Practical No	Title
1	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) Aim: Using deep feed forward network with two hidden layers for performing multiclass classification and predicting the class. b) Aim: Using a deep feed forward network with two hidden layers for performing classification and predicting the probability of class. c) Aim: Using a deep feed forward network with two hidden layers for performing linear regression and predicting values.
5	a) Evaluating feed forward deep network for regression using KFold cross validation. b) Evaluating feed forward deep network for multiclass Classification using KFold cross-validation.
6	Implementing regularization to avoid overfitting in binary classification.
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

Aim: Performing matrix multiplication and finding eigen vectors and eigen values 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)
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:

```
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:
[[7.791751  6.3527837]
 [6.8659496 5.229142 ]]


Eigen Vectors:
[[-0.63896394  0.7692366 ]
 [ 0.7692366   0.63896394]]

Eigen Values:
[-0.47403672 13.494929  ]

(venv) PS D:\keras>
```

Practical No:2

Aim: 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=1000,batch_size=4)
print(model.get_weights())
print(model.predict(X,batch_size=4))
```

OUTPUT:

Administrator: Windows PowerShell

```
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
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([[ 0.324126  ,  0.06514561],
       [-0.06398606,  0.25455737]], dtype=float32), array([0., 0.], dtype=float32), array([[-1.166442 ],
       [ 1.0120543]], dtype=float32), array([0.], dtype=float32)]
2021-04-17 12:17:11.354966: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:116] None of the MLIR optimization passes are enabled (registe
red 2)
Epoch 1/1000
1/1 [==============================] - 2s 2s/step - loss: 0.7076 - accuracy: 0.5000
Epoch 2/1000
1/1 [==============================] - 0s 7ms/step - loss: 0.7073 - accuracy: 0.2500
Epoch 3/1000
1/1 [==============================] - 0s 6ms/step - loss: 0.7071 - accuracy: 0.2500
Epoch 4/1000
1/1 [==============================] - 0s 6ms/step - loss: 0.7069 - accuracy: 0.2500
Epoch 5/1000
1/1 [==============================] - 0s 7ms/step - loss: 0.7066 - accuracy: 0.2500
Epoch 6/1000
1/1 [==============================] - 0s 4ms/step - loss: 0.7064 - accuracy: 0.2500
Epoch 7/1000
1/1 [==============================] - 0s 2ms/step - loss: 0.7062 - accuracy: 0.2500
Epoch 8/1000
1/1 [==============================] - 0s 2ms/step - loss: 0.7059 - accuracy: 0.2500
Epoch 9/1000
1/1 [==============================] - 0s 4ms/step - loss: 0.7057 - accuracy: 0.2500
```

Administrator: Windows PowerShell

```
1/1 [==============================] - 0s 4ms/step - loss: 0.5057 - accuracy: 1.0000
Epoch 989/1000
1/1 [==============================] - 0s 3ms/step - loss: 0.5054 - accuracy: 1.0000
Epoch 990/1000
1/1 [==============================] - 0s 5ms/step - loss: 0.5052 - accuracy: 1.0000
Epoch 991/1000
1/1 [==============================] - 0s 5ms/step - loss: 0.5049 - accuracy: 1.0000
Epoch 992/1000
1/1 [==============================] - 0s 2ms/step - loss: 0.5048 - accuracy: 1.0000
Epoch 993/1000
1/1 [==============================] - 0s 4ms/step - loss: 0.5045 - accuracy: 1.0000
Epoch 994/1000
1/1 [==============================] - 0s 2ms/step - loss: 0.5042 - accuracy: 1.0000
Epoch 995/1000
1/1 [==============================] - 0s 4ms/step - loss: 0.5040 - accuracy: 1.0000
Epoch 996/1000
1/1 [==============================] - 0s 2ms/step - loss: 0.5037 - accuracy: 1.0000
Epoch 997/1000
1/1 [==============================] - 0s 2ms/step - loss: 0.5035 - accuracy: 1.0000
Epoch 998/1000
1/1 [==============================] - 0s 4ms/step - loss: 0.5032 - accuracy: 1.0000
Epoch 999/1000
1/1 [==============================] - 0s 4ms/step - loss: 0.5030 - accuracy: 1.0000
Epoch 1000/1000
1/1 [==============================] - 0s 4ms/step - loss: 0.5027 - accuracy: 1.0000
[array([[0.6900411 , 0.5139764 ],
       [0.6899988 , 0.50888795]], dtype=float32), array([-0.6898802 ,  0.00666144], dtype=float32), array([[-2.4736412],
       [ 1.6274512]], dtype=float32), array([-0.41508964], dtype=float32)]
[[0.40029204]
 [0.60435593]
 [0.60630935]
 [0.39012325]]
(venv) PS D:\keras>
```

Practical No:3

Aim: Implementing deep neural network for performing classification task.

Problem statement: the given dataset comprises of health information about diabetic women patient. we need to create deep feed forward network that will classify women suffering from diabetes mellitus as 1.

