(Dense(4,activation='relu'))

models.add(Dense(1,activation='sigmoid'))

models.compile(loss='binary_crossentropy',optimizer='adam')

models.fit(X,Y,epochs=500) import numpy as np

Xnew,Yreal=make_blobs(n_samples=3,centers=2,n_features=2,random_state=1)

Xnew=scalar.transform(Xnew)

Yclass=np.argmax(models.predict(Xnew), axis=-1)

Ynew=models.predict(Xnew) for i in

range(len(Xnew)):

  print("X=%s,Predicted_probability=%s,Predicted_class=%s"%(Xnew[i],Ynew[i],Yclass[i]))
```

Output

IDLE Shell 3.11.2

File Edit Shell Debug Options Window Help

```
Python 3.11.2 (tags/v3.11.2:878ead1, Feb  7 2023, 16:38:35) [MSC v.1934 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: C:/Users/COMP/Desktop/Uzma Khan/DL prct4B.py ============
Epoch 1/500
1/4 [======>.......................] - ETA: 4s - loss: 0.67252/4 [
==============>...............] - ETA: 0s - loss: 0.66974/4 [======
========================] - 2s 29ms/step - loss: 0.6726
Epoch 2/500
1/4 [======>.......................] - ETA: 0s - loss: 0.66684/4 [
==============================] - 0s 3ms/step - loss: 0.6712
Epoch 3/500
1/4 [======>.......................] - ETA: 0s - loss: 0.67314/4 [
==============================] - 0s 2ms/step - loss: 0.6697
Epoch 4/500
1/4 [======>.......................] - ETA: 0s - loss: 0.66394/4 [
==============================] - 0s 3ms/step - loss: 0.6685
Epoch 5/500
1/4 [======>.......................] - ETA: 0s - loss: 0.67004/4 [
==============================] - 0s 2ms/step - loss: 0.6670
Epoch 6/500
1/4 [======>.......................] - ETA: 0s - loss: 0.66814/4 [
==============================] - 0s 2ms/step - loss: 0.6659
Epoch 7/500
1/4 [======>.......................] - ETA: 0s - loss: 0.67114/4 [
==============================] - 0s 2ms/step - loss: 0.6646
Epoch 8/500
1/4 [======>.......................] - ETA: 0s - loss: 0.66684/4 [
==============================] - 0s 2ms/step - loss: 0.6632
Epoch 9/500
1/4 [======>.......................] - ETA: 0s - loss: 0.66554/4 [
==============================] - 0s 2ms/step - loss: 0.6619
Epoch 10/500
1/4 [======>.......................] - ETA: 0s - loss: 0.67144/4 [
==============================] - 0s 3ms/step - loss: 0.6605
Epoch 11/500
1/4 [======>.......................] - ETA: 0s - loss: 0.65124/4 [
```

```
1/4 [======>.......................] - ETA: 0s - loss: 0.00364/4 [
==============================] - 0s 2ms/step - loss: 0.0036
Epoch 492/500
1/4 [======>.......................] - ETA: 0s - loss: 0.00364/4 [
==============================] - 0s 2ms/step - loss: 0.0035
Epoch 493/500
1/4 [======>.......................] - ETA: 0s - loss: 0.00314/4 [
==============================] - 0s 2ms/step - loss: 0.0035
Epoch 494/500
1/4 [======>.......................] - ETA: 0s - loss: 0.00374/4
==============================] - 0s 2ms/step - loss: 0.0035
Epoch 495/500
1/4 [======>.......................] - ETA: 0s - loss: 0.00354/4 [
==============================] - 0s 2ms/step - loss: 0.0035
Epoch 496/500
1/4 [======>.......................] - ETA: 0s - loss: 0.00294/4 [
==============================] - 0s 3ms/step - loss: 0.0035
Epoch 497/500
1/4 [======>.......................] - ETA: 0s - loss: 0.00274/4 [
==============================] - 0s 3ms/step - loss: 0.0034
Epoch 498/500
1/4 [======>.......................] - ETA: 0s - loss: 0.00414/4 [
==============================] - 0s 2ms/step - loss: 0.0034
Epoch 499/500
1/4 [======>.......................] - ETA: 0s - loss: 0.00334/4 [
==============================] - 0s 2ms/step - loss: 0.0034
Epoch 500/500
1/4 [======>.......................] - ETA: 0s - loss: 0.00434/4 [
==============================] - 0s 2ms/step - loss: 0.0034
1/1 [==============================] - ETA: 0s1/1 [==============================
] - 0s 97ms/step
1/1 [==============================] - ETA: 0s1/1 [==============================
] - 0s 58ms/step
X=[0.89337759 0.65864154],Predicted_probability=[0.0025765],Predicted_class=0
X=[0.29097707 0.12978982],Predicted_probability=[0.99506545],Predicted_class=0
X=[0.78082614 0.75391697],Predicted_probability=[0.00432174],Predicted_class=0
>>>
```

[C] Aim: Using a deep field forward network with two hidden layers for performing linear regression and predicting values.

Source Code

```
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=1000,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

```
============= RESTART: C:/Users/COMP/Desktop/Uzma Khan/DL prctt4C.py =========
1/1 [==============================] - ETA: 0s
1/1 [==============================] - 0s 86ms/step
X=[0.29466096 0.30317302],Predicted=[0.18350504]
X=[0.39445118 0.79390858],Predicted=[0.7582105]
X=[0.02884127 0.6208843 ],Predicted=[0.39358082]
>>
```

PRACTICAL NO.:5

[A] Aim: Evaluating feed forward deep network for regression using KFold cross validation.

```
Source Code import
numpy as np
from sklearn.model_selection import KFold from
tensorflow.keras.models import Sequential from
tensorflow.keras.layers import Dense

# Generate sample data X =
np.random.rand(1000, 10) y =
np.sum(X, axis=1)

# Define KFold cross-validation
kfold = KFold(n_splits=5, shuffle=True, random_state=42)

# Initialize list to store evaluation metrics eval_metrics
= []

# Iterate through each fold for train_index,
test_index in kfold.split(X):
  # Split data into training and testing sets
X_train, X_test = X[train_index], X[test_index]
y_train, y_test = y[train_index], y[test_index]

  # Define and compile model
model = Sequential()
  model.add(Dense(64, activation='relu', input_dim=10))
model.add(Dense(1))
```

model.compile(optimizer='adam', loss='mse', metrics=['mae'])

 # Fit model to training data

model.fit(X_train, y_train, epochs=100, batch_size=32, verbose=0)

 # Evaluate model on testing data

eval_metrics.append(model.evaluate(X_test, y_test))

 # Print average evaluation metrics across all folds print("Average evaluation metrics:")

print("Loss:", np.mean([m[0] for m in eval_metrics])) print("MAE:", np.mean([m[1] for m in eval_metrics]))

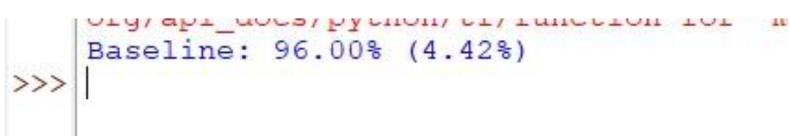
Output

```
============ RESTART: C:\Users\COMP\Desktop\Uzma Khan\DL prct 5A.py ============
1/7 [===>..........................] - ETA: 0s - loss: 4.9209e-04 - mae: 0.0185
2/7 [=======>......................] - ETA: 0s - loss: 5.1225e-04 - mae: 0.0177
7
/7 [==============================] - 0s 13ms/step - loss: 4.9869e-04 - mae: 0.0
178
1/7 [===>..........................] - ETA: 0s - loss: 4.7571e-04 - mae: 0.018 1
7/7 [==============================] - 0s 1ms/step - loss: 5.3811e-04 - mae: 0.0
181
1/7 [===>..........................] - ETA: 0s - loss: 4.2641e-04 - mae: 0.011 5
7/7 [==============================] - 0s 2ms/step - loss: 2.6705e-04 - mae: 0.0
099
1/7 [===>..........................] - ETA: 0s - loss: 2.6106e-04 - mae: 0.011 9
7/7 [==============================] - 0s 1ms/step - loss: 2.6055e-04 - mae: 0.0
117
1/7 [===>..........................] - ETA: 0s - loss: 3.0869e-04 - mae: 0.014 6
7/7 [==============================] - 0s 2ms/step - loss: 6.3452e-04 - mae: 0.0
196
Average evaluation metrics:
Loss: 0.0004397855664137751
MAE: 0.015424948930740357
>>
```

[B] Aim: Evaluating feed forward deep network for multiclass Classification using KFold cross-validation.

Source Code
```
import pandas from
keras.models import Sequential from
keras.layers import Dense
from keras.wrappers.scikit_learn import KerasClassifier from
keras.utils import np_utils
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold from
sklearn.preprocessing import LabelEncoder from
sklearn import datasets from sklearn.pipeline import
Pipeline dataset = datasets.load_iris() X =
dataset.data[:,0:4].astype(float)
Y = dataset.target encoder
= LabelEncoder()
encoder.fit(Y)
encoded_Y = encoder.transform(Y)
# convert integers to dummy variables (i.e. one hot encoded)
dummy_y = np_utils.to_categorical(encoded_Y)
# define baseline model def
baseline_model():    model
= Sequential()
  model.add(Dense(8, input_dim=4, activation='relu'))
model.add(Dense(3, activation='softmax'))
  model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
return model
estimator = KerasClassifier(build_fn=baseline_model, epochs=200, 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



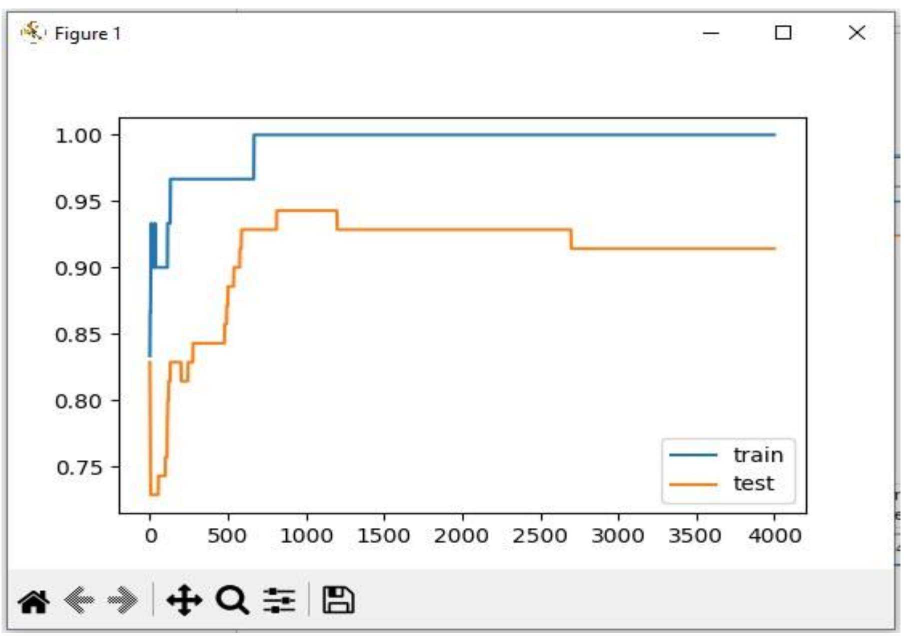
PRACTICAL NO.:6

[A] Aim: Implementing regularization to avoid overfitting in binary classification.

Source Code

```
from matplotlib import pyplot from
sklearn.datasets import make_moons
from keras.models import Sequential
from keras.layers import Dense
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(Dense(500,input_dim=2,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=4000)
pyplot.plot(history.history['accuracy'],label='train')
pyplot.plot(history.history['val_accuracy'],label='test')
pyplot.legend() pyplot.show()
```

Output



```
============ RESTART: C:/Users/COMP/Desktop/Uzma Khan/DL prct 6A.py ============
Epoch 1/4000
1/1 [==============================] - ETA: 0s - loss: 0.6695 - accuracy: 0.833
3
1/1 [==============================] - 1s 804ms/step - loss: 0.6695 - accuracy
: 0.8333 - val_loss: 0.6596 - val_accuracy: 0.8286
Epoch 2/4000
1/1 [==============================] - ETA: 0s - loss: 0.6536 - accuracy: 0.866
7
1/1 [==============================] - 0s 30ms/step - loss: 0.6536 - accuracy:
 0.8667 - val_loss: 0.6495 - val_accuracy: 0.8000
Epoch 3/4000
1/1 [==============================] - ETA: 0s - loss: 0.6381 - accuracy: 0.866
7
1/1 [==============================] - 0s 31ms/step - loss: 0.6381 - accuracy:
 0.8667 - val_loss: 0.6398 - val_accuracy: 0.7571
Epoch 4/4000
1/1 [==============================] - ETA: 0s - loss: 0.6230 - accuracy: 0.866
7
1/1 [==============================] - 0s 30ms/step - loss: 0.6230 - accuracy:
 0.8667 - val_loss: 0.6304 - val_accuracy: 0.7429
Epoch 5/4000
1/1 [==============================] - ETA: 0s - loss: 0.6082 - accuracy: 0.866
7
1/1 [==============================] - 0s 30ms/step - loss: 0.6082 - accuracy:
 0.8667 - val_loss: 0.6212 - val_accuracy: 0.7429
Epoch 6/4000
1/1 [==============================] - ETA: 0s - loss: 0.5938 - accuracy: 0.900
0
1/1 [==============================] - 0s 32ms/step - loss: 0.5938 - accuracy:
 0.9000 - val_loss: 0.6124 - val_accuracy: 0.7286
Epoch 7/4000
```

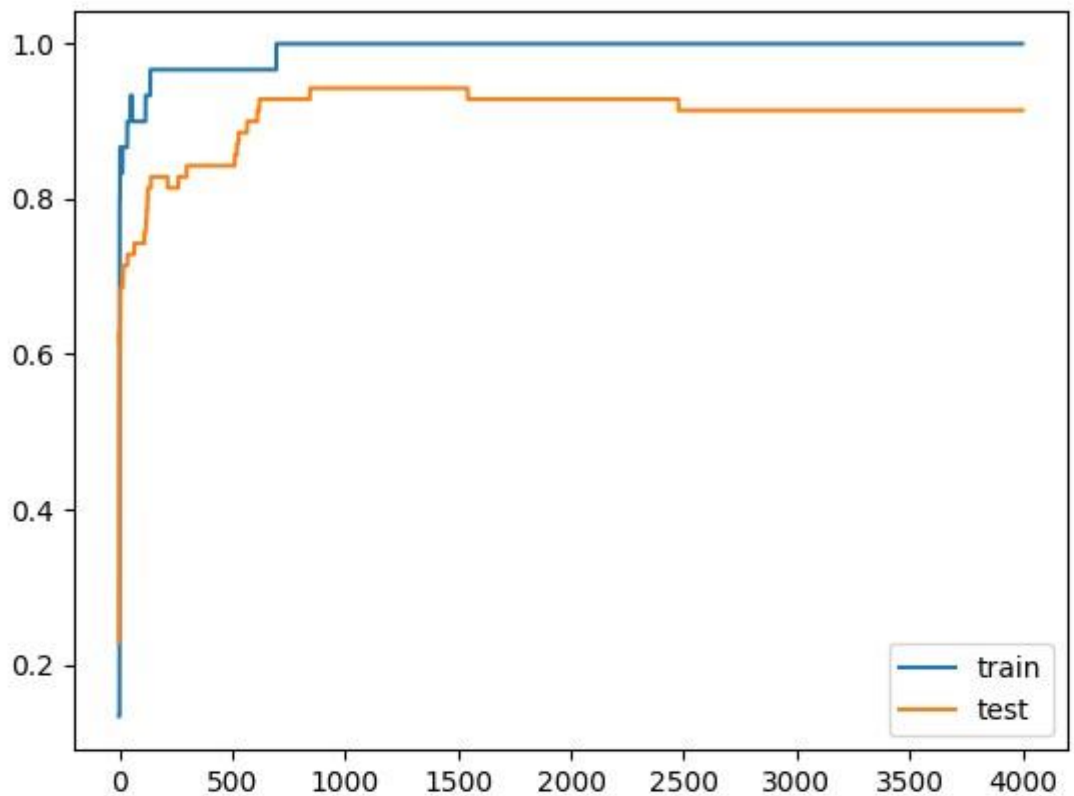
```
Epoch 3994/4000
1/1 [==============================] - ETA: 0s - loss: 1.8399e-04 - accuracy: 1
.0000
1/1 [==============================] - 0s 35ms/step - loss: 1.8399e-04
 - accuracy: 1.0000 - val_loss: 0.5016 - val_accuracy: 0.9143
Epoch 3995/4000
1/1 [==============================] - ETA: 0s - loss: 1.8386e-04 - accuracy: 1
.0000
1/1 [==============================] - 0s 35ms/step - loss: 1.8386e-04
 - accuracy: 1.0000 - val_loss: 0.5016 - val_accuracy: 0.9143
Epoch 3996/4000
1/1 [==============================] - ETA: 0s - loss: 1.8371e-04 - accuracy: 1
.0000
1/1 [==============================] - 0s 35ms/step - loss: 1.8371e-04
 - accuracy: 1.0000 - val_loss: 0.5017 - val_accuracy: 0.9143
Epoch 3997/4000
1/1 [==============================] - ETA: 0s - loss: 1.8356e-04 - accuracy: 1
.0000
1/1 [==============================] - 0s 36ms/step - loss: 1.8356e-04
 - accuracy: 1.0000 - val_loss: 0.5017 - val_accuracy: 0.9143
Epoch 3998/4000
1/1 [==============================] - ETA: 0s - loss: 1.8341e-04 - accuracy: 1
.0000
1/1 [==============================] - 0s 36ms/step - loss: 1.8341e-04
 - accuracy: 1.0000 - val_loss: 0.5018 - val_accuracy: 0.9143
Epoch 3999/4000
1/1 [==============================] - ETA: 0s - loss: 1.8327e-04 - accuracy: 1
.0000
1/1 [==============================] - 0s 35ms/step - loss: 1.8327e-04
 - accuracy: 1.0000 - val_loss: 0.5018 - val_accuracy: 0.9143
Epoch 4000/4000
1/1 [==============================] - ETA: 0s - loss: 1.8313e-04 - accuracy: 1
.0000
1/1 [==============================] - 0s 34ms/step - loss: 1.8313e-04
 - accuracy: 1.0000 - val_loss: 0.5019 - val_accuracy: 0.9143
```

[B] Aim: Implementing L2 regularization Source

Code

```
from matplotlib import pyplot from
sklearn.datasets import make_moons
from keras.models import Sequential
from keras.layers import Dense
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(Dense(500,input_dim=2,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=4000)
```

Output



```
============= RESTART: C:/Users/COMP/Desktop/Uzma Khan/DL prct6B.py ============
Epoch 1/4000
1/1 [==============================] - ETA: 0s - loss: 0.7051 - accuracy: 0.533
3
1/1 [==============================] - 1s 758ms/step - loss: 0.7051 - accuracy
: 0.5333 - val_loss: 0.6946 - val_accuracy: 0.4714
Epoch 2/4000
1/1 [==============================] - ETA: 0s - loss: 0.6887 - accuracy: 0.533
3
1/1 [==============================] - 0s 32ms/step - loss: 0.6887 - accuracy:
 0.5333 - val_loss: 0.6840 - val_accuracy: 0.6000
Epoch 3/4000
1/1 [==============================] - ETA: 0s - loss: 0.6727 - accuracy: 0.833
3
1/1 [==============================] - 0s 29ms/step - loss: 0.6727 - accuracy:
 0.8333 - val_loss: 0.6737 - val_accuracy: 0.6857
Epoch 4/4000
1/1 [==============================] - ETA: 0s - loss: 0.6572 - accuracy: 0.833
3
1/1 [==============================] - 0s 31ms/step - loss: 0.6572 - accuracy:
 0.8333 - val_loss: 0.6638 - val_accuracy: 0.6857
Epoch 5/4000
1/1 [==============================] - ETA: 0s - loss: 0.6421 - accuracy: 0.833
3
1/1 [==============================] - 0s 32ms/step - loss: 0.6421 - accuracy:
 0.8333 - val_loss: 0.6542 - val_accuracy: 0.6857
Epoch 6/4000
1/1 [==============================] - ETA: 0s - loss: 0.6274 - accuracy: 0.833
3
1/1 [==============================] - 0s 31ms/step - loss: 0.6274 - accuracy:
 0.8333 - val_loss: 0.6450 - val_accuracy: 0.6857
Epoch 7/4000
```

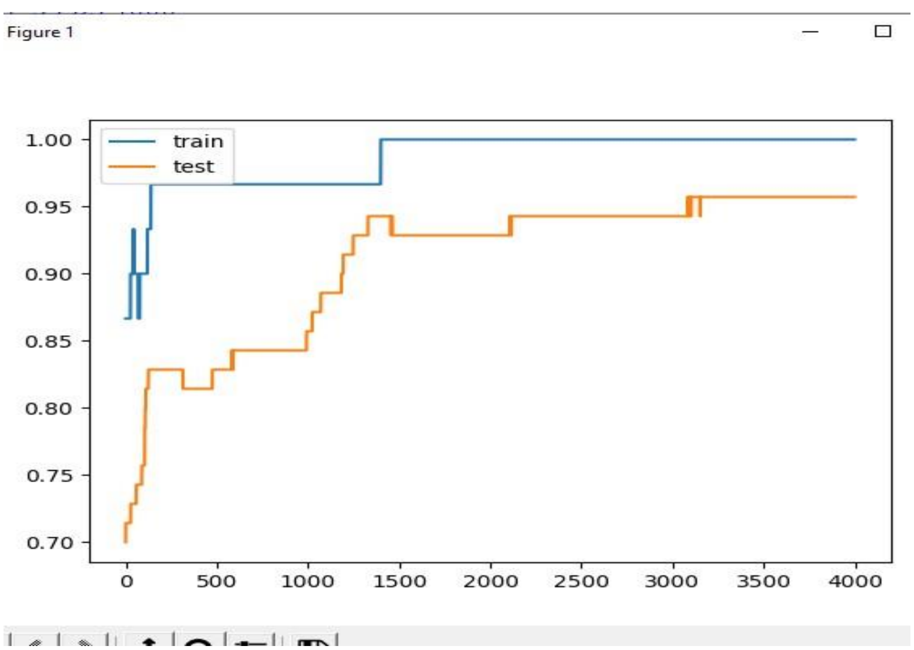
```
Epoch 3994/4000
1/1 [==============================] - ETA: 0s - loss: 0.0151 - accuracy: 1.00 0
0
1/1 [==============================] - 0s 35ms/step - loss: 0.0151 - accuracy:
 1.0000 - val_loss: 0.2760 - val_accuracy: 0.9429
Epoch 3995/4000
1/1 [==============================] - ETA: 0s - loss: 0.0151 - accuracy: 1.000
0
1/1 [==============================] - 0s 35ms/step - loss: 0.0151 - accuracy:
 1.0000 - val_loss: 0.2760 - val_accuracy: 0.9429
Epoch 3996/4000
1/1 [==============================] - ETA: 0s - loss: 0.0150 - accuracy: 1.000
0
1/1 [==============================] - 0s 36ms/step - loss: 0.0150 - accuracy:
 1.0000 - val_loss: 0.2759 - val_accuracy: 0.9429
Epoch 3997/4000
1/1 [==============================] - ETA: 0s - loss: 0.0150 - accuracy: 1.000
0
1/1 [==============================] - 0s 34ms/step - loss: 0.0150 - accuracy:
 1.0000 - val_loss: 0.2758 - val_accuracy: 0.9429
Epoch 3998/4000
1/1 [==============================] - ETA: 0s - loss: 0.0150 - accuracy: 1.000
0
1/1 [==============================] - 0s 35ms/step - loss: 0.0150 - accuracy:
 1.0000 - val_loss: 0.2757 - val_accuracy: 0.9429
Epoch 3999/4000
1/1 [==============================] - ETA: 0s - loss: 0.0150 - accuracy: 1.000
0
1/1 [==============================] - 0s 35ms/step - loss: 0.0150 - accuracy:
 1.0000 - val_loss: 0.2756 - val_accuracy: 0.9429
Epoch 4000/4000
1/1 [==============================] - ETA: 0s - loss: 0.0150 - accuracy: 1.000
0
1/1 [==============================] - 0s 35ms/step - loss: 0.0150 - accuracy:
 1.0000 - val_loss: 0.2756 - val_accuracy: 0.9429
```

[C] Aim: Replacing L2 regularizer with L1 regularizer

Source Code

```
from matplotlib import pyplot from
sklearn.datasets import make_moons
from keras.models import Sequential
from keras.layers import Dense 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(Dense(500,input_dim=2,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=4000)
pyplot.plot(history.history['accuracy'],label='train')
pyplot.plot(history.history['val_accuracy'],label='test') pyplot.legend()
pyplot.show()
```

Output



```
=============== RESTART: C:\Users\COMP\Desktop\DL\practical 6c.py ==============

Warning (from warnings module):
  File "C:\Users\COMP\AppData\Local\Programs\Python\Python310\lib\site-packages\numpy\_distributor_init.py", line 3
    warnings.warn("loaded more than 1 DLL from .libs:"
UserWarning: loaded more than 1 DLL from .libs:
C:\Users\COMP\AppData\Local\Programs\Python\Python310\lib\site-packages\numpy\.libs\libopenblas.EL2C6PLE4ZYW3ECEVIV
gfortran-win_amd64.dll
C:\Users\COMP\AppData\Local\Programs\Python\Python310\lib\site-packages\numpy\.libs\libopenblas.FB5AE2TYXYH2IJRDKGD
gfortran-win_amd64.dll
Epoch 1/4000
1/1 [==============================] - ETA: 0s - loss: 0.7268 - accuracy: 0.8667
1/1 [==============================] - 1s 798ms/step - loss: 0.7268 - accuracy: 0.8667 - val_los
s: 0.7268 - val_accuracy: 0.7000
Epoch 2/4000
1/1 [==============================] - ETA: 0s - loss: 0.7105 - accuracy: 0.8667
1/1 [==============================] - 0s 32ms/step - loss: 0.7105 - accuracy: 0.8667 - val_loss
: 0.7165 - val_accuracy: 0.7143
Epoch 3/4000
1/1 [==============================] - ETA: 0s - loss: 0.6947 - accuracy: 0.8667
1/1 [==============================] - 0s 31ms/step - loss: 0.6947 - accuracy: 0.8667 - val_loss
: 0.7065 - val_accuracy: 0.7143
Epoch 4/4000
1/1 [==============================] - ETA: 0s - loss: 0.6794 - accuracy: 0.8667
1/1 [==============================] - 0s 31ms/step - loss: 0.6794 - accuracy: 0.8667 - val_loss
: 0.6970 - val_accuracy: 0.7143
Epoch 5/4000
1/1 [==============================] - ETA: 0s - loss: 0.6646 - accuracy: 0.8667
1/1 [==============================] - 0s 32ms/step - loss: 0.6646 - accuracy: 0.8667 - val_loss
: 0.6878 - val_accuracy: 0.7143
Epoch 6/4000
1/1 [==============================] - ETA: 0s - loss: 0.6502 - accuracy: 0.8667
1/1 [==============================] - 0s 41ms/step - loss: 0.6502 - accuracy: 0.8667 - val_loss
: 0.6790 - val_accuracy: 0.7143
```

DLE Shell 3.10.4*

Edit Shell Debug Options Window Help

```
Epoch 3992/4000
1/1 [==============================] - ETA: 0s - loss: 0.0426 - accuracy: 1.0000
1/1 [==============================] - 0s 34ms/step - loss: 0.0426 - accuracy: 1.0000 - val_loss
: 0.2181 - val_accuracy: 0.9571
Epoch 3993/4000
1/1 [==============================] - ETA: 0s - loss: 0.0426 - accuracy: 1.0000
1/1 [==============================] - 0s 33ms/step - loss: 0.0426 - accuracy: 1.0000 - val_loss
: 0.2180 - val_accuracy: 0.9571
Epoch 3994/4000
1/1 [==============================] - ETA: 0s - loss: 0.0426 - accuracy: 1.0000
1/1 [==============================] - 0s 33ms/step - loss: 0.0426 - accuracy: 1.0000 - val_loss
: 0.2180 - val_accuracy: 0.9571
Epoch 3995/4000
1/1 [==============================] - ETA: 0s - loss: 0.0426 - accuracy: 1.0000
1/1 [==============================] - 0s 35ms/step - loss: 0.0426 - accuracy: 1.0000 - val_loss
: 0.2181 - val_accuracy: 0.9571
Epoch 3996/4000
1/1 [==============================] - ETA: 0s - loss: 0.0426 - accuracy: 1.0000
1/1 [==============================] - 0s 35ms/step - loss: 0.0426 - accuracy: 1.0000 - val_loss
: 0.2182 - val_accuracy: 0.9571
Epoch 3997/4000
1/1 [==============================] - ETA: 0s - loss: 0.0426 - accuracy: 1.0000
1/1 [==============================] - 0s 32ms/step - loss: 0.0426 - accuracy: 1.0000 - val_loss
: 0.2182 - val_accuracy: 0.9571
Epoch 3998/4000
1/1 [==============================] - ETA: 0s - loss: 0.0426 - accuracy: 1.0000
1/1 [==============================] - 0s 33ms/step - loss: 0.0426 - accuracy: 1.0000 - val_loss
: 0.2182 - val_accuracy: 0.9571
Epoch 3999/4000
1/1 [==============================] - ETA: 0s - loss: 0.0426 - accuracy: 1.0000
1/1 [==============================] - 0s 34ms/step - loss: 0.0426 - accuracy: 1.0000 - val_loss
: 0.2182 - val_accuracy: 0.9571
Epoch 4000/4000
1/1 [==============================] - ETA: 0s - loss: 0.0425 - accuracy: 1.0000
1/1 [==============================] - 0s 33ms/step - loss: 0.0425 - accuracy: 1.0000 - val_loss
: 0.2182 - val_accuracy: 0.9571
```

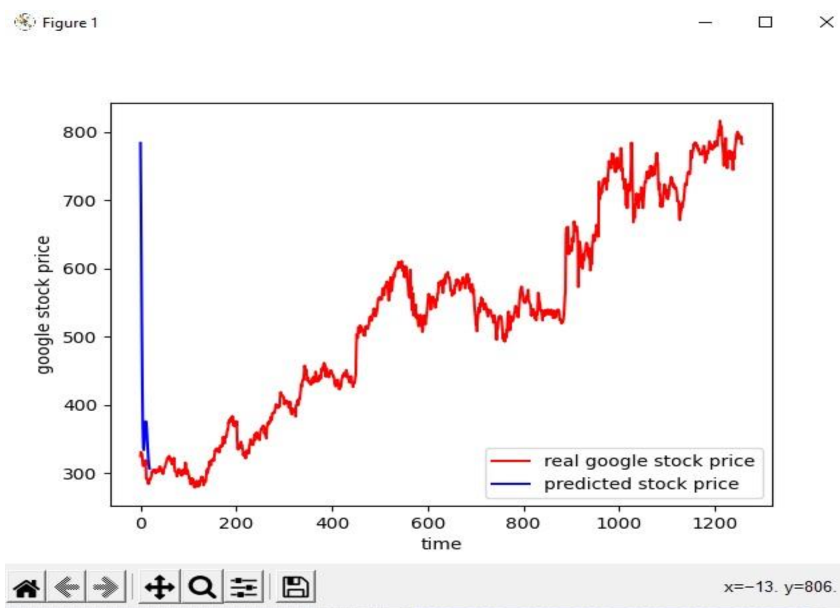
PRACTICAL NO.:7

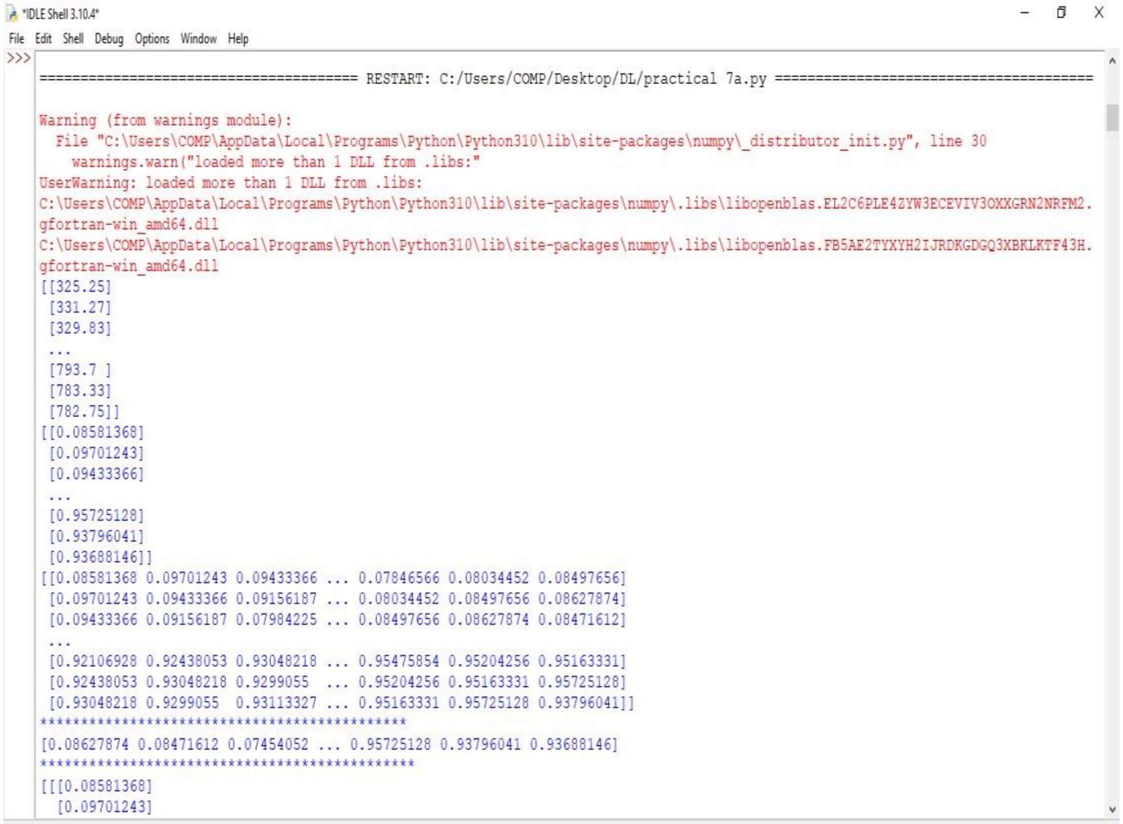
Aim: Demonstrate recurrent neural network that learns to perform sequence analysis for stock price.

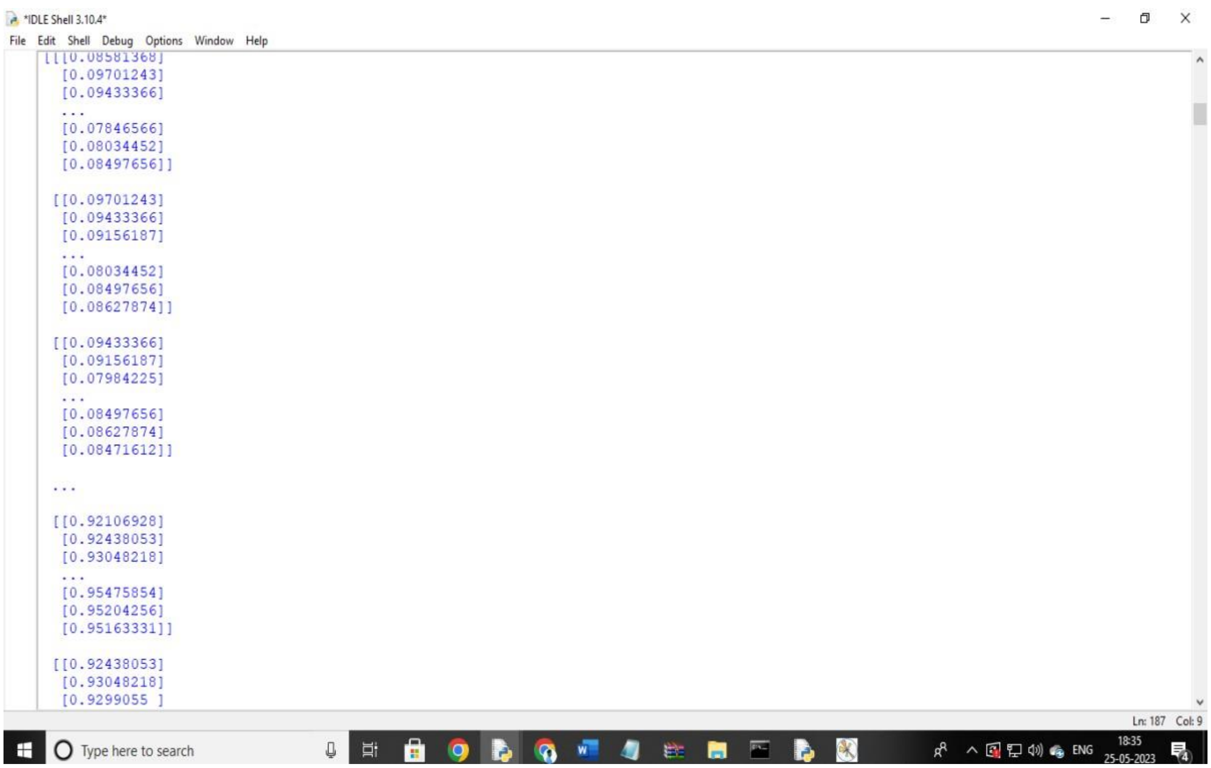
Source Code import numpy as np
import matplotlib.pyplot as plt
import pandas as pd from
keras.models import Sequential from
keras.layers import Dense from
keras.layers import LSTM from
keras.layers import Dropout
from sklearn.preprocessing import MinMaxScaler
dataset_train=pd.read_csv("C:/Users/Admin/Downloads/Google_Stock_Price_Train.csv")
training_set=dataset_train.iloc[:,1:2].values print(training_set)
sc=MinMaxScaler(feature_range=(0,1)) training_set_scaled=sc.fit_transform(training_set)
print(training_set_scaled)
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/Admin/Downloads/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




IDLE Shell 3.10.4
File Edit Shell Debug Options Window Help
>>>
===================================== RESTART: C:/Users/COMP/Desktop/DL/practical 7a.py =====================================
Warning (from warnings module):
File "C:\Users\COMP\AppData\Local\Programs\Python\Python310\lib\site-packages\numpy_distributor_init.py", line 30
warnings.warn("loaded more than 1 DLL from .libs:"
UserWarning: loaded more than 1 DLL from .libs:
C:\Users\COMP\AppData\Local\Programs\Python\Python310\lib\site-packages\numpy\.libs\libopenblas.EL2C6PLE4ZYW3ECEVIV3OXXGRN2NRFM2.gfortran-win_amd64.dll
C:\Users\COMP\AppData\Local\Programs\Python\Python310\lib\site-packages\numpy\.libs\libopenblas.FB5AE2TYXYH2IJRDKGDGQ3XBKLKTF43H.gfortran-win_amd64.dll
[[325.25]
[331.27]
[329.83]
...
[793.7]
[783.33]
[782.75]]
[[0.08581368]
[0.09701243]
[0.09433366]
...
[0.95725128]
[0.93796041]
[0.93688146]]
[[0.08581368 0.09701243 0.09433366 ... 0.07846566 0.08034452 0.08497656]
[0.09701243 0.09433366 0.09156187 ... 0.08034452 0.08497656 0.08627874]
[0.09433366 0.09156187 0.07984225 ... 0.08497656 0.08627874 0.08471612]
...
[0.92106928 0.92438053 0.93048218 ... 0.95475854 0.95204256 0.95163331]
[0.92438053 0.93048218 0.9299055 ... 0.95204256 0.95163331 0.95725128]
[0.93048218 0.9299055 0.93113327 ... 0.95163331 0.95725128 0.93796041]]
****
[0.08627874 0.08471612 0.07454052 ... 0.95725128 0.93796041 0.93688146]
****
[[[0.08581368]
[0.09701243]


IDLE Shell 3.10.4
File Edit Shell Debug Options Window Help
[[[0.08581368]
[0.09701243]
[0.09433366]
...
[0.07846566]
[0.08034452]
[0.08497656]]
[[0.09701243]
[0.09433366]
[0.09156187]
...
[0.08034452]
[0.08497656]
[0.08627874]]
[[0.09433366]
[0.09156187]
[0.07984225]
...
[0.08497656]
[0.08627874]
[0.08471612]]
...
[[0.92106928]
[0.92438053]
[0.93048218]
...
[0.95475854]
[0.95204256]
[0.95163331]]
[[0.92438053]
[0.93048218]
[0.9299055]
Ln: 187 Col: 9
Type here to search
ENG
18:35
25-05-2023

PRACTICAL NO.:8

Aim: Performing encoding and decoding of images using deep autoencoder.

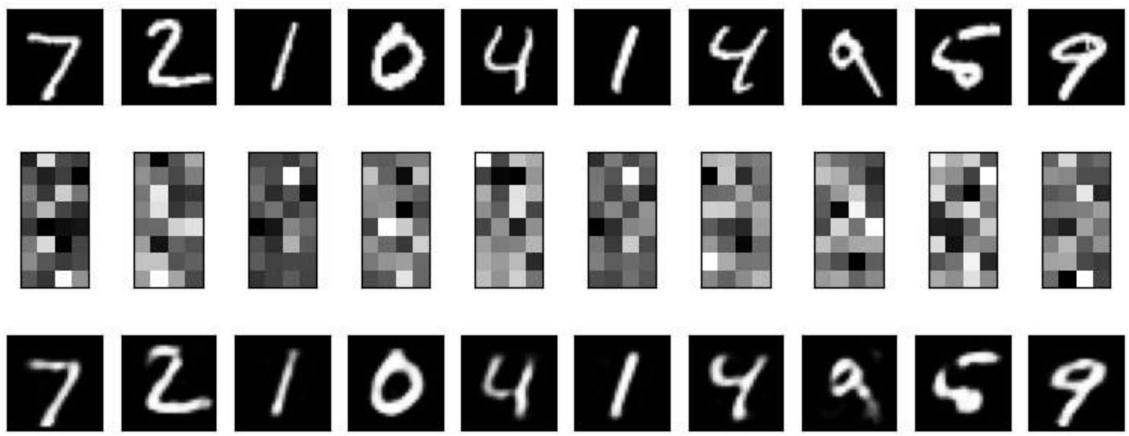
Source Code import keras from
keras import layers from
keras.datasets import mnist
import numpy as np
encoding_dim=32
#this is our input image
input_img=keras.Input(shape=(784,))
#"encoded" is the encoded representation of the input
encoded=layers.Dense(encoding_dim, activation='relu')(input_img)
#"decoded" is the lossy reconstruction of the input decoded=layers.Dense(784,
activation='sigmoid')(encoded)
#creating autoencoder model
autoencoder=keras.Model(input_img,decoded)
#create the encoder model
encoder=keras.Model(input_img,encoded)
encoded_input=keras.Input(shape=(encoding_dim,)) *#Retrive the*
last layer of the autoencoder model
decoder_layer=autoencoder.layers[-1] *#create the decoder model*
decoder=keras.Model(encoded_input,decoder_layer(encoded_input)
) autoencoder.compile(optimizer='adam',loss='binary_crossentropy')
#scale and make train and test dataset
(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)
#train autoencoder with training dataset
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 #
How many digits we will display
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

```
11490434/11490434 [==============================] - 1s 0us/step
(60000, 784)
(10000, 784)
Epoch 1/50
235/235 [==============================] - 1s 4ms/step - loss: 0.2784 - val_loss: 0.1932
Epoch 2/50
235/235 [==============================] - 1s 3ms/step - loss: 0.1722 - val_loss: 0.1533
Epoch 3/50
235/235 [==============================] - 1s 3ms/step - loss: 0.1434 - val_loss: 0.1330
Epoch 4/50
235/235 [==============================] - 1s 3ms/step - loss: 0.1274 - val_loss: 0.1199
Epoch 5/50
235/235 [==============================] - 1s 3ms/step - loss: 0.1170 - val_loss: 0.1114
Epoch 6/50
235/235 [==============================] - 1s 3ms/step - loss: 0.1101 - val_loss: 0.1059
Epoch 7/50
235/235 [==============================] - 1s 3ms/step - loss: 0.1054 - val_loss: 0.1020
```

```
Epoch 46/50
235/235 [==============================] - 1s 3ms/step - loss: 0.0928 - val_loss: 0.0916
Epoch 47/50
235/235 [==============================] - 1s 3ms/step - loss: 0.0927 - val_loss: 0.0916
Epoch 48/50
235/235 [==============================] - 1s 3ms/step - loss: 0.0927 - val_loss: 0.0916
Epoch 49/50
235/235 [==============================] - 1s 3ms/step - loss: 0.0927 - val_loss: 0.0916
Epoch 50/50
235/235 [==============================] - 1s 3ms/step - loss: 0.0927 - val_loss: 0.0916
313/313 [==============================] - 0s 551us/step
313/313 [==============================] - 0s 601us/step
```



PRACTICAL NO.:9

Aim: Implementation of convolutional neural network to predict numbers from number images

Source Code

```
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
#download mnist data and split into train and test sets
(X_train,Y_train),(X_test,Y_test)=mnist.load_data()
#plot the first image in the
dataset 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() #add
model layers #learn image
features
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'])
#train
```

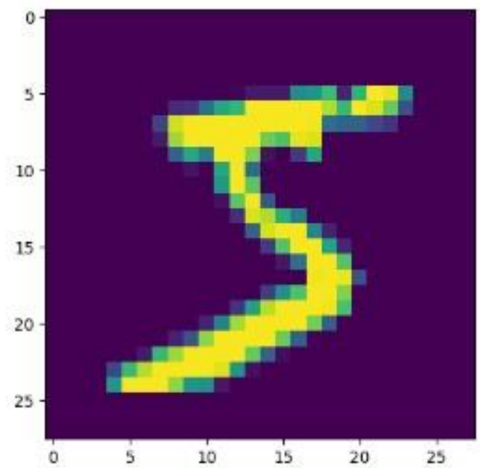
model.fit(X_train,Y_train,validation_data=(X_test,Y_test),epochs=3)

print(model.predict(X_test[:4]))

#actual results for 1st 4 images in the test set

print(Y_test[:4])

Output




```
(28, 28)
[0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
Epoch 1/3
1875/1875 [==============================] - 187s 98ms/step - loss: 0.2184 - accuracy: 0.9511 - val_loss: 0.0879 - val_accurac
y: 0.9712
Epoch 2/3
1875/1875 [==============================] - 185s 99ms/step - loss: 0.0681 - accuracy: 0.9787 - val_loss: 0.0711 - val_accurac
y: 0.9768
Epoch 3/3
1875/1875 [==============================] - 106s 56ms/step - loss: 0.0476 - accuracy: 0.9845 - val_loss: 0.0749 - val_accurac
y: 0.9766
1/1 [==============================] - 0s 157ms/step
[[1.04731434e-07 1.48680113e-13 6.74852799e-07 2.50030007e-06
  1.03254471e-12 2.63807891e-08 7.58218697e-14 9.99996662e-01
  1.24603516e-09 6.71565648e-09]
 [1.07021076e-08 2.64701677e-10 1.00000000e+00 1.47416768e-12
  6.76271365e-12 5.32229711e-16 5.56320212e-10 2.90745680e-17
  1.19557652e-12 3.46894376e-16]
 [8.33379363e-06 9.99377072e-01 3.87140957e-04 1.31158998e-07
  1.09840294e-04 6.11870128e-05 7.35324284e-06 2.21644623e-05
  2.67573287e-05 1.98122185e-09]
 [9.99581993e-01 2.09098822e-10 4.04037892e-05 1.66949174e-08
  9.53089696e-08 4.28229299e-07 8.36978361e-05 2.73629808e-09
  2.27557484e-06 2.91137636e-04]]
[[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 NO.:10

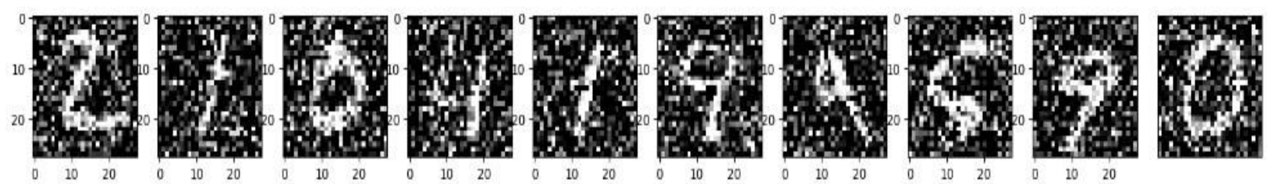
Aim: Denoising of images using autoencoder.

Source Code 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.sha
pe)
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




```
Epoch 1/3
469/469 [==============================] - 142s 289ms/step - loss: 0.1556 - val_loss: 0.1164
Epoch 2/3
469/469 [==============================] - 120s 256ms/step - loss: 0.1135 - val_loss: 0.1089
Epoch 3/3
469/469 [==============================] - 149s 317ms/step - loss: 0.1084 - val_loss: 0.1058
313/313 [==============================] - 19s 32ms/step
```