DEEP LEARNING

Certified Journal

Submitted in partial fulfilment of the

Requirements for the award of the Degree of

MASTER OF SCIENCE

(INFORMATION_TECHNOLOGY)

By

Anjali Rameshwar Nimje

DEPARTMENT OF INFORMATION TECHNOLOGY

KERALEEYA SAMAJAM (REGD.) DOMBIVLI'S

MODEL COLLEGE (AUTONOMOUS)

Re-Accredited 'A' Grade by NAAC

(Affiliated to University of Mumbai)

FOR THE YEAR

(2023-24)

Keraleeya Samajam(Regd.) Dombivli's

MODEL COLLEGE

Re-Accredited Grade "A" by NAAC

Kanchan Goan Village, Khambalpada, Thakurli East – 421201
Contact No – 7045682157, 7045682158. www.model-college.edu.in

DEPARTMENT OF INFORMATION TECHNOLOGY AND COMPUTER SCIENCE

CERTIFICATE

This is to certify that Mr. /Miss ______________________________

Studying in Class _______________ *Seat No.* ______________

Has completed the prescribed practicals in the subject _______________

During the academic year ______________________________

Date : ____________

External Examiner

Internal Examiner
M.Sc. Information Technology

INDEX

Sr No.	PRACTICAL NAME	DATE	SIGNATURE
1	**Performing matrix multiplication and finding eigen vectors and eigen values using TensorFlow**	**20/04/2024**	
2	**Solving XOR problem using deep feed forward network.**	**04/05/2024**	
3	**Implementing deep neural network for performing classification task.**	**04/05/2024**	
4A	**Using deep feed forward network with two hidden layers for performing classification and predicting the class.**	**08/06/2024**	
4B	**Using a deep field forward network with two hidden layers for performing classification and predicting the probability of class.**	**08/06/2024**	
4C	**Using a deep field forward network with two hidden layers for performing linear regression and predicting values.**	**08/06/2024**	
5A	**Evaluating feed forward deep network for regression using KFold cross validation.**	**08/06/2024**	
5B	**Evaluating Feed Forward Deep Network For Multiclass Classification Using Kfold Cross-Validation.**	**22/06/2024**	
6	**Implementing regularization to avoid overfitting in binary classification**	**22/06/2024**	

7	**Demonstrate recurrent neural network that learns to perform sequence analysis for stock price.**	**22/06/2024**	
8	**Performing encoding and decoding of images using deep autoencoder**	**22/06/2024**	
9	**Implementation of convolutional neural network to predict numbers from number images**	**22/06/2024**	
10	**Denoising of images using autoencoder.**	**22/06/2024**	

PRACTICAL 1

Performing matrix multiplication and finding eigen vectors and eigen values using TensorFlow

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:

```
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:
[[6.028684  6.8949666]
 [4.5444255 5.41298  ]]


Eigen Vectors:
[[-0.6827928  0.730612 ]
 [ 0.730612   0.6827928]]

Eigen Values:
[ 1.1659912 10.27567  ]
```