```
>>> from numpy import loadtxt
>>> from keras.models import Sequential
>>> from keras.layers import Dense
>>>
```

```
Administrator: Windows PowerShell
>>> dataset=loadtxt('pima-indians-diabetes.csv',delimiter=',')
>>> dataset
array([[  6.   , 148.   ,  72.   , ...,   0.627,  50.   ,   1.   ],
       [  1.   ,  85.   ,  66.   , ...,   0.351,  31.   ,   0.   ],
       [  8.   , 183.   ,  64.   , ...,   0.672,  32.   ,   1.   ],
       ...,
       [  5.   , 121.   ,  72.   , ...,   0.245,  30.   ,   0.   ],
       [  1.   , 126.   ,  60.   , ...,   0.349,  47.   ,   1.   ],
       [  1.   ,  93.   ,  70.   , ...,   0.315,  23.   ,   0.   ]])
>>> X=dataset[:,0:8]
>>> Y=dataset[:,8]
>>> X
array([[  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.   ]])
>>>
>>> Y
array([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.,
```

Creating model:

```
>>> model=Sequential()
```

```
>>> model.add(Dense(12,input_dim=8,activation='relu'))
>>> model.add(Dense(8,activation='relu'))
>>> model.add(Dense(1,activation='sigmoid'))
>>>
```

Compiling and fitting model:

```
>>> model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
>>> model.fit(X,Y,epochs=150,batch_size=10)
```

```
Administrator: Windows PowerShell
>>> model.add(Dense(1,activation='sigmoid'))
>>> model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
>>> model.fit(X,Y,epochs=150,batch_size=10)
2021-04-05 17:40:32.289557: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:116] None of the MLIR optimization passes are enabled (registe
red 2)
Epoch 1/150
77/77 [==============================] - 2s 2ms/step - loss: 2.6770 - accuracy: 0.4399
Epoch 2/150
77/77 [==============================] - 0s 1ms/step - loss: 1.1332 - accuracy: 0.5064
Epoch 3/150
77/77 [==============================] - 0s 2ms/step - loss: 0.8624 - accuracy: 0.5592
Epoch 4/150
77/77 [==============================] - 0s 2ms/step - loss: 0.8135 - accuracy: 0.5700
Epoch 5/150
77/77 [==============================] - 0s 2ms/step - loss: 0.7369 - accuracy: 0.6089
Epoch 6/150
77/77 [==============================] - 0s 1ms/step - loss: 0.7405 - accuracy: 0.6269
Epoch 7/150
77/77 [==============================] - 0s 2ms/step - loss: 0.7157 - accuracy: 0.6060
Epoch 8/150
77/77 [==============================] - 0s 1ms/step - loss: 0.6852 - accuracy: 0.6354
Epoch 9/150
77/77 [==============================] - 0s 2ms/step - loss: 0.6585 - accuracy: 0.6398
Epoch 10/150
77/77 [==============================] - 0s 2ms/step - loss: 0.6524 - accuracy: 0.6330
Epoch 11/150
77/77 [==============================] - 0s 2ms/step - loss: 0.6671 - accuracy: 0.6584
Epoch 12/150
77/77 [==============================] - 0s 2ms/step - loss: 0.6216 - accuracy: 0.6857
Epoch 13/150
77/77 [==============================] - 0s 2ms/step - loss: 0.6656 - accuracy: 0.6469
Epoch 14/150
77/77 [==============================] - 0s 2ms/step - loss: 0.6304 - accuracy: 0.6870
Epoch 15/150
77/77 [==============================] - 0s 2ms/step - loss: 0.6290 - accuracy: 0.6594
Epoch 16/150
77/77 [==============================] - 0s 2ms/step - loss: 0.6033 - accuracy: 0.6722
```

Evaluating the accuracy:

```
>>> _,accuracy=model.evaluate(X,Y)
24/24 [==============================] - 0s 1ms/step - loss: 0.4849 - accuracy: 0
.7591
>>> print('Accuracy of model is',(accuracy*100))
Accuracy of model is 75.91145634651184
>>>
```

Using model for prediction class:

```
>>> prediction=model.predict_classes(X)
```

```
>>> exec("for i in range(5):print(X[i].tolist(),prediction[i],Y[i])")
[6.0, 148.0, 72.0, 35.0, 0.0, 33.6, 0.627, 50.0] [1] 1.0
[1.0, 85.0, 66.0, 29.0, 0.0, 26.6, 0.351, 31.0] [0] 0.0
[8.0, 183.0, 64.0, 0.0, 0.0, 23.3, 0.672, 32.0] [1] 1.0
[1.0, 89.0, 66.0, 23.0, 94.0, 28.1, 0.167, 21.0] [0] 0.0
[0.0, 137.0, 40.0, 35.0, 168.0, 43.1, 2.288, 33.0] [1] 1.0
>>>
```

Practical No:4

a) Aim: Using 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=500)

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

Ynew=model.predict_classes(Xnew)
for i in range(len(Xnew)):
    print("X=%s,Predicted=%s,Desired=%s"%(Xnew[i],Ynew[i],Yreal[i]))
```

OUTPUT:

Administrator: Windows PowerShell

```
4/4 [==============================] - 0s 2ms/step - loss: 0.6935
Epoch 488/500
4/4 [==============================] - 0s 2ms/step - loss: 0.6927
Epoch 489/500
4/4 [==============================] - 0s 3ms/step - loss: 0.6931
Epoch 490/500
4/4 [==============================] - 0s 3ms/step - loss: 0.6928
Epoch 491/500
4/4 [==============================] - 0s 2ms/step - loss: 0.6938
Epoch 492/500
4/4 [==============================] - 0s 5ms/step - loss: 0.6929
Epoch 493/500
4/4 [==============================] - 0s 2ms/step - loss: 0.6928
Epoch 494/500
4/4 [==============================] - 0s 3ms/step - loss: 0.6928
Epoch 495/500
4/4 [==============================] - 0s 2ms/step - loss: 0.6930
Epoch 496/500
4/4 [==============================] - 0s 2ms/step - loss: 0.6934
Epoch 497/500
4/4 [==============================] - 0s 2ms/step - loss: 0.6934
Epoch 498/500
4/4 [==============================] - 0s 2ms/step - loss: 0.6933
Epoch 499/500
4/4 [==============================] - 0s 3ms/step - loss: 0.6930
Epoch 500/500
4/4 [==============================] - 0s 2ms/step - loss: 0.6940
D:\keras\venv\lib\site-packages\tensorflow\python\keras\engine\sequential.py:450: UserWarning: `model.predict_classes()` is deprecated and will be re
moved after 2021-01-01. Please use instead:* `np.argmax(model.predict(x), axis=-1)`,   if your model does multi-class classification   (e.g. if it us
es a `softmax` last-layer activation).* `(model.predict(x) > 0.5).astype("int32")`,   if your model does binary classification   (e.g. if it uses a `
sigmoid` last-layer activation).
  warnings.warn('`model.predict_classes()` is deprecated and '
X=[0.89337759 0.65864154],Predicted=[0]
X=[0.29097707 0.12978982],Predicted=[0]
X=[0.78082614 0.75391697],Predicted=[0]
(venv) PS D:\keras>
```

