Practical 1

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,na
me="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:

```
Matrix Multiplication Demo
tf.Tensor(
[[1 2 3]
 [4 5 6]], shape=(2, 3), dtype=int32)
tf.Tensor(
[[ 7  8]
 [ 9 10]
 [11 12]], shape=(3, 2), dtype=int32)
Product: tf.Tensor(
[[ 58  64]
 [139 154]], shape=(2, 2), dtype=int32)
Matrix A:
[[6.142591  7.4448104]
 [5.5417624 8.675958 ]]


Eigen Vectors:
[[-0.78192836 -0.6233683 ]
 [ 0.6233683  -0.78192836]]

Eigen Values:
[ 1.7245919 13.093957 ]
```

Practical 2

Solving XOR problem using deep feed forward network.

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=['accurac
y'])
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:

```
28ms/step - accuracy: 0.7500 - loss: 0.5941
Epoch 1000/1000
▯[1m1/1▯[0m ▯[32m ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━▯[0m▯[37m▯[0m ▯[1m0s▯[0m 24ms/step -
accuracy: 0.7500 - loss: 0.5940▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯
▯▯▯▯▯▯▯▯▯▯▯▯▯[1m1/1▯[0m ▯[32m ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━▯[0m▯[37m▯[0m ▯[1m0s▯[0m
29ms/step - accuracy: 0.7500 - loss: 0.5940
[array([[-1.0614685 ,  0.8837547 ],
       [-1.2204928 ,  0.88403314]], dtype=float32), array([ 0.        , -0.8840992], dtype=
float32), array([[ 1.183175 ],
       [-1.0946559]], dtype=float32), array([0.20952563], dtype=float32)]
▯[1m1/1▯[0m ▯[32m ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━▯[0m▯[37m▯[0m ▯[1m0s▯[0m 38ms/step  ▯▯▯
▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯[1m1/1▯[0m ▯[32m ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━▯[0
m▯[37m▯[0m ▯[1m0s▯[0m 86ms/step
[[0.55219066]
 [0.55219066]
 [0.55219066]
 [0.31912208]]
>>>
```

Practical 3

Implementing deep neural network for performing classification task.

Code:

```
from numpy import loadtxt
from keras.models import Sequential
from keras.layers import Dense
dataset=loadtxt('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.,
```

```
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=['accurac
y'])
model.fit(X,Y,epochs=150,batch_size=10)
```

```
Epoch 150/150
[1m 1/77[0m [37m ━━━━━━━━━━━━━━━━━━━━[0m [1m2s[0m 36ms/s
tep - accuracy: 0.8000 - loss: 0.3461
[1m34/77[0m [32m ━━━━━━━━[0m
[37m━━━━━━━━━━━━[0m [1m0s[0m 2ms/step - accuracy: 0.8153 - loss
: 0.4157
[1m65/77[0m [32m ━━━━━━━━━━━━━━━━━[0m[37m ━━━━[0m 
[1m0s[0m 2ms/step - accuracy: 0.8062 - loss: 0.4324
[1m77/77[0m [32m ━━
━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m 2ms/step - accuracy:
0.8033 - loss: 0.4370
<keras.src.callbacks.history.History object at 0x000002683CD18D30>
```

```
_,accuracy=model.evaluate(X,Y)
```

```
[1m 1/24[0m [37m ━━━━━━━━━━━━━━━━━━━━━━━━[0m [1m3s[0m 150ms/
step - accuracy: 0.7812 - loss: 0.5488
[1m24/24[0m [32m ━━━━━━━━━━━━
━━━━━━━━━━━━[0m[37m[0m [1m0s[0m 2ms/step - accuracy: 0.7832 - los
s: 0.4714
```

```
print('Accuracy of model is',(accuracy*100))
```

```
Accuracy of model is 79.81770634651184
```

```
predict_X=model.predict(X)
```

```
[1m 1/24[0m [37m ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m [1m1s[0m 70ms/s
tep[1m24/24[0m [32m ━━━━━━━━
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m 2ms/step
>>>
```

Practical 4

A. Using deep feed forward network with two hidden layers for performing classification and predicting the class.

Code:

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

Output:

```
Epoch 500/500
▯[1m1/4▯[0m ▯[32m ━━━━━━━━▯[0m▯[37m ━━━━━━━━━━━━━━━━━━━━━━━━▯[0m ▯[1m0s▯[0m 32ms/
step - loss: 0.1066▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯[1m4/4▯
[0m ▯[32m ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━▯[0m▯[37m▯[0m ▯[1m0s▯[0m 4ms/step - los
s: 0.0928
▯[1m1/4▯[0m ▯[32m ━━━━━━━━▯[0m▯[37m ━━━━━━━━━━━━━━━━━━━━━━━━▯[0m ▯[1m0s▯[0m 66ms/
step▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯[1m4/4▯[0m ▯[32m ━━━━━━━━━━━━━━━━━━━━━
━━━━━━━━━━━▯[0m▯[37m▯[0m ▯[1m0s▯[0m 16ms/step
X=[0.89337759 0.65864154],Predicted=[0.00859113],Desired=0
X=[0.29097707 0.12978982],Predicted=[0.8368741],Desired=1
X=[0.78082614 0.75391697],Predicted=[0.00381127],Desired=0
>>>
```

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

Code:

```
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(X)
Ynew=model.predict(Xnew)
for i in range(len(Xnew)):
print("X=%s,Predicted_probability=%s,Predicted_class=%s"%(Xnew[i],Ynew[i],Yclass[i]))
```

Output:

```
Epoch 500/500
[1m1/4[0m [32m ━━━━━[0m[37m ━━━━━━━━━━━━━━━[0m [1m0s[0m 32ms/step - loss: 0.0047[1m4/4[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m 4ms/step - loss: 0.0045
[1m1/4[0m [32m ━━━━━[0m[37m ━━━━━━━━━━━━━━━[0m [1m0s[0m 48ms/step[1m4/4[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m 20ms/step[1m4/4[0m [32m ━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m 23ms/step
[1m1/1[0m [32m ━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m 26ms/step[1m1/1[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m 56ms/step
X=[0.89337759 0.65864154],Predicted_probability=[0.00085907],Predicted_class=[0.00085907]
X=[0.29097707 0.12978982],Predicted_probability=[0.99156004],Predicted_class=[0.99397004]
X=[0.78082614 0.75391697],Predicted_probability=[0.00253656],Predicted_class=[0.00578618]
>>>
```

C. **Using a deep field forward network with two hidden layers for performing linear regression and predicting values.**

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

Output:

```
▯[1m1/4▯[0m ▯[32m ━━━━━━━━━▯[0m▯[37m ━━━━━━━━━━━━━━━━━━━━━━━━━━━▯[0m ▯[1m0s▯[0m 55ms/
step▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯[1m4/4▯[0m ▯[32m ━━━━━━━━━━━━━━━━━━━━━━━━
━━━━━━━━━━━━━▯[0m▯[37m▯[0m ▯[1m0s▯[0m 23ms/step ▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯
▯▯▯▯[1m4/4▯[0m ▯[32m ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━▯[0m▯[37m▯[0m ▯[1m0s▯[0m 40
ms/step
X=[0.29466096 0.30317302],Predicted=[0.6097332]
X=[0.39445118 0.79390858],Predicted=[0.28695926]
X=[0.02884127 0.6208843 ],Predicted=[0.282664]
>>>
```

Practical 5

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

Code:

```
import pandas as pd
from keras.models import Sequential
from keras.layers import Dense
from scikeras.wrappers import KerasRegressor
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
dataframe=pd.read_csv('housing.csv',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=10,ba
tch_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:

```
Epoch 10/10
▯[1m 1/91▯[0m ▯[37m ━━━━━━━━━━━━━━━━━━━━▯[0m ▯[1m3s▯[0m 36ms/step -
loss: 107.7146▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯[1m36/91
▯[0m ▯[32m ━━━━━━━▯[0m▯[37m ━━━━━━━━━━━━━▯[0m ▯[1m0s▯[0m 1ms/step - lo
ss: 34.8920 ▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯[1m69/91▯[0
m ▯[32m ━━━━━━━━━━━━━━━▯[0m▯[37m ━━━━━▯[0m ▯[1m0s▯[0m 1ms/step - loss:
 31.1838▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯[1m91/91▯[0m ▯[3
2m━━━━━━━━━━━━━━━━━━━━▯[0m▯[37m▯[0m ▯[1m0s▯[0m 2ms/step - loss: 30.1
219
▯[1m 1/11▯[0m ▯[32m ━▯[0m▯[37m ━━━━━━━━━━━━━━━━━━━▯[0m ▯[1m0s▯[0m 53m
s/step▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯[1m11/11▯[0m ▯[32m ━━━━━━━━━
━━━━━━━━━━━▯[0m▯[37m▯[0m ▯[1m0s▯[0m 7ms/step ▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯
▯▯▯▯▯[1m11/11▯[0m ▯[32m ━━━━━━━━━━━━━━━━━━━━▯[0m▯[37m▯[0m ▯[1m
0s▯[0m 8ms/step
```

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

Code:

```
import pandas as pd
from keras.models import Sequential
from keras.layers import Dense
from scikeras.wrappers import KerasClassifier
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.preprocessing import LabelEncoder
#loading dataset
df=pd.read_csv('flowers.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=10,shuffle=True)
action=estimator.predict(X)
```

```
for i in range(25):
 print(dummy_Y[i])
print('^^^^^^^^^^^^^^^^^^^^^^')
for i in range(25):
 print(action[i])
```

Output:

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

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

Code 2:

```
import pandas as pd
import tensorflow as tf
from keras.models import Sequential
from keras.layers import Dense
from scikeras.wrappers import KerasClassifier
from tensorflow.keras.utils import to_categorical
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.preprocessing import LabelEncoder
dataset=pd.read_csv("flowers.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=tf.keras.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=['acc
uracy'])
   return model
estimator=KerasClassifier(build_fn=baseline_model,epochs=10,batch_size=5)
kfold = KFold(n_splits=1, shuffle=True)
results = cross_val_score(estimator, X, dummy_Y, cv=kfold)
print("Baseline: %.2f%% (%.2f%%)" % (results.mean()*100, results.std()*100))
```

Output:

```
['setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa'
 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa'
 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa'
 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa'
 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa'
 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa'
 'setosa' 'setosa' 'versicolor' 'versicolor' 'versicolor' 'versicolor'
 'versicolor' 'versicolor' 'versicolor' 'versicolor' 'versicolor'
 'versicolor' 'versicolor' 'versicolor' 'versicolor' 'versicolor'
 'versicolor' 'versicolor' 'versicolor' 'versicolor' 'versicolor'
 'versicolor' 'versicolor' 'versicolor' 'versicolor' 'versicolor'
 'versicolor' 'versicolor' 'versicolor' 'versicolor' 'versicolor'
 'versicolor' 'versicolor' 'versicolor' 'versicolor' 'versicolor'
 'versicolor' 'versicolor' 'versicolor' 'versicolor' 'versicolor'
 'versicolor' 'versicolor' 'versicolor' 'versicolor' 'versicolor'
 'versicolor' 'versicolor' 'versicolor' 'versicolor' 'versicolor'
 'versicolor' 'virginica' 'virginica' 'virginica' 'virginica' 'virginica'
 'virginica' 'virginica' 'virginica' 'virginica' 'virginica' 'virginica'
 'virginica' 'virginica' 'virginica' 'virginica' 'virginica' 'virginica'
 'virginica' 'virginica' 'virginica' 'virginica' 'virginica' 'virginica'
 'virginica' 'virginica' 'virginica' 'virginica' 'virginica' 'virginica'
 'virginica' 'virginica' 'virginica' 'virginica' 'virginica' 'virginica'
 'virginica' 'virginica' 'virginica' 'virginica' 'virginica' 'virginica'
 'virginica' 'virginica' 'virginica' 'virginica' 'virginica' 'virginica'
 'virginica' 'virginica' 'virginica']
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 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 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]

▯▯▯▯▯▯▯▯[1m15/15▯[0m ▯[32m
0m 6ms/step
Baseline: 28.67% (7.33%)
>>>
```

Practical 6

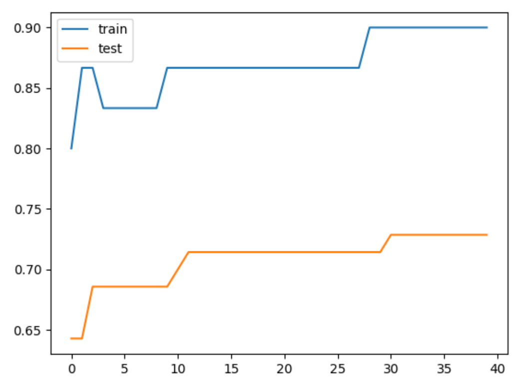
Implementing regularization to avoid overfitting in binary classification.

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:]
#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=['accurac
y'])
history=model.fit(trainX,trainY,validation_data=(testX,testY),epochs=40)
pyplot.plot(history.history['accuracy'],label='train')
pyplot.plot(history.history['val_accuracy'],label='test')
pyplot.legend()
pyplot.show()
```

Output:

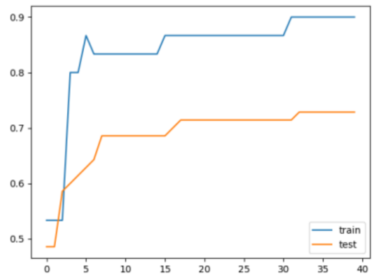
```
Epoch 40/40
[1m1/1[0m [32m ━━━━━━━━━━━━━━━━━━━━ [0m[37m[0m [1m0s[0m 36ms/
step - accuracy: 0.9000 - loss: 0.2972
[1m1/1[0m [32m ━━━━━━━━━━━━━━━━━━━━
[0m[37m[0m [1m0s[0m 76ms/step - accuracy: 0.9000 - loss: 0.2972 - val_accura
cy: 0.7286 - val_loss: 0.4723
```



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(0.0
01)))
model.add(Dense(1,activation='sigmoid'))
model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accurac
y'])
history=model.fit(trainX,trainY,validation_data=(testX,testY),epochs=40)
pyplot.plot(history.history['accuracy'],label='train')
pyplot.plot(history.history['val_accuracy'],label='test')
pyplot.legend()
pyplot.show()
```

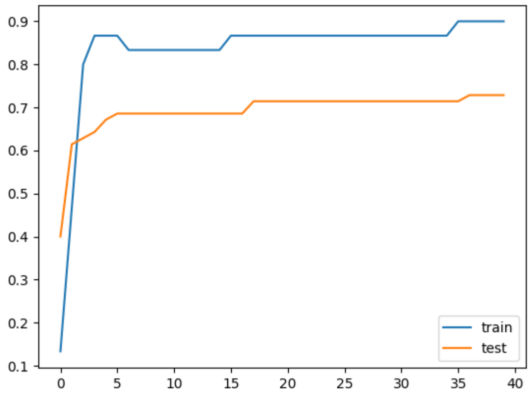
```
Epoch 40/40
▯[1m1/1▯[0m ▯[32m ━━━━━━━━━━━━━━━━━━━━ ▯[0m▯[37m▯[0m ▯[1m0s▯[0m 36ms/
step - accuracy: 0.9000 - loss: 0.3152▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯
▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯[1m1/1▯[0m ▯[32m ━━━━━━━━━━━━━━━━━━━━▯
[0m▯[37m▯[0m ▯[1m0s▯[0m 87ms/step - accuracy: 0.9000 - loss: 0.3152 - val_accura
cy: 0.7286 - val_loss: 0.4828
```



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

```
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
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(0.0
01)))
model.add(Dense(1,activation='sigmoid'))
model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accurac
y'])
history=model.fit(trainX,trainY,validation_data=(testX,testY),epochs=40)
pyplot.plot(history.history['accuracy'],label='train')
pyplot.plot(history.history['val_accuracy'],label='test')
pyplot.legend()
pyplot.show()
```

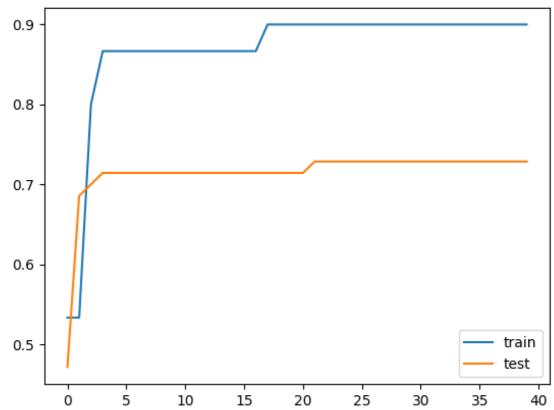
```
Epoch 40/40
[1m1/1[0m [32m ━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m 42ms   /
step - accuracy: 0.9000 - loss: 0.3776
[1m1/1[0m [32m ━━━━━━━━━━━━━━━━━━━━
[0m[37m[0m [1m0s[0m 83ms/step - accuracy: 0.9000 - loss: 0.3776 - val_accura
cy: 0.7286 - val_loss: 0.5415
```

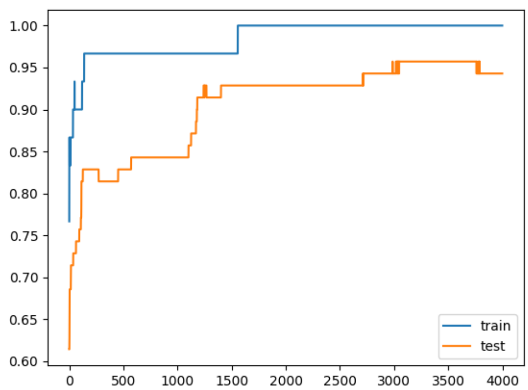


By applying l1 and l2 regularizer we can observe the following changes in accuracy of both training and testing data.

```
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=40)
pyplot.plot(history.history['accuracy'],label='train')
pyplot.plot(history.history['val_accuracy'],label='test')
pyplot.legend()
pyplot.show()
```

```
Epoch 40/40
[1m1/1[0m [32m ━━━━━━━━━━━━━━━━━━━━ [0m[37m[0m [1m0s[0m 33ms/
step - accuracy: 0.9000 - loss: 0.3727
[1m1/1[0m [32m ━━━━━━━━━━━━━━━━━━━━
[0m[37m[0m [1m0s[0m 76ms/step - accuracy: 0.9000 - loss: 0.3727 - val_accura
cy: 0.7286 - val_loss: 0.5316
```





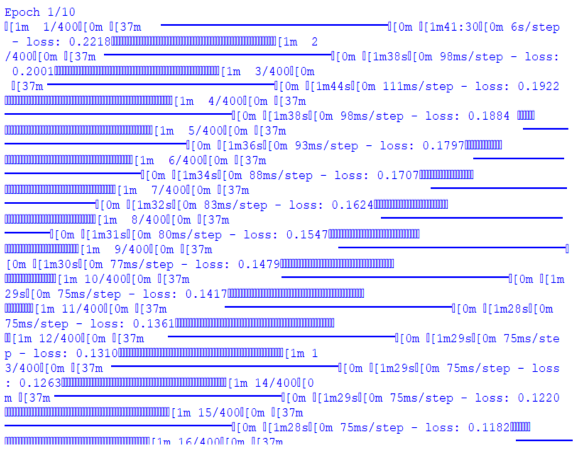
Practical 7

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

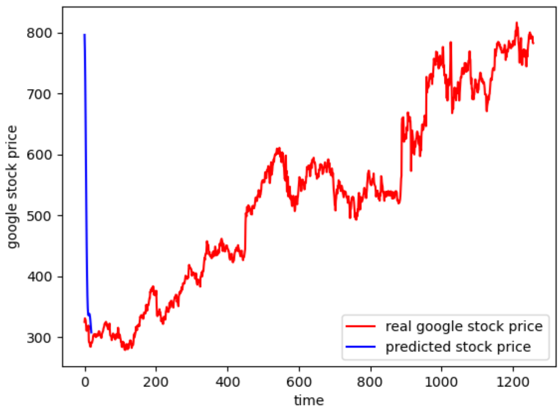
Code:

```
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('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.sh
ape[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=10,batch_size=3)
dataset_test=pd.read_csv('Google_Stock_Price_Train.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()
```



