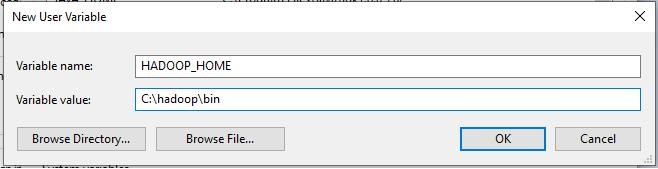
# Ex.No 1: Procedure to set up single node cluster using hadoop

**Step 1: Verify the Java installed** javac -version

**Step 2: Extract Hadoop at C:\Hadoop**

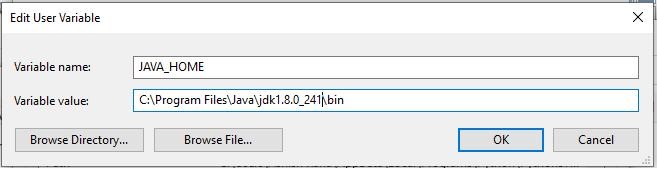
**Step 3: Setting up the HADOOP\_HOME variable**

Use windows environment variable setting for Hadoop Path setting.

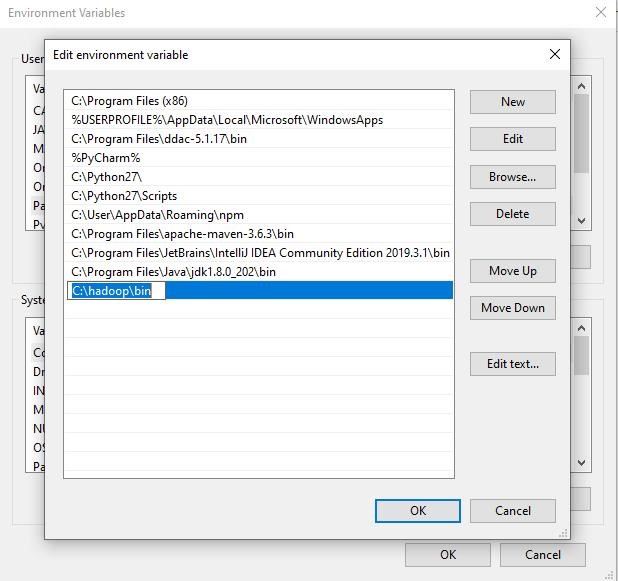


**Step 4: Set JAVA\_HOME variable**

Use windows environment variable setting for Hadoop Path setting.



**Step 5: Set Hadoop and Java bin directory path**



**Step 6: Hadoop Configuration :**

For Hadoop Configuration we need to modify Six files that are listed below-

1. Core-site.xml
2. Mapred-site.xml
3. Hdfs-site.xml
4. Yarn-site.xml
5. Hadoop-env.cmd
6. Create two folders datanode and namenode

**Step 6.1: Core-site.xml configuration**

<configuration>

<property>

<name>fs.defaultFS</name>

<value>hdfs://localhost:9000</value>

</property>

</configuration>

**Step 6.2: Mapred-site.xml configuration**

<configuration>

<property>

<name>mapreduce.framework.name</name>

<value>yarn</value>

</property>

</configuration>

**Step 6.3: Hdfs-site.xml configuration**

<configuration>

<property>

<name>dfs.replication</name>

<value>1</value>

</property>

<property>

<name>dfs.namenode.name.dir</name>

<value>C:\hadoop-2.8.0\data\namenode</value>

</property>

<property>

<name>dfs.datanode.data.dir</name>

<value>C:\hadoop-2.8.0\data\datanode</value>

</property>

</configuration>

**Step 6.4: Yarn-site.xml configuration**

<configuration>

<property>

<name>yarn.nodemanager.aux-services</name>

<value>mapreduce\_shuffle</value>

</property> <property>

<name>yarn.nodemanager.auxservices.mapreduce.shuffle.class</name>

<value>org.apache.hadoop.mapred.ShuffleHandler</value>

</property>

</configuration>

**Step 6.5: Hadoop-env.cmd configuration**

Set "JAVA\_HOME=C:\Java" (On C:\java this is path to file jdk.18.0)



**Step 6.6: Create datanode and namenode folders**

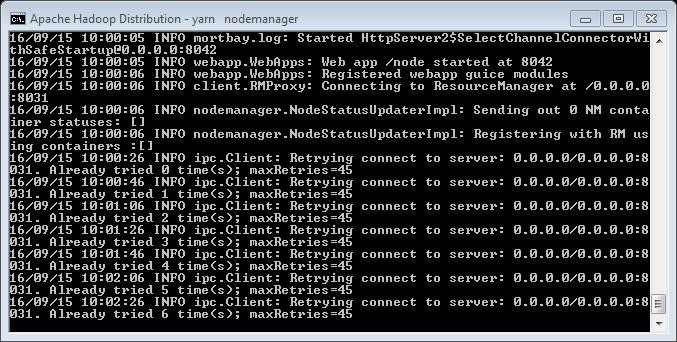
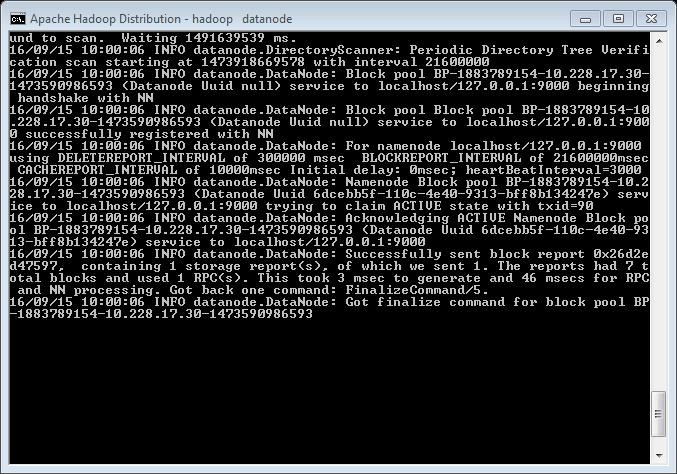
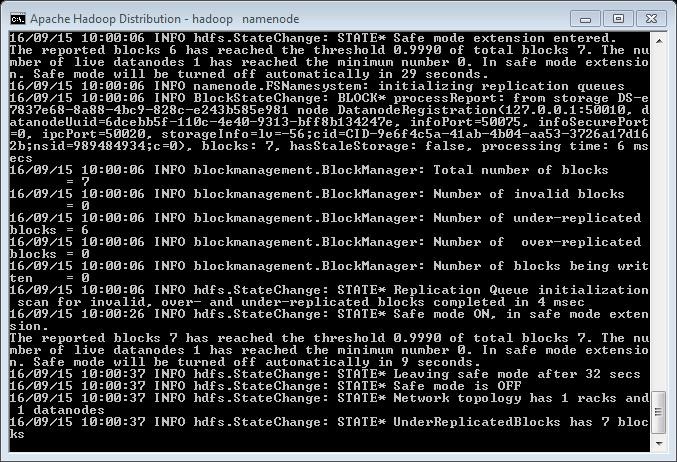
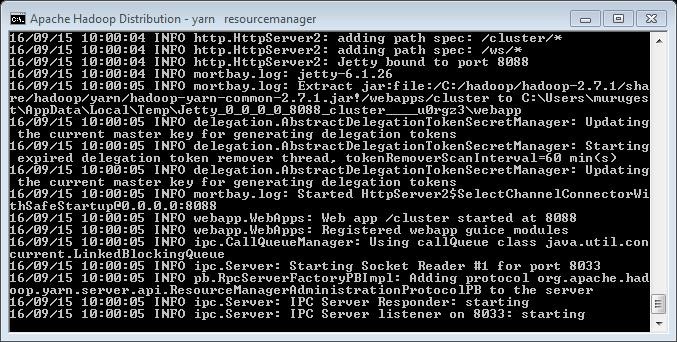
1. Create folder "data" under "C:\Hadoop-2.8.0"
2. Create folder "datanode" under "C:\Hadoop-2.8.0\data"
3. Create folder "namenode" under "C:\Hadoop-2.8.0\data"

**Step 7: Format the namenode folder**

Open command window (cmd) and typing command “hdfs namenode –format” **Step 8: Testing the setup**

Open command window (cmd) and typing command “start-all.cmd”

C:\hadoop\hadoop-2.7.1\sbin>start-all.cmd



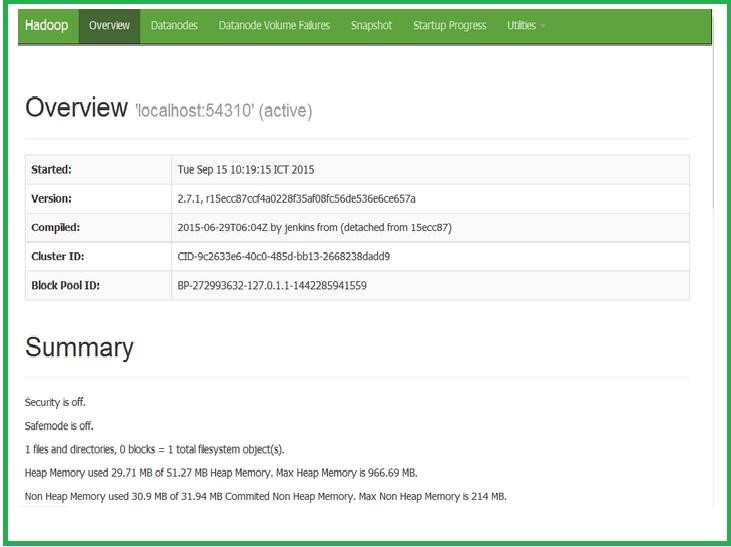
C:\hadoop\hadoop-2.7.1\bin>hadoop fs -mkdir /cluster1

C:\hadoop\hadoop-2.7.1\bin>

Ensure that namenode, datanode, and Resource manager are running **Step 9:** Open: http://localhost:8088



**Step 10:** Open: http://localhost:50070



# Ex.No 2: Word Count Program to use of Map and Reduce Tasks

import java.io.IOException; import java.util.StringTokenizer; import org.apache.hadoop.conf.Configuration; import org.apache.hadoop.fs.Path; import org.apache.hadoop.io.IntWritable; import org.apache.hadoop.io.Text; import org.apache.hadoop.mapreduce.Job; import org.apache.hadoop.mapreduce.Mapper; import org.apache.hadoop.mapreduce.Reducer; import org.apache.hadoop.mapreduce.lib.input.FileInputFormat; import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WordCount {

public static class TokenizerMapper

extends Mapper<Object, Text, Text, IntWritable>{

private final static IntWritable one = new IntWritable(1); private Text word = new Text();

public void map(Object key, Text value, Context context

) throws IOException, InterruptedException { StringTokenizer itr = new StringTokenizer(value.toString()); while (itr.hasMoreTokens()) {

word.set(itr.nextToken()); context.write(word, one);

}

}

}

public static class IntSumReducer extends Reducer<Text,IntWritable,Text,IntWritable> {

private IntWritable result = new IntWritable(); public void reduce(Text key, Iterable<IntWritable> values,

Context context

) throws IOException, InterruptedException { int sum = 0; for (IntWritable val : values) {

sum += val.get();

}

result.set(sum); context.write(key, result);

}

} public static void main(String[] args) throws Exception { Configuration conf = new Configuration(); Job job = Job.getInstance(conf, "word count"); job.setJarByClass(WordCount.class); job.setMapperClass(TokenizerMapper.class); job.setCombinerClass(IntSumReducer.class); job.setReducerClass(IntSumReducer.class); job.setOutputKeyClass(Text.class); job.setOutputValueClass(IntWritable.class); FileInputFormat.addInputPath(job, new Path(args[0]));

FileOutputFormat.setOutputPath(job, new Path(args[1]));

System.exit(job.waitForCompletion(true) ? 0 : 1);

}

}

**Output:**

C:\hadoop\hadoop-2.7.1\bin>set classpath=c:\hadoop\hadoop-core-1.2.1.jar

C:\hadoop\hadoop-2.7.1\bin>set classpath=c:\program files\java\jdk1.7.0\bin

C:\hadoop\hadoop-2.7.1\bin>javac WordCount.java

C:\hadoop\hadoop-2.7.1\bin>jar -cvf wc.jar WordCount\*.class

C:\hadoop\hadoop-2.7.1\bin>hadoop fs -mkdir /f1

C:\hadoop\hadoop-2.7.1\bin>hadoop fs -cat file1.txt

Hello world Bye world

C:\hadoop\hadoop-2.7.1\bin>hadoop fs -put file1.txt /f1

C:\hadoop\hadoop-2.7.1\bin>hadoop jar wc.jar WordCount /f1 /f2

File System Counters

FILE: Number of bytes read=6604

FILE: Number of bytes written=566732

FILE: Number of read operations=0

FILE: Number of large read operations=0

FILE: Number of write operations=0

HDFS: Number of bytes read=42

HDFS: Number of bytes written=22

HDFS: Number of read operations=13

HDFS: Number of large read operations=0

HDFS: Number of write operations=4

Map-Reduce Framework

Map input records=1

Map output records=4

Map output bytes=38

Map output materialized bytes=40

Input split bytes=99

Combine input records=4

Combine output records=3

Reduce input groups=3

Reduce shuffle bytes=40

Reduce input records=3

Reduce output records=3

Spilled Records=6

Shuffled Maps =1

Failed Shuffles=0

Merged Map outputs=1

GC time elapsed (ms)=0

Total committed heap usage (bytes)=641728512

Shuffle Errors

BAD\_ID=0

CONNECTION=0

IO\_ERROR=0

WRONG\_LENGTH=0 WRONG\_MAP=0

WRONG\_REDUCE=0

File Input Format Counters

Bytes Read=21

File Output Format Counters

Bytes Written=22

C:\hadoop\hadoop-2.7.1\sbin>hadoop fs -cat /f3/part-r-00000

Bye 1

Hello 1 world 2

# Ex.No 3: Python basic for pandas

1. import pandas as pd
2. # Reading data from google drive in to a data frame data=pd.read\_csv("/content/drive/My Drive/ Code/data/cafe.csv")
3. # Viewing data head (default: first 5 rows) data.head()
4. # Sum of a column

data["AFF-Mon"].sum()

1. # Minimum value in a column data["AFF-Mon"].min()
2. # Numbr of values in a column data["AFF-Mon"].count()
3. # Mean value of a column data["AFF-Mon"].mean()
4. # Basic statistical values of a column data["AFF-Mon"].describe()
5. # STatistics of the whole data frame data.describe()
6. # STatistics of the whole data frame including all columns (numeric and non-numeric) data.describe(include="all")
7. # View last 5 rows of a data frame data.tail()
8. # View column names data.columns
9. # Unique values in a column data["State"].unique()
10. # Number of unique values in a column data["State"].nunique()
11. # Number of rows and columns in a data frame data.shape
12. # Add column to a data frame

# First create a list of numbers for 85 rows

x=[i for i in range(85)] # Assign the list to new column data["New Index"]=x

1. # View column names data.columns
2. data.shape
3. data.head()
4. # Dropping (removing) a column data.drop("New Index",axis=1,inplace=True) # axis=1 for columns and axis=0 for rows.
5. # Conditional statements (or filtering data using conditions)
6. # Example, all store data from the state Tamil Nadu (TN) data[data["State"]=="TN"]
7. # Detecting missing values data.isna()
8. # Detecting missing values in a single column data['Store ID'].isna()
9. # Number of missing values per column data.isna().sum()
10. # Number of missing valyes in the data frame data.isna().sum().sum()
11. # Reading "train.csv" data from google drive in to a data frame df=pd.read\_csv("/content/drive/My Drive/Colab Notebooks/ATAL FDP GGV Python Code/data/train.csv")
12. # View first 5 rows df.head()
13. # Detecting missing values df.isna().sum()
14. # Filling missing values with mean data
15. df["Age"]=df["Age"].fillna(df["Age"].mean())
16. df.isna().sum()

# Ex.No 4: Python Code for Matplotlib

1. import matplotlib.pyplot as plt
2. plt.plot([1,2,3,4,5],[1,4,6,8,9],"ro") plt.show()
3. plt.plot([1,2,3,4,5],[1,4,6,8,9]) plt.title("Graph") plt.xlabel("X-axis") plt.ylabel("Y-axis") plt.show()
4. import numpy as np
5. t=np.arange(0.,5.,0.2) t
6. array([0. , 0.2, 0.4, 0.6, 0.8, 1. , 1.2, 1.4, 1.6, 1.8, 2. , 2.2, 2.4,

2.6, 2.8, 3. , 3.2, 3.4, 3.6, 3.8, 4. , 4.2, 4.4, 4.6, 4.8])

plt.plot(t,t,"r--",t,t\*\*2,'bs',t,t\*\*3,'g^')

plt.title("Linear, Quadratic and Cubic graphs")

plt.xlabel("X-axis") plt.ylabel("Y-axis") plt.show()

1. x1=[5,8,10] y1=[12,6,16] x2=[6,9,11] y2=[6,15,17] plt.plot(x1,y1,"green",label="Age",linewidth=5) plt.plot(x2,y2,"blue",label="Income",linewidth=5)

plt.legend()

plt.grid(True,color='lightgrey') plt.show()

1. # Bar graph x1=[5,8,10] y1=[12,6,16] plt.bar(x1,y1,color="green") plt.show()
2. # Scatter plot x=[6,9,11,14,20,24,30] y=[6,15,17,22,30,37,44] plt.scatter(x,y)

plt.show()

1. # Plotting categorial graph x=[100,200,300,400] y=[12,20,30,40] plt.figure(figsize=(9,3)) plt.subplot(131) plt.bar(x,y) plt.subplot(132) plt.scatter(x,y) plt.subplot(133) plt.plot(x,y)
2. import pandas as pd data=pd.read\_csv("/content/drive/My Drive/cafe.csv") data.head()
3. addr=data["Address"].head() addr
4. mon=data["AFF-Mon"].head() plt.bar(addr,mon) plt.show()
5. plt.bar(addr,mon)

plt.xticks(addr,('a','b','c','d','e')) plt.shw()

1. tue=data["AFF-Tue"].head() wed=data["AFF-Wed"].head()

# Saving graph to a file

plt.plot(addr,mon,'r--',addr,tue,'bs',addr,wed,'g^') plt.xticks(rotation=90)

plt.savefig("/content/drive/My Drive/Colab Notebooks/MyGraph.png",format='png') plt.show()

1. plt.scatter(addr,mon)
2. plt.xticks(rotation=90)
3. plt.show()
4. import folium as f

f.Map(location=[18.932308,72.834091])

# Ex.No 5: Python Code for Seaborn

1. import pandas as pd import numpy as np

import matplotlib.pyplot as plt import seaborn as sns

data=pd.read\_csv("/content/drive/My Drive/train.csv") data.head()

1. # Default Seaborn datasets sns.get\_dataset\_names()
2. # Loading Seaborn predefined dataset df=sns.load\_dataset('tips') df.head()
3. # Range and outliers

plt.scatter(data["Sex"],data["Age"]) plt.show()

1. plt.style.use("seaborn") fig,ax=plt.subplots(figsize=(15,9)) ax.boxplot([data["Fare"]]) plt.show()
2. data["Fare"].max()

512.3292

1. # Identifying outliers using z-score z\_score=(data["Age"]-data["Age"].mean())/data["Age"].std() print(z\_score)
2. # Example: use only data where z-score is less than 3 z\_score<3
3. # Adding z\_score column to data data['z\_score']=z\_score data.head()
4. # Filtering data where z\_score < 3 data[data['z\_score']<3]
5. data.skew()
6. # Histogram (Frequency distribution) sns.displot(data["Age"],kde=True) plt.show()
7. data.groupby(["Sex","Ticket"]).describe()
8. data.groupby(["Embarked","Sex","Pclass"]).size() data.groupby(["Embarked","Sex","Pclass"]).size().unstack()
9. # Arranging data into bins group\_data=pd.DataFrame(pd.cut(data["Age"],bins=[0,15,30,45,60,75,90])) group\_data.head()
10. group\_data.groupby(["Age"]).size()
11. # Bar graph

group\_data.groupby(["Age"]).size().plot(kind="bar") plt.show()

1. # Pie plot group\_data.groupby(["Age"]).size().plot(kind="pie") plt.show()

# 3D plots

1. from mpl\_toolkits import mplot3d fig=plt.figure() ax=plt.axes(projection="3d") plt.show()
2. ax=plt.axes(projection="3d") z=np.linspace(0,15,1000) x=np.sin(z) y=np.cos(z) ax.plot3D(x,y,z,"grey")

zdata=15\*np.random.random(100) xdata=np.sin(zdata)+0.1\*np.random.random(100) ydata=np.cos(zdata)+0.1\*np.random.random(100)

plt.figure(figsize=(10,7))

sns.pairplot(data) plt.show()

# Ex.No 6: Regression Algorithm

1. import pandas as pd
2. Stock\_Market = {'Year':

[2017,2017,2017,2017,2017,2017,2017,2017,2017,2017,2017,2017,2016,2016,2016,2016,

2016,2016,2016,2016,2016,2016,2016,2016],

'Month': [12, 11,10,9,8,7,6,5,4,3,2,1,12,11,10,9,8,7,6,5,4,3,2,1], 'Interest\_Rate':

[2.75,2.5,2.5,2.5,2.5,2.5,2.5,2.25,2.25,2.25,2,2,2,1.75,1.75,1.75,1.75,1.75,1.75,1.75,1.75,1.

75,1.75,1.75],

'Unemployment\_Rate':

[5.3,5.3,5.3,5.3,5.4,5.6,5.5,5.5,5.5,5.6,5.7,5.9,6,5.9,5.8,6.1,6.2,6.1,6.1,6.1,5.9,6.2,6.2,6.1], 'Stock\_Index\_Price':

[1464,1394,1357,1293,1256,1254,1234,1195,1159,1167,1130,1075,1047,965,943,958,971, 949,884,866,876,822,704,719]

}

1. df = pd.DataFrame Stock\_Market,columns=

['Year','Month','Interest\_Rate','Unemployment\_Rate', 'Stock\_Index\_Price']) print (df)

1. import os

os.chdir('/content/drive/My Drive/') os.getcwd()

1. df.to\_csv('data/stock.csv', index=False) df=pd.read\_csv('data/stock.csv') df.head()
2. # Check for Linearity import matplotlib.pyplot as plt

plt.scatter(df['Interest\_Rate'], df['Stock\_Index\_Price'], color='red') plt.title('Stock Index Price Vs Interest Rate', fontsize=14)

plt.xlabel('Interest Rate', fontsize=14) plt.ylabel('Stock Index Price', fontsize=14)

plt.grid(True) plt.show()

1. plt.scatter(df['Unemployment\_Rate'], df['Stock\_Index\_Price'], color='blue') plt.title('Stock Index Price Vs Unemployment Rate', fontsize=14) plt.xlabel('Unemployment Rate', fontsize=14) plt.ylabel('Stock Index Price', fontsize=14)

plt.grid(True) plt.show()

1. from sklearn import linear\_model import statsmodels.api as sm
2. # Perform Multi-Linear Regression

# here we have 2 variables for multiple regression. If you just want to use one

# variable for simple linear regression, then use X = df['Interest\_Rate'] for

# example. Alternatively, you may add additional variables within the brackets

* 1. = df[['Interest\_Rate','Unemployment\_Rate']]
  2. = df['Stock\_Index\_Price']

# with sklearn

regr = linear\_model.LinearRegression() regr.fit(X, Y) print('Intercept: \n', regr.intercept\_) print('Coefficients: \n', regr.coef\_)

# prediction with sklearn

New\_Interest\_Rate = 2.75

New\_Unemployment\_Rate = 5.3

print ('Predicted Stock Index Price: \n', regr.predict([[New\_Interest\_Rate ,New\_Unemployment\_Rate]]))

# With StatModels

X = sm.add\_constant(X) # adding a constant model = sm.OLS(Y, X).fit() # Ordinary Least Squares

predictions = model.predict(X) print\_model = model.summary() print(print\_model)

# Ex.No 7: Model Training

1. from google.colab import drive drive.mount('/content/drive')
2. import os

os.chdir('/content/drive/My Drive/ ') os.getcwd()

1. from keras import layers from keras import models
2. model = models.Sequential()
3. model.add(layers.Conv2D(32, (3, 3), activation='relu', input\_shape=(150, 150, 3)))
4. model.add(layers.MaxPooling2D((2, 2))) model.add(layers.Conv2D(64, (3, 3), activation='relu')) model.add(layers.MaxPooling2D((2, 2))) model.add(layers.Conv2D(128, (3, 3), activation='relu')) model.add(layers.MaxPooling2D((2, 2))) model.add(layers.Conv2D(128, (3, 3), activation='relu')) model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Flatten()) model.add(layers.Dropout(0.5)) model.add(layers.Dense(512, activation='relu')) model.add(layers.Dense(1, activation='sigmoid'))

1. \*\*\*Let’s look at how the dimensions of the feature maps change with every successive layer:\*\*\*

model.summary()

1. # \*\*Configuring the model for training\*\*

\*\*For the compilation step, you’ll go with the RMSprop optimizer. Because you ended the network with a single sigmoid unit, you’ll use binary crossentropy as the loss\*\* from keras import optimizers model.compile(loss='binary\_crossentropy', optimizer=optimizers.RMSprop(lr=1e-4), metrics=['accuracy'])

1. # \*\*Data preprocessing\*\*

Currently, the data sits on a drive as JPEG files, so the steps for getting it into the network are roughly as follows:

* 1. Read the picture files.
  2. Decode the JPEG content to RGB grids of pixels.
  3. Convert these into floating-point tensors.
  4. Rescale the pixel values (between 0 and 255) to the [0, 1] interval (as you know, neural networks prefer to deal with small input values). \*\*Using ImageDataGenerator to read images from directories\*\*

from keras.preprocessing.image import ImageDataGenerator train\_datagen = ImageDataGenerator(rescale=1./255) #test\_datagen = ImageDataGenerator(rescale=1./255)

train\_generator = train\_datagen.flow\_from\_directory('data/images',target\_size=(150, 150), batch\_size=20, class\_mode='binary')

1. The output of one of these generators: it yields batches of 150 × 150 RGB images (shape (20, 150, 150, 3)) and binary labels (shape (20,)).There are 20 samples in each batch (the batch size). for data\_batch, labels\_batch in train\_generator: print('data batch shape:', data\_batch.shape) print('labels batch shape:', labels\_batch.shape) break
2. # \*\*Fitting the model using a batch generator\*\* history = model.fit\_generator( train\_generator,steps\_per\_epoch=10,epochs=5)
3. # \*\*Saving the model\*\* model.save('output/mymodel.h5')
4. ## \*\*Displaying curves of loss and accuracy during training\*\* import matplotlib.pyplot as plt acc = history.history['accuracy'] loss = history.history['loss'] epochs = range(1, len(acc) + 1) plt.plot(epochs, acc, 'b', label='Training acc') plt.title('Training accuracy') plt.legend() plt.figure()

plt.plot(epochs, loss, 'b', label='Training loss')

plt.title('Training loss') plt.legend() plt.show()

# Ex.No 8: Neural\_Networks

1. import keras from keras.layers import Dense from keras.models import Sequential
2. import os

os.chdir('/content/drive/My Drive/') os.getcwd()

1. import pandas as pd import numpy as np
2. ds=pd.read\_csv("Iris.csv") ds.head()
3. arr=ds.values x=arr[:,0:4] y=arr[:,4]
4. model=Sequential() model.add(Dense(units=64,input\_dim=4,activation="relu")) model.add(Dense(units=32,activation="relu")) model.add(Dense(units=3,activation="softmax"))
5. model.compile(loss='categorical\_crossentropy', optimizer='adam', metrics=['accuracy'])
6. from sklearn.preprocessing import LabelEncoder from keras.utils import np\_utils

encoder = LabelEncoder() encoder.fit(y)

encoded\_Y = encoder.transform(y) dummy\_y = np\_utils.to\_categorical(encoded\_Y)

1. h=model.fit(x.astype('float'),dummy\_y,epochs=10)

Epoch 1/10

5/5 [==============================] - 1s 5ms/step - loss: 3.8956 - accuracy:

0.5867

Epoch 2/10

5/5 [==============================] - 0s 3ms/step - loss: 1.5611 - accuracy:

0.4000

Epoch 3/10

5/5 [==============================] - 0s 4ms/step - loss: 0.9363 - accuracy:

0.4800

Epoch 4/10

5/5 [==============================] - 0s 3ms/step - loss: 0.9004 - accuracy:

0.5667

Epoch 5/10

5/5 [==============================] - 0s 4ms/step - loss: 0.7987 - accuracy:

0.6133

Epoch 6/10

5/5 [==============================] - 0s 3ms/step - loss: 0.6825 - accuracy:

0.6600

Epoch 7/10

5/5 [==============================] - 0s 3ms/step - loss: 0.6173 - accuracy:

0.7933

Epoch 8/10

5/5 [==============================] - 0s 5ms/step - loss: 0.6038 - accuracy:

0.7667

Epoch 9/10

5/5 [==============================] - 0s 3ms/step - loss: 0.5508 - accuracy:

0.7733

Epoch 10/10

5/5 [==============================] - 0s 3ms/step - loss: 0.5352 - accuracy: 0.7867

10. from matplotlib import pyplot as plt plt.plot(h.history['accuracy']) plt.plot(h.history['loss']) plt.ylabel('Accuracy/Loss') plt.xlabel('Epoch')

plt.legend(['Accuracy', 'Loss'], loc='upper left') plt.show()