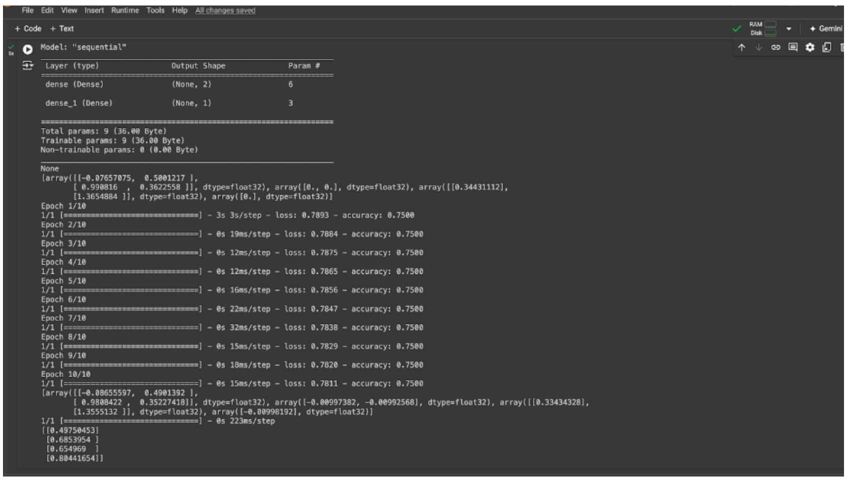
PRACTICAL 2

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=['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=10,batch_size=4)
print(model.get_weights())
print(model.predict(X,batch_size=4))
```

Output:



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 2)                 6

 dense_1 (Dense)             (None, 1)                 3

=================================================================
Total params: 9 (36.00 Byte)
Trainable params: 9 (36.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
[array([[-0.07657075,  0.5001217 ],
       [ 0.990816  ,  0.3622558 ]], dtype=float32), array([0., 0.], dtype=float32), array([[0.34431112],
       [1.3654884 ]], dtype=float32), array([0.], dtype=float32)]
Epoch 1/10
1/1 [==============================] - 3s 3s/step - loss: 0.7893 - accuracy: 0.7500
Epoch 2/10
1/1 [==============================] - 0s 19ms/step - loss: 0.7884 - accuracy: 0.7500
Epoch 3/10
1/1 [==============================] - 0s 12ms/step - loss: 0.7875 - accuracy: 0.7500
Epoch 4/10
1/1 [==============================] - 0s 12ms/step - loss: 0.7865 - accuracy: 0.7500
Epoch 5/10
1/1 [==============================] - 0s 16ms/step - loss: 0.7856 - accuracy: 0.7500
Epoch 6/10
1/1 [==============================] - 0s 22ms/step - loss: 0.7847 - accuracy: 0.7500
Epoch 7/10
1/1 [==============================] - 0s 32ms/step - loss: 0.7838 - accuracy: 0.7500
Epoch 8/10
1/1 [==============================] - 0s 15ms/step - loss: 0.7829 - accuracy: 0.7500
Epoch 9/10
1/1 [==============================] - 0s 18ms/step - loss: 0.7820 - accuracy: 0.7500
Epoch 10/10
1/1 [==============================] - 0s 15ms/step - loss: 0.7811 - accuracy: 0.7500
[array([[-0.08655597,  0.4901392 ],
       [ 0.9808422 ,  0.35227418]], dtype=float32), array([-0.00997382, -0.00992568], dtype=float32), array([[0.33434328],
       [1.3555132 ]], dtype=float32), array([-0.00998192], dtype=float32)]
1/1 [==============================] - 0s 223ms/step
[[0.49750453]
 [0.6853954 ]
 [0.654969  ]
 [0.80441654]]
```

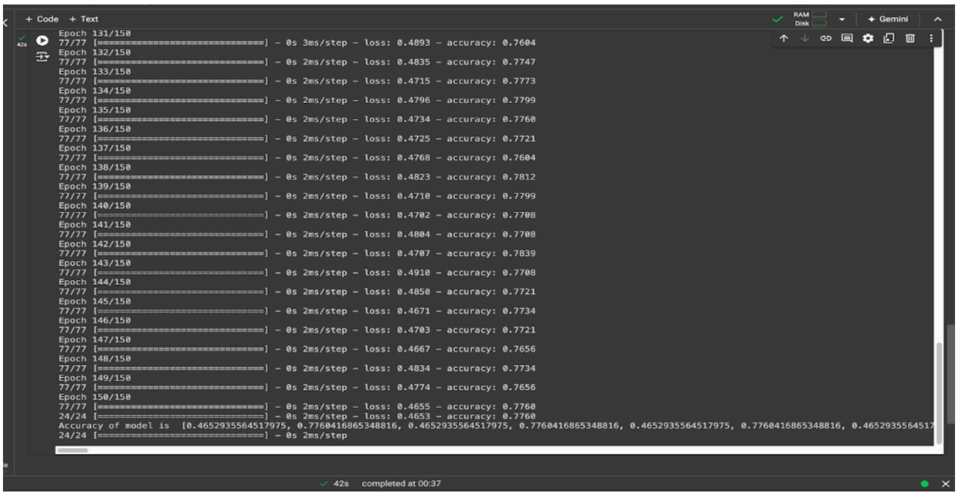
PRACTICAL 3

Implementing deep neural network for performing classification task.