```
Epoch 1/10
[1m  1/400[0m [37m ──────────[0m [1m41:30[0m 6s/step
 - loss: 0.2218[1m  2
/400[0m [37m ──────────[0m [1m38s[0m 98ms/step - loss:
 0.2001[1m  3/400[0m
 [37m ──────────[0m [1m44s[0m 111ms/step - loss: 0.1922
[1m  4/400[0m [37m
──────────[0m [1m38s[0m 98ms/step - loss: 0.1884
[1m  5/400[0m [37m
──────────[0m [1m36s[0m 93ms/step - loss: 0.1797
[1m  6/400[0m [37m
──────────[0m [1m34s[0m 88ms/step - loss: 0.1707
[1m  7/400[0m [37m
──────────[0m [1m32s[0m 83ms/step - loss: 0.1624
[1m  8/400[0m [37m
──────────[0m [1m31s[0m 80ms/step - loss: 0.1547
[1m  9/400[0m [37m ──────────
[0m [1m30s[0m 77ms/step - loss: 0.1479
[1m 10/400[0m [37m ──────────[0m [1m
29s[0m 75ms/step - loss: 0.1417
[1m 11/400[0m [37m ──────────[0m [1m28s[0m
75ms/step - loss: 0.1361
[1m 12/400[0m [37m ──────────[0m [1m29s[0m 75ms/ste
p - loss: 0.1310[1m 1
3/400[0m [37m ──────────[0m [1m29s[0m 75ms/step - loss
: 0.1263[1m 14/400[0
m [37m ──────────[0m [1m29s[0m 75ms/step - loss: 0.1220
[1m 15/400[0m [37m
──────────[0m [1m28s[0m 75ms/step - loss: 0.1182
[1m 16/400[0m [37m
```



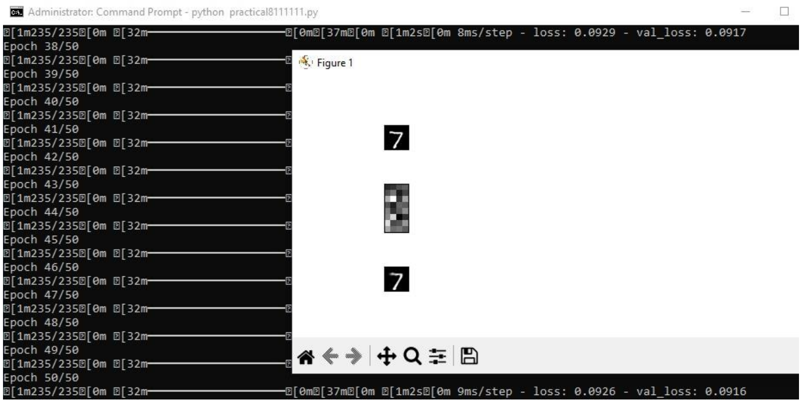
Practical 8

Performing encoding and decoding of images using deep autoencoder.

```
import keras
from keras import layers
from keras.datasets import mnist
import numpy as np
encoding_dim=32
#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=1,
 batch_size=2,
 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:



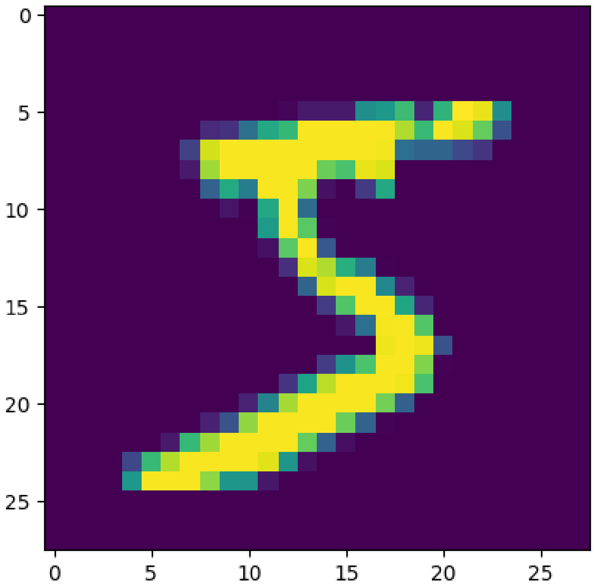
Practical 9

Implementation of convolutional neural network to predict numbers from number images

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=['acc
uracy'])
#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:


0
5
10
15
20
25
0
5
10
15
20
25

Practical 10

Denoising of images using autoencoder.

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.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=Fals
e)])
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: