SRI RAMAKRISHNA ENGINEERING COLLEGE

[Educational Service : SNR Sons Charitable Trust]

[Autonomous Institution, Reaccredited by NAAC with ‘A+’ Grade]

[Approved by AICTE and Permanently Affiliated to Anna University, Chennai] [ISO 9001:2015 Certified and all eligible programmes Accredited by NBA]

VATTAMALAIPALAYAM, N.G.G.O. COLONY POST, COIMBATORE- 641 022.

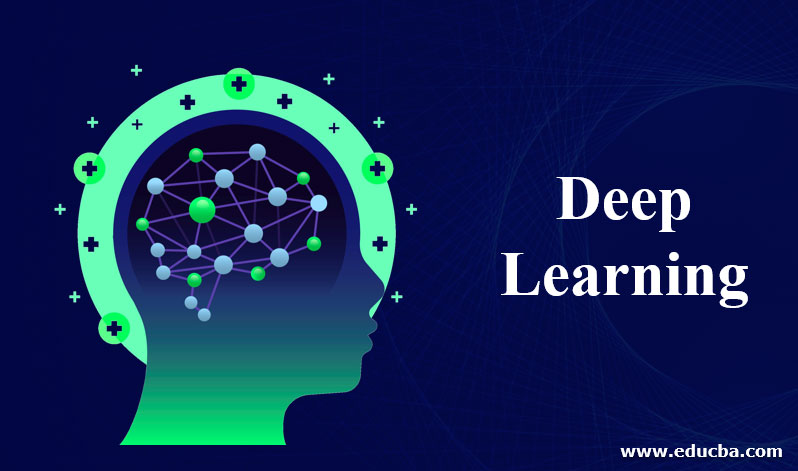
**DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE**

**Laboratory Manual**

For

**20AD275 DEEP LEARNING LABORATORY**

SEMESTER V



**COURSE INSTRUCTOR**

Mrs. B. SUGANYA, AP (OG)/AI & DS

**PROGRAMME: ARTIFICIAL INTELLIGENCE AND DATA SCIENCE**

# VISION AND MISSION

**VISION**

To achieve academic excellence in the domain of Artificial Intelligence and Data Science and produce globally competent professionals to solve futuristic societal challenges

# MISSION

* To actively engage in implementation of innovative intelligent solutions for inter disciplinary Artificial Intelligence based solutions with ethical standards
* To promote research, innovation and entrepreneurial skills through industry and academic collaboration

# PROGRAM EDUCATIONAL OBJECTIVES (PEOS)

The graduates of this program after four to five years will,

**PEO 1:** Design and develop solutions for real world problems based on business and societal needs, as skilled professionals or entrepreneurs.

**PEO 2:** Apply Artificial Intelligence and Data Science knowledge and skills to develop innovative solutions for multi-disciplinary problems, adhering to ethical standards

**PEO 3:** Engage in constructive research, professional development and life-long learning to adapt with emerging technologies

**PROGRAMME: ARTIFICIAL INTELLIGENCE AND DATA SCIENCE**

# Program Outcomes and Program Specific Outcomes (POs and PSOs)

Program Outcomes as stated by NBA: Engineering Graduates will be able to

1. **Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions**: Design solutions for complex engineering 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.
4. **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.
5. **Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. **The engineer 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 engineering practice.
7. **Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication:** Communicate effectively on complex engineering activities with the engineering 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.
11. **Project management and finance**: Demonstrate knowledge and understanding of the engineering 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.
12. **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.

# PROGRAM SPECIFIC OUTCOMES:

Graduates of Artificial Intelligence and Data Science at the time of graduation will be able to

**PSO 1:** Analyze, design and build sustainable intelligent solutions to solve challenges imposed by industry and society.

**PSO2:**Demonstrate data analysis skills to achieve effective insights and decision making to solve real life problems.

**PSO3:**Apply mathematical and statistical models to solve computational task, model real world problems using appropriate AI / ML algorithms.

# 20AD275 DEEP LEARNING LABORATORY 0 0 2 1

**PREREQUISITE**

* + 20IT244 - Python Programming

**COURSEOUTCOMES**

**On successful completion of the course, students will be able to**

|  |  |  |
| --- | --- | --- |
| **CO1:** | Make use of vector and matrix representation to build Feed Forward  Neural and back propagation | PO1,PO2,PO3, PO12 |
| **CO2:** | Apply CNN model to solve multi class image classification problem | PO1, PO2, PO3, PO4, PO12 |
| **CO3:** | Analyze model performance using regularization and optimization techniques | PO2, PO3, PO4, PO5, PO12 |
| **CO4:** | Experiment with sequence modeling using RNN | PO1, PO2, PO3, PO4, PO12 |

**LISTOF EXPERIMENTS**

1. Vectors and Matrix representation and manipulation
2. Deep Learning Frameworks
3. Simple Neural Network Formation
4. Perceptron for binary classification
5. Feed forward DNN
6. Back propagation for classification
7. Implementing CNN using ASL Dataset
8. Transfer Learning over CIFAR Dataset
9. Experiment with various optimizer and regularization techniques
10. One hot encoding and word embeddings for sentiment analysis
11. Headline generation using RNN

|  |  |
| --- | --- |
| **EX.NO:1** | **VECTORS AND MATRIX REPRESENTATION AND MANIPULATION** |
|  |

**AIM:**

To write a python program to represent & manipulate vectors and matrices

**ALGORITHM:**

Step 1: Create a python notebook

Step 2: Initialize a vector and perform addition, subtraction and multiplication

Step 3: Plot the vector and visualize the direction of movement

Step 4: Initialize a matrix and perform addition, subtraction and multiplication

Step 5:Calculate the time taken to perform operation using numpy and normal matrix representation

**Vector:**

Vectors: A vector, in programming, is a type of array that is one dimensional. Vectors are a logical element in programming languages that are used for storing a sequence of data elements of the same basic type. Members of a vector are called components.' The major difference between and array and a vector is that the container size of a vector can be easily increased and decreased to complement different data storage types.

**PROGRAM:**

import numpy as np

import matplotlib.pyplot as pltplt.quiver(0,0,3,4)

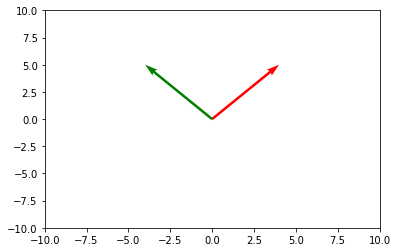
plt.quiver(0,0,4,5, scale\_units='xy',angles='xy',scale=1, color='r')

plt.quiver(0,0,-4,5, scale\_units='xy',angles='xy',scale=1, color='g')

plt.xlim(-10,10)

plt.ylim(-10,10)

plt.show()



**Define a function to display a vector passed as an argument**

def plot\_vector(vecs):

colors=['r','g','b','y']

i=0

for vec in vecs:

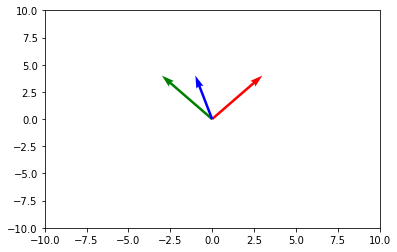
plt.quiver(vec[0],vec[1],vec[2],vec[3],scale\_units="xy",angles="xy",scale=1,color=colors[i%len(colors)])

i+=1

plt.xlim(-10,10)

plt.ylim(-10,10)

plt.show()



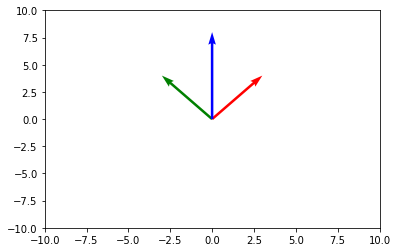
**ADDITION, SUBTRACTION AND MULTIPLICATION OF VECTORS**

**Addition:**

vecs=[np.asarray([0,0,3,4]),np.asarray([0,0,-3,4])]

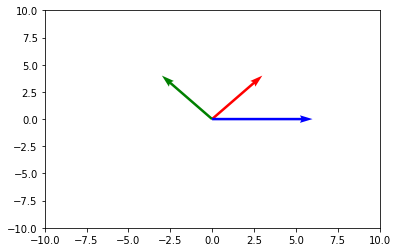
vecs[0]+vecs[1]

plot\_vector([vecs[0],vecs[1],vecs[0]+vecs[1]])

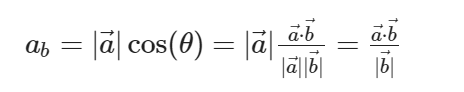


**Subtraction:**

plot\_vector([vecs[0],vecs[1],vecs[0]-vecs[1]])



**Multiplication:**

****

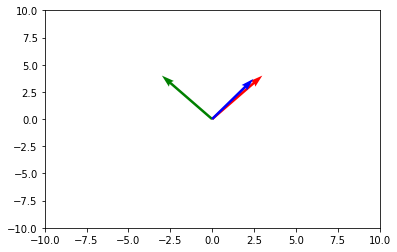
c=np.dot(np.asarray([2,4]),np.asarray([4,6]))/np.linalg.norm(np.asarray([4,6]))

b=np.asarray([4,6])

vec\_c=(c/(np.linalg.norm(np.asarray([4,6])))\*b)

print(vec\_c)

plot\_vector([vecs[0],vecs[1],[0,0,2.46153846,3.69230769]])



**Matrix Addition:**

a = np.array([0, 1, 2],dtype=np.int64)

b = np.array([5, 5, 5])

print("Matrix A\n", a.dtype)

print("Matrix B\n", b.dtype)

print("Regular matrix addition A+B\n", a + b)

print("Addition using Broadcasting A+5\n", a + 5)

**Output:**

Matrix A

int64

Matrix B

int64

Regular matrix addition A+B

[5 6 7]

Addition using Broadcasting A+5

[5 6 7]

# 2D matrix Addition

c = np.array([[0, 1, 2],[3, 4, 5],[6, 7, 8]])

d = np.array([[1, 2, 3],[1, 2, 3],[1, 2, 3]])

e = np.array([1, 2, 3])

print("Matrix C\n", c)

print("Matrix D\n", d)

print("Matrix E\n", e)

print("Regular matrix addition C+D\n", c + d)

print("Addition using Broadcasting C+E\n", c + e)

**Output:**

Matrix C

[[0 1 2]

[3 4 5]

[6 7 8]]

Matrix D

[[1 2 3]

[1 2 3]

[1 2 3]]

Matrix E

[1 2 3]

Regular matrix addition C+D

[[ 1 3 5]

[ 4 6 8]

[ 7 9 11]]

Addition using Broadcasting C+E

[[ 1 3 5]

[ 4 6 8]

[ 7 9 11]]

**Describing a Matrix:**print(c.size)#indicates total number of elements in the matrix

print(c.shape)#indicates dimension of the matrix

print(c.data)# indicates start addr of the matrix

print(c.dtype)#indicate the datatype

print(len(c))

**Output:**

9

(3, 3)

<memory at 0x7f3ae38cdbb0>

int64

3

**Matrix Multiplication:**

print("Vectorized Matrix Multiplication\n",c.dot(d),"\n")

**Output:**

Vectorized Matrix Multiplication

[[ 3 6 9]

[12 24 36]

[21 42 63]]

**RESULT:**

Thus, the python program to represent & manipulate vectors and matrices was successfully executed.

|  |  |
| --- | --- |
| **EX.NO:2** | **DEEP LEARNING FRAMEWORKS** |
|  |

**AIM:**

To study and work with different python frameworks and libraries used for implementing Deep Learning libraries.

**DESCRIPTION:**

**TensorFlow**

The most popular library for Machine Learning, TensorFlow is the best Python application development tool for advanced solutions. It simplifies building Machine Learning models for beginners and professionals. It has built-in modules for visualization, inspection and model serialization. TensorFlow is backed by the Google brain team, ensuring regular updates. It is useful for natural language processing, deep neural networks, image and speech recognition, and other functions for Deep Learning.

**Keras**

One of the fastest-growing Deep Learning framework packages, Keras enables using high-level network AP, along with a clean user interface. It enables engineers to combine standalone modules with low restrictions. Keras is highly used in building neural layers, solutions with activation and cost functions, batch normalization, and more. It works on top of TensorFlow, which extends its functionality for ML-based projects.

**PyTorch**

The primary aim of PyTorch is to speed up the entire process of Python app development for Machine Learning solutions. It has a C++ frontend along with the Python interface. PyTorch enables quick production deployment, providing companies with rapid solutions.

PyTorch offers training, building, and deploying small prototypes with ease. It is useful for neural networks, accelerated processing via GPU, and integrates quickly with the rest of Python’s ecosystem.

**Scikit-Learn**

One of the top Python libraries for Machine Learning, Scikit Learn integrates swiftly with NumPy and Pandas. It enables building Machine Learning models for classification, regression, clustering, dimensionality reduction, and other types of algorithms.

The main purpose of Scikit Learn is to focus only on data modeling. It is the fundamental library that engineers use to build end-to-end Machine Learning applications. There are also some excellent data pre-processing tools in the library.

**Theano**

Built on NumPy, Theano is a dynamic Machine Learning framework with a powerful interface, similar to the NumPy library. It is useful for manipulating and evaluating various mathematical expressions. Using Theano with GPU delivers faster results as it can compute 140 times faster on a GPU than a CPU. Theano helps to build efficient Machine Learning algorithms. It offers faster and stable monitoring of the most complicated variables.

**MXNet**

Known as one of the most popular Deep Learning frameworks for neural network development, MXNet is a flexible framework as it supports multiple programming languages, including Python, Java, C++, Scala, Go, R, and more. MXNet is one of the best Python frameworks for Deep learning as it is portable and scales to multiple GPU ports. It also offers faster context switching and optimized computation for different functions.

**Pandas**

Another of the highly known Python Machine Learning libraries in Python. Engineers use the library for data manipulation and analysis. It works amazingly well with structured data for Machine Learning algorithms. It offers great features to deploy ML and DL-based applications. Pandas assists with data reshaping, dataset joining, data filtration, alignment and easily handles missing data as well. It also provides a 2-D representation of data to make things convenient for python developers.

**NumPy**

An emerging package and one of the most useful frameworks for Machine Learning engineers, NumPy enables developers to process large amounts of multidimensional arrays. It is also useful in Fourier transforms, linear algebra, and other mathematical functions.

NumPy offers developers the capability to add speedy computations in the solution. Complicated functions can be easily executed – all thanks to NumPy’s power for scientific and numerical computing.

**NLTK**

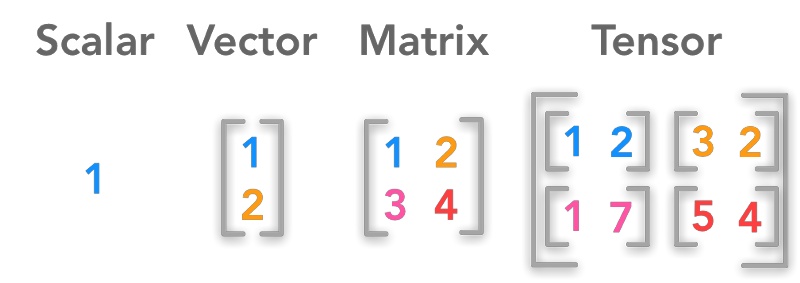
Also known as the Natural Language ToolKit, NLTK is used by a Python web development company to integrate Natural Language Processing. The tool is useful for Deep Learning solutions that require high amounts of text and speech processing. NLTK works well with FrameNet, WordNet, and Word2Vec for proper language processing. It also offers keyword search, optimization of tokens, voice recognition, and much more to ensure solutions that provide language capabilities work well.

**Spark ML**

The Spark ML framework simplifies matrix multiplication for Machine Learning. It divides the matrix into slices and runs the calculation on different servers. It requires a distributed architecture, ensuring that the computer doesn’t run out of memory while performing valuable operations. Engineers that use Spark for Big Data and Data Analytics may find it easy to work with Spark ML. It is one of the top Machine Learning frameworks and libraries that eliminate the complexities in preparing and processing large amounts of data.

**Tensors**

A tensor is an array that represents the types of data in the TensorFlow Python deep-learning library. A tensor, as compared to a one-dimensional vector or array or a two-dimensional matrix, can have n dimensions. The values in a tensor contain identical data types with a specified shape. Dimensionality is represented by the shape. A vector, for example, is a one-dimensional tensor, a matrix is a two-dimensional tensor, and a scalar is a zero-dimensional tensor.



Example:

# importing tensorflow

import tensorflow as tf

# creating nodes in computation graph

node1 = tf.constant(3, dtype=tf.int32)

node2 = tf.constant(5, dtype=tf.int32)

node3 = tf.add(node1, node2)

# create tensorflow session object

sess = tf.Session()

# evaluating node3 and printing the result

print("Sum of node1 and node2 is:",sess.run(node3))

# closing the session

sess.close()

**KERAS**

The Keras Workflow Model

* Define the training data—the input tensor and the target tensor
* Build a model or a set of Keras layers, which leads to the target tensor
* Structure a learning process by adding metrics, choosing a loss function, and defining the optimizer
* Use the fit() method to work through the training data and teach the model

**Example:**

from keras import models

from keras import layers

model = models.Sequential()

model.add(layers.Dense(32, activation='relu', input\_shape=(784,)))

model.add(layers.Dense(10, activation='softmax'))

input\_tensor = layers.Input(shape=(784,))

x = layers.Dense(32, activation='relu')(input\_tensor)

output\_tensor = layers.Dense(10, activation='softmax')(x)

model = models.Model(inputs=input\_tensor, outputs=output\_tensor)

from keras import optimizers

model.compile(optimizer=optimizers.RMSprop(lr=0.001),

loss='mse',

metrics=['accuracy'])

model.fit(input\_tensor, target\_tensor, batch\_size=128, epochs=10)

**RESULT:**

Thus, the study on different framework and libraries used for deep learning was completed.

|  |  |
| --- | --- |
| **EX.NO:3a** | **SIMPLE NEURAL NETWORK FORMATION** |
|  |

**AIM:**

To implement simple neural network for AND & OR gate using McCulloch-Pitts neuron.

**ALGORITHM:**

Step 1: Create a python notebook

Step 2: Initialize the input vector

Step 3: Initialize the threshold value

Step 4: Calculate weighted sum of neuron in the output layer

Step 5: Find the output based on the relationship between weighted sum and threshold value

**CODE:**

**AND :**

import numpy as np

def nparray(a,b):

  a=np.array([0,0,1,1])

  b=np.array([0,1,0,1])

  sum=np.add(a,b)

  threshold=2

  for i in sum:

    if i>=threshold:

      print(1)

    else:

      print(0)

a=[0,0,1,1]

b=[0,1,0,1]

nparray(a,b)

**Output:**

0

0

0

1

**OR:**

import numpy as np

def nparray(a,b):

  a=np.array([0,0,1,1])

  b=np.array([0,1,0,1])

  sum=np.add(a,b)

  threshold=1

  for i in sum:

    if i>=threshold:

      print(1)

    else:

      print(0)

a=[0,0,1,1]

b=[0,1,0,1]

nparray(a,b)

**Output:**

0

1

1

1

**RESULT:**

Thus, implementation of simple neural network for AND & OR gate using McCulloch-Pitts neuron was done successfully.

|  |  |
| --- | --- |
| **EX.NO:3b** | **SIMPLE NEURAL NETWORK FORMATION FOR BREAST CANCER DATASET** |
|  |

**AIM:**

To implement simple neural network over breast cancer dataset using McCulloch-Pitts neuron.

**ALGORITHM:**

Step 1: Create a python notebook

Step 2: Initialize the input vector

Step 3: Initialize the threshold value

Step 4: Calculate weighted sum of neuron in the output layer

Step 5: Find the output based on the relationship between weighted sum and threshold value

**CODE:**

import sklearn.datasets

import numpy as np

breast\_cancer = sklearn.datasets.load\_breast\_cancer()

X = breast\_cancer.data

Y = breast\_cancer.target

print(X.shape, Y.shape)

import pandas as pd

data = pd.DataFrame(breast\_cancer.data, columns=breast\_cancer.feature\_names)

data['class'] = breast\_cancer.target

data.head()

data.describe()

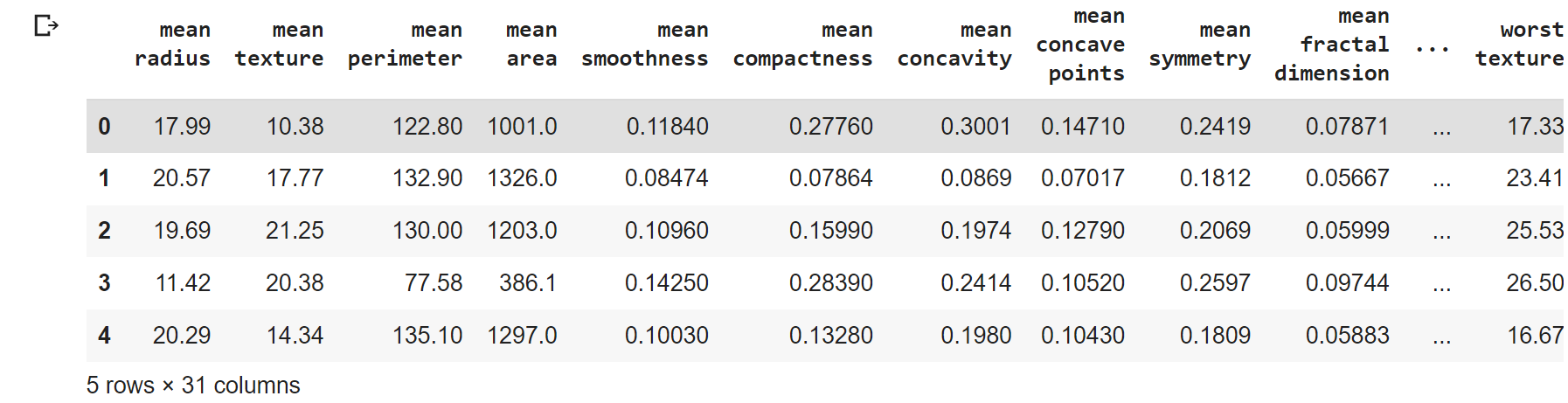
print(data['class'].value\_counts())

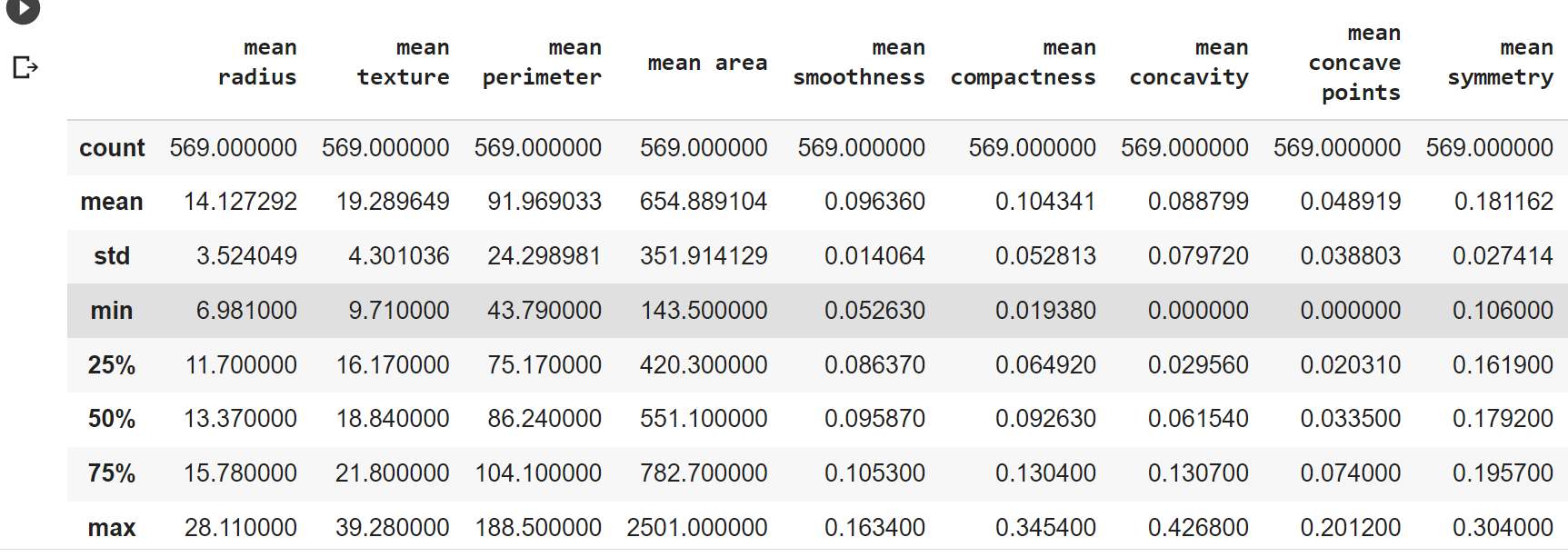
print(breast\_cancer.target\_names)

data.groupby('class').mean()

**output:**

(569, 30) (569,)



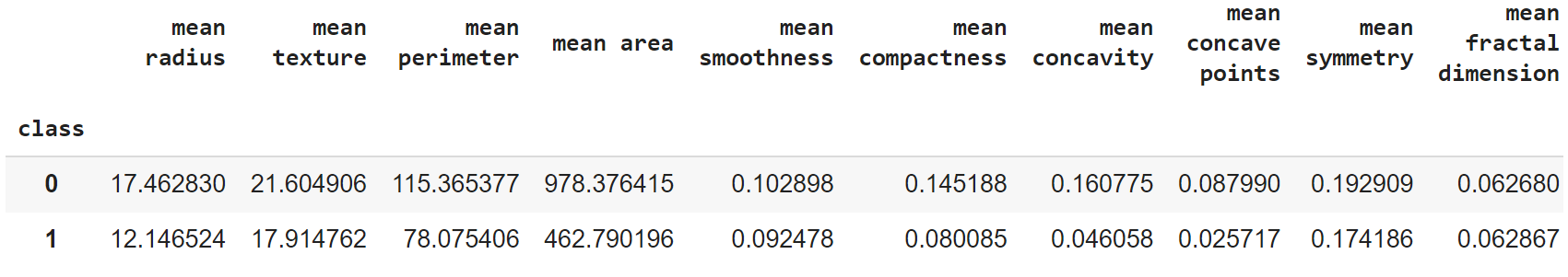
****

1 357

0 212

Name: class, dtype: int64

['malignant' 'benign']



**Train test split**

from sklearn.model\_selection import train\_test\_split

X = data.drop('class', axis=1)

Y = data['class']

type(X)

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y)

print(Y.shape, Y\_train.shape, Y\_test.shape)

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.1)

print(Y.mean(), Y\_train.mean(), Y\_test.mean())

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.1, stratify = Y)

print(X\_train.mean(), X\_test.mean(), X.mean())

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.1, stratify = Y, random\_state=1)

print(X\_train.mean(), X\_test.mean(), X.mean())

**output:**

pandas.core.frame.DataFrame

(569,) (426,) (143,)

0.6274165202108963 0.625 0.6491228070175439

# Binarisation of input

import matplotlib.pyplot as plt

plt.plot(X\_test.T, '\*')

plt.xticks(rotation='vertical')

plt.show()

X\_binarised\_3\_train = X\_train['mean area'].map(lambda x: 0 if x < 1000 else 1)

plt.plot(X\_binarised\_3\_train, '\*')

X\_binarised\_train = X\_train.apply(pd.cut, bins=2, labels=[1,0])

plt.plot(X\_binarised\_train.T, '\*')

plt.xticks(rotation='vertical')

plt.show()

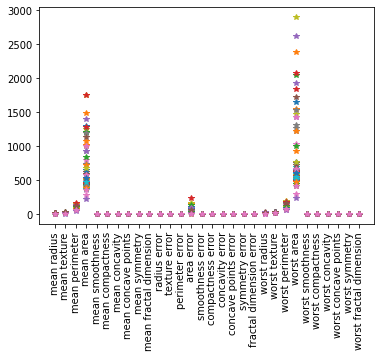
X\_binarised\_test = X\_test.apply(pd.cut, bins=2, labels=[1,0])

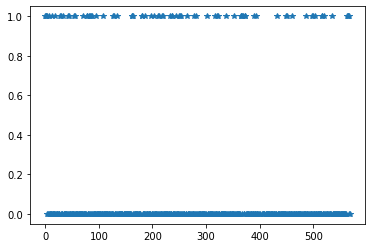
type(X\_binarised\_test)

X\_binarised\_test = X\_binarised\_test.values

X\_binarised\_train = X\_binarised\_train.values

type(X\_binarised\_test)

**Output:**

****



pandas.core.frame.DataFrame

numpy.ndarray

**MP Neuron Class**

class MPNeuron:

  def \_\_init\_\_(self):

    self.b = None

  def model(self, x):

    return(sum(x) >= self.b)

  def predict(self, X):

    Y = []

    for x in X:

      result = self.model(x)

      Y.append(result)

    return np.array(Y)

  def fit(self, X, Y):

    accuracy = {}

    for b in range(X.shape[1] + 1):

      self.b = b

      Y\_pred = self.predict(X)

      accuracy[b] = accuracy\_score(Y\_pred, Y)

    best\_b = max(accuracy, key = accuracy.get)

    self.b = best\_b

    print('Optimal value of b is', best\_b)

    print('Highest accuracy is', accuracy[best\_b])

mp\_neuron = MPNeuron()

mp\_neuron.fit(X\_binarised\_train, Y\_train)

Y\_test\_pred = mp\_neuron.predict(X\_binarised\_test)

accuracy\_test = accuracy\_score(Y\_test\_pred, Y\_test)

print(accuracy\_test)

**Output**

Optimal value of b is 28

Highest accuracy is 0.849609375

0.7894736842105263

**Result:**

Thus, the implementation of simple neural network over breast cancer dataset using McCulloch-Pitts neuron was executed successfully.

|  |  |
| --- | --- |
| **EX.NO:4** | **PERCEPTRON FOR BINARY CLASSIFICATION** |
|  |

**AIM:**

To implement simple neural network over breast cancer dataset using perceptron learning algorithm.

**ALGORITHM:**

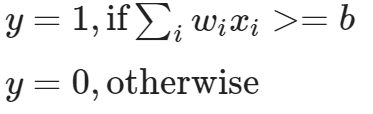
Step 1: Create a python notebook

Step 2: Initialize the input vector and weight matrix

Step 3: Calculate weighted sum of neuron in the output layer

Step 4: Apply activation function over the weighted sum

Step 5: Calculate new weight value based on perceptron learning rule



Step 6: Repeat step 3 to 5 until network converges

**CODE:**

import sklearn.datasets

import numpy as np

breast\_cancer = sklearn.datasets.load\_breast\_cancer()

X = breast\_cancer.data

Y = breast\_cancer.target

print(X.shape, Y.shape)

import pandas as pd

data = pd.DataFrame(breast\_cancer.data, columns=breast\_cancer.feature\_names)

data['class'] = breast\_cancer.target

data.head()

data.describe()

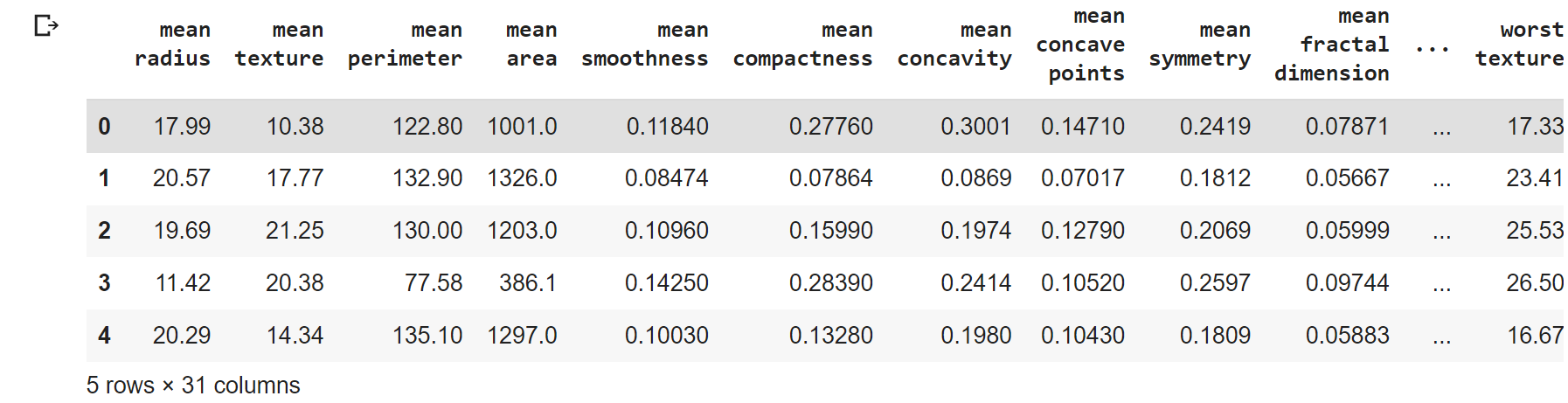
print(data['class'].value\_counts())

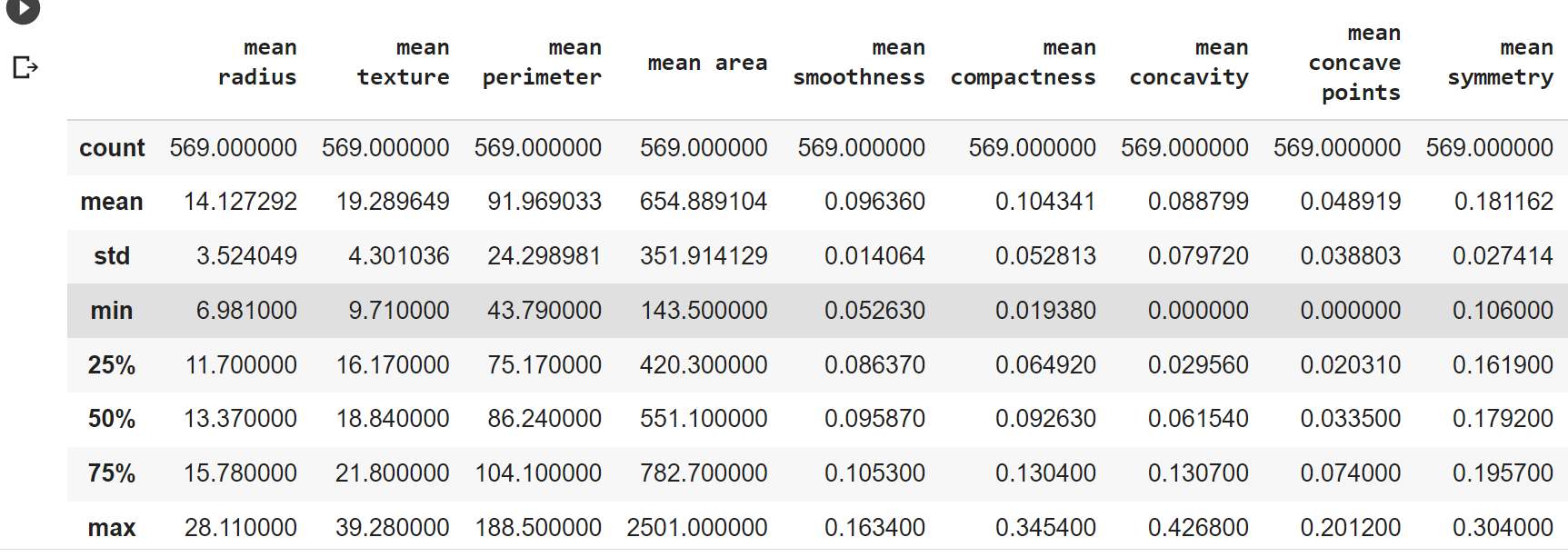
print(breast\_cancer.target\_names)

data.groupby('class').mean()

output:

(569, 30) (569,)



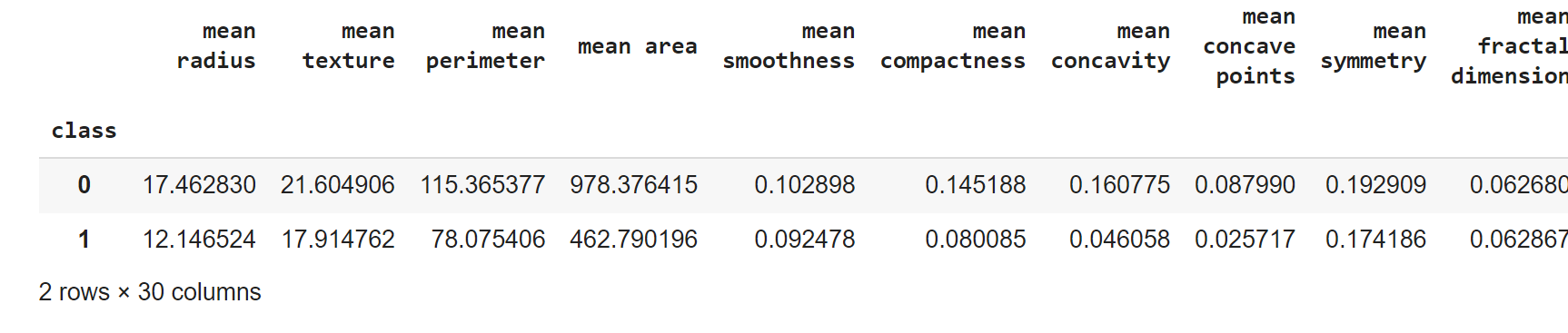


1 357

0 212

Name: class, dtype: int64

['malignant' 'benign']

****

**Train test split**

from sklearn.model\_selection import train\_test\_split

X = data.drop('class', axis=1)

Y = data['class']

type(X)

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y)

print(Y.shape, Y\_train.shape, Y\_test.shape)

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.1,

                                                    stratify = Y,

                                                    random\_state=1)

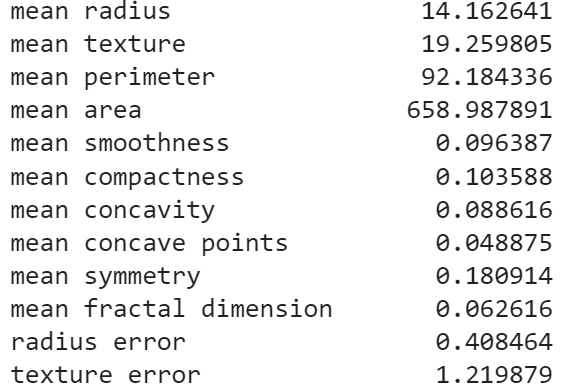
print(X\_train.mean(), X\_test.mean(), X.mean())

**Output:**

pandas.core.frame.DataFrame

(569,) (426,) (143,)

0.6274165202108963 0.625 0.6491228070175439

****

**Class Perceptron:**

X\_train = X\_train.values

X\_test = X\_test.values

from sklearn.metrics import accuracy\_score

import matplotlib.pyplot as plt

class Perceptron:

  def \_\_init\_\_ (self):

    self.w = None

    self.b = None

  def model(self, x):

    return 1 if (np.dot(self.w, x) >= self.b) else 0

  def predict(self, X):

    Y = []

    for x in X:

      result = self.model(x)

      Y.append(result)

    return np.array(Y)

  def fit(self, X, Y, epochs = 1):

    self.w = np.ones(X.shape[1])

    self.b = 0

    accuracy = {}

    max\_accuracy = 0

    wt\_matrix = []

    for i in range(epochs):

      for x, y in zip(X, Y):

        y\_pred = self.model(x)

        if y == 1 and y\_pred == 0:

          self.w = self.w +  x

          self.b = self.b - 1

        elif y == 0 and y\_pred == 1:

          self.w = self.w -  x

          self.b = self.b + 1

      wt\_matrix.append(self.w)

      accuracy[i] = accuracy\_score(self.predict(X), Y)

      if (accuracy[i] > max\_accuracy):

        max\_accuracy = accuracy[i]

        chkptw = self.w

        chkptb = self.b

    self.w = chkptw

    self.b = chkptb

    print(max\_accuracy)

    plt.plot(np.array(list(accuracy.values())).astype(float))

    plt.ylim([0, 1])

    plt.show()

    return np.array(wt\_matrix)

perceptron = Perceptron()

wt\_matrix = perceptron.fit(X\_train, Y\_train, 10)

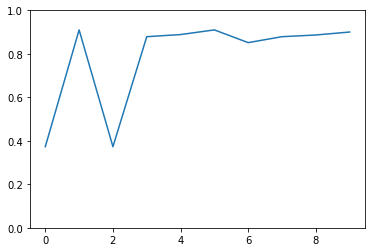
Y\_pred\_test = perceptron.predict(X\_test)

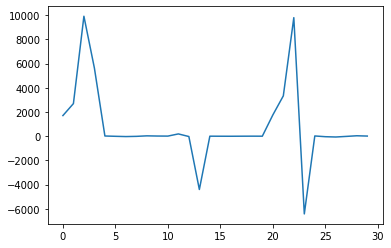
print(accuracy\_score(Y\_pred\_test, Y\_test))

plt.plot(wt\_matrix[-1,:])

plt.show()

Output:

0.91015625

0.9473684210526315

**RESULT:**

Thus, the implementation of simple neural network over breast cancer dataset using perceptron learning algorithm was successfully completed.

|  |  |
| --- | --- |
| **EX.NO:5a** | **FEED FORWARD DNN-BINARY CLASSIFICATION** |
|  |

**AIM:**

To write a python program to implement feed forward deep neural network for binary classification

**ALGORITHM:**

Step 1: Create a python notebook.

Step 2: Load the dataset

Step 3: Split the input and class labels in the dataset into training and testing data.

Step 4: Construct the neural network (Specify the no. of. neurons & activation function).

Step 5: Compile the model with binary loss function and optimizer.

Step 6: Train the model for 150 epochs using backpropagation algorithm and observe the accuracy.

**PROGRAM:**

from numpy import loadtxt

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

import pandas as pd

from google.colab import drive

drive.mount('/content/drive')

dataset=pd.read\_csv('/content/drive/MyDrive/DL/diabetes.csv')

dataset.head()

from sklearn.model\_selection import train\_test\_split

train,test = train\_test\_split(dataset, test\_size=0.25, random\_state=0, stratify=dataset['Outcome'])

train\_X = train[train.columns[:8]]

test\_X = test[test.columns[:8]]

train\_Y = train['Outcome']

test\_Y = test['Outcome']

train\_X.head()

train\_Y.head()

model = Sequential()

model.add(Dense(12, input\_shape=(8,), activation='relu'))

model.add(Dense(8, activation='relu'))

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

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

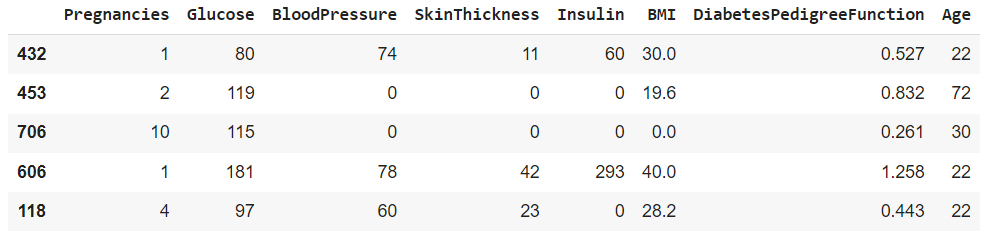
model.summary()

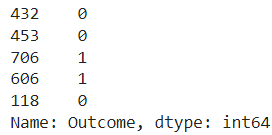
model.fit(train\_X, train\_Y, epochs=150, batch\_size=10)

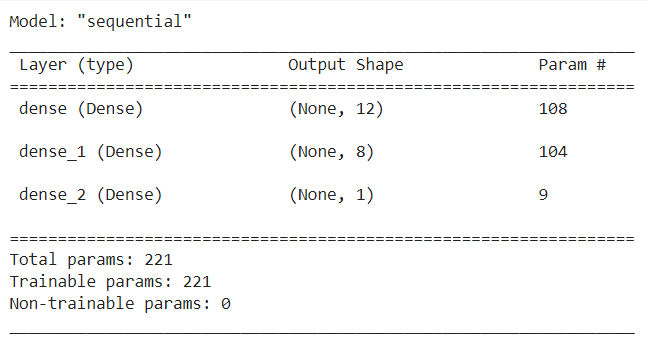
\_, accuracy = model.evaluate(test\_X, test\_Y)

print('Accuracy: %.2f' % (accuracy\*100))

**OUTPUT:**









**RESULT:**

Thus, the python program to implement feed forward deep neural network for binary classification is executed successfully.

|  |  |
| --- | --- |
| **EX.NO:5b** | **FEED FORWARD DNN-MULTICLASS CLASSIFICATION** |
|  |

**AIM:**

To write a python program to implement feed forward deep neural network for multi class classification

**ALGORITHM:**

Step 1: Create a python notebook.

Step 2: Load the dataset

Step 3: Split the input and class labels in the dataset into training and testing data.

Step 4: Convert output data into one-hot encoded representation

Step 5: Construct the neural network (Specify the no. of. neurons & activation function).

Step 6: Compile the model with binary loss function and optimizer.

Step 7: Train the model for 150 epochs using backpropagation algorithm and observe the accuracy.

**PROGRAM:**

from tensorflow.keras.datasets import mnist

(x\_train, y\_train), (x\_valid, y\_valid) = mnist.load\_data()

x\_train.shape

x\_valid.shape

x\_train.dtype

x\_train.min()

x\_train.max()

x\_train[0]

import matplotlib.pyplot as plt

image = x\_train[0]

plt.imshow(image, cmap='gray')

y\_train[0]

x\_train = x\_train.reshape(60000, 784)

x\_valid = x\_valid.reshape(10000, 784)

x\_train.shape

x\_train[0]

x\_train = x\_train / 255

x\_valid = x\_valid / 255

x\_train.dtype

x\_train.min()

x\_train.max()

import tensorflow.keras as keras

num\_categories = 10

y\_train = keras.utils.to\_categorical(y\_train, num\_categories)

y\_valid = keras.utils.to\_categorical(y\_valid, num\_categories)

y\_train[0:9]

from tensorflow.keras.models import Sequential

model = Sequential()

from tensorflow.keras.layers import Dense

model.add(Dense(units=512, activation='relu', input\_shape=(784,)))

model.add(Dense(units = 512, activation='relu'))

model.add(Dense(units = 10, activation='softmax'))

model.summary()

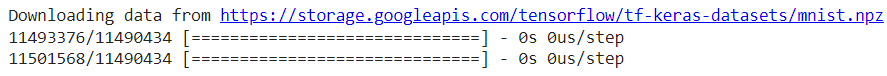
model.compile(loss='categorical\_crossentropy', metrics=['accuracy'])

history = model.fit(

    x\_train, y\_train, epochs=5, verbose=1, validation\_data=(x\_valid, y\_valid)

)

**OUTPUT:**

****

****



****

****

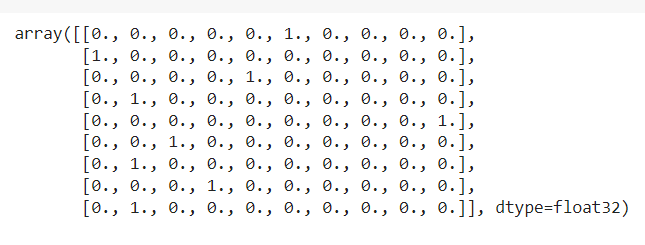
****

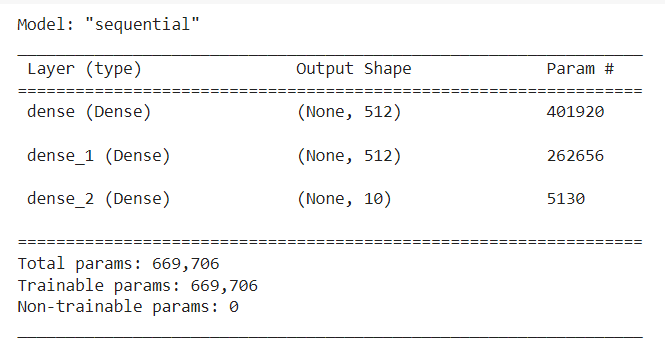
****

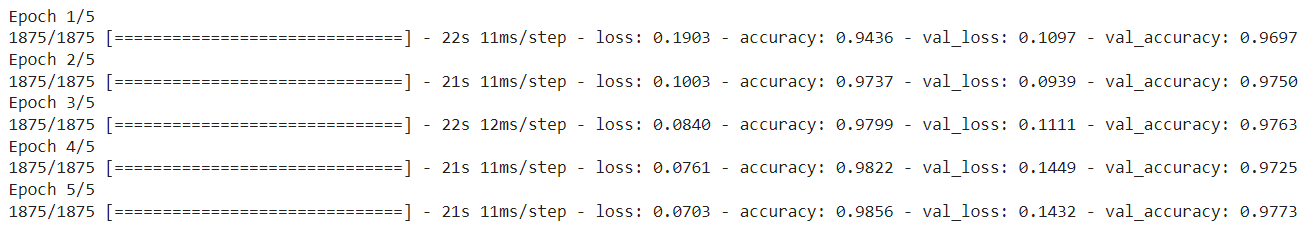
****

****

****

****

****

****

**RESULT:**

Thus, the python program to implement feed forward deep neural network for multi class classification is executed successfully.

|  |  |
| --- | --- |
| **EX.NO:6** | **BACK PROPAGATION** |
|  |

**AIM:**

To implement backpropagation algorithm for training feed forward network.

**ALGORITHM:**

Step 1: Create a python notebook.

Step 2: Load and pre-process the data set

Step 3: Split the data into training and testing set.

Step 4: The input is modeled using true weights W. Weights are usually chosen randomly.

Step 5: In feed forward pass, calculate the output of each neuron from the input layer to the hidden layer and to the output layer.

Step 6: Calculate the error in the output layer

Backpropagation Error= Actual Output – Desired Output

Step 7: In feedback pass, from the output layer, go back to the hidden layer to adjust the weights to reduce the error.

Step 8: Adjust the weight value between hidden and input layer after calculating the error at hidden layer.

Step 9: Update all the weight and bias values

Step 10: Repeat the process until the network converges.

**PROGRAM:**

from random import seed

from random import randrange

from random import random

from csv import reader

from math import exp

def load\_csv(filename):

  dataset = list()

  with open(filename, 'r') as file:

    csv\_reader = reader(file)

    for row in csv\_reader:

      if not row:

        continue

      dataset.append(row)

  return dataset

def str\_column\_to\_float(dataset, column):

  for row in dataset:

    row[column] = float(row[column].strip())

def str\_column\_to\_int(dataset, column):

  class\_values = [row[column] for row in dataset]

  unique = set(class\_values)

  lookup = dict()

  for i, value in enumerate(unique):

    lookup[value] = i

  for row in dataset:

    row[column] = lookup[row[column]]

  return lookup

def dataset\_minmax(dataset):

  minmax = list()

  stats = [[min(column), max(column)] for column in zip(\*dataset)]

  return stats

def normalize\_dataset(dataset, minmax):

  for row in dataset:

    for i in range(len(row)-1):

      row[i] = (row[i] - minmax[i][0]) / (minmax[i][1] - minmax[i][0])

def cross\_validation\_split(dataset, n\_folds):

  dataset\_split = list()

  dataset\_copy = list(dataset)

  fold\_size = int(len(dataset) / n\_folds)

  for i in range(n\_folds):

    fold = list()

    while len(fold) < fold\_size:

      index = randrange(len(dataset\_copy))

      fold.append(dataset\_copy.pop(index))

    dataset\_split.append(fold)

  return dataset\_split

def accuracy\_metric(actual, predicted):

  correct = 0

  for i in range(len(actual)):

    if actual[i] == predicted[i]:

      correct += 1

  return correct / float(len(actual)) \* 100.0

def evaluate\_algorithm(dataset, algorithm, n\_folds, \*args):

  folds = cross\_validation\_split(dataset, n\_folds)

  scores = list()

  for fold in folds:

    train\_set = list(folds)

    train\_set.remove(fold)

    train\_set = sum(train\_set, [])

    test\_set = list()

    for row in fold:

      row\_copy = list(row)

      test\_set.append(row\_copy)

      row\_copy[-1] = None

    predicted = algorithm(train\_set, test\_set, \*args)

    actual = [row[-1] for row in fold]

    accuracy = accuracy\_metric(actual, predicted)

    scores.append(accuracy)

  return scores

def activate(weights, inputs):

  activation = weights[-1]

  for i in range(len(weights)-1):

    activation += weights[i] \* inputs[i]

  return activation

def transfer(activation):

  return 1.0 / (1.0 + exp(-activation))

def forward\_propagate(network, row):

  inputs = row

  for layer in network:

    new\_inputs = []

    for neuron in layer:

      activation = activate(neuron['weights'], inputs)

      neuron['output'] = transfer(activation)

      new\_inputs.append(neuron['output'])

    inputs = new\_inputs

  return inputs

def transfer\_derivative(output):

  return output \* (1.0 - output)

def backward\_propagate\_error(network, expected):

  for i in reversed(range(len(network))):

    layer = network[i]

    errors = list()

    if i != len(network)-1:

      for j in range(len(layer)):

        error = 0.0

        for neuron in network[i + 1]:

          error += (neuron['weights'][j] \* neuron['delta'])

        errors.append(error)

    else:

      for j in range(len(layer)):

        neuron = layer[j]

        errors.append(neuron['output'] - expected[j])

    for j in range(len(layer)):

      neuron = layer[j]

      neuron['delta'] = errors[j] \* transfer\_derivative(neuron['output'])

def update\_weights(network, row, l\_rate):

  for i in range(len(network)):

    inputs = row[:-1]

    if i != 0:

      inputs = [neuron['output'] for neuron in network[i - 1]]

    for neuron in network[i]:

      for j in range(len(inputs)):

        neuron['weights'][j] -= l\_rate \* neuron['delta'] \* inputs[j]

      neuron['weights'][-1] -= l\_rate \* neuron['delta']

def train\_network(network, train, l\_rate, n\_epoch, n\_outputs):

  for epoch in range(n\_epoch):

    for row in train:

      outputs = forward\_propagate(network, row)

      expected = [0 for i in range(n\_outputs)]

      expected[row[-1]] = 1

      backward\_propagate\_error(network, expected)

      update\_weights(network, row, l\_rate)

def initialize\_network(n\_inputs, n\_hidden, n\_outputs):

  network = list()

  hidden\_layer = [{'weights':[random() for i in range(n\_inputs + 1)]} for i in range(n\_hidden)]

  network.append(hidden\_layer)

  output\_layer = [{'weights':[random() for i in range(n\_hidden + 1)]} for i in range(n\_outputs)]

  network.append(output\_layer)

  return network

def predict(network, row):

  outputs = forward\_propagate(network, row)

  return outputs.index(max(outputs))

def back\_propagation(train, test, l\_rate, n\_epoch, n\_hidden):

  n\_inputs = len(train[0]) - 1

  n\_outputs = len(set([row[-1] for row in train]))

  network = initialize\_network(n\_inputs, n\_hidden, n\_outputs)

  train\_network(network, train, l\_rate, n\_epoch, n\_outputs)

  predictions = list()

  for row in test:

    prediction = predict(network, row)

    predictions.append(prediction)

  return(predictions)

seed(1)

filename = 'wheat-seeds.csv'

dataset = load\_csv(filename)

for i in range(len(dataset[0])-1):

  str\_column\_to\_float(dataset, i)

str\_column\_to\_int(dataset, len(dataset[0])-1)

minmax = dataset\_minmax(dataset)

normalize\_dataset(dataset, minmax)

n\_folds = 5

l\_rate = 0.3

n\_epoch = 500

n\_hidden = 5

scores = evaluate\_algorithm(dataset, back\_propagation, n\_folds, l\_rate, n\_epoch, n\_hidden)

print('Scores: %s' % scores)

print('Mean Accuracy: %.3f%%' % (sum(scores)/float(len(scores))))

**OUTPUT:**

****

**RESULT:**

Thus, the python program to implement backpropagation algorithm was successfully completed.

|  |  |
| --- | --- |
| **EX.NO:7a** | **IMPLEMENT DNN USING ASL DATASET** |
|  |

**AIM:**

To implement image classification in Deep Neural Network (DNN) using American Sign Language (ASL) dataset.

**ALGORITHM:**

Step 1: Create a python notebook.

Step 2: Load the dataset

Step 3: Split the input and class labels in the dataset into training and testing data.

Step 4: Convert output data into one-hot encoded representation

Step 5: Construct the neural network (Specify the no. of. neurons & activation function).

Step 6: Compile the model with binary loss function and optimizer.

Step 7: Train the model for 150 epochs using backpropagation algorithm and observe the accuracy.

**PROGRAM:**

import pandas as pd

train\_df = pd.read\_csv("data/asl\_data/sign\_mnist\_train.csv")

valid\_df = pd.read\_csv("data/asl\_data/sign\_mnist\_valid.csv")

train\_df.head()

y\_train = train\_df['label']

y\_valid = valid\_df['label']

del train\_df['label']

del valid\_df['label']

x\_train = train\_df.values

x\_valid = valid\_df.values

x\_train.shape

y\_train.shape

y\_train.shape

y\_train.shape

import matplotlib.pyplot as plt

plt.figure(figsize=(40,40))

num\_images = 20

for i in range(num\_images):

    row = x\_train[i]

    label = y\_train[i]

    image = row.reshape(28,28)

    plt.subplot(1, num\_images, i+1)

    plt.title(label, fontdict={'fontsize': 30})

    plt.axis('off')

    plt.imshow(image, cmap='gray')

x\_train.min()

x\_train.max()

x\_train = train\_df.values / 255

x\_valid = valid\_df.values / 255

import tensorflow.keras as keras

num\_classes = 24

if not y\_train.shape[-1] == 24:  # Avoid running multiple times

    y\_train = keras.utils.to\_categorical(y\_train, num\_classes)

    y\_valid = keras.utils.to\_categorical(y\_valid, num\_classes)

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

model = Sequential()

model.add(Dense(units = 512, activation='relu', input\_shape=(784,)))

model.add(Dense(units = 512, activation='relu'))

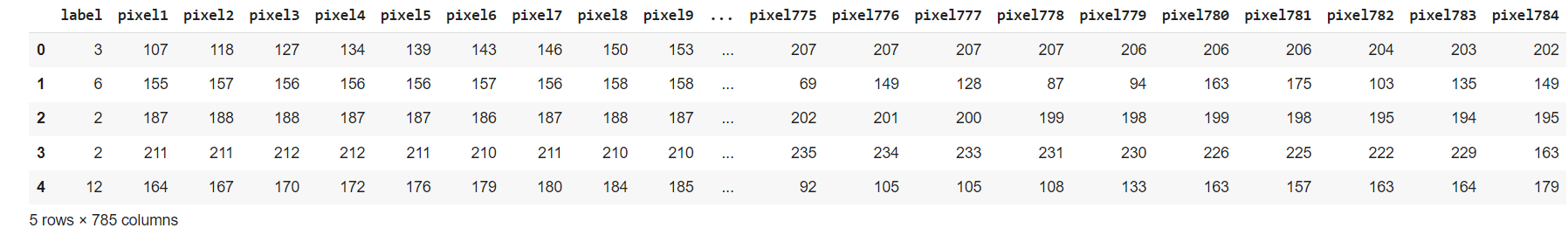
model.add(Dense(units = num\_classes, activation='softmax'))

model.summary()

model.compile(loss='categorical\_crossentropy', metrics=['accuracy'])

model.fit(x\_train, y\_train, epochs=20, verbose=1, validation\_data=(x\_valid, y\_valid))

**Output:**





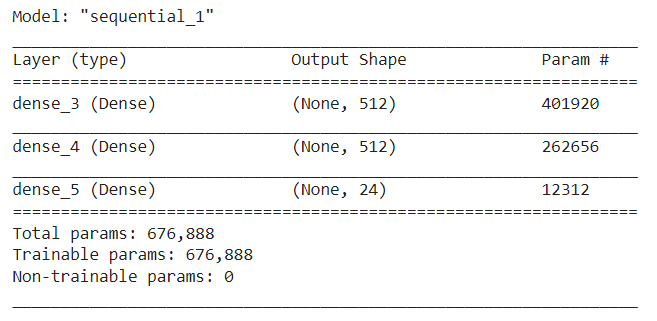












**RESULT:**

Thus, the python program to implement image classification in Deep Neural Network using American Sign Language dataset is executed successfully.

|  |  |
| --- | --- |
| **EX.NO:7b** | **IMPLEMENT CNN USING ASL DATASET** |
|  |

**AIM:**

To implement image classification using Convolutional Neural Network (CNN) for American Sign Language (ASL) dataset

**ALGORITHM:**

Step 1: Create a python notebook.

Step 2: Load the dataset

Step 3: Split the input and class labels in the dataset into training and testing data.

Step 4: Convert output data into one-hot encoded representation

Step 5: Construct the neural network (Specify the no. of. neurons, different layers & activation function).

Step 6: Compile the model with categorical cross entropy as loss function and accuracy measure.

Step 7: Train the model for 150 epochs using backpropagation algorithm and observe the accuracy.

**PROGRAM:**

import tensorflow.keras as keras

import pandas as pd

train\_df = pd.read\_csv("data/asl\_data/sign\_mnist\_train.csv")

valid\_df = pd.read\_csv("data/asl\_data/sign\_mnist\_valid.csv")

y\_train = train\_df['label']

y\_valid = valid\_df['label']

del train\_df['label']

del valid\_df['label']

x\_train = train\_df.values

x\_valid = valid\_df.values

num\_classes = 24

y\_train = keras.utils.to\_categorical(y\_train, num\_classes)

y\_valid = keras.utils.to\_categorical(y\_valid, num\_classes)

x\_train = x\_train / 255

x\_valid = x\_valid / 255

x\_train.shape, x\_valid.shape

x\_train = x\_train.reshape(-1,28,28,1)

x\_valid = x\_valid.reshape(-1,28,28,1)

x\_train.shape

x\_valid.shape

x\_train.shape, x\_valid.shape

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import (

    Dense,

    Conv2D,

    MaxPool2D,

    Flatten,

    Dropout,

    BatchNormalization,

)

model = Sequential()

model.add(Conv2D(75, (3, 3), strides=1, padding="same", activation="relu",

                 input\_shape=(28, 28, 1)))

model.add(BatchNormalization())

model.add(MaxPool2D((2, 2), strides=2, padding="same"))

model.add(Conv2D(50, (3, 3), strides=1, padding="same", activation="relu"))

model.add(Dropout(0.2))

model.add(BatchNormalization())

model.add(MaxPool2D((2, 2), strides=2, padding="same"))

model.add(Conv2D(25, (3, 3), strides=1, padding="same", activation="relu"))

model.add(BatchNormalization())

model.add(MaxPool2D((2, 2), strides=2, padding="same"))

model.add(Flatten())

model.add(Dense(units=512, activation="relu"))

model.add(Dropout(0.3))

model.add(Dense(units=num\_classes, activation="softmax"))

model.summary()

model.compile(loss="categorical\_crossentropy", metrics=["accuracy"])

model.fit(x\_train, y\_train, epochs=10, verbose=1, validation\_data=(x\_valid, y\_valid))

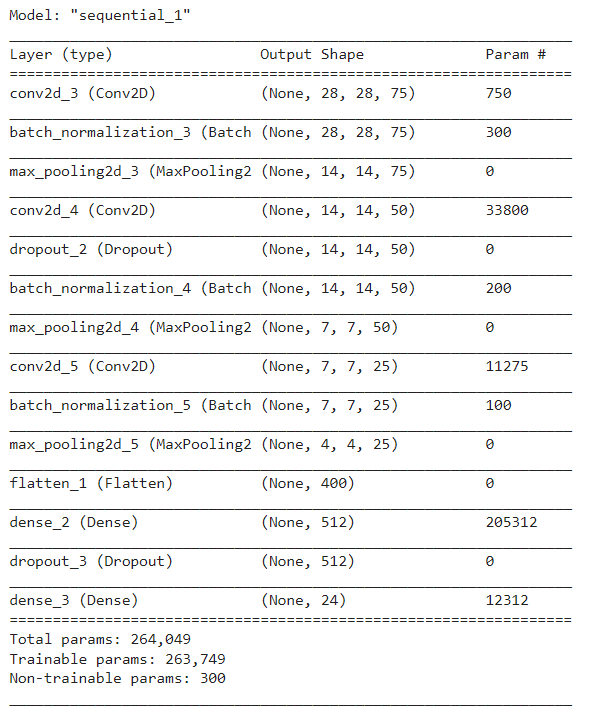
**OUTPUT:**

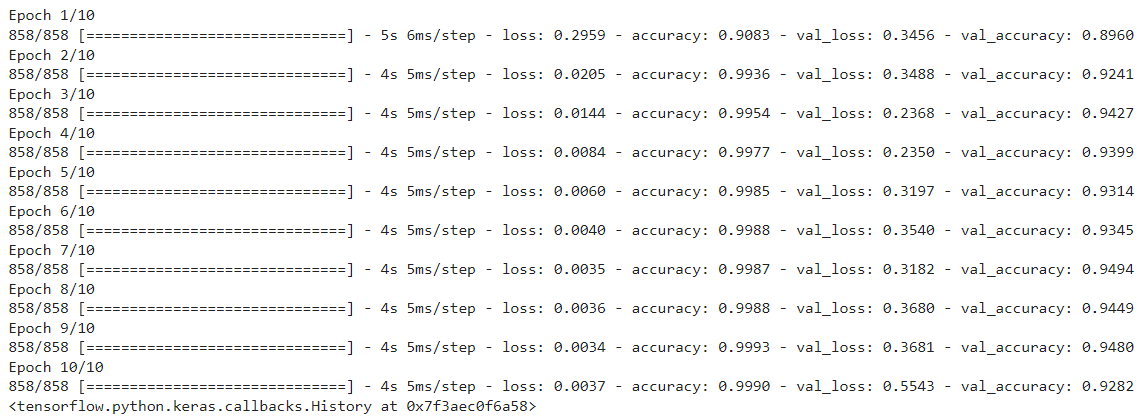












**Result:**

Thus, the python program to implement Image Classification in Convolutional Neural Network using American Sign Language dataset is executed successfully

|  |  |
| --- | --- |
| **EX.NO:8** | **TRANSFER LEARNING** |
|  |

**AIM:**

To implement transfer learning using VGGNet16 to perform automatic doggy door classifier.

**ALGORITHM:**

Step 1: Create a python notebook.

Step 2: Load the dataset

Step 3: Split the input and class labels in the dataset into training and testing data.

Step 4: Convert output data into one-hot encoded representation

Step 5: Prepare a pretrained model for transfer learning by freezing the base model.

Step 6: Perform data augmentation with rotation, zooming, flipping and shifting

Step 7: Compile the model with binary loss function and optimizer.

Step 8: Train the model for 20 epochs using and observe the accuracy.

**PROGRAM:**

from tensorflow import keras

base\_model = keras.applications.VGG16(

    weights='imagenet',

    input\_shape=(224, 224, 3),

    include\_top=False)

base\_model.summary()

base\_model.trainable = False

inputs = keras.Input(shape=(224, 224, 3))

x = base\_model(inputs, training=False)

x = keras.layers.GlobalAveragePooling2D()(x)

outputs = keras.layers.Dense(1)(x)

model = keras.Model(inputs, outputs)

model.summary()

model.compile(loss=keras.losses.BinaryCrossentropy(from\_logits=True), metrics=[keras.metrics.BinaryAccuracy()])

from tensorflow.keras.preprocessing.image import ImageDataGenerator

datagen\_train = ImageDataGenerator(

    samplewise\_center=True,

    rotation\_range=10,

    zoom\_range=0.1,

    width\_shift\_range=0.1,

    height\_shift\_range=0.1,

    horizontal\_flip=True,

    vertical\_flip=False,

)

datagen\_valid = ImageDataGenerator(samplewise\_center=True)

train\_it = datagen\_train.flow\_from\_directory(

    "data/presidential\_doggy\_door/train/",

    target\_size=(224, 224),

    color\_mode="rgb",

    class\_mode="binary",

    batch\_size=8,

)

valid\_it = datagen\_valid.flow\_from\_directory(

    "data/presidential\_doggy\_door/valid/",

    target\_size=(224, 224),

    color\_mode="rgb",

    class\_mode="binary",

    batch\_size=8,

)

model.fit(train\_it, steps\_per\_epoch=12, validation\_data=valid\_it, validation\_steps=4, epochs=20)

base\_model.trainable = True

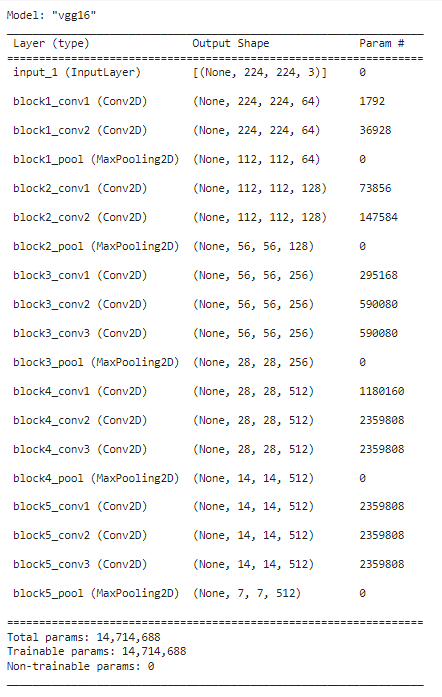
model.compile(optimizer=keras.optimizers.RMSprop(learning\_rate = .00001),

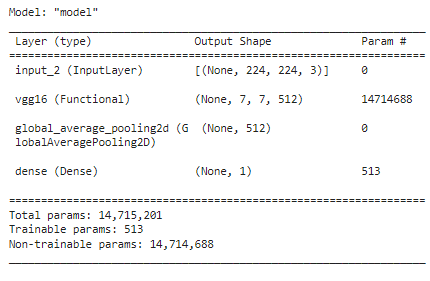
              loss=keras.losses.BinaryCrossentropy(from\_logits=True),

              metrics=[keras.metrics.BinaryAccuracy()])

model.fit(train\_it, steps\_per\_epoch=12, validation\_data=valid\_it, validation\_steps=4, epochs=10)

**OUTPUT:**

****

****

**RESULT:**

Thus, the python program to implement transfer learning is executed successfully

|  |  |
| --- | --- |
| **EX.NO:9** | **ONE HOT ENCODING** |
|  |

**AIM:**

To implement one hot encoding to convert categorical data to numerical data.

**ALGORITHM:**

Step 1: Create a python notebook.

Step 2: Initialize the input

Step 3: Create an empty dictionary

Step 4: Create a counter for counting the number of key-value pairs in the token\_length

Step 5: Select the elements of the samples which are the two sentences

Step 6: If the considered word is not present in the dictionary token\_index, add it to the token\_index.

Step 7: Update the integer value into binary value in one hot representation.

**PROGRAM:**

import numpy as np

samples = {'Jupiter has seventy nine known moons', 'Neptune has fourteen confirmed moons'}

token\_index = []

counter = 0

for sample in samples:

  for considered\_word in sample.split():

    if considered\_word not in token\_index:

token\_index

from numpy import array

from numpy import argmax

from sklearn.preprocessing import LabelEncoder

from sklearn.preprocessing import OneHotEncoder

data = token\_index

values = array(data)

print(values)

label\_encoder = LabelEncoder()

integer\_encoded = label\_encoder.fit\_transform(values)

print(integer\_encoded)

onehot\_encoder = OneHotEncoder(sparse=False)

integer\_encoded = integer\_encoded.reshape(len(integer\_encoded), 1)

onehot\_encoded = onehot\_encoder.fit\_transform(integer\_encoded)

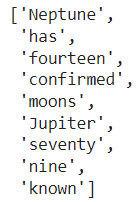
print(onehot\_encoded)

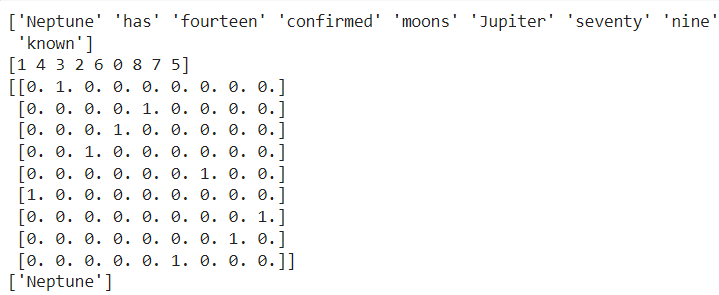
inverted = label\_encoder.inverse\_transform([argmax(onehot\_encoded[0, :])])

print(inverted)

model.fit(train\_it, steps\_per\_epoch=12, validation\_data=valid\_it, validation\_steps=4, epochs=10)

**OUTPUT:**





**RESULT:**

Thus, the implementation of one hot encoding is executed successfully

|  |  |
| --- | --- |
| **EX.NO:10** | **WORD EMBEDDING FOR SENTIMENT ANALYSIS** |
|  |

**AIM:**

To implement word embedding for sentiment analysis.

**ALGORITHM:**

Step 1: Create a python notebook

Step 2: Initialize the input

Step 3: Represent text string inputs to numbers

Step 4: Use the embedding layer to initialize weights randomly

Step 5: Create, compile and train the model

Step 6: Retrieve the word embeddings learned during training.

**PROGRAM:**

import numpy as np

from tensorflow.keras.preprocessing.text import one\_hot

from tensorflow.keras.preprocessing.sequence import pad\_sequences

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

from tensorflow.keras.layers import Flatten

from tensorflow.keras.layers import Embedding

reviews = ['nice food',

        'amazing restaurant',

        'too good',

        'just loved it!',

        'will go again',

        'horrible food',

        'never go there',

        'poor service',

        'poor quality',

        'needs improvement']

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

one\_hot("amazing restaurant",30)

vocab\_size = 30

encoded\_reviews = [one\_hot(d, vocab\_size) for d in reviews]

print(encoded\_reviews)

max\_length = 4

padded\_reviews = pad\_sequences(encoded\_reviews, maxlen=max\_length, padding='post')

print(padded\_reviews)

embeded\_vector\_size = 5

model = Sequential()

model.add(Embedding(vocab\_size, embeded\_vector\_size, input\_length=max\_length,name="embedding"))

model.add(Flatten())

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

X = padded\_reviews

y = sentiment

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

print(model.summary())

model.fit(X, y, epochs=50, verbose=0)

loss, accuracy = model.evaluate(X, y)

accuracy

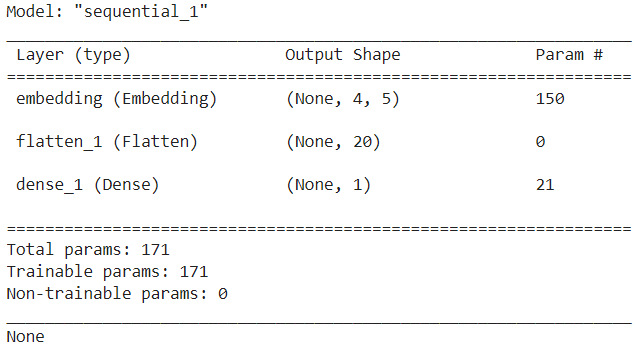
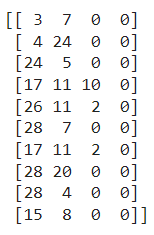
weights = model.get\_layer('embedding').get\_weights()[0]

len(weights)

weights[13]

**OUTPUT:**

****

****

****

****

****

****

**RESULT:**

Thus, python program to implement word embedding for sentiment analysis is executed successfully.

|  |  |
| --- | --- |
| **EX.NO:11** | **CIFAR10 USING CNN AND VARIOUS OPTIMIZER TECHNIQUES** |
|  |

**AIM:**

To implement image classification in Convolutional Neural Network (CNN) using CIFAR10 dataset with various optimizer technique.

**ALGORITHM:**

Step 1: Create a python notebook.

Step 2: Load the dataset

Step 3: Split the input and class labels in the dataset into training and testing data.

Step 4: Construct the neural network (Specify the no. of. neurons & activation function).

Step 5: Compile the model with sparse categorical loss function and optimizer.

Step 6: Train the model for 15 epochs and obtain the test accuracy.

Step 7: Use different optimizer and test the network performance

**PROGRAM:**

import tensorflow.keras as keras

import matplotlib.pyplot as plt

from tensorflow.keras.datasets import cifar10

class\_names=['airplane','automobile','bird','cat','deer','dog','frog','horse','ship','truck']

(x\_train,y\_train),(x\_test,y\_test)=cifar10.load\_data()

x\_train=x\_train/255.0

x\_train.shape

x\_test=x\_test/255.0

x\_test.shape

plt.imshow(x\_test[215])

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import (

    Dense,

    Conv2D,

    MaxPool2D,

    Flatten,

    Dropout,

    BatchNormalization,

)

model=Sequential()

model.add(Conv2D(filters=32,kernel\_size=3,padding="same", activation="relu", input\_shape=[32,32,3]))

model.add(Conv2D(filters=32,kernel\_size=3,padding="same", activation="relu"))

model.add(MaxPool2D(pool\_size=2,strides=2,padding='valid'))

model.add(Conv2D(filters=64,kernel\_size=3,padding="same", activation="relu"))

model.add(Conv2D(filters=64,kernel\_size=3,padding="same", activation="relu"))

model.add(MaxPool2D(pool\_size=2,strides=2,padding='valid'))

model.add(Flatten())

model.add(Dropout(0.5,noise\_shape=None,seed=None))

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

model.add(Dense(units=10,activation='softmax'))

model.summary()

**Adam**

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

model.fit(x\_train,y\_train,epochs=15)

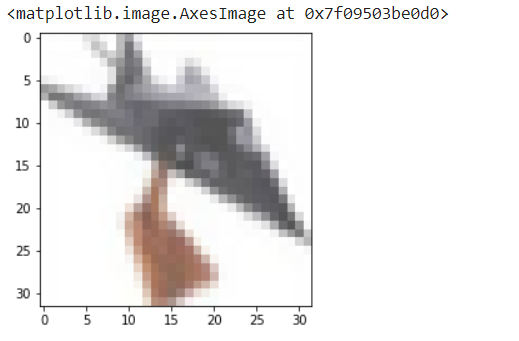
test\_loss, test\_accuracy = model.evaluate(x\_test, y\_test)

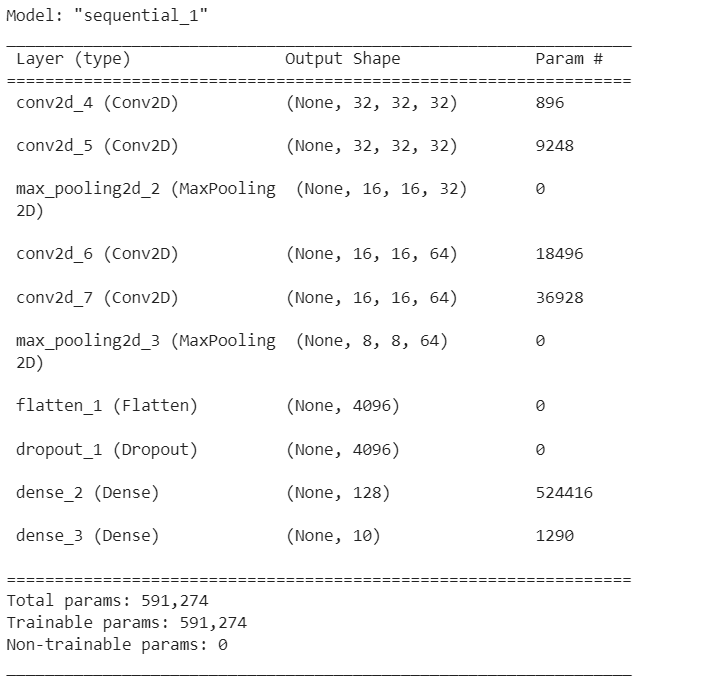
print("Test accuracy: {}".format(test\_accuracy))

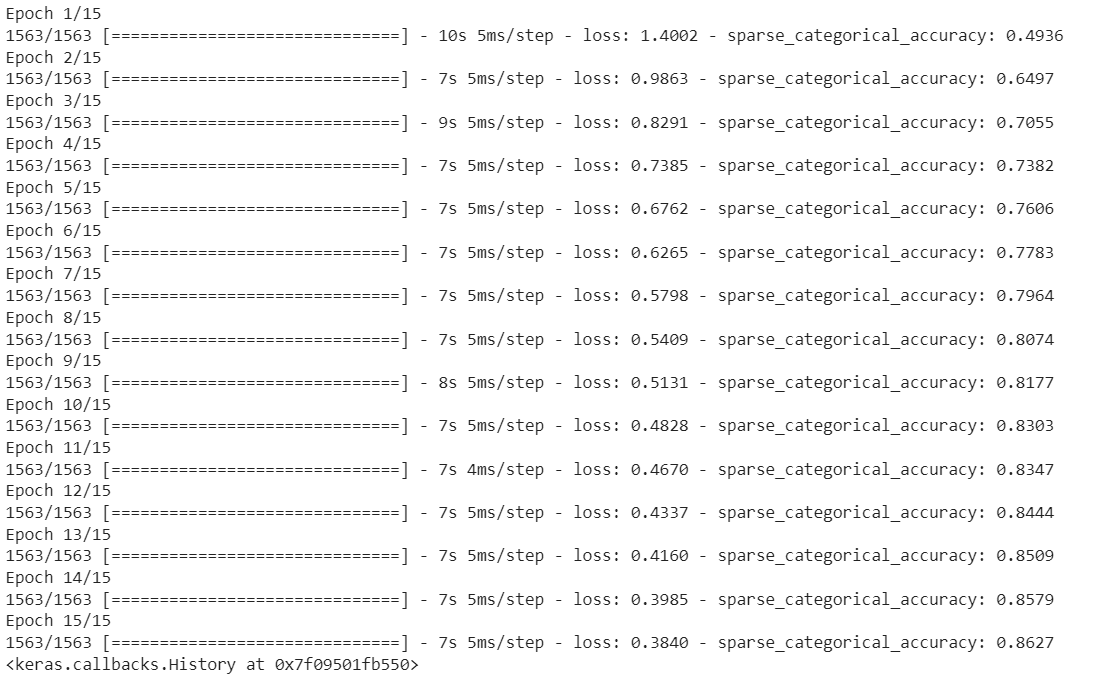
**Output:**

****

****

****

****

****

313/313 [==============================] - 1s 3ms/step - loss: 0.6794 sparse\_categorical\_accuracy: 0.7795

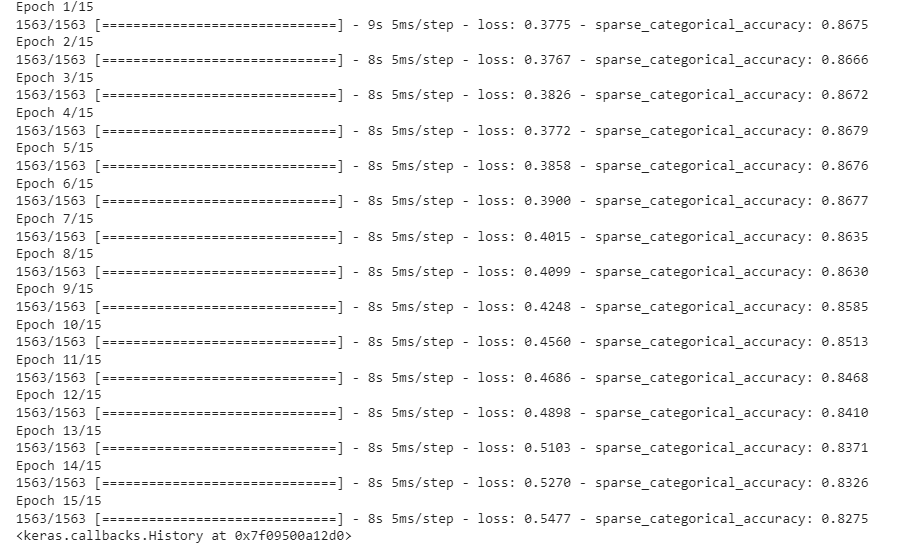
Test accuracy: 0.7795000076293945

**RMS Prop**

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

model.fit(x\_train,y\_train,epochs=15)

test\_loss, test\_accuracy = model.evaluate(x\_test, y\_test)

****print("Test accuracy: {}".format(test\_accuracy))

313/313 [==============================] - 1s 4ms/step - loss: 0.8123 - sparse\_categorical\_accuracy: 0.7839

Test accuracy: 0.7839000225067139

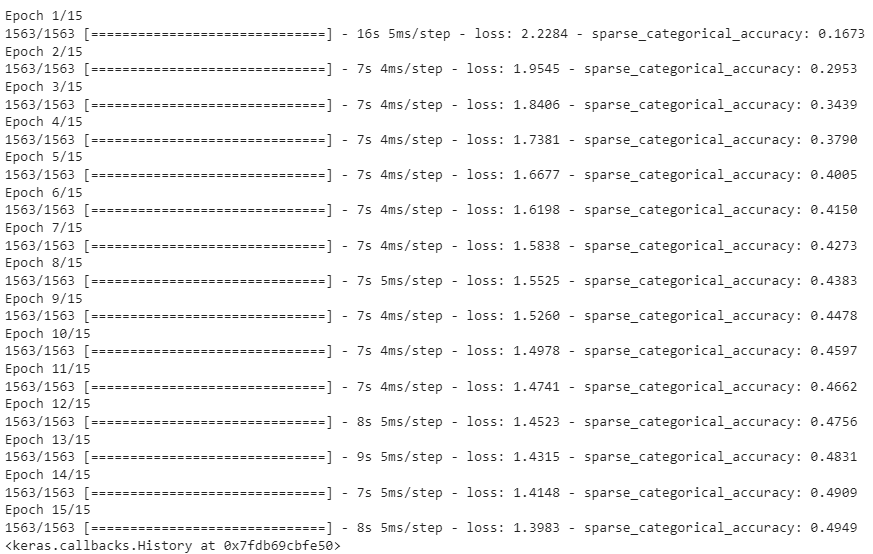
**Adaptive Gradient Descent**

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

model.fit(x\_train,y\_train,epochs=15)

test\_loss, test\_accuracy = model.evaluate(x\_test, y\_test)

print("Test accuracy: {}".format(test\_accuracy))

test\_loss, test\_accuracy = model.evaluate(x\_test, y\_test)

313/313 [==============================] - 1s 3ms/step - loss: 1.3510 - sparse\_categorical\_accuracy: 0.5214

Test accuracy: 0.521399974822998

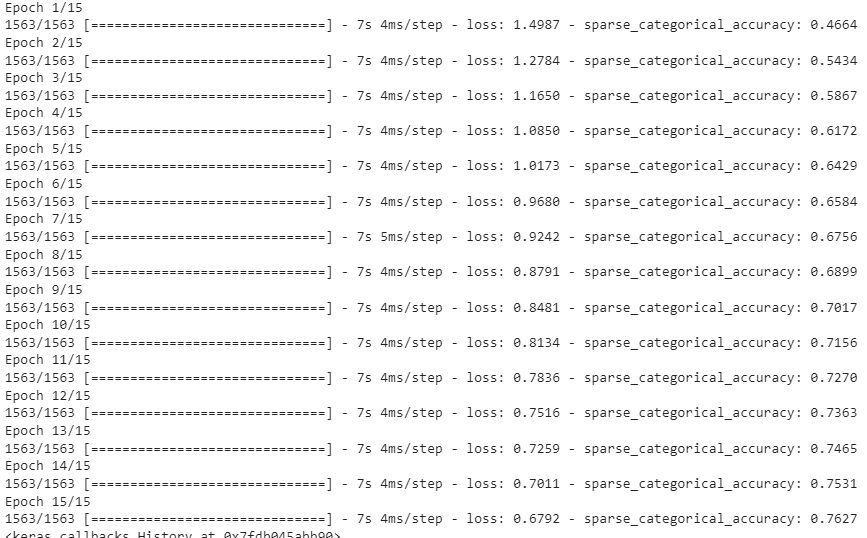
**Stochastic Gradient Descent**

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

model.fit(x\_train,y\_train,epochs=15)

test\_loss, test\_accuracy = model.evaluate(x\_test, y\_test)

print("Test accuracy: {}".format(test\_accuracy))

****

313/313 [==============================] - 1s 3ms/step - loss: 0.7533 - sparse\_categorical\_accuracy: 0.7464

Test accuracy: 0.746399998664856

**RESULT:**

Thus, the python program to implement image classification in Convolutional Neural Network using CIFAR10 dataset with various optimizers was executed successfully.

|  |  |
| --- | --- |
| **EX.NO:12** | **HEADLINE GENERATION USING RNN** |
|  |

**AIM:**

To generate news headlines from the given documents using RNN

**ALGORITHM:**

Step 1: Create a python notebook.

Step 2: Load the dataset

Step 3: Tokenize the data in the dataset

Step 4: Represent the tokens as set of sequence

Step 5: Construct the network with embedding and LSTM layers

Step 6: Train the model for 30 epochs and obtain the test accuracy.

Step 7: Generate headlines from the trained data

**PROGRAM:**

import os

import pandas as pd

nyt\_dir = 'data/nyt\_dataset/articles/'

all\_headlines = []

for filename in os.listdir(nyt\_dir):

if 'Articles' in filename:

# Read in all the data from the CSV file

headlines\_df = pd.read\_csv(nyt\_dir + filename)

# Add all of the headlines to our list

all\_headlines.extend(list(headlines\_df.headline.values))

len(all\_headlines)

all\_headlines[:20]

# Remove all headlines with the value of "Unknown"

all\_headlines = [h for h in all\_headlines if h != "Unknown"]

len(all\_headlines)

all\_headlines[:20]

tf.keras.preprocessing.text.Tokenizer(

num\_words=None, filters='!"#$%&()\*+,-./:;<=>?@[\\]^\_`{|}~\t\n', lower=True,

split=' ', char\_level=False, oov\_token=None, document\_count=0, \*\*kwargs

)

from tensorflow.keras.preprocessing.text import Tokenizer

# Tokenize the words in our headlines

tokenizer = Tokenizer()

tokenizer.fit\_on\_texts(all\_headlines)

total\_words = len(tokenizer.word\_index) + 1

print('Total words: ', total\_words)

tokenizer.texts\_to\_sequences(['a','man','a','plan','a','canal','panama'])

# Convert data to sequence of tokens

input\_sequences = []

for line in all\_headlines:

# Convert our headline into a sequence of tokens

token\_list = tokenizer.texts\_to\_sequences([line])[0]

# Create a series of sequences for each headline

for i in range(1, len(token\_list)):

partial\_sequence = token\_list[:i+1]

input\_sequences.append(partial\_sequence)

print(tokenizer.sequences\_to\_texts(input\_sequences[:5]))

input\_sequences[:5]

from tensorflow.keras.preprocessing.sequence import pad\_sequences

import numpy as np

# Determine max sequence length

max\_sequence\_len = max([len(x) for x in input\_sequences])

# Pad all sequences with zeros at the beginning to make them all max length

input\_sequences = np.array(pad\_sequences(input\_sequences, maxlen=max\_sequence\_len, padding='pre'))

input\_sequences[0]

# Predictors are every word except the last

predictors = input\_sequences[:,:-1]

# Labels are the last word

labels = input\_sequences[:,-1]

labels[:5]

from tensorflow.keras import utils

labels = utils.to\_categorical(labels, num\_classes=total\_words)

from tensorflow.keras.layers import Embedding, LSTM, Dense, Dropout

from tensorflow.keras.models import Sequential

# Input is max sequence length - 1, as we've removed the last word for the label

input\_len = max\_sequence\_len - 1

model = Sequential()

# Add input embedding layer

model.add(Embedding(total\_words, 10, input\_length=input\_len))

# Add LSTM layer with 100 units

model.add(LSTM(100))

model.add(Dropout(0.1))

# Add output layer

model.add(Dense(total\_words, activation='softmax'))

model.compile(loss='categorical\_crossentropy', optimizer='adam')

model.fit(predictors, labels, epochs=30, verbose=1)

Output:

9335

['My Beijing: The Sacred City', '6 Million Riders a Day, 1930s Technology', 'Seeking a Cross-Border Conference', 'Questions for: ‘Despite the “Yuck Factor,” Leeches Are Big in Russian Medicine’', 'Who Is a ‘Criminal’?', 'An Antidote to Europe’s Populism', 'The Cost of a Speech', 'Degradation of the Language', 'On the Power of Being Awful', 'Trump Garbles Pitch on a Revised Health Bill', 'What’s Going On in This Picture? | May 1, 2017', 'Unknown', 'When Patients Hit a Medical Wall', 'Unknown', 'For Pregnant Women, Getting Serious About Whooping Cough', 'Unknown', 'New York City Transit Reporter in Wonderland: Riding the London Tube', 'How to Cut an Avocado Without Cutting Yourself', 'In Fictional Suicide, Health Experts Say They See a Real Cause for Alarm', 'Claims of Liberal Media Bias Hit ESPN, Too']

8603

Epoch 1/30

1666/1666 [==============================] - 8s 5ms/step - loss: 7.8916

Epoch 2/30

1666/1666 [==============================] - 8s 5ms/step - loss: 7.4797

Epoch 3/30

1666/1666 [==============================] - 9s 5ms/step - loss: 7.2874

Epoch 4/30

1666/1666 [==============================] - 8s 5ms/step - loss: 7.0613

Epoch 5/30

1666/1666 [==============================] - 8s 5ms/step - loss: 6.8260

Epoch 6/30

1666/1666 [==============================] - 8s 5ms/step - loss: 6.5773

Epoch 7/30

1666/1666 [==============================] - 8s 5ms/step - loss: 6.3224

Epoch 8/30

1666/1666 [==============================] - 8s 5ms/step - loss: 6.0801

Epoch 9/30

1666/1666 [==============================] - 8s 5ms/step - loss: 5.8491

Epoch 10/30

1666/1666 [==============================] - 8s 5ms/step - loss: 5.6249

Epoch 11/30

1666/1666 [==============================] - 9s 5ms/step - loss: 5.4119

Epoch 12/30

1666/1666 [==============================] - 8s 5ms/step - loss: 5.2020

Epoch 13/30

1666/1666 [==============================] - 8s 5ms/step - loss: 5.0042

Epoch 14/30

1666/1666 [==============================] - 8s 5ms/step - loss: 4.8211

Epoch 15/30

1666/1666 [==============================] - 8s 5ms/step - loss: 4.6421

Epoch 16/30

1666/1666 [==============================] - 8s 5ms/step - loss: 4.4760

Epoch 17/30

1666/1666 [==============================] - 8s 5ms/step - loss: 4.3206

Epoch 18/30

1666/1666 [==============================] - 8s 5ms/step - loss: 4.1733

Epoch 19/30

1666/1666 [==============================] - 9s 5ms/step - loss: 4.0349

Epoch 20/30

1666/1666 [==============================] - 8s 5ms/step - loss: 3.9000

Epoch 21/30

1666/1666 [==============================] - 8s 5ms/step - loss: 3.7829

Epoch 22/30

1666/1666 [==============================] - 8s 5ms/step - loss: 3.6709

Epoch 23/30

1666/1666 [==============================] - 8s 5ms/step - loss: 3.5615

Epoch 24/30

1666/1666 [==============================] - 8s 5ms/step - loss: 3.4667

Epoch 25/30

1345/1666 [=======================>......] - ETA: 1s - loss: 3.3369

**RESULT:**

Thus, the python program to implement RNN was executed successfully.

**CONTENT BEYOND SYLLABUS**

|  |
| --- |
| **AUTOENCODERS** |

**AIM:**

To implement autoencoders over image dataset.

**ALGORITHM:**

Step 1: Create a python notebook.

Step 2: Load the dataset

Step 3: Define encoder model

Step 4: Define decoder model to identify real and fake images

Step 5: Train the encoder for 50 epochs and obtain the test accuracy.

Step 7: Reconstruct the image from the bottlneck

**PROGRAM:**

import keras

from keras import layers

# This is the size of our encoded representations

encoding\_dim = 32  # 32 floats -> compression of factor 24.5, assuming the input is 784 floats

# This is our input image

input\_img = keras.Input(shape=(784,))

# "encoded" is the encoded representation of the input

encoded = layers.Dense(encoding\_dim, activation='relu')(input\_img)

# "decoded" is the lossy reconstruction of the input

decoded = layers.Dense(784, activation='sigmoid')(encoded)

# This model maps an input to its reconstruction

autoencoder = keras.Model(input\_img, decoded)

encoder = keras.Model(input\_img, encoded)

# This is our encoded (32-dimensional) input

encoded\_input = keras.Input(shape=(encoding\_dim,))

# Retrieve the last layer of the autoencoder model

decoder\_layer = autoencoder.layers[-1]

# Create the decoder model

decoder = keras.Model(encoded\_input, decoder\_layer(encoded\_input))

autoencoder.compile(optimizer='adam', loss='binary\_crossentropy')

from keras.datasets import mnist

import numpy as np

(x\_train, \_), (x\_test, \_) = mnist.load\_data()

x\_train = x\_train.astype('float32') / 255.

x\_test = x\_test.astype('float32') / 255.

x\_train = x\_train.reshape((len(x\_train), np.prod(x\_train.shape[1:])))

x\_test = x\_test.reshape((len(x\_test), np.prod(x\_test.shape[1:])))

print(x\_train.shape)

print(x\_test.shape)

#Now let's train our autoencoder for 50 epochs:

autoencoder.fit(x\_train, x\_train,

                epochs=50,

                batch\_size=256,

                shuffle=True,

                validation\_data=(x\_test, x\_test))

encoded\_imgs = encoder.predict(x\_test)

decoded\_imgs = decoder.predict(encoded\_imgs)

import matplotlib.pyplot as plt

n = 10  # How many digits we will display

plt.figure(figsize=(20, 4))

for i in range(n):

    # Display original

    ax = plt.subplot(2, n, i + 1)

    plt.imshow(x\_test[i].reshape(28, 28))

    plt.gray()

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

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

    # Display reconstruction

    ax = plt.subplot(2, n, i + 1 + n)

    plt.imshow(decoded\_imgs[i].reshape(28, 28))

    plt.gray()

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

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

plt.show()

OUTPUT:

(60000, 784)

(10000, 784)

235/235 [==============================] - 3s 12ms/step - loss: 0.0933 - val\_loss: 0.0920

Epoch 23/50

235/235 [==============================] - 3s 12ms/step - loss: 0.0933 - val\_loss: 0.0919

Epoch 24/50

235/235 [==============================] - 3s 13ms/step - loss: 0.0932 - val\_loss: 0.0920

Epoch 25/50

235/235 [==============================] - 3s 12ms/step - loss: 0.0931 - val\_loss: 0.0919

Epoch 26/50

235/235 [==============================] - 3s 12ms/step - loss: 0.0931 - val\_loss: 0.0918

Epoch 27/50

235/235 [==============================] - 3s 12ms/step - loss: 0.0930 - val\_loss: 0.0919

Epoch 28/50

235/235 [==============================] - 4s 16ms/step - loss: 0.0930 - val\_loss: 0.0920

Epoch 29/50

235/235 [==============================] - 3s 13ms/step - loss: 0.0930 - val\_loss: 0.0918

Epoch 30/50

235/235 [==============================] - 3s 12ms/step - loss: 0.0930 - val\_loss: 0.0918

Epoch 31/50

235/235 [==============================] - 3s 12ms/step - loss: 0.0929 - val\_loss: 0.0917

Epoch 32/50

235/235 [==============================] - 3s 12ms/step - loss: 0.0929 - val\_loss: 0.0917

Epoch 33/50

235/235 [==============================] - 3s 12ms/step - loss: 0.0929 - val\_loss: 0.0918

Epoch 34/50

235/235 [==============================] - 3s 12ms/step - loss: 0.0929 - val\_loss: 0.0917

Epoch 35/50

235/235 [==============================] - 3s 12ms/step - loss: 0.0928 - val\_loss: 0.0917

Epoch 36/50

235/235 [==============================] - 3s 12ms/step - loss: 0.0928 - val\_loss: 0.0916

Epoch 37/50

235/235 [==============================] - 3s 12ms/step - loss: 0.0928 - val\_loss: 0.0917

Epoch 38/50

235/235 [==============================] - 3s 12ms/step - loss: 0.0928 - val\_loss: 0.0916

Epoch 39/50

235/235 [==============================] - 3s 12ms/step - loss: 0.0928 - val\_loss: 0.0916

Epoch 40/50

235/235 [==============================] - 3s 12ms/step - loss: 0.0928 - val\_loss: 0.0915

Epoch 41/50

235/235 [==============================] - 3s 12ms/step - loss: 0.0927 - val\_loss: 0.0916

Epoch 42/50

235/235 [==============================] - 3s 12ms/step - loss: 0.0927 - val\_loss: 0.0916

Epoch 43/50

235/235 [==============================] - 3s 12ms/step - loss: 0.0927 - val\_loss: 0.0915

Epoch 44/50

235/235 [==============================] - 3s 12ms/step - loss: 0.0927 - val\_loss: 0.0916

Epoch 45/50

235/235 [==============================] - 3s 12ms/step - loss: 0.0927 - val\_loss: 0.0915

Epoch 46/50

235/235 [==============================] - 3s 12ms/step - loss: 0.0927 - val\_loss: 0.0915

Epoch 47/50

235/235 [==============================] - 3s 12ms/step - loss: 0.0927 - val\_loss: 0.0915

Epoch 48/50

235/235 [==============================] - 3s 12ms/step - loss: 0.0927 - val\_loss: 0.0916

Epoch 49/50

235/235 [==============================] - 4s 17ms/step - loss: 0.0927 - val\_loss: 0.0916

Epoch 50/50

235/235 [==============================] - 3s 12ms/step - loss: 0.0927 - val\_loss: 0.0916

<keras.callbacks.History at 0x7fa7b0295c90>

313/313 [==============================] - 1s 1ms/step

313/313 [==============================] - 1s 2ms/step

****

**RESULT:**

Thus, the python program to implement autoencoder was executed successfully.