GALGOTIAS UNIVERSITY

LAB MANUAL

DEEP LEARNING

Name of School: SCHOOL OF COMPUTING SCIENCE & ENGINEERING

Department: COMPUTER SCIENCE & ENGINEERING

Year: 2023-2024

GALGOTIAS UNIVERSITY

(Under the Uttar Pradesh Private Universities Act No. 12 of 2019)

SUBJECT	**Deep Learning**	**PROGRAMME**	**B. Tech.**
SUBJECT CODE	**R1UC604C**	**SEMESTER**	**VI**
CREDITS	**5**	**DURATION OF SEMESTER**	**15 Weeks**
PREREQUISITE SUBJECTS	**Machine Learning, Python**	**SESSION DURATION**	**3 + 2 Hrs per Week**

Vision

To be recognized globally as a premier School of Computing Science and Engineering for imparting quality and value based education within a multi-disciplinary and collaborative research based environment.

Mission

The mission of the school is to:

M1: Develop a strong foundation in fundamentals of computing science and engineering with responsiveness towards emerging technologies.

M2: Establish state-of-the-art facilities and adopt education 4.0 practices to analyze, develop, test and deploy sustainable ethical IT solutions by involving multiple stakeholders.

M3: Foster multidisciplinary collaborative research in association with academia and industry through focused research groups, Centre of Excellence, and Industry Oriented R&D Labs.

PROGRAM EDUCATIONAL OBJECTIVES

The Graduates of Computer Science and Engineering shall:

PEO1: be engaged with leading Global Software Services and Product development companies handling projects in cutting edge technologies.

PEO2: serve in technical or managerial roles at Government firms, Corporates and contributing to thesociety as successful entrepreneurs through startup.

PEO3: undertake higher education, research or academia at institutions of transnational reputation.

PROGRAMME SPECIFIC OUTCOME (PSO)

The students of Computer Science and Engineering shall:

PSO1: Have the ability to work with emerging technologies in computing requisite to Industry 4.0.

PSO2: Demonstrate Engineering Practice learned through industry internship and research project to solve live problems in various domains.

PROGRAMME OUTCOME (PO)

PO1 Computing Science knowledge: Apply the knowledge of mathematics, statistics, computing science and information science fundamentals to the solution of complex computer application problems.

PO2 Problem analysis: Identify, formulate, review research literature, and analyze complex computing science problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and computer sciences.

PO3 Design/development of solutions: Design solutions for complex computing problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

PO4 Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO5 Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern computing science and IT tools including prediction and modeling to complex computing activities with an understanding of the limitations.

PO6 IT specialist and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional computing science and information science practice.

PO7 Environment and sustainability: Understand the impact of the professional computing science solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO8 Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the computing science practice.

PO9 Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10 Communication: Communicate effectively on complex engineering activities with the IT analyst community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

PO11 Project management and finance: Demonstrate knowledge and understanding of the computing science and management principles and apply these to one’s own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO12 Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

COURSE OBJECTIVE

- To understand the fundamentals of neural networks.
- To apply the concept of neural network in real-time use of deep learning.
- To apply the basic concept of deep learning in the implementation of convolutional neural network over various datasets.
- To implement various automatic model of Deep Learning neural networks and evaluate their performance.

COURSE OUTCOMES(COs)

After the completion of the course, the student will be able to:

CO No.	Course Outcomes
R1UC604C.1	Understand the fundamentals of Neural Networks.
R1UC604C.2	Apply the concepts of Neural Networks in the development of deep learning models.
R1UC604C.3	Develop and Analyze the deep learning models like RNN, CNN and LSTM.
R1UC604C.4	To be able to evaluate and optimize deep learning models

BLOOM'S LEVEL OF THE COURSE OUTCOMES

CO No.	Remember **KL1**	Understand **KL 2**	Apply **KL 3**	Analyse **KL 4**	Evaluate **KL 5**	Create **KL 6**
R1UC604C.1	√	√				
R1UC604C.2			√			
R1UC604C.3				√		
R1UC604C.4					√	√

COURSE ARTICULATION MATRIX

The Course articulation matrix indicates the correlation between Course Outcomes and Program Outcomes and their expected strength of mapping in three levels (low, medium, and high).

COs#/ POs	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2
R1UC604C.1	1													
R1UC604C.2	3					1							1	
R1UC604C.3		3		3										2
R1UC604C.4			3		2								3	

Note: 1-Low, 2-Medium, 3-High

COURSE ASSESSMENT

Type of Course (C)	CIE			Total Marks		Final Marks CIE*0.5+SEE*0.5
	LAB@ (Work+ Record)	MTE	Course-based Project^	CIE	SEE	
COMPREHENSIVE	25	50	25	100	100	100

Rubrics for Practical IA

S. No.	Rubrics - Parts	Marks
1	Performance	**5**
2	Result	**7**
3	File	**5**
4	Viva	**8**
Total		**25**

List of Programs

List of Experiments	
Sr. No	**Experiments**
1	Python Programming Fundamental Revision.
2	Python Programming Libraries Revision.
3	Python program to implement k-Nearest Neighbor algorithm to classify the iris dataset. Print both correct and wrong predictions.
4	Python program to implement the non-parametric locally weighted regression algorithm in order to fit the data point. Select the appropriate data set for your experiment and draw graphs.
5	Python program to build a machine learning model which will predict whether or not it will rain tomorrow by studying past data.
6	Python program to implement AND, OR gates using perceptron.
7	Python program for classification of an XOR problem with multi-layer perceptron.
8	Python program to implement classification linearly separable data with perceptron.
9	Python program to recognize handwritten digits using neural network.
10	Python program to study a bank credit data set and determine whether a transaction is fraudulent or not based on past data.
11	Python program for implementation of CNN.
12	Python program for implementation of RNN.
13	Python program for implementation of LSTM.
14	Python program for implementation of Bidirectional LSTM.
15	Revision

List of course based projects	
1	AI-based Game Playing Agent: Create an AI agent capable of playing and winning games like chess, Go, or Atari games using reinforcement learning.
2	Handwritten Digit Recognition: Implement a neural network to classify handwritten digits from the MNIST dataset.
3	Sentiment Analysis: Build a model to analyze sentiment in text data, such as movie reviews or tweets.
4	Image Classification: Develop a convolutional neural network (CNN) to classify images from datasets like CIFAR-10 or ImageNet.
5	Predictive Maintenance: Create a model to predict equipment failure or maintenance needs based on sensor data.
6	Healthcare Diagnosis: Use machine learning techniques to assist in medical diagnosis based on patient data or medical images.
7	Stock Price Prediction: Build a model to predict stock prices using historical data and technical indicators.
8	Spam Email Detection: Develop a classifier to identify spam emails from a given dataset.
9	Recommendation Systems: Create a recommendation system for movies, products, or music based on user preferences.
10	Anomaly Detection: Build a model to detect anomalies in data, such as fraudulent transactions or network intrusions.
11	Natural Language Processing (NLP): Implement NLP techniques for tasks like text summarization, named entity recognition, or language translation.
12	Facial Recognition: Develop a system to recognize faces in images or videos using deep learning techniques.
13	Autonomous Vehicles Simulation: Create a simulated environment where autonomous vehicles navigate using reinforcement learning algorithms.
14	Gesture Recognition: Build a model to recognize hand gestures from images or videos, which can be used for human-computer interaction.
15	Customer Churn Prediction: Develop a model to predict customer churn for businesses based on historical customer data.
16	Fake News Detection: Implement a model to classify news articles as real or fake based on their content.
17	Predictive Analytics for Energy Consumption: Use machine learning to predict energy consumption patterns and optimize energy usage.
18	Credit Risk Assessment: Build a model to assess credit risk for loan applicants using historical financial data.
19	Object Detection: Develop a model to detect and localize objects within images or videos.
20	Human Activity Recognition: Create a system to recognize different human activities, such as walking, running, or sitting, from sensor data.
21	Brain-Computer Interface: Explore EEG data and build a model to interpret brain signals for controlling devices or applications.
22	Customer Segmentation: Use clustering algorithms to segment customers based on their behavior or demographics.
23	Language Translation: Implement a neural machine translation system to translate text between different languages.

24	Document Classification: Build a model to classify documents into predefined categories based on their content.
25	Image Style Transfer: Develop a model to transfer the style of one image onto another, creating artistic effects.
26	Time Series Forecasting: Use recurrent neural networks (RNNs) or long short-term memory networks (LSTMs) to forecast time series data, such as stock prices or weather patterns.
27	Emotion Recognition: Build a model to recognize emotions from facial expressions in images or videos.
28	Chatbot Development: Create a conversational AI chatbot that can answer user queries or assist in tasks.
29	Semantic Segmentation: Develop a model to segment images at the pixel level, identifying different objects or regions within the image.
30	Speech Recognition: Build a system to transcribe spoken language into text using deep learning techniques.
31	Customer Lifetime Value Prediction: Develop a model to predict the lifetime value of customers for a business based on their past behavior.
32	Video Activity Recognition: Extend human activity recognition to videos, identifying activities performed over a sequence of frames.
33	Automated Essay Scoring: Build a model to automatically score essays based on their content, coherence, and grammar.
34	Fraud Detection: Develop a model to detect fraudulent transactions in banking or e-commerce systems.
35	Music Genre Classification: Create a model to classify music into different genres based on audio features.
36	Topic Modeling: Use techniques like Latent Dirichlet Allocation (LDA) to identify topics within a collection of documents.
37	Object Tracking: Implement algorithms to track objects in videos across frames, useful for surveillance or self-driving cars.
38	Satellite Image Analysis: Analyze satellite imagery for tasks like land cover classification, urban development monitoring, or disaster response.
39	Hand Gesture Recognition: Extend gesture recognition to recognize hand gestures in real-time for controlling devices or applications.
40	E-commerce Recommendation: Build a personalized recommendation system for an e-commerce platform based on user browsing and purchase history.
41	Drug Discovery: Use machine learning to predict the biological activity of molecules and assist in drug discovery processes.
42	Smart Home Automation: Develop a system to automate tasks in a smart home environment based on sensor data and user preferences.
43	Facial Attribute Detection: Build a model to detect facial attributes such as age, gender, or facial hair from images.
44	Speech Emotion Recognition: Develop a model to recognize emotions from speech signals, useful for applications like customer service or mental health monitoring.
45	Urban Mobility Prediction: Predict traffic flow or transportation demand in urban areas using machine learning techniques.

Coding Implementation of Lab Problems

1. Python program to calculate the percentage of marks:

```
math=float(input('enter marks of maths subject : '))
english=float(input('enter marks of english subject : '))
physics=float(input('enter marks of physics subject : '))
chemistry=float(input('enter marks of chemistry subject : '))
Art=float(input('enter marks of Art subject : '))
sum=(maths +english+physics+chemistry+Art)
print('Sum of marks is : ',sum) percentage=(sum/250)*100
print('Percentage is : ',percentage)
if(percentage<40):
	print('You are fail')
elif(40<percentage<60):
	print('You got C grade')
elif(60<percentage<80):
	print('You got B grade')
elif(percentage>=80):
	print('You got A grade')
else:
	print('Default')
```

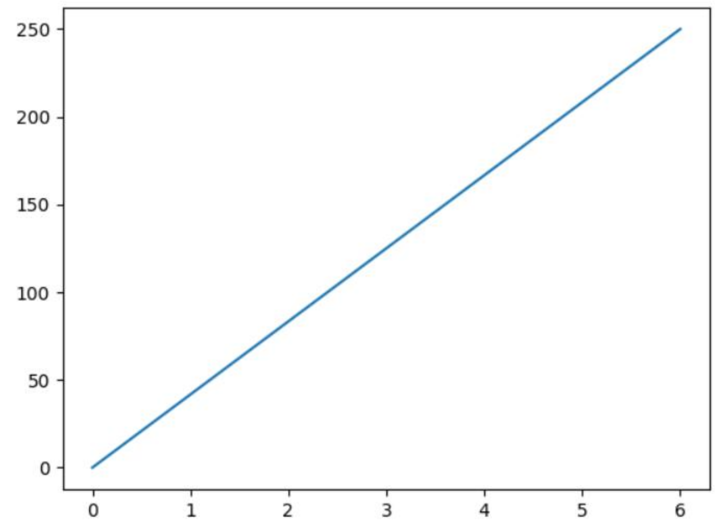
Output:

```
enter marks of maths subject : 80
enter marks of english subject : 60
enter marks of physics subject : 50
enter marks of chemistry subject : 90
enter marks of Art subject : 40
Sum of marks is :  320.0
Percentage is :  128.0
You got A grade
```

2. Python program for working on Data Visualization:

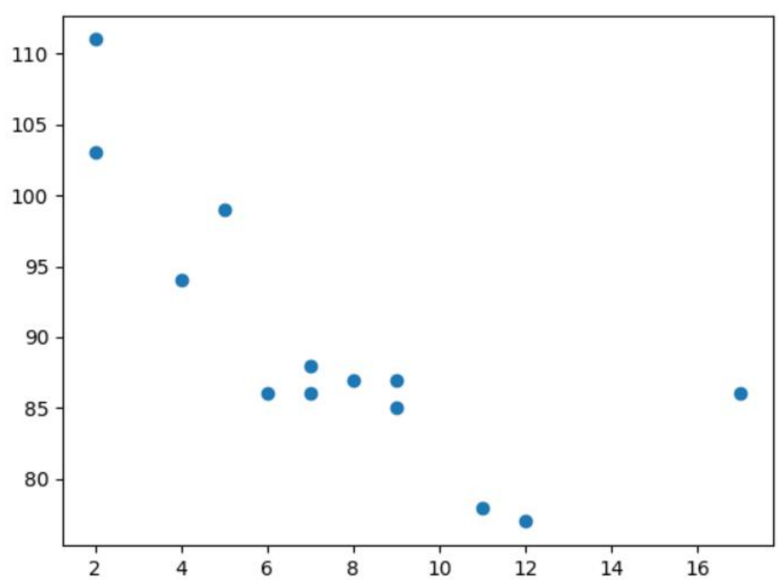
```
import sys
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
import numpy as np

xpoints = np.array([0, 6])
ypoints = np.array([0, 250])
plt.plot(xpoints, ypoints)
plt.show()
plt.savefig(sys.stdout.buffer)
sys.stdout.flush()
```

Output:



x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
plt.scatter(x, y) plt.show()
plt.savefig(sys.stdout.buffer)
sys.stdout.flush()

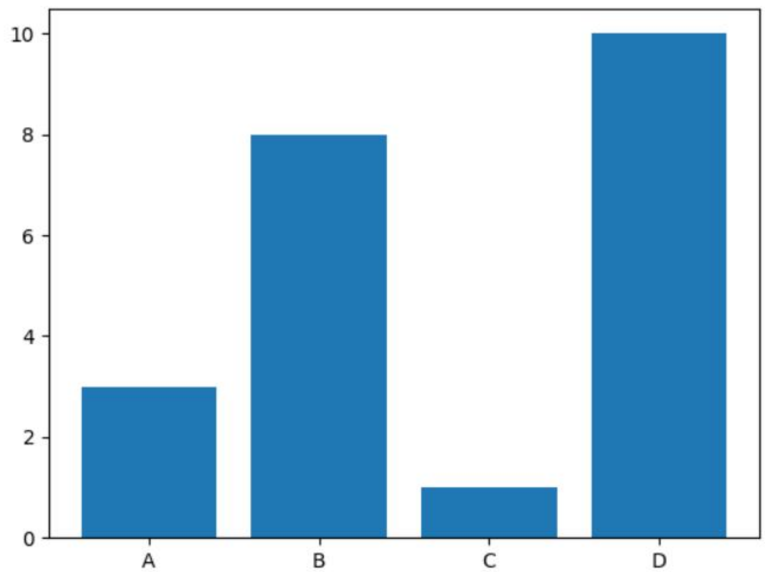
Output:



x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])
plt.bar(x,y) plt.show()
plt.savefig(sys.stdout.buffer)

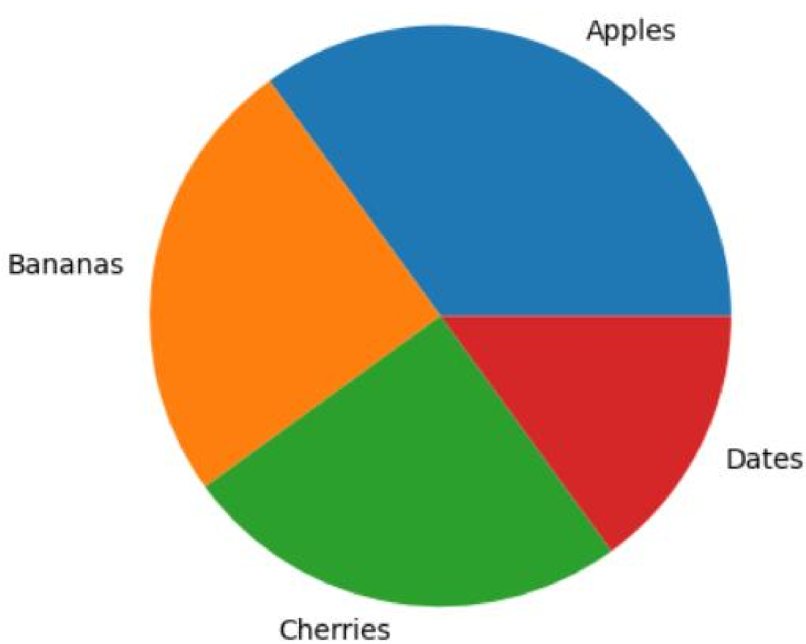
sys.stdout.flush()

Output:



y = np.array([35, 25, 25, 15])
mylabels = ["Apples", "Bananas", "Cherries", "Dates"]
plt.pie(y, labels = mylabels) plt.show()
plt.savefig(sys.stdout.buffer)
sys.stdout.flush()

Output:



3. Data Pre-processing using Linear Regression

```
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
df = pd.DataFrame({"Job Position": ['CEO', 'Senior Manager', 'Junior Manager', 'Employee',
'Assistant Staff'], "Years of Experience":[5, 4, 3, None, 1], "Salary":[100000,80000,None,40000,
20000]})
#print(df)

dropped_df = df.drop([2,3],axis=0)
print("After removing the missing values")
print(dropped_df)
#df['Years of Experience'] = df['Years of Experience'].fillna(df['Years of Experience'].mean())
#df['Salary'] = df['Salary'].fillna(df['Salary'].mean())
#print(df)
train_df = df.drop([2,3],axis=0)

# Creating linear regression model
regr = LinearRegression()
# Here the target is the Salary and the feature is Years of Experience
regr.fit(train_df[['Years of Experience']],train_df[['Salary']])
# Predicting for 3 years of experience
regr.predict([[3]])
print("Salary with 3 year of experience")
print(regr.predict([[3]]))
regr.fit(train_df[['Salary']],
train_df[['Years of Experience']])
print("Year of experience with Salary 40000.0")
print(regr.predict([[40000.0]]))
```

Output:

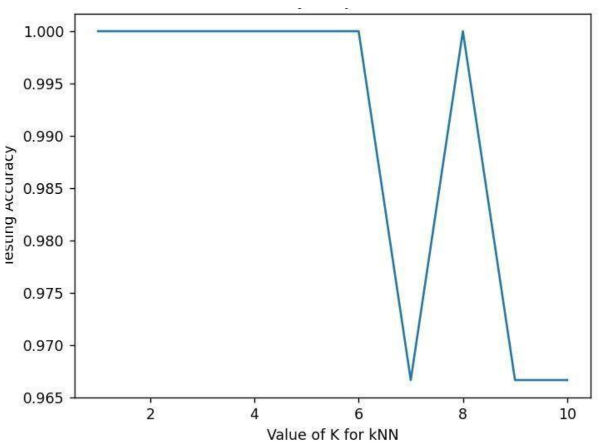
```
Tanya Rajawat, 21SCSE1420066
      Job Position  Years of Experience    Salary
0              CEO                  5.0  100000.0
1   Senior Manager                  4.0   80000.0
2   Junior Manager                  3.0       NaN
3         Employee                  NaN   40000.0
4  Assistant Staff                  1.0   20000.0
Salary with 3 year of experience
[[60000.]]
Year of experience with Salary 40000.0
[[2.]]
```

4. Python program to implement k-Nearest Neighbor algorithm to classify the iris dataset. Print both correct and wrong predictions.

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn import datasets
from sklearn.neighbors import KNeighborsClassifier
# import the iris dataset
iris = datasets.load_iris()
#print(iris)
X = iris.data #print(X)
y = iris.target
# splitting X and y into training and testing sets in 80:20 Ratio
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)
k_range = range(1,11)
score = {}
scores_list = []
for k in k_range:
        knn = KNeighborsClassifier(n_neighbors=k)
        knn.fit(X_train,y_train)
        knn_pred = knn.predict(X_test)
        score[k] = accuracy_score(y_test, knn_pred)
        scores_list.append(accuracy_score(y_test, knn_pred))
import matplotlib.pyplot as plt
plt.plot(k_range, scores_list)
plt.xlabel("Value of K for kNN")
plt.ylabel("Testing Accuracy")
plt.show()

knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train,y_train)
knn_pred = knn.predict(X_test)
print("Final Accuracy : ", accuracy_score(y_test, knn_pred))
classes={0:'setosa',1:'versicolor',2:'virginica'}
x_new=[[3,4,5,6],[5,4,4,4]]
y_predict= knn.predict(x_new)
print(classes[y_predict[0]])
print(classes[y_predict[1]])
```

Output:

```
Final Accuracy :  1.0
virginica
virginica
```



5. Python program to implement the non-parametric locally weighted regression algorithm in order to fit the data point. Select the appropriate data set for your experiment and draw graphs

```
import numpy as np from bokeh.plotting import figure, show,
output_notebook from bokeh.layouts import gridplot from bokeh.io
import push_notebook
output_notebook()
import numpy as np
def local_regression(x0, X, Y, tau):
        x0 = np.r_[1, x0]
        X = np.c_[np.ones(len(X)), X]
          xw = X.T * radial_kernel(x0, X, tau)
         beta = np.linalg.pinv(xw @ X) @ xw @ Y # @ Matrix Multiplication or Dot Product
          # predict value
          return x0 @ beta # @ Matrix Multiplication or Dot Product for prediction
def radial_kernel(x0, X, tau):
        return np.exp(np.sum((X - x0) ** 2, axis=1) / (-2 * tau * tau))
n = 1000
# generate dataset
X = np.linspace(-3, 3, num=n)
print("The Data Set ( 10 Samples) X :\n",X[1:10]) Y =
np.log(np.abs(X ** 2 - 1) + .5)
print("The Fitting Curve Data Set (10 Samples) Y :\n",Y[1:10])
X += np.random.normal(scale=.1, size=n)
print("Normalised (10 Samples) X :\n",X[1:10])
```

Output:

```
The Data Set ( 10 Samples) X :
 [-2.99399399 -2.98798799 -2.98198198 -2.97597598 -2.96996997 -2.96396396
 -2.95795796 -2.95195195 -2.94594595]
The Fitting Curve Data Set (10 Samples) Y :
 [2.13582188 2.13156806 2.12730467 2.12303166 2.11874898 2.11445659
 2.11015444 2.10584249 2.10152068] Normalised (10
Samples) X :
 [-2.98752194 -3.06993616 -3.04170655 -2.87074126 -2.91058693 -2.82554065
```

```
-2.98545513 -2.88038733 -2.81409832]
```

Program continues:

```
domain = np.linspace(-3, 3, num=300)
print(" Xo Domain Space(10 Samples) :\n",domain[1:10])
def plot_lwr(tau):
        # prediction through regression prediction = [local_regression(x0, X, Y, tau)
        for x0 in domain] plot = figure(plot_width=400, plot_height=400)
        plot.title.text='tau=%g' % tau plot.scatter(X, Y, alpha=.3) plot.line(domain,
        prediction, line_width=2, color='red') return plot
```

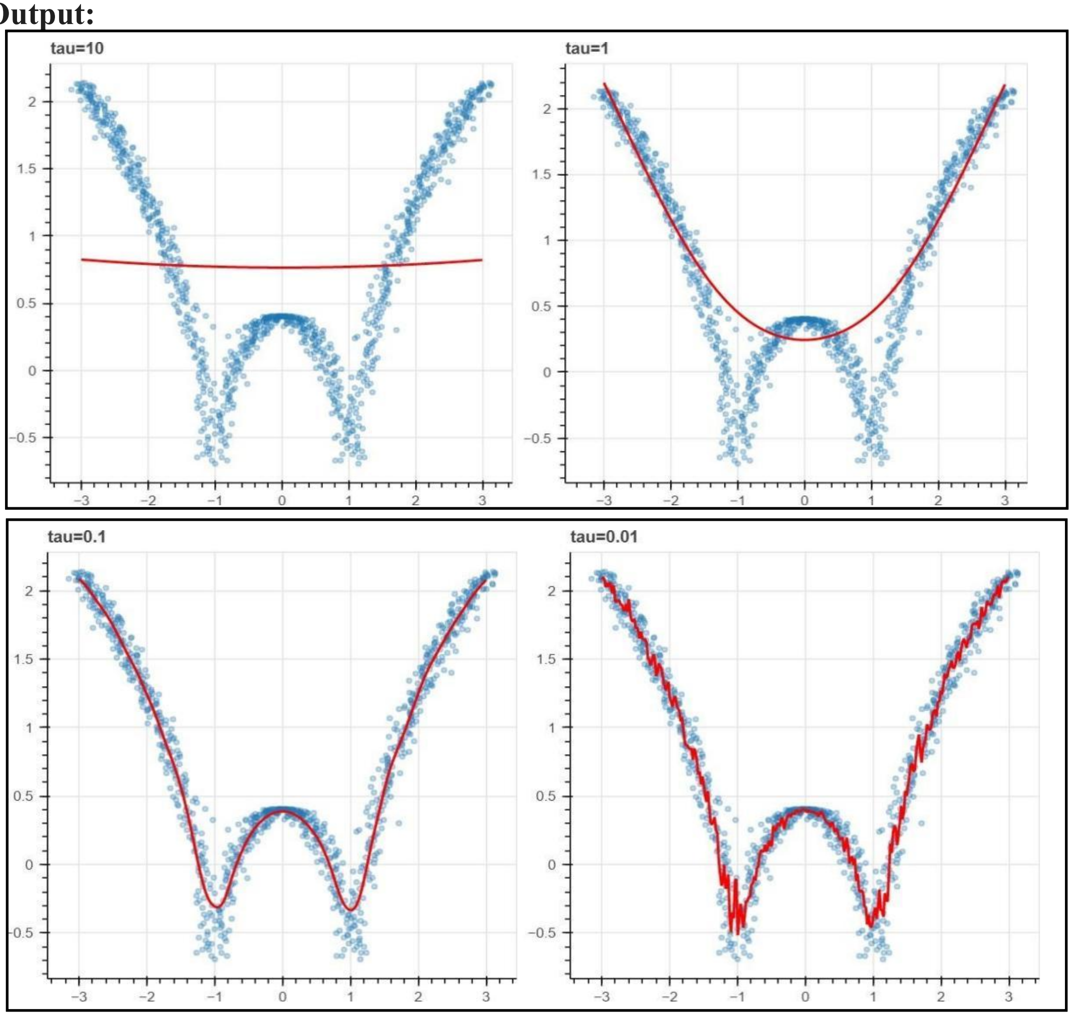
Output:

```
Xo Domain Space(10 Samples) :
 [-2.97993311 -2.95986622 -2.93979933 -2.91973244 -2.89966555 -2.87959866
 -2.85953177 -2.83946488 -2.81939799]
```

Program continues:

```
# Plotting the curves with different tau value show(gridplot([
        [plot_lwr(10.), plot_lwr(1.)],
        [plot_lwr(0.1), plot_lwr(0.01)]]))
```

Output:

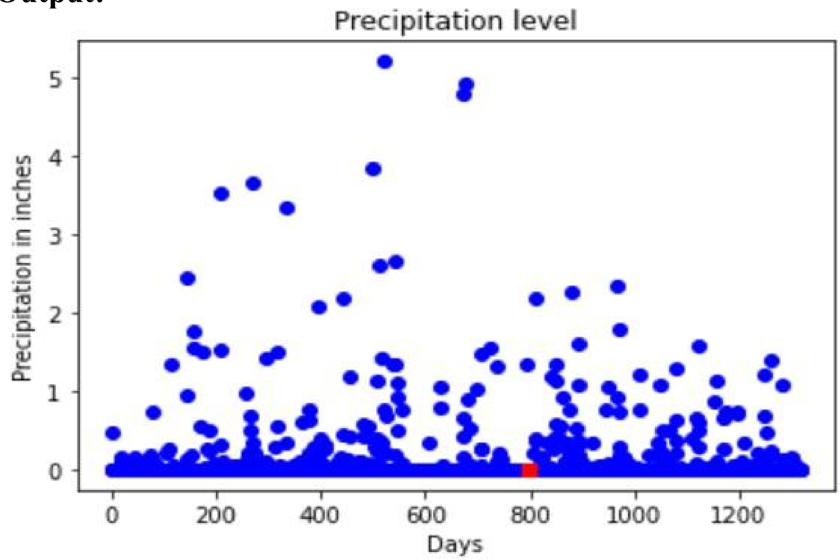


6. Python program to build a machine learning model which will predict whether or not it will rain tomorrow by studying past data

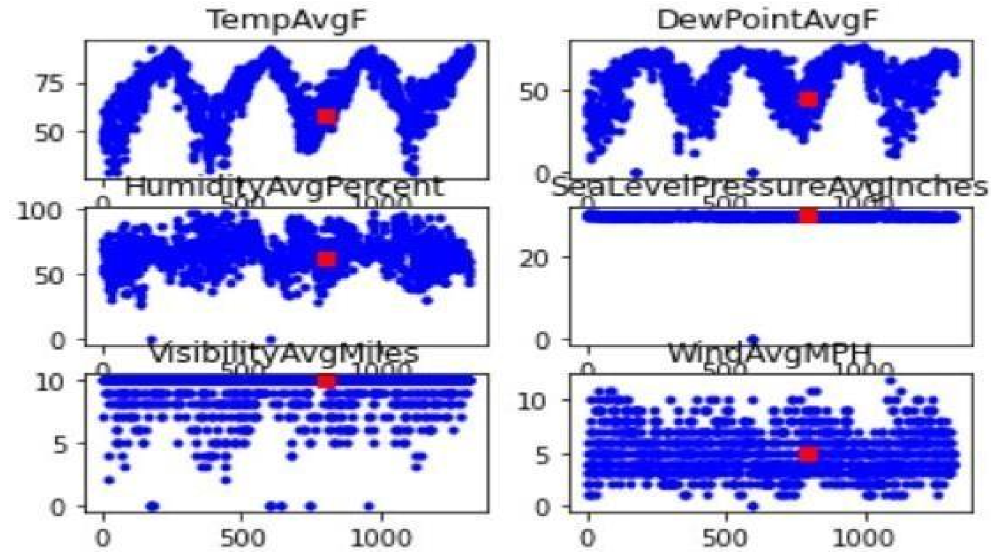
```
 # for loading the austin_weather.csv file
 from google.colab import files
 uploaded = files.upload()
 import pandas as pd import numpy as np
 data = pd.read_csv("austin_weather.csv")
data = data.drop(['Events', 'Date', 'SeaLevelPressureHighInches', 'SeaLevelPressureLowInches'],
axis = 1)
 data = data.replace('T', 0.0)
 data = data.replace('-', 0.0)
 # save the data in a csv file
 data.to_csv('austin_final.csv')

 import sklearn as sk from sklearn.linear_model import LinearRegression
 import matplotlib.pyplot as plt
 data = pd.read_csv("austin_final.csv")
 X = data.drop(['PrecipitationSumInches'], axis = 1)
 Y = data['PrecipitationSumInches']
 Y = Y.values.reshape(-1, 1)

 day_index = 798 # consider a random day in the dataset
 days = [i for i in range(Y.size)]
 clf = LinearRegression() clf.fit(X, Y)
 inp = np.array([[74], [60], [45], [67], [49], [43], [33], [45],
             [57], [29.68], [10], [7], [2], [0], [20], [4], [31]])
inp = inp.reshape(1, -1)

 # print the output.
 print('The precipitation in inches for the input is:', clf.predict(inp))
 print("the precipitation trend graph: ")
plt.scatter(days, Y,color = 'b')
plt.scatter(days[day_index], Y[day_index], color ='r', marker= 's')
plt.title("Precipitation level")
plt.xlabel("Days")
plt.ylabel("Precipitation in inches")
plt.show()
x_vis = X.filter(['TempAvgF', 'DewPointAvgF', 'HumidityAvgPercent',
'SeaLevelPressureAvgInches', 'VisibilityAvgMiles', 'WindAvgMPH'], axis = 1)

 print("Precipitation vs selected attributes graph: ")

 for i in range(x_vis.columns.size):
      plt.subplot(3, 2, i + 1) plt.scatter(days, x_vis[x_vis.columns.values[i][:100]], color
      = 'b',
     marker=".")
                    plt.scatter(days[day_index], x_vis[x_vis.columns.values[i]][day_index], color
     ='r', marker= "s") plt.title(x_vis.columns.values[i])
```

plt.show()

Output:




```
Precipitation vs selected attributes graph:
```



7. Python program to implement AND, OR gates using perceptron

```
import numpy as np x =
np.array([[0,0,0],
          [0,0,1],
          [0,1,0],
          [0,1,1],
          [1,0,0],
          [1,0,1],
          [1,1,0],
          [1,1,1]])
y = np.array([[0,0,0,0,0,0,0,1]]).T def
```

```
fun(x,d=False): op = 1/(1+np.exp(-x)) if
d==False:
            return op
      else:
            return x*(1-x)
np.random.seed(1)

b = 2*np.random.random((3,1))-1 for it
in range(200000): p = x q =
fun(np.dot(p,b))
      err = y - q
      q_del = err * fun(q,True)
b  +=  np.dot(p.T,q_del)  print("Output
After Training") w=[] for x in q:
      if x < 0.6:
            w.append(0)
      else:
            w.append(1)
print(w)
```

Output:

```
Output After Training
[0, 0, 0, 0, 0, 0, 0, 0]
```

Implementation of OR Gate using single layer network with 3 inputs **Program:**

```
import numpy as np
x = np.array([[0,0,0], [0,0,1],
                      [0,1,0],
                      [0,1,1],
                      [1,0,0],
                      [1,0,1],
                      [1,1,0],
                      [1,1,1]])
y   =   np.array([[0,1,1,1,1,1,1,1]]).T   def
sig(x,der=False):  op  =  1/(1+np.exp(-x))  if
der==False: return op
      else:
            return x*(1-x)
np.random.seed(1)
syn0  =  2*np.random.random((3,1))-1  for  it
in    range(200000):    l0    =    x    l1    =
sig(np.dot(l0,syn0))
      err = y - l1
           l1_del = err * sig(l1,True)
syn0 += np.dot(l0.T,l1_del)
print("Output After Training")
```

```
l2=[] for x in l1:
        if x < 0.6:
                l2.append(0)
        else:
                l2.append(1)

print(l2)
```

Output:

```
Output After Training
[0, 1, 1, 1, 1, 1, 1, 1]
```

8. Python program for classification of an XOR problem with multi-layer perceptron

Description of the multilayer perceptron used in the program

- **Input Layer Units** = 2 (Can be modified)
- **Hidden Layer Units** = 2 (Can be modified)
- **Output Layer Units** = 1 (Since this is problem specific, it can't be modified)
- **No. of hidden layers** = 1
- **Learning Algorithm** = Backpropagation

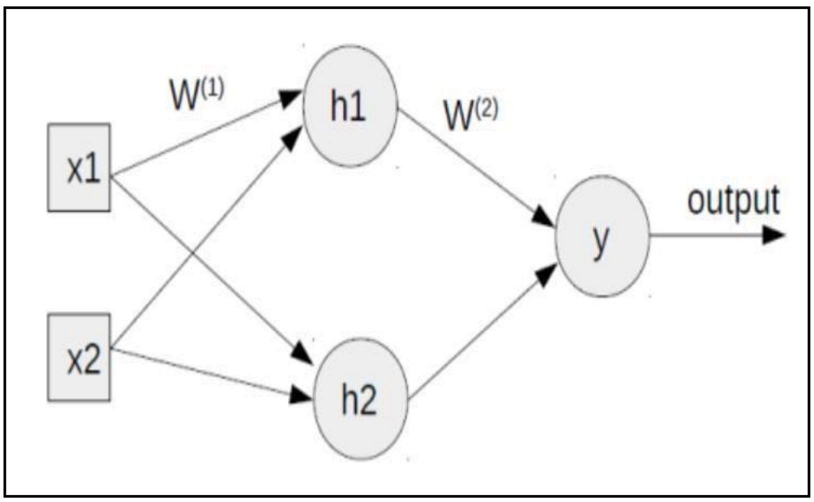


Fig: Showing the perceptron used for the classification of XOR gate

```
import numpy as np
import matplotlib.pyplot as plt import
sys

X = np.array([[0, 1], [1, 0], [1, 1], [0, 0]])
y = np.array([ [1], [1], [0], [0] ])

num_i_units = 2
```

```
num_h_units = 2
num_o_units = 1
learning_rate = 0.1
reg_param = 0 max_iter =
5000
m = 4 # Number of training examples

# The model needs to be over fit to make predictions
np.random.seed(1)
W1 = np.random.normal(0, 1, (num_h_units, num_i_units))
W2 = np.random.normal(0, 1, (num_o_units, num_h_units))

B1 = np.random.random((num_h_units, 1)) B2
= np.random.random((num_o_units, 1)) def
sigmoid(z, derv=False): if derv: return z * (1 - z)
return 1 / (1 + np.exp(-z))

def forward(x, predict=False):
   a1 = x.reshape(x.shape[0], 1) # Getting the training example as a column vector.
   z2 = W1.dot(a1) + B1 a2 = sigmoid(z2)
   z3 = W2.dot(a2) + B2 a3 = sigmoid(z3)

     if predict:
        return a3
   return (a1, a2, a3)

dW1 = 0
dW2 = 0

dB1 = 0
dB2 = 0

cost = np.zeros((max_iter, 1))
for i in range(max_iter):
     c = 0

     dW1 = 0
     dW2 = 0

     dB1 = 0 dB2 = 0 for j
     in range(m):
          sys.stdout.write("\rIteration: {} and {}".format(i + 1, j + 1))

          # Forward Propagation
          a0 = X[j].reshape(X[j].shape[0], 1)
          z1 = W1.dot(a0) + B1 a1
          = sigmoid(z1)
```

```
z2 = W2.dot(a1) + B2 a2 =
sigmoid(z2)

# Back propagation
dz2 = a2 - y[j] dW2 +=
dz2 * a1.T

     dz1 = np.multiply((W2.T * dz2), sigmoid(a1, derv=True))
dW1 += dz1.dot(a0.T)

dB1 += dz1 dB2 += dz2

c = c + (-(y[j] * np.log(a2)) - ((1 - y[j]) * np.log(1 - a2))) sys.stdout.flush()
W1 = W1 - learning_rate * (dW1 / m) + ( (reg_param / m) * W1)
W2 = W2 - learning_rate * (dW2 / m) + ( (reg_param / m) * W2)

B1 = B1 - learning_rate * (dB1 / m) B2 = B2 -
learning_rate * (dB2 / m) cost[i] = (c / m) + (
    (reg_param / (2 * m)) *
    ( np.sum(np.power(W1, 2)) + np.sum(np.power(W2, 2)) ) )
```

Output:

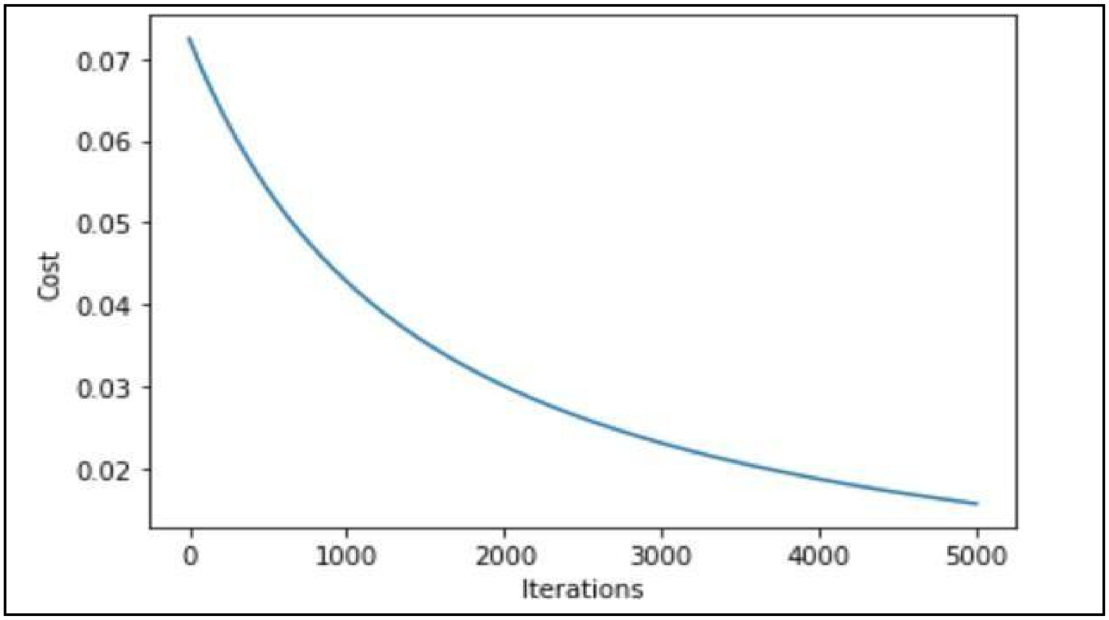
Iteration: 5000 and 4

Program continues:

```
for x in X: print("\n") print(x)
        print(forward(x, predict=True))

plt.plot(range(max_iter), cost)
plt.xlabel("Iterations")
plt.ylabel("Cost") plt.show()
```

Output:

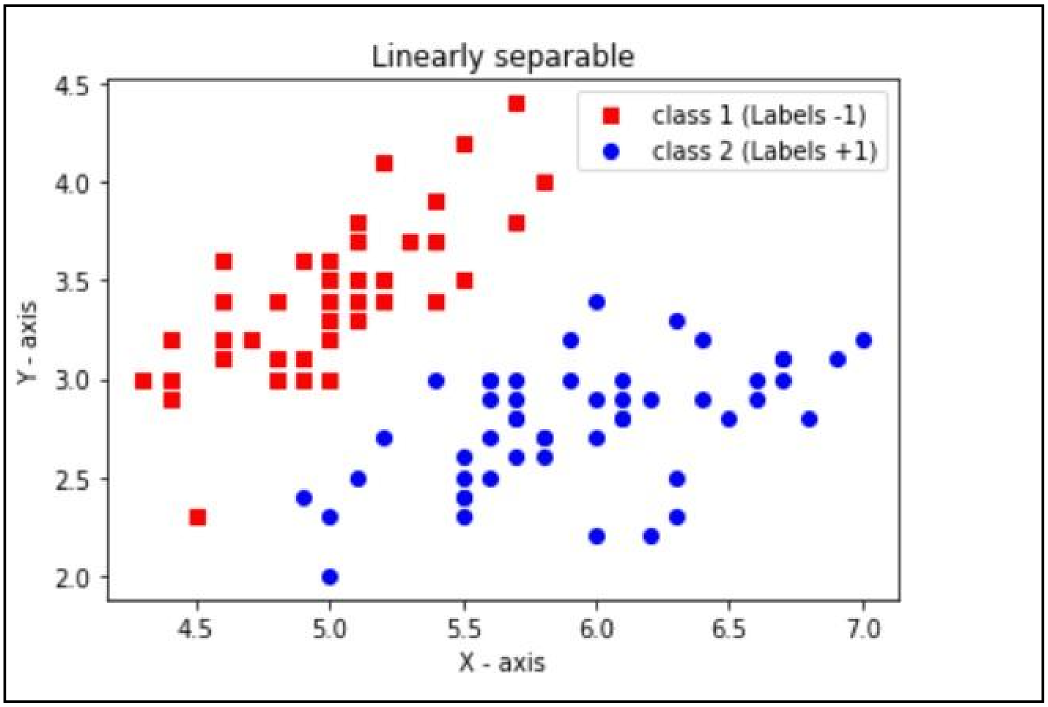
[0 1]
[[0.98273928]] [1 0]
[[0.98328979]] [1 1]
[[0.01663556]]
[0 0]
[[0.01173706]]



9. Python program to implement classification linearly separable data with perceptron

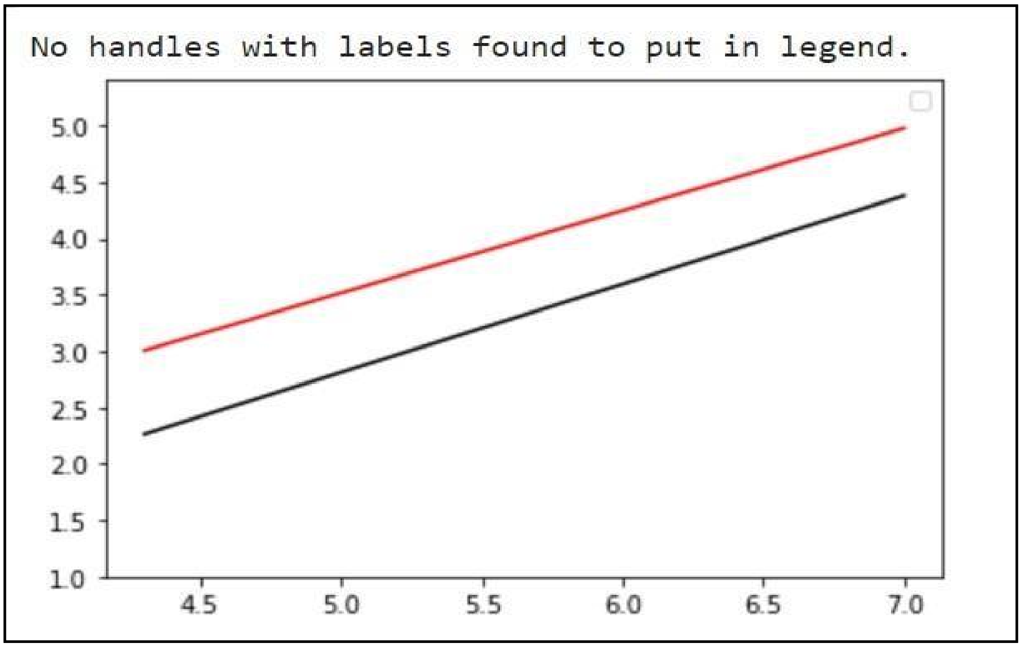
```
import matplotlib.pyplot as plt from sklearn import
datasets import numpy as np from sklearn.linear_model
import Perceptron np.random.seed(6) def
hypothesis_prediction(W,X):
    prediction = np.dot(W,X) if
    prediction>=0:
        return 1
    else:
        return -1
def update_weights(weights, X, y):
    for x,y in zip(X,y):
        predicted_label = hypothesis_prediction(weights, x) error =
        eta*(y-predicted_label) delta_w = error *x weights =
        weights+delta_w
    return weights
if name== "_main_":
    num_iter = 500 eta =
    0.01
iris = datasets.load_iris() X = iris.data[:, :2] y = iris.target plt.scatter(X[:50,0],X[:50,1],
    c='r',marker= 's',label='class 1
   (Labels -1)') plt.scatter(X[50:100,0],X[50:100,1], c='b',label='class 2 (Labels
   +1)') plt.title('Linearly separable')
    plt.xlabel('X - axis') plt.ylabel('Y - axis')
    plt.legend(loc='upper right') plt.show()
```

Output:




```
training_data = X[:100,:]

X = np.c_[np.ones(training_data.shape[0]), training_data] y = y[:100,]

y[np.where(y==0)] = -1 weights =
np.zeros(X.shape[1])
for i in range(num_iter): weights = update_weights(weights, X,
y) weights = weights

clf = Perceptron() clf.fit(X,y)
w0,w1,w2 = clf.coef_[0]
x_min = min(X[:100,1])
x_max = max(X[:100,1])
y_min = min(X[:100,2]) y_max =
max(X[:100,2])

x_axis = np.linspace(x_min,x_max) y_axis =
np.linspace(y_min,y_max)

Y = -(weights[0]+weights[1]*x_axis)/weights[2] Y2 = -
(w0+w1*x_axis)/w2 plt.plot(x_axis,Y,c='black')
plt.plot(x_axis,Y2, c='r')
plt.ylim(y_min-1,y_max+1)
plt.legend(loc='best') plt.show()
```

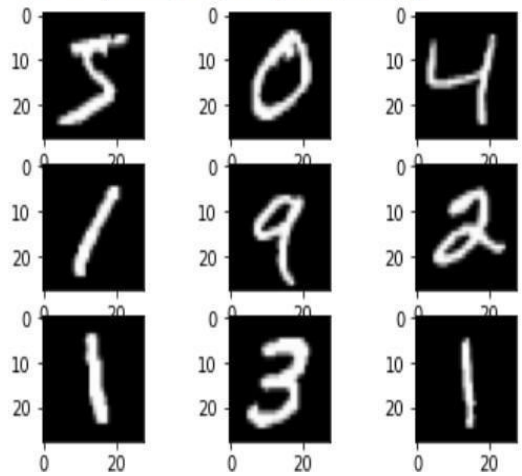
Output:



10. Python program to recognize handwritten digits using neural network

```
from keras . datasets import mnist
from matplotlib import pyplot
# load dataset
(trainX, trainy), (testX, testy) = mnist. load_data( )

# summarize loaded dataset
print ( 'Train: X=%s, y=%s' % (trainX. shape, trainy . shape) )
print ( 'Test: X=s, y=%%s' % (testX. shape, testy . shape) )
# plot first few images
for i in range(9) :
# define subplot
        pyplot. subplot (330 + 1 + i) # plot raw pixel data pyplot.
        imshow(trainX[i], cmap=pyplot . get_cmap( 'gray' ) )
# show the figure (on next page)
Pyplot.show()

# Setup train and test splits
(x_train, y_train), (x_test, y_test) = mnist. load_data()
print("Training data shape: ", x_train. shape) # (60000, 28, 28) -- 60000 images, each 28x28 pixels
print("Test data shape", x_test. shape) # (10000, 28, 28) -- 10000 images, each 28x28

# Flatten the images image_vector_size = 28*28
x_train = x_train. reshape(x_train. shape[0], image_vector_size)
x_test = x_test. reshape(x_test. shape[0], image_vector_size)
Training data shape: (60000, 28, 28)
Test data shape (10000, 28, 28)
from keras. layers import Dense # Dense layers are "fully connected" layers from keras. models
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11493376/11490434 [==============================] - 0s 0us/step
11501568/11490434 [==============================] - 0s 0us/step
Train: X=(60000, 28, 28), y=(60000,)
Test: X=(10000, 28, 28), y=(10000,)
```



import Sequential image_size = 784 # 28*28
num_classes = 10 # ten unique digits model = Sequential()

The input layer requires the special input_shape parameter which should match # the shape of our training data. # HERE THE HIDDEN LAYER HAS 500 NODES
model. add (Dense(units=500, activation='sigmoid', input_shape=(image_size,)))
model . add (Dense (units=num_classes, activation= 'softmax'))
model.summary()

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 500)               392500
_________________________________________________________________
dense_1 (Dense)              (None, 10)                5010
=================================================================
Total params: 397,510
Trainable params: 397,510
Non-trainable params: 0
_________________________________________________________________
```

from keras . layers import Dense
from keras. models import Sequential
image_size = 784 # 28*28
num_classes = 10 # ten unique digits
model = Sequential( )

HERE THE HIDDEN LAYER HAS 1000 NODES
model. add (Dense (units=1000, activation='sigmoid', input_shape=(image_size,)))
model . add (Dense(units=num_classes, activation=' softmax')) model . summary ( )

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_2 (Dense)              (None, 1000)              785000
_________________________________________________________________
dense_3 (Dense)              (None, 10)                10010
=================================================================
Total params: 795,010
Trainable params: 795,010
Non-trainable params: 0
_________________________________________________________________
```

from keras . layers import Dense
from keras. models import Sequential

image_size = 784 # 28*28
num_classes = 10 # ten unique digits

model = Sequential( )
# HERE THE HIDDEN LAYER HAS 20000 NODES model. add (Dense (units=2000, activation='sigmoid', input_shape=(image_size,
)))
model . add (Dense (units=num_classes, activation='softmax')) mode.summary()

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_4 (Dense)              (None, 2000)              1570000
_________________________________________________________________
dense_5 (Dense)              (None, 10)                20010
=================================================================
Total params: 1,590,010
Trainable params: 1,590,010
Non-trainable params: 0
_________________________________________________________________
```

11. Python program to study a bank credit data set and determine whether a transaction is fraudulent or not based on past data

```
#Packages related to general operating system & warnings
import os
import warnings warnings.filterwarnings('ignore')
#Packages related to data importing, manipulation, exploratory data analysis, data understanding
import numpy as np
import pandas as pd
from pandas import Series, DataFrame
from termcolor import colored as cl # text customization
#Packages related to data visualization
import seaborn as sns
import matplotlib.pyplot as plt
#Setting plot sizes and type of plot
plt.rc("font", size=14)
plt.rcParams['axes.grid'] = True
plt.figure(figsize=(6,3))
plt.gray()
from matplotlib.backends.backend_pdf import PdfPages
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn import metrics
from sklearn.impute import MissingIndicator, SimpleImputer
from sklearn.preprocessing import PolynomialFeatures, KBinsDiscretizer, F unctionTransformer
from sklearn.preprocessing import StandardScaler, MinMaxScaler, MaxAbsScal er
from sklearn.preprocessing import LabelEncoder, OneHotEncoder, LabelBinari zer, OrdinalEncoder
import statsmodels.formula.api as smf
import statsmodels.tsa as tsa
from sklearn.linear_model import LogisticRegression, LinearRegression, Ela sticNet, Lasso, Ridge
from sklearn.neighbors import KNeighborsClassifier, KNeighborsRegressor
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor, export_graphviz, export
from sklearn.ensemble import BaggingClassifier, BaggingRegressor, RandomForestClassifier,
RandomForestRegressor
from sklearn.ensemble import GradientBoostingClassifier,GradientBoostingRegressor,
AdaBoostClassifier, AdaBoostRegressor
from sklearn.svm import LinearSVC, LinearSVR, SVC, SVR
from xgboost import XGBClassifier
from sklearn.metrics import f1_score
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
data=pd.read_csv("creditcard.csv")
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V2
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928	0.12853
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846	0.16717
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281	-0.32764
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.175575	0.64737
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.141267	-0.20601
5	2.0	-0.425966	0.960523	1.141109	-0.168252	0.420987	-0.029728	0.476201	0.260314	-0.568671	...	-0.208254	-0.559825	-0.026398	-0.371427	-0.23279
6	4.0	1.229658	0.141004	0.045371	1.202613	0.191881	0.272708	-0.005159	0.081213	0.464960	...	-0.167716	-0.270710	-0.154104	-0.780055	0.75013
7	7.0	-0.644269	1.417964	1.074380	-0.492199	0.948934	0.428118	1.120631	-3.807864	0.615375	...	1.943465	-1.015455	0.057504	-0.649709	-0.41526
8	7.0	-0.894286	0.286157	-0.113192	-0.271526	2.669599	3.721818	0.370145	0.851084	-0.392048	...	-0.073425	-0.268092	-0.204233	1.011592	0.37320
9	9.0	-0.338262	1.119593	1.044367	-0.222187	0.499361	-0.246761	0.651583	0.069539	-0.736727	...	-0.246914	-0.633753	-0.120794	-0.385050	-0.06973

10 rows × 31 columns

```
Total_transactions = len(data)
normal = len(data[data.Class == 0])
fraudulent = len(data[data.Class == 1])
fraud_percentage = round(fraudulent/normal*100, 2)
print(cl('Total number of Transactions are {}'.format(Total_transactions), attrs = ['bold']))
print(cl('Number of Normal Transactions are {}'.format(normal), attrs = [' bold']))
print(cl('Number of fraudulent Transactions are {}'.format(fraudulent), at trs = ['bold']))
print(cl('Percentage of fraud Transactions is {}'.format(fraud_percentage), attrs = ['bold']))
```

```
Total number of Trnsactions are 284807
Number of Normal Transactions are 284315
Number of fraudulent Transactions are 492
Percentage of fraud Transactions is 0.17
```

```
#Only 0.17% of transactions are fraudulent.
data.info()
```

```
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   Time    284807 non-null  float64
 1   V1      284807 non-null  float64
 2   V2      284807 non-null  float64
 3   V3      284807 non-null  float64
 4   V4      284807 non-null  float64
 5   V5      284807 non-null  float64
 6   V6      284807 non-null  float64
 7   V7      284807 non-null  float64
 8   V8      284807 non-null  float64
 9   V9      284807 non-null  float64
 10  V10     284807 non-null  float64
 11  V11     284807 non-null  float64
 12  V12     284807 non-null  float64
 13  V13     284807 non-null  float64
 14  V14     284807 non-null  float64
 15  V15     284807 non-null  float64
 16  V16     284807 non-null  float64
 17  V17     284807 non-null  float64
 18  V18     284807 non-null  float64
 19  V19     284807 non-null  float64
 20  V20     284807 non-null  float64
 21  V21     284807 non-null  float64
 22  V22     284807 non-null  float64
 23  V23     284807 non-null  float64
 24  V24     284807 non-null  float64
 25  V25     284807 non-null  float64
 26  V26     284807 non-null  float64
 27  V27     284807 non-null  float64
 28  V28     284807 non-null  float64
 29  Amount  284807 non-null  float64
 30  Class   284807 non-null  int64  
dtypes: float64(30), int64(1)
```

```
sc = StandardScaler()
amount = data['Amount'].values
data['Amount'] = sc.fit_transform(amount.reshape(-1, 1))
data.drop(['Time'], axis=1, inplace=True)
```

```
data.shape
(284807, 30)
```

```
data.drop_duplicates(inplace=True)
```

```
data.shape
(275663, 30)
```

```
#We were having around ~9000 duplicate transactions
X = data.drop('Class', axis = 1).values y = data['Class'].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25 , random_state = 1)
#We now have two different data set — Train data we will be used for train ing our model and the data which is unseen will be used for testing.
```

Decision Tree

```
DT = DecisionTreeClassifier(max_depth = 4, criterion = 'entropy')
DT.fit(X_train, y_train)
dt_yhat = DT.predict(X_test)
print('Accuracy score of the Decision Tree model is {}'.format(accuracy_score(y_test, tree_yhat)))
```

Accuracy score of the Decision Tree model is 0.999288989494457

```
print('F1 score of the Decision Tree model is {}'.format(f1_score(y_test, tree_yhat)))
```

F1 score of the Decision Tree model is 0.776255707762557

```
confusion_matrix(y_test, tree_yhat, labels = [0, 1])
```

```
array([[68782,    18],
       [   31,    85]], dtype=int64)
```

K-Nearest Neighbors

```
n = 7
KNN = KNeighborsClassifier(n_neighbors = n)
KNN.fit(X_train, y_train)
knn_yhat = KNN.predict(X_test)
print('Accuracy score of the K-Nearest Neighbors model is {}'.format(accuracy_score(y_test, knn_yhat)))
```

Accuracy score of the K-Nearest Neighbors model is 0.999506645771664

```
print('F1 score of the K-Nearest Neighbors model is {}'.format(f1_score(y_test, knn_yhat)))
```

F1 score of the K-Nearest Neighbors model is 0.8365384615384616

```
#We just received 99.95% accuracy in our credit card fraud detection.
```

12. Python program for implementation of CNN

```
import tensorflow as tf
from keras.preprocessing.image import ImageDataGenerator
from keras.preprocessing import image import numpy as np

train_datagen = ImageDataGenerator(rescale=1./255, shear_range=0.2, zoom_range=0.2,
horizontal_flip=True)
train_set = train_datagen.flow_from_directory('training_set', target_size=(64, 64),
batch_size=32, class_mode='binary')
test_datagen = ImageDataGenerator(rescale=1./255)
test_set = test_datagen.flow_from_directory('test_set', target_size=(64, 64), batch_size=32,
class_mode='binary')
print(test_set)

cnn = tf.keras.models.Sequential()

cnn.add(tf.keras.layers.Conv2D(filters = 32, kernel_size = 3, activation = 'relu',
    input_shape=[64,64,3]))
cnn.add(tf.keras.layers.MaxPool2D(pool_size=2 ,strides=2)
cnn.add(tf.keras.layers.Conv2D(filters = 32, kernel_size = 3, activation = 'relu'))
cnn.add(tf.keras.layers.MaxPool2D(pool_size=2 ,strides=2))
cnn.add(tf.keras.layers.Flatten())
cnn.add(tf.keras.layers.Dense(units = 128, activation = 'relu'))
cnn.add(tf.keras.layers.Dense(units = 1, activation = 'sigmoid'))

cnn.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
cnn.fit(x = train_set, validation_data = test_set, epochs = 25)

test_image = image.load_img('single_prediction/cat_or_dog_1.jpg', target_size = (64, 64))
test_image = image.img_to_array(test_image)
test_image = np.expand_dims(test_image, axis = 0)
result = cnn.predict(test_image)
print(train_set.class_indices)
if result[0][0] == 1:
    prediction = 'dog'
else:
    prediction = 'cat'
print(prediction)

test_image2 = image.load_img('single_prediction/cat_or_dog_2.jpg', target_size = (64, 64))
test_image2 = image.img_to_array(test_image2)
test_image2 = np.expand_dims(test_image2, axis = 0)
result2 = cnn.predict(test_image2)
if result2[0][0] == 1:
    prediction2 = 'dog'
else:
    prediction2 = 'cat'
 print(prediction2)
```

Output:

```
251/251 [==============================] - 36s 142ms/step - loss: 0.3630 - accuracy: 0.8363 - val_loss: 0.4835 - val_accuracy: 0.783
Epoch 15/25
251/251 [==============================] - 36s 145ms/step - loss: 0.3493 - accuracy: 0.8390 - val_loss: 0.4801 - val_accuracy: 0.797
Epoch 16/25
251/251 [==============================] - 37s 146ms/step - loss: 0.3539 - accuracy: 0.8403 - val_loss: 0.4794 - val_accuracy: 0.800
Epoch 17/25
251/251 [==============================] - 36s 144ms/step - loss: 0.3329 - accuracy: 0.8550 - val_loss: 0.4808 - val_accuracy: 0.797
Epoch 18/25
251/251 [==============================] - 35s 138ms/step - loss: 0.3165 - accuracy: 0.8594 - val_loss: 0.5069 - val_accuracy: 0.808
Epoch 19/25
251/251 [==============================] - 34s 137ms/step - loss: 0.3128 - accuracy: 0.8688 - val_loss: 0.5349 - val_accuracy: 0.793
Epoch 20/25
251/251 [==============================] - 34s 137ms/step - loss: 0.2932 - accuracy: 0.8676 - val_loss: 0.4962 - val_accuracy: 0.790
Epoch 21/25
251/251 [==============================] - 34s 137ms/step - loss: 0.2658 - accuracy: 0.8876 - val_loss: 0.5106 - val_accuracy: 0.796
Epoch 22/25
251/251 [==============================] - 34s 137ms/step - loss: 0.2755 - accuracy: 0.8781 - val_loss: 0.5209 - val_accuracy: 0.798
Epoch 23/25
251/251 [==============================] - 35s 138ms/step - loss: 0.2416 - accuracy: 0.8989 - val_loss: 0.5025 - val_accuracy: 0.812
Epoch 24/25
251/251 [==============================] - 37s 149ms/step - loss: 0.2381 - accuracy: 0.8989 - val_loss: 0.5604 - val_accuracy: 0.787
Epoch 25/25
251/251 [==============================] - 37s 147ms/step - loss: 0.2371 - accuracy: 0.9056 - val_loss: 0.5098 - val_accuracy: 0.799
{'cats': 0, 'dogs': 1}
dog
cat
```

13. Python program for implementation of RNN

```
import logging
import pandas as pd
import numpy as np
from numpy import random
import gensim
import nltk
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.metrics import accuracy_score, confusion_matrix
import matplotlib.pyplot as plt
from nltk.corpus import stopwords
import re
from bs4 import BeautifulSoup
import pandas as pd
import os
import re
import spacy
from gensim.models.phrases import Phrases, Phraser
from time import time
import multiprocessing
from gensim.models import Word2Vec
import bokeh.plotting as bp
from bokeh.models import HoverTool, BoxSelectTool
from bokeh.plotting import figure, show, output_notebook
from sklearn.manifold import TSNE
from sklearn.model_selection import train_test_split
import numpy as np
from sklearn.preprocessing import scale
```

```
import keras
from keras.models import Sequential, Model
from keras import layers
from keras.layers import Dense, Conv1D, MaxPooling1D, Flatten, Dropout, Input, Embedding
from keras.layers.merge import Concatenate
from sklearn.feature_extraction.text import TfidfVectorizer
from wordcloud import WordCloud
from nltk.tokenize import RegexpTokenizer
from sklearn.metrics import confusion_matrix

X=df[['text']]
y=df[['target']]
description_list = df['text'].tolist()
text=np.array(df['target'].tolist())

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
import pickle
count_vect = CountVectorizer()
#count_vect._validate_vocabulary()
x_train_counts = count_vect.fit_transform(description_list)
tfidf_transformer = TfidfTransformer()
x_train_tfidf = tfidf_transformer.fit_transform(x_train_counts)
# Save the vectorizer
vec_file = 'vectorizer.pickle'
pickle.dump(count_vect, open(vec_file, 'wb'))

# Save the model
# mod_file = 'classification.model'
# pickle.dump(model, open(mod_file, 'wb'))

#building a simple RNN model
model = Sequential()
model.add(keras.layers.InputLayer(input_shape=(15,1)))
keras.layers.embeddings.Embedding(nb_words, 15, weights=[embedding_matrix], nput_length=15,
trainable=False)
model.add(keras.layers.recurrent.SimpleRNN(units = 100, activation='relu', use_bias=True))
model.add(keras.layers.Dense(units=1000, input_dim = 2000, activation='sigmoid'))
model.add(keras.layers.Dense(units=500, input_dim=1000, activation='relu'))
model.add(keras.layers.Dense(units=2, input_dim=500,activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

14. Python program for implementation of LSTM

Dataset-- https://www.kaggle.com/c/sentiment-analysis-on-movie-reviews

```
from google.colab import drive
drive.mount('/content/drive') #For linux platform only
from tensorflow.keras.preprocessing import text_dataset_from_directory
```

```
# Change the relative paths below.
train_data = text_dataset_from_directory("/content/My Drive/moviereviews-
dataset/train")
test_data = text_dataset_from_directory("/content/drive/My Drive/moviereviews-
dataset/test")

from tensorflow.keras.preprocessing import text_dataset_from_directory
from tensorflow.strings import regex_replace
def prepareData(dir):
data = text_dataset_from_directory(dir)
return data.map(
lambda text, label: (regex_replace(text, '<br />', ' '), label),
)
train_data = prepareData('/content/drive/My Drive/movie-reviewsdataset/
train')
test_data = prepareData('/content/drive/My Drive/movie-reviewsdataset/
test')
-
for text_batch, label_batch in train_data.take(1):
	print(text_batch.numpy()[0])
	print(label_batch.numpy()[0]) # 0 = negative, 1 = positive

from tensorflow.keras.models import Sequential
from tensorflow.keras import Input
model = Sequential()
model.add(Input(shape=(1,), dtype="string"))

from tensorflow.keras.layers.experimental.preprocessing import TextVectorization
max_tokens = 1000
max_len = 100
vectorize_layer = TextVectorization(max_tokens=max_tokens, output_mode="int",
output_sequence_length=max_len)

train_texts = train_data.map(lambda text, label: text)
vectorize_layer.adapt(train_texts)
model.add(vectorize_layer)

from tensorflow.keras.layers import Embedding
max_tokens = 1000
#model.add(vectorize_layer)

model.add(Embedding(max_tokens + 1, 128))
from tensorflow.keras.layers import LSTM
model.add(LSTM(64))
from tensorflow.keras.layers import Dense
model.add(Dense(64, activation="relu"))
model.add(Dense(1, activation="sigmoid"))
```

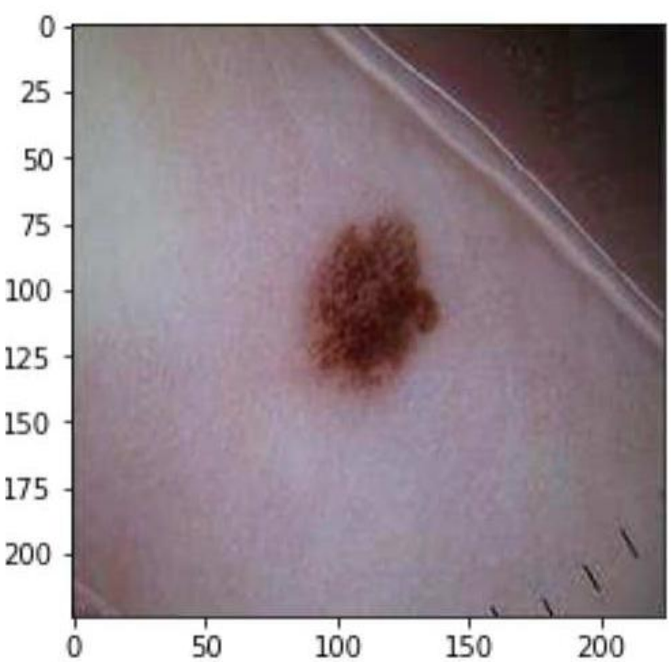
```
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.fit(train_data, epochs=10)

print(model.predict(["i loved it! highly recommend it to anyone and everyone looking for a great
movie to watch.",]))
# Should print a very low score like 0.01.
print(model.predict(["this was awful! i hated it so much, nobody should watch this. the actin
g was terrible, the music was terrible, overall it was just bad.",]))
# Expected output—1 0
```

Value-Added Experiment-1: Image Classification using CNN

Dataset -- https://www.kaggle.com/fanconic/skin-cancer-malignant-vs-benign

```
from mpl_toolkits.mplot3d import Axes3D
from sklearn.preprocessing import StandardScaler
import os
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import os
from glob import glob
import seaborn as sns
from PIL import Image
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

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

benign_train = '/content/drive/My Drive/archive/data/train/benign'
malignant_train = '/content/drive/My Drive/archive/data/train/malignant'
benign_test = '/content/drive/My Drive/archive/data/test/benign'
malignant_test = '/content/drive/My Drive/archive/data/test/malignant'
-
read = lambda imname: np.asarray(Image.open(imname).convert("RGB"))
# Load in training pictures
ims_benign = [read(os.path.join(benign_train, filename)) for filename in os.listdir(benign_train)]
X_benign = np.array(ims_benign, dtype='uint8')
ims_malignant = [read(os.path.join(malignant_train, filename)) for filename in os.listdir(malignant_train)]
X_malignant = np.array(ims_malignant, dtype='uint8')
ims_benign = [read(os.path.join(benign_test, filename)) for filename in os.listdir(be
nign_test)]
X_benign_test = np.array(ims_benign, dtype='uint8')
ims_malignant = [read(os.path.join(malignant_test, filename)) for filename in os.list
dir(malignant_test)]
X_malignant_test = np.array(ims_malignant, dtype='uint8')
```

```
# Create labels
y_benign = np.zeros(X_benign.shape[0])
y_malignant = np.ones(X_malignant.shape[0])
y_benign_test = np.zeros(X_benign_test.shape[0])
y_malignant_test = np.ones(X_malignant_test.shape[0])

# Merge data
X_train = np.concatenate((X_benign, X_malignant), axis = 0)
y_train = np.concatenate((y_benign, y_malignant), axis = 0)
X_test = np.concatenate((X_benign_test, X_malignant_test), axis = 0)
y_test = np.concatenate((y_benign_test, y_malignant_test), axis = 0)

s = np.arange(X_train.shape[0])
np.random.shuffle(s)
X_train = X_train[s]
y_train = y_train[s]
s = np.arange(X_test.shape[0])
np.random.shuffle(s)
X_test = X_test[s]
y_test = y_test[s]

plt.imshow(X_test[1], interpolation='nearest')
plt.show()
```




```
X_train = X_train/255
X_test = X_test/255
import tensorflow as tf
X_Train = tf.keras.utils.normalize(X_train)
X_Test = tf.keras.utils.normalize(X_test)
```

```
-
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Conv2D(128,(3,3), input_shape = X_Train.shape[1:] ,activation =
tf.nn.relu ))
model.add(tf.keras.layers.MaxPool2D(pool_size=(3,3),strides=None))
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(64,activation=tf.nn.relu))
model.add(tf.keras.layers.Dropout(0.3))
model.add(tf.keras.layers.Dense(32,activation=tf.nn.relu))
model.add(tf.keras.layers.Dropout(0.25))
model.add(tf.keras.layers.Dense(2,activation=tf.nn.softmax))
from keras.callbacks import ModelCheckpoint,EarlyStopping,ReduceLROnPlateau
from keras.layers import Dense, Dropout, Activation, Flatten
model.compile(optimizer="adam",loss="sparse_categorical_crossentropy",metrics=["accuracy"])
earlystop=EarlyStopping(monitor='val_loss',min_delta=0, patience=5, verbose=1,
restore_best_weights=True)
callbacks=[earlystop]
model.fit(X_Train, y_train, epochs = 50,callbacks=callbacks, shuffle=True,batch_size=50,
validation_split = 0.1)

y_pred = model.predict(X_Test)
yp =[]
for i in range(0,660):
	if y_pred[i][0] >= 0.5:
		yp.append(0)
	else:
		yp.append(1)
print(accuracy_score(y_test, yp))
```

Value-Added Experiment-2: Python program for implementation of Bidirectional LSTM

```
import numpy as np
from keras.preprocessing import sequence
from keras.models import Sequential
from keras.layers import Dense, Dropout, Embedding, LSTM, Bidirectional
from keras.datasets import imdb

# Using the IMDB data set for text classification using keras and bi-LSTM network
n_unique_words = 10000 # cut texts after this number of words
maxlen = 200
batch_size = 128

# Loading the data set from the Keras library.
(x_train, y_train),(x_test, y_test) = imdb.load_data(num_words=n_unique_words)

# To fit the data into any neural network, we need to convert the data into sequence matrices.
# For this, we are using the pad_sequence module from keras.preprocessing.
x_train = sequence.pad_sequences(x_train, maxlen=maxlen)
```

```
x_test = sequence.pad_sequences(x_test, maxlen=maxlen)
y_train = np.array(y_train)
y_test = np.array(y_test)
```

Output:

```
[   5   25  100   43  838  112   50  670    2    9   35  480  284    5
  150    4  172  112  167    2  336  385   39    4  172 4536 1111   17
  546   38   13  447    4  192   50   16    6  147 2025   19   14   22
    4 1920 4613  469    4   22   71   87   12   16   43  530   38   76
   15   13 1247    4   22   17  515   17   12   16  626   18    2    5
   62  386   12    8  316    8  106    5    4 2223 5244   16  480   66
 3785   33    4  130   12   16   38  619    5   25  124   51   36  135
   48   25 1415   33    6   22   12  215   28   77   52    5   14  407
   16   82    2    8    4  107  117 5952   15  256    4    2    7 3766
    5  723   36   71   43  530  476   26  400  317   46    7    4    2
 1029   13  104   88    4  381   15  297   98   32 2071   56   26  141
    6  194 7486   18    4  226   22   21  134  476   26  480    5  144
   30 5535   18   51   36   28  224   92   25  104    4  226   65   16
   38 1334   88   12   16  283    5   16 4472  113  103   32   15   16
 5345   19  178   32]
```

```
# Making a model with bi-LSTM layer.
model = Sequential()
model.add(Embedding(n_unique_words, 128, input_length=maxlen))
model.add(Bidirectional(LSTM(64)))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
# Training model with training data set with 12 epochs.
history=model.fit(x_train, y_train, batch_size=batch_size, epochs=12, validation_data=[x_test,
y_test])
print(history.history['loss'])
print(history.history['accuracy'])
```

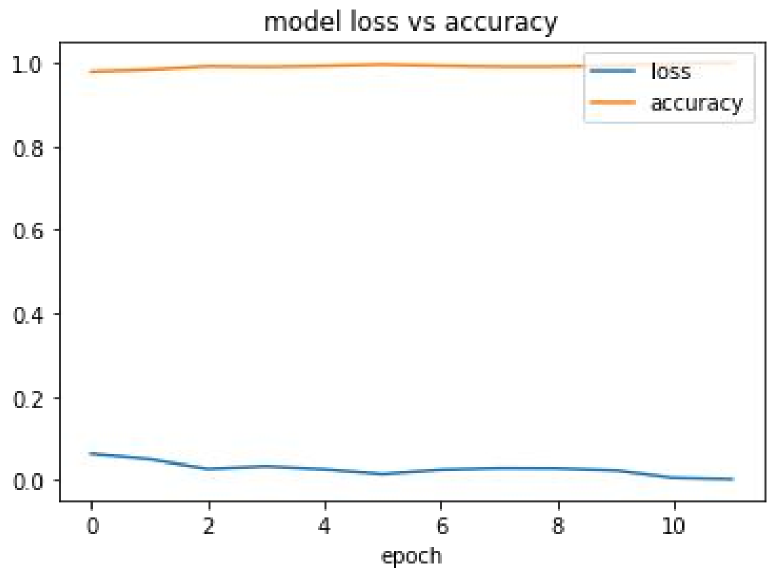
Output:

```
Epoch 1/12
196/196 [==============================] - 148s 753ms/step - loss: 0.0638 - accuracy: 0.9787 - val_loss: 0.0000e+00 - val_accuracy: 0.0000e+00
Epoch 2/12
196/196 [==============================] - 146s 745ms/step - loss: 0.0509 - accuracy: 0.9835 - val_loss: 0.0000e+00 - val_accuracy: 0.0000e+00
Epoch 3/12
196/196 [==============================] - 147s 750ms/step - loss: 0.0274 - accuracy: 0.9917 - val_loss: 0.0000e+00 - val_accuracy: 0.0000e+00
Epoch 4/12
196/196 [==============================] - 147s 749ms/step - loss: 0.0340 - accuracy: 0.9900 - val_loss: 0.0000e+00 - val_accuracy: 0.0000e+00
Epoch 5/12
196/196 [==============================] - 148s 753ms/step - loss: 0.0266 - accuracy: 0.9926 - val_loss: 0.0000e+00 - val_accuracy: 0.0000e+00
Epoch 6/12
196/196 [==============================] - 147s 749ms/step - loss: 0.0151 - accuracy: 0.9960 - val_loss: 0.0000e+00 - val_accuracy: 0.0000e+00
Epoch 7/12
196/196 [==============================] - 147s 752ms/step - loss: 0.0257 - accuracy: 0.9930 - val_loss: 0.0000e+00 - val_accuracy: 0.0000e+00
Epoch 8/12
196/196 [==============================] - 147s 751ms/step - loss: 0.0294 - accuracy: 0.9908 - val_loss: 0.0000e+00 - val_accuracy: 0.0000e+00
Epoch 9/12
196/196 [==============================] - 148s 755ms/step - loss: 0.0289 - accuracy: 0.9908 - val_loss: 0.0000e+00 - val_accuracy: 0.0000e+00
Epoch 10/12
196/196 [==============================] - 148s 755ms/step - loss: 0.0240 - accuracy: 0.9936 - val_loss: 0.0000e+00 - val_accuracy: 0.0000e+00
Epoch 11/12
196/196 [==============================] - 148s 755ms/step - loss: 0.0061 - accuracy: 0.9988 - val_loss: 0.0000e+00 - val_accuracy: 0.0000e+00
Epoch 12/12
196/196 [==============================] - 148s 756ms/step - loss: 0.0028 - accuracy: 0.9996 - val_loss: 0.0000e+00 - val_accuracy: 0.0000e+00
[0.06376810371875763, 0.0509498231112957, 0.027414456009864807, 0.03396657481789589, 0.02660427801311016, 0.01507935207337141, 0.02565259113907814, 0.029360182583332062, 0.028
[0.9787200093269348, 0.983519971370697, 0.9917200207710266, 0.9900400042533875, 0.9926400184631348, 0.9960399866104126, 0.9929599761962891, 0.9908400177955627, 0.9908400177955
```

Image is not clearer because the number of content in one place is high
Using plots to know the model's performance.

```
from matplotlib import pyplot
pyplot.plot(history.history['loss'])
pyplot.plot(history.history['accuracy'])
pyplot.title('model loss vs accuracy')
pyplot.xlabel('epoch')
pyplot.legend(['loss', 'accuracy'], loc='upper right')
pyplot.show()
```

Output:



Value-Added Experiment-3: Python program for implementation of Bidirectional RNN

Python code for designing a Bidirectional Recurrent Neural Network (BRNN) in TensorFlow for classifying MNIST digits.

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.contrib import rnn
```

```python
# Data Dimension
num_input = 28          # MNIST data input (image shape: 28x28)
timesteps = 28          # Timesteps
n_classes = 10          # Number of classes, one class per digit


def load_data(mode='train'): # Helper function to load the MNIST data
    """
    Function to (download and) load the MNIST data
    :param mode: train or test
    :return: images and the corresponding labels
    """
    from tensorflow.examples.tutorials.mnist import input_data
    mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
    if mode == 'train':
        x_train, y_train, x_valid, y_valid = mnist.train.images, mnist.train.labels, \
                                             mnist.validation.images, mnist.validation.labels
        return x_train, y_train, x_valid, y_valid
    elif mode == 'test':
        x_test, y_test = mnist.test.images, mnist.test.labels
        return x_test, y_test

def randomize(x, y):  # Helper function to load the MNIST data
    """ Randomizes the order of data samples and their corresponding labels"""
    permutation = np.random.permutation(y.shape[0])
    shuffled_x = x[permutation, :]
    shuffled_y = y[permutation]
    return shuffled_x, shuffled_y

def get_next_batch(x, y, start, end):  # Helper function to load the MNIST data
    x_batch = x[start:end]
    y_batch = y[start:end]
    return x_batch, y_batch
```

```python
x_train, y_train, x_valid, y_valid = load_data(mode='train')
print("Size of:")
print("- Training-set:\t\t{}".format(len(y_train)))
print("- Validation-set:\t{}".format(len(y_valid)))
```

Output:

```
Extracting MNIST_data/train-images-idx3-ubyte.gz
Extracting MNIST_data/train-labels-idx1-ubyte.gz
Extracting MNIST_data/t10k-images-idx3-ubyte.gz
Extracting MNIST_data/t10k-labels-idx1-ubyte.gz
Size of:
- Training-set:		55000
- Validation-set:	5000
```

Program Continued:

```python
learning_rate = 0.001 # The optimization initial learning rate
epochs = 10           # Total number of training epochs
batch_size = 100      # Training batch size
display_freq = 100    # Frequency of displaying the training results
num_hidden_units = 128  # Number of hidden units of the RNN
```

```python
# weight and bais wrappers
def weight_variable(shape):
    """
    Create a weight variable with appropriate initialization
    :param name: weight name
    :param shape: weight shape
    :return: initialized weight variable
    """
    initer = tf.truncated_normal_initializer(stddev=0.01)
    return tf.get_variable('W', dtype=tf.float32, shape=shape, initializer=initer)

def bias_variable(shape):
    """
    Create a bias variable with appropriate initialization
    :param name: bias variable name
    :param shape: bias variable shape
    :return: initialized bias variable
    """
    initial = tf.constant(0., shape=shape, dtype=tf.float32)
    return tf.get_variable('b', dtype=tf.float32, initializer=initial)
```

```python
def BiRNN(x, weights, biases, timesteps, num_hidden): # Helper function for creating Bi-RNN
    # Prepare data shape to match `rnn` function requirements
    # Current data input shape: (batch_size, timesteps, n_input)
    # Required shape: 'timesteps' tensors list of shape (batch_size, num_input)

    # Unstack to get a list of 'timesteps' tensors of shape (batch_size, num_input)
    x = tf.unstack(x, timesteps, 1)

    # Define lstm cells with tensorflow
    # Forward direction cell
    lstm_fw_cell = rnn.BasicLSTMCell(num_hidden, forget_bias=1.0)
    # Backward direction cell
    lstm_bw_cell = rnn.BasicLSTMCell(num_hidden, forget_bias=1.0)

    # Get BiRNN cell output
    outputs, _, _ = rnn.static_bidirectional_rnn(lstm_fw_cell, lstm_bw_cell, x, dtype=tf.float32)

    # Linear activation, using rnn inner loop last output
    return tf.matmul(outputs[-1], weights) + biases
```

```
# Create the network graph
# Placeholders for inputs (x) and outputs(y)
x = tf.placeholder(tf.float32, shape=[None, timesteps, num_input], name='X')
y = tf.placeholder(tf.float32, shape=[None, n_classes], name='Y')

# create weight matrix initialized randomely from N~(0, 0.01)
W = weight_variable(shape=[2*num_hidden_units, n_classes])

# create bias vector initialized as zero
b = bias_variable(shape=[n_classes])
output_logits = BiRNN(x, W, b, timesteps, num_hidden_units)
y_pred = tf.nn.softmax(output_logits)

# Model predictions
cls_prediction = tf.argmax(output_logits, axis=1, name='predictions')

# Define the loss function, optimizer, and accuracy
loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(labels=y, logits=output_logits), na
me='loss')
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate, name='Adam-op').minimize(loss)
correct_prediction = tf.equal(tf.argmax(output_logits, 1), tf.argmax(y, 1), name='correct_pred')
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32), name='accuracy')

# Creating the op for initializing all variables
init = tf.global_variables_initializer()

# Training the RNN model
sess = tf.InteractiveSession()
sess.run(init)
global_step = 0
# Number of training iterations in each epoch
num_tr_iter = int(len(y_train) / batch_size)
for epoch in range(epochs):
    print('Training epoch: {}'.format(epoch + 1))
    x_train, y_train = randomize(x_train, y_train)
    for iteration in range(num_tr_iter):
        global_step += 1
        start = iteration * batch_size
        end = (iteration + 1) * batch_size
        x_batch, y_batch = get_next_batch(x_train, y_train, start, end)
        x_batch = x_batch.reshape((batch_size, timesteps, num_input))
        # Run optimization op (backprop)
        feed_dict_batch = {x: x_batch, y: y_batch}
        sess.run(optimizer, feed_dict=feed_dict_batch)

        if iteration % display_freq == 0:
            # Calculate and display the batch loss and accuracy
            loss_batch, acc_batch = sess.run([loss, accuracy], feed_dict=feed_dict_batch)
```

```
        print("iter {0:3d}:\t Loss={1:.2f},\tTraining Accuracy={2:.01%}".
              format(iteration, loss_batch, acc_batch))

    # Run validation after every epoch
    feed_dict_valid = {x: x_valid[:1000].reshape((-1, timesteps, num_input)), y: y_valid[:1000]}
    loss_valid, acc_valid = sess.run([loss, accuracy], feed_dict=feed_dict_valid)
    print('---------------------------------------------------------')
    print("Epoch: {0}, validation loss: {1:.2f}, validation accuracy: {2:.01%}".
          format(epoch + 1, loss_valid, acc_valid))
    print('---------------------------------------------------------')
```

Output:

```
Training epoch: 1
iter   0:	 Loss=2.30,	Training Accuracy=24.0%
iter 100:	 Loss=0.74,	Training Accuracy=77.0%
iter 200:	 Loss=0.51,	Training Accuracy=88.0%
iter 300:	 Loss=0.51,	Training Accuracy=83.0%
iter 400:	 Loss=0.30,	Training Accuracy=88.0%
iter 500:	 Loss=0.21,	Training Accuracy=94.0%
---------------------------------------------------------
Epoch: 1, validation loss: 0.23, validation accuracy: 93.5%
---------------------------------------------------------
Training epoch: 2
iter   0:	 Loss=0.22,	Training Accuracy=93.0%
iter 100:	 Loss=0.22,	Training Accuracy=94.0%
iter 200:	 Loss=0.15,	Training Accuracy=96.0%
iter 300:	 Loss=0.13,	Training Accuracy=96.0%
iter 400:	 Loss=0.21,	Training Accuracy=96.0%
iter 500:	 Loss=0.12,	Training Accuracy=97.0%
---------------------------------------------------------
Epoch: 2, validation loss: 0.14, validation accuracy: 96.0%
---------------------------------------------------------
Training epoch: 3
iter   0:	 Loss=0.10,	Training Accuracy=97.0%
iter 100:	 Loss=0.05,	Training Accuracy=99.0%
iter 200:	 Loss=0.10,	Training Accuracy=98.0%
iter 300:	 Loss=0.08,	Training Accuracy=98.0%
iter 400:	 Loss=0.20,	Training Accuracy=93.0%
iter 500:	 Loss=0.06,	Training Accuracy=99.0%
---------------------------------------------------------
Epoch: 3, validation loss: 0.10, validation accuracy: 97.0%
---------------------------------------------------------
Training epoch: 4
iter   0:	 Loss=0.05,	Training Accuracy=99.0%
iter 100:	 Loss=0.06,	Training Accuracy=99.0%
iter 200:	 Loss=0.07,	Training Accuracy=99.0%
iter 300:	 Loss=0.14,	Training Accuracy=97.0%
```

```
iter 400:     Loss=0.17,    Training Accuracy=97.0%
iter 500:     Loss=0.09,    Training Accuracy=99.0%
---------------------------------------------------------
Epoch: 4, validation loss: 0.09, validation accuracy: 97.6%
---------------------------------------------------------
Training epoch: 5
iter   0:     Loss=0.05,    Training Accuracy=99.0%
iter 100:     Loss=0.09,    Training Accuracy=98.0%
iter 200:     Loss=0.19,    Training Accuracy=92.0%
iter 300:     Loss=0.01,    Training Accuracy=100.0%
iter 400:     Loss=0.05,    Training Accuracy=98.0%
iter 500:     Loss=0.06,    Training Accuracy=98.0%
---------------------------------------------------------
Epoch: 5, validation loss: 0.08, validation accuracy: 97.7%
---------------------------------------------------------
Training epoch: 6
iter   0:     Loss=0.07,    Training Accuracy=98.0%
iter 100:     Loss=0.03,    Training Accuracy=99.0%
iter 200:     Loss=0.04,    Training Accuracy=98.0%
iter 300:     Loss=0.06,    Training Accuracy=99.0%
iter 400:     Loss=0.02,    Training Accuracy=99.0%
iter 500:     Loss=0.06,    Training Accuracy=97.0%
---------------------------------------------------------
Epoch: 6, validation loss: 0.08, validation accuracy: 97.8%
---------------------------------------------------------
Training epoch: 7
iter   0:     Loss=0.04,    Training Accuracy=99.0%
iter 100:     Loss=0.02,    Training Accuracy=100.0%
iter 200:     Loss=0.04,    Training Accuracy=99.0%
iter 300:     Loss=0.04,    Training Accuracy=99.0%
iter 400:     Loss=0.03,    Training Accuracy=99.0%
iter 500:     Loss=0.06,    Training Accuracy=97.0%
---------------------------------------------------------
Epoch: 7, validation loss: 0.09, validation accuracy: 97.8%
---------------------------------------------------------
Training epoch: 8
iter   0:     Loss=0.07,    Training Accuracy=99.0%
iter 100:     Loss=0.15,    Training Accuracy=98.0%
iter 200:     Loss=0.11,    Training Accuracy=99.0%
iter 300:     Loss=0.06,    Training Accuracy=99.0%
iter 400:     Loss=0.02,    Training Accuracy=100.0%
iter 500:     Loss=0.01,    Training Accuracy=99.0%
---------------------------------------------------------
Epoch: 8, validation loss: 0.06, validation accuracy: 98.5%
---------------------------------------------------------
Training epoch: 9
iter   0:     Loss=0.02,    Training Accuracy=100.0%
```

```
iter 100:     Loss=0.02,    Training Accuracy=100.0%
iter 200:     Loss=0.03,    Training Accuracy=99.0%
iter 300:     Loss=0.02,    Training Accuracy=99.0%
iter 400:     Loss=0.02,    Training Accuracy=100.0%
iter 500:     Loss=0.01,    Training Accuracy=100.0%
---------------------------------------------------------
Epoch: 9, validation loss: 0.06, validation accuracy: 98.4%
---------------------------------------------------------
Training epoch: 10
iter   0:     Loss=0.11,    Training Accuracy=98.0%
iter 100:     Loss=0.02,    Training Accuracy=99.0%
iter 200:     Loss=0.02,    Training Accuracy=99.0%
iter 300:     Loss=0.02,    Training Accuracy=99.0%
iter 400:     Loss=0.05,    Training Accuracy=99.0%
iter 500:     Loss=0.01,    Training Accuracy=100.0%
---------------------------------------------------------
Epoch: 10, validation loss: 0.07, validation accuracy: 97.7%
---------------------------------------------------------
```

Program Continued:

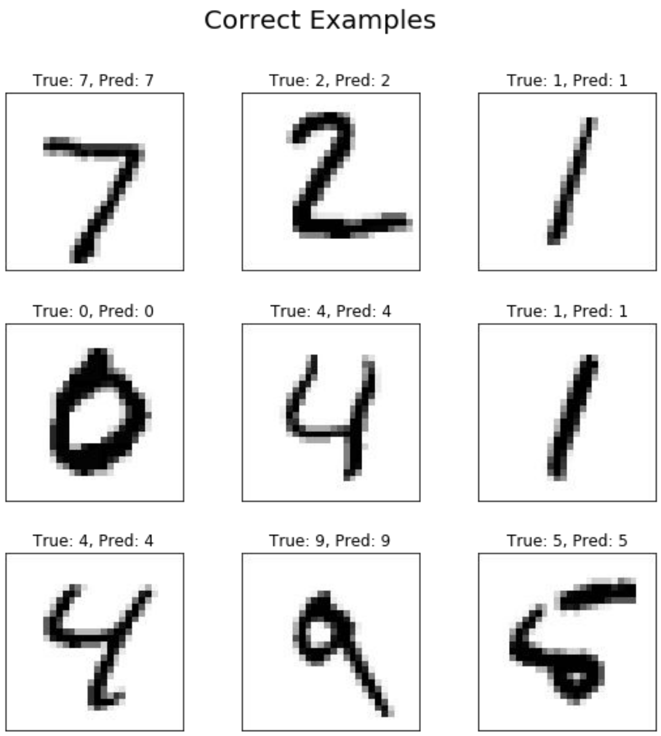
```
#Testing and Plotting the results
def plot_images(images, cls_true, cls_pred=None, title=None):
    """
    Create figure with 3x3 sub-plots.
    :param images: array of images to be plotted, (9, img_h*img_w)
    :param cls_true: corresponding true labels (9,)
    :param cls_pred: corresponding true labels (9,)
    """
    fig, axes = plt.subplots(3, 3, figsize=(9, 9))
    fig.subplots_adjust(hspace=0.3, wspace=0.3)
    for i, ax in enumerate(axes.flat):
        # Plot image.
        ax.imshow(np.squeeze(images[i]).reshape(28, 28), cmap='binary')

        # Show true and predicted classes.
        if cls_pred is None:
            ax_title = "True: {0}".format(cls_true[i])
        else:
            ax_title = "True: {0}, Pred: {1}".format(cls_true[i], cls_pred[i])

        ax.set_title(ax_title)
        # Remove ticks from the plot.
        ax.set_xticks([])
        ax.set_yticks([])

    if title:
        plt.suptitle(title, size=20)
    plt.show(block=False)
```

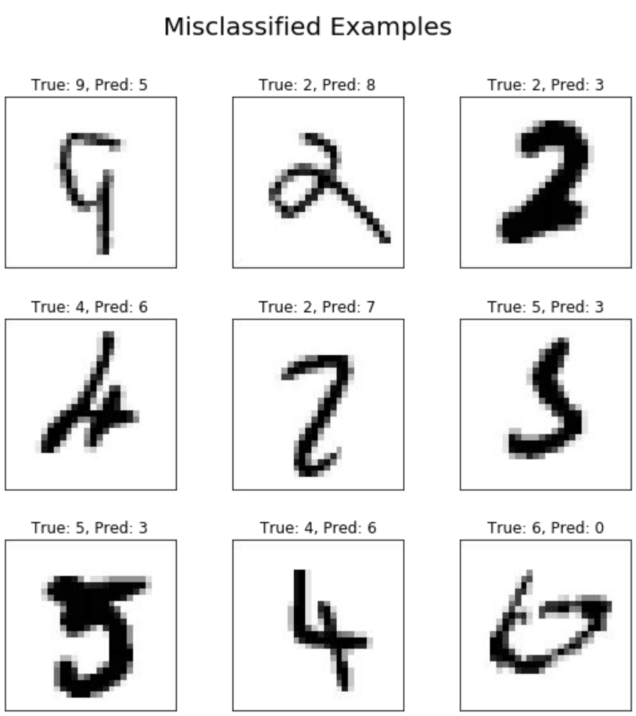
```
def plot_example_errors(images, cls_true, cls_pred, title=None):
    """
    Function for plotting examples of images that have been mis-classified
    :param images: array of all images, (#imgs, img_h*img_w)
    :param cls_true: corresponding true labels, (#imgs,)
    :param cls_pred: corresponding predicted labels, (#imgs,)
    """
    # Negate the boolean array.
    incorrect = np.logical_not(np.equal(cls_pred, cls_true))

    # Get the images from the test-set that have been incorrectly classified.
    incorrect_images = images[incorrect]

    # Get the true and predicted classes for those images.
    cls_pred = cls_pred[incorrect]
    cls_true = cls_true[incorrect]

    # Plot the first 9 images.
    plot_images(images=incorrect_images[0:9], cls_true=cls_true[0:9], cls_pred=cls_pred[0:9], title
=title)
```

```
# Test the network (only on 1000 samples) after training
x_test, y_test = load_data(mode='test')
feed_dict_test = {x: x_test[:1000].reshape((-1, timesteps, num_input)), y: y_test[:1000]}
loss_test, acc_test = sess.run([loss, accuracy], feed_dict=feed_dict_test)
print('---------------------------------------------------------')
print("Test loss: {0:.2f}, test accuracy: {1:.01%}".format(loss_test, acc_test))
print('---------------------------------------------------------')

# Plot some of the correct and misclassified examples
cls_pred = sess.run(cls_prediction, feed_dict=feed_dict_test)
cls_true = np.argmax(y_test, axis=1)
plot_images(x_test, cls_true, cls_pred, title='Correct Examples')
plot_example_errors(x_test[:1000], cls_true[:1000], cls_pred, title='Misclassified Examples')
plt.show()
```

Output:

```
Extracting MNIST_data/train-images-idx3-ubyte.gz
Extracting MNIST_data/train-labels-idx1-ubyte.gz
Extracting MNIST_data/t10k-images-idx3-ubyte.gz
Extracting MNIST_data/t10k-labels-idx1-ubyte.gz
---------------------------------------------------------
Test loss: 0.08, test accuracy: 97.7%
```





Program Continued:

```
# close the session after you are done with testing
sess.close()
```