Practical 1

Aim: Implement Feed-forward Neural Network and train the network with different optimizers and compare the results.

What is feed-forward Neural Network ?

A feed-forward neural network (FFNN) is a type of artificial neural network where the information flows in one direction, from the input layer to the output layer, without loops or cycles. It is also called a multi-layer perceptron (MLP) because it consists of multiple layers of perceptrons, which are the basic units of computation in neural networks.

The FFNN is composed of three or more layers: an input layer, one or more hidden layers, and an output layer. Each layer consists of one or more nodes, also known as neurons. Neurons in the input layer receive the input data and neurons in the output layer produce the output. Neurons in the hidden layers perform computations on the input data and pass the results to the next layer until the output is produced.

```python
# import the necessary packages
from sklearn.preprocessing import LabelBinarizer
from sklearn.metrics import classification_report
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.datasets import mnist
from tensorflow.keras import backend as K
import matplotlib.pyplot as plt
import numpy as np
import argparse

# grab the MNIST dataset (if this is your first time using this
# dataset then the 11MB download may take a minute)
print("[INFO] accessing MNIST...")
((trainX, trainY), (testX, testY)) = mnist.load_data()
# each image in the MNIST dataset is represented as a 28x28x1
# image, but in order to apply a standard neural network we must
# first "flatten" the image to be simple list of 28x28=784 pixels
trainX = trainX.reshape((trainX.shape[0], 28 * 28 * 1))
testX = testX.reshape((testX.shape[0], 28 * 28 * 1))
# scale data to the range of [0, 1]
trainX = trainX.astype("float32") / 255.0
testX = testX.astype("float32") / 255.0
```

Output:

```
[INFO] accessing MNIST...
Downloading data from https://storage.googleapis.com/tensorflow/tf-ker
11490434/11490434 [==============================] - 0s 0us/step
```

```
# convert the labels from integers to vectors
lb = LabelBinarizer()
trainY = lb.fit_transform(trainY)
testY = lb.transform(testY)

model = Sequential()
model.add(Dense(256, input_shape=(784,), activation="sigmoid"))
model.add(Dense(128, activation="sigmoid"))
model.add(Dense(10, activation="softmax"))

print("[INFO] training network...")
sgd = SGD(0.01)
model.compile(loss="categorical_crossentropy", optimizer=sgd, metrics=[
"accuracy"])
H = model.fit(trainX, trainY, validation_data=(testX, testY), epochs=10
0, batch_size=128)
```

Output:

```
print("[INFO] training network...")
sgd = SGD(0.01)
model.compile(loss="categorical_crossentropy", optimizer=sgd, metrics=["accuracy"])
H = model.fit(trainX, trainY, validation_data=(testX, testY), epochs=100, batch_size=128)
```

```
469/469 [==============================] - 4s 7ms/step - loss: 2.2761 - accuracy: 0.1929 - val_
Epoch 2/100
469/469 [==============================] - 5s 10ms/step - loss: 2.1987 - accuracy: 0.3972 - val
Epoch 3/100
469/469 [==============================] - 7s 15ms/step - loss: 2.1058 - accuracy: 0.5024 - val
Epoch 4/100
469/469 [==============================] - 4s 9ms/step - loss: 1.9717 - accuracy: 0.5816 - val_
Epoch 5/100
469/469 [==============================] - 3s 7ms/step - loss: 1.7891 - accuracy: 0.6378 - val_
Epoch 6/100
469/469 [==============================] - 7s 15ms/step - loss: 1.5745 - accuracy: 0.6858 - val
Epoch 7/100
469/469 [==============================] - 7s 14ms/step - loss: 1.3638 - accuracy: 0.7247 - val
Epoch 8/100
469/469 [==============================] - 8s 17ms/step - loss: 1.1846 - accuracy: 0.7559 - val
Epoch 9/100
469/469 [==============================] - 5s 11ms/step - loss: 1.0432 - accuracy: 0.7793 - val
Epoch 10/100
469/469 [==============================] - 6s 12ms/step - loss: 0.9330 - accuracy: 0.7959 - val
Epoch 11/100
469/469 [==============================] - 5s 11ms/step - loss: 0.8461 - accuracy: 0.8100 - val
Epoch 12/100
469/469 [==============================] - 5s 11ms/step - loss: 0.7764 - accuracy: 0.8210 - val
Epoch 13/100
469/469 [==============================] - 5s 10ms/step - loss: 0.7192 - accuracy: 0.8307 - val
Epoch 14/100
469/469 [==============================] - 3s 7ms/step - loss: 0.6720 - accuracy: 0.8389 - val_
Epoch 15/100
469/469 [==============================] - 3s 7ms/step - loss: 0.6326 - accuracy: 0.8459 - val_
Epoch 16/100
469/469 [==============================] - 5s 10ms/step - loss: 0.5990 - accuracy: 0.8515 - val
Epoch 17/100
469/469 [==============================] - 5s 11ms/step - loss: 0.5704 - accuracy: 0.8571 - val
Epoch 18/100
469/469 [==============================] - 4s 9ms/step - loss: 0.5457 - accuracy: 0.8618 - val
```

```
# evaluate the network
```

```
print("[INFO] evaluating network...")
predictions = model.predict(testX, batch_size=128)
print(classification_report(testY.argmax(axis=1), predictions.argmax(ax
is=1), target_names=[str(x) for x in lb.classes_]))
```

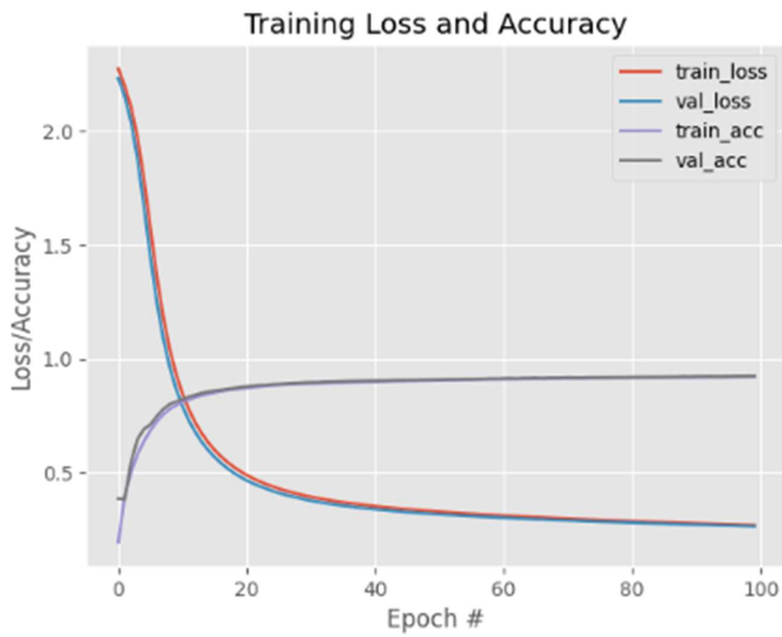
Output:

```
[INFO] evaluating network...
79/79 [==============================] - 0s 3ms/step
              precision    recall  f1-score   support

           0       0.94      0.98      0.96       980
           1       0.97      0.98      0.97      1135
           2       0.92      0.90      0.91      1032
           3       0.91      0.91      0.91      1010
           4       0.92      0.93      0.93       982
           5       0.90      0.87      0.88       892
           6       0.94      0.95      0.94       958
           7       0.94      0.92      0.93      1028
           8       0.90      0.89      0.89       974
           9       0.91      0.91      0.91      1009

    accuracy                           0.92     10000
   macro avg       0.92      0.92      0.92     10000
weighted avg       0.92      0.92      0.92     10000
```

```
plt.style.use("ggplot")
plt.figure()
plt.plot(np.arange(0, 100), H.history["loss"], label="train_loss")
plt.plot(np.arange(0, 100), H.history["val_loss"], label="val_loss")
plt.plot(np.arange(0, 100), H.history["accuracy"], label="train_acc")
plt.plot(np.arange(0, 100), H.history["val_accuracy"], label="val_acc")
plt.title("Training Loss and Accuracy")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend()
```

Output:

```
<matplotlib.legend.Legend at 0x7fe00b59a5f0>
```




```python
# import the necessary packages
from sklearn.preprocessing import LabelBinarizer
from sklearn.metrics import classification_report
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.datasets import cifar10
import matplotlib.pyplot as plt
import numpy as np
import argparse


print("[INFO] loading CIFAR-10 data...")
((trainX, trainY), (testX, testY)) = cifar10.load_data()
trainX = trainX.astype("float") / 255.0
testX = testX.astype("float") / 255.0
trainX = trainX.reshape((trainX.shape[0], 3072))
testX = testX.reshape((testX.shape[0], 3072))
```

Output:

```
[INFO] loading CIFAR-10 data...
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 [==============================] - 4s 0us/step
```

```python
lb = LabelBinarizer()
trainY = lb.fit_transform(trainY)
testY = lb.transform(testY)
# initialize the label names for the CIFAR-10 dataset
```

```
labelNames = ["airplane", "automobile", "bird", "cat", "deer","dog", "frog", "horse", "ship", "truck"]


model = Sequential()
model.add(Dense(1024, input_shape=(3072,), activation="relu"))
model.add(Dense(512, activation="relu"))
model.add(Dense(10, activation="softmax"))


print("[INFO] training network...")
sgd = SGD(0.01)
model.compile(loss="categorical_crossentropy", optimizer=sgd, metrics=["accuracy"])
H = model.fit(trainX, trainY, validation_data=(testX, testY), epochs=100, batch_size=32)
```

Output:

```
print("[INFO] training network...")
sgd = SGD(0.01)
model.compile(loss="categorical_crossentropy", optimizer=sgd, metrics=["accuracy"])
H = model.fit(trainX, trainY, validation_data=(testX, testY), epochs=100, batch_size=32)

[INFO] training network...
Epoch 1/100
1563/1563 [==============================] - 57s 36ms/step - loss: 1.8368 - accuracy: 0.3446 -
Epoch 2/100
1563/1563 [==============================] - 49s 31ms/step - loss: 1.6517 - accuracy: 0.4146 -
Epoch 3/100
1563/1563 [==============================] - 49s 32ms/step - loss: 1.5681 - accuracy: 0.4485 -
Epoch 4/100
1563/1563 [==============================] - 49s 31ms/step - loss: 1.5081 - accuracy: 0.4687 -
Epoch 5/100
1563/1563 [==============================] - 49s 31ms/step - loss: 1.4594 - accuracy: 0.4851 -
Epoch 6/100
1563/1563 [==============================] - 51s 33ms/step - loss: 1.4188 - accuracy: 0.5013 -
Epoch 7/100
1563/1563 [==============================] - 54s 35ms/step - loss: 1.3837 - accuracy: 0.5118 -
Epoch 8/100
1563/1563 [==============================] - 49s 32ms/step - loss: 1.3509 - accuracy: 0.5256 -
Epoch 9/100
1563/1563 [==============================] - 49s 31ms/step - loss: 1.3212 - accuracy: 0.5334 -
Epoch 10/100
1563/1563 [==============================] - 51s 33ms/step - loss: 1.2924 - accuracy: 0.5473 -
Epoch 11/100
1563/1563 [==============================] - 52s 33ms/step - loss: 1.2657 - accuracy: 0.5524 -
Epoch 12/100
1563/1563 [==============================] - 50s 32ms/step - loss: 1.2409 - accuracy: 0.5648 -
Epoch 13/100
1563/1563 [==============================] - 51s 33ms/step - loss: 1.2170 - accuracy: 0.5717 -
Epoch 14/100
1563/1563 [==============================] - 52s 33ms/step - loss: 1.1906 - accuracy: 0.5828 -
Epoch 15/100
1563/1563 [==============================] - 49s 31ms/step - loss: 1.1680 - accuracy: 0.5900 -
Epoch 16/100
1563/1563 [==============================] - 51s 33ms/step - loss: 1.1451 - accuracy: 0.5961 -
```

```
print("[INFO] evaluating network...")
predictions = model.predict(testX, batch_size=32)
print(classification_report(testY.argmax(axis=1),predictions.argmax(axis=1), target_names=labelNames))
```

Output:

```
print("[INFO] evaluating network...")
predictions = model.predict(testX, batch_size=32)
print(classification_report(testY.argmax(axis=1),predictions.argmax(axis=1), target_names=labe
```

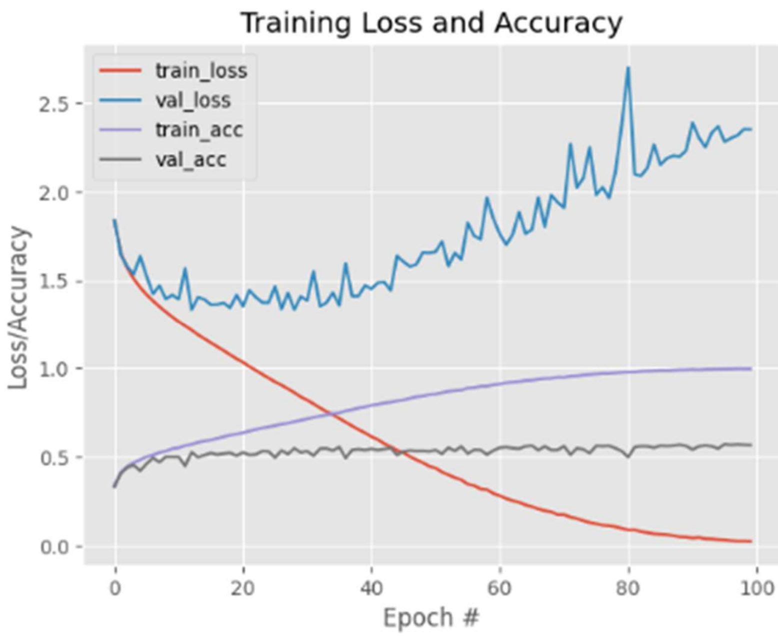
```
[INFO] evaluating network...
313/313 [==============================] - 5s 15ms/step
              precision    recall  f1-score   support

    airplane       0.65      0.64      0.64      1000
  automobile       0.70      0.68      0.69      1000
        bird       0.49      0.42      0.45      1000
         cat       0.39      0.38      0.38      1000
        deer       0.47      0.51      0.49      1000
         dog       0.43      0.54      0.48      1000
        frog       0.64      0.61      0.63      1000
       horse       0.69      0.56      0.62      1000
        ship       0.66      0.73      0.69      1000
       truck       0.61      0.61      0.61      1000

    accuracy                           0.57     10000
   macro avg       0.57      0.57      0.57     10000
weighted avg       0.57      0.57      0.57     10000
```

```
plt.style.use("ggplot")
plt.figure()
plt.plot(np.arange(0, 100), H.history["loss"], label="train_loss")
plt.plot(np.arange(0, 100), H.history["val_loss"], label="val_loss")
plt.plot(np.arange(0, 100), H.history["accuracy"], label="train_acc")
plt.plot(np.arange(0, 100), H.history["val_accuracy"], label="val_acc")
plt.title("Training Loss and Accuracy")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend()
```

Output:

```
plt.style.use("ggplot")
plt.figure()
plt.plot(np.arange(0, 100), H.history["loss"], label="train_loss")
plt.plot(np.arange(0, 100), H.history["val_loss"], label="val_loss")
plt.plot(np.arange(0, 100), H.history["accuracy"], label="train_acc")
plt.plot(np.arange(0, 100), H.history["val_accuracy"], label="val_acc")
plt.title("Training Loss and Accuracy")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7fe033482ce0>
```




```python
from sklearn.preprocessing import LabelBinarizer
from sklearn.metrics import classification_report
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.datasets import cifar10
import matplotlib.pyplot as plt
import numpy as np
import argparse


# load the training and testing data, scale it into the range [0, 1],
# then reshape the design matrix
print("[INFO] loading CIFAR-10 data...")
((trainX, trainY), (testX, testY)) = cifar10.load_data()
trainX = trainX.astype("float") / 255.0
testX = testX.astype("float") / 255.0
trainX = trainX.reshape((trainX.shape[0], 3072))
testX = testX.reshape((testX.shape[0], 3072))
```

Output:

```
# load the training and testing data, scale it into the range [0, 1],
# then reshape the design matrix
print("[INFO] loading CIFAR-10 data...")
((trainX, trainY), (testX, testY)) = cifar10.load_data()
trainX = trainX.astype("float") / 255.0
testX = testX.astype("float") / 255.0
trainX = trainX.reshape((trainX.shape[0], 3072))
testX = testX.reshape((testX.shape[0], 3072))
```

```
[INFO] loading CIFAR-10 data...
```

```
# convert the labels from integers to vectors
lb = LabelBinarizer()
trainY = lb.fit_transform(trainY)
testY = lb.transform(testY)
# initialize the label names for the CIFAR-10 dataset
labelNames = ["airplane", "automobile", "bird", "cat", "deer",
  "dog", "frog", "horse", "ship", "truck"]


# define the 3072-1024-512-10 architecture using Keras
model = Sequential()
model.add(Dense(1024, input_shape=(3072,), activation="relu"))
model.add(Dense(512, activation="relu"))
model.add(Dense(10, activation="softmax"))


# train the model using SGD
print("[INFO] training network...")
sgd = SGD(0.01)
model.compile(loss="categorical_crossentropy", optimizer=sgd, metrics=[
"accuracy"])
H = model.fit(trainX, trainY, validation_data=(testX, testY), epochs=10
0, batch_size=32)
```

Output:

```
# train the model using SGD
print("[INFO] training network...")
sgd = SGD(0.01)
model.compile(loss="categorical_crossentropy", optimizer=sgd, metrics=["accuracy"])
H = model.fit(trainX, trainY, validation_data=(testX, testY), epochs=100, batch_size=32)
```

```
1563/1563 [==============================] - 49s 31ms/step - loss: 0.1678 - accuracy: 0.9542
Epoch 73/100
1563/1563 [==============================] - 50s 32ms/step - loss: 0.1576 - accuracy: 0.9571
Epoch 74/100
1563/1563 [==============================] - 50s 32ms/step - loss: 0.1541 - accuracy: 0.9572
Epoch 75/100
1563/1563 [==============================] - 47s 30ms/step - loss: 0.1459 - accuracy: 0.9605
Epoch 76/100
1563/1563 [==============================] - 47s 30ms/step - loss: 0.1363 - accuracy: 0.9634
Epoch 77/100
1563/1563 [==============================] - 47s 30ms/step - loss: 0.1221 - accuracy: 0.9696
Epoch 78/100
1563/1563 [==============================] - 47s 30ms/step - loss: 0.1142 - accuracy: 0.9716
Epoch 79/100
1563/1563 [==============================] - 49s 31ms/step - loss: 0.1098 - accuracy: 0.9738
Epoch 80/100
1563/1563 [==============================] - 48s 31ms/step - loss: 0.1039 - accuracy: 0.9759
Epoch 81/100
1563/1563 [==============================] - 48s 31ms/step - loss: 0.1014 - accuracy: 0.9759
Epoch 82/100
1563/1563 [==============================] - 49s 32ms/step - loss: 0.0880 - accuracy: 0.9804
Epoch 83/100
1563/1563 [==============================] - 45s 29ms/step - loss: 0.0865 - accuracy: 0.9809
Epoch 84/100
1563/1563 [==============================] - 49s 31ms/step - loss: 0.0839 - accuracy: 0.9809
Epoch 85/100
1563/1563 [==============================] - 47s 30ms/step - loss: 0.0721 - accuracy: 0.9853
Epoch 86/100
1563/1563 [==============================] - 47s 30ms/step - loss: 0.0707 - accuracy: 0.9853
Epoch 87/100
1563/1563 [==============================] - 48s 31ms/step - loss: 0.0629 - accuracy: 0.9880
Epoch 88/100
1563/1563 [==============================] - 48s 31ms/step - loss: 0.0596 - accuracy: 0.9891
Epoch 89/100
1563/1563 [==============================] - 47s 30ms/step - loss: 0.0563 - accuracy: 0.9901
```

```
print("[INFO] evaluating network...")
predictions = model.predict(testX, batch_size=32)
print(classification_report(testY.argmax(axis=1),
  predictions.argmax(axis=1), target_names=labelNames))
```

Output:

```
print("[INFO] evaluating network...")
predictions = model.predict(testX, batch_size=32)
print(classification_report(testY.argmax(axis=1),
  predictions.argmax(axis=1), target_names=labelNames))
```

```
[INFO] evaluating network...
313/313 [==============================] - 4s 13ms/step
              precision    recall  f1-score   support

    airplane       1.00      0.11      0.20     10000
  automobile       0.00      0.00      0.00         0
        bird       0.00      0.00      0.00         0
         cat       0.00      0.00      0.00         0
        deer       0.00      0.00      0.00         0
         dog       0.00      0.00      0.00         0
        frog       0.00      0.00      0.00         0
       horse       0.00      0.00      0.00         0
        ship       0.00      0.00      0.00         0
       truck       0.00      0.00      0.00         0

    accuracy                           0.11     10000
   macro avg       0.10      0.01      0.02     10000
weighted avg       1.00      0.11      0.20     10000

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetric\
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetric\
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetric\
  _warn_prf(average, modifier, msg_start, len(result))
```

```
# plot the training loss and accuracy
plt.style.use("ggplot")
plt.figure()
plt.plot(np.arange(0, 100), H.history["loss"], label="train_loss")
plt.plot(np.arange(0, 100), H.history["val_loss"], label="val_loss")
plt.plot(np.arange(0, 100), H.history["accuracy"], label="train_acc")
plt.plot(np.arange(0, 100), H.history["val_accuracy"], label="val_acc")
plt.title("Training Loss and Accuracy")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend()
plt.savefig(args["output"])
```

Output:


Training Loss and Accuracy
train_loss
val_loss
train_acc
val_acc
Loss/Accuracy
Epoch #
0.0
0.5
1.0
1.5
2.0
2.5
0
20
40
60
80
100

Practical 2

Aim: Write a Program to implement regularization to prevent the model from overfitting

What is Regularization ?

Regularization is a technique used in machine learning to prevent overfitting, which is a phenomenon where a model performs well on the training data but poorly on new, unseen data. Overfitting occurs when the model is too complex or has too many parameters relative to the size of the training data, and as a result, it memorizes the training data instead of learning to generalize.

Regularization works by adding a penalty term to the loss function of the model during training. The penalty term encourages the model to have smaller weights or simpler representations, which reduces its capacity to fit the training data too closely. This makes the model more robust and less prone to overfitting, which in turn improves its ability to generalize to new data.

There are two common types of regularization:

1. **L1 regularization** (also known as Lasso regularization) adds a penalty proportional to the absolute value of the weights. This results in sparse weight matrices, where many of the weights are zero.
2. **L2 regularization** (also known as Ridge regularization) adds a penalty proportional to the square of the weights. This results in smaller weight values overall, but does not force any of the weights to be zero.

```
import numpy as np
import pandas as pd
from sklearn import metrics
from sklearn.linear_model import Lasso

df_train = pd.read_csv('/content/train.csv')
df_test = pd.read_csv('/content/test.csv')

df_train = df_train.dropna()
df_test = df_test.dropna()

x_train = df_train['x']
x_train = x_train.values.reshape(-1,1)
y_train = df_train['y']
y_train = y_train.values.reshape(-1,1)

x_test = df_test['x']
x_test = x_test.values.reshape(-1,1)
y_test = df_test['y']
y_test = y_test.values.reshape(-1,1)

lasso = Lasso()
```

```
lasso.fit(x_train, y_train)
```

Output:

```
▾ Lasso
Lasso()
```

```
print("Lasso Train RMSE:", np.round(np.sqrt(metrics.mean_squared_error(y_train,
lasso.predict(x_train))), 5))
print("Lasso Test RMSE:", np.round(np.sqrt(metrics.mean_squared_error(y_test,
lasso.predict(x_test))), 5))
```

Output:

```
Lasso Train RMSE: 2.80516
Lasso Test RMSE: 3.07592
```

```
import numpy as np
import pandas as pd
from sklearn import metrics
from sklearn.linear_model import Ridge

df_train = pd.read_csv('train.csv')
df_test = pd.read_csv('test.csv')

df_train = df_train.dropna()
df_test = df_test.dropna()

x_train = df_train['x']
x_train = x_train.values.reshape(-1,1)
y_train = df_train['y']
y_train = y_train.values.reshape(-1,1)

x_test = df_test['x']
x_test = x_test.values.reshape(-1,1)
y_test = df_test['y']
y_test = y_test.values.reshape(-1,1)

ridge = Ridge()

ridge.fit(x_train, y_train)
print("Ridge Train RMSE:", np.round(np.sqrt(metrics.mean_squared_error(y_train,
ridge.predict(x_train))), 5))
print("Ridge Test RMSE:", np.round(np.sqrt(metrics.mean_squared_error(y_test,
ridge.predict(x_test))), 5))
```

Output:

```
Ridge Train RMSE: 2.80495
Ridge Test RMSE: 3.07131
```

Practical 3

Aim: Implement deep learning for recognizing classes for datasets like CIFAR-10 images for previously unseen images and assign them to one of the 10 classes.

What is CIFAR?

Deep learning is a type of machine learning that uses artificial neural networks to learn from data. Neural networks are inspired by the human brain, and they can be used to solve a variety of problems, including image classification.

The CIFAR-10 (Canadian Institute for Advanced Research)dataset is a popular dataset for image classification. It consists of 60,000 32x32 color images, divided into 10 classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck.

```
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt
import numpy as np

(X_train, y_train), (X_test,y_test) = datasets.cifar10.load_data()
X_train.shape
```

OUTPUT

```
(X_train, y_train), (X_test,y_test) = datasets.cifar10.load_data()
X_train.shape
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-pytho
170498071/170498071 [==============================] - 11s 0us/step
(50000, 32, 32, 3)
```

```
X_test.shape
```

OUTPUT:

```
(10000, 32, 32, 3)
```

```
y_train.shape
```

OUTPUT:

```
(50000, 1)
```

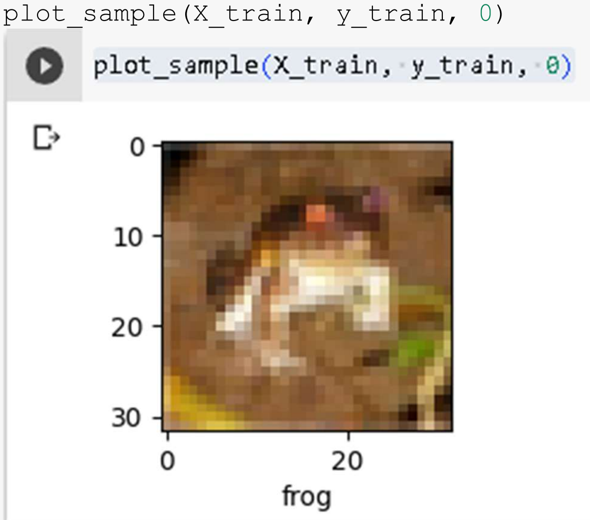
```
y_train[:5]
```

OUTPUT:

```
y_train[:5]
```

```
array([[6],
       [9],
       [9],
       [4],
       [1]], dtype=uint8)
```

```
y_train = y_train.reshape(-1,)
y_train[:5]
OUTPUT:
array([6, 9, 9, 4, 1], dtype=uint8)

y_test = y_test.reshape(-1,)

classes =
["airplane","automobile","bird","cat","deer","dog","frog","horse","ship
","truck"]

def plot_sample(X, y, index):
    plt.figure(figsize = (15,2))
    plt.imshow(X[index])
    plt.xlabel(classes[y[index]])

plot_sample(X_train, y_train, 0)
```




```
X_train = X_train / 255.0
X_test = X_test / 255.0

ann = models.Sequential([
        layers.Flatten(input_shape=(32,32,3)),
        layers.Dense(3000, activation='relu'),
        layers.Dense(1000, activation='relu'),
        layers.Dense(10, activation='softmax')
    ])

ann.compile(optimizer='SGD',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

ann.fit(X_train, y_train, epochs=5)
```

```
ann.fit(X_train, y_train, epochs=5)
```

```
Epoch 1/5
1563/1563 [==============================] - 171s 109ms/step - l
Epoch 2/5
1563/1563 [==============================] - 153s 98ms/step - lo
Epoch 3/5
1563/1563 [==============================] - 149s 95ms/step - lo
Epoch 4/5
1563/1563 [==============================] - 146s 93ms/step - lo
Epoch 5/5
1563/1563 [==============================] - 145s 93ms/step - lo
<keras.callbacks.History at 0x7f00a964d570>
```

```
from sklearn.metrics import confusion_matrix , classification_report
import numpy as np
y_pred = ann.predict(X_test)
y_pred_classes = [np.argmax(element) for element in y_pred]
```


```
from sklearn.metrics import confusion_matrix , classification_r
import numpy as np
y_pred = ann.predict(X_test)
y_pred_classes = [np.argmax(element) for element in y_pred]
```


```
313/313 [==============================] - 10s 31ms/step
```

```
print("Classification Report: \n", classification_report(y_test,
y_pred_classes))
```


```
print("Classification Report: \n", classification_report(y_test, y
```

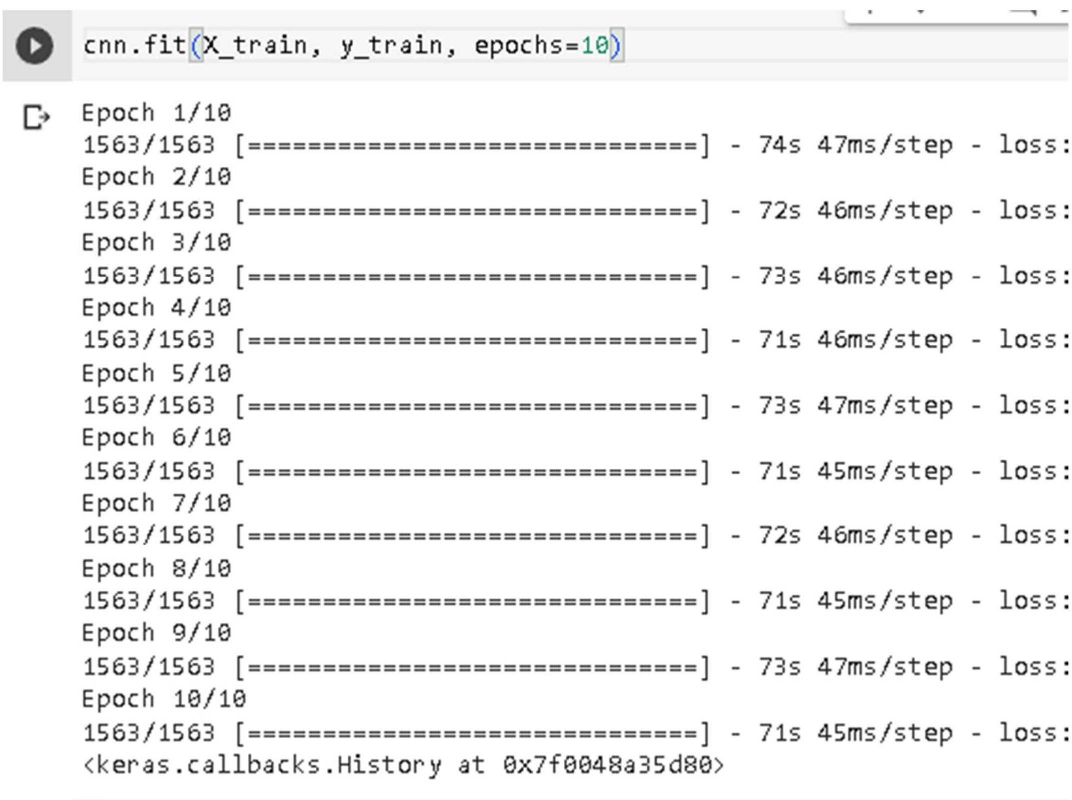

```
Classification Report:
               precision    recall  f1-score   support

           0       0.59      0.49      0.54      1000
           1       0.74      0.36      0.48      1000
           2       0.38      0.20      0.27      1000
           3       0.34      0.25      0.29      1000
           4       0.57      0.12      0.19      1000
           5       0.33      0.47      0.39      1000
           6       0.24      0.88      0.38      1000
           7       0.67      0.33      0.44      1000
           8       0.69      0.51      0.59      1000
           9       0.56      0.54      0.55      1000

    accuracy                           0.41     10000
   macro avg       0.51      0.41      0.41     10000
weighted avg       0.51      0.41      0.41     10000
```

```
cnn = models.Sequential([
```

```
    layers.Conv2D(filters=32, kernel_size=(3, 3), activation='relu',
input_shape=(32, 32, 3)),
    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(filters=64, kernel_size=(3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),

    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax')
])
```

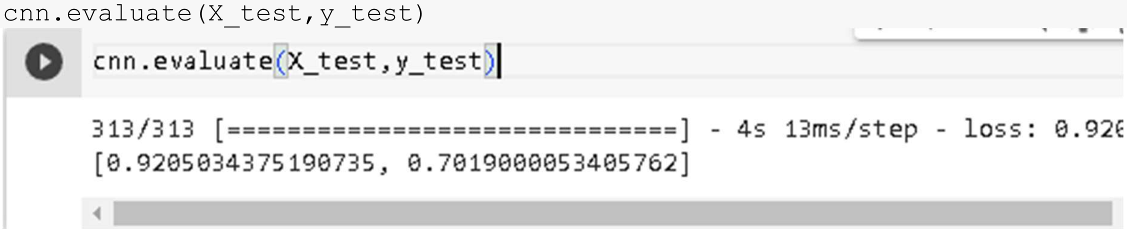
```
cnn.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

```
cnn.fit(X_train, y_train, epochs=10)
```



```
cnn.fit(X_train, y_train, epochs=10)

Epoch 1/10
1563/1563 [==============================] - 74s 47ms/step - loss:
Epoch 2/10
1563/1563 [==============================] - 72s 46ms/step - loss:
Epoch 3/10
1563/1563 [==============================] - 73s 46ms/step - loss:
Epoch 4/10
1563/1563 [==============================] - 71s 46ms/step - loss:
Epoch 5/10
1563/1563 [==============================] - 73s 47ms/step - loss:
Epoch 6/10
1563/1563 [==============================] - 71s 45ms/step - loss:
Epoch 7/10
1563/1563 [==============================] - 72s 46ms/step - loss:
Epoch 8/10
1563/1563 [==============================] - 71s 45ms/step - loss:
Epoch 9/10
1563/1563 [==============================] - 73s 47ms/step - loss:
Epoch 10/10
1563/1563 [==============================] - 71s 45ms/step - loss:
<keras.callbacks.History at 0x7f0048a35d80>
```

```
cnn.evaluate(X_test,y_test)
```



```
cnn.evaluate(X_test,y_test)

313/313 [==============================] - 4s 13ms/step - loss: 0.920
[0.9205034375190735, 0.7019000053405762]
```

```
y_pred = cnn.predict(X_test)
y_pred[:5]
```

```
y_pred = cnn.predict(X_test)
y_pred[:5]

313/313 [==============================] - 4s 13ms/step
array([[1.2953458e-03, 1.5625084e-02, 9.7176002e-04, 9.2179567e-01,
        6.7511945e-05, 5.3039845e-02, 1.1029240e-03, 3.9241560e-05,
        1.1590639e-03, 4.9034832e-03],
       [6.0114651e-03, 8.4940299e-02, 8.6275963e-07, 1.6291340e-06,
        2.7299544e-08, 1.4836646e-08, 5.2201538e-10, 1.0773475e-08,
        9.0835708e-01, 6.8866793e-04],
       [4.8600271e-02, 7.7459343e-02, 1.7677578e-03, 7.1387104e-04,
        4.8549013e-04, 7.5397038e-05, 1.1611767e-04, 2.7459415e-04,
        8.2916188e-01, 4.1345209e-02],
       [9.3123561e-01, 3.5838925e-03, 2.6432954e-02, 3.0681299e-04,
        1.0242155e-04, 7.7731947e-06, 5.9064700e-05, 4.6392943e-06,
        3.8172353e-02, 9.4434283e-05],
       [7.1224264e-07, 4.4954545e-06, 2.4407450e-03, 4.1027173e-02,
        8.8628292e-01, 5.0876092e-04, 6.9584943e-02, 1.3673306e-04,
        1.1855822e-05, 1.5827114e-06]], dtype=float32)
```

```
y_classes = [np.argmax(element) for element in y_pred]
y_classes[:5]
```

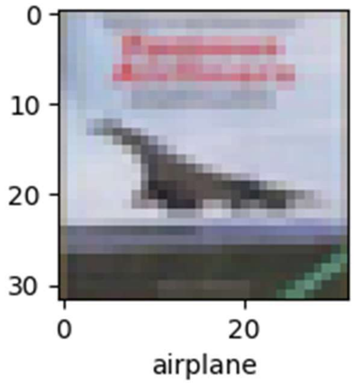
```
y_classes = [np.argmax(element) for element in y_pred]
y_classes[:5]

[3, 8, 8, 0, 4]
```

```
y_test[:5]
```

```
y_test[:5]

array([3, 8, 8, 0, 6], dtype=uint8)
```

```
plot_sample(X_test, y_test,3)
```

```
plot_sample(X_test, y_test,3)
```




```
[ ]  classes[y_classes[3]]
```

```
'airplane'
```


```
[ ]  classes[y_classes[3]]
```

```
'airplane'
```

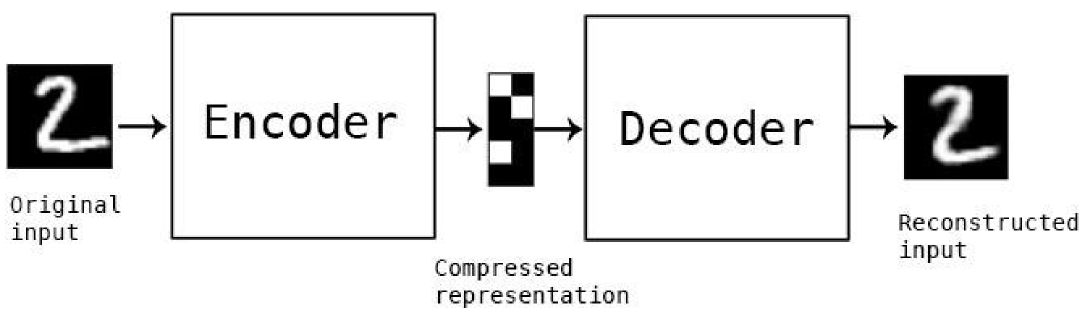
Practical 4

Aim: Implement deep learning for the Prediction of the autoencoder from the test data (e.g., MNIST data set)

An autoencoder is actually an Artificial Neural Network that is used to decompress and compress the input data provided in an unsupervised manner. Decompression and compression operations are lossy and data specific.

Data specific means that the autoencoder will only be able to actually compress the data on which it has been trained. For example, if you train an autoencoder with images of dogs, then it will give a bad performance for cats. The autoencoder plans to learn the representation which is known as the encoding for a whole set of data. This can result in the reduction of the dimensionality by the training network. The reconstruction part is also learned with this.

Lossy operations mean that the reconstructed image is often not an as sharp or high resolution in quality as the original one and the difference is greater for reconstructions with a greater loss and this is known as a lossy operation. The following image shows how the image is encoded and decoded with a certain loss factor.



The Autoencoder is a particular type of feed-forward neural network and the input should be similar to the output. Hence we would need an encoding method, loss function, and a decoding method. The end goal is to perfectly replicate the input with minimum loss.

The Input will be passed through a layer of encoders which are actually a fully connected neural network that also makes the code decoder and hence use the same code for encoding and decoding like an ANN.

Code:

```
from keras.layers import Dense,Conv2D,MaxPooling2D,UpSampling2D

from keras import Input, Model

from keras.datasets import mnist

import numpy as np

import matplotlib.pyplot as plt
```

```
encoding_dim = 15
input_img = Input(shape=(784,))
# encoded representation of input
encoded = Dense(encoding_dim, activation='relu')(input_img)
# decoded representation of code
decoded = Dense(784, activation='sigmoid')(encoded)
# Model which take input image and shows decoded images
autoencoder = Model(input_img, decoded)

# This model shows encoded images
encoder = Model(input_img, encoded)
# Creating a decoder model
encoded_input = Input(shape=(encoding_dim,))
# last layer of the autoencoder model
decoder_layer = autoencoder.layers[-1]
# decoder model
decoder = Model(encoded_input, decoder_layer(encoded_input))

autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
(x_train, y_train), (x_test, y_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)
```

Output:

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 1s 0us/step
```

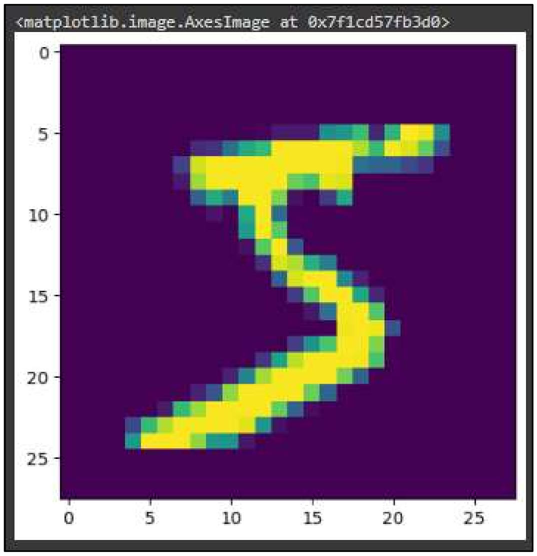
(60000, 784)

(10000, 784)

Code:

plt.imshow(x_train[0].reshape(28,28))

Output:




```
autoencoder.fit(x_train, x_train,
                epochs=15,
                batch_size=256,
                validation_data=(x_test, x_test))
```

Output:

Epoch 1/15

235/235 [==============================] - 4s 13ms/step - loss: 0.3108 - val_loss: 0.2170

Epoch 2/15

235/235 [==============================] - 2s 11ms/step - loss: 0.1982 - val_loss: 0.1811

Epoch 3/15

235/235 [==============================] - 4s 16ms/step - loss: 0.1731 - val_loss: 0.1642

Epoch 4/15

235/235 [==============================] - 3s 11ms/step - loss: 0.1605 - val_loss: 0.1551

Epoch 5/15

235/235 [==============================] - 3s 11ms/step - loss: 0.1531 - val_loss: 0.1490

Epoch 6/15

235/235 [==============================] - 3s 11ms/step - loss: 0.1482 - val_loss: 0.1452

Epoch 7/15

235/235 [==============================] - 3s 12ms/step - loss: 0.1451 - val_loss: 0.1424

Epoch 8/15

235/235 [==============================] - 3s 14ms/step - loss: 0.1428 - val_loss: 0.1404

Epoch 9/15

235/235 [==============================] - 3s 11ms/step - loss: 0.1409 - val_loss: 0.1387

Epoch 10/15

235/235 [==============================] - 3s 11ms/step - loss: 0.1393 - val_loss: 0.1372

Epoch 11/15

235/235 [==============================] - 3s 11ms/step - loss: 0.1379 - val_loss: 0.1360

Epoch 12/15

235/235 [==============================] - 3s 13ms/step - loss: 0.1368 - val_loss: 0.1349

Epoch 13/15

235/235 [==============================] - 3s 13ms/step - loss: 0.1359 - val_loss: 0.1340

Epoch 14/15

235/235 [==============================] - 3s 11ms/step - loss: 0.1352 - val_loss: 0.1335

Epoch 15/15

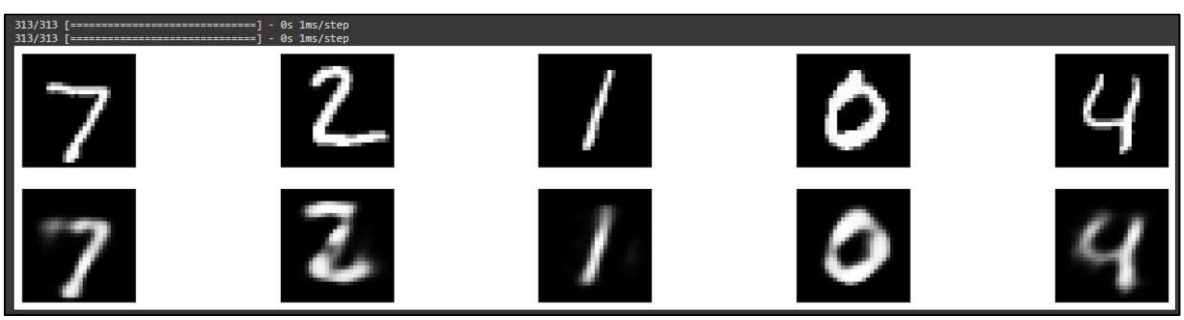
235/235 [==============================] - 3s 11ms/step - loss: 0.1346 - val_loss: 0.1329

<keras.callbacks.History at 0x7f1c44c1f7c0>

Code:

```
encoded_img = encoder.predict(x_test)

decoded_img = decoder.predict(encoded_img)

plt.figure(figsize=(20, 4))

for i in range(5):

    # Display original

    ax = plt.subplot(2, 5, 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 reconstruction

    ax = plt.subplot(2, 5, i + 1 + 5)

    plt.imshow(decoded_img[i].reshape(28, 28))

    plt.gray()

    ax.get_xaxis().set_visible(False)

    ax.get_yaxis().set_visible(False)

plt.show()
```

Output:



Practical 5

Aim: Implement Convolutional Neural Network for Digit Recognition on the MNIST Dataset.

Convolutional Neural Network (CNN): A is a type of Deep Learning neural network architecture commonly used in Computer Vision. Computer vision is a field of Artificial Intelligence that enables a computer to understand and interpret the image or visual data. When it comes to Machine Learning, Artificial Neural Networks perform really well. Neural Networks are used in various datasets like images, audio, and text. Different types of Neural Networks are used for different purposes, for example for predicting the sequence of words we use Recurrent Neural Networks more precisely an LSTM, similarly for image classification we use Convolution Neural networks. In this blog, we are going to build a basic building block for CNN.
In a regular Neural Network there are three types of layers:

1. **Input Layers:** It's the layer in which we give input to our model. The number of neurons in this layer is equal to the total number of features in our data (number of pixels in the case of an image).
2. **Hidden Layer:** The input from the Input layer is then feed into the hidden layer. There can be many hidden layers depending upon our model and data size. Each hidden layer can have different numbers of neurons which are generally greater than the number of features. The output from each layer is computed by matrix multiplication of output of the previous layer with learnable weights of that layer and then by the addition of learnable biases followed by activation function which makes the network nonlinear.
3. **Output Layer:** The output from the hidden layer is then fed into a logistic function like sigmoid or softmax which converts the output of each class into the probability score of each class.

MNIST Dataset : The MNIST dataset (Modified National Institute of Standards and Technology database) is one of the most popular datasets in machine learning. MNIST is a dataset of 60,000 square 28×28 pixel images of handwritten single digits between 0 and 9. The images are in grayscale format.

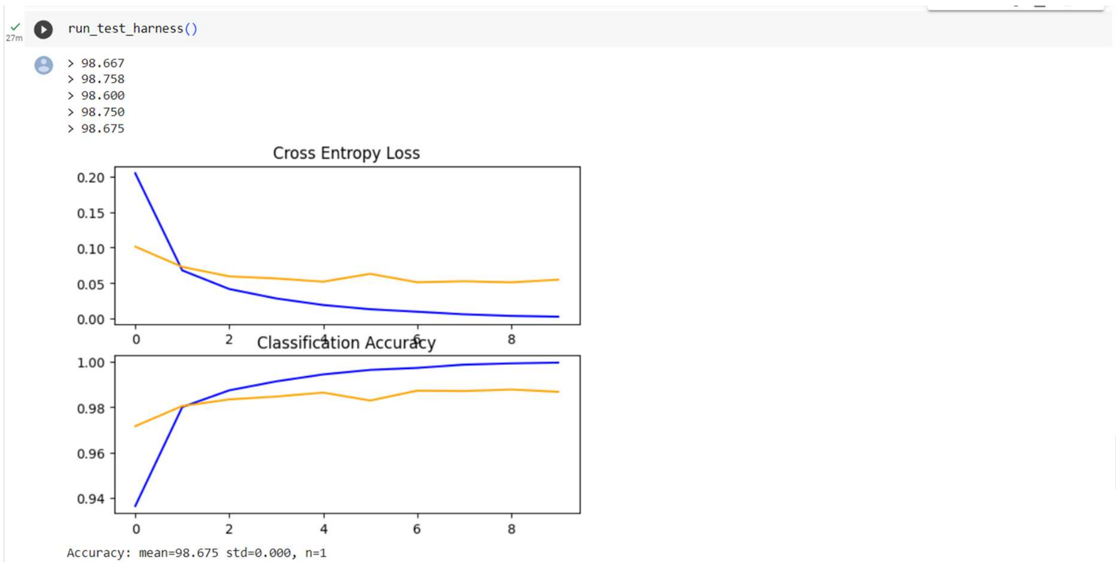
Code:

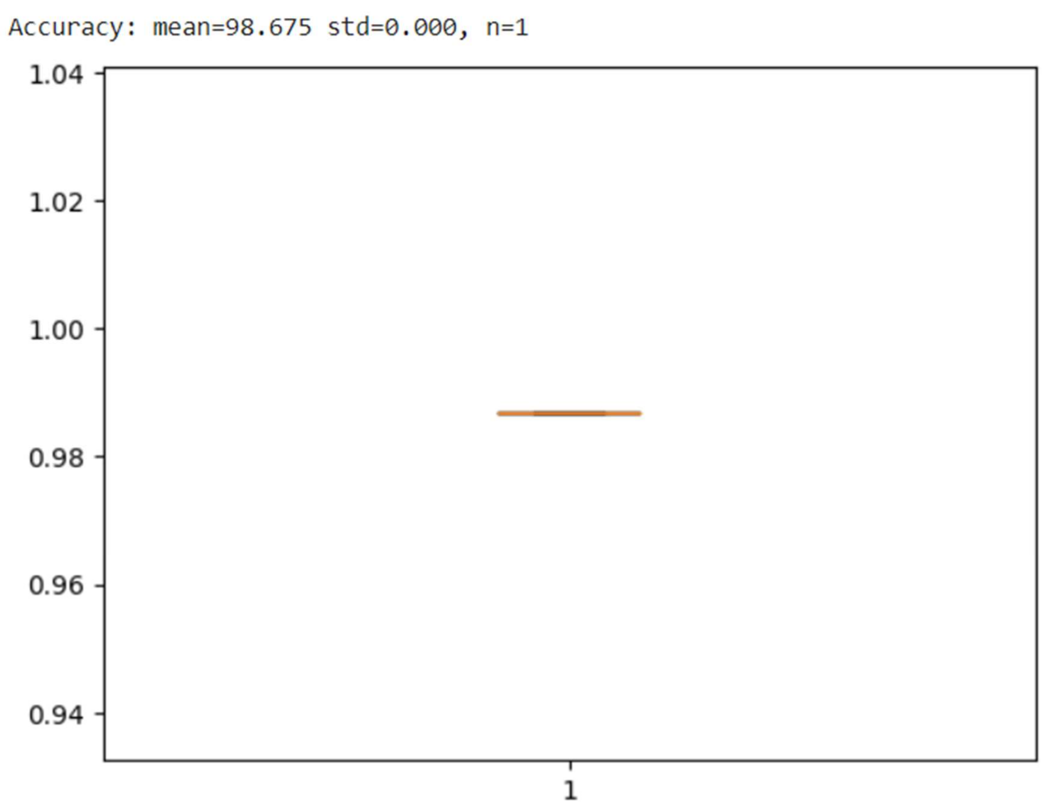
```
# baseline cnn model for mnist
from numpy import mean
from numpy import std
from matplotlib import pyplot as plt
from sklearn.model_selection import KFold
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import MaxPooling2D
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Flatten
from tensorflow.keras.optimizers import SGD

# load train and test dataset
```

```
def load_dataset():
 # load dataset
 (trainX, trainY), (testX, testY) = mnist.load_data()
 # reshape dataset to have a single channel
 trainX = trainX.reshape((trainX.shape[0], 28, 28, 1))
 testX = testX.reshape((testX.shape[0], 28, 28, 1))
 # one hot encode target values
 trainY = to_categorical(trainY)
 testY = to_categorical(testY)
 return trainX, trainY, testX, testY

# scale pixels
def prep_pixels(train, test):
 # convert from integers to floats
 train_norm = train.astype('float32')
 test_norm = test.astype('float32')
 # normalize to range 0-1
 train_norm = train_norm / 255.0
 test_norm = test_norm / 255.0
 # return normalized images
 return train_norm, test_norm

# define cnn model
def define_model():
 model = Sequential()
 model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform',
input_shape=(28, 28, 1)))
 model.add(MaxPooling2D((2, 2)))
 model.add(Flatten())
 model.add(Dense(100, activation='relu', kernel_initializer='he_uniform'))
 model.add(Dense(10, activation='softmax'))
 # compile model
 opt = SGD(learning_rate=0.01, momentum=0.9)
 model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
 return model

# evaluate a model using k-fold cross-validation
def evaluate_model(dataX, dataY, n_folds=5):
 scores, histories = list(), list()
 # prepare cross validation
 kfold = KFold(n_folds, shuffle=True, random_state=1)
 # enumerate splits
 for train_ix, test_ix in kfold.split(dataX):
 # define model
  model = define_model()
 # select rows for train and test
  trainX, trainY, testX, testY = dataX[train_ix], dataY[train_ix], dataX[test_ix],
dataY[test_ix]
 # fit model
```

```
  history = model.fit(trainX, trainY, epochs=10, batch_size=32, validation_data=(testX,
testY), verbose=0)
 # evaluate model
  _, acc = model.evaluate(testX, testY, verbose=0)
  print('> %.3f' % (acc * 100.0))
 # stores scores
 scores.append(acc)
 histories.append(history)
 return scores, histories

def summarize_diagnostics(histories):
 for i in range(len(histories)):
  # plot loss
  plt.subplot(2, 1, 1)
  plt.title('Cross Entropy Loss')
  plt.plot(histories[i].history['loss'], color='blue', label='train')
  plt.plot(histories[i].history['val_loss'], color='orange', label='test')
  # plot accuracy
  plt.subplot(2, 1, 2)
  plt.title('Classification Accuracy')
  plt.plot(histories[i].history['accuracy'], color='blue', label='train')
  plt.plot(histories[i].history['val_accuracy'], color='orange', label='test')
  plt.show()

# summarize model performance
def summarize_performance(scores):
 # print summary
 print('Accuracy: mean=%.3f std=%.3f, n=%d' % (mean(scores)*100, std(scores)*100,
len(scores)))
 # box and whisker plots of results
 plt.boxplot(scores)
 plt.show()

def run_test_harness():
 # load dataset
 trainX, trainY, testX, testY = load_dataset()
 # prepare pixel data
 trainX, testX = prep_pixels(trainX, testX)
 # evaluate model
 scores, histories = evaluate_model(trainX, trainY)
 # learning curves
 summarize_diagnostics(histories)
 # summarize estimated performance
 summarize_performance(scores)

run_test_harness()
```

Output:


run_test_harness()
> 98.667
> 98.758
> 98.600
> 98.750
> 98.675
Cross Entropy Loss
Classification Accuracy
Accuracy: mean=98.675 std=0.000, n=1


Accuracy: mean=98.675 std=0.000, n=1

Practical No. 6

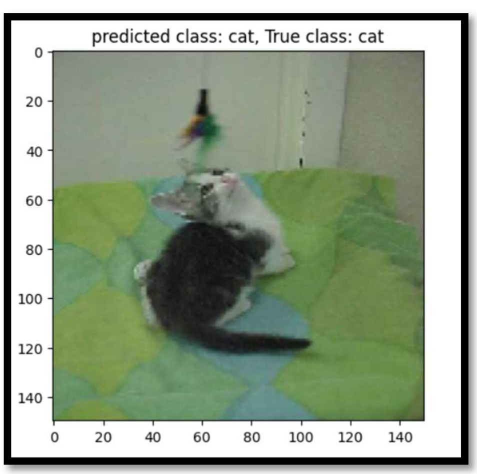
Aim: Write a program to implement Transfer Learning on the suitable dataset (e.g. classify the cats versus dogs dataset from Kaggle).

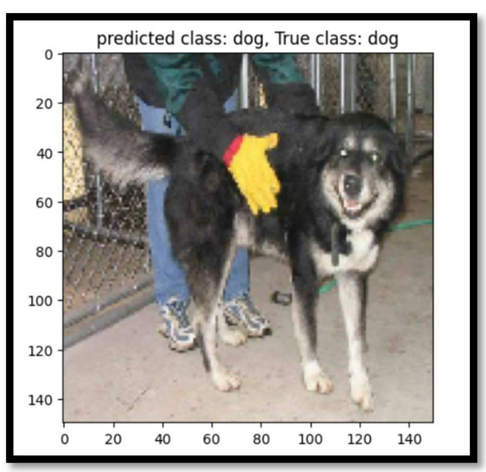
Source Code:-

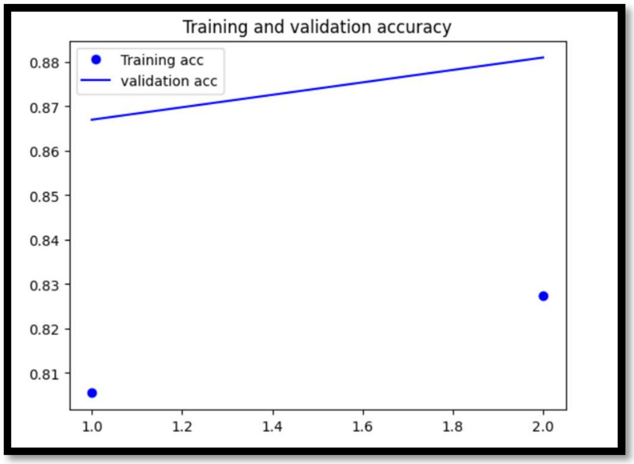
```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import os
import zipfile
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import VGG16

filename = "cats_and_dogs_filtered.zip"

with zipfile.ZipFile("cats_and_dogs_filtered.zip", "r") as zip_ref :
  zip_ref.extractall()

train_dir = os.path.join(os.getcwd(),"cats_and_dogs_filtered","train")
train_dir

validation_dir = os.path.join(os.getcwd(),"cats_and_dogs_filtered","validation")

train_datagen =
ImageDataGenerator(rescale=1./255,rotation_range=20,width_shift_range=0.2,height_shift_r
ange=0.2,shear_range=0.2,zoom_range=0.2,horizontal_flip=True)

validate_datagen = ImageDataGenerator(rescale=1./255)

train_generator =
train_datagen.flow_from_directory(train_dir,target_size=(150,150),batch_size=20,class_mod
e="binary")

validation_generator =
validate_datagen.flow_from_directory(validation_dir,target_size=(150,150),batch_size=20,cl
ass_mode="binary")

conv_base = VGG16(weights="imagenet",include_top=False, input_shape=(150,150,3))

conv_base.trainable = False

model = tf.keras.models.Sequential()
model.add(conv_base)
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(256,activation ="relu"))
model.add(tf.keras.layers.Dropout(0.5))
model.add(tf.keras.layers.Dense(1,activation = "sigmoid"))
```

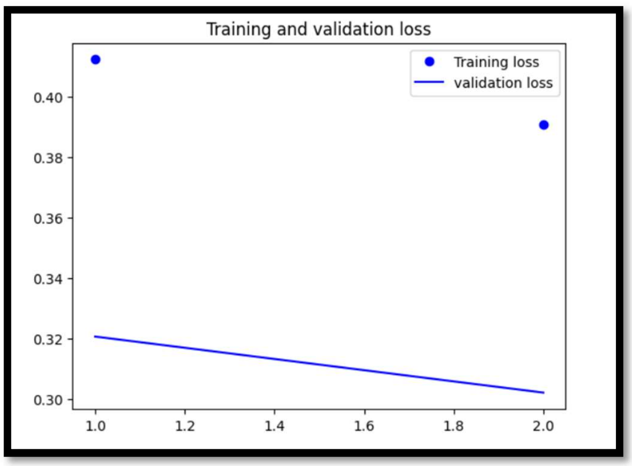
```
model.compile(loss="binary_crossentropy",optimizer=tf.keras.optimizers.RMSprop(learning
_rate=2e-5),metrics=["accuracy"])

history = model.fit(train_generator,steps_per_epoch=100, epochs=2,
validation_data=validation_generator,validation_steps=50)

x, y_true = next(validation_generator)
y_pred = model.predict(x)
class_names = ['cat', 'dog']
for i in range(len(x)):
  plt.imshow(x[i])
  plt.title(f'predicted class: {class_names[int(round(y_pred[i][0]))]}, True class:
{class_names[int(y_true[i])]}')
  plt.show()

acc = history.history["accuracy"]
val_acc = history.history["val_accuracy"]
loss = history.history["loss"]
val_loss = history.history["val_loss"]

epochs = range(1, len(acc) +1)
plt.plot(epochs, acc, "bo", label="Training acc")
plt.plot(epochs, val_acc, "b", label="validation acc")
plt.title("Training and validation accuracy")
plt.legend()
plt.figure()
plt.plot(epochs, loss, "bo", label="Training loss")
plt.plot(epochs, val_loss, "b", label="validation loss")
plt.title("Training and validation loss")
plt.legend()
plt.show()
```

Output:-






Training and validation accuracy
Training acc
validation acc
0.88
0.87
0.86
0.85
0.84
0.83
0.82
0.81
1.0
1.2
1.4
1.6
1.8
2.0


Training and validation loss
Training loss
validation loss
0.40
0.38
0.36
0.34
0.32
0.30
1.0
1.2
1.4
1.6
1.8
2.0

Practical 7

Aim: - Write a program for the Implementation of a Generative Adversarial Network for generating synthetic shapes (like digits).

Generative Adversarial Network

Generative Adversarial Networks (GANs) can indeed be used to generate synthetic shapes. GANs are a class of machine learning models that consist of two main components: a generator and a discriminator.

The generator's role is to generate synthetic data, in this case, shapes. It takes random noise as input and tries to produce realistic shapes based on that noise. The discriminator, on the other hand, acts as a critic and tries to distinguish between real shapes from a training dataset and the synthetic shapes generated by the generator.

During training, the generator and discriminator are pitted against each other in a game-like setting. The generator aims to produce shapes that can fool the discriminator into believing they are real, while the discriminator tries to accurately classify real and synthetic shapes. This adversarial process helps the generator improve its ability to generate more realistic shapes over time.

To generate synthetic shapes, you would typically represent shapes using vectors or matrices, where each element represents a pixel or a feature of the shape. The generator would take random noise vectors as input and generate shape representations, which can then be converted into visual outputs.

There are various techniques and architectures that can be employed for shape generation using GANs. For instance, convolutional neural networks (CNNs) can be used as the underlying architecture for both the generator and discriminator to capture spatial information and generate complex shapes.

GANs have been successfully applied to generate synthetic shapes in various domains, such as computer graphics, computer vision, and even in creative applications like artwork generation. They have the potential to generate diverse and novel shapes, which can be useful for tasks like data augmentation, creating synthetic datasets, or exploring new design possibilities.

It's worth noting that training GANs can be challenging, requiring careful tuning of hyperparameters, architecture design, and large amounts of training data. Furthermore, GANs tend to be sensitive to the choice of loss functions and can suffer from mode collapse, where the generator produces limited variations of shapes. However, with appropriate techniques and advancements, GANs can generate impressive and realistic synthetic shapes.

Source Code:

```
from numpy import expand_dims

from numpy import ones
```

```
from numpy import zeros
from numpy.random import rand
from numpy.random import randint
from keras.datasets.mnist import load_data
from keras.optimizers import Adam
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Conv2D
from keras.layers import Flatten
from keras.layers import Dropout
from keras.layers import LeakyReLU

# define the standalone discriminator model
def define_discriminator(in_shape=(28,28,1)):
 model = Sequential()
 model.add(Conv2D(64, (3,3), strides=(2, 2), padding='same', input_shape=in_shape))
 model.add(LeakyReLU(alpha=0.2))
 model.add(Dropout(0.4))
 model.add(Conv2D(64, (3,3), strides=(2, 2), padding='same'))
 model.add(LeakyReLU(alpha=0.2))
 model.add(Dropout(0.4))
 model.add(Flatten())
 model.add(Dense(1, activation='sigmoid'))
 # compile model
 opt = Adam(lr=0.0002, beta_1=0.5)
 model.compile(loss='binary_crossentropy', optimizer=opt, metrics=['accuracy'])
 return model

# load and prepare mnist training images
def load_real_samples():
```

```
 # load mnist dataset
 (trainX, _), (_, _) = load_data()
 # expand to 3d, e.g. add channels dimension
 X = expand_dims(trainX, axis=-1)
 # convert from unsigned ints to floats
 X = X.astype('float32')
 # scale from [0,255] to [0,1]
 X = X / 255.0
 return X

# select real samples
def generate_real_samples(dataset, n_samples):
 # choose random instances
 ix = randint(0, dataset.shape[0], n_samples)
 # retrieve selected images
 X = dataset[ix]
 # generate 'real' class labels (1)
 y = ones((n_samples, 1))
 return X, y

# generate n fake samples with class labels
def generate_fake_samples(n_samples):
 # generate uniform random numbers in [0,1]
 X = rand(28 * 28 * n_samples)
 # reshape into a batch of grayscale images
 X = X.reshape((n_samples, 28, 28, 1))
 # generate 'fake' class labels (0)
 y = zeros((n_samples, 1))
 return X, y
```

```
# train the discriminator model
def train_discriminator(model, dataset, n_iter=100, n_batch=256):
 half_batch = int(n_batch / 2)
 # manually enumerate epochs
 for i in range(n_iter):
 # get randomly selected 'real' samples
  X_real, y_real = generate_real_samples(dataset, half_batch)
 # update discriminator on real samples
  _, real_acc = model.train_on_batch(X_real, y_real)
 # generate 'fake' examples
  X_fake, y_fake = generate_fake_samples(half_batch)
 # update discriminator on fake samples
  _, fake_acc = model.train_on_batch(X_fake, y_fake)
 # summarize performance
  print('>%d real=%.0f%% fake=%.0f%%' % (i+1, real_acc*100, fake_acc*100))

# define the discriminator model
model = define_discriminator()
# load image data
dataset = load_real_samples()
# fit the model
train_discriminator(model, dataset)
```

Output:-

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 0s 0us/step
/usr/local/lib/python3.10/dist-packages/keras/optimizers/legacy/adam.py:117: UserWarning: The `lr` argument is deprecated, use `learning_rate` instea
  super().__init__(name, **kwargs)
>1 real=56% fake=31%
>2 real=64% fake=50%
>3 real=67% fake=63%
>4 real=68% fake=78%
>5 real=64% fake=94%
>6 real=70% fake=99%
>7 real=70% fake=99%
>8 real=62% fake=100%
>9 real=60% fake=100%
>10 real=66% fake=100%
>11 real=62% fake=100%
>12 real=68% fake=100%
>13 real=70% fake=100%
>14 real=70% fake=100%
>15 real=78% fake=100%
>16 real=65% fake=100%
>17 real=79% fake=100%
```

Practical 8

Aim: Write a program to implement a simple form of a recurrent neural network.
a. E.g. (4-to-1 RNN) to show that the quantity of rain on a certain day also depends on the values of the previous day
b. LSTM for sentiment analysis on datasets like UMICH SI650 for similar.

- **RNN**

  RNN stands for Recurrent Neural Network. It is a type of artificial neural network that is specifically designed for processing sequential data. RNNs have connections between the nodes that form a directed cycle, allowing information to persist and be passed from one step to the next.

  The key feature of RNNs is their ability to capture and utilize temporal dependencies in sequential data. Unlike feedforward neural networks, which process each input independently, RNNs can take into account the context and information from previous inputs in the sequence. This makes them well-suited for tasks such as natural language processing, speech recognition, machine translation, and time series analysis.

  In an RNN, each node (or "cell") receives an input and produces an output while maintaining an internal hidden state. The hidden state serves as the memory of the network, allowing it to capture and remember information from previous inputs. The output of each node is typically fed into the next node in the sequence, creating a recurrent connection.

  One of the challenges with traditional RNNs is that they can suffer from the "vanishing gradient" problem, which makes it difficult to capture long-term dependencies. To address this, various RNN variants have been developed, such as the Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU), which have gating mechanisms that help alleviate the vanishing gradient problem.

  Overall, RNNs are powerful models for processing sequential data and have been widely used in many applications where the order and context of the data are important.

a. E.g. (4-to-1 RNN) to show that the quantity of rain on a certain day also depends on the values of the previous day.

Source code:

```
import tensorflow as tf

import numpy as np

import matplotlib.pyplot as plt


rain_data = np.array([2.3, 1.5, 3.1, 2.0, 2.5, 1.7, 2.9, 3.5, 3.0, 2.1,
```

```
2.5, 2.2, 2.8, 3.2, 1.8, 2.7, 1.9, 3.1, 3.3, 2.0,
2.5, 2.2, 2.4, 3.0, 2.1, 2.5, 3.2, 3.1, 1.9, 2.7,
2.2, 2.8, 3.1, 2.0, 2.5, 1.7, 2.9, 3.5, 3.0, 2.1,
2.5, 2.2, 2.8, 3.2, 1.8, 2.7, 1.9, 3.1, 3.3, 2.0])


def create_sequences(values, time_steps):
  x = []
  y = []
  for i in range(len(values)-time_steps):
    x.append(values[i:i+time_steps])
    y.append(values[i+time_steps])
  return np.array(x), np.array(y)
time_steps = 4
x_train, y_train = create_sequences(rain_data, time_steps)


model = tf.keras.models.Sequential([tf.keras.layers.SimpleRNN(8, input_shape=(time_steps,
1)),tf.keras.layers.Dense(1)])


model.compile(optimizer="adam", loss="mse")


history = model.fit(x_train.reshape(-1, time_steps, 1), y_train, epochs=100)


loss = history.history["loss"]
epochs = range(1, len(loss) + 1)
plt.plot(epochs, loss, "bo", label="Training loss")
plt.title("Training loss")
plt.legend()
plt.show()
test_sequence = np.array([2.5, 2.2, 2.8, 3.2])
x_test = np.array([test_sequence])
y_test = model.predict(x_test.reshape(-1, time_steps, 1))
```

```
print("Previous days' rain data:", test_sequence)
print("Expected rain amount for next day:", y_test[0][0])
prediction = model.predict(np.array([test_sequence]).reshape(1, time_steps, 1))
print("Prediction:", prediction[0][0])
```

Output:

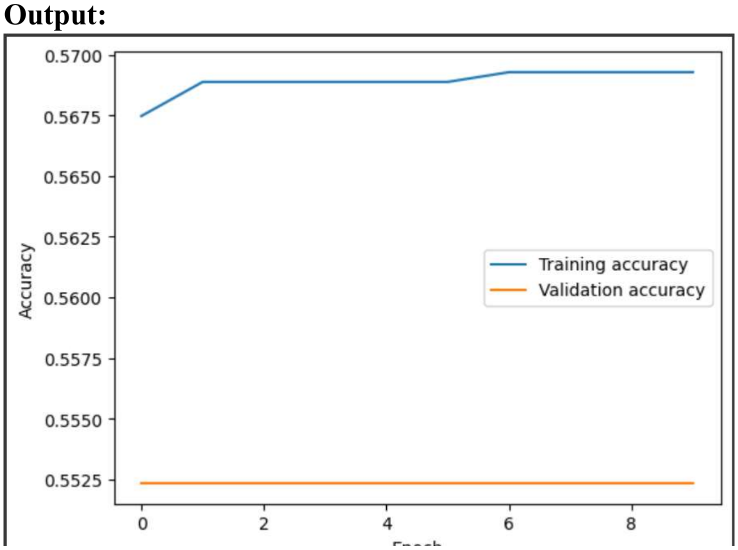
```
Previous days' rain data: [2.5 2.2 2.8 3.2]
Expected rain amount for next day: 1.645736
1/1 [==============================] - 0s 15ms/step
Prediction: 1.645736
```

Source code:

```
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
data = pd.read_csv("/content/training (2).txt", delimiter="\t", names=["label", "text"])

X_train, X_test, y_train, y_test = train_test_split(data["text"],data["label"], test_size=0.2,
random_state=42)

tokenizer = Tokenizer(num_words=5000, oov_token="<OOV>")
tokenizer.fit_on_texts(X_train)

X_train_seq = tokenizer.texts_to_sequences(X_train)
X_test_seq = tokenizer.texts_to_sequences(X_test)

max_length = 100
X_train_pad = pad_sequences(X_train_seq, maxlen=max_length,
padding="post",truncating="post")
X_test_pad = pad_sequences(X_test_seq, maxlen=max_length,
padding="post",truncating="post")
```

```
model = tf.keras.models.Sequential([tf.keras.layers.Embedding(input_dim=5000,
output_dim=32,input_length=max_length),tf.keras.layers.LSTM(units=64, dropout=0.2,
recurrent_dropout=0.2),tf.keras.layers.Dense(1, activation="sigmoid")])

model.compile(optimizer="adam", loss="binary_crossentropy",metrics=["accuracy"])

history = model.fit(X_train_pad, y_train, epochs=10, batch_size=32,validation_split=0.1)

loss, accuracy = model.evaluate(X_test_pad, y_test)
print("Test loss:", loss)
print("Test accuracy:", accuracy)
plt.plot(history.history["accuracy"], label="Training accuracy")
plt.plot(history.history["val_accuracy"], label="Validation accuracy")
plt.xlabel("Epoch")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
predictions = model.predict(X_test_pad)

index = np.random.randint(0, len(X_test_pad))
text = tokenizer.sequences_to_texts([X_test_pad[index]])[0]
label = y_test.values[index]
prediction = predictions[index][0]
print("Text:", text)
print("Actual label:", label)
print("Predicted label:", round(prediction))
```

Output:



```
44/44 [==============================] - 1s 20ms/step
Text: i hate harry potter it's retarted gay and stupid and there's only one black guy <OOV> <OOV> <OOV> <OOV> <
Actual label: 0
Predicted label: 1
```

- **LSTM**

  LSTM stands for Long Short-Term Memory. It is a type of recurrent neural network (RNN) architecture that addresses the limitations of traditional RNNs in capturing long-term dependencies in sequential data.

  LSTMs were introduced by Hochreiter and Schmidhuber in 1997 and have become a popular choice for various tasks such as natural language processing, speech recognition, and time series analysis.

  The key idea behind LSTMs is the inclusion of memory cells and gating mechanisms, which enable them to selectively remember and forget information over long sequences. These memory cells allow LSTMs to learn and retain information for longer durations, making them effective in capturing dependencies that span across multiple time steps.

  The main components of an LSTM cell are as follows:

  Cell State (Ct): It serves as the memory of the LSTM. It runs linearly through time and has the ability to propagate information throughout the entire sequence.

  Input Gate (i): It determines how much of the input at the current time step should be added to the cell state.

  Forget Gate (f): It determines how much of the previous cell state should be forgotten or discarded.

  Output Gate (o): It determines how much of the cell state should be output or exposed to the next layer of the network.

  Hidden State (h): It represents the output of the LSTM cell at a given time step. It is a filtered version of the cell state that is selectively passed to the next time step.

  LSTMs utilize these gates to regulate the flow of information, allowing them to remember or forget specific information over time. This enables LSTMs to effectively handle long sequences and capture long-term dependencies.

  By incorporating memory cells and gating mechanisms, LSTMs have proven to be very effective in a wide range of tasks that involve sequential data. They have become a standard choice for many applications due to their ability to capture long-term dependencies and mitigate the vanishing gradient problem that traditional RNNs often face.

b. LSTM for sentiment analysis on datasets like UMICH SI650 for similar.

```
from __future__ import division, print_function
from keras.layers.core import Dense, Activation
from keras.layers import Embedding
from keras.layers import LSTM
from keras.models import Sequential
from keras.preprocessing import sequence
from sklearn.model_selection import train_test_split
import collections
import nltk
import numpy as np
import collections
import nltk

nltk.download('punkt')  # Download the 'punkt' resource

maxlen = 0
word_freqs = collections.Counter()
num_recs = 0
ftrain = open("umich-sentiment-train.txt", "r")  # Open the file in text mode
for line in ftrain:
    label, sentence = line.strip().split("\t")
    words = nltk.word_tokenize(sentence.lower())
    if len(words) > maxlen:
        maxlen = len(words)
    for word in words:
        word_freqs[word] += 1
    num_recs += 1
ftrain.close()
print("maxlen: %d, vocab size: %d" % (maxlen, len(word_freqs)))
```

Output:

```
print("maxlen: %d, vocab size: %d" % (maxlen, len(word_freqs)))
```

```
maxlen: 42, vocab size: 2268
```