Administrator: Windows PowerShell

```
4/4 [==============================] - 0s 2ms/step - loss: 0.0031
Epoch 489/500
4/4 [==============================] - 0s 2ms/step - loss: 0.0031
Epoch 490/500
4/4 [==============================] - 0s 2ms/step - loss: 0.0034
Epoch 491/500
4/4 [==============================] - 0s 2ms/step - loss: 0.0030
Epoch 492/500
4/4 [==============================] - 0s 2ms/step - loss: 0.0031
Epoch 493/500
4/4 [==============================] - 0s 2ms/step - loss: 0.0031
Epoch 494/500
4/4 [==============================] - 0s 1ms/step - loss: 0.0031
Epoch 495/500
4/4 [==============================] - 0s 2ms/step - loss: 0.0028
Epoch 496/500
4/4 [==============================] - 0s 1ms/step - loss: 0.0028
Epoch 497/500
4/4 [==============================] - 0s 3ms/step - loss: 0.0030
Epoch 498/500
4/4 [==============================] - 0s 2ms/step - loss: 0.0031
Epoch 499/500
4/4 [==============================] - 0s 3ms/step - loss: 0.0028
Epoch 500/500
4/4 [==============================] - 0s 2ms/step - loss: 0.0032
D:\keras\venv\lib\site-packages\tensorflow\python\keras\engine\sequential.py:450: User
Warning: `model.predict_classes()` is deprecated and will be removed after 2021-01-01.
 Please use instead:* `np.argmax(model.predict(x), axis=-1)`,   if your model does mul
ti-class classification   (e.g. if it uses a `softmax` last-layer activation).* `(mode
l.predict(x) > 0.5).astype("int32")`,   if your model does binary classification   (e.
g. if it uses a `sigmoid` last-layer activation).
  warnings.warn('`model.predict_classes()` is deprecated and '
X=[0.89337759 0.65864154],Predicted=[0],Desired=0
X=[0.29097707 0.12978982],Predicted=[1],Desired=1
X=[0.78082614 0.75391697],Predicted=[0],Desired=0
(venv) PS D:\keras>
```

b) Aim: 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=500)

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

Xnew=scalar.transform(Xnew)

Yclass=model.predict_classes(Xnew)

Ynew=model.predict_proba(Xnew)

for i in range(len(Xnew)):

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

OUTPUT:

```
Administrator: Windows PowerShell
4/4 [==============================] - 0s 1ms/step - loss: 0.1005
Epoch 491/500
4/4 [==============================] - 0s 3ms/step - loss: 0.0994
Epoch 492/500
4/4 [==============================] - 0s 2ms/step - loss: 0.1013
Epoch 493/500
4/4 [==============================] - 0s 2ms/step - loss: 0.0983
Epoch 494/500
4/4 [==============================] - 0s 4ms/step - loss: 0.0969
Epoch 495/500
4/4 [==============================] - 0s 3ms/step - loss: 0.1075
Epoch 496/500
4/4 [==============================] - 0s 2ms/step - loss: 0.0966
Epoch 497/500
4/4 [==============================] - 0s 2ms/step - loss: 0.0987
Epoch 498/500
4/4 [==============================] - 0s 2ms/step - loss: 0.1013
Epoch 499/500
4/4 [==============================] - 0s 3ms/step - loss: 0.1020
Epoch 500/500
4/4 [==============================] - 0s 1ms/step - loss: 0.1065
D:\keras\venv\lib\site-packages\tensorflow\python\keras\engine\sequential.py:450: User
Warning: `model.predict_classes()` is deprecated and will be removed after 2021-01-01.
 Please use instead:* `np.argmax(model.predict(x), axis=-1)`,   if your model does mul
ti-class classification   (e.g. if it uses a `softmax` last-layer activation).* `(mode
l.predict(x) > 0.5).astype("int32")`,   if your model does binary classification   (e.
g. if it uses a `sigmoid` last-layer activation).
  warnings.warn('`model.predict_classes()` is deprecated and '
D:\keras\venv\lib\site-packages\tensorflow\python\keras\engine\sequential.py:425: User
Warning: `model.predict_proba()` is deprecated and will be removed after 2021-01-01. P
lease use `model.predict()` instead.
  warnings.warn('`model.predict_proba()` is deprecated and '
X=[0.89337759 0.65864154],Predicted_probability=[0.1775814],Predicted_class=[0]
X=[0.29097707 0.12978982],Predicted_probability=[0.9977311],Predicted_class=[1]
X=[0.78082614 0.75391697],Predicted_probability=[0.1775814],Predicted_class=[0]
(venv) PS D:\keras>
```

c) Aim: 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=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:

```
X=[0.29466096 0.30317302],Predicted=[0.18255734]
X=[0.39445118 0.79390858],Predicted=[0.7581165]
X=[0.02884127 0.6208843 ],Predicted=[0.3932857]
(venv) PS D:\keras>
```

Practical No:5(a)

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

```
import pandas as pd
from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn 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("housing.csv",delim_whitespace=True,header=None)
dataset=dataframe.values
X=dataset[:,0:13]
Y=dataset[:,13]
def wider_model():
    model=Sequential()
    model.add(Dense(15,input_dim=13,kernel_initializer='normal',activation='relu'))
    model.add(Dense(13,kernel_initializer='normal',activation='relu'))
    model.add(Dense(1,kernel_initializer='normal'))
    model.compile(loss='mean_squared_error',optimizer='adam')
    return model
estimators=[]
estimators.append(('standardize',StandardScaler()))
estimators.append(('mlp',KerasRegressor(build_fn=wider_model,epochs=100,batch_size=5)))
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: -20.88 (24.29) MSE
(venv) PS D:\keras>
```

(After changing neuron)

```
model.add(Dense(20, input_dim=13,kernel_initializer='normal',activation='relu'))
```

```
Wider: -22.17 (24.38) MSE
(venv) PS D:\keras>
```

Practical No:5b

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

```
#loading libraries

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

#loading dataset

df=pandas.read_csv('Flower.csv',header=None)

print(df)

#splitting dataset into input and output variables

X = df.iloc[:,0:4].astype(float)

y=df.iloc[:,4]

#print(X)

#print(y)

#encoding string output into numeric output

encoder=LabelEncoder()

encoder.fit(y)

encoded_y=encoder.transform(y)

print(encoded_y)

dummy_Y=np_utils.to_categorical(encoded_y)

print(dummy_Y)

def baseline_model():

	# create model

	model = Sequential()

	model.add(Dense(8, input_dim=4, activation='relu'))

	model.add(Dense(3, activation='softmax'))

	# Compile model

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

```
Epoch 98/100
5/5 [==============================] - 0s 0s/step - loss: 0.3899 - accuracy: 0.9313
Epoch 99/100
5/5 [==============================] - 0s 0s/step - loss: 0.3896 - accuracy: 0.9230
Epoch 100/100
5/5 [==============================] - 0s 0s/step - loss: 0.3682 - accuracy: 0.9361
[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.]
```

```
^^^^^^^^^^^^^^^^^^^^^^
[0.9145307  0.08423453 0.00123477]
[0.88751584 0.1100563  0.00242792]
[0.8999843  0.09803853 0.00197715]
[0.858188   0.13759544 0.00421653]
[0.9138275  0.08489472 0.00127787]
[0.8994011  0.09916449 0.0014343 ]
[0.8872866  0.11023647 0.00247695]
[0.89339536 0.10458492 0.00201967]
[0.8545533  0.14064151 0.00480518]
[0.87742513 0.11963753 0.00293737]
[0.9203753  0.07866727 0.00095734]
[0.8665611  0.1300417  0.00339716]
[0.88403696 0.11323617 0.0027269 ]
[0.9008803  0.09682965 0.00229002]
[9.5539063e-01 4.4350266e-02 2.5906262e-04]
[9.4327897e-01 5.6333560e-02 3.8754733e-04]
[9.3672138e-01 6.2714875e-02 5.6370755e-04]
[0.91191673 0.08680107 0.00128225]
[0.9100969  0.08882014 0.00108295]
[0.91078293 0.08794734 0.00126965]
[0.8827079  0.11510085 0.00219123]
[0.9060573  0.09255142 0.00139134]
[9.3434143e-01 6.4821333e-02 8.3730859e-04]
[0.85551745 0.14102885 0.00345369]
[0.80272377 0.1895675  0.00770868]
```

Code 2:

```
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


dataset=pandas.read_csv("Flower.csv",header=None)
```

```
dataset1=dataset.values
X=dataset1[:,0:4].astype(float)
Y=dataset1[:,4]
print(Y)
encoder=LabelEncoder()
encoder.fit(Y)
encoder_Y=encoder.transform(Y)
print(encoder_Y)
dummy_Y=np_utils.to_categorical(encoder_Y)
print(dummy_Y)
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=100,batch_size=5)
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))
```

```
3/3 [==============================] - 0s 2ms/step - loss: 0.2491 - accuracy: 0.9333
Baseline: 96.00% (4.42%)
```

(Changing neuron)

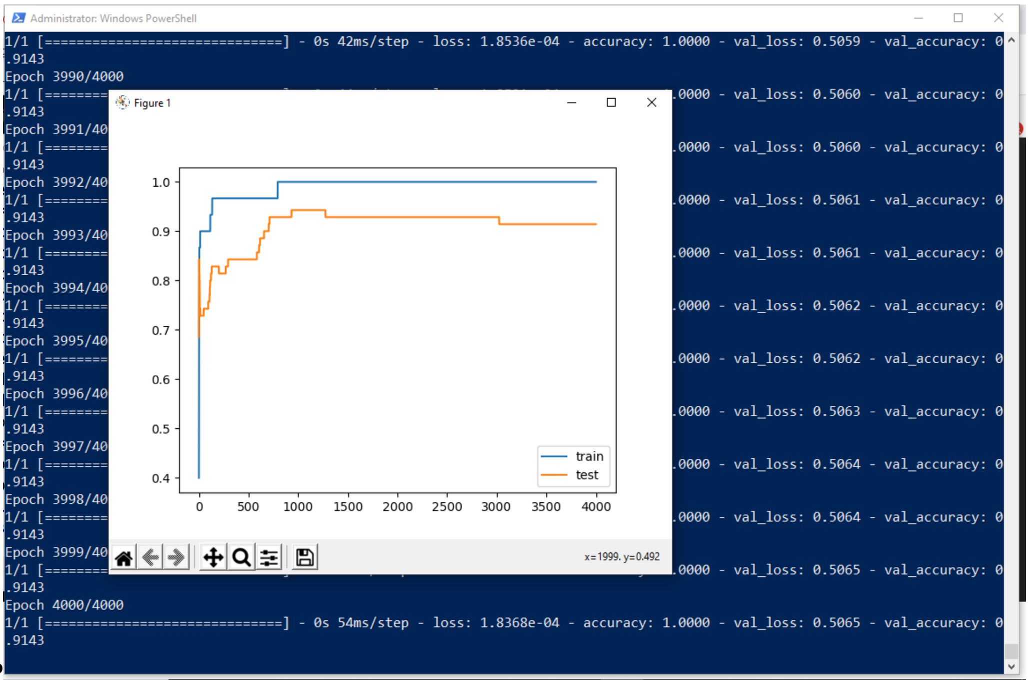
```
model.add(Dense(10,input_dim=4,activation='relu'))
```

```
3/3 [==============================] - 0s 999us/step - loss: 0.1436 - accuracy: 1.0000
Baseline: 98.67% (2.67%)
```

Practical No :6

Aim: 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

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:]

#print(trainX)

#print(trainY)

#print(testX)

#print(testY)

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()
```



P

The above code and resultant graph demonstrate overfitting with accuracy of testing data less than accuracy of training data also the accuracy of testing data increases once and then start decreases gradually.to solve this problem we can use regularization

Hence, we will add two lines in the above code as highlighted below to 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

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:]

#print(trainX)

#print(trainY)

#print(testX)

#print(testY)

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'])

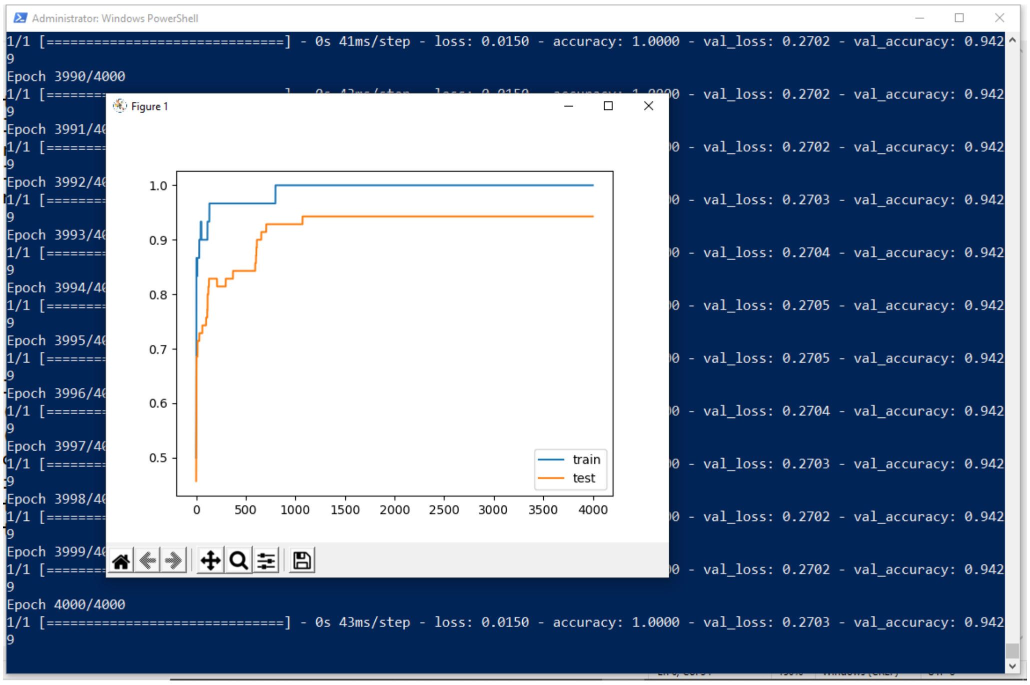
history=model.fit(trainX,trainY,validation_data=(testX,testY),epochs=4000)

pyplot.plot(history.history['accuracy'],label='train')

