DELHI TECHNOLOGICAL UNIVERSITY

DEPARTMENT OF INFORMATION TECHNOLOGY

AMAN JHA

DEEP LEARNING LAB FILE

INDEX

Experiment – I

Operations on Google Colaboratory

AIM:

Write a program to perform the following operations on Google Colab :

1. Upload a file to Colab
2. Download a file from Colab
3. Change the Colab runtime
4. Install packages in Colab
5. Unzip a file in colab
6. Using matplotlib libray for visualization
7. Exploring the numpy library in python to perform fundamental operations on array.

THEORY:

Google Colaboratory, or “Colab” for short, allows us to write and execute Python in our browser in Interactive Python Notebook format, with

- Zero configuration required
- Free access to GPUs
- Easy sharing

Colab has 3 runtime configuration types, namely:

- CPU : Central processing unit
- GPU : Graphic processing unit
- TPU : Tensor processing unit

Matplotlib is a comprehensive library in python for creating static, animated, and interactive python visualization.

Numpy is a scientific computing library in Python. It offers the implementation of vectors, tensors, matrix computations, mathematical functions, linear algebra methods and various other mathematical tools.

Google colab has a set of basic libraries already installed in its virtual system. However, files and libraries can also be uploaded or installed during run-time.

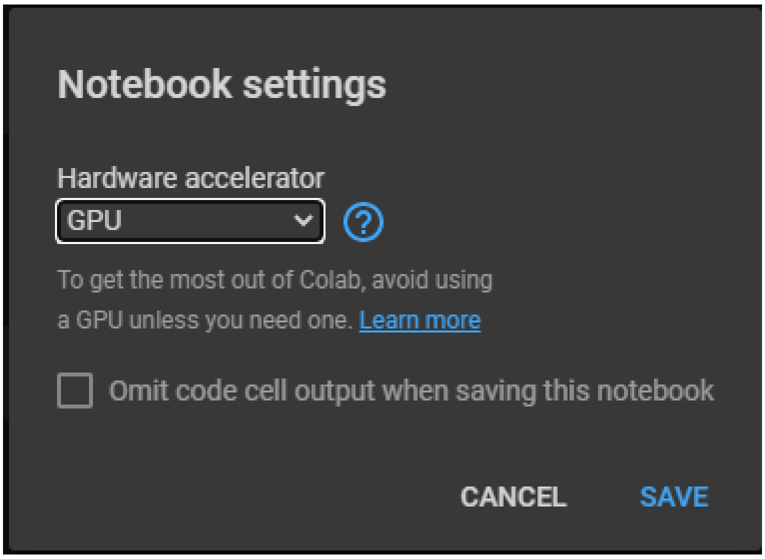
CODE / INPUT / OUTPUT:

```
# Upload a file from Google Colaboratory

from google.colab import files

uploaded = files.upload()

Choose Files No file chosen        Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving WIKI_GOOGL.csv to WIKI_GOOGL.csv
```

Img I : Uploading a file named 'WIKI_GOOGL.csv' to Colab.

```
[ ]  # Download file from Colab

     files.download('WIKI_GOOGL.csv')
```

Img II : Downloading the file from Colab



Img III : Changing the Colab Runtime

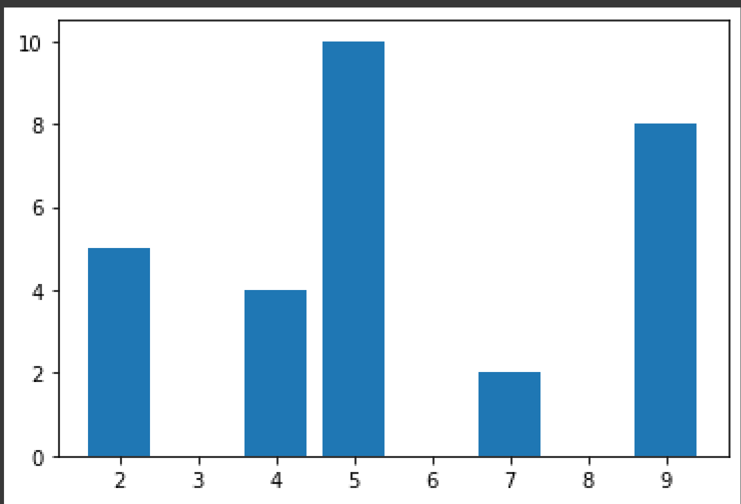
```
[ ]  # Install polyglot package

     !pip install polyglot

     Collecting polyglot
       Downloading https://files.pythonhosted.org/packages/e7/98/e24e2489114c5112b083714277204d92d372f5bbe00d5507acf40370edb9/polyglot-16.7.4.tar.gz (126kB)
          |████████████████████████████████| 133kB 13.1MB/s
     Building wheels for collected packages: polyglot
       Building wheel for polyglot (setup.py) ... done
       Created wheel for polyglot: filename=polyglot-16.7.4-py2.py3-none-any.whl size=52557 sha256=c4e0cb356af271b5f09f8ec322fb9a1f0a3e4a358b02fa9eb3366a19a3324838
       Stored in directory: /root/.cache/pip/wheels/5e/91/ef/f1369fdc1203b0a9347d4b24f149b83a305f39ab047986d9da
     Successfully built polyglot
     Installing collected packages: polyglot
     Successfully installed polyglot-16.7.4
```

Img IV : Installing package 'polyglot' on Colab

```
[ ] # Upload a file from Google Colaboratory

    from google.colab import files

    uploaded = files.upload()

    Choose Files  No file chosen        Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
    Saving WIKI_GOOGL.zip to WIKI_GOOGL.zip

[ ] # Unzip a file from Colab

    !unzip WIKI_GOOGL.zip

    Archive:  WIKI_GOOGL.zip
      inflating: WIKI_GOOGL.csv
```

Img V : Uploading then unzipping a file on Colab : Namely 'WIKI_GOOGL.zip'

```
[ ] # Using matplotlib library for visualization

    from matplotlib import pyplot as plt

    x = [5,2,9,4,7]
    y = [10,5,8,4,2]

    plt.bar(x,y)
    plt.show()
```




```
[ ] # Exploring the Numpy Library

    import numpy as np
    arr = np.array([[1,2,3],[4,5,6]])
    print(arr.size)

    6
```

Img VI : Exploring Matplotlib library

```
[ ] for x in np.nditer(arr): # Iterates throught the array row-major
      print(x)

    1
    2
    3
    4
    5
    6

[ ] arrx  = np.array((1,2,3)) # creates a 1-D array from a tuple
    print(arrx)

    [1 2 3]

[ ] arr1 = np.zeros((3,4)) # Creates a 2-Dimensional Array with all values as 0
    print(arr1)

    [[0. 0. 0. 0.]
     [0. 0. 0. 0.]
     [0. 0. 0. 0.]]

[ ] f = np.arange(0,30,5) # fills the array with numbers in range(0,30) with a step of 5
    print(f)

    [ 0  5 10 15 20 25]

[ ] newarr = arr1.reshape(2,2,3) # reshapes the arrat - horizontal mazor
    print(arr1)
    print("\n\n")
    print(newarr)

    [[0. 0. 0. 0.]
     [0. 0. 0. 0.]
     [0. 0. 0. 0.]]
```

```
    [[[0. 0. 0.]
      [0. 0. 0.]]

     [[0. 0. 0.]
      [0. 0. 0.]]]

[ ] flarr = arr.flatten() # Flattens the array horizontal mazor
    print(flarr)

    [1 2 3 4 5 6]

[ ] temp = arr[:2,::2] # first 2 rows and alternate columns with a step of 2
    temp

    array([[1, 3],
           [4, 6]])
```

Img VII : Exploring numpy library

<u>CONCLUSION</u>:

This experiment helps us to understand the basic use of Google Collaboratory, and implementation of libraries like ‘matplotlib’ and ‘numpy’ in it.

Matplotlib provided various functionalities for visualization of data.

Numpy provided various functionalities for creating vectors & n-dimensional arrays (Matrices) and functions to manipulate and compute on them.

Experiment – II

Pandas library and Data-Frames

AIM:

Write a program using 'Pandas' library to create and display data frames from a CSV and explore the functionalities of data-frames.

THEORY:

Pandas is a fast, powerful and easy to use open source data analysis and manipulation tool, built on top of python language.

Pandas enables us to create data-frames and perform various operations pertaining to its columns and rows.

CODE :

```
[ ] import pandas as pd
```

```
[ ] # insert a list of datatypes into a cell of index 0

    lst = ['Welcome','Ladies','And','Gentleman']

    df = pd.DataFrame(lst)
```

Input I : Importing Pandas and Creating a small data-frame of size (4,1).

```
[ ] import numpy as np

    exam_data = { 'name' : ['Anastasia','Dima','Katherine','James','Emily','Michael','Mathew','Laura','Kevin','Jonas'],
                  'score' : [12.5, 9, 16.5, np.nan, 9, 20, 14.5, np.nan, 8, 19],
                  'attempts' : [1,3,2,3,2,3,1,1,2,1],
                  'qualify' : ['yes','no','yes','no','no','yes','yes','no','no','yes'] }

    labels = ['a','b','c','d','e','f','g','h','i','j']

    df = pd.DataFrame(exam_data, index = labels)
```

Input II : Creating a data-frame 'exam_data' with column's {'name', 'score', 'attempts', 'qualify' } namely with 10 entries.

```
[ ] df['name'] = df['name'].replace('James','John')
```

Input III : Replacing all the 'name' entries named 'James' to 'John'

```
[ ]  # Find the highest score in each of the three attemps

     attm = df.groupby('attempts')
     priceDf = attm['score'].max()
```

Input IV : Grouping data-frame on basis of 'attempts' and finding the max 'score'

```
# Sort all values by score column

scoreDF = df.sort_values(by = ['score'])
```

Input V : Sorting the data entries by 'score'

```
df.pop('attempts')
```

```
color = ['Red','Blue','Orange','Red','White','White','Blue','Green','Green','Red']
df['color'] = color
print("]nNew DataFrame after inserting the 'color' column")
print(df)
```

Input VI : Deleting the Column 'attempts' from the data-frame and adding a new column 'color'

```
[ ]  print('Data from new_file.csv file: ')
     df.to_csv('new_file.csv',sep='\t',index = False)

     Data from new_file.csv file:
```

Input VII : Saving the '.csv' file

```
from google.colab import files

uploaded = files.upload()

Choose Files No file chosen        Upload widget is only available when the cell has been executed in the curr
Saving Automobile_data.csv to Automobile_data.csv

[ ]  path_csv_file = r"/content/drive/MyDrive/Colab Notebooks/Classroom/Automobile_data.csv"
```

```
df = pd.read_csv(path_csv_file)

df.head()[:10]
```

Input VIII : Uploading a '.csv' file namely 'Automobile_data.csv' and displaying the top 10 entries.

```
df['company'].value_counts()
```

Input IX : Displaying the frequency of occurrence of various entries in 'company' column.

```
group_by_company = df.groupby('company')
highest_price = group_by_company['price'].max()
highest_price
```

Input X : Displaying the 'mean' of the prices of automobiles, grouped by 'company'

```
# removing duplicate cars if any

df.drop_duplicates(subset = None,keep = 'first',inplace=False)
```

```
df.sort_values(['price'],axis=0,ascending=True) # sorting in ascending order
```

Input XI : Sorting all the entries in ascending orders of their 'price'

OUTPUT:

	0
0	Welcome
1	Ladies
2	And
3	Gentleman

Output I : Small data-frame of size (4,1)

	name	score	attempts	qualify
a	Anastasia	12.5	1	yes
b	Dima	9.0	3	no
c	Katherine	16.5	2	yes
d	James	NaN	3	no
e	Emily	9.0	2	no
f	Michael	20.0	3	yes
g	Mathew	14.5	1	yes
h	Laura	NaN	1	no
i	Kevin	8.0	2	no
j	Jonas	19.0	1	yes

Output II : Dataframe 'exam_data'

```
        name  score  attempts qualify
a  Anastasia   12.5         1     yes
b       Dima    9.0         3      no
c  Katherine   16.5         2     yes
d       John    NaN         3      no
e      Emily    9.0         2      no
f    Michael   20.0         3     yes
g     Mathew   14.5         1     yes
h      Laura    NaN         1      no
i      Kevin    8.0         2      no
j      Jonas   19.0         1     yes
```

Output III : 'exam_data' after replacing name 'James' with 'John'

```
attempts
1    12.5
2     8.0
3     9.0
Name: score, dtype: float64
```

Output IV : 'max_score' of each attempt

	name	score	attempts	qualify
i	Kevin	8.0	2	no
b	Dima	9.0	3	no
e	Emily	9.0	2	no
a	Anastasia	12.5	1	yes
g	Mathew	14.5	1	yes

Output V : Data-frame sorted by score in ascending order. Showing top 5 entries only.

```
]nNew DataFrame after inserting the 'color' column
        name  score qualify   color
a  Anastasia   12.5     yes     Red
b       Dima    9.0      no    Blue
c  Katherine   16.5     yes  Orange
d       John    NaN      no     Red
e      Emily    9.0      no   White
f    Michael   20.0     yes   White
g     Mathew   14.5     yes    Blue
h      Laura    NaN      no   Green
i      Kevin    8.0      no   Green
j      Jonas   19.0     yes     Red
```

Output VI : Data-frame after popping column 'attempts' and adding column 'color'

	company	body-style	wheel-base	length	engine-type	num-of-cylinders	horsepower	average-mileage	price
0	alfa-romero	convertible	88.6	168.8	dohc	four	111	21	13495.0
1	alfa-romero	convertible	88.6	168.8	dohc	four	111	21	16500.0
2	alfa-romero	hatchback	94.5	171.2	ohcv	six	154	19	16500.0
3	audi	sedan	99.8	176.6	ohc	four	102	24	13950.0
4	audi	sedan	99.4	176.6	ohc	five	115	18	17450.0

Output VIII : Displaying the data-frame 'Automobile_data.csv'

```
toyota           7
bmw              6
nissan           5
mazda            5
mercedes-benz    4
mitsubishi       4
audi             4
volkswagen       4
porsche          3
alfa-romero      3
chevrolet        3
honda            3
isuzu            3
jaguar           3
dodge            2
volvo            2
Name: company, dtype: int64
```

Output IX : Frequency of occurrence, grouped by 'company'

```
company
alfa-romero      20.333333
audi             20.000000
bmw              19.000000
chevrolet        41.000000
dodge            31.000000
honda            26.333333
isuzu            33.333333
jaguar           14.333333
mazda            28.000000
mercedes-benz    18.000000
mitsubishi       29.500000
nissan           31.400000
porsche          17.000000
toyota           28.714286
volkswagen       31.750000
volvo            23.000000
Name: average-mileage, dtype: float64
```

Output X : Mean of prices ranging for cars, grouped by the same 'company'

	company	body-style	wheel-base	length	engine-type	num-of-cylinders	horsepower	average-mileage	price
13	chevrolet	hatchback	88.4	141.1	l	three	48	47	5151.0
27	mazda	hatchback	93.1	159.1	ohc	four	68	30	5195.0
48	toyota	hatchback	95.7	158.7	ohc	four	62	35	5348.0
36	mitsubishi	hatchback	93.7	157.3	ohc	four	68	37	5389.0
28	mazda	hatchback	93.1	159.1	ohc	four	68	31	6095.0
...	...	...	...	...	...	...	...	...	...
11	bmw	sedan	103.5	193.8	ohc	six	182	16	41315.0
35	mercedes-benz	hardtop	112.0	199.2	ohcv	eight	184	14	45400.0
22	isuzu	sedan	94.5	155.9	ohc	four	70	38	NaN
23	isuzu	sedan	94.5	155.9	ohc	four	70	38	NaN
47	porsche	hatchback	98.4	175.7	dohcv	eight	288	17	NaN

61 rows × 9 columns

Output XI : All entries sorted in ascending order of 'prices'

<u>CONCLUSION</u>:

The experiment helps understand the basis use of pandas library in python – a prerequisite for being able to manipulate data-frames in Python, also teaching us how to analyse and manipulate data in accordance per rows and columns.

Experiment – III

Artificial Neural Network

AIM:

Write a Program to implement Artificial Neural Network for MNIST dataset.

THEORY:

A computing architecture consisting of a set of artificial neurons constituting layers. The layers are connected to input and output layers.

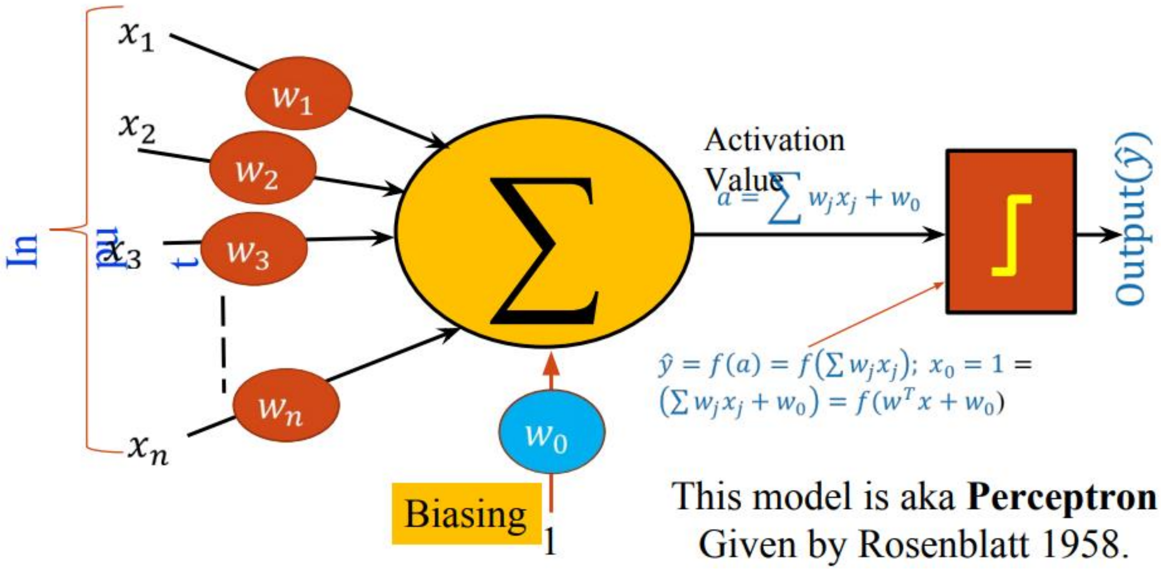


Image I : An artificial neuron

Each hidden layer has a weighted input connection from each of the units in the preceding layer.

The hidden layers, extract high level features from the model in an incremental manner, where feature extraction and classification occur together.

For our experiment, we take **MNIST** dataset consisting images of hand-written digits (0 to 9) of size 28x28 pixels.

The training set contains 60,000 images, while the testing set contains 10,000 images.

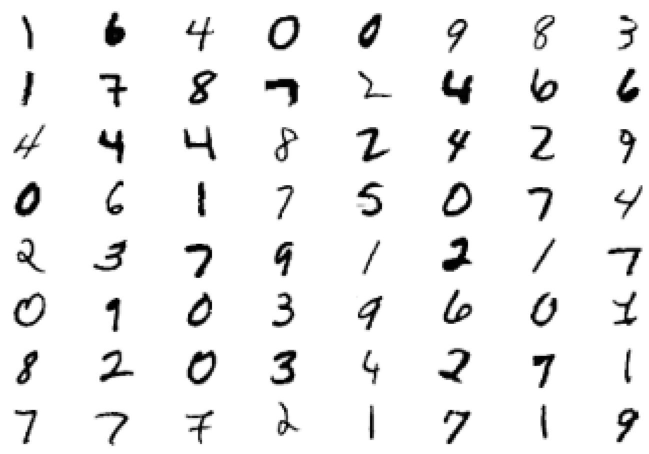


Image II : Showing 64 images from the MNIST training set.

Details about the architecture & Training:

1. Programming Framework: Keras – Python
2. Type of Architecture: Feed-Forward Artificial Neural Network
3. Activation function used: Sigmoid
4. Input Layer: 784 inputs
5. Output Layer: 10 Output neurons (For digits 0 to 9)
6. No. of Hidden Layers: 2
7. No. of Units per Layer : 100

Training Details:

1. Optimizer: Stochastic gradient descent
2. Loss: Categorical Cross entropy
3. Batch-Size: 128
4. Epochs: 100
5. Test-Validation Split: 10%

<u>CODE</u>:

```python
import matplotlib.pyplot as py
import seaborn as sns
import numpy as np
from sklearn.metrics import confusion_matrix

import keras
from keras.datasets import mnist
from keras.layers import Dense
from keras.models import Sequential
from matplotlib import pyplot as plt
from random import randint
```

```python
# Preparing the dataset
# Setup train and test splits

(x_train, y_train),(x_test,y_test) = mnist.load_data()
```

```python
[ ]  image_size = 784 # since its 28 X 28
     x_train = x_train.reshape(x_train.shape[0],image_size)
     x_test = x_test.reshape(x_test.shape[0],image_size)
```

```python
[ ]  num_classes = 10
     y_train = keras.utils.to_categorical(y_train,num_classes)
     y_test = keras.utils.to_categorical(y_test,num_classes)
```

```python
[ ]  model = Sequential() # feed forward - backward propagate
     # eager execution enabled - session online

     model.add(Dense(units = 100,activation = 'sigmoid',input_shape=(image_size,)))
     model.add(Dense(units=100,activation='sigmoid'))
     model.add(Dense(units=num_classes,activation='softmax'))
     model.summary()
```

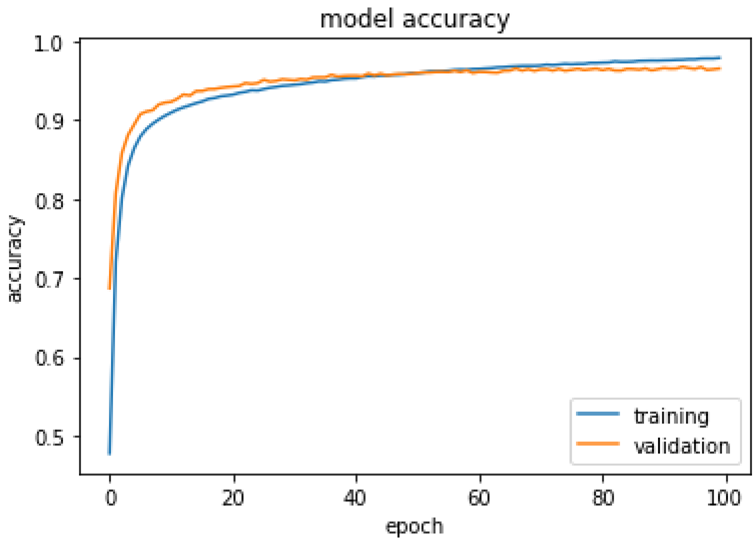
```python
[ ]  model.compile(optimizer="sgd",loss = 'categorical_crossentropy',metrics = ['accuracy'])
     history = model.fit(x_train,y_train,batch_size=128,epochs = 100,verbose=True,validation_split = .1)
```

Input : Constructing a feed-forward architecture using Keras.

<u>OUTPUT</u>:

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 100)               78500
_________________________________________________________________
dense_1 (Dense)              (None, 100)               10100
_________________________________________________________________
dense_2 (Dense)              (None, 10)                1010
=================================================================
Total params: 89,610
Trainable params: 89,610
Non-trainable params: 0
_________________________________________________________________
```

Output I : Model Architecture details



Output II: The Validation and Training Accuracy per epoch achieved by the architecture.

<u>CONCLUSION</u>:

In this experiment, we have created an Artificial Neural Network architecture with **89,610** trainable parameters, able to achieve a validation accuracy of **95.71%**.

Experiment – IV

Performance of Shallow Neural Networks

AIM:

Write a program to implement and verify the performance of shallow Neural Networks with different number of neurons.

THEORY:

Generally, it is believed that as the number of neurons per layer increase the performance of the model increases. However, the inference may depend on the dataset and other training conditions.

Here, we again make use of the MNIST dataset (Refer '*Experiment III*' for more details).

The architectural and training details are same as '*Experiment III*' with **1 Hidden layer** and the number of neurons varying.

CODE:

```
import matplotlib.pyplot as py
import seaborn as sns
import numpy as np
from sklearn.metrics import confusion_matrix

import keras
from keras.datasets import mnist
from keras.layers import Dense
from keras.models import Sequential
from matplotlib import pyplot as plt
from random import randint
```

```
[ ]  # Preparing the dataset
     # Setup train and test splits

     (x_train, y_train),(x_test,y_test) = mnist.load_data()
```

```
[ ]  image_size = 784 # since its 28 X 28
     x_train = x_train.reshape(x_train.shape[0],image_size)
     x_test = x_test.reshape(x_test.shape[0],image_size)
```

```
[ ]  num_classes = 10
     y_train = keras.utils.to_categorical(y_train,num_classes)
     y_test = keras.utils.to_categorical(y_test,num_classes)
```

Input I : Pre-processing of the MNIST Dataset.

```
model_25 = Sequential() # feed forward - backward propagate
# eager execution enabled - session online

model_25.add(Dense(units = 25,activation = 'sigmoid',input_shape=(image_size,)))
model_25.add(Dense(units=num_classes,activation='softmax'))
```

```
model_25.compile(optimizer="sgd",loss = 'categorical_crossentropy',metrics = ['accuracy'])
history = model_25.fit(x_train,y_train,batch_size=128,epochs = 100,verbose=True,validation_split = .1)
```

Input II : Training Model with 25 Neurons

```
model_50 = Sequential() # feed forward - backward propagate
# eager execution enabled - session online

model_50.add(Dense(units = 50,activation = 'sigmoid',input_shape=(image_size,)))
model_50.add(Dense(units=num_classes,activation='softmax'))
```

```
model_50.compile(optimizer="sgd",loss = 'categorical_crossentropy',metrics = ['accuracy'])
history = model_50.fit(x_train,y_train,batch_size=128,epochs = 100,verbose=True,validation_split = .1)
```

Input III : Training Model with 50 Neurons

```
model_100 = Sequential() # feed forward - backward propagate
# eager execution enabled - session online

model_100.add(Dense(units = 100,activation = 'sigmoid',input_shape=(image_size,)))
model_100.add(Dense(units=num_classes,activation='softmax'))
```

```
model_100.compile(optimizer="sgd",loss = 'categorical_crossentropy',metrics = ['accuracy'])
history = model_100.fit(x_train,y_train,batch_size=128,epochs = 100,verbose=True,validation_split = .1)
```

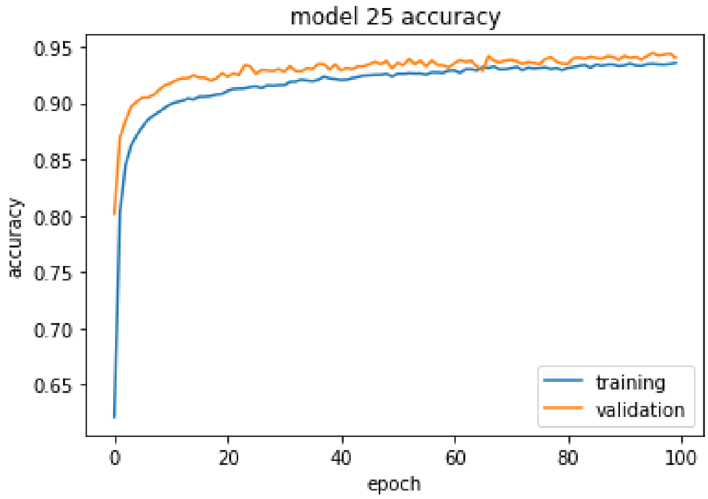
Inuput III : Training Model with 100 Neurons

```
model_200 = Sequential() # feed forward - backward propagate
# eager execution enabled - session online

model_200.add(Dense(units = 200,activation = 'sigmoid',input_shape=(image_size,)))
model_200.add(Dense(units=num_classes,activation='softmax'))
```

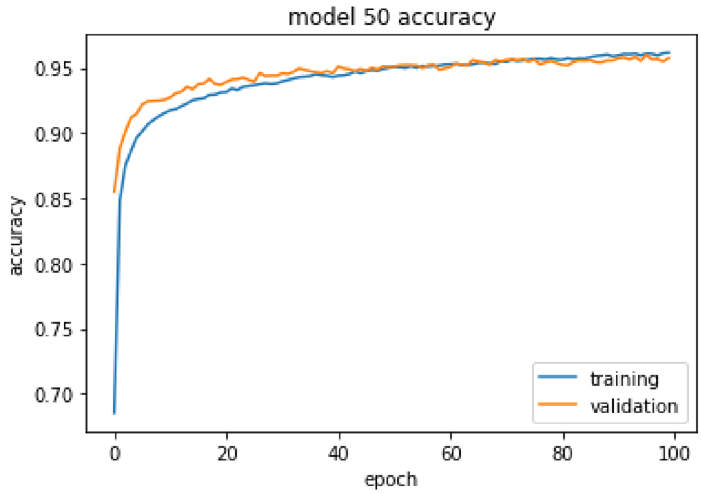
```
model_200.compile(optimizer="sgd",loss = 'categorical_crossentropy',metrics = ['accuracy'])
history = model_200.fit(x_train,y_train,batch_size=128,epochs = 100,verbose=True,validation_split = .1)
```

Input IV : Training Model with 200 neurons

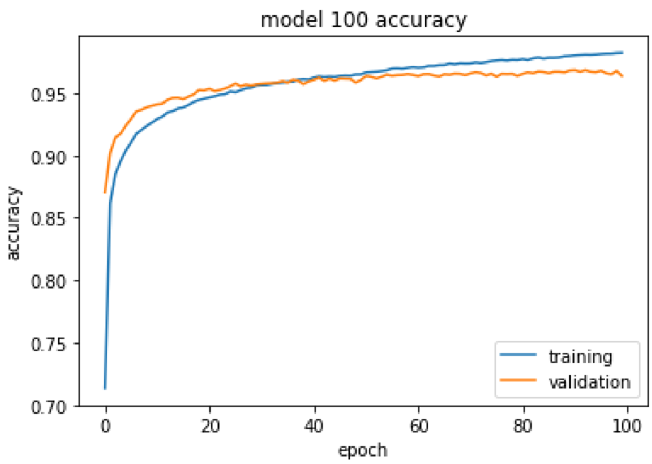
OUTPUT:



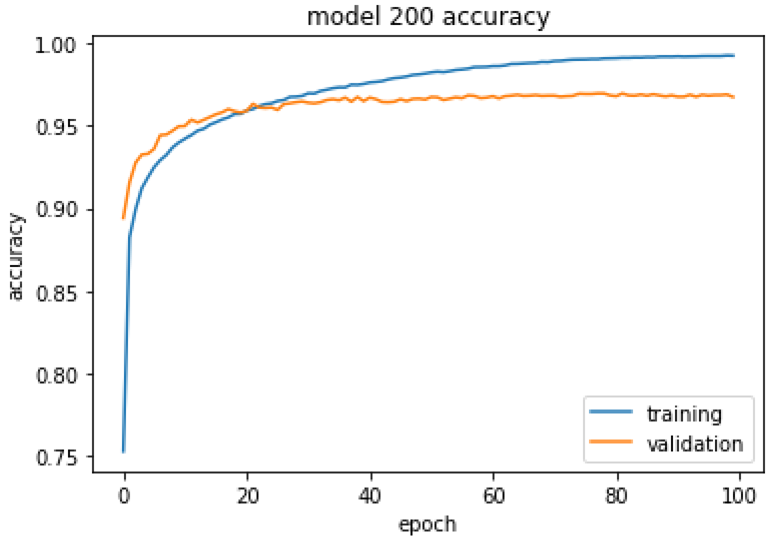
Output II : Accuracy Vs. Epoch curve over Training and Validation set for 25 Neuron NN Arch.



Output III : Accuracy Vs. Epoch curve over Training and Validation set for 50 Neuron NN Arch.



Output IV : Accuracy Vs. Epoch curve over Training and Validation set for 100 Neuron NN Arch.



Output V : Accuracy Vs. Epoch curve over Training and Validation set for 200 Neuron NN Arch.

CONCLUSION:

S.No	No. of Neurons	No. of Parameters	Test-Set Accuracy
I.	25	19,885	92.87%
II.	50	39,760	94.64%
III.	100	79,510	95.85%
IV.	200	1,59,010	96.35%

Table : Depicting the Test-Set Accuracy obtained on various experiments.

Through the experiment, we conclude that increasing the number of neurons can result in increase in the accuracy obtained by the model as more features are extracted for classification purposes.

Experiment – V

Deep Neural Network

AIM:

Write a program to implement a Deep Neural Network and verify the performance on MNIST dataset.

THEORY:

Generally, it is believed that as the number of hidden layers increases the performance of the model increases. However, the inference may depend on the dataset and other training conditions.

Here, we again make use of the MNIST dataset (Refer '*Experiment III*' for more details).

Details about the architecture:

8. Programming Framework: Keras – Python
9. Type of Architecture: Feed-Forward Artificial Neural Network
10. Activation function used: Sigmoid for Experiment I and ReLu for Experiment II
11. Input Layer: 784 inputs
12. Output Layer: 10 Output neurons (For digits 0 to 9)
13. No. of Hidden Layers: 4
14. No. of Units per layer: 128

Training details are the same as '*Experiment III*'.

CODE:

The MNIST dataset is first pre-processed (Refer to '*Input I*' of '*Experiment IV*').

```
model = Sequential()

model.add(Dense(units = 128,activation = 'sigmoid',input_shape=(image_size,)))
model.add(Dense(units = 128,activation = 'sigmoid'))
model.add(Dense(units = 128,activation = 'sigmoid'))
model.add(Dense(units = 128,activation = 'sigmoid'))
model.add(Dense(units=num_classes,activation='sigmoid'))
```

```
model.compile(optimizer="sgd",loss = 'categorical_crossentropy',metrics = ['accuracy'])
history = model.fit(x_train,y_train,batch_size=128,epochs = 100,verbose=True,validation_split = .1)
```

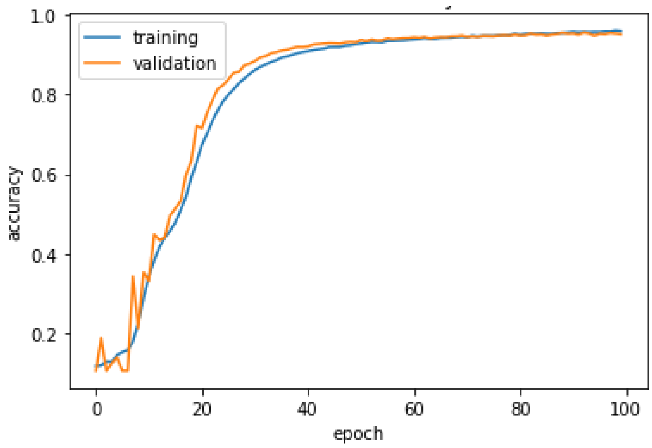
Input I : Deep Neural Network with Activation function as 'Sigmoid' and optimizer as 'SGD'.

```
model_2 = Sequential()

model_2.add(Dense(units = 128,activation = 'relu',input_shape=(image_size,)))
model_2.add(Dense(units = 128,activation = 'relu'))
model_2.add(Dense(units = 128,activation = 'relu'))
model_2.add(Dense(units = 128,activation = 'relu'))
model_2.add(Dense(units=num_classes,activation='softmax'))
```

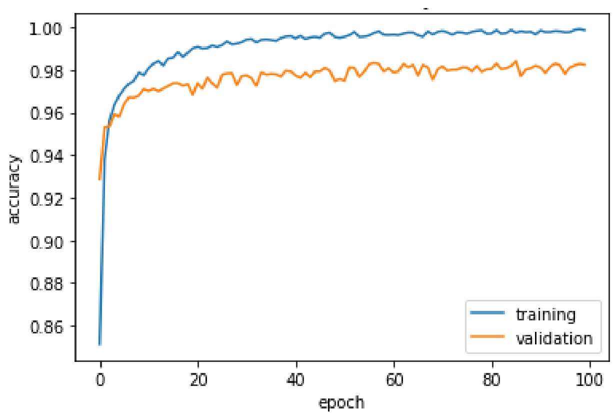
```
model_2.compile(optimizer="adam",loss = 'categorical_crossentropy',metrics = ['accuracy'])
history = model_2.fit(x_train,y_train,batch_size=128,epochs = 100,verbose=True,validation_split = .1)
```

Input II : Deep Neural Network with Activation function as 'ReLu' and optimizer as 'Adam'.

<u>OUTPUT</u>:



Output I : Accuracy Vs. Epoch curve over Training and Validation set for 'Input I' Arch.



Output II : Accuracy Vs. Epoch curve over Training and Validation set for 'Input II' Arch.

CONCLUSION:

S. No.	**Activation Function**	**No. of Parameter**	**Test-Set Accuracy**
I.	Sigmoid	1,51,306	95.60%
II.	ReLu	151,306	98.30%

From the results obtained in '*Experiment IV*' (Page No. 21) we can conclude that as we increase the number of Hidden layers in the model architecture, the accuracy obtained by the model may increase.

Also, the choice of Activation function and the Optimizer may also hamper the accuracy obtained by the model significantly.

Experiment – VI

Convolutional Neural Network

AIM:

Build a Convolutional Neural Network and evaluate its performance on MNIST dataset.

THEORY:

A Convolutional Neural Network is a Deep Learning algorithm, which takes a matrix as an input, assign importance (Learnable weights and biases) to various aspects of the matrix and be able to differentiate one from another.

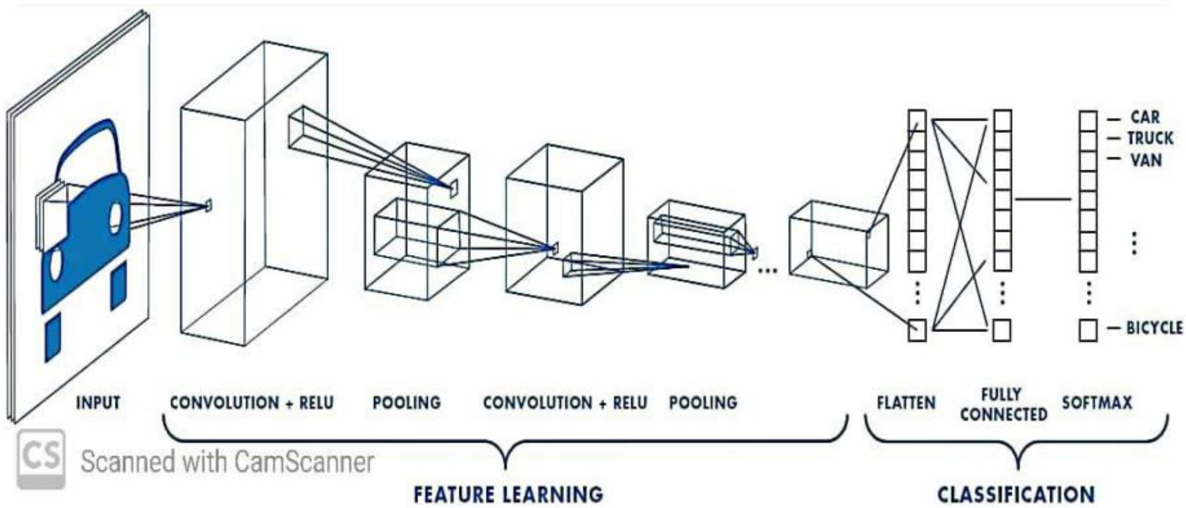


Image : Fundamental CNN Architecture

It has a combination of namely 4 layers: Convolutional, Pooling, Flattening and Fully Connected.

CODE:

```
from numpy import mean,std
from matplotlib import pyplot
from sklearn.model_selection import KFold
from keras.datasets import mnist
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Dense
from keras.layers import Flatten
from keras.optimizers import SGD
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.metrics import confusion_matrix
```

```
[ ] def load_dataset() :

      (trainX, trainY), (testX, testY) = mnist.load_data()
      trainX = trainX.reshape((trainX.shape[0],28,28,1))
      testX = testX.reshape((testX.shape[0],28,28,1))

      trainY = to_categorical(trainY)
      testY = to_categorical(testY)

      return trainX, trainY, testX, testY
```

```
def prep_pixels(train,test):
  train_norm = train.astype('float32')
  test_norm = test.astype('float32')

  train_norm /= 255
  test_norm /= 255

  return train_norm, test_norm
```

```
[ ] trainX, trainY, testX, testY = load_dataset()

[ ] trainX, testX = prep_pixels(trainX, testX)
```

```
[ ] 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'))

      opt = SGD(lr=0.01, momentum = 0.9)
      model.compile(optimizer=opt, loss = 'categorical_crossentropy', metrics = ['accuracy'])
      return model
```

```
model = define_model()
```

```
history = model.fit(trainX, trainY,batch_size=128,epochs=25,validation_data = (testX,testY),verbose=2)
```

```
y_pred = model.predict(testX)

Y_pred = np.argmax(y_pred,1)
Y_test = np.argmax(testY,1)

mat = confusion_matrix(Y_test, Y_pred)
```

Input : Defining the Convolutional Neural Network Architecture

OUTPUT:

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_4 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 13, 13, 32)        0
_________________________________________________________________
flatten_3 (Flatten)          (None, 5408)              0
_________________________________________________________________
dense_6 (Dense)              (None, 100)               540900
_________________________________________________________________
dense_7 (Dense)              (None, 10)                1010
=================================================================
Total params: 542,230
Trainable params: 542,230
Non-trainable params: 0
_________________________________________________________________
```

Output: The Summary of the Model Architecture

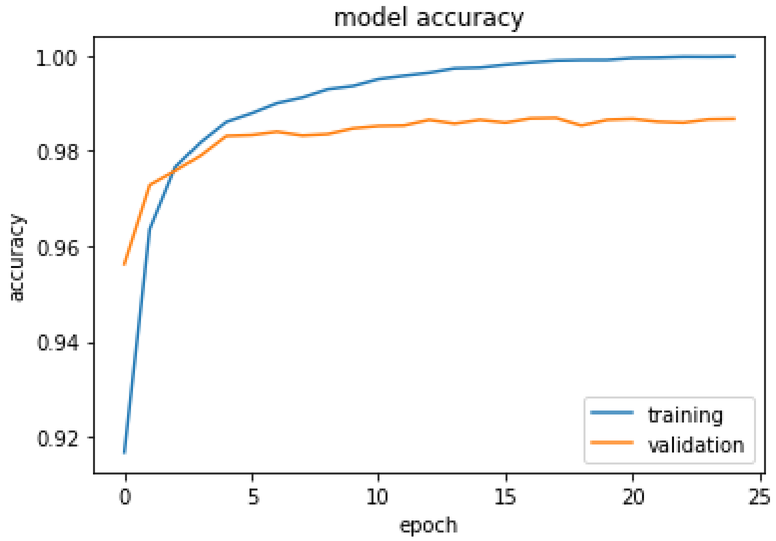


Image I : Graph depicting Accuracy per epoch obtained on the Training and Validation set.

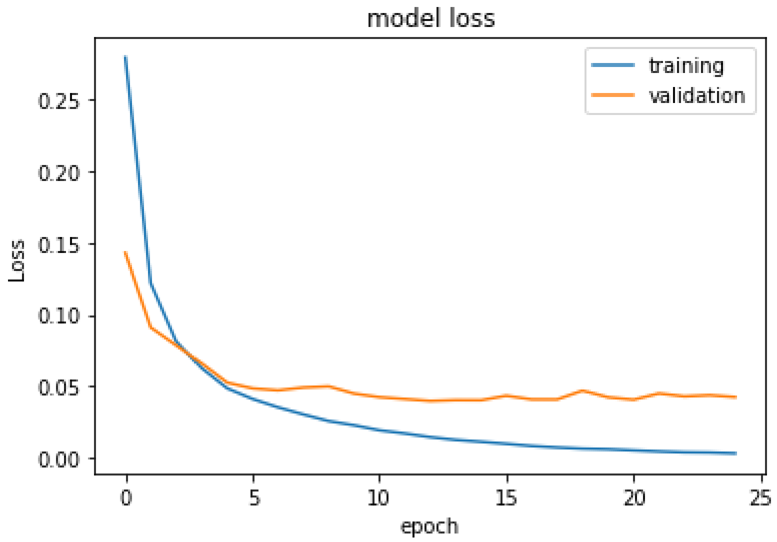


Image II : Graph depicting Loss per epoch obtained on the Training and Validation set.

```
[[ 972    0    1    0    0    1    4    1    1    0]
 [    0 1128    1    1    0    2    1    1    1    0]
 [    1    1 1018    1    2    0    1    4    4    0]
 [    0    0    3  996    0    6    0    1    3    1]
 [    0    1    1    0  974    0    1    0    0    5]
 [    1    0    0    5    0  880    4    0    1    1]
 [    5    3    0    0    3    1  946    0    0    0]
 [    0    1    9    0    1    0    0 1015    0    2]
 [    5    0    1    2    1    3    0    2  956    4]
 [    1    0    0    4   16    3    0    2    1  982]]
```

Image III: Confusion matrix obtained on the MNIST Dataset.

<u>CONCLUSION</u>:

Through this experiment, we can conclude that the Convolutional neural network is able to achieve an accuracy of **98.67%** which is better than that achieved through Artificial Neural Network (Refer '*Experiment 6*').

Experiment – VII

Performance of Different Optimizers

AIM:

Write a program to evaluate the performance of different optimizers on MNIST dataset.

THEORY:

Optimizers during the training process, change the parameters (weights) of our models to try and minimize the loss function, and make our predictions as correct and optimized as possible. They tie together loss function and model parameters by updating the model in response to the output of the loss function.

Gradient descent is one of the most popular algorithms to perform optimization. It optimizes the parameters in the opposite direction of the gradient of the objective function.

Stochastic Gradient Descent (SGD) : Performs a parameter update for each training example instead of the whole training set.

Adagrad: Adapts learning rate to the parameters performing smaller updates for parameters associated with frequently occurring features, and larger updates for parameters associated with infrequent features.

RMSprop : An adaptive learning rate method which divides the learning rate by an exponentially decaying average of squared gradients.

Adam : Adaptive Moment estimation is a method that computes adaptive learning rates for each parameter. In addition to storing an exponentially decaying average of past square gradient, it also keeps an exponentially decaying average of past gradient similar to momentum.

CODE:

```
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.metrics import confusion_matrix
```

```
[ ] import keras
    from keras.datasets import mnist
    from keras.layers import Dense
    from keras.models import Sequential
    from matplotlib import pyplot as plt
    from random import randint
```

```
[ ] (x_train, y_train), (x_test, y_test) = mnist.load_data()
```

```
image_size = 784
x_train = x_train.reshape(x_train.shape[0], image_size)
x_test = x_test.reshape(x_test.shape[0],image_size)
```

```
num_classes = 10
y_train = keras.utils.to_categorical(y_train,num_classes)
y_test = keras.utils.to_categorical(y_test,num_classes)
```

```
model = Sequential()

model.add(Dense( units= 2048, activation='sigmoid', input_shape=(image_size,)))
model.add(Dense(units=num_classes,activation='softmax'))
```

Input I : Pre-processing the MNIST dataset and defining the Model Architecture. Note 'model', 'model_2', 'model_3' & 'model_4' are all same architectures.

```
model.compile(optimizer = 'sgd', loss = 'categorical_crossentropy', metrics = ['accuracy'])
history = model.fit(x_train, y_train, batch_size=128, epochs = 50, verbose=2, validation_split=.1)
```

Input II : Training the model with Optimizer as 'Stochastic Gradient Descent'

```
model_2.compile(optimizer = 'RMSprop', loss = 'categorical_crossentropy', metrics = ['accuracy'])
history = model_2.fit(x_train, y_train, batch_size=128, epochs = 50, verbose=2, validation_split=.1)
```

Input III: Training the model with Optimizer as 'RMS prop'

```
model_3.compile(optimizer = 'Adam', loss = 'categorical_crossentropy', metrics = ['accuracy'])
history = model_3.fit(x_train, y_train, batch_size=128, epochs = 50, verbose=2, validation_split=.1)
```

Input IV: Training the model with Optimizer as 'Adaptive moment estimation propagation'

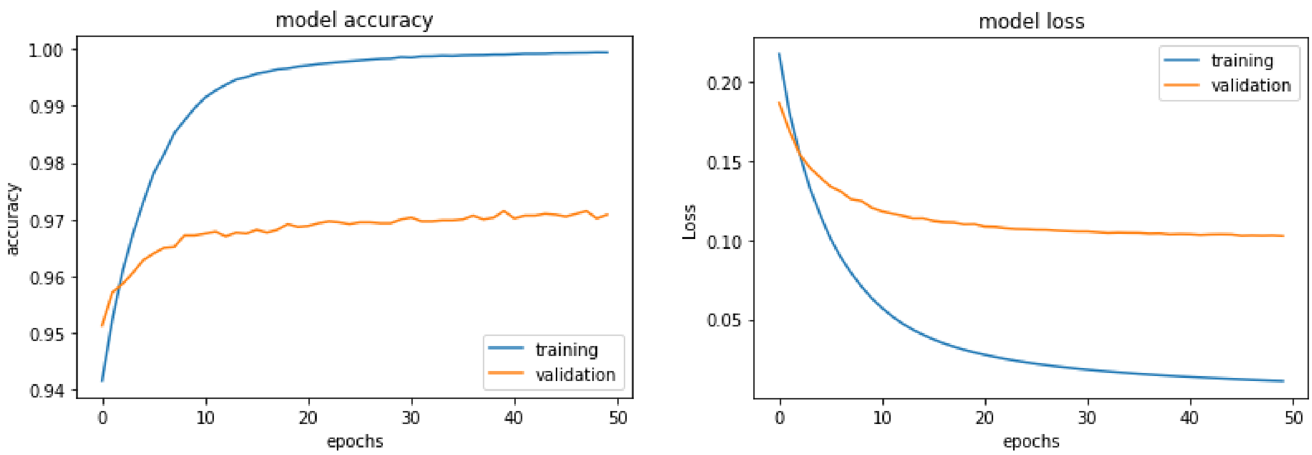
```
model_4.compile(optimizer = 'Adagrad', loss = 'categorical_crossentropy', metrics = ['accuracy'])
history = model_4.fit(x_train, y_train, batch_size=128, epochs = 50, verbose=2, validation_split=.1)
```

Input V: Training the model with Optimizer as 'Adagrad'.

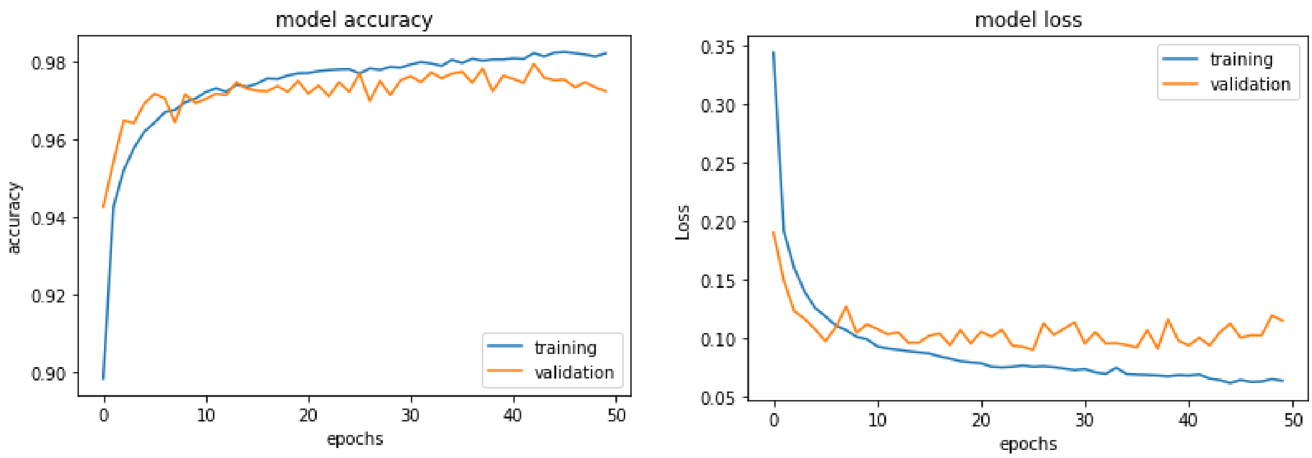
OUTPUT:

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 2048)              1607680
_________________________________________________________________
dense_1 (Dense)              (None, 10)                20490
=================================================================
Total params: 1,628,170
Trainable params: 1,628,170
Non-trainable params: 0
_________________________________________________________________
```

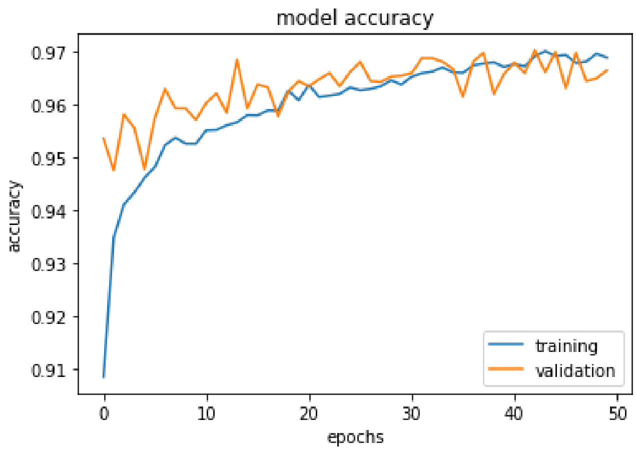
Output I : The Model Architecture

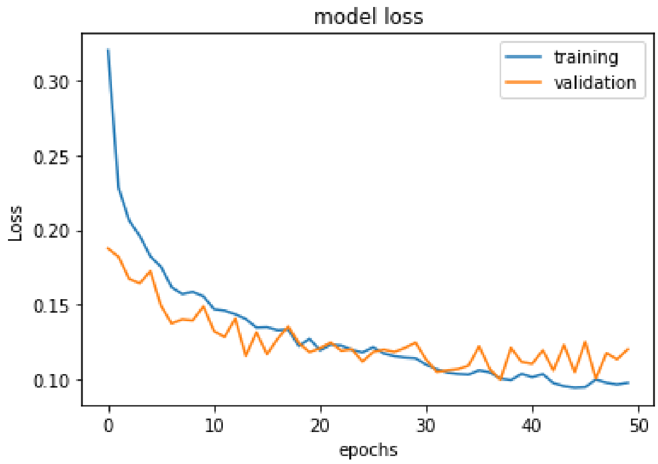


Output II : Accuracy vs epoch curve (Left-hand side) and Loss vs epoch curve (Right-hand side) for training and validation obtained by 'SGD'

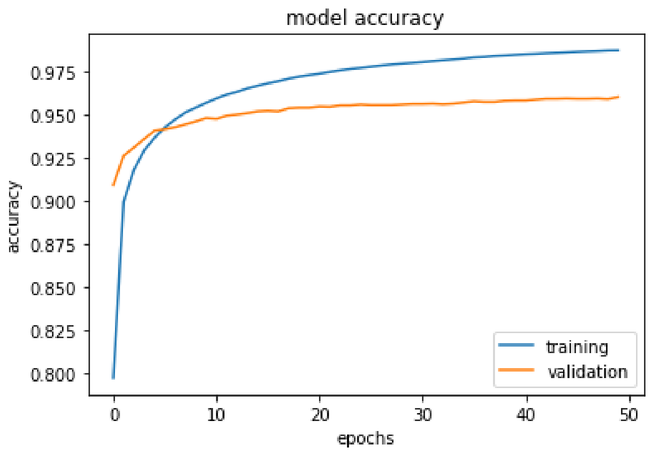


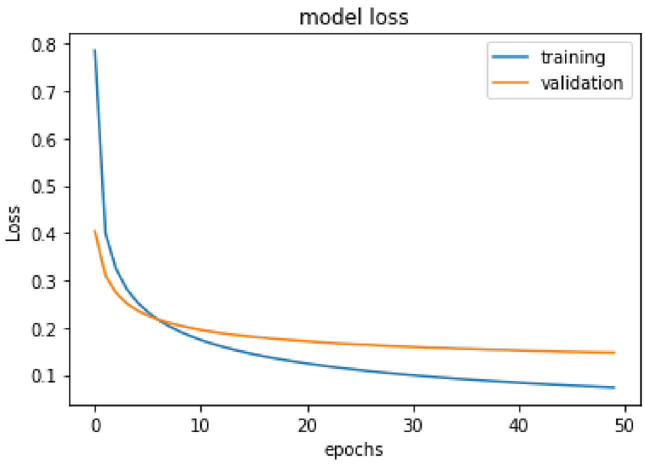
Output III : Accuracy vs epoch curve (Left-hand side) and Loss vs epoch curve (Right-hand side) for training and validation obtained by 'RMSprop'





Output IV : Accuracy vs epoch curve (Left-hand side) and Loss vs epoch curve (Right-hand side) for training and validation obtained by 'Adam'





Output V : Accuracy vs epoch curve (Left-hand side) and Loss vs epoch curve (Right-hand side) for training and validation obtained by 'Adagrad'

CONCLUSION:

S. No.	Optimizer	Test-Set Accuracy
I.	SGD	97.15%
II.	**RMSprop**	**97.93%**
III.	Adam	97.03%
IV.	Adagrad	96.03%

Through this experiment, we have made use of various optimizers. For the MNIST dataset however, for the architecture implemented – We conclude that RMS prop achieves the highest accuracy on test-set.

Experiment – VIII

Fine Tuning CNN Architecture

AIM:

To fine tune a pre-trained CNN architecture and evaluate its performance on a dataset.

THEORY:

Fine-Tuning is a technique, where we train the whole model on a custom data-set, of which the classes are a subset of a pre-trained model. During the training process we freeze the initial layers of the pre-trained Model, and retrain the model on our custom dataset.

In this experiment, we take the VGG16 Architecture pre-trained on 1000 classes from the Image-Net dataset.

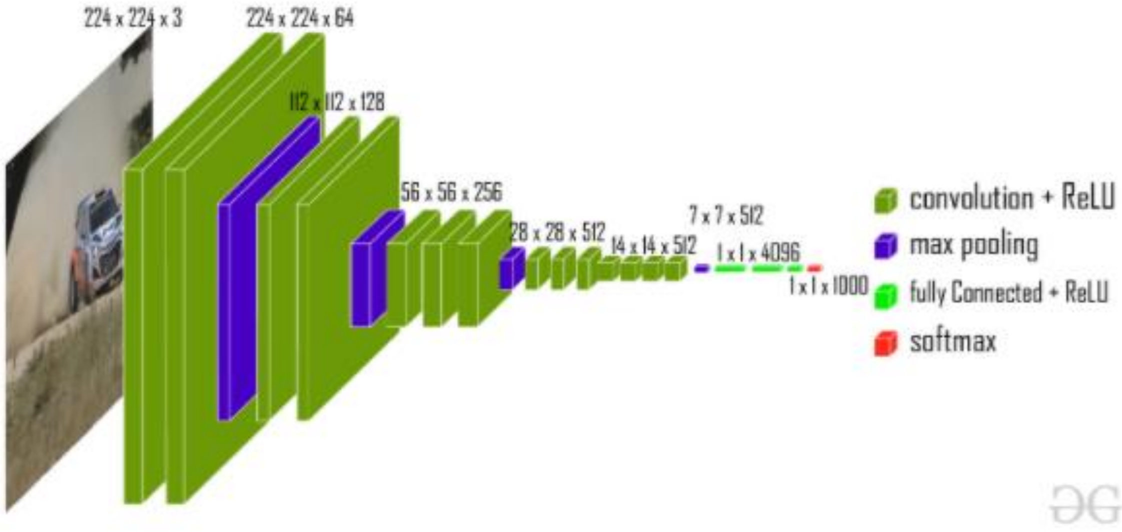


Image : The VGG16 Arcitecture.

CODE:

```
import pandas as pd
from keras.models import Model
import numpy as np
import matplotlib.pyplot as plt
from glob import glob
from keras.layers import Flatten, Dense
from keras.applications import VGG16
from keras.preprocessing.image import img_to_array, ImageDataGenerator
```

```
!unzip "/content/drive/MyDrive/classroom/Datasets/images.zip" -d "/content/drive/MyDrive/classroom/Datasets"
```

```
traindf = pd.read_csv("/content/drive/MyDrive/classroom/Datasets/emergency_train.csv", dtype=str)
```

```
[ ] train_datagen = ImageDataGenerator(rescale=1./255, shear_range = 0.2, zoom_range = 0.2, horizontal_flip = True)

[ ] test_datagen = ImageDataGenerator(rescale = 1./255., validation_split=0.10)

[ ] valid_datagen = ImageDataGenerator(rescale = 1./255., validation_split=0.15)

[ ] train_generator = train_datagen.flow_from_dataframe(
        dataframe = traindf,
        directory = "/content/drive/MyDrive/classroom/Datasets/images",
        x_col = 'image_names',
        y_col = 'emergency_or_not',
        subset = 'training',
        batch_size = 16,
        seed = 42,
        shuffle = True,
        class_mode = "binary",
        target_size = (224,224))
```

```
[ ] test_generator = test_datagen.flow_from_dataframe(
        dataframe = traindf,
        directory = "/content/drive/MyDrive/classroom/Datasets/images",
        x_col = 'image_names',
        y_col = 'emergency_or_not',
        subset = 'validation',
        batch_size = 16,
        seed = 42,
        shuffle = True,
        class_mode = "binary",
        target_size = (224,224)
    )
```

```
[ ] valid_generator = valid_datagen.flow_from_dataframe(
        dataframe = traindf,
        directory = "/content/drive/MyDrive/classroom/Datasets/images",
        x_col = 'image_names',
        y_col = 'emergency_or_not',
        subset = 'validation',
        batch_size = 16,
        seed = 42,
        shuffle = True,
        class_mode = "binary",
        target_size = (224,224)
    )
```

```
vgg = VGG16(include_top = False, weights = 'imagenet', input_shape = (224,224,3))

for layer in vgg.layers :
  layer.trainable=False
```

```
[ ]  x = Flatten()(vgg.output)
     x = Dense(1,activation='sigmoid')(x)

     model = Model(inputs = vgg.input, outputs = x)
     model.compile(loss='binary_crossentropy', optimizer = 'adam', metrics = ['accuracy'])
```

```
[ ]  train_generator.class_indices

     {'0': 0, '1': 1}
```

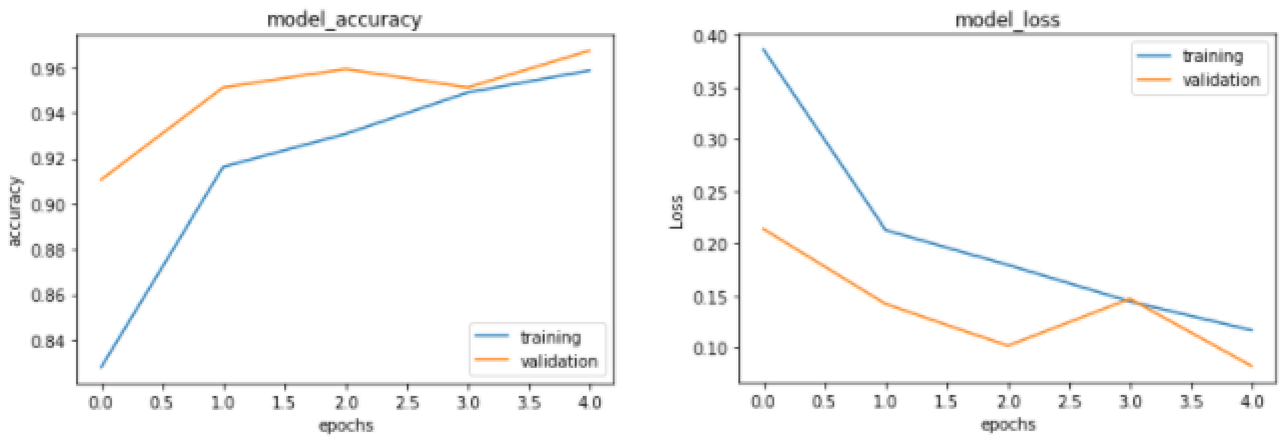
```
[ ]  history = model.fit_generator(train_generator, epochs = 5, validation_data=valid_generator)
```

Input : Fine-Tuning VGG16 Model for our custom Dataset

OUTPUT:

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 224, 224, 3)]     0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
_________________________________________________________________
flatten_2 (Flatten)          (None, 25088)             0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 25089
=================================================================
Total params: 14,739,777
Trainable params: 25,089
Non-trainable params: 14,714,688
_________________________________________________________________
```

Output I : Fine-Tuning Model Summary



Output II : Accuracy vs. Epoch Curve (Right-Hand side) and Loss vs Epoch Curve (Left-Hand Side) On Training and Validation Set.

CONCLUSION:

Through this experiment, we see that Pre-Trained models can be used to transfer domain, by freezing the initial layers and training on the fully connected layer.

We train our models, with overall **25,089** Trainable parameters (**0.17%** of the total model parameters) and achieve an accuracy of **96.34%** on our test-set.

Experiment – IX

Recurrent Neural Networks

AIM:

To apply Recurrent Neural Models for text classification.

THEORY:

Recurrent Neural Networks are a type of artificial neural network model designed to recognize patterns in sequences of data. LSTM or Long Short-Term Memory networks are RNN which are capable of learning long-term dependencies, i.e it uses recent and past data to perform operations.

CODE:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from keras.models import Model
from keras.layers import LSTM, Activation, Dense, Dropout, Input, Embedding
from keras.optimizers import RMSprop
from keras.preprocessing.text import Tokenizer
from keras.preprocessing import sequence
from keras.utils import to_categorical
from keras.callbacks import EarlyStopping

%matplotlib inline
```

```
[ ] %cd /content/drive/MyDrive/Colab Notebooks/Classroom/mnist_sequential_model

    df = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/Classroom/spam.csv', delimiter = ',', encoding = 'latin-1')
```

```
[ ] df.drop(['Unnamed: 2', 'Unnamed: 3', 'Unnamed: 4'], axis=1, inplace=True)
```

Input I : Pre-processing the Data-Frame

```
[ ] X = df.v2
    Y = df.v1
    le = LabelEncoder()
    Y = le.fit_transform(Y)
```

```
Y = Y.reshape(-1,1)
```

```
[ ] X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.15)
```

```
max_words = 1000
max_len = 150
tok = Tokenizer(num_words = max_words)
tok.fit_on_texts(X_train)
sequences = tok.texts_to_sequences(X_train)
sequence_matrix = sequence.pad_sequences(sequences, maxlen= max_len)
```

```
[ ] def RNN() :

      inputs = Input(name='inputs', shape = [max_len])
      layer = Embedding(max_words,50,input_length=max_len)(inputs)
      layer = LSTM(64)(layer)
      layer = Dense(256, name='FC1')(layer)
      layer = Activation('relu')(layer)
      layer = Dropout(0.5)(layer)
      layer = Dense(1, name='out_layer')(layer)
      layer = Activation('sigmoid')(layer)
      model = Model(inputs = inputs, outputs = layer)
      return model
```

```
[ ] model = RNN()
    model.summary()
    model.compile(loss='binary_crossentropy', optimizer = RMSprop(), metrics = ['accuracy'])
```

```
[ ] model.fit(sequence_matrix, Y_train, batch_size = 128, epochs = 10, validation_split = 0.2, callbacks = [EarlyStopping(monitor = 'val_loss')])
```

```
[ ] test_sequences = tok.texts_to_sequences(X_test)
    test_sequences_matrix = sequence.pad_sequences(test_sequences, maxlen = max_len)
```

```
[ ] accr = model.evaluate(test_sequences_matrix, Y_test)
```

```
[ ] print('Test set\n Loss: {:0.3f}\n Accuracy: {:0.3f}'.format(accr[0],accr[1]))
```

Input II : Defining the LSTM Architecture and testing the Model after Training.

OUTPUT:

	v1	v2
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...

Output I : Processed Data-Frame (Top 5 entries only)

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
inputs (InputLayer)          [(None, 150)]             0
_________________________________________________________________
embedding (Embedding)        (None, 150, 50)           50000
_________________________________________________________________
lstm (LSTM)                  (None, 64)                29440
_________________________________________________________________
FC1 (Dense)                  (None, 256)               16640
_________________________________________________________________
activation (Activation)      (None, 256)               0
_________________________________________________________________
dropout (Dropout)            (None, 256)               0
_________________________________________________________________
out_layer (Dense)            (None, 1)                 257
_________________________________________________________________
activation_1 (Activation)    (None, 1)                 0
=================================================================
Total params: 96,337
Trainable params: 96,337
Non-trainable params: 0
```

Output II : The LSTM Architecture

```
print('Test set\n Loss: {:0.3f}\n Accuracy: {:0.3f}'.format(accr[0],accr[1]))
```

```
Test set
 Loss: 0.059
 Accuracy: 0.982
```

Output III : The accuracy obtained by the trained model on Test-Set

CONCLUSION:

Through this experiment we learn to implement a Long short term memory (LSTM model) which provides us an accuracy **98.2%** for Spam and Non-Spam test classification.

Experiment – X

Application of Natural Language Processing: Sentiment Classification using TFIDF Approach

AIM:

Apply NLP: Perform sentiment classification using TF-IDF approach.

THEORY:

In information retrieval, tf–idf, TF*IDF, or TFIDF, short for term frequency–inverse document frequency, is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus.[1] It is often used as a weighting factor in searches of information retrieval, text mining, and user modelling. The tf–idf value increases proportionally to the number of times a word appears in the document and is offset by the number of documents in the corpus that contain the word, which helps to adjust for the fact that some words appear more frequently in general.

CODE:

```
[ ]  import numpy as np, re, nltk, pickle
     from sklearn.datasets import load_files
     nltk.download('stopwords')
     from nltk.corpus import stopwords

     [nltk_data] Downloading package stopwords to /root/nltk_data...
     [nltk_data]   Package stopwords is already up-to-date!

[ ]  movie_data = load_files("/content/drive/MyDrive/Colab Notebooks/Classroom/positive and negative tweets")

     X, y = movie_data.data, movie_data.target
```

```
[ ] documents = []
    from nltk.stem import WordNetLemmatizer
    stemmer = WordNetLemmatizer()

[ ] import nltk
    nltk.download('wordnet')

    [nltk_data] Downloading package wordnet to /root/nltk_data...
    [nltk_data]   Package wordnet is already up-to-date!
    True

[ ] for sen in range(0, len(X)) :

      document = re.sub(r'\W', ' ', str(X[sen]))

      document = re.sub(r'\s+[a-zA-Z]\s+', ' ', document)

      document = re.sub(r'\^[a-zA-Z]\s+', ' ', document)

      document = re.sub(r'\s+', ' ', document)

      document = re.sub(r'b\s+', '', document)

      document = document.lower()

      # Lemmatization
      document = document.split()

      document = [stemmer.lemmatize(word) for word in document]
      document = ' '.join(document)

      documents.append(document)
```

Input I : Loading dataset & Text Pre-Processing.

```
from sklearn.feature_extraction.text import CountVectorizer

vectorizer = CountVectorizer(max_features = 1500, min_df = 5, max_df = 0.7, stop_words = stopwords.words('english'))
X = vectorizer.fit_transform(documents).toarray()
```

Input II: Converting text to word vectors.

```
[ ] from sklearn.feature_extraction.text import TfidfTransformer

    tfidfconverter = TfidfTransformer()

    X = tfidfconverter.fit_transform(X).toarray()
```

```
[ ] from sklearn.feature_extraction.text import TfidfVectorizer

    tfidfconverter = TfidfVectorizer(max_features=1500, min_df=5, max_df=0.7, stop_words = stopwords.words('english'))
    X = tfidfconverter.fit_transform(documents).toarray()
```

Input III: Converting word vectors to TF-IDF vectors

```
[ ] from sklearn.model_selection import train_test_split
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state=0)
```

```
from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier(n_estimators = 1000, random_state=0)
classifier.fit(X_train, y_train) # Using Random Forest Classifier
```

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=1000,
                       n_jobs=None, oob_score=False, random_state=0, verbose=0,
                       warm_start=False)
```

```
[ ] y_pred = classifier.predict(X_test)
```

Input IV : Training the Random-Forest tree classifier

OUTPUT:

```
[ ] from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

    print("Confusion Matrix:", confusion_matrix(y_test,y_pred))
    print("Classification Report:", classification_report(y_test,y_pred))
    print("Accuracy:", accuracy_score(y_test, y_pred))
```

```
Confusion Matrix: [[181  24]
 [ 50 147]]
Classification Report:               precision    recall  f1-score   support

           0       0.78      0.88      0.83       205
           1       0.86      0.75      0.80       197

    accuracy                           0.82       402
   macro avg       0.82      0.81      0.81       402
weighted avg       0.82      0.82      0.81       402

Accuracy: 0.8159203980099502
```

Output: Evaluating the Random Forest Tree classifier on Test-Set.

CONCLUSION:

Through this approach, we learn how to build a Random forest Tree text classifier and to create word vector using the TF-IDF approach.

The accuracy obtained by our model on the test-set is **81.59%.**