Code:

```
from numpy import loadtxt
import pandas as pd
from keras.models import Sequential
from keras.layers import Dense
dataset=pd.read_csv('diabetes.csv')
X = dataset.iloc[:,:-1]
Y = dataset.iloc[:,-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=['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)
```

Output:



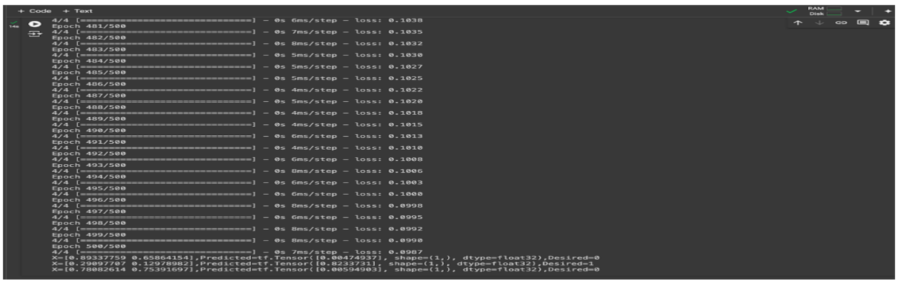
PRACTICAL 4A

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

Output:



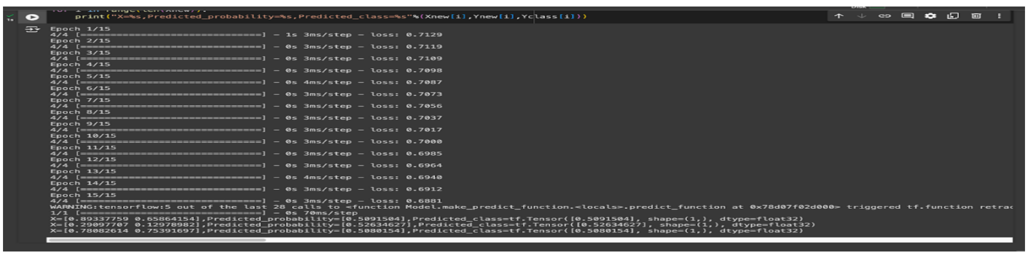
PRACTICAL 4B

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=15)
Xnew,Yreal=make_blobs(n_samples=3,centers=2,n_features=2,random_state=1)
Xnew=scalar.transform(Xnew)
Yclass=model.predict_step(Xnew)
Ynew=model.predict(Xnew)
for i in range(len(Xnew)):
  print("X=%s,Predicted_probability=%s,Predicted_class=%s"%(Xnew[i],Ynew[i],Yclass[i])
)
```

Output:



PRACTICAL 4C

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=200,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:

```
1/1 [==============================] - 0s 59ms/step
X=[0.29466096 0.30317302],Predicted=[0.29197517]
X=[0.39445118 0.79390858],Predicted=[0.6969597]
X=[0.02884127 0.6208843 ],Predicted=[0.4817084]
```

PRACTICAL 5A

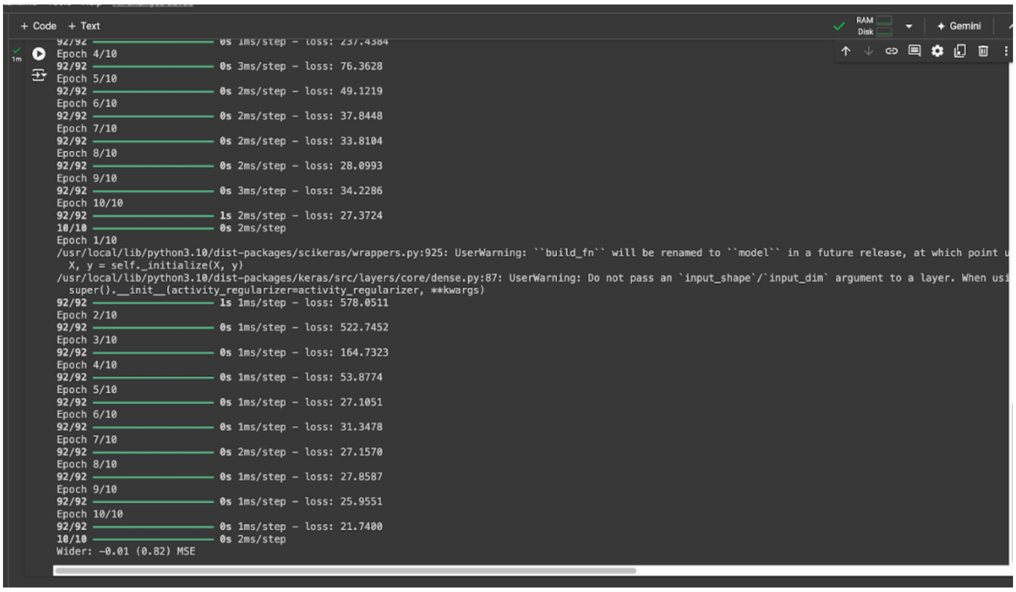
Evaluating feed forward deep network for regression using KFold cross validation.

Code:

```
#!pip install scikeras
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("hoousing.csv", sep='\s+',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,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:



PRACTICAL 5B

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

Code:

```
import pandas
from keras.models import Sequential
from keras.layers import Dense
from scikeras.wrappers import KerasClassifier  # Import from scikeras
from tensorflow.keras.utils import to_categorical  # Import to_categorical directly
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.preprocessing import LabelEncoder
# ... rest of your code ...
#loading dataset
df=pandas.read_csv('flowers.csv',header=None)
print(df)
#splitting dataset into input and output variables
# The first row is treated as data, but it contains headers.
# We should skip the first row when selecting data for X.
X = df.iloc[1:,0:4].astype(float)  # Start from the second row (index 1)
y=df.iloc[1:,4]  # Start from the second row for y as well
#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= to_categorical(encoded_y) # Use to_categorical directly
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:

PRACTICAL 6

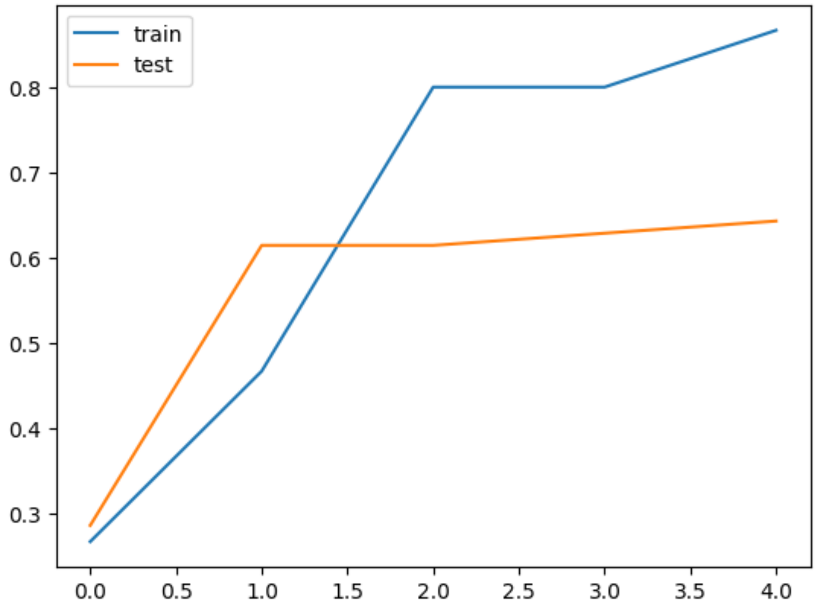
Implementing regularization to avoid overfitting in binary classification

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=['accuracy'])
history=model.fit(trainX,trainY,validation_data=(testX,testY),epochs=5)
pyplot.plot(history.history['accuracy'],label='train')
pyplot.plot(history.history['val_accuracy'],label='test')
pyplot.legend()
pyplot.show()
```

Output:

```
Epoch 1/5
1/1 [==============================] - 3s 3s/step - loss: 0.7098 - accuracy: 0.2667 - val_loss: 0.7008 - val_accuracy: 0.2857
Epoch 2/5
1/1 [==============================] - 0s 96ms/step - loss: 0.6930 - accuracy: 0.4667 - val_loss: 0.6899 - val_accuracy: 0.6143
Epoch 3/5
1/1 [==============================] - 0s 118ms/step - loss: 0.6766 - accuracy: 0.8000 - val_loss: 0.6795 - val_accuracy: 0.6143
Epoch 4/5
1/1 [==============================] - 0s 226ms/step - loss: 0.6607 - accuracy: 0.8000 - val_loss: 0.6694 - val_accuracy: 0.6286
Epoch 5/5
1/1 [==============================] - 0s 214ms/step - loss: 0.6452 - accuracy: 0.8667 - val_loss: 0.6597 - val_accuracy: 0.6429
```



train
test
0.8
0.7
0.6
0.5
0.4
0.3
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0

PRACTICAL 7

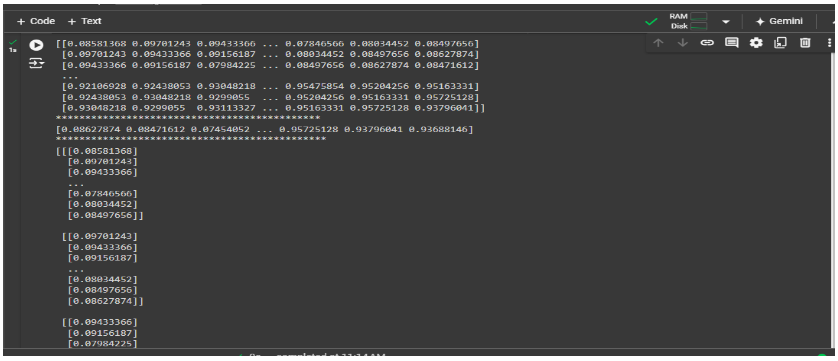
Demonstrate recurrent neural network that learns to perform sequence analysis for stock price.

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):
    # Indent the lines within the for loop
    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
```

Output:



PRACTICAL 8

Performing encoding and decoding of images using deep autoencoder

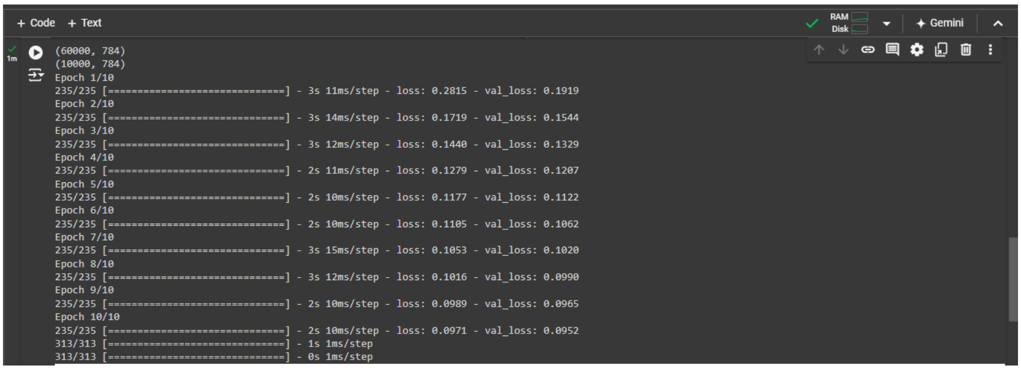
Code:

```
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.
```

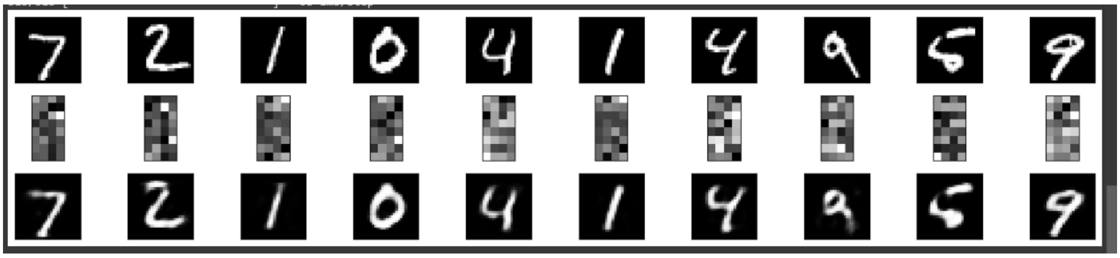
```
X_test=X_test.astype('float32')/255.
X_train=X_train.reshape((len(X_train),np.prod(X_train.shape[1:])))
X_test=X_test.reshape((len(X_test),np.prod(X_test.shape[1:])))
print(X_train.shape)
print(X_test.shape)
#train autoencoder with training dataset
autoencoder.fit(X_train,X_train,
epochs=10,
batch_size=256,
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) # Indent the code block within the for loop
    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) # Indent the code block within the for loop
    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) # Indent the code block within the for loop
    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:




```
(60000, 784)
(10000, 784)
Epoch 1/10
235/235 [==============================] - 3s 11ms/step - loss: 0.2815 - val_loss: 0.1919
Epoch 2/10
235/235 [==============================] - 3s 14ms/step - loss: 0.1719 - val_loss: 0.1544
Epoch 3/10
235/235 [==============================] - 3s 12ms/step - loss: 0.1440 - val_loss: 0.1329
Epoch 4/10
235/235 [==============================] - 2s 11ms/step - loss: 0.1279 - val_loss: 0.1207
Epoch 5/10
235/235 [==============================] - 2s 10ms/step - loss: 0.1177 - val_loss: 0.1122
Epoch 6/10
235/235 [==============================] - 2s 10ms/step - loss: 0.1105 - val_loss: 0.1062
Epoch 7/10
235/235 [==============================] - 3s 15ms/step - loss: 0.1053 - val_loss: 0.1020
Epoch 8/10
235/235 [==============================] - 3s 12ms/step - loss: 0.1016 - val_loss: 0.0990
Epoch 9/10
235/235 [==============================] - 2s 10ms/step - loss: 0.0989 - val_loss: 0.0965
Epoch 10/10
235/235 [==============================] - 2s 10ms/step - loss: 0.0971 - val_loss: 0.0952
313/313 [==============================] - 1s 1ms/step
313/313 [==============================] - 0s 1ms/step
```



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=['accuracy'])

#train

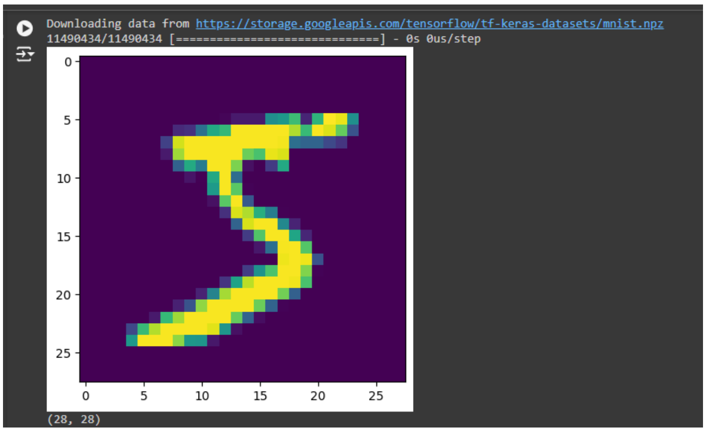
model.fit(X_train,Y_train,validation_data=(X_test,Y_test),epochs=3)

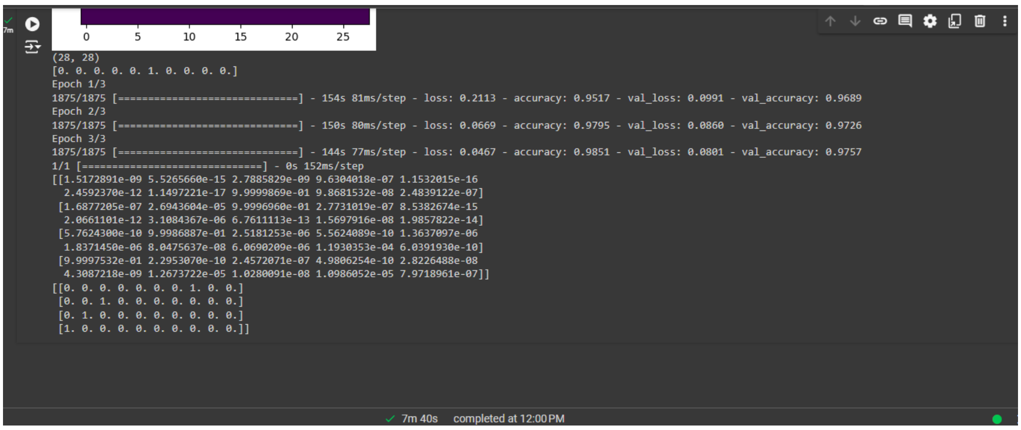
print(model.predict(X_test[:4]))

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

print(Y_test[:4])

Output:





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) # Indent this line
    plt.imshow(X_test_noisy[i].reshape(28,28)) # Indent this line
    plt.gray() # Indent this line
```

```
    ax.get_xaxis().set_visible(False) # Indent this line
    ax.get_yaxis().set_visible(False) # Indent this line
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) # Indent this line
    plt.imshow(predictions[i].reshape(28,28)) # Indent this line
```

```
    plt.gray() # Indent this line
    ax.get_xaxis().set_visible(False) # Indent this line
    ax.get_yaxis().set_visible(False) # Indent this line
plt.show()
```

Output: