

# Lab Manual of Machine Learning [CS-602]

**B. Tech. VI Semester Jan - June 2023**

# Department of Computer Science and Information Technology

**Submitted to Submitted By**

Dr. Shilpa Bhalerao Naman Mehta

HOD, CSIT Dept. 0827CI201114

**ACROPOLIS INSTITUTE OF TECHNOLOGY & RESEARCH, INDORE**

**Department of Computer Science and Information Technology**

**Certificate**

This is to certify that the experimental work entered in this journal as per the B Tech III year syllabus prescribed by the RGPV was done by Mr. Naman Mehta BTech VI semester CI in the Machine Learning Laboratory of this institute during the academic year Jan June 2023

Signature of Faculty

INDEX PAGE

Programs to be uploaded on Github Github Link:

|  |  |  |  |
| --- | --- | --- | --- |
| Experiment No | Program | Commit  date ( in Github) | Sign of faculty |
| 1 | Python Basic Programming including Python Data Structures such as List, Tuple, Strings, Dictionary, Lambda Functions, Python  Classes and Objects and Python Libraries such as Numpy, Pandas, Mat plotlib etc. |  |  |
| 2 | Python List Comprehension with examples |  |  |
| 3 | Basic of Numpy, Pandas and Matplotlib |  |  |
| 4 | Brief Study of Machine Learning Frameworks such as Open CV, Scikit Learn,  Keras, Tensorflow etc. |  |  |
| 5 | For a given set of training data examples stored in a .CSV file, implement and demonstrate the scratch Implementation  of **Linear Regression Algorithm** |  |  |
| 6 | For a given set of training data examples stored in a .CSV file, implement and demonstrate the Implementation of **Linear Regression Algorithm** Linear Regression  using Python library (for any given CSV dataset ) |  |  |
| 7 | For a given set of training data examples stored in a .CSV file, implement and demonstrate the scratch Implementation for binary classification using **Logistic**  **Regression Algorithm** |  |  |
| 8 | Build an **Artificial Neural Network (ANN)** by implementing the Backpropagation algorithm and test the same using MNIST **Handwritten Digit Multiclass classification** data sets with use of use of batch normalization, early stopping and dropout |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| 9 | Build an **Artificial Neural Network** by implementing the Backpropagation algorithm and test the same using **CIFAR 100 Multiclass classification** data sets with use  of use of batch normalization, early stopping and drop out |  |  |
| 10 | ANN implementation use of batch normalization, early stopping and drop out  (For Image Dataset such as Covid Dataset) |  |  |
| 11 | Build a **Convolutional Neural Network** by implementing the Backpropagation algorithm and test the same using **MNIST Handwritten Digit Multiclass classification** data sets. |  |  |
| 12 | Build a **Convolutional Neural Network** by implementing the Backpropagation algorithm and test the same using **CIFAR 100**  **Multiclass classification** data sets. |  |  |
| 13 | Implementation of Transfer Learning (VGG  16) |  |  |
| 14 | Implementation of RNN |  |  |

EXPERIMENT 1

AIM:

Understand Python Basic Programming including Python Data Structures such as List, Tuple, Strings, Dictionary, Lambda Functions, Python Classes and Objects and Python Libraries such as Numpy, Pandas, Matplotlib etc.

PROGRAM/ IPYNB FILE:

**Python Basics**

**Printing**

To print a string in Python, use the print() function. For example:

print("Hello, world!")

**Variables**

In Python, you can create variables without specifying a type. Simply assign a value to a variable name, and Python will infer the type. For example:

x = 5

y = "hello"

**Control flow**

Python has several control flow structures, including if, while, and for loops. For example:

if x > 10:

print("x is greater than 10")

else:

print("x is less than or equal to 10")

while x < 100:

print(x)

x = x \* 2

for i in range(10):

print(i)

**Data Structures**

Python has several built-in data structures, including lists, tuples, strings, and dictionaries.

**String**

A string is a sequence of characters. To create a string, surround the characters with either single or double quotes. For example:

my\_string = "hello, world!"

You can access individual characters of a string using square brackets and an index, just like a list.

**Lists**

A list is a mutable sequence of values. To create a list, use square brackets and separate the values with commas. For example:

my\_list = [1, 2, 3, 4, 5]

You can access individual elements of a list using square brackets and an index. For example:

print(my\_list[0]) prints 1

You can also slice a list to get a subset of its elements. For example:

print(my\_list[1:3]) prints [2, 3]

**Tuple**

A tuple is an immutable sequence of values. To create a tuple, use parentheses and separate the values with commas. For example:

my\_tuple = (1, 2, 3, 4, 5)

**Dictionaries**

A dictionary is a collection of key-value pairs. To create a dictionary, use curly braces and separate the key-value pairs with commas. For example:

my\_dict = {"apple": 3, "banana": 2, "orange": 1}

You can access individual values of a dictionary using square brackets and a key. For example:

print(my\_dict["apple"]) prints 3

**Classes and Objects**

Python is an object-oriented language, which means that you can define your own classes and create objects from those classes. To define a class, use the class keyword. For example:

class Person:

def \_\_init\_\_(self, name, age):

self.name = name

self.age = age

def say\_hello(self):

print("Hello, my name is", self.name)

This Person class has a constructor (\_\_init\_\_) that takes a name and age and initializes the corresponding instance variables. It also has a method (say\_hello) that prints a greeting.

To create an instance of a class, use the class name followed by parentheses. For example:

person1 = Person("Alice", 25)

person2 = Person("Bob", 30)

print(person1.name) prints "Alice"

person2.say\_hello() prints "Hello, my name is Bob"

**Lambda Functions**

A lambda function is a small, anonymous function. It can take any number of arguments but can only have one expression. Lambda functions are useful for writing short, simple functions without defining a separate function. For example:

my\_lambda = lambda x: x \* 2

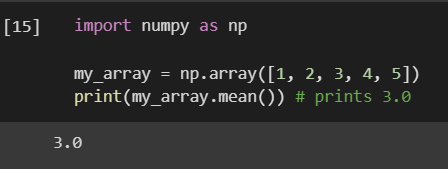
print(my\_lambda(3)) prints 6

**Python Libraries**

Python has a vast ecosystem of third-party libraries that can help you accomplish a wide variety of tasks. Here are a few popular libraries:

**NumPy**

NumPy is a library for working with arrays of numerical data. It provides efficient and fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation, and much more.



**Pandas**

Pandas is a library for working with data sets. It provides data structures for efficiently storing and manipulating large data sets and tools for cleaning, exploring, and analyzing data.

Text, website

Description automatically generated

**Matplotlib**

Matplotlib is a library for creating plots and visualizations. It provides a wide range of plotting tools, including line plots, scatter plots, bar plots, histograms, and more.

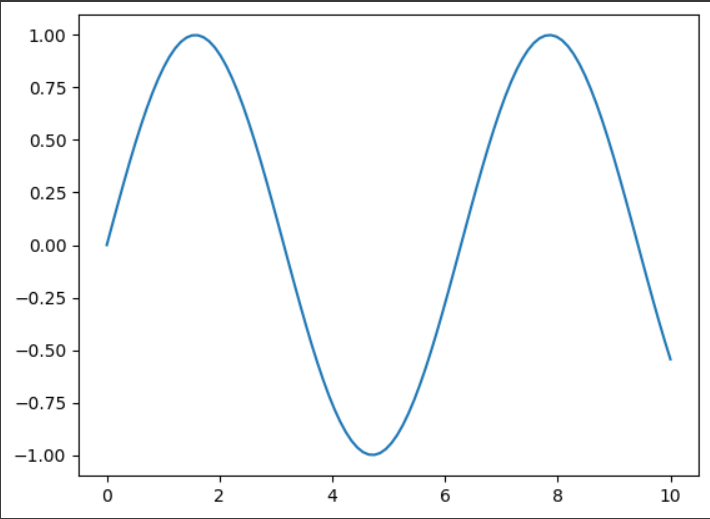
import matplotlib.pyplot as plt

x = np.linspace(0, 10, 100)

y = np.sin(x)

plt.plot(x, y)

plt.show() displays the plot



EXPERIMENT 2

AIM:

Understand Python List Comprehension with examples

PYTHON FILE / IPYNB:

Python List Comprehension is a concise way of creating a new list by performing some operation on each item of an existing list or iterable. It combines loops and conditional statements into a single line of code.

Here is the general syntax of a list comprehension in Python:

new\_list = [expression for item in iterable if condition]

The `expression` is the operation performed on each item in the iterable, `item` represents each item in the iterable, and `iterable` is the original list, tuple, set, or any other iterable object. The `if condition` is an optional condition that filters the items based on a certain criterion.

Here are some examples of Python list comprehension:

Example 1: Creating a new list by squaring each element of an existing list

numbers = [1, 2, 3, 4, 5]

squares = [x \*\* 2 for x in numbers]

print(squares) # Output: [1, 4, 9, 16, 25]

Example 2: Creating a new list by filtering out even numbers from an existing list

numbers = [1, 2, 3, 4, 5]

odds = [x for x in numbers if x % 2 != 0]

print(odds) # Output: [1, 3, 5]

Example 3: Creating a new list by flattening a 2D list

matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

flatten\_matrix = [num for row in matrix for num in row]

print(flatten\_matrix) # Output: [1, 2, 3, 4, 5, 6, 7, 8, 9]

Example 4: Creating a new list of even and odd numbers from an existing list

numbers = [1, 2, 3, 4, 5]

even\_odd = ["even" if x % 2 == 0 else "odd" for x in numbers]

print(even\_odd) # Output: ["odd", "even", "odd", "even", "odd"]

Example 5: Creating a new list of tuples from two existing lists

fruits = ["apple", "banana", "cherry"]

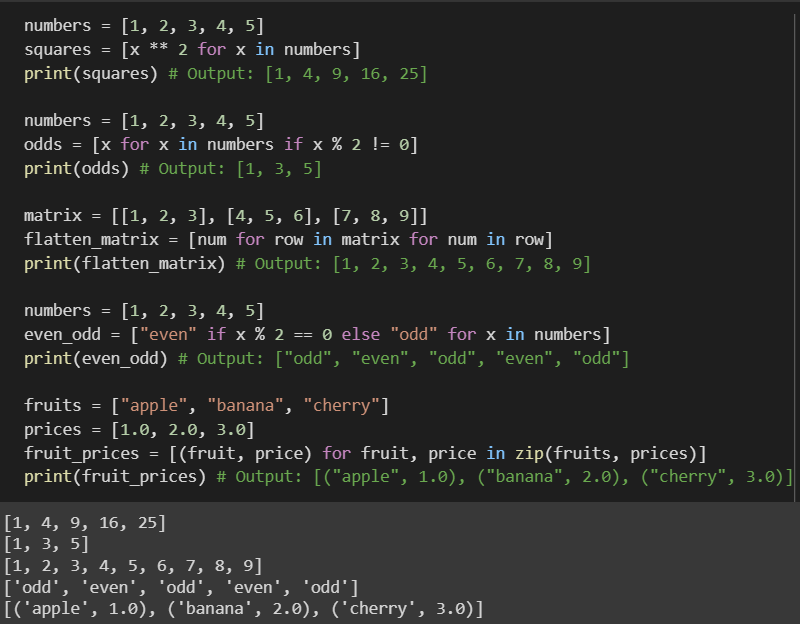
prices = [1.0, 2.0, 3.0]

fruit\_prices = [(fruit, price) for fruit, price in zip(fruits, prices)]

print(fruit\_prices) # Output: [("apple", 1.0), ("banana", 2.0), ("cherry", 3.0)]

List comprehension is a powerful technique in Python that allows you to create new lists in a concise and efficient way. By using list comprehension, you can avoid writing long and complex loops, making your code more readable and easier to maintain.

RESULT:



EXPERIMENT 3

AIM:

Basic of Numpy, Pandas and Matplotlib

PROGRAM/ IPYNB FILE:

**NumPy**

NumPy stands for "Numerical Python". It is a library for Python that provides support for large, multidimensional arrays and matrices, along with a large collection of highlevel mathematical functions to operate on these arrays. It is the fundamental package for scientific computing in Python.

Here are some of the key features of NumPy:

- Efficient handling of multi-dimensional arrays and matrices

- Fast mathematical operations on arrays, including linear algebra and Fourier

transform

- Random number generation

- Integration with C, C++, and Fortran code

Here is an example of creating a NumPy array:

Creating arrays

import numpy as np

# Creating a 1D array

a = np.array([1, 2, 3])

print(a) # Output: [1 2 3]

# Creating a 2D array

b = np.array([[1, 2], [3, 4]])

print(b) # Output: [[1 2]

         #          [3 4]]

# Creating an array of zeros

c = np.zeros((2, 3))

print(c) # Output: [[0. 0. 0.]

         #          [0. 0. 0.]]

# Creating an array of ones

d = np.ones((2, 3))

print(d) # Output: [[1. 1. 1.]

         #          [1. 1. 1.]]

# Creating a range of values

e = np.arange(0, 10, 2)

print(e) # Output: [0 2 4 6 8]

Basic operations

import numpy as np

# Element-wise addition

a = np.array([1, 2, 3])

b = np.array([4, 5, 6])

c = a + b

print(c) # Output: [5 7 9]

# Element-wise multiplication

a = np.array([1, 2, 3])

b = np.array([4, 5, 6])

c = a \* b

print(c) # Output: [ 4 10 18]

# Dot product

a = np.array([1, 2])

b = np.array([3, 4])

c = np.dot(a, b)

print(c) # Output: 11

# Transpose

a = np.array([[1, 2], [3, 4]])

b = a.T

print(b) # Output: [[1 3]

         #          [2 4]]

Statistical functions

import numpy as np

# Mean

a = np.array([1, 2, 3])

b = np.mean(a)

print(b) # Output: 2.0

# Standard deviation

a = np.array([1, 2, 3])

b = np.std(a)

print(b) # Output: 0.816496580927726

# Max

a = np.array([1, 2, 3])

b = np.max(a)

print(b) # Output: 3

# Min

a = np.array([1, 2, 3])

b = np.min(a)

print(b) # Output: 1

Indexing and slicing arrays

import numpy as np

# Indexing a 1D array

a = np.array([1, 2, 3])

print(a[0]) # Output: 1

# Slicing a 1D array

a = np.array([1, 2, 3, 4, 5])

print(a[1:4]) # Output: [2 3 4]

# Indexing a 2D array

a = np.array([[1, 2], [3, 4]])

print(a[0, 1]) # Output: 2

# Slicing a 2D array

a = np.array([[1, 2], [3, 4], [5, 6]])

print(a[1:, :]) # Output: [[3 4]

                #          [5 6]]

Reshaping arrays

import numpy as np

# Reshaping a 1D array

a = np.array([1, 2, 3, 4, 5, 6])

b = a.reshape((2, 3))

print(b) # Output: [[1 2 3]

         #          [4 5 6]]

# Reshaping a 2D array

a = np.array([[1, 2], [3, 4]])

b = a.reshape((4,))

print(b) # Output: [1 2 3 4]

# Flattening an array

a = np.array([[1, 2], [3, 4]])

b = a.flatten()

print(b) # Output: [1 2 3 4]

Concatenating arrays

import numpy as np

# Concatenating 1D arrays

a = np.array([1, 2, 3])

b = np.array([4, 5, 6])

c = np.concatenate((a, b))

print(c) # Output: [1 2 3 4 5 6]

# Concatenating 2D arrays

a = np.array([[1, 2], [3, 4]])

b = np.array([[5, 6], [7, 8]])

c = np.concatenate((a, b), axis=0)

print(c) # Output: [[1 2]

         #          [3 4]

         #          [5 6]

         #          [7 8]]

# Stacking arrays

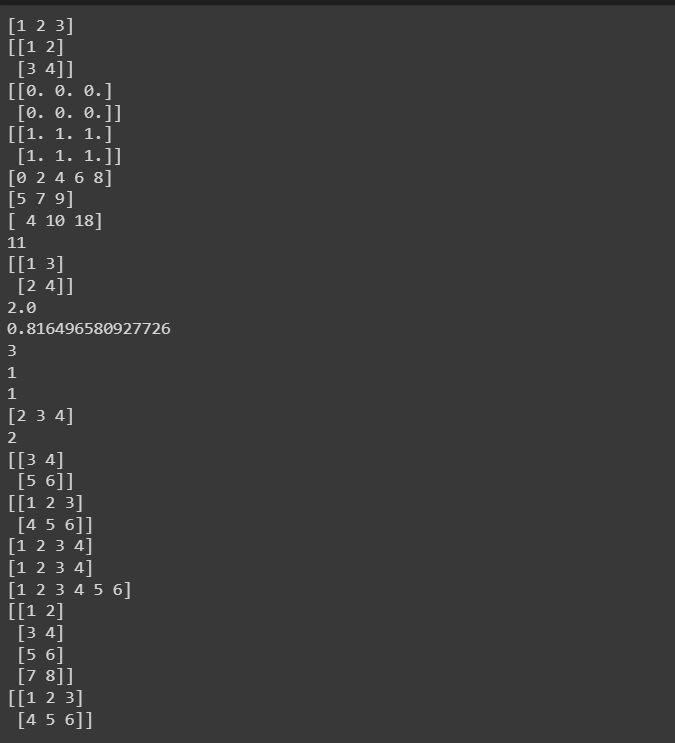
a = np.array([1, 2, 3])

b = np.array([4, 5, 6])

c = np.stack((a, b))

print(c) # Output: [[1 2 3]

         #          [4 5 6]]



**Pandas**

Pandas is a library for Python that provides highlevel data structures and functions  for working with structured data. It provides data structures like Series and DataFrame, which are optimized for data analysis, and allows for easy importing and exporting of data from a variety of sources.

Here are some of the key features of Pandas:

- Data structures for working with structured data, including time series data

- Data manipulation and cleaning tools

- Merge, join, and reshaping of data sets

- Data alignment and handling of missing data

Here is an example of creating a Pandas DataFrame:

Creating data structures

import pandas as pd

# Create a Series

s = pd.Series([1, 2, 3])

# Create a DataFrame

df = pd.DataFrame({'Name': ['Alice', 'Bob', 'Charlie'], 'Age': [25, 30, 35]})

Indexing and slicing

import pandas as pd

# Create a DataFrame

df = pd.DataFrame({'Name': ['Alice', 'Bob', 'Charlie'], 'Age': [25, 30, 35]})

# Indexing

print(df.loc[0, 'Name'])   # Output: 'Alice'

print(df.iloc[0, 0])       # Output: 'Alice'

# Slicing

print(df.iloc[:2, :])      # Output:

                           #       Name  Age

                           #    0  Alice   25

                           #    1    Bob   30

Data manipulation

import pandas as pd

# Create a DataFrame

df = pd.DataFrame({'Name': ['Alice', 'Bob', 'Charlie'], 'Age': [25, 30, 35]})

# Select a subset of columns

df2 = df[['Name']]

# Add a new column

df['Gender'] = ['F', 'M', 'M']

# Apply a function to a column

df['Age'] = df['Age'].apply(lambda x: x + 1)

# Group by a column and calculate mean

df.groupby('Gender')['Age'].mean()

Shape

Description automatically generated with medium confidence

**Matplotlib**

Matplotlib is a library for Python that provides tools for creating visualizations, such as line charts, scatter plots, and histograms. It is built on NumPy and provides an easy-to-use interface for creating high-quality plots.

Here are some of the key features of Matplotlib:

* Support for a wide variety of 2D and 3D plots
* Customizable and interactive visualizations
* Integration with Pandas and NumPy data structures
* High-quality output in a variety of formats, including PDF, SVG, and PNG

For Example:

Creating basic plots

import matplotlib.pyplot as plt

import numpy as np

# Creating a line plot

x = np.linspace(0, 10, 100)

y = np.sin(x)

plt.plot(x, y)

plt.show()

# Creating a scatter plot

x = np.random.rand(100)

y = np.random.rand(100)

plt.scatter(x, y)

plt.show()

# Creating a bar plot

x

Creating basic plots

# Creating a line plot

x = np.linspace(0, 10, 100)

y = np.sin(x)

plt.plot(x, y)

plt.show()

# Creating a scatter plot

x = np.random.rand(100)

y = np.random.rand(100)

plt.scatter(x, y)

plt.show()

# Creating a bar plot

x = ['A', 'B', 'C', 'D', 'E']

y = [10, 24, 36, 40, 62]

plt.bar(x, y)

plt.show()

Customizing plots

# Adding labels and titles to a plot

x = np.linspace(0, 10, 100)

y = np.sin(x)

plt.plot(x, y)

plt.xlabel('Time (s)')

plt.ylabel('Amplitude')

plt.title('Sine Wave')

plt.show()

# Changing the style of a plot

plt.style.use('ggplot')

plt.plot(x, y)

plt.show()

# Adding legends to a plot

x = np.linspace(0, 10, 100)

y1 = np.sin(x)

y2 = np.cos(x)

plt.plot(x, y1, label='Sine')

plt.plot(x, y2, label='Cosine')

plt.legend()

plt.show()

Saving plots

# Saving a plot to a file

x = np.linspace(0, 10, 100)

y = np.sin(x)

plt.plot(x, y)

plt.savefig('sine\_wave.png')

Subplots

# Creating subplots

x = np.linspace(0, 10, 100)

y1 = np.sin(x)

y2 = np.cos(x)

fig, ax = plt.subplots(nrows=2, ncols=1, figsize=(8, 6))

ax[0].plot(x, y1)

ax[1].plot(x, y2)

plt.show()

Histograms

# Creating a histogram

x = np.random.normal(size=1000)

plt.hist(x, bins=30)

plt.show()

Heatmaps

# Creating a heatmap

x = np.random.rand(10, 10)

plt.imshow(x, cmap='hot', interpolation='nearest')

plt.colorbar()

plt.show()

3D plotting

from mpl\_toolkits.mplot3d import Axes3D

# Creating a 3D scatter plot

fig = plt.figure()

ax = fig.add\_subplot(111, projection='3d')

x = np.random.rand(100)

y = np.random.rand(100)

z = np.random.rand(100)

ax.scatter(x, y, z)

plt.show()

# Creating a 3D surface plot

fig = plt.figure()

ax = fig.add\_subplot(111, projection='3d')

x = np.arange(-5, 5, 0.25)

y = np.arange(-5, 5, 0.25)

x, y = np.meshgrid(x, y)

r = np.sqrt(x\*\*2 + y\*\*2)

z = np.sin(r)

ax.plot\_surface(x, y, z)

plt.show()

Annotations and Text

# Adding text to a plot

x = np.linspace(0, 10, 100)

y = np.sin(x)

plt.plot(x, y)

plt.text(5, 0, 'Maximum', fontsize=12, ha='center', va='center')

plt.annotate('Inflection Point', xy=(np.pi/2, 1), xytext=(np.pi/2+1, 0.5),

arrowprops=dict(facecolor='black', shrink=0.05))

plt.show()

Subplots and Shared Axes

# Creating subplots with shared x-axis

x = np.linspace(0, 10, 100)

y1 = np.sin(x)

y2 = np.cos(x)

fig, (ax1, ax2) = plt.subplots(nrows=2, ncols=1, sharex=True, figsize=(8, 6))

ax1.plot(x, y1)

ax2.plot(x, y2)

plt.show()

# Creating subplots with shared y-axis

x = np.linspace(0, 10, 100)

y1 = np.sin(x)

y2 = np.cos(x)

fig, (ax1, ax2) = plt.subplots(nrows=1, ncols=2, sharey=True, figsize=(8, 6))

ax1.plot(x, y1)

ax2.plot(x, y2)

plt.show()

Multiple Plots on the Same Axes

# Creating multiple plots on the same axes

x = np.linspace(0, 10, 100)

y1 = np.sin(x)

y2 = np.cos(x)

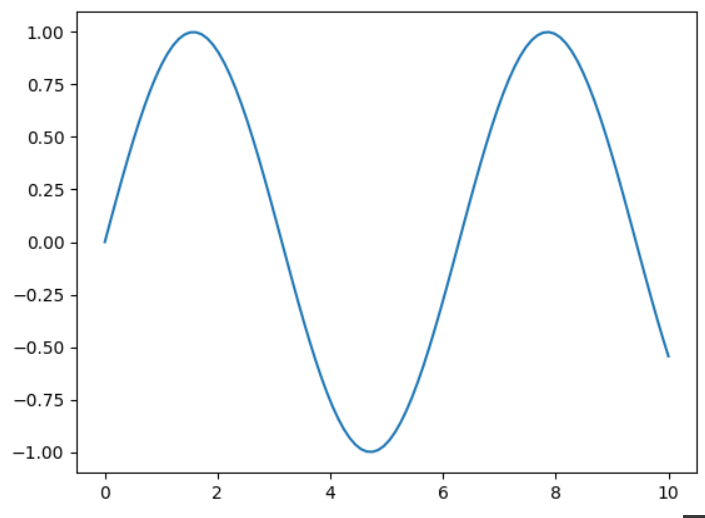
plt.plot(x, y1, label='Sine')

plt.plot(x, y2, label='Cosine')

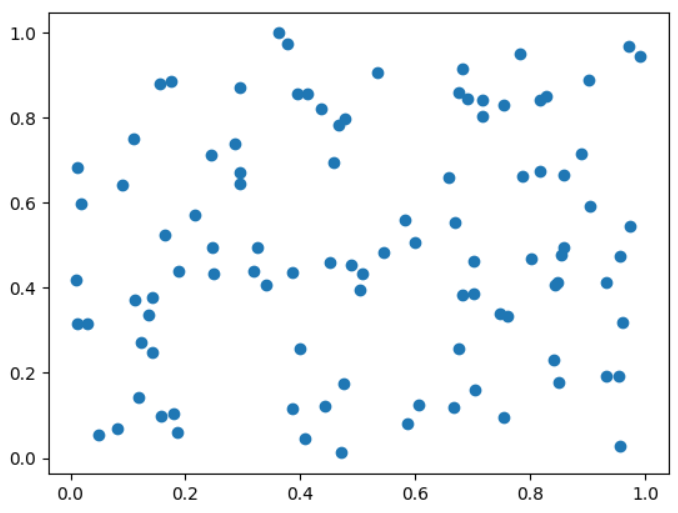
plt.plot(x, y1\*y2, label='Product')

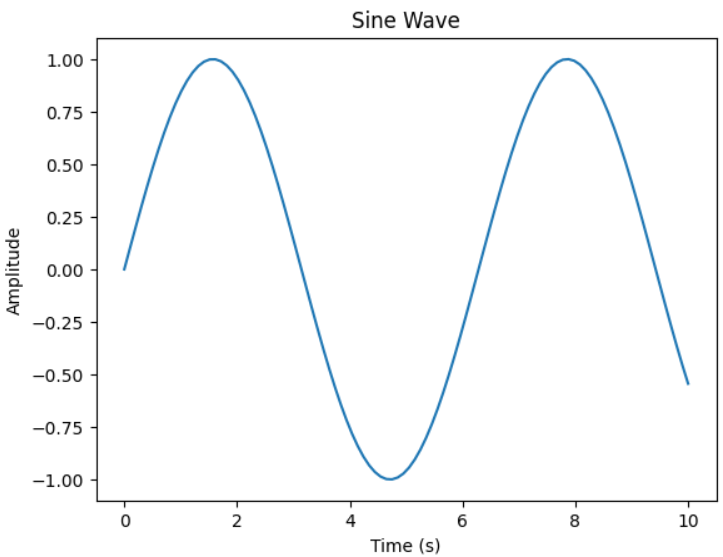
plt.legend()

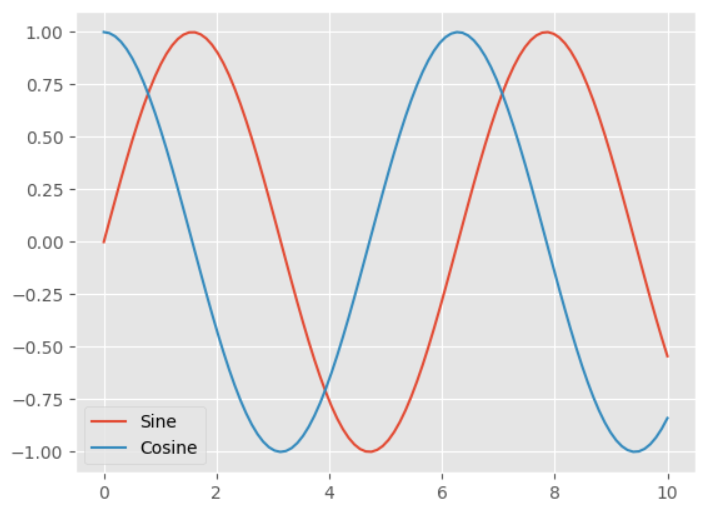
plt.show()

 Chart, scatter chart

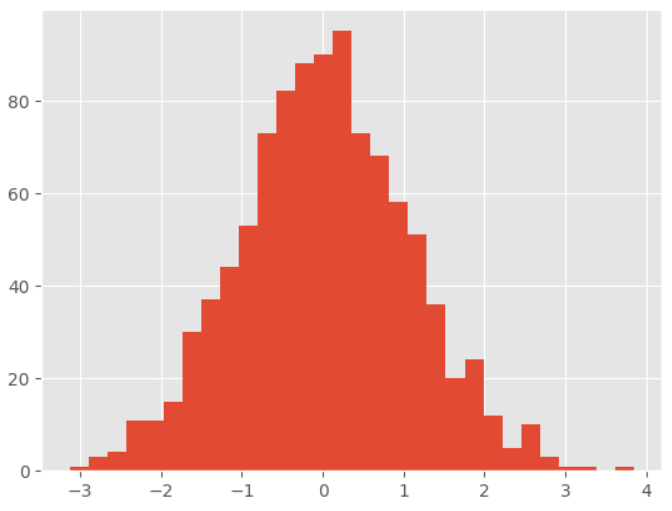
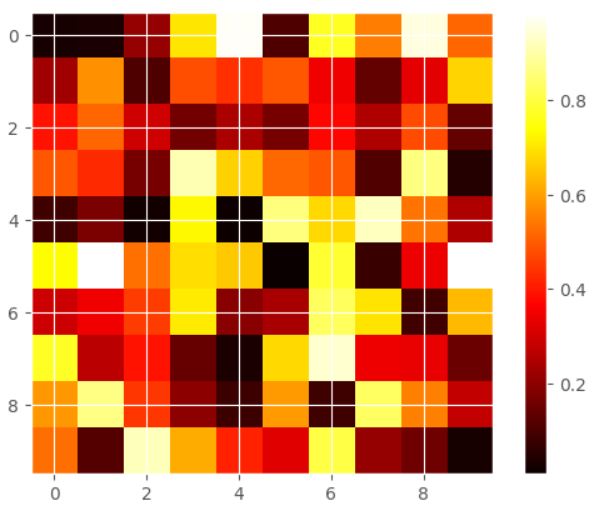
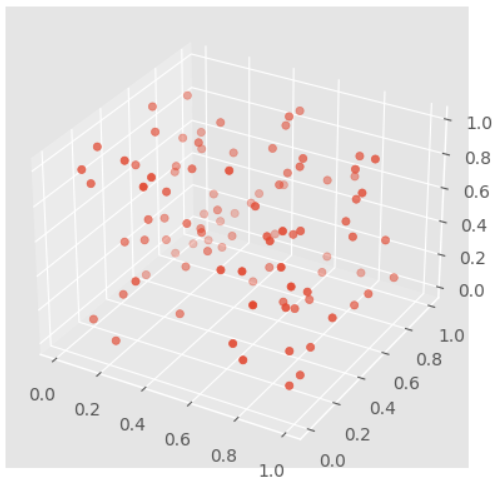
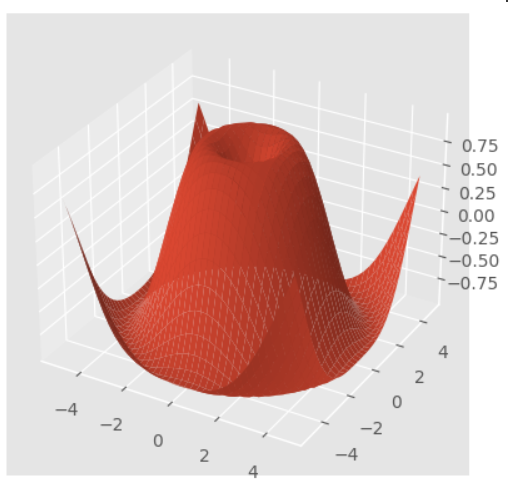
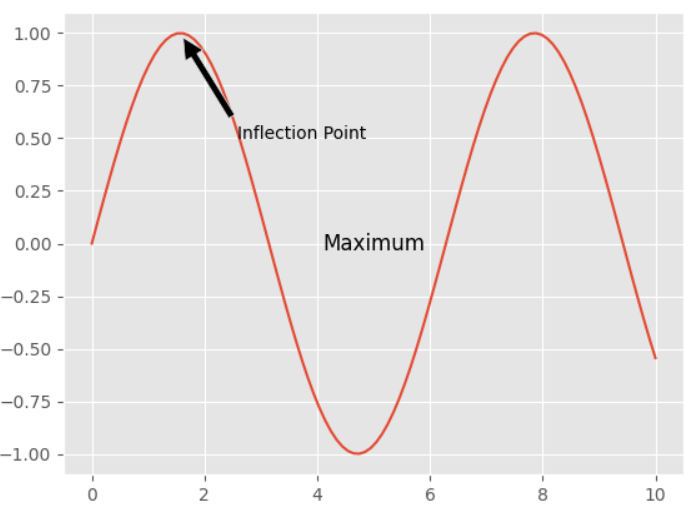
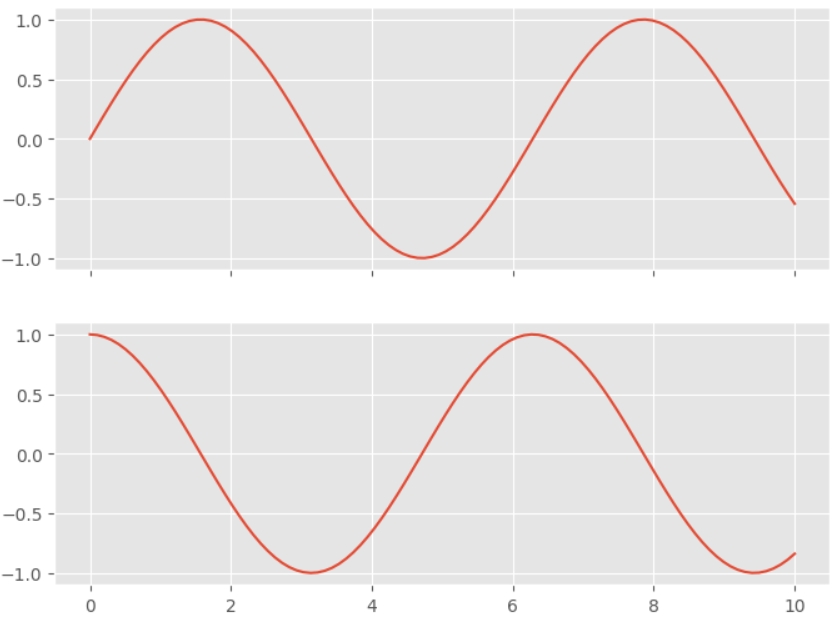
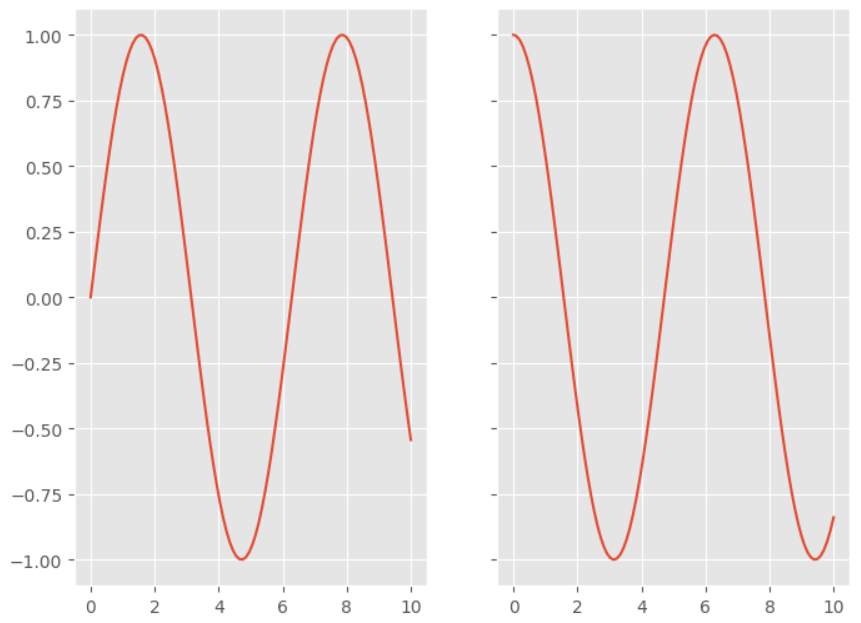
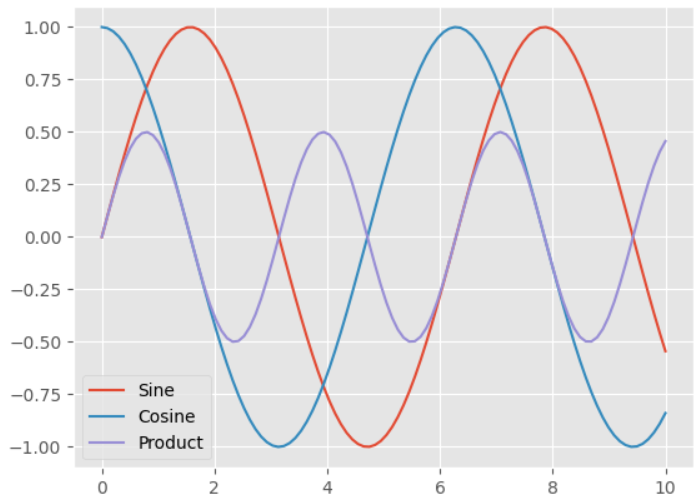
Description automatically generated Chart

Description automatically generated  Chart, bar chart

Description automatically generated  Chart, line chart

Description automatically generated  Chart, line chart

Description automatically generated Chart, line chart

Description automatically generated       

EXPERIMENT 4

AIM:

Brief Study of Machine Learning Framework such as Open CV, Scikit Learn, Keras, Tensorflow etc

Program / Ipynb:

Machine Learning frameworks are libraries or software tools that allow developers to build machine learning models with ease. These frameworks provide various functionalities such as data preprocessing, model building, and evaluation. In this brief study, we will discuss some popular Machine Learning frameworks such as OpenCV, Scikit-Learn, Keras, and TensorFlow.

**OpenCV**

OpenCV is a popular Computer Vision library that provides various functions for image and video processing. It is written in C++ but also provides a Python interface. OpenCV can be used for tasks such as object detection, image recognition, and face detection. OpenCV also includes pre-trained models for face detection and recognition. Some of the key features of OpenCV are:

- Image and Video processing

- Object Detection

- Face Detection

- Machine Learning algorithms

- Computer Vision applications

**Scikit-Learn**

Scikit-Learn is a popular Machine Learning library that provides various functions for data preprocessing, model building, and evaluation. It is written in Python and is built on top of NumPy, SciPy, and matplotlib. Scikit-Learn can be used for tasks such as regression, classification, and clustering. Some of the key features of Scikit-Learn are:

- Data Preprocessing

- Model Selection

- Model Evaluation

- Regression

- Classification

- Clustering

**Keras**

Keras is a high-level Neural Network library that provides an easy-to-use API for building and training deep learning models. It is written in Python and can be run on top of TensorFlow, Theano, and CNTK. Keras can be used for tasks such as image recognition, natural language processing, and speech recognition. Some of the key features of Keras are:

- Easy-to-use API

- Neural Network models

- Image Processing

- Natural Language Processing

- Speech Recognition

**TensorFlow**

TensorFlow is a popular open-source Machine Learning library developed by Google. It provides various functions for building and training Machine Learning models. TensorFlow can be used for tasks such as image recognition, natural language processing, and speech recognition. TensorFlow also includes pre-trained models for image recognition and natural language processing. Some of the key features of TensorFlow are:

- Machine Learning models

- Neural Network models

- Distributed Computing

- Image Processing

- Natural Language Processing

In summary, these are some of the popular Machine Learning frameworks used by developers. Each of these frameworks has its own strengths and weaknesses and is suited for different tasks. Understanding these frameworks and their capabilities can help developers build better Machine Learning models.

EXPERIMENT 5

AIM:

For a given set of training data examples stored in a .CSV file, implement and demonstrate the scratch Implementation of **Linear Regression Algorithm**

Program:

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

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

dataset = pd.read\_csv('/content/drive/MyDrive/ML\_lab\_0827CI201114/Salary\_Data.csv')

X = dataset.iloc[ : , : 1 ].values

Y = dataset.iloc[ : , 1 ].values

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split( X, Y, test\_size = 1/4, random\_state = 0)

X\_train

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error

from sklearn.metrics import r2\_score

lin\_model = LinearRegression()

lin\_model.fit(X\_train, Y\_train)

y\_train\_predict = lin\_model.predict(X\_train)

rmse = (np.sqrt(mean\_squared\_error(Y\_train, y\_train\_predict)))

r2 = r2\_score(Y\_train, y\_train\_predict)

print("The model performance for training set")

print("--------------------------------------")

print('RMSE is {}'.format(rmse))

print('R2 score is {}'.format(r2))

print("\n")

# model evaluation for testing set

y\_test\_predict = lin\_model.predict(X\_test)

rmse = (np.sqrt(mean\_squared\_error(Y\_test, y\_test\_predict)))

r2 = r2\_score(Y\_test, y\_test\_predict)

print("The model performance for testing set")

print("--------------------------------------")

print('RMSE is {}'.format(rmse))

print('R2 score is {}'.format(r2))

plt.scatter(X\_train, Y\_train)

plt.xlabel('x')

plt.ylabel('y')

# predicted values

plt.plot(X\_train, y\_train\_predict, color='r')

plt.show()

Y\_train

from sklearn.linear\_model import LinearRegression

regressor = LinearRegression()

regressor = regressor.fit(X\_train, Y\_train)

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

plt.scatter(X\_train , Y\_train, color = 'red')

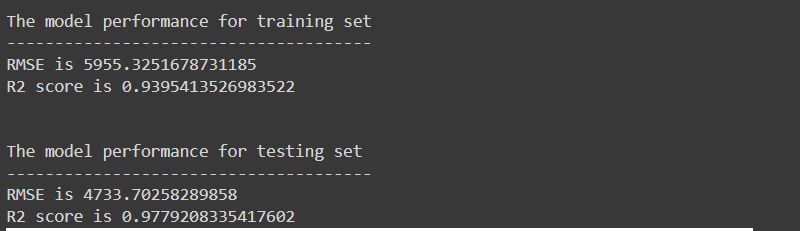
plt.plot(X\_train , regressor.predict(X\_train), color ='blue')

plt.scatter(X\_test , Y\_test, color = 'red')

plt.plot(X\_test , regressor.predict(X\_test), color ='blue')

RESULT:

Chart, scatter chart

Description automatically generated

Chart, scatter chart

Description automatically generated

EXPERIMENT – 6

AIM:

For a given set of training data examples stored in a .CSV file, implement and demonstrate the Implementation of **Linear Regression Algorithm** Linear Regression using Python library (for any given CSV dataset )

Program:

import pandas as pd

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

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error

import matplotlib.pyplot as plt

# Load data from CSV file

def load\_data(filename):

    df = pd.read\_csv(filename)

    return df

# Test the linear regression algorithm

if \_\_name\_\_ == '\_\_main\_\_':

    # Load data from CSV file

    dataset = load\_data('/content/drive/MyDrive/ML\_lab\_0827CI201114/Salary\_Data.csv')

    # Split the data into features (X) and target (y)

    X = dataset.iloc[:, :-1]

    y = dataset.iloc[:, -1]

    # Split the data into training and testing sets

    X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=0)

    # Train the linear regression model

    model = LinearRegression()

    model.fit(X\_train, y\_train)

    # Make predictions on the test set

    y\_pred = model.predict(X\_test)

    # Print the mean squared error

    mse = mean\_squared\_error(y\_test, y\_pred)

    print('Mean squared error:', mse)

    # Plot the data

    plt.scatter(X\_train, y\_train, color='blue')

    plt.plot(X\_train, model.predict(X\_train), color='red')

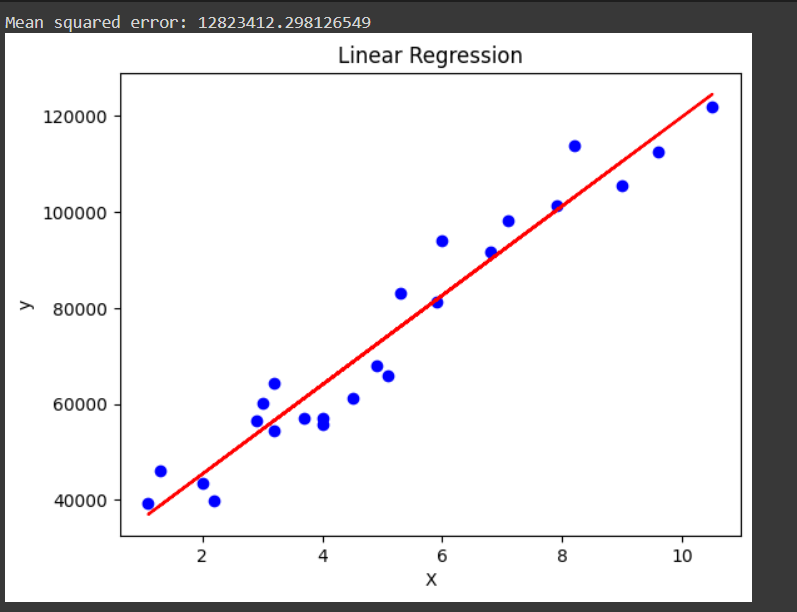
    plt.title('Linear Regression')

    plt.xlabel('X')

    plt.ylabel('y')

    plt.show()

RESULT:



EXPERIMENT 7

AIM:

For a given set of training data examples stored in a .CSV file, implement and demonstrate the scratch Implementation for binary classification using **Logistic**

**Regression Algorithm**

PROGRAM:

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

# Load data from CSV file

def load\_data(filename):

df = pd.read\_csv(filename)

return df

# Sigmoid function

def sigmoid(z):

return 1 / (1 + np.exp(-z))

# Cost function

def compute\_cost(X, y, theta):

m = len(y)

h = sigmoid(X.dot(theta))

J = (-1/m) \* np.sum(y\*np.log(h) + (1-y)\*np.log(1-h))

return J

# Gradient descent function

def gradient\_descent(X, y, theta, alpha, num\_iters):

m = len(y)

J\_history = []

for i in range(num\_iters):

h = sigmoid(X.dot(theta))

theta = theta - (alpha/m) \* X.T.dot(h-y)

J\_history.append(compute\_cost(X, y, theta))

return theta, J\_history

# Test the logistic regression algorithm

if \_\_name\_\_ == '\_\_main\_\_':

# Load data from CSV file

dataset = load\_data('/content/drive/MyDrive/ML\_lab\_0827CI201114/diabetes.csv')

# Split the data into features (X) and target (y)

X = dataset.iloc[:, :-1]

y = dataset.iloc[:, -1]

# Add a column of ones to X for the intercept term

X = np.hstack((np.ones((len(X), 1)), X))

# Initialize theta to zeros

theta = np.zeros(X.shape[1])

# Set hyperparameters

alpha = 0.01

num\_iters = 1000

# Run gradient descent to find optimal theta

theta, J\_history = gradient\_descent(X, y, theta, alpha, num\_iters)

# Print the optimal theta

print('Optimal theta:', theta)

# Plot the cost function over iterations

plt.plot(J\_history)

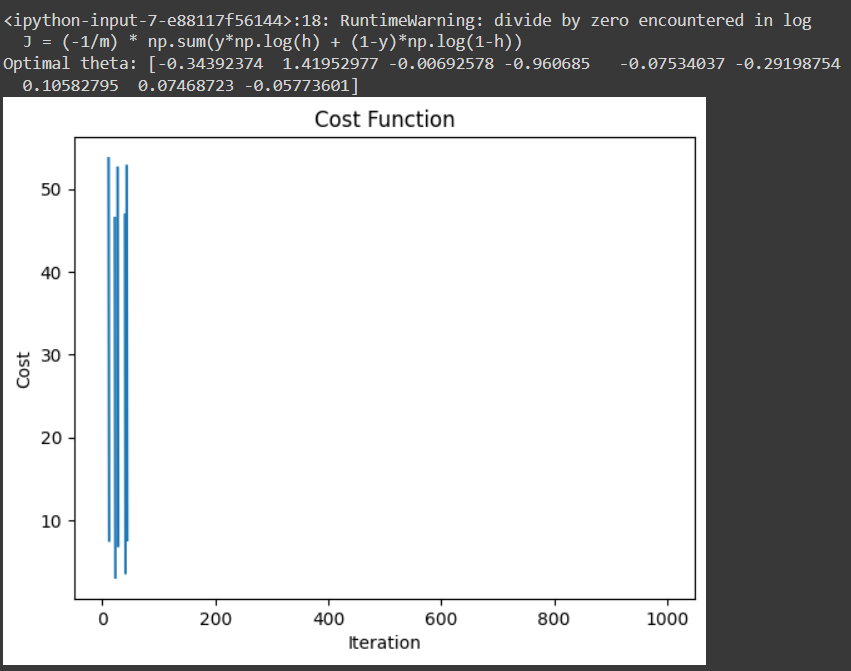
plt.title('Cost Function')

plt.xlabel('Iteration')

plt.ylabel('Cost')

plt.show()

RESULT:



EXPERIMENT 8

AIM:

Build an **Artificial Neural Network (ANN)** by implementing the Backpropagation algorithm and test the same using MNIST **Handwritten Digit Multiclass classification** data sets with use of use of batch normalization, early stopping and dropout

PROGRAM:

import tensorflow as tf

from tensorflow.keras.datasets import mnist

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Dropout, BatchNormalization

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.callbacks import EarlyStopping

import matplotlib.pyplot as plt

# Load the MNIST dataset

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

# Preprocess the data

x\_train = x\_train.reshape(60000, 784).astype('float32') / 255.0

x\_test = x\_test.reshape(10000, 784).astype('float32') / 255.0

y\_train = tf.keras.utils.to\_categorical(y\_train, num\_classes=10)

y\_test = tf.keras.utils.to\_categorical(y\_test, num\_classes=10)

# Define the model architecture

model = Sequential([

    Dense(512, activation='relu', input\_shape=(784,)),

    BatchNormalization(),

    Dropout(0.5),

    Dense(256, activation='relu'),

    BatchNormalization(),

    Dropout(0.5),

    Dense(10, activation='softmax')

])

# Compile the model

model.compile(optimizer=Adam(learning\_rate=0.001),

              loss='categorical\_crossentropy',

              metrics=['accuracy'])

# Define the early stopping callback

early\_stopping = EarlyStopping(monitor='val\_loss', patience=10)

# Train the model

history = model.fit(x\_train, y\_train, batch\_size=128, epochs=10, validation\_data=(x\_test, y\_test), callbacks=[early\_stopping])

loss = history.history['loss']

val\_loss = history.history['val\_loss']

acc = history.history['accuracy']

val\_acc = history.history['val\_accuracy']

epochs = range(len(loss))

plt.figure(figsize=(8, 4))

plt.subplot(1, 2, 1)

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

plt.plot(epochs, val\_loss, 'b', label='Validation loss')

plt.title('Training and Validation Loss')

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.legend()

plt.subplot(1, 2, 2)

plt.plot(epochs, acc, 'r', label='Training accuracy')

plt.plot(epochs, val\_acc, 'b', label='Validation accuracy')

plt.title('Training and Validation Accuracy')

plt.xlabel('Epochs')

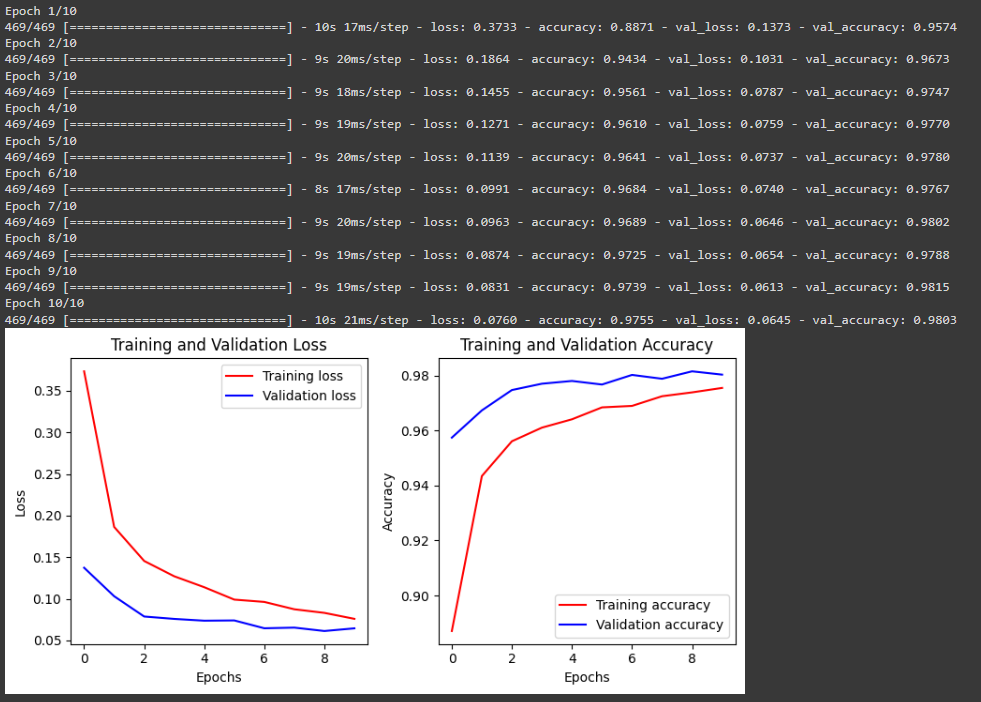
plt.ylabel('Accuracy')

plt.legend()

plt.tight\_layout()

plt.show()

RESULT:



EXPERIMENT 9

AIM:

Build an **Artificial Neural Network** by implementing the Backpropagation algorithm and test the same using **CIFAR 100 Multiclass classification** data sets with use

of use of batch normalization, early stopping and drop out

PROGRAM:

import tensorflow as tf

from tensorflow import keras

from tensorflow.keras import layers

import numpy as np

import matplotlib.pyplot as plt

# Load the CIFAR-100 dataset

(x\_train, y\_train), (x\_test, y\_test) = keras.datasets.cifar100.load\_data()

# Preprocess the data

num\_classes = 100

x\_train = x\_train.astype('float32') / 255.0

x\_test = x\_test.astype('float32') / 255.0

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

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

# Define the model architecture

model = keras.Sequential([

layers.Conv2D(32, (3,3), activation='relu', padding='same', input\_shape=x\_train.shape[1:]),

layers.BatchNormalization(),

layers.Dropout(0.25),

layers.Conv2D(64, (3,3), activation='relu', padding='same'),

layers.BatchNormalization(),

layers.Dropout(0.25),

layers.Conv2D(128, (3,3), activation='relu', padding='same'),

layers.BatchNormalization(),

layers.Dropout(0.25),

layers.Flatten(),

layers.Dense(512, activation='relu'),

layers.BatchNormalization(),

layers.Dropout(0.5),

layers.Dense(num\_classes, activation='softmax')

])

# Compile the model

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

# Define the early stopping callback

early\_stopping = keras.callbacks.EarlyStopping(monitor='val\_loss', patience=10)

# Train the model

history = model.fit(x\_train, y\_train, batch\_size=128, epochs=100,

validation\_data=(x\_test, y\_test), callbacks=[early\_stopping])

# Plot the training and validation loss and accuracy

loss = history.history['loss']

val\_loss = history.history['val\_loss']

acc = history.history['accuracy']

val\_acc = history.history['val\_accuracy']

epochs = range(len(loss))

plt.figure(figsize=(8, 4))

plt.subplot(1, 2, 1)

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

plt.plot(epochs, val\_loss, 'b', label='Validation loss')

plt.title('Training and Validation Loss')

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.legend()

plt.subplot(1, 2, 2)

plt.plot(epochs, acc, 'r', label='Training accuracy')

plt.plot(epochs, val\_acc, 'b', label='Validation accuracy')

plt.title('Training and Validation Accuracy')

plt.xlabel('Epochs')

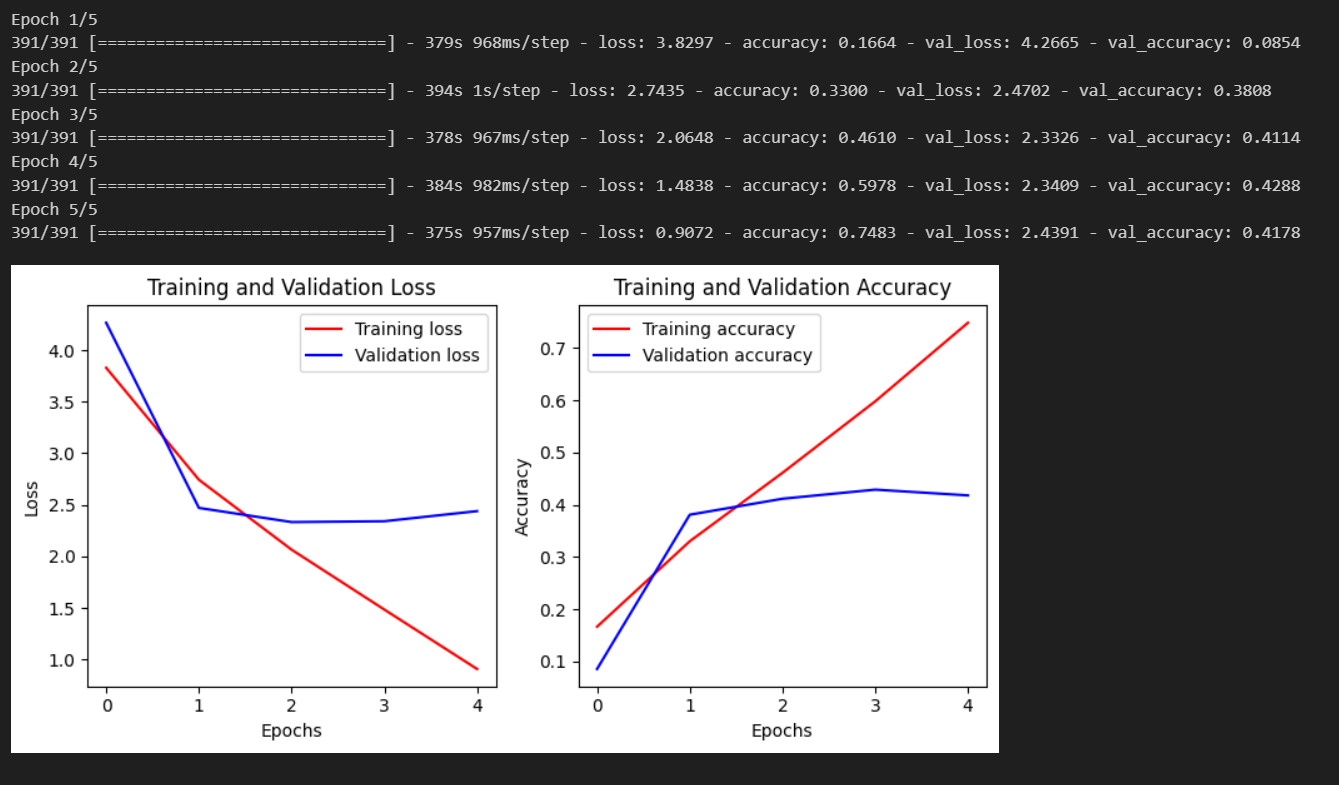
plt.ylabel('Accuracy')

plt.legend()

plt.tight\_layout()

plt.show()

RESULT:



EXPERIMENT 10

AIM:

ANN implementation use of batch normalization, early stopping and drop out (For Image Dataset such as Covid Dataset)

PROGRAM:

import tensorflow as tf

from tensorflow import keras

from tensorflow.keras import layers

import numpy as np

import matplotlib.pyplot as plt

# Load the COVID-19 dataset

data = keras.preprocessing.image.ImageDataGenerator(

rescale=1./255, validation\_split=0.2)

train\_data = data.flow\_from\_directory(

'path/to/training/data',

target\_size=(224, 224),

batch\_size=32,

class\_mode='binary',

subset='training')

val\_data = data.flow\_from\_directory(

'path/to/training/data',

target\_size=(224, 224),

batch\_size=32,

class\_mode='binary',

subset='validation')

# Define the model architecture

model = keras.Sequential([

layers.Conv2D(32, (3,3), activation='relu', padding='same', input\_shape=(224, 224, 3)),

layers.BatchNormalization(),

layers.Dropout(0.25),

layers.Conv2D(64, (3,3), activation='relu', padding='same'),

layers.BatchNormalization(),

layers.Dropout(0.25),

layers.Conv2D(128, (3,3), activation='relu', padding='same'),

layers.BatchNormalization(),

layers.Dropout(0.25),

layers.Flatten(),

layers.Dense(512, activation='relu'),

layers.BatchNormalization(),

layers.Dropout(0.5),

layers.Dense(1, activation='sigmoid')

])

# Compile the model

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

# Define the early stopping callback

early\_stopping = keras.callbacks.EarlyStopping(monitor='val\_loss', patience=10)

# Train the model

history = model.fit(train\_data, epochs=100, validation\_data=val\_data, callbacks=[early\_stopping])

# Plot the training and validation loss and accuracy

loss = history.history['loss']

val\_loss = history.history['val\_loss']

acc = history.history['accuracy']

val\_acc = history.history['val\_accuracy']

epochs = range(len(loss))

plt.figure(figsize=(8, 4))

plt.subplot(1, 2, 1)

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

plt.plot(epochs, val\_loss, 'b', label='Validation loss')

plt.title('Training and Validation Loss')

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.legend()

plt.subplot(1, 2, 2)

plt.plot(epochs, acc, 'r', label='Training accuracy')

plt.plot(epochs, val\_acc, 'b', label='Validation accuracy')

plt.title('Training and Validation Accuracy')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.legend()

plt.tight\_layout()

plt.show()

RESULT:

EXPERIMENT 11

AIM:

Build a **Convolutional Neural Network** by implementing the Backpropagation algorithm and test the same using **MNIST Handwritten Digit Multiclass classification** data sets

PROGRAM:

import tensorflow as tf

from tensorflow import keras

from tensorflow.keras import layers

from tensorflow.keras.datasets import mnist

import numpy as np

import matplotlib.pyplot as plt

# Load the MNIST dataset

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

# Normalize the data

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

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

# Reshape the data

x\_train = np.expand\_dims(x\_train, axis=-1)

x\_test = np.expand\_dims(x\_test, axis=-1)

# Convert the labels to one-hot encoded vectors

num\_classes = 10

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

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

# Define the model architecture

model = keras.Sequential([

layers.Conv2D(32, (3,3), activation='relu', input\_shape=(28, 28, 1)),

layers.MaxPooling2D((2,2)),

layers.Conv2D(64, (3,3), activation='relu'),

layers.MaxPooling2D((2,2)),

layers.Conv2D(128, (3,3), activation='relu'),

layers.Flatten(),

layers.Dense(128, activation='relu'),

layers.Dense(num\_classes, activation='softmax')

])

# Compile the model

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

# Train the model

history = model.fit(x\_train, y\_train, epochs=10, batch\_size=32, validation\_data=(x\_test, y\_test))

# Evaluate the model on the test data

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

print('Test loss:', test\_loss)

print('Test accuracy:', test\_acc)

# Plot the training and validation loss and accuracy

loss = history.history['loss']

val\_loss = history.history['val\_loss']

acc = history.history['accuracy']

val\_acc = history.history['val\_accuracy']

epochs = range(len(loss))

plt.figure(figsize=(8, 4))

plt.subplot(1, 2, 1)

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

plt.plot(epochs, val\_loss, 'b', label='Validation loss')

plt.title('Training and Validation Loss')

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.legend()

plt.subplot(1, 2, 2)

plt.plot(epochs, acc, 'r', label='Training accuracy')

plt.plot(epochs, val\_acc, 'b', label='Validation accuracy')

plt.title('Training and Validation Accuracy')

plt.xlabel('Epochs')

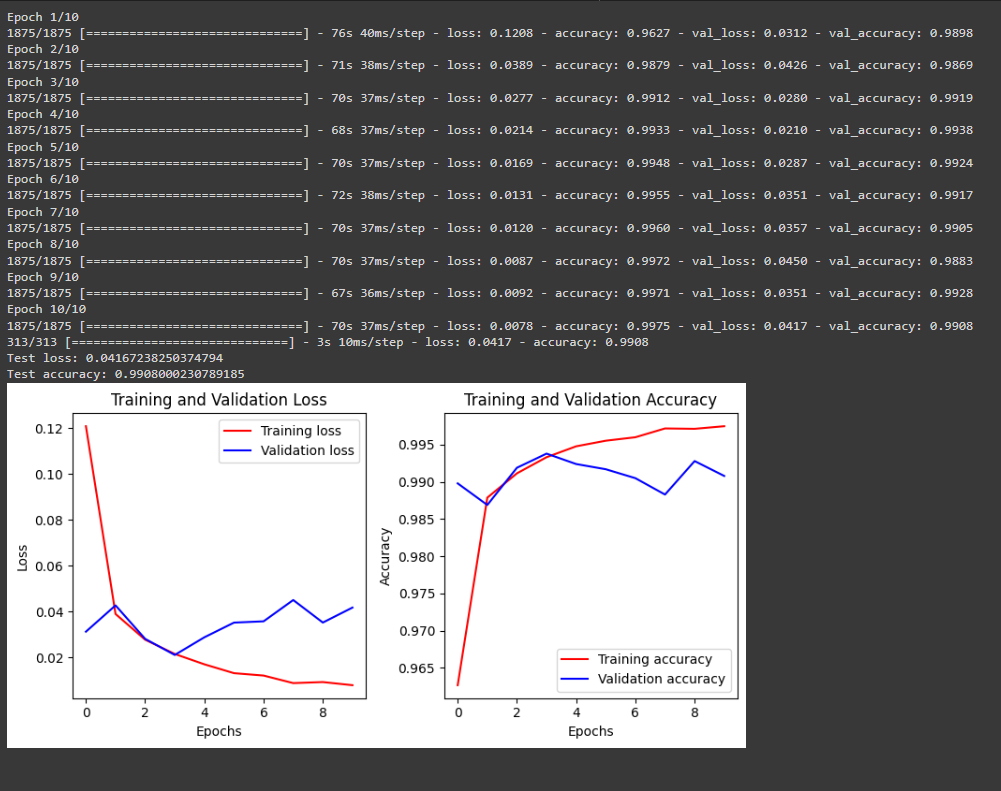
plt.ylabel('Accuracy')

plt.legend()

plt.tight\_layout()

plt.show()

RESULT:



EXPERIMENT 12

AIM:

Build a **Convolutional Neural Network** by implementing the Backpropagation algorithm and test the same using **CIFAR 100 Multiclass classification** data sets.

PROGRAM:

import tensorflow as tf

from tensorflow import keras

from tensorflow.keras import layers

from tensorflow.keras.datasets import cifar100

# Load the CIFAR-100 dataset

(x\_train, y\_train), (x\_test, y\_test) = cifar100.load\_data()

# Normalize the pixel values to be between 0 and 1

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

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

# Convert the labels to one-hot encoding

y\_train = keras.utils.to\_categorical(y\_train, num\_classes=100)

y\_test = keras.utils.to\_categorical(y\_test, num\_classes=100)

# Define the model architecture

model = keras.Sequential(

[

# Convolutional layers

layers.Conv2D(32, (3, 3), activation="relu", padding="same", input\_shape=(32, 32, 3)),

layers.BatchNormalization(),

layers.Conv2D(32, (3, 3), activation="relu", padding="same"),

layers.BatchNormalization(),

layers.MaxPooling2D(pool\_size=(2, 2)),

layers.Dropout(0.25),

layers.Conv2D(64, (3, 3), activation="relu", padding="same"),

layers.BatchNormalization(),

layers.Conv2D(64, (3, 3), activation="relu", padding="same"),

layers.BatchNormalization(),

layers.MaxPooling2D(pool\_size=(2, 2)),

layers.Dropout(0.25),

# Dense layers

layers.Flatten(),

layers.Dense(512, activation="relu"),

layers.BatchNormalization(),

layers.Dropout(0.5),

layers.Dense(100, activation="softmax"),

]

)

# Define the optimizer, loss function, and metric

optimizer = keras.optimizers.RMSprop(learning\_rate=1e-4)

loss\_fn = keras.losses.CategoricalCrossentropy()

metric = keras.metrics.CategoricalAccuracy()

# Compile the model

model.compile(optimizer=optimizer, loss=loss\_fn, metrics=[metric])

# Define the early stopping callback

early\_stopping\_cb = keras.callbacks.EarlyStopping(monitor="val\_loss", patience=10)

# Train the model

history = model.fit(

x\_train,

y\_train,

batch\_size=32,

epochs=100,

validation\_split=0.1,

callbacks=[early\_stopping\_cb],

)

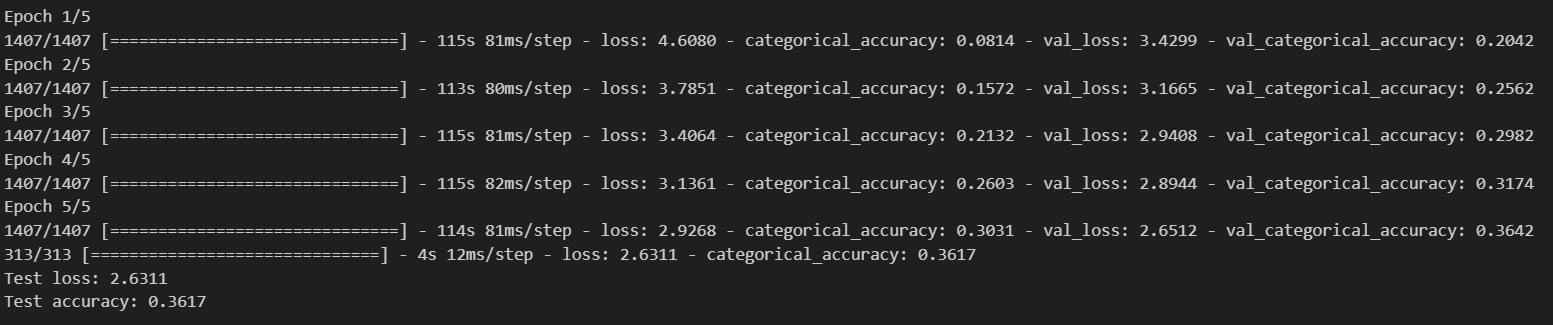
# Evaluate the model on the test set

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

print(f"Test loss: {test\_loss:.4f}")

print(f"Test accuracy: {test\_accuracy:.4f}")

RESULT:



EXPERIMENT 13

AIM:

Implementation of Transfer Learning (VGG 16)

PROGRAM:

import tensorflow as tf

from tensorflow.keras.layers import Dense, Flatten

from tensorflow.keras.models import Model

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.applications.vgg16 import VGG16

# load VGG16 model without classifier layers

vgg = VGG16(input\_shape=(224, 224, 3), include\_top=False)

# mark loaded layers as not trainable

for layer in vgg.layers:

layer.trainable = False

# Add new classifier layers

flat1 = Flatten()(vgg.layers[-1].output)

class1 = Dense(1024, activation='relu')(flat1)

output = Dense(5, activation='softmax')(class1)

# define the new model

model = Model(inputs=vgg.inputs, outputs=output)

# compile the model

model.compile(optimizer=Adam(learning\_rate=0.001), loss='categorical\_crossentropy', metrics=['accuracy'])

# print summary of the model

model.summary()

# Create dummy data

import numpy as np

import os

from PIL import Image

train\_dir = 'train\_data'

os.makedirs(train\_dir, exist\_ok=True)

test\_dir = 'test\_data'

os.makedirs(test\_dir, exist\_ok=True)

# Create 100 training images and 20 test images for each class

classes = ['class\_0', 'class\_1', 'class\_2', 'class\_3', 'class\_4']

for c in classes:

os.makedirs(os.path.join(train\_dir, c), exist\_ok=True)

os.makedirs(os.path.join(test\_dir, c), exist\_ok=True)

for i in range(100):

img = Image.fromarray(np.random.randint(0, 256, (224, 224, 3), dtype=np.uint8))

img.save(os.path.join(train\_dir, c, '{}.png'.format(i)))

for i in range(20):

img = Image.fromarray(np.random.randint(0, 256, (224, 224, 3), dtype=np.uint8))

img.save(os.path.join(test\_dir, c, '{}.png'.format(i)))

# Create data generators

train\_datagen = ImageDataGenerator(rescale=1.0/255.0,

width\_shift\_range=0.1,

height\_shift\_range=0.1,

horizontal\_flip=True,

validation\_split=0.2)

test\_datagen = ImageDataGenerator(rescale=1.0/255.0)

train\_generator = train\_datagen.flow\_from\_directory(train\_dir,

target\_size=(224, 224),

batch\_size=32,

class\_mode='categorical',

subset='training')

validation\_generator = train\_datagen.flow\_from\_directory(train\_dir,

target\_size=(224, 224),

batch\_size=32,

class\_mode='categorical',

subset='validation')

# Train the model

history = model.fit(train\_generator,

steps\_per\_epoch=len(train\_generator),

validation\_data=validation\_generator,

validation\_steps=len(validation\_generator),

epochs=10)

# Evaluate the model on test data

test\_generator = test\_datagen.flow\_from\_directory(test\_dir,

target\_size=(224, 224),

batch\_size=32,

class\_mode='categorical')

loss, accuracy = model.evaluate(test\_generator, steps=len(test\_generator))

print('Test accuracy: {}'.format(accuracy))

import matplotlib.pyplot as plt

# Plot the training and validation accuracy

plt.plot(history.history['accuracy'])

plt.plot(history.history['val\_accuracy'])

plt.title('Model accuracy')

plt.ylabel('Accuracy')

plt.xlabel('Epoch')

plt.legend(['Train', 'Validation'], loc='upper left')

plt.show()

# Plot the training and validation loss

plt.plot(history.history['loss'])

plt.plot(history.history['val\_loss'])

plt.title('Model loss')

plt.ylabel('Loss')

plt.xlabel('Epoch')

plt.legend(['Train', 'Validation'], loc='upper left')

plt.show()

RESULT:

Text

Description automatically generated

Chart, line chart

Description automatically generated Chart, line chart

Description automatically generated

EXPERIMENT 14

AIM:

Implementation of RNN

PROGRAM:

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, SimpleRNN

import numpy as np

import matplotlib.pyplot as plt

# Define the input sequence and output sequence

input\_sequence = np.array([[0.1, 0.2, 0.3], [0.2, 0.3, 0.4], [0.3, 0.4, 0.5], [0.4, 0.5, 0.6]])

output\_sequence = np.array([0.4, 0.5, 0.6, 0.7])

# Define the RNN model

model = Sequential()

model.add(SimpleRNN(32, input\_shape=(3, 1)))

model.add(Dense(1))

# Compile the model

model.compile(optimizer='adam', loss='mean\_squared\_error')

# Train the model

history = model.fit(input\_sequence.reshape(4, 3, 1), output\_sequence, epochs=100, verbose=0)

# Plot the training loss

plt.plot(history.history['loss'])

plt.title('Model loss')

plt.ylabel('Loss')

plt.xlabel('Epoch')

plt.show()

RESULT:

Chart, histogram

Description automatically generated