Deep Learning

**Index**

|  |  |  |  |
| --- | --- | --- | --- |
| **Sr. No** | **Title** | **Date** | **Sign** |
| 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. | Evaluating feed forward deep network for regression using KFold cross validation. |  |  |
| 6. | Implementing regularization to avoid overfitting in binary classification. |  |  |
| 7. | Demonstrate recurrent neural network that learns to perform sequence analysis. |  |  |
| 8. | Performing encoding & decoding of images 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.

**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)

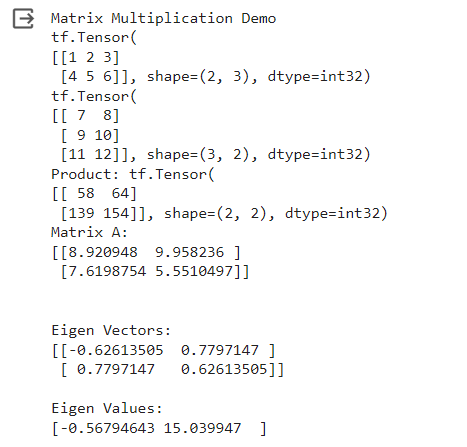
e\_matrix\_A=tf.random.uniform([2,2],minval=3,maxval=10,dtype=tf.float32,name="matrixA")

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:**



**Practical No: 2**

**Aim:** Solving XOR problem using deep feed forward network.

**Code:**

import numpy as np

def unitStep(v):

if v >= 0:

return 1

else:

return 0

def perceptronModel(x, w, b):

v = np.dot(w, x) + b

y = unitStep(v)

return y

def NOT\_logicFunction(x):

wNOT = -1

bNOT = 0.5

return perceptronModel(x, wNOT, bNOT)

def AND\_logicFunction(x):

w = np.array([1, 1])

bAND = -1.5

return perceptronModel(x, w, bAND)

def OR\_logicFunction(x):

w = np.array([1, 1])

bOR = -0.5

return perceptronModel(x, w, bOR)

def XOR\_logicFunction(x):

y1 = AND\_logicFunction(x)

y2 = OR\_logicFunction(x)

y3 = NOT\_logicFunction(y1)

final\_x = np.array([y2, y3])

finalOutput = AND\_logicFunction(final\_x)

y3 = NOT\_logicFunction(y1)

return finalOutput

test1 = np.array([0, 1])

test2 = np.array([1, 1])

test3 = np.array([0, 0])

test4 = np.array([1, 0])

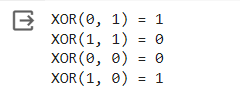
print("XOR({}, {}) = {}".format(0, 1, XOR\_logicFunction(test1)))

print("XOR({}, {}) = {}".format(1, 1, XOR\_logicFunction(test2)))

print("XOR({}, {}) = {}".format(0, 0, XOR\_logicFunction(test3)))

print("XOR({}, {}) = {}".format(1, 0, XOR\_logicFunction(test4)))

**Output:**



**Practical No: 3**

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

**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.preprocessing import LabelEncoder

from sklearn.model\_selection import StratifiedKFold

from sklearn.preprocessing import StandardScaler

from sklearn.pipeline import Pipeline

# load dataset

dataframe = pd.read\_csv("//content//sonar.all-data", header=None)

dataset = dataframe.values

# split into input (X) and output (Y) variables

X = dataset[:,0:60].astype(float)

Y = dataset[:,60]

# encode class values as integers

encoder = LabelEncoder()

encoder.fit(Y)

encoded\_Y = encoder.transform(Y)

# baseline model

def create\_baseline():

# create model

model = Sequential()

model.add(Dense(60, input\_dim=60, activation='relu'))

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

# Compile model

model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])

return model

# evaluate model with standardized dataset

estimator = KerasClassifier(build\_fn=create\_baseline, epochs=100, batch\_size=5, verbose=0)

kfold = StratifiedKFold(n\_splits=10, shuffle=True)

results = cross\_val\_score(estimator, X, encoded\_Y, cv=kfold)

print("Baseline: %.2f%% (%.2f%%)" % (results.mean()\*100, results.std()\*100))

# evaluate baseline model with standardized dataset

estimators = []

estimators.append(('standardize', StandardScaler()))

estimators.append(('mlp', KerasClassifier(build\_fn=create\_baseline, epochs=100, batch\_size=5, verbose=0)))

pipeline = Pipeline(estimators)

kfold = StratifiedKFold(n\_splits=10, shuffle=True)

results = cross\_val\_score(pipeline, X, encoded\_Y, cv=kfold)

print("Standardized: %.2f%% (%.2f%%)" % (results.mean()\*100, results.std()\*100))

def create\_smaller():

# create model

model = Sequential()

model.add(Dense(30, input\_dim=60, activation='relu'))

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

model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])

return model

estimators = []

estimators.append(('standardize', StandardScaler()))

estimators.append(('mlp', KerasClassifier(build\_fn=create\_smaller, epochs=100, batch\_size=5, verbose=0)))

pipeline = Pipeline(estimators)

kfold = StratifiedKFold(n\_splits=10, shuffle=True)

results = cross\_val\_score(pipeline, X, encoded\_Y, cv=kfold)

print("Smaller: %.2f%% (%.2f%%)" % (results.mean()\*100, results.std()\*100))

# larger model

def create\_larger():

# create model

model = Sequential()

model.add(Dense(60, input\_dim=60, activation='relu'))

model.add(Dense(30, activation='relu'))

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

# Compile model

model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])

return model

estimators = []

estimators.append(('standardize', StandardScaler()))

estimators.append(('mlp', KerasClassifier(build\_fn=create\_larger, epochs=100, batch\_size=5, verbose=0)))

pipeline = Pipeline(estimators)

kfold = StratifiedKFold(n\_splits=10, shuffle=True)

results = cross\_val\_score(pipeline, X, encoded\_Y, cv=kfold)

print("Larger: %.2f%% (%.2f%%)" % (results.mean()\*100, results.std()\*100))

**Output:**

Baseline: 82.69% (9.24%)

Standardized: 87.52% (7.73%)

Smaller: 83.12% (5.11%)

Larger: 86.10% (7.49%)

**Practical No: 4**

**Aim:** A]Using deep feed forward network with two hidden layers for performing multiclass 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.summary()

model.fit(X,Y,epochs=200)

Xnew,Yreal=make\_blobs(n\_samples=3,centers=2,n\_features=2,random\_state=1)

Xnew=scalar.transform(Xnew)

Yclass=model.predict(Xnew)

import numpy as np

def predict\_prob(number):

return [number[0],1-number[0]]

y\_prob = np.array(list(map(predict\_prob, model.predict(Xnew))))

y\_prob

for i in range(len(Xnew)):

print("X=%s,Predicted\_probability=%s,Predicted\_class=%s"%(Xnew[i],y\_prob[i],Yclass[i]))

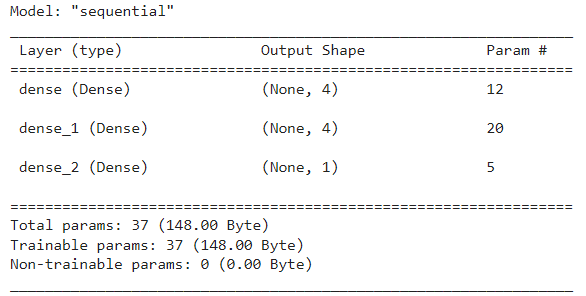
#second way

predict\_prob=model.predict([Xnew])

predict\_classes=np.argmax(predict\_prob,axis=1)

predict\_classes

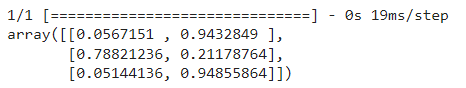
**Output:**

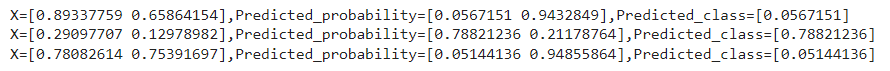














**Aim:** B]Using a deep feed forward network with two hidden layers for performing classification and predicting the probability of class.

**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)

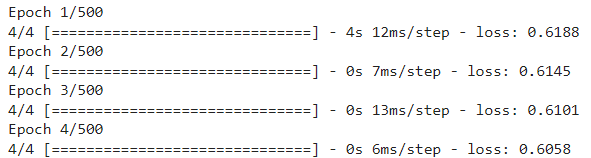
Xnew=scalar.transform(Xnew)

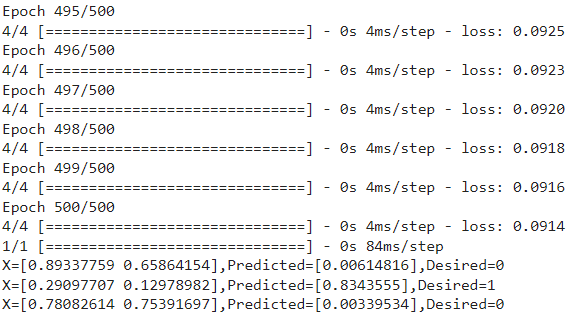
Ynew=model.predict(Xnew)

for i in range(len(Xnew)):

print("X=%s,Predicted=%s,Desired=%s"%(Xnew[i],Ynew[i],Yreal[i]))

**Output:**





**Aim:** C]Using a deep feed 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)

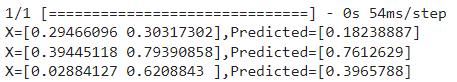
Xnew=scalarX.transform(Xnew)

Ynew=model.predict(Xnew)

for i in range(len(Xnew)):

print("X=%s,Predicted=%s"%(Xnew[i],Ynew[i]))

**Output:**



**Practical No: 5**

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

**Code:**

from tensorflow.keras.datasets import cifar10

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPooling2D

from tensorflow.keras.losses import sparse\_categorical\_crossentropy

from tensorflow.keras.optimizers import Adam

import matplotlib.pyplot as plt

# Model configuration

batch\_size = 50

img\_width, img\_height, img\_num\_channels = 32, 32, 3

loss\_function = sparse\_categorical\_crossentropy

no\_classes = 100

no\_epochs = 10 # you can increase it to 20,50,70, 100

optimizer = Adam()

verbosity = 1

# Load CIFAR-10 data

(input\_train, target\_train), (input\_test, target\_test) = cifar10.load\_data()

# Determine shape of the data

input\_shape = (img\_width, img\_height, img\_num\_channels)

# Parse numbers as floats

input\_train = input\_train.astype('float32')

input\_test = input\_test.astype('float32')

# Normalize data

input\_train = input\_train / 255

input\_test = input\_test / 255

# Create the model

model = Sequential()

model.add(Conv2D(32, kernel\_size=(3, 3), activation='relu', input\_shape=input\_shape))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Conv2D(64, kernel\_size=(3, 3), activation='relu'))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Flatten())

model.add(Dense(256, activation='relu'))

model.add(Dense(128, activation='relu'))

model.add(Dense(no\_classes, activation='softmax'))

model.summary()

# Compile the model

model.compile(loss=loss\_function, optimizer=optimizer,metrics=['accuracy'])

# Fit data to model (this will take little time to train)

history = model.fit(input\_train, target\_train, batch\_size=batch\_size, epochs=no\_epochs, verbose=verbosity)

# Generate generalization metrics

score = model.evaluate(input\_test, target\_test, verbose=0)

print(f'Test loss: {score[0]} / Test accuracy: {score[1]}')

# Visualize history

# Plot history: Loss

plt.plot(history.history['loss'])

plt.title('Validation loss history')

plt.ylabel('Loss value')

plt.xlabel('No. epoch')

plt.show()

# Plot history: Accuracy

plt.plot(history.history['accuracy'])

plt.title('Validation accuracy history')

plt.ylabel('Accuracy value (%)')

plt.xlabel('No. epoch')

plt.show()

# By Adding k fold cross validation

from tensorflow.keras.datasets import cifar10

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPooling2D

from tensorflow.keras.losses import sparse\_categorical\_crossentropy

from tensorflow.keras.optimizers import Adam

from sklearn.model\_selection import KFold

KFold

import numpy as np

# Model configuration

batch\_size = 50

img\_width, img\_height, img\_num\_channels = 32, 32, 3

loss\_function = sparse\_categorical\_crossentropy

no\_classes = 100

no\_epochs = 10

optimizer = Adam()

verbosity = 1

num\_folds = 5

# Load CIFAR-10 data

(input\_train, target\_train), (input\_test, target\_test) = cifar10.load\_data()

# Determine shape of the data

input\_shape = (img\_width, img\_height, img\_num\_channels)

# Parse numbers as floats

input\_train = input\_train.astype('float32')

input\_test = input\_test.astype('float32')

# Normalize data

input\_train = input\_train / 255

input\_test = input\_test / 255

# Define per-fold score containers

acc\_per\_fold = []

loss\_per\_fold = []

# Merge inputs and targets

inputs = np.concatenate((input\_train, input\_test), axis=0)

targets = np.concatenate((target\_train, target\_test), axis=0)

# Define the K-fold Cross Validator

kfold = KFold(n\_splits=num\_folds, shuffle=True)

# K-fold Cross Validation model evaluation

fold\_no = 1

for train, test in kfold.split(inputs, targets):

# Define the model architecture

model = Sequential()

model.add(Conv2D(32, kernel\_size=(3, 3), activation='relu', input\_shape=input\_shape))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Conv2D(64, kernel\_size=(3, 3), activation='relu'))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Flatten())

model.add(Dense(256, activation='relu'))

model.add(Dense(128, activation='relu'))

model.add(Dense(no\_classes, activation='softmax'))

# Compile the model

model.compile(loss=loss\_function,

optimizer=optimizer,

metrics=['accuracy'])

# Generate a print

print('------------------------------------------------------------------------')

print(f'Training for fold {fold\_no} ...')

# Fit data to model

history = model.fit(inputs[train], targets[train],

batch\_size=batch\_size,

epochs=no\_epochs,

verbose=verbosity)

# Generate generalization metrics

scores = model.evaluate(inputs[test], targets[test], verbose=0)

print(f'Score for fold {fold\_no}: {model.metrics\_names[0]} of {scores[0]}; {model.metrics\_names[1]} of {scores[1]\*100}%')

acc\_per\_fold.append(scores[1] \* 100)

loss\_per\_fold.append(scores[0])

# Increase fold number

fold\_no = fold\_no + 1

# == Provide average scores ==

print('------------------------------------------------------------------------')

print('Score per fold')

for i in range(0, len(acc\_per\_fold)):

print('------------------------------------------------------------------------')

print(f'> Fold {i+1} - Loss: {loss\_per\_fold[i]} - Accuracy: {acc\_per\_fold[i]}%')

print('------------------------------------------------------------------------')

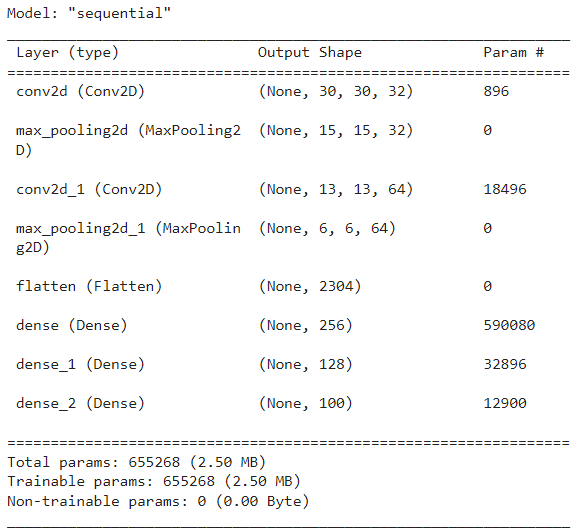
print('Average scores for all folds:')

print(f'> Accuracy: {np.mean(acc\_per\_fold)} (+- {np.std(acc\_per\_fold)})')

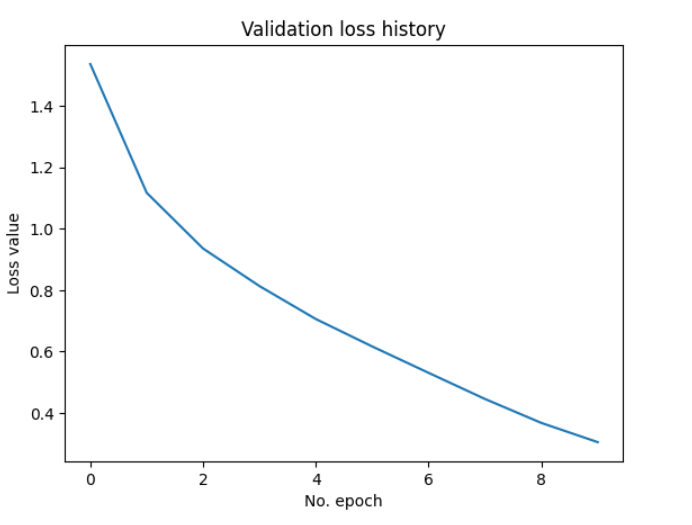
print(f'> Loss: {np.mean(loss\_per\_fold)}')

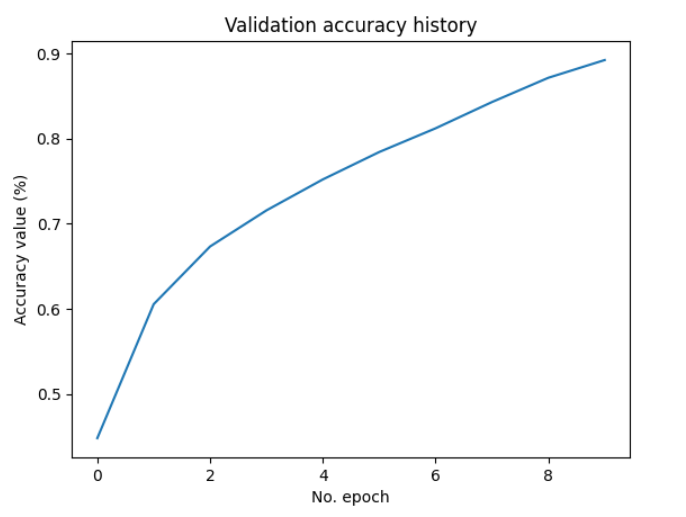
print('------------------------------------------------------------------------')

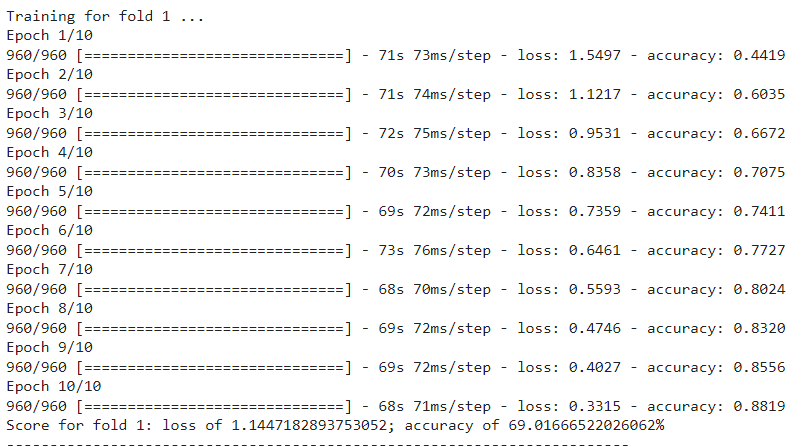
**Output:**

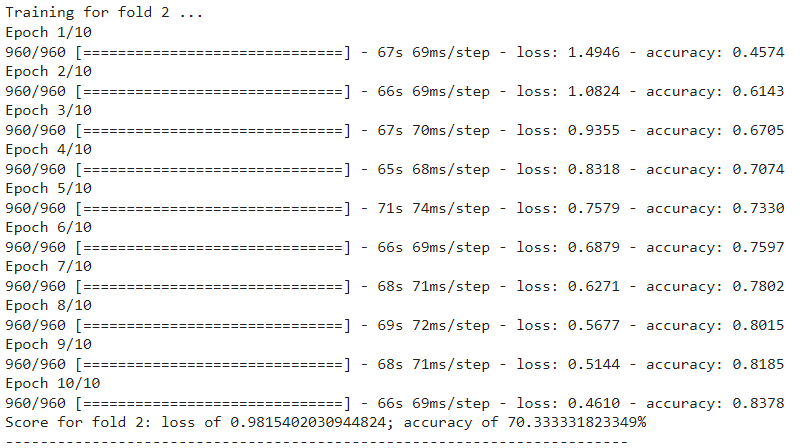


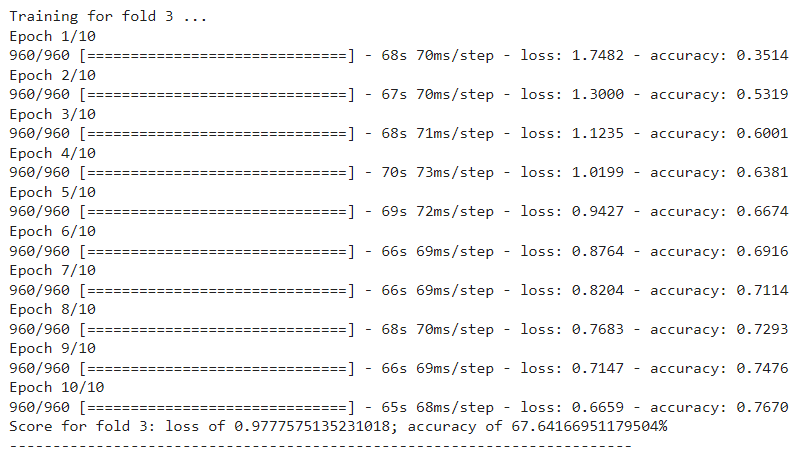


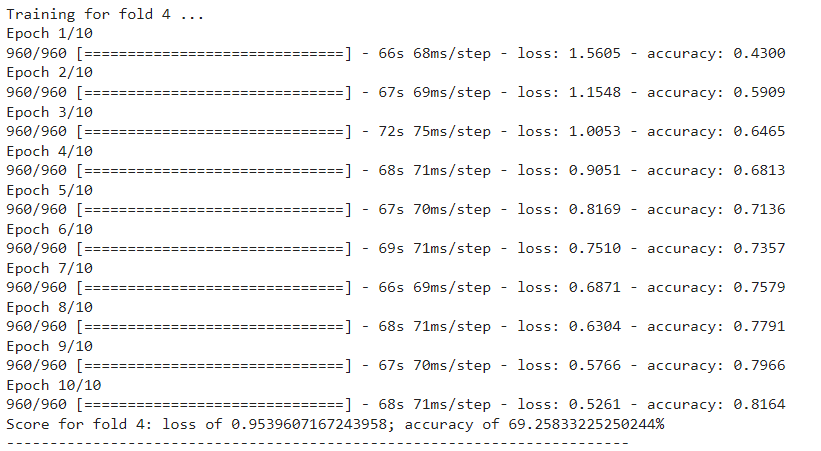


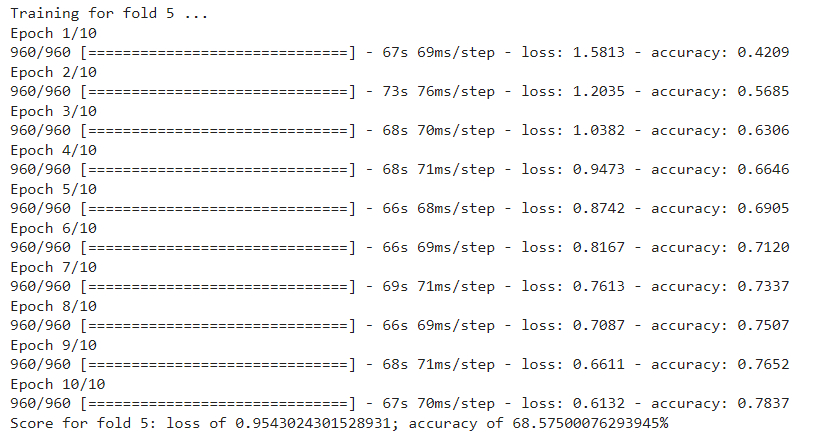


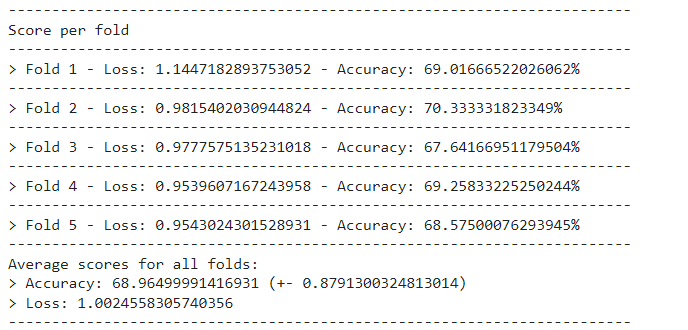












**Practical No: 6**

**Aim:** 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.shape)

print(trainY.shape)

print(testX.shape)

print(testY.shape)

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'])

model.summary()

history=model.fit(trainX,trainY,validation\_data=(testX,testY),epochs=100)

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

pyplot.plot(history.history['val\_accuracy'],label='test')

pyplot.legend()

pyplot.show()

from keras.regularizers import l2

model=Sequential()

model.add(Dense(500,input\_dim=2,activation='relu',kernel\_regularizer=l2(0.001)))

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

model.summary()

model.compile(loss='binary\_crossentropy',optimizer='adam',metrics=['accuracy'])

history=model.fit(trainX,trainY,validation\_data=(testX,testY),epochs=100)

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

pyplot.plot(history.history['val\_accuracy'],label='test')

pyplot.legend()

pyplot.show()

from keras.regularizers import l1\_l2

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'])

model.summary()

history=model.fit(trainX,trainY,validation\_data=(testX,testY),epochs=100)

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

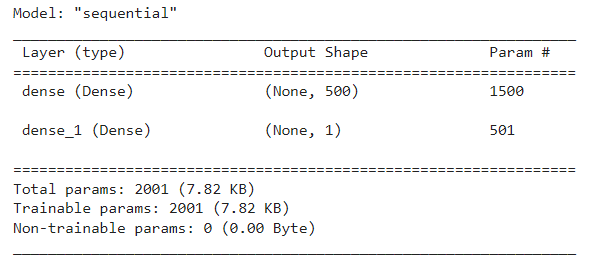
pyplot.plot(history.history['val\_accuracy'],label='test')

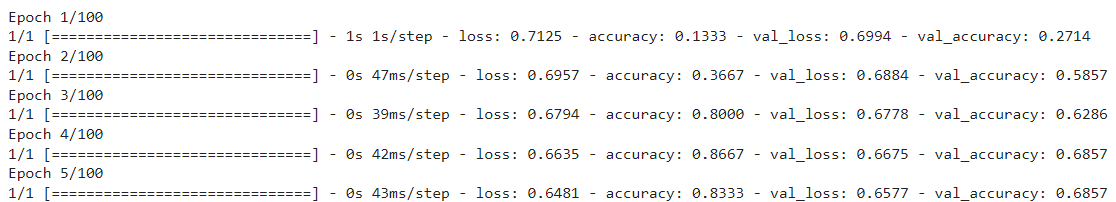
pyplot.legend()

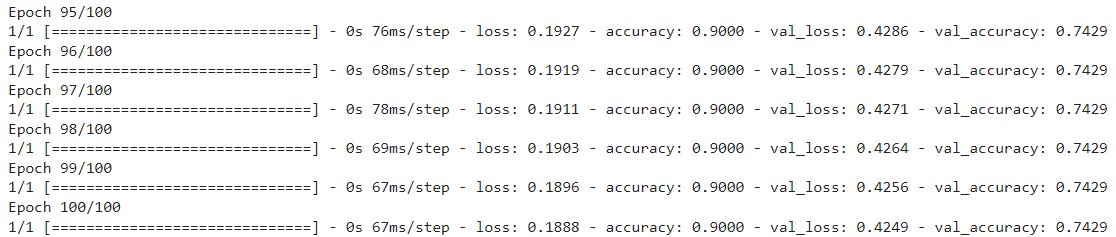
pyplot.show()

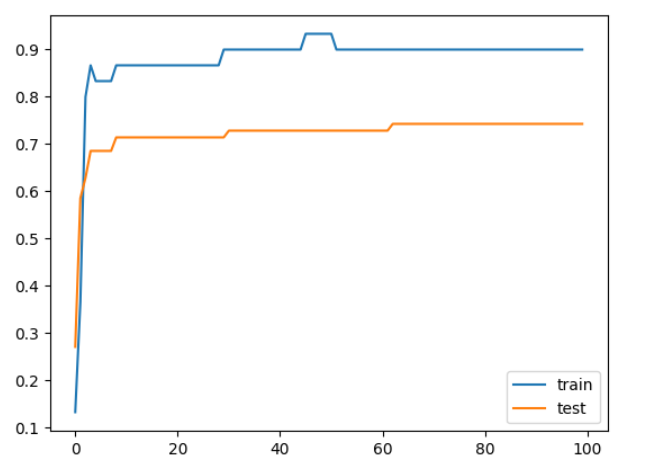
**Output:**

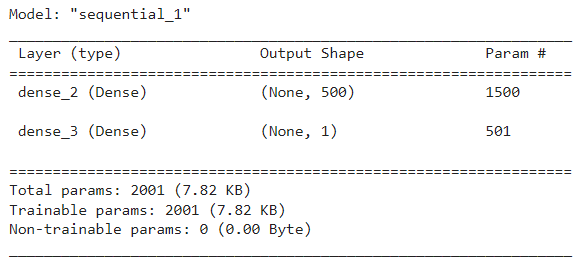


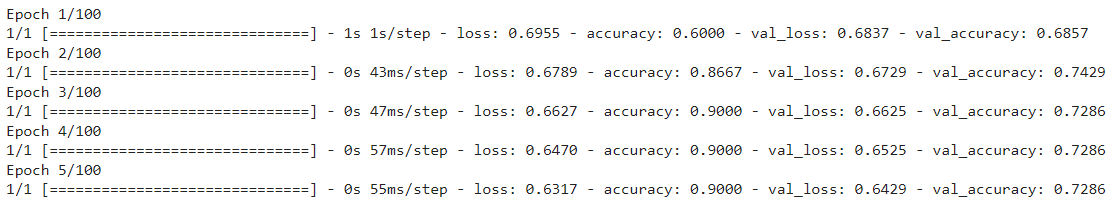


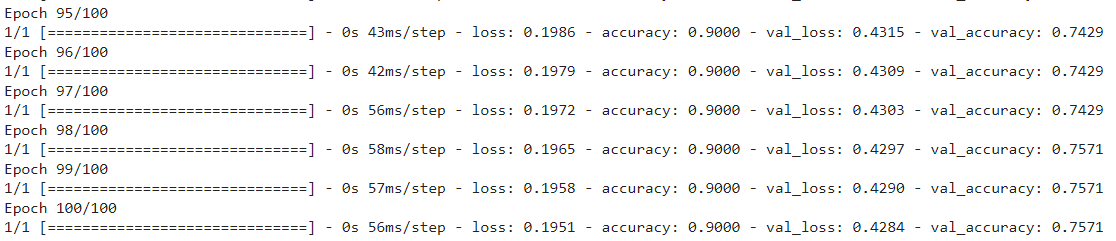


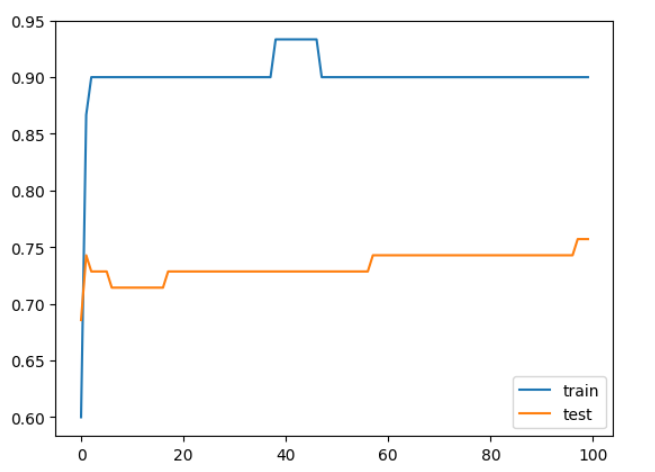


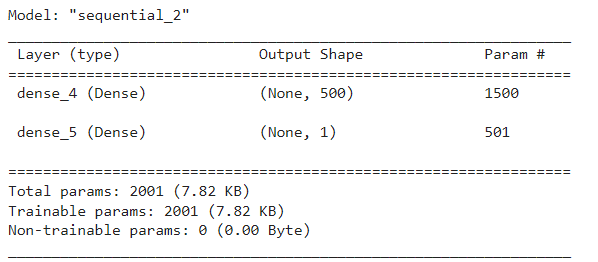


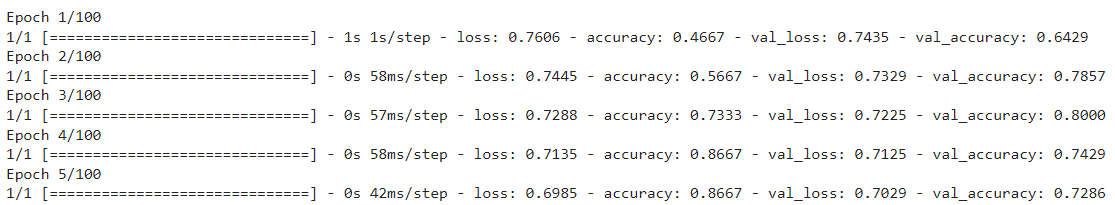


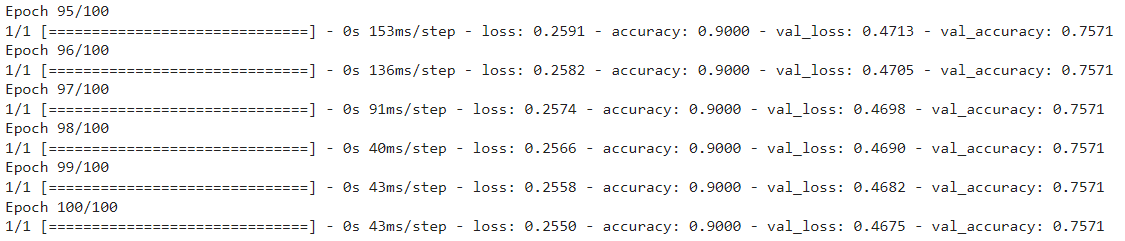


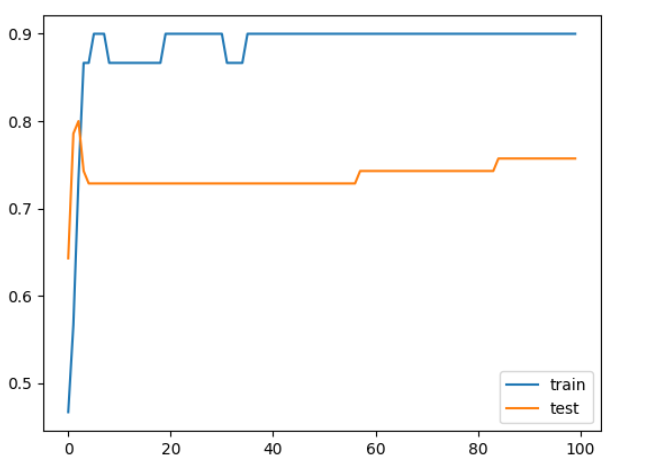












**Practical No: 7**

**Aim:** Demonstrate recurrent neural network that learns to perform sequence analysis.

**Code:**

import numpy as np

import tensorflow\_datasets as tfds

import tensorflow as tf

tfds.disable\_progress\_bar()

import matplotlib.pyplot as plt

def plot\_graphs(history, metric):

plt.plot(history.history[metric])

plt.plot(history.history['val\_'+metric], '')

plt.xlabel("Epochs")

plt.ylabel(metric)

plt.legend([metric, 'val\_'+metric])

dataset, info = tfds.load('imdb\_reviews', with\_info=True,

as\_supervised=True)

train\_dataset, test\_dataset = dataset['train'], dataset['test']

train\_dataset.element\_spec

for example, label in train\_dataset.take(5):

print('text: ', example.numpy())

print('label: ', label.numpy())

BUFFER\_SIZE = 10000

BATCH\_SIZE = 64

train\_dataset = train\_dataset.shuffle(BUFFER\_SIZE).batch(BATCH\_SIZE).prefetch(tf.data.AUTOTUNE)

test\_dataset = test\_dataset.batch(BATCH\_SIZE).prefetch(tf.data.AUTOTUNE)

for example, label in train\_dataset.take(1):

print('texts: ', example.numpy()[:3])

print()

print('labels: ', label.numpy()[:3])

VOCAB\_SIZE = 1000

encoder = tf.keras.layers.TextVectorization(max\_tokens=VOCAB\_SIZE)

encoder.adapt(train\_dataset.map(lambda text, label: text))

vocab = np.array(encoder.get\_vocabulary())

vocab[:20]

encoded\_example = encoder(example)[:3].numpy()

encoded\_example

for n in range(3):

print("Original: ", example[n].numpy())

print("Round-trip: ", " ".join(vocab[encoded\_example[n]]))

print()

model = tf.keras.Sequential([

encoder,

tf.keras.layers.Embedding(

input\_dim=len(encoder.get\_vocabulary()),

output\_dim=64,

# Use masking to handle the variable sequence lengths

mask\_zero=True),

tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64)),

tf.keras.layers.Dense(64, activation='relu'),

tf.keras.layers.Dense(1)

])

print([layer.supports\_masking for layer in model.layers])

# predict on a sample text without padding.

sample\_text = ('The movie was cool. The animation and the graphics '

'were out of this world. I would recommend this movie.')

predictions = model.predict(np.array([sample\_text]))

print(predictions[0])

# predict on a sample text with padding

padding = "the " \* 2000

predictions = model.predict(np.array([sample\_text, padding]))

print(predictions[0])

model.compile(loss=tf.keras.losses.BinaryCrossentropy(from\_logits=True),

optimizer=tf.keras.optimizers.Adam(1e-4),

metrics=['accuracy'])

history = model.fit(train\_dataset, epochs=10,

validation\_data=test\_dataset,

validation\_steps=30)

test\_loss, test\_acc = model.evaluate(test\_dataset)

print('Test Loss:', test\_loss)

print('Test Accuracy:', test\_acc)

plt.figure(figsize=(16, 8))

plt.subplot(1, 2, 1)

plot\_graphs(history, 'accuracy')

plt.ylim(None, 1)

plt.subplot(1, 2, 2)

plot\_graphs(history, 'loss')

plt.ylim(0, None)

sample\_text = ('The movie was cool. The animation and the graphics '

'were out of this world. I would recommend this movie.')

predictions = model.predict(np.array([sample\_text]))

predictions

model = tf.keras.Sequential([

encoder,

tf.keras.layers.Embedding(len(encoder.get\_vocabulary()), 64, mask\_zero=True),

tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64, return\_sequences=True)),

tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32)),

tf.keras.layers.Dense(64, activation='relu'),

tf.keras.layers.Dropout(0.5),

tf.keras.layers.Dense(1)

])

model.compile(loss=tf.keras.losses.BinaryCrossentropy(from\_logits=True),

optimizer=tf.keras.optimizers.Adam(1e-4),

metrics=['accuracy'])

history = model.fit(train\_dataset, epochs=10,

validation\_data=test\_dataset,

validation\_steps=30)

test\_loss, test\_acc = model.evaluate(test\_dataset)

print('Test Loss:', test\_loss)

print('Test Accuracy:', test\_acc)

# predict on a sample text without padding.

sample\_text = ('The movie was not good. The animation and the graphics '

'were terrible. I would not recommend this movie.')

predictions = model.predict(np.array([sample\_text]))

print(predictions)

plt.figure(figsize=(16, 6))

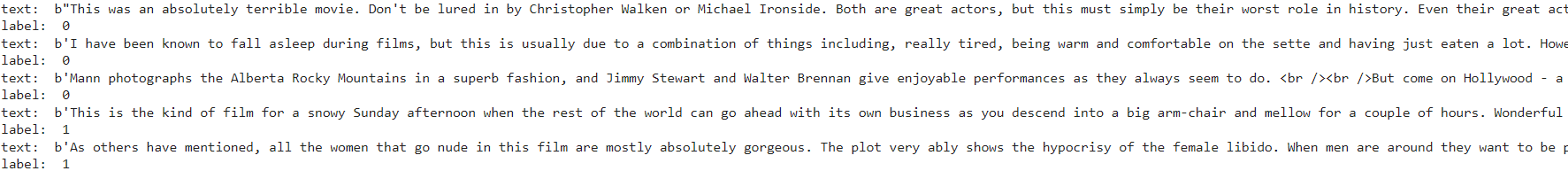
plt.subplot(1, 2, 1)

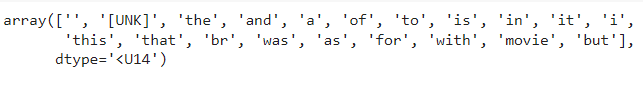
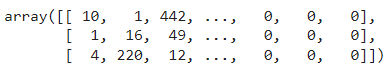
plot\_graphs(history, 'accuracy')

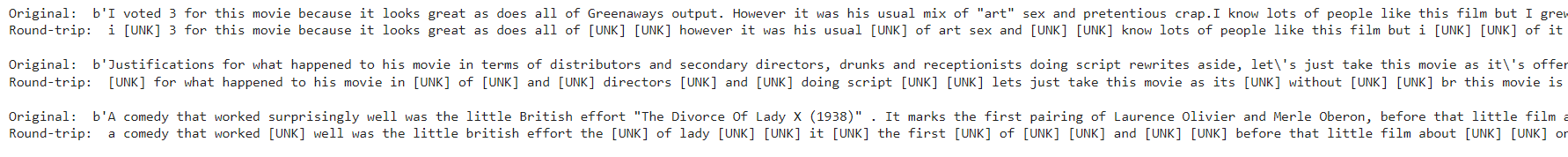
plt.subplot(1, 2, 2)

plot\_graphs(history, 'loss')

**Output:**

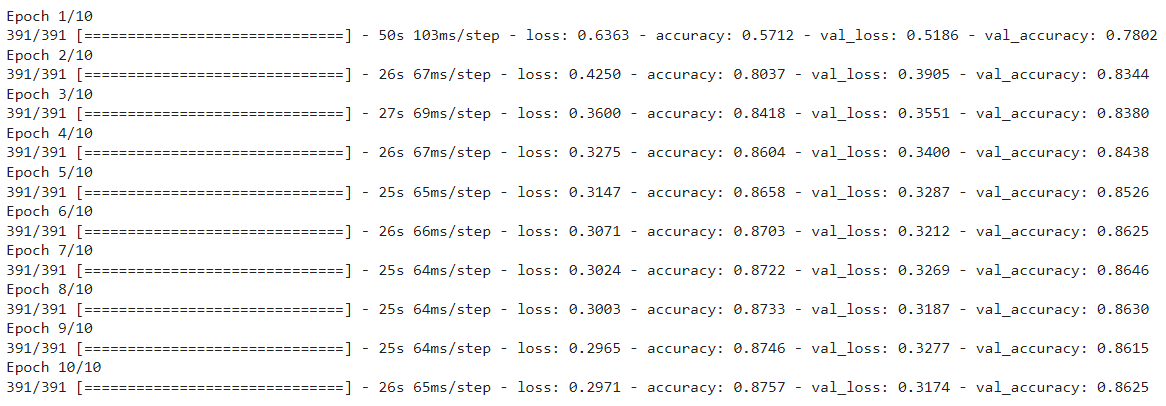


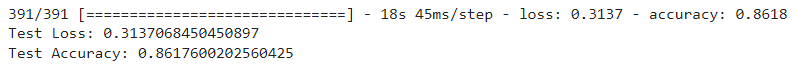


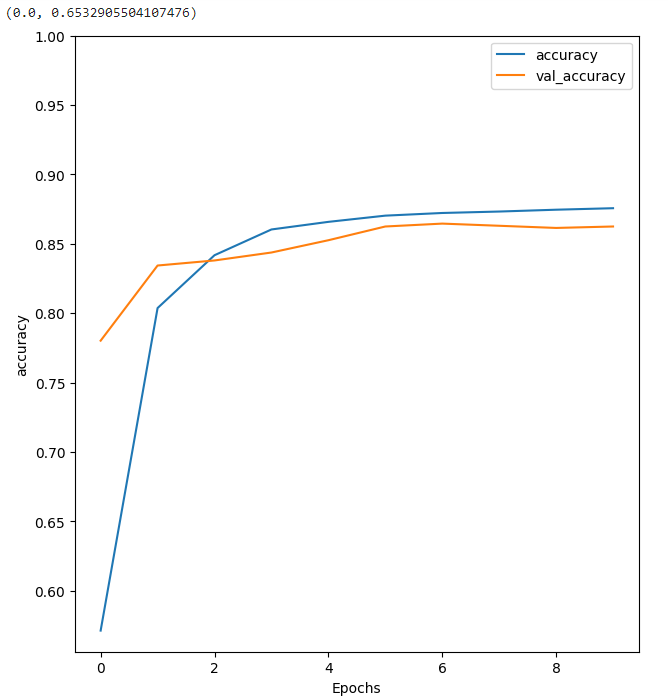


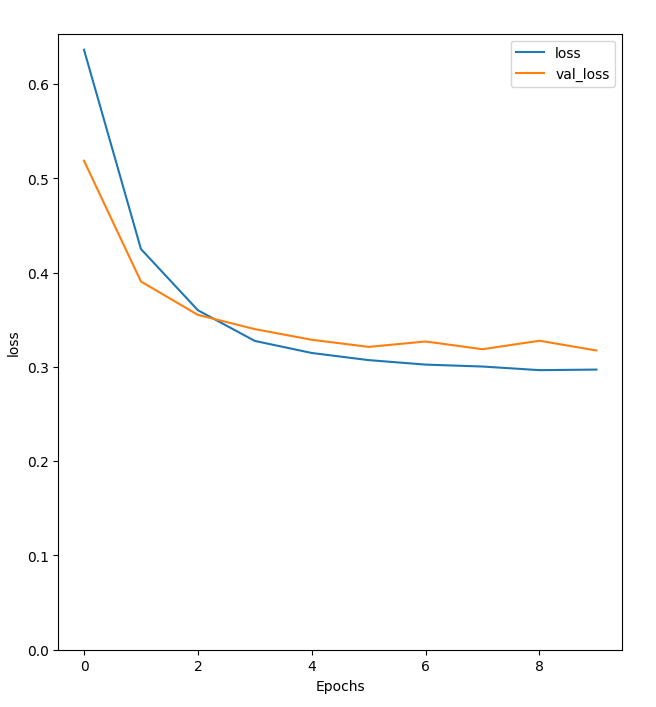


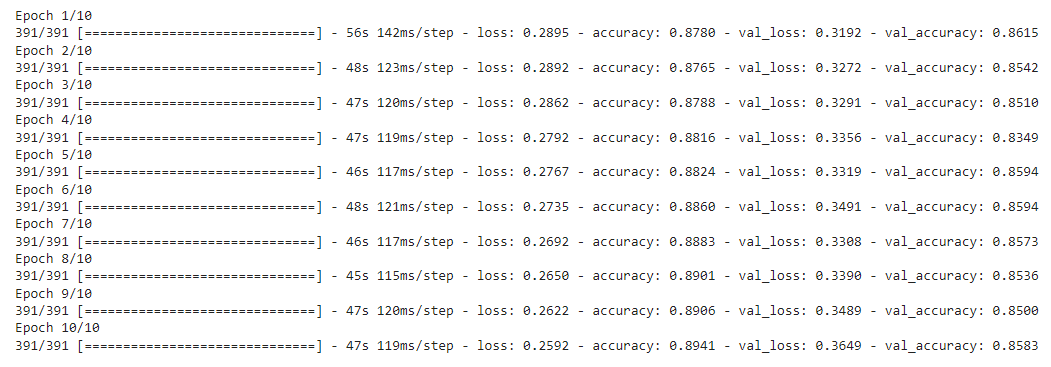


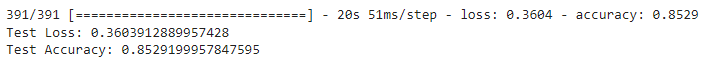


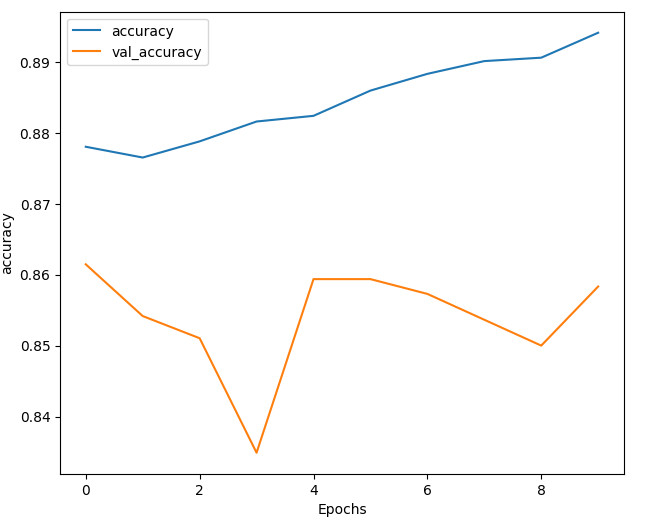


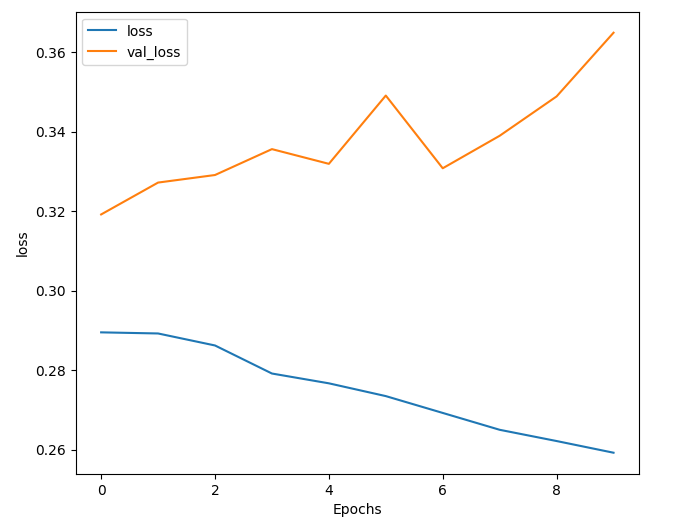












**Practical No: 8**

**Aim:** 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=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))

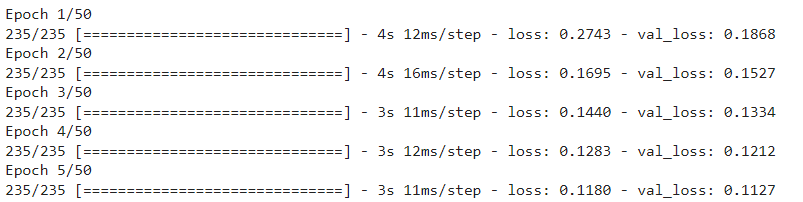
plt.gray()

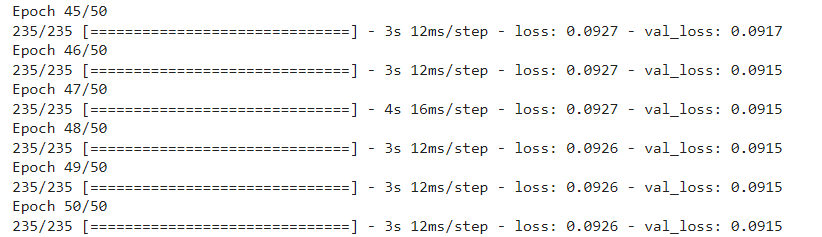
ax.get\_xaxis().set\_visible(False)

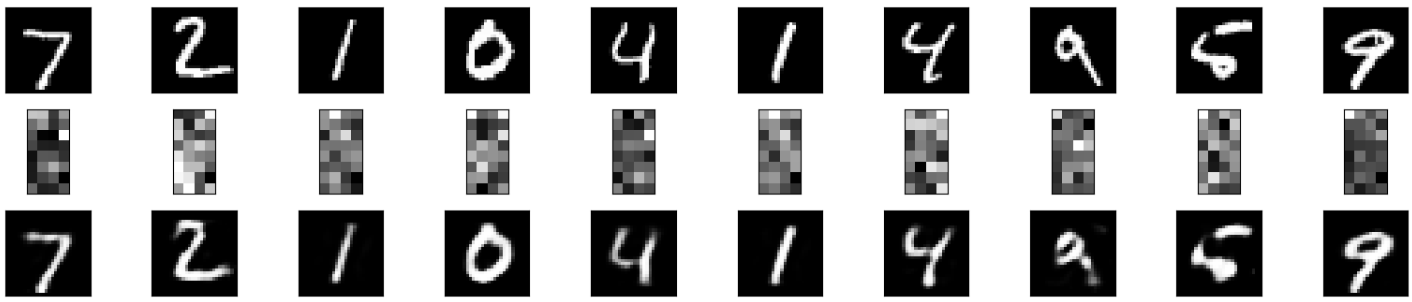
ax.get\_yaxis().set\_visible(False)

plt.show()

**Output:**







**Practical No: 9**

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

**Code:**

import tensorflow as tf

mnist = tf.keras.datasets.mnist

(X\_train, y\_train), (X\_test, y\_test) = mnist.load\_data()

X\_train.shape

y\_train.shape

X\_test.shape

y\_test.shape

import matplotlib.pyplot as plt

plt.imshow(X\_train[2])

plt.show()

plt.imshow(X\_train[2], cmap=plt.cm.binary)

X\_train[2]

X\_train = tf.keras.utils.normalize(X\_train, axis=1)

X\_test = tf.keras.utils.normalize(X\_test, axis=1)

plt.imshow(X\_train[2], cmap=plt.cm.binary)

print(X\_train[2])

import tensorflow as tf

import tensorflow.keras.layers as KL

import tensorflow.keras.models as KM

inputs = KL.Input(shape=(28, 28, 1))

c = KL.Conv2D(32, (3, 3), padding="valid", activation=tf.nn.relu)(inputs)

m = KL.MaxPool2D((2, 2), (2, 2))(c)

d = KL.Dropout(0.5)(m)

c = KL.Conv2D(64, (3, 3), padding="valid", activation=tf.nn.relu)(d)

m = KL.MaxPool2D((2, 2), (2, 2))(c)

d = KL.Dropout(0.5)(m)

c = KL.Conv2D(128, (3, 3), padding="valid", activation=tf.nn.relu)(d)

f = KL.Flatten()(c)

outputs = KL.Dense(10, activation=tf.nn.softmax)(f)

model = KM.Model(inputs, outputs)

model.summary()

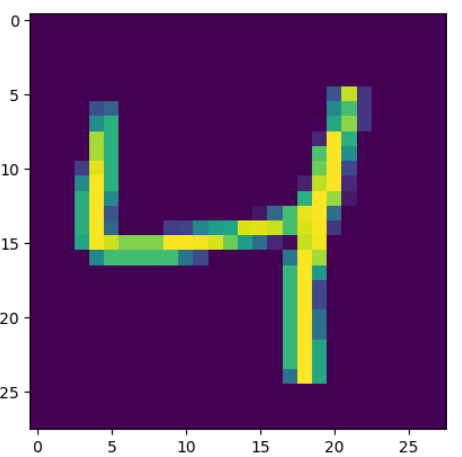
model.compile(optimizer="adam", loss="sparse\_categorical\_crossentropy", metrics=["accuracy"])

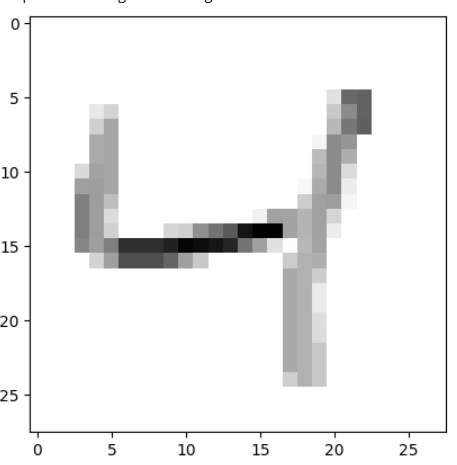
model.fit(X\_train, y\_train, epochs=5)

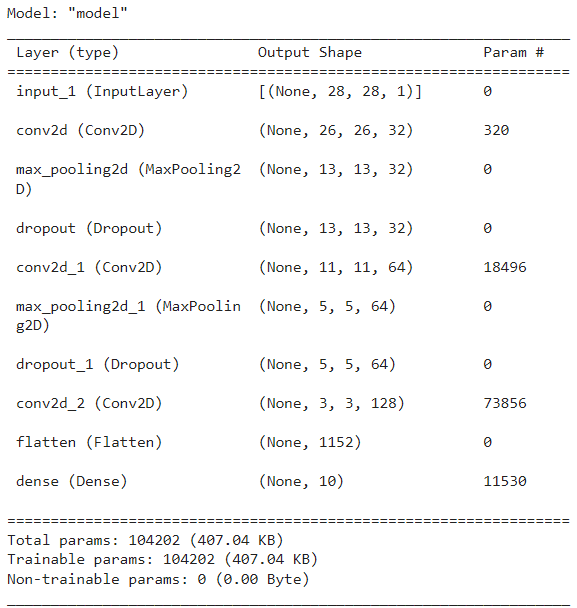
test\_loss, test\_acc = model.evaluate(X\_test, y\_test)

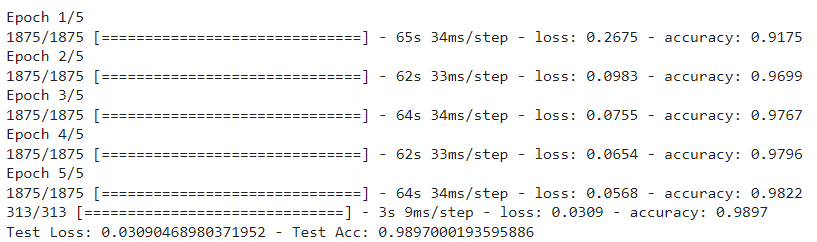
print("Test Loss: {0} - Test Acc: {1}".format(test\_loss, test\_acc))

**Output:**









**Practical No: 10**

**Aim:** 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=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:**