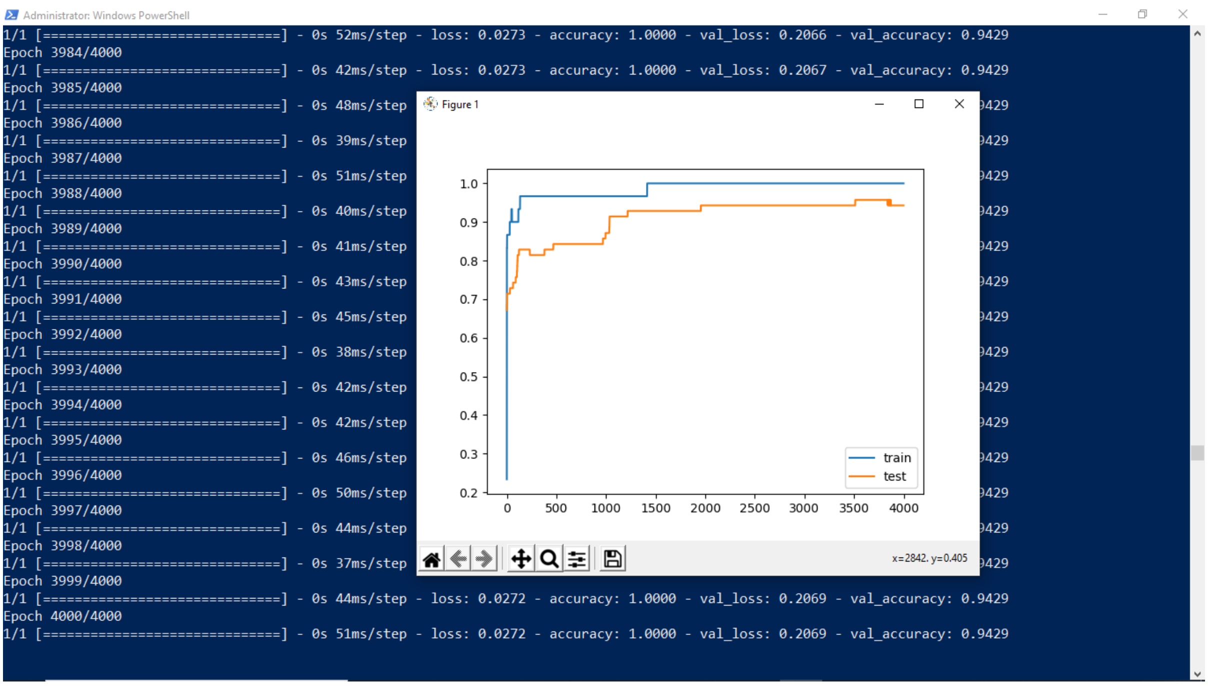
pyplot.plot(history.history['val_accuracy'],label='test')

pyplot.legend()

pyplot.show()



By replacing l2 regularizer with l1 regularizer at the same learning rate 0.001 we get the following output.



By applying l1 and l2 regularizer we can observe the following changes in accuracy of both trainig and testing data. The changes in code are also highlighted.

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:]

#print(trainX)

#print(trainY)

#print(testX)

#print(testY)

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)

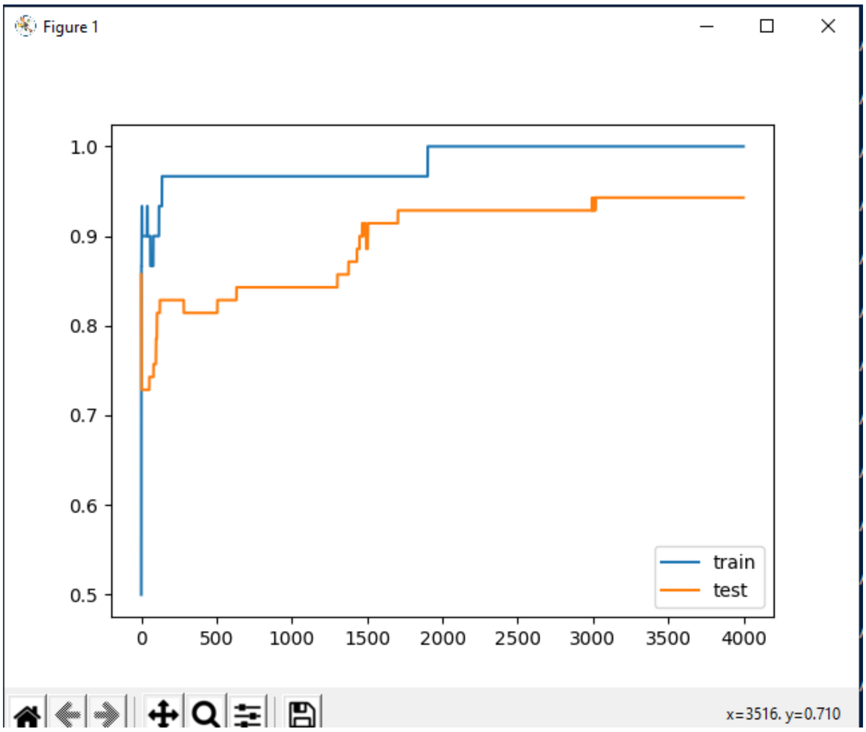
pyplot.plot(history.history['accuracy'],label='train')

pyplot.plot(history.history['val_accuracy'],label='test')

pyplot.legend()

pyplot.show()

OUTPUT



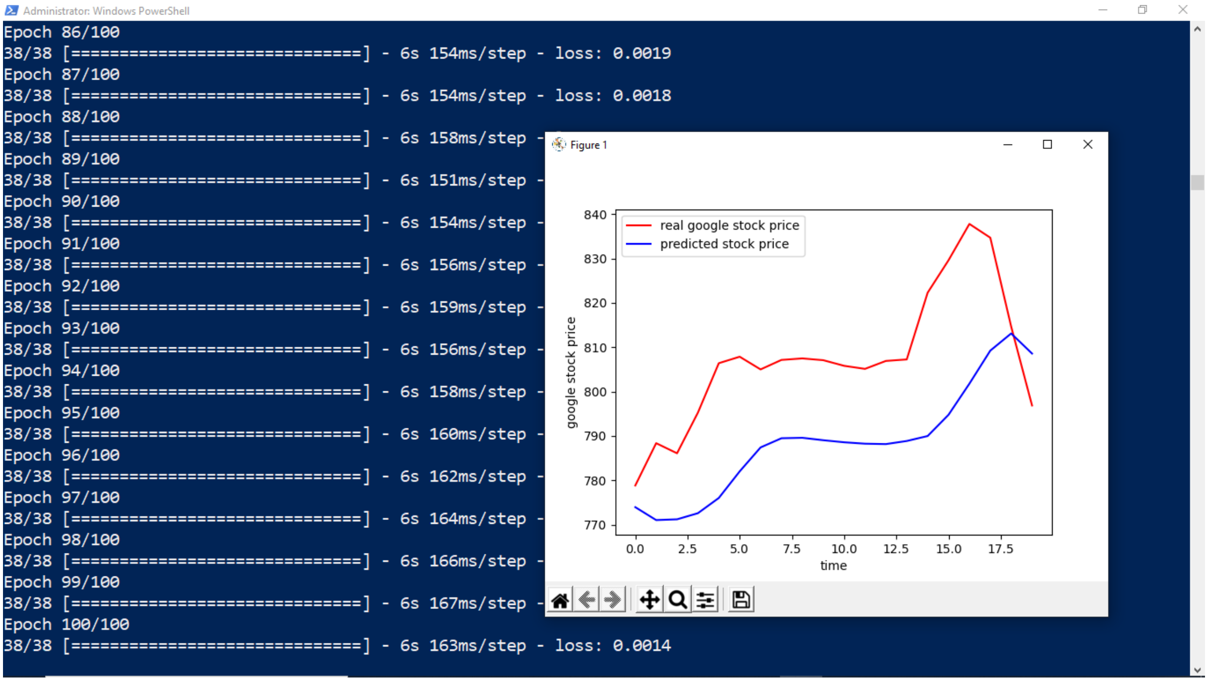
Practical No:7

Aim: 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('Google_Stock_price_train.csv')
#print(dataset_train)
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('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


Administrator: Windows PowerShell
Epoch 86/100
38/38 [==============================] - 6s 154ms/step - loss: 0.0019
Epoch 87/100
38/38 [==============================] - 6s 154ms/step - loss: 0.0018
Epoch 88/100
38/38 [==============================] - 6s 158ms/step -
Epoch 89/100
38/38 [==============================] - 6s 151ms/step -
Epoch 90/100
38/38 [==============================] - 6s 154ms/step -
Epoch 91/100
38/38 [==============================] - 6s 156ms/step -
Epoch 92/100
38/38 [==============================] - 6s 159ms/step -
Epoch 93/100
38/38 [==============================] - 6s 156ms/step -
Epoch 94/100
38/38 [==============================] - 6s 158ms/step -
Epoch 95/100
38/38 [==============================] - 6s 160ms/step -
Epoch 96/100
38/38 [==============================] - 6s 162ms/step -
Epoch 97/100
38/38 [==============================] - 6s 164ms/step -
Epoch 98/100
38/38 [==============================] - 6s 166ms/step -
Epoch 99/100
38/38 [==============================] - 6s 167ms/step -
Epoch 100/100
38/38 [==============================] - 6s 163ms/step - loss: 0.0014
Figure 1
real google stock price
predicted stock price
google stock price
time
840
830
820
810
800
790
780
770
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

Practical No:8

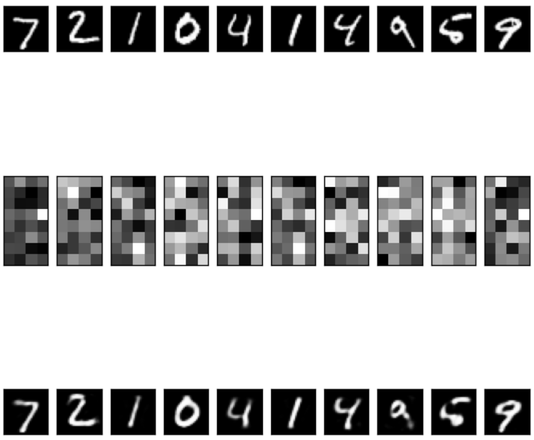
Aim: Performing encoding and decoding of images using deep autoencoder.

```
import keras
from keras import layers
from keras.datasets import mnist
import numpy as np
encoding_dim=32
#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):
    # display original
    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)
    # display encoded image
    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)
    # display reconstruction
    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:


Administrator: Windows PowerShell
235/235 [============================] - 3s 13ms/step - loss: 0.0929 - val_loss: 0.0918
Epoch 34/50
235/235 [============================] - 3s 13ms/step - loss: 0.0926 - val_loss: 0.0917
Epoch 35/50
235/235 [============================] - 3s 13ms/step - loss: 0.0928 - val_loss: 0.0917
Epoch 36/50
235/235 [=====
Epoch 37/50
235/235 [=====
Epoch 38/50
235/235 [=====
Epoch 39/50
235/235 [=====
Epoch 40/50
235/235 [=====
Epoch 41/50
235/235 [=====
Epoch 42/50
235/235 [=====
Epoch 43/50
235/235 [=====
Epoch 44/50
235/235 [=====
Epoch 45/50
235/235 [=====
Epoch 46/50
235/235 [=====
Epoch 47/50
Epoch 48/50
235/235 [============================] - 3s 14ms/step - loss: 0.0927 - val_loss: 0.0915
Epoch 49/50
235/235 [============================] - 3s 13ms/step - loss: 0.0927 - val_loss: 0.0915
Epoch 50/50
235/235 [============================] - 3s 15ms/step - loss: 0.0926 - val_loss: 0.0914
Figure 1



Practical No:9

Aim: 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

#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

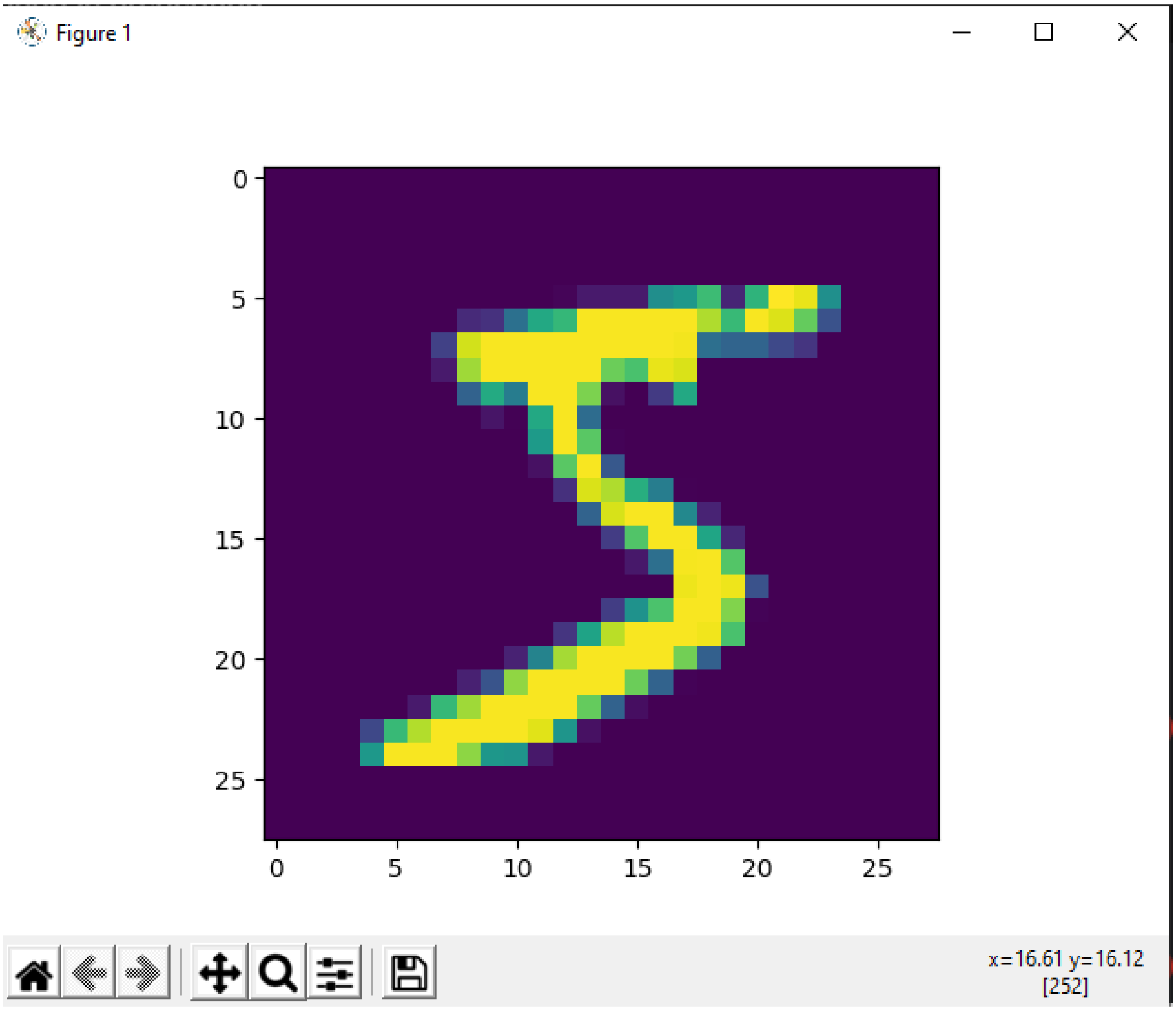
model.fit(X_train,Y_train,validation_data=(X_test,Y_test),epochs=3)

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.]

```
(venv) PS D:\keras> python pract6.py
(28, 28)
[0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
```

```
Epoch 1/3
1875/1875 [==============================] - 235s 124ms/step - loss: 0.9714 - accuracy: 0.9111
- val_loss: 0.1084 - val_accuracy: 0.9661
Epoch 2/3
1875/1875 [==============================] - 242s 129ms/step - loss: 0.0663 - accuracy: 0.9789
- val_loss: 0.0787 - val_accuracy: 0.9758
Epoch 3/3
1875/1875 [==============================] - 241s 128ms/step - loss: 0.0458 - accuracy: 0.9854
- val_loss: 0.0904 - val_accuracy: 0.9751
[[8.5066381e-09 1.9058415e-15 1.5103029e-09 6.2544638e-07 4.8599115e-14
  3.8009873e-13 8.0967405e-13 9.9999940e-01 2.3813423e-10 1.8504194e-09]
 [4.6695381e-10 4.9075446e-09 1.0000000e+00 1.4425230e-12 5.5351397e-15
  1.4244286e-16 4.9031729e-10 2.1196991e-15 8.1773255e-13 2.7225001e-19]
 [1.4877173e-06 9.9855584e-01 1.0760028e-04 1.4199993e-07 1.0726219e-03
  6.1853432e-05 5.0982948e-05 6.4035441e-05 8.5100648e-05 3.5164564e-07]
 [9.9999988e-01 7.7231385e-13 9.2269055e-08 2.9055267e-10 1.8901826e-10
  2.9204628e-09 8.1175129e-09 4.1387605e-12 6.0085120e-10 1.4425010e-08]]
[[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.]]
(venv) PS D:\keras>
```

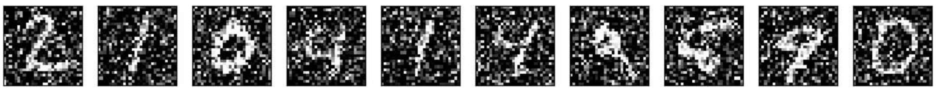
Practical No:10

Aim: 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 3 epochs: