



Lab Manual of Machine Learning [CS-602]

B. Tech. VI Semester

Jan - June 2024

**Department of Computer Science and
Information Technology**

Submitted to Submitted By Prof. Write your name

Designation, CSIT Dept. Roll number

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. / Ms.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	How to set up a python environment for Machine Learning & Deep Learning with Anaconda.	
2	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.	

3	Python List Comprehension with examples	
4	Basic of Numpy, Pandas and Matplotlib	
5	Brief Study of Machine Learning Frameworks such as Open CV, Scikit Learn, Keras, Tensorflow etc.	
6	For a given set of training data examples stored in a .CSV file, implement and demonstrate the scratch Implementation of Linear Regression Algorithm	
7	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) salary.csv	
8	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 and KNN. Compare the accuracy of both algorithm using confusion matrix	

Experiment- 1

Aim: Setting up a Python environment for Machine Learning and Deep Learning with Anaconda is a great choice as Anaconda simplifies package management and environment creation. Here's a step-by-step guide to setting it up:

Steps To Setup a python environment for Machine Learning & Deep Learning with Anaconda

Step 1 : Install Anaconda:

- Download Anaconda from the official website:
<https://www.anaconda.com/products/distribution>
- Follow the installation instructions for your operating system.

Step 2 : Create a New Environment:

- Open Anaconda Navigator (you can find it in your applications folder or search for it).
- Click on the "Environment" tab on the left sidebar.
- Click the "Create" button.
- Give your environment a name, for example, "ml_env".
- Choose the Python version you want to use (typically Python 3.x).
- Select the necessary packages like numpy, scipy, pandas, matplotlib, scikit-learn, jupyter, etc. For deep learning, you might need packages like TensorFlow or PyTorch. You can add these later if needed.
- Click the "Create" button.

Step 3: Activate the Environment:

- Once the environment is created, you need to activate it.
- Go back to the "Home" tab in Anaconda Navigator.
- Select the environment you just created from the drop-down menu.
- Click on the "Home" button and choose "Open Terminal" or "Open Command Prompt".

- In the terminal or command prompt, type:

```
conda activate ml_env
```

Step 4 : Install Additional Packages:

- If there are additional packages you need, you can install them using `conda install` or `pip install` while the environment is activated.

For example:

```
conda install tensorflow
```

Or

```
pip install tensorflow
```

Step 5 : Verify Installation:

- You can verify that everything is set up correctly by running a Python interpreter or opening a Jupyter Notebook within the activated environment.
- To launch Jupyter Notebook, simply type `jupyter notebook` in the terminal or command prompt, and a browser window should open with Jupyter running. From there, you can create new notebooks or open existing ones.

Experiment- 2

Basic Concepts Of Python Programming

here's a brief overview of some basic programming concepts in Python:

Variables and Data Types: In Python, variables are used to store data values. Variables don't need to be declared with their type explicitly; Python infers the type based on the value assigned to it. Common data types include integers, floats, strings, booleans, lists, tuples, dictionaries, etc.

Operators: Python supports various types of operators such as arithmetic operators (+, -, *, /), comparison operators (==, !=, <, >, <=, >=), logical operators (and, or, not), assignment operators (=, +=, -=, *=, /=), and more.

Control Structures: Python provides control structures like if statements, for loops, while loops, and try-except blocks. These are used for decision making and looping constructs in programs.

Functions: Functions are blocks of reusable code that perform a specific task. In Python, you can define functions using the def keyword. Functions can take parameters and return values.

Modules and Packages: Python modules are files containing Python code, and packages are directories of Python modules. Modules and packages help organize code and promote code reuse.

Strings and String Manipulation: Strings are sequences of characters. Python provides powerful string manipulation methods and operations such as slicing, concatenation, formatting, and more.

Lists, Tuples, and Dictionaries: These are data structures used to store collections of items. Lists are mutable sequences, tuples are immutable sequences, and dictionaries are collections of key-value pairs.

File Handling: Python provides functions and methods for working with files. You can open, read, write, and close files using built-in functions like open(), read(), write(), and close().

Exception Handling: Python's try-except blocks are used for handling exceptions and errors that may occur during program execution. This helps in writing robust and error-tolerant code.

Object-Oriented Programming (OOP): Python supports object-oriented programming paradigms. Classes and objects are used to model real-world entities, encapsulate data, and define behavior through methods.

Modules for Specific Tasks: Python comes with a rich standard library that provides modules for various tasks such as mathematical operations, working with dates and times, networking, file I/O, and more. Additionally, there are numerous third-party libraries available for specialized tasks.

These are some fundamental concepts in Python programming. Mastering these concepts will provide a solid foundation for further exploration and development in Python.

Python Data Structures and Dictionary and Lambda Functions

Here's a breakdown of Python data structures and lambda functions:

Lists:

Definition: Lists are ordered collections of items, which can be of different data types. They are mutable, meaning the elements can be changed after the list is created.

Initialization: Lists are created by placing comma-separated values inside square brackets [].

Example:

```
my_list = [1, 2, 3, 'a', 'b', 'c']
```

Common Operations:

Accessing elements: `my_list[index]`

Slicing: `my_list[start:end:step]`

Modifying elements: `my_list[index] = new_value`

Appending elements: `my_list.append(item)`

Removing elements: `my_list.remove(item)` or `del my_list[index]`

Length of the list: `len(my_list)`

Tuples:

Definition: Tuples are ordered collections similar to lists, but they are immutable, meaning once created, the elements cannot be changed.

Initialization: Tuples are created by placing comma-separated values inside parentheses ().

Example:

```
my_tuple = (1, 2, 3, 'a', 'b', 'c')
```

Common Operations:

Accessing elements: `my_tuple[index]`

Slicing: `my_tuple[start:end:step]`

Length of the tuple: `len(my_tuple)`

Strings:

Definition: Strings are sequences of characters. They are immutable like tuples.

Initialization: Strings are created by enclosing characters inside single `' '` or double `" "` quotes.

Example:

```
my_string = "Hello, world!"
```

Common Operations:

Concatenation: `string1 + string2`

Indexing: `my_string[index]`

Slicing: `my_string[start:end:step]`

Length of the string: `len(my_string)`

Methods like `upper()`, `lower()`, `strip()`, `split()`, etc.

Dictionaries:

Definition: Dictionaries are unordered collections of items. Each item is a key-value pair. Keys are unique and immutable, and values can be of any data type.

Initialization: Dictionaries are created by placing comma-separated key-value pairs inside curly braces `{ }`.

Example:

```
my_dict = {'name': 'John', 'age': 30, 'city': 'New York'}
```

Common Operations:

Accessing values: `my_dict[key]`

Adding or modifying items: `my_dict[key] = value`

Removing items: `del my_dict[key]`

Length of the dictionary: `len(my_dict)`

Checking if key exists: `key in my_dict`

Lambda Functions:

Definition: Lambda functions, also known as anonymous functions, are small, single-expression functions without a name. They can take any number of arguments but can only have one expression.

Syntax: `lambda arguments: expression`

Example:

```
square = lambda x: x ** 2
```

Common Use Cases:

As arguments to higher-order functions like `map()`, `filter()`, and `reduce()`.

Writing short, throwaway functions.

Simplifying code by avoiding the need to define a separate named function.

These are some of the core data structures in Python along with lambda functions, which are often used for functional programming paradigms and concise code

expressions. Understanding these concepts will greatly enhance your ability to manipulate data and write efficient Python code.

Python Classes and Objects

Let's delve into Python classes and objects, as well as some popular libraries like NumPy, Pandas, and Matplotlib:

Python Classes and Objects:

Classes: In Python, a class is a blueprint for creating objects. It defines the properties (attributes) and behaviors (methods) that objects of the class should have.

Definition: A class is defined using the class keyword followed by the class name and a colon. Inside the class definition, you specify the attributes and methods.

Example:

```
class Person:
```

```
    def __init__(self, name, age):
        self.name = name
        self.age = age
```

```
    def greet(self):
```

```
        return f"Hello, my name is {self.name} and I am {self.age} years old."
```

Objects: Objects are instances of classes. They are created using the class constructor (often referred to as instantiation) and can access the attributes and methods defined in the class.

Instantiation: Objects are created by calling the class name followed by parentheses, optionally passing arguments to the class constructor (`__init__` method).

Example:

```
person1 = Person("Alice", 30)
```

```
print(person1.greet()) # Output: "Hello, my name is Alice and I am 30 years old."
```

Python Libraries

1. NumPy:

Description: NumPy is a powerful library for numerical computing in Python. It provides support for multi-dimensional arrays and matrices, along with a collection of

mathematical functions to operate on these arrays efficiently.

Key Features:

Multi-dimensional array objects (ndarray).

Broadcasting: performing arithmetic operations on arrays of different shapes.

Linear algebra, Fourier transform, and random number capabilities.

Example:

```
import numpy as np
```

```
# Creating a NumPy array
```

```
arr = np.array([1, 2, 3, 4, 5])
```

```
# Performing operations on the array
```

```
arr_sum = np.sum(arr)
```

```
print(arr_sum) # Output: 15
```

2. Pandas:

Description: Pandas is a fast, powerful, and flexible library for data manipulation and analysis in Python. It provides data structures like DataFrame and Series, which are ideal for handling structured data.

Key Features:

DataFrame: a 2-dimensional labeled data structure with columns of potentially different types.

Series: a one-dimensional labeled array capable of holding any data type.

Data alignment, indexing, reshaping, merging, and grouping operations.

Example:

```
import pandas as pd
```

```
# Creating a DataFrame
```

```
data = {'Name': ['Alice', 'Bob', 'Charlie'],  
        'Age': [30, 25, 35]}
```

```
df = pd.DataFrame(data)
```

```
# Performing operations on the DataFrame
```

```
df_mean = df['Age'].mean()
```

```
print(df_mean) # Output: 30.0
```

3. Matplotlib:

Description: Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. It provides a MATLAB-like interface and supports

a wide variety of plots and customization options.

Key Features:

Line plots, scatter plots, bar plots, histograms, pie charts, etc.

Support for customization: labels, colors, styles, annotations, etc.

Seamless integration with NumPy arrays and Pandas DataFrames.

Example:

```
import matplotlib.pyplot as plt
```

```
# Creating a simple line plot
```

```
x = [1, 2, 3, 4, 5]
```

```
y = [2, 3, 5, 7, 11]
```

```
plt.plot(x, y)
```

```
plt.xlabel('X-axis')
```

```
plt.ylabel('Y-axis')
```

```
plt.title('Simple Line Plot')
```

```
plt.show()
```

These libraries greatly enhance Python's capabilities for data manipulation, analysis, and visualization. Understanding how to utilize them effectively can significantly streamline development workflows and enable the creation of powerful data-driven applications and visualizations.

EXPERIMENT - 3

Aim - Python List Comprehension with examples.

List Comprehension

List comprehension offers a shorter syntax when you want to create a new list based on the values of an existing list.

Lists, also called arrays, are data types in programming languages. The classification or arrangement of data into different categories based on their characteristics are called data types. The most common examples of data types in Python include integers, strings, characters, int, floats, and Boolean. Lists are built-in versatile data types in Python that store a specific data category.

They store multiple items in a single variable and follow an index from 0 to 1 to number all the items. The first item in a list is marked 0, the second is marked 1, and so on. Moreover, the items in a list are arranged in a particular order. It means you cannot add an item in between. The new items automatically go to the end of the list. You can create a new list using Python's `list()` constructor.

However, when you want to create a new list based on an existing list, you need to use comprehension in Python. It helps generate a concise code by creating new sequences from the existing sequences. This is one of the biggest advantages of using Python. It allows you to write clear, easy-to-read code.

There are four types of comprehension in Python for different data types – list comprehension, dictionary comprehension, set comprehension, and generator comprehension.

Python List Comprehension Syntax

List comprehension generates new lists by applying an expression to items extracted from another iterable (such as a list, range, or tuple). In Python, the syntax looks like the following:

list = [expression for element in iterable if condition]

The syntax consists of the following parts:

- expression is a calculation performed on an element. The element resulting from the expression appends to the newly created list.
- element is an item extracted from an iterable on which the expression applies.
- iterable is an iterable from which the elements are extracted.
- condition is an optional check whether the element meets the provided condition.

Benefits of List Comprehension

List comprehension is one of the most remarkable features of Python that enables writing clear and concise codes. Some of its significant benefits are:

- Facilitates writing the code in fewer lines
- Allows writing codes that are easier to understand and that adhere to Python guidelines
- Converts iterable (list) into a formula
- Filters and maps the items in a list to boost code performance

When to Not Use List Comprehension

Even though list comprehension can make writing codes easier, you do not have to use it every time. You should not use list comprehension in the following circumstances.

Elaborate Code

When your code is too elaborate or complex, it is better to avoid list comprehension as it can be difficult to understand the code and hamper its performance. You can consider using a loop or other functions if the list comprehension expression is too lengthy.

No Use of Existing List

The whole purpose of using list comprehension in Python is to generate a new list that is related to or dependent on an existing list. However, if you don't have to modify an existing list, you should not use list comprehension.

Writing Matrix

You should not use list comprehension when you are writing a matrix because it flattens the code and makes it difficult to understand.

Python List Comprehension Examples

1. Creating a List of Squares

Use list comprehension to create a list of squares. The `range()` function generates a number range for this task. For example:

```
squares_list = [number**2 for number in range(0, 10)]  
print(squares_list)
```

2. Filtering a List

Add a condition to filter a generated or existing list. For example:

```
odd = [number for number in range(0, 10) if number % 2 != 0]
print(odd)
```

3. Formatted Strings List

List comprehension can be used to format a list of strings quickly. For example:

```
words = ["These", "Words", "Are", "Capitalized"]
lowercase = [word.lower() for word in words]
print(lowercase)
```

4. Filtering words

List comprehension lets you filter a list containing words based on a condition. The following code shows how to filter words based on the number of letters a word has:

```
words = ["Filtering", "words", "based", "on", "length"]
five = [word for word in words if len(word) >= 5]
print(five)
```

EXPERIMENT 4

AIM- Basic of Numpy, Pandas and Matplotlib


Introduction to NumPy Library

NumPy is a Python library used for scientific computing. This is Python's scientific computing core library, providing high-performance multidimensional array objects, tools for manipulating those arrays, and various mathematical functions. It also contains useful linear algebra, Fourier transform, and random number capabilities.

Benefits of Using NumPy Library

1. **Easy to use:** NumPy is very easy to use, and its syntax is simple, making it easier to code .
2. **Speed:** NumPy is very fast as it uses highly optimized C and Fortran libraries under the hood.
3. **Memory efficiency:** NumPy is very memory efficient as it stores data in a compact form and uses less memory compared to other libraries.
4. **Compatibility:** NumPy is compatible with many other libraries such as SciPy, Scikit-learn, Matplotlib, etc.

Array broadcasting: Array broadcasting allows you to perform operations on arrays of different shapes. This helps in writing efficient and concise code.

1. **Math library:** NumPy has an extensive math library that provides many mathematical functions such as trigonometric functions , logarithms, etc.
2. **Linear algebra support:** NumPy supports linear algebra operations such as matrix multiplication, vector operations, etc. .

Working with Pandas Library

Pandas is an open-source library for data analysis and manipulation. It provides a wide range of data structures and tools for working with data. It is designed for easy data wrangling and manipulation and can be used for a variety of tasks such as data cleaning, data analysis, data visualization, and more. Pandas can be used

for data analysis in Python and other languages such as R and Julia.

Exploring Seaborn Library

Seaborn is a Python library for creating attractive and informative statistical graphics. It is built on the popular matplotlib library and provides a high-level interface for creating intricate statistical graphics. Seaborn provides a range of data visualization tools, such as heat maps, pair plots, and violin plots. Seaborn also provides statistical estimation and inference tools, such as linear models, clustering, and bootstrapping. Seaborn is particularly well-suited for exploring relationships between multiple variables, as it provides tools for visualizing high-dimensional datasets.

Exploring Sklearn Library

Sklearn is a library of Python modules for machine learning and data mining. It is built on NumPy, SciPy, and matplotlib and provides a range of supervised and unsupervised learning algorithms. It is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy. There are various classification, regression, and clustering algorithms, such as support vector machines, random forests, gradient boosting, k-means, and DBSCAN. It also provides a way to reduce data's dimensionality and tools for preprocessing data. Sklearn also features built-in cross-validation and scoring methods.

Comparing Different Libraries

1. **Pandas:** Pandas is a data-analysis library that provides high-level data structures and robust data analysis tools. It is used for data wrangling, cleaning, and preparation. It is designed to make data manipulation and analysis easy and intuitive.
2. **Numpy:** NumPy is a scientific computing library for Python. It provides powerful tools for manipulating and analyzing numerical data. It is used for array-based computations, linear algebra, Fourier transforms random number functions, and more.
1. **Scikit-learn:** Scikit-learn is a machine-learning library for Python. It provides tools for supervised and unsupervised learning, data preprocessing, model selection, etc. It is designed to be easy to use,

efficient, and robust.

2. **Seaborn:** Seaborn is a data visualization library for Python. It provides high-level plotting functions for creating attractive and informative visualizations. It is optimized for working with pandas data structures and can integrate with NumPy and Scikit-learn.

Tips and Tricks for Optimization

1. Pandas

- a. Try to use vectorized operations for data manipulation and extraction. This is often much faster than iterating through a DataFrame or Series.
- b. Use the `.info()` method to get an overview of the dataframe, such as the number of non-null values and the data types of the columns.
- c. Use the `.describe()` method to get summary statistics of numeric columns.
- d. Use the `.isnull()` method to check for missing values.
- e. Use the `.groupby()` method to aggregate and filter data.

2. Numpy

- a. Use boolean masks instead of explicit loops for vectorized operations.
- b. Use the `.reshape()` method to manipulate the shape of arrays.
- c. Use the `.concatenate()` method to combine multiple arrays.
- d. Use the `.stack()` method to convert a 2-dimensional array into a 1-dimensional array.
- e. Use the `.tile()` method to repeat an array multiple times.

EXPERIMENT 5

AIM: Brief Study of Machine Learning Frameworks such as Open CV, Scikit Learn, Keras, Tensorflow etc

Theory: A machine learning framework is a software library or tool that provides the necessary infrastructure and building blocks to develop, train, evaluate, and deploy machine learning models efficiently. These frameworks offer a variety of functionalities and components

1. TensorFlow: TensorFlow is a free end-to-end open-source platform that has a wide variety of tools, libraries, and resources for Machine Learning. It was developed by the Google Brain team and initially released on November 9, 2015. You can easily build and train Machine Learning models with high-level APIs such as Keras using TensorFlow. It also provides multiple levels of abstraction so you can choose the option you need for your model.

TensorFlow also allows you to deploy Machine Learning models anywhere such as the cloud, browser, or your own device. You should use TensorFlow Extended (TFX) if you want the full experience, TensorFlow Lite if you want usage on mobile devices, and TensorFlow.js if you want to train and deploy models in JavaScript environments. TensorFlow is available for Python and C APIs and also for [C++](#), [Java](#), [JavaScript](#), [Go](#), [Swift](#), etc. but without an API backward compatibility guarantee. Third-party packages are also available for **MATLAB**, **C#**, **Julia**, **Scala**, **R**, **Rust**, etc.

2. Scikit-learn: Scikit-learn is a free software library for Machine Learning coding primarily in the Python programming language. It was initially developed as a Google Summer of Code project by David Cournapeau and originally released in June 2007. Scikit-learn is built on top of other Python libraries like NumPy, SciPy, [Matplotlib](#), [Pandas](#), etc. and so it provides full interoperability

While Scikit-learn is written mainly in Python, it has also used Cython to write somecore algorithms in order to improve performance. You can implement various Supervised and Unsupervised Machine learning models on Scikit-learn like Classification, Regression, Support Vector Machines, Random Forests, Nearest Neighbors, Naive Bayes, Decision Trees, Clustering, etc. with Scikit-learn.

3. Open CV : OpenCV, short for Open Source Computer Vision Library, is a powerful open-source framework widely used for computer vision and machine learning tasks. Originally developed by Intel, it now boasts a community-driven development model. OpenCV offers a comprehensive suite of tools and algorithms for image and video processing, enabling tasks like reading and writing image files, as well as performing various transformations and filtering operations. Moreover, it provides robust functionalities for feature detection and description, crucial for applications such as object recognition and image matching. Its object detection and tracking capabilities, including pre-trained models, further enhance its utility for real-world applications. OpenCV's versatility and extensive documentation make it a go-to choice for developers and researchers seeking to implement computer vision solutions efficiently.

4. Keras : Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling fast experimentation and prototyping of deep learning models. Keras offers a user-friendly and intuitive interface, making it accessible to both beginners and experts in the field. Its modular design allows for easy construction of complex neural network architectures, with layers, activations, optimizers, and other components readily available. Keras supports both convolutional and recurrent networks, as well as combinations of the two, facilitating the development of models for various tasks such as image classification, natural language processing, and sequence generation. With its emphasis on simplicity, flexibility, and extensibility, Keras has become one of the most popular frameworks for building and deploying deep learning models.

Experiment- 6

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

Introduction:

Linear regression is a fundamental statistical method used for modeling the relationship between a dependent variable and one or more independent variables. It is widely employed in various fields including economics, finance, engineering, and social sciences for predictive analysis and inference. In this practical demonstration, we aim to implement the linear regression algorithm from scratch using Python programming language and apply it to a dataset stored in a .CSV file.

Linear Regression Algorithm:

1. Model Representation:

Linear regression assumes a linear relationship between the independent variables (target) y . Mathematically it can be represented as.

$$y = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n$$

Where:

- y is the dependent variable.
- x_1, x_2, \dots, x_n are the independent variables.
- $\theta_0, \theta_1, \dots, \theta_n$ are the parameters or coefficients to be learned.

2. Cost Function:

The goal of linear regression is to minimize the difference between the predicted values and the actual values. This is achieved by defining a cost function, often referred to as the Mean

Squared Error (MSE), which measures the average squared difference between the predicted and actual values. The cost function is given by:

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

Where:

- m is the number of training examples.
- $h_{\theta}(x)$ is the hypothesis function, which predicts the value of y given x .
- $x^{(i)}$ and $y^{(i)}$ are the feature and target values of the i th training example.

3. Gradient Descent:

Gradient Descent is an optimization algorithm used to minimize the cost function by adjusting the parameters θ iteratively. The update rule for gradient descent is given by:

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

Where:

- α is the learning rate, determining the step size of each iteration.

4. Implementation Steps:

1. Load the dataset from the .CSV file.
2. Preprocess the data if necessary (e.g., handle missing values, feature scaling).
3. Initialize the parameters θ with zeros or random values.
4. Implement the hypothesis function $h_{\theta}(x)$ and the cost function $J(\theta)$.
5. Implement the gradient descent algorithm to update the parameters θ .
6. Iterate until convergence or a maximum number of iterations is reached.
7. Use the learned parameters to make predictions on new data.

Description:

1. Load the Data: Read the weatherHistory.csv file to extract the training data.

Each row represents a data example, with humidity and temperature as features, and the target variable could be something like precipitation, temperature at a certain time, or any other weather-related metric.

data.head()

	Formatted Date	Summary	Precip Type	Temperature (C)	Apparent Temperature (C)	Humidity	Wind Speed (km/h)	Wind Bearing (degrees)	Visibility (km)	Loud Cover	Pressure (millibars)	Daily Summary
0	2006-04-01 00:00:00.000 +0200	Partly Cloudy	rain	9.472222	7.388889	0.89	14.1197	251.0	15.8263	0.0	1015.13	Partly cloudy throughout the day.
1	2006-04-01 01:00:00.000 +0200	Partly Cloudy	rain	9.355556	7.227778	0.86	14.2646	259.0	15.8263	0.0	1015.63	Partly cloudy throughout the day.
2	2006-04-01 02:00:00.000 +0200	Mostly Cloudy	rain	9.377778	9.377778	0.89	3.9284	204.0	14.9569	0.0	1015.94	Partly cloudy throughout the day.
3	2006-04-01 03:00:00.000 +0200	Partly Cloudy	rain	8.288889	5.944444	0.83	14.1036	269.0	15.8263	0.0	1016.41	Partly cloudy throughout the day.
4	2006-04-01 04:00:00.000 +0200	Mostly Cloudy	rain	8.755556	6.977778	0.83	11.0446	259.0	15.8263	0.0	1016.51	Partly cloudy throughout the day.

data.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 96453 entries, 0 to 96452
Data columns (total 12 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   Formatted Date                        96453 non-null  object
1   Summary                              96453 non-null  object
2   Precip Type                          95936 non-null  object
3   Temperature (C)                      96453 non-null  float64
4   Apparent Temperature (C)             96453 non-null  float64
5   Humidity                             96453 non-null  float64
6   Wind Speed (km/h)                    96453 non-null  float64
7   Wind Bearing (degrees)               96453 non-null  float64
8   Visibility (km)                      96453 non-null  float64
9   Loud Cover                           96453 non-null  float64
10  Pressure (millibars)                  96453 non-null  float64
11  Daily Summary                        96453 non-null  object
dtypes: float64(8), object(4)
memory usage: 8.8+ MB
```

data.size

1157436

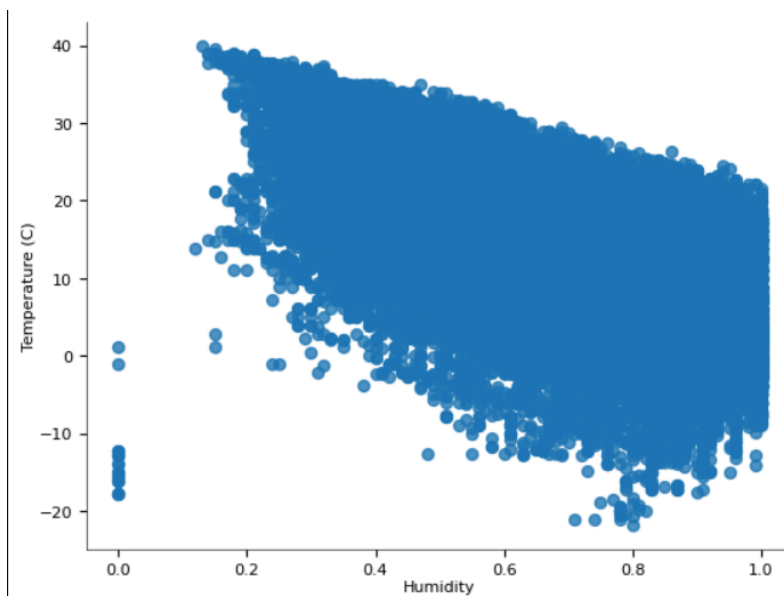
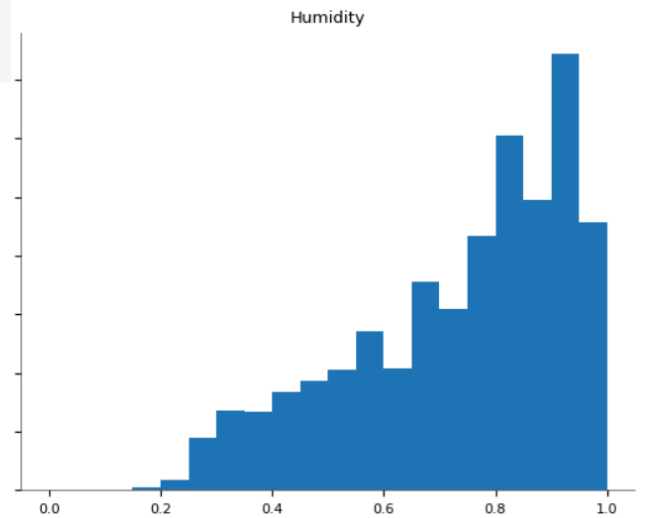
2. Data Preprocessing: Check for any missing values or outliers in the data. If there are missing values, you may need to handle them by imputing or removing the respective

rows. Outliers may also need to be treated appropriately depending on their impact on the model.

```
data=data[['Humidity','Temperature (C)']]  
data.columns=['Humidity','Temperature (C)']  
data
```

	Humidity	Temperature (C)
0	0.89	9.472222
1	0.86	9.355556
2	0.89	9.377778
3	0.83	8.288889
4	0.83	8.755556
...
96448	0.43	26.016667
96449	0.48	24.583333
96450	0.56	22.038889
96451	0.60	21.522222
96452	0.61	20.438889

96453 rows × 2 columns



- 2. Split the Data:** Divide the dataset into training and testing sets. The training set is used to train the model, while the testing set is used to evaluate its performance.

```
corr=data.corr()  
corr
```

	Humidity	Temperature (C)
Humidity	1.000000	-0.632255
Temperature (C)	-0.632255	1.000000

- 3. Implement Linear Regression:** Write code to implement the linear regression algorithm from scratch. This involves defining a cost function (such as mean squared error) and minimizing it using optimization techniques like gradient descent. The parameters of the linear regression model (slope and intercept) are adjusted iteratively to minimize the cost function.
- 4. Train the Model:** Use the training data to train the linear regression model by fitting it to the training examples. This involves finding the optimal parameters that minimize the cost function.
- 5. Evaluate the Model:** Once the model is trained, evaluate its performance using the testing data. You can calculate metrics like mean squared error, R-squared, or others to assess how well the model generalizes to unseen data.

```
model.score(train_x,train_y)
```

```
0.40328663462236447
```

```
score=cross_val_score(model,train_x,train_y,scoring='neg_mean_absolute_error',cv=3)  
score=(-score)
```

```
score.mean()
```

```
6.019287664023691
```

```
test_x.shape
```

```
(64623, 1)
```

```
test_x=scaler.transform(test_x)  
test_predict=model.predict(test_x)  
score=mean_absolute_error(test_y,test_predict)  
score
```

```
6.01912644329253
```

Conclusion:

In this practical demonstration, we have discussed the theory behind linear regression and its implementation from scratch using Python. By understanding the fundamentals of the algorithm and its components, we can apply it to real-world datasets for predictive modeling and analysis.

Experiment- 7

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) salary.csv

Introduction:

Linear regression is a statistical method used to model the relationship between a dependent variable and one or more independent variables. In this practical demonstration, we aim to implement linear regression using Python libraries, specifically focusing on the "salary.csv" dataset. This dataset contains information about employees' years of experience and their corresponding salaries, making it suitable for demonstrating the application of linear regression in predicting salaries based on experience.

Linear Regression with Python Libraries:

1. Data Preparation:

The first step in implementing linear regression is to prepare the data. This involves loading the dataset from the provided .CSV file, examining its structure, and handling any missing or inconsistent values. In the case of the "salary.csv" dataset, we have two columns: "YearsExperience" and "Salary", where "YearsExperience" represents the independent variable and "Salary" represents the dependent variable.

2. Data Visualization:

Before applying linear regression, it's often helpful to visualize the data to gain insights into the relationship between the independent and dependent variables. We can create scatter plots to observe any linear trends or patterns in the data, which will inform our modeling approach.

3. Model Building:

Python offers several libraries for implementing linear regression, including scikit-learn, statsmodels, and TensorFlow. For this demonstration, we will use scikit-learn, a popular machine learning library in Python.

a. Model Training:

Using scikit-learn, we will split the dataset into training and testing sets to train our linear regression model. We will fit the model to the training data, allowing it to learn the relationship between years of experience and salary.

b. Model Evaluation:

After training the model, we will evaluate its performance using metrics such as mean squared error (MSE) or R-squared value. These metrics quantify how well the model predicts salaries based on the provided features.

4. Prediction:

Once the model is trained and evaluated, we can use it to make predictions on new data. We will demonstrate how to input new values for years of experience and obtain salary predictions from the trained model.

Conclusion:

In this practical demonstration, we have illustrated the implementation of linear regression using Python libraries, focusing on the "salary.csv" dataset. By leveraging Python's rich ecosystem of machine learning tools, we can efficiently build and evaluate linear regression models for predicting salaries based on years of experience. This approach showcases the power and flexibility of Python in performing data analysis and predictive modeling tasks.

Experiment- 8

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 and KNN. Compare the accuracy of both algorithm using confusion matrix.**

Logistic Regression Theory:

Logistic Regression is a statistical method used for binary classification problems. It models the relationship between the dependent binary variable and one or more independent variables by estimating probabilities using a logistic function, which is the cumulative logistic distribution.

The logistic regression model predicts $P(Y=1)$ as:

$$P(Y=1) = 1 / (1 + e^{(-z)})$$

where $z = b_0 + b_1x_1 + b_2x_2 + \dots + b_nx_n$

The coefficients $b_0, b_1, b_2, \dots, b_n$ are estimated using maximum likelihood estimation.

KNN Theory:

KNN (K-Nearest Neighbors) is a simple, instance-based learning algorithm used for classification and regression. The idea behind KNN is to find the 'k' nearest training samples to a test sample and classify the test sample based on the majority class of these 'k' nearest neighbors.

In KNN, the function to be learned is approximated locally and all computation is deferred until classification. To classify a new sample, the KNN algorithm finds the 'k' training samples closest to the new sample and assigns the new sample the same class as the majority of these 'k' nearest neighbors.

Now, let's move on to the implementation part. I'll provide you with a basic implementation of both algorithms in Python.

Logistic Regression Implementation:

```
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix

# Load dataset
data = pd.read_csv('your_file.csv')
X = data.drop('target', axis=1).values
y = data['target'].values

# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create Logistic Regression classifier
clf = LogisticRegression()

# Fit the classifier to the data
clf.fit(X_train, y_train)

# Predict the response for test dataset
y_pred = clf.predict(X_test)

# Evaluate the model
confusion_mat = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", confusion_mat)
```

KNN Implementation:

```
import numpy as np
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix

# Load dataset
data = pd.read_csv('your_file.csv')
```

```
X = data.drop('target', axis=1).values
y = data['target'].values

# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create KNN classifier
knn = KNeighborsClassifier(n_neighbors=5)

# Fit the classifier to the data
knn.fit(X_train, y_train)

# Predict the response for test dataset
y_pred = knn.predict(X_test)

# Evaluate the model
confusion_mat = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", confusion_mat)
```

EXPERIMENT 9

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 batch normalization, early stopping and drop out.

Theory:

1. Data Preparation:

In this step, the dataset is loaded and preprocessed. For the MNIST dataset, preprocessing usually involves normalizing the pixel values to a range between 0 and 1 to facilitate training. Additionally, the dataset is often split into training and testing sets

to evaluate the model's performance.

2. Model Architecture:

The model architecture defines the structure of the neural network, including the number of layers, the number of neurons in each layer, and the activation functions used. For example, a simple fully connected neural network for image classification tasks might consist of an input layer, one or more hidden layers, and an output layer. The hidden layers typically use activation functions like ReLU (Rectified Linear Unit) to introduce non-linearity.

3. Batch Normalization:

Batch normalization is a technique used to stabilize and speed up the training process of neural networks. It normalizes the inputs of each layer to have zero mean and unit variance. This helps in mitigating the problem of internal covariate shift, making the training process more efficient and allowing the use of higher learning rates.

4. Dropout:

Dropout is a regularization technique used to prevent overfitting in neural networks. During training, dropout randomly deactivates a fraction of neurons in the network with a specified probability. This prevents the network from relying too much on specific neurons and forces it to learn more robust features. Dropout effectively acts as an ensemble of multiple models, leading to improved generalization performance.

5. Early Stopping:

Early stopping is another regularization technique used to prevent overfitting by monitoring the model's performance on a validation dataset during training. It stops the training process when the performance on the validation set starts to degrade, indicating that the model is overfitting to the training data. Early stopping helps in finding the optimal balance between bias and variance, leading to better generalization performance on unseen data.

Implementation Steps:

1. Data Preparation: Load and preprocess the MNIST dataset.
2. Model Architecture: Design the ANN architecture with input, hidden, and output layers.
3. Batch Normalization: Add batch normalization layers after the activation function in each hidden layer.
4. Dropout: Add dropout layers after the activation function in each hidden layer.
5. Training: Train the model using the Backpropagation algorithm with early stopping.

6. Evaluation: Evaluate the model on a separate test set.

EXPERIMENT -10

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 batch normalization, early stopping and drop out.

Theory:

Sure, let's break down the steps to build an Artificial Neural Network (ANN) using the Backpropagation algorithm and test it using the CIFAR-100 dataset for multiclass classification. We'll incorporate batch normalization, early stopping, and dropout regularization. Here's a detailed guide:

1. Data Preparation:

CIFAR-100 is a dataset containing 60,000 32x32 color images in 100 different classes. First, you need to load and preprocess the CIFAR-100 dataset. Preprocessing typically involves normalization of pixel values and splitting the dataset into training and testing

sets.

2. Model Architecture:

Design the architecture of the neural network. For this task, a convolutional neural network (CNN) is commonly used due to its effectiveness in image classification tasks. You can design a CNN with multiple convolutional layers followed by pooling layers, followed by fully connected layers for classification. Each convolutional layer can be followed by batch normalization and dropout layers for regularization.

3. Batch Normalization:

Add batch normalization layers after the activation functions in each convolutional layer. Batch normalization helps in stabilizing and accelerating the training process by normalizing the activations.

4. Dropout:

Add dropout layers after the activation functions in each convolutional layer and fully connected layer. Dropout helps prevent overfitting by randomly dropping a fraction of neurons during training.

5. Early Stopping:

Implement early stopping by monitoring the performance of the model on a validation set during training. If the performance does not improve for a certain number of epochs, training can be stopped early to prevent overfitting.

Implementation Steps:

1. Data Loading and Preprocessing: Load CIFAR-100 dataset, normalize pixel values, and split into training and testing sets.
2. Model Definition: Design a CNN architecture with convolutional layers, pooling layers, fully connected layers, batch normalization, and dropout.
3. Model Compilation: Compile the model with appropriate loss function, optimizer, and metrics.
4. Training: Train the model on the training data with early stopping callback to prevent overfitting.
5. Evaluation: Evaluate the trained model on the testing data to assess its performance.

EXPERIMENT 11

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

→ Batch Normalization

Batch normalization is a technique that normalizes the inputs of each batch, which can help speed up learning, reduce the impact of initialization, and make the network more stable during training. Here's an example of how you might implement batch normalization in Keras:

```
from keras.layers import BatchNormalization
```

```

# add a batch normalization layer after each convolutional layer
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(64, 64, 3)))
model.add(BatchNormalization())
model.add(MaxPooling2D((2, 2)))

model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D((2, 2)))

model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(Flatten())

```

In this example, we're adding a batch normalization layer after each convolutional layer. This helps to normalize the outputs of each layer, which can improve the stability and speed of training.

→ Early Stopping

Early stopping is a technique that stops training when the validation loss stops improving, which can help prevent overfitting. Here's an example of how you might implement early stopping in Keras:

```

from keras.callbacks import EarlyStopping

# create an early stopping callback
early_stopping = EarlyStopping(monitor='val_loss', patience=3)

# compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# train the model with early stopping
model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_val, y_val),
callbacks=[early_stopping])

```

In this example, we're creating an early stopping callback that stops training when the validation loss hasn't improved for 3 epochs. We then compile the model and train it with the early stopping callback.

→ Dropout

Dropout is a technique that randomly drops out neurons during training, which can help prevent overfitting. Here's an example of how you might implement dropout in Keras:

```
from keras.layers import Dropout
```

```
# add a dropout layer after each convolutional layer
```

```
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(64, 64, 3)))
```

```
model.add(BatchNormalization())
```

```
model.add(MaxPooling2D((2, 2)))
```

```
model.add(Dropout(0.25))
```

```
model.add(Conv2D(64, (3, 3), activation='relu'))
```

```
model.add(BatchNormalization())
```

```
model.add(MaxPooling2D((2, 2)))
```

```
model.add(Dropout(0.25))
```

```
model.add(Conv2D(64, (3, 3), activation='relu'))
```

```
model.add(BatchNormalization())
```

```
model.add(Flatten())
```

```
model.add(Dropout(0.5))
```

In this example, we're adding a dropout layer after each convolutional layer. The dropout rate is set to 0.25 for the first two layers and 0.5 for the last layer. This means that 25% and 50% of the neurons will be randomly dropped out during training, respectively. By combining these techniques, you can create a more robust and accurate ANN for the Covid dataset.

EXPERIMENT 12

AIM:

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

THEORY:

Convolutional Neural Networks (CNNs) are a type of deep learning neural network architecture that is particularly well suited to image classification and object recognition tasks. A CNN works by transforming an input image into a feature map, which is then processed through multiple convolutional and pooling layers to produce a predicted output.

The **MNIST** dataset is a large database of handwritten digits that is commonly used for training various image processing systems. The dataset includes 60,000 training images and 10,000 test images. Each image is a 28x28 grayscale image of a single handwritten digit, from 0 to 9.

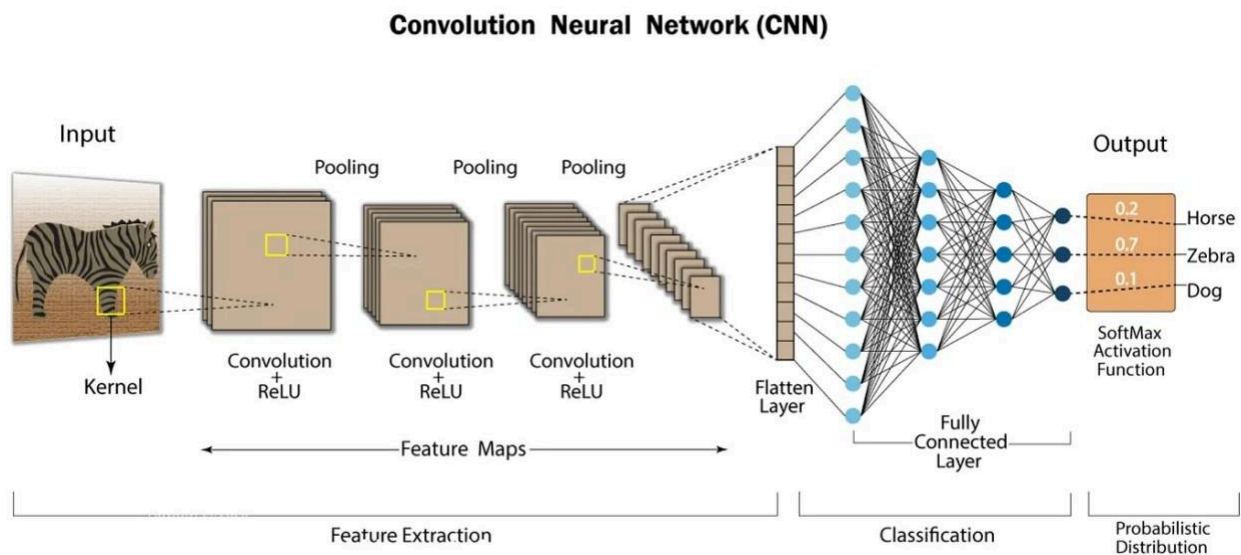
The MNIST dataset is a popular choice for training machine learning models because it is relatively small and easy to work with, while still being large enough to train effective models. Additionally, the MNIST dataset is well-labeled, which makes it easy to evaluate the performance of machine learning models.

Working of Convolutional Neural Network:

A convolutional neural network starts by taking an input image, which is then transformed into a feature map through a series of convolutional and pooling layers. The convolutional layer applies a set of filters to the input image, each filter producing a feature map that highlights a specific aspect of the input image. The pooling layer

then down samples the feature map to reduce its size, while retaining the most important information.

The feature map produced by the convolutional layer is then passed through multiple additional convolutional and pooling layers, each layer learning increasingly complex features of the input image. The final output of the network is a predicted class label or probability score for each class, depending on the task.



DESCRIPTION:

1. **Import Dependencies:** Necessary libraries are imported including TensorFlow, Matplotlib, Seaborn, NumPy, Pandas, and datetime.
2. **Load the Data:** MNIST dataset containing 60,000 training images and 10,000 test images of handwritten digits from 0 to 9 is loaded using TensorFlow's `mnist.load_data()`.
3. **Explore the Data:** Data exploration is performed by displaying sample images from the dataset using Matplotlib.
4. **Reshape the Data:** The data is reshaped to include color channels and normalized to $[0,1]$ range.
5. **Build the Model:** A Sequential Keras model is constructed with two pairs of Convolution2D and MaxPooling2D layers, followed by a Flatten layer, a Dense layer with ReLU activation, a Dropout layer, and a Dense layer with Softmax

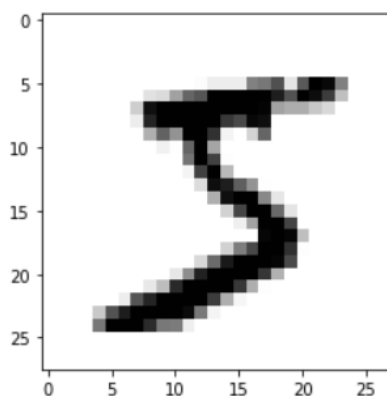
activation.

6. **Compile the Model:** The model is compiled with Adam optimizer, sparse categorical cross entropy loss, and accuracy metric.
7. **Train the Model:** The model is trained for 10 epochs using the training dataset, with validation performed on the test dataset. TensorBoard is used for visualization and debugging during training.
8. **Evaluate Model Accuracy:** The accuracy of the model is evaluated on both the training and test sets.
9. **Save the Model:** The trained model is saved in HDF5 format.
10. **Use the Model (Predictions):** The saved model is loaded and used to make predictions on the test dataset. Predictions are converted to class labels and visualized alongside the corresponding test images.
11. **Plot Confusion Matrix:** A confusion matrix is plotted to analyze the model's performance in recognizing different digits.
12. **Debugging with TensorBoard:** TensorBoard is used for debugging and visualization of model metrics during training.
13. **Conversion to Web-format:** The saved model is converted to a web-compatible format using tensorflowjs_converter for deployment on web applications.

RESULTS/GRAPHS:



```
plt.imshow(x_train[0], cmap=plt.cm.binary)  
plt.show()
```



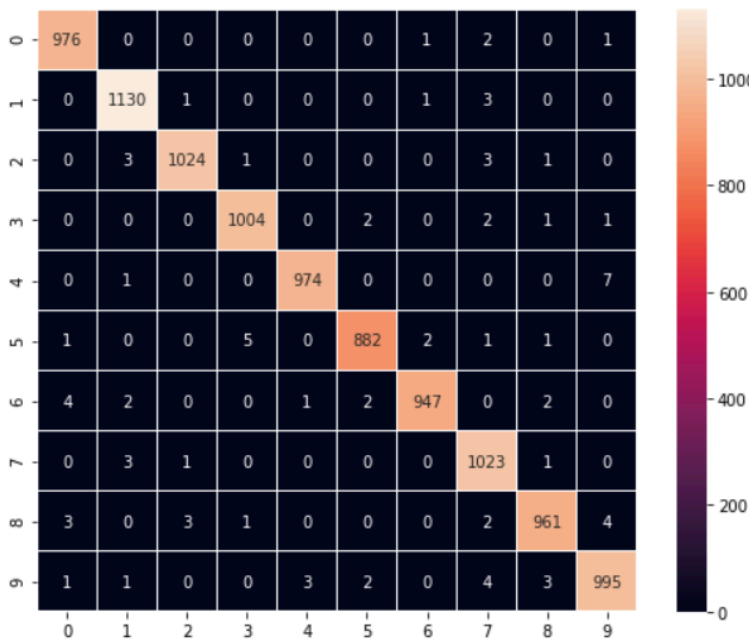
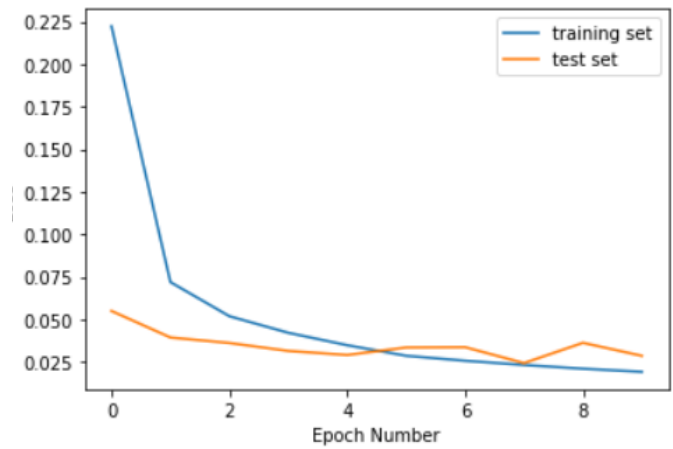
```
model.summary()
```

Model: "sequential"

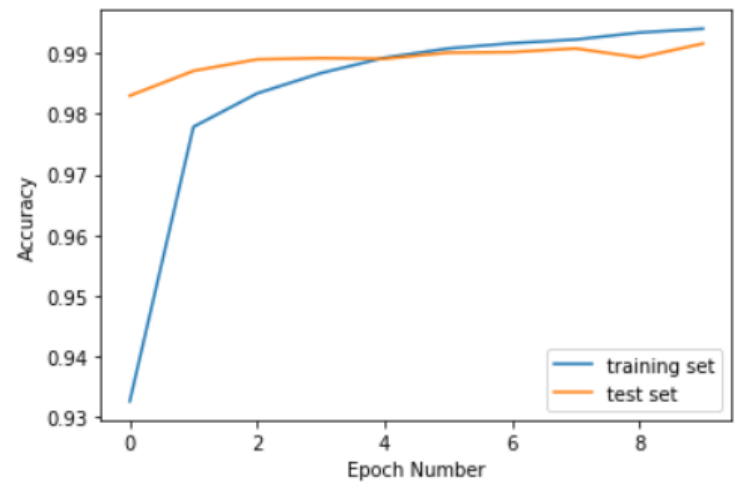
Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 24, 24, 8)	208
max_pooling2d (MaxPooling2D)	(None, 12, 12, 8)	0
conv2d_1 (Conv2D)	(None, 8, 8, 16)	3216
max_pooling2d_1 (MaxPooling2D)	(None, 4, 4, 16)	0
flatten (Flatten)	(None, 256)	0
dense (Dense)	(None, 128)	32896
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 10)	1290

Total params: 37,610
Trainable params: 37,610
Non-trainable params: 0

<matplotlib.legend.Legend at 0x1551d61d0>



<matplotlib.legend.Legend at 0x1551c77d0>



EXPERIMENT 13

AIM:

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

THEORY:

Convolutional Neural Networks (CNNs) are a type of deep learning neural network architecture that is particularly well suited to image classification and object recognition tasks.

The Backpropagation algorithm is a supervised learning method for multilayer feed-forward networks from the field of Artificial Neural Networks.

Feed-forward neural networks are inspired by the information processing of one or more neural cells, called a neuron. A neuron accepts input signals via its dendrites, which pass the electrical signal down to the cell body. The axon carries the signal out to synapses, which are the connections of a cell's axon to another cell's dendrites.

The principle of the backpropagation approach is to model a given function by modifying internal weightings of input signals to produce an expected output signal. The system is trained using a supervised learning method, where the error between the system's output and a known expected output is presented to the system and used to modify its internal state.

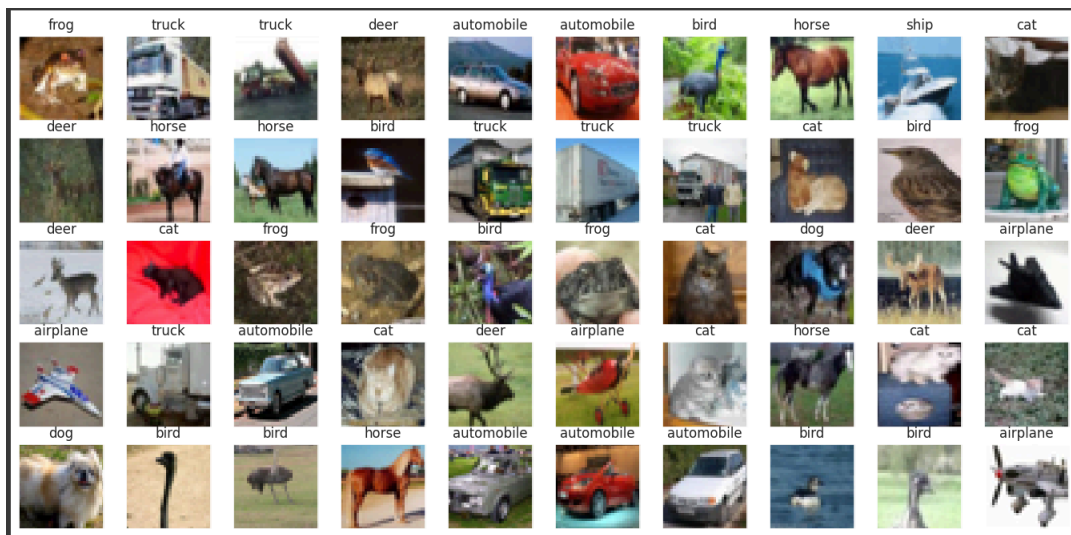
Technically, the backpropagation algorithm is a method for training the weights in a multilayer feed-forward neural network. As such, it requires a network structure to be defined of one or more layers where one layer is fully connected to the next layer. A

standard network structure is one input layer, one hidden layer, and one output layer.

Backpropagation can be used for both classification and regression problems, but we will focus on classification in this tutorial.

In classification problems, best results are achieved when the network has one neuron in the output layer for each class value. For example, a 2-class or binary classification problem with the class values of A and B. These expected outputs would have to be transformed into binary vectors with one column for each class value. Such as $[1, 0]$ and $[0, 1]$ for A and B respectively. This is called a one hot encoding.

The **CIFAR-100** dataset (Canadian Institute for Advanced Research, 100 classes) is a subset of the Tiny Images dataset and consists of 60000 32x32 color images. The 100 classes in the CIFAR-100 are grouped into 20 superclasses. There are 600 images per class. Each image comes with a "fine" label (the class to which it belongs) and a "coarse" label (the superclass to which it belongs). There are 500 training images and 100 testing images per class.





THEORY:

Imports and Setup:

The script begins with importing necessary libraries, including OpenCV (cv2), NumPy (numpy), Matplotlib (matplotlib.pyplot), Seaborn (seaborn), and Keras modules (keras).

The script sets the style for Seaborn plots using `sns.set()`.

Loading CIFAR-10 Dataset:

The CIFAR-10 dataset is loaded using Keras' `cifar10` module.

The dataset consists of 50,000 training images and 10,000 test images, each categorized into one of 10 classes (e.g., airplane, automobile, bird, cat, etc.).

Data Visualization:

The script visualizes a subset of images from the training dataset using Matplotlib.

It displays 50 images in a grid of 5 rows and 10 columns, with each image labeled with its corresponding class.

Data Preprocessing:

The script converts the images to grayscale using OpenCV's `cv2.cvtColor()` function.

Grayscale images are visualized to ensure the conversion was successful.

Normalization:

The script normalizes the pixel values of the images to the range [0, 1] by dividing them by 255.

One-Hot Encoding:

The script performs one-hot encoding on the class labels (`y_train` and `y_test`) using Scikit-learn's `OneHotEncoder` to convert them into binary vectors.

Model Definition:

The CNN model architecture is defined using Keras' Sequential API.

The model consists of multiple convolutional layers followed by max-pooling layers, fully connected layers, and dropout layers for regularization.

The final layer uses a softmax activation function to output class probabilities.

Model Compilation:

The model is compiled with appropriate loss function, optimizer, and evaluation metric using Keras' `compile()` method.

Model Training:

The model is trained on the training data (`X_train` and `y_train`) using Keras' `fit()` method.

The training process includes specifying the number of epochs, batch size, and optional early stopping criteria.

Model Evaluation:

The trained model is evaluated on the test data (`X_test` and `y_test`) using Keras' `evaluate()` method.

The evaluation results, including test loss and test accuracy, are printed to the console.

Visualizing Model Performance:

The script plots the validation loss and validation accuracy history using Matplotlib to

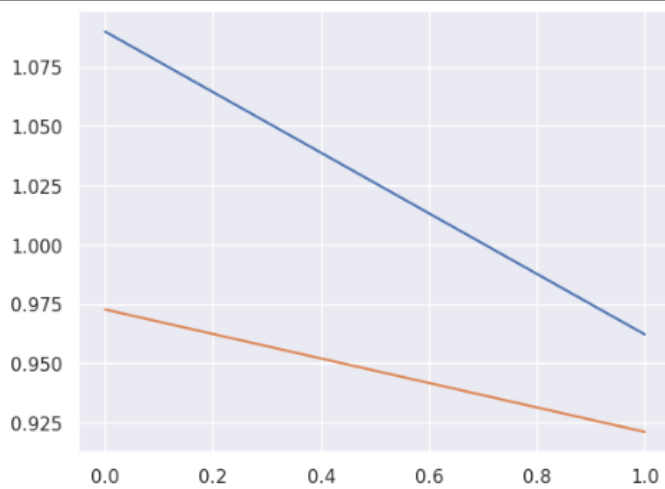
visualize the model's performance during training.

Visualizing Predictions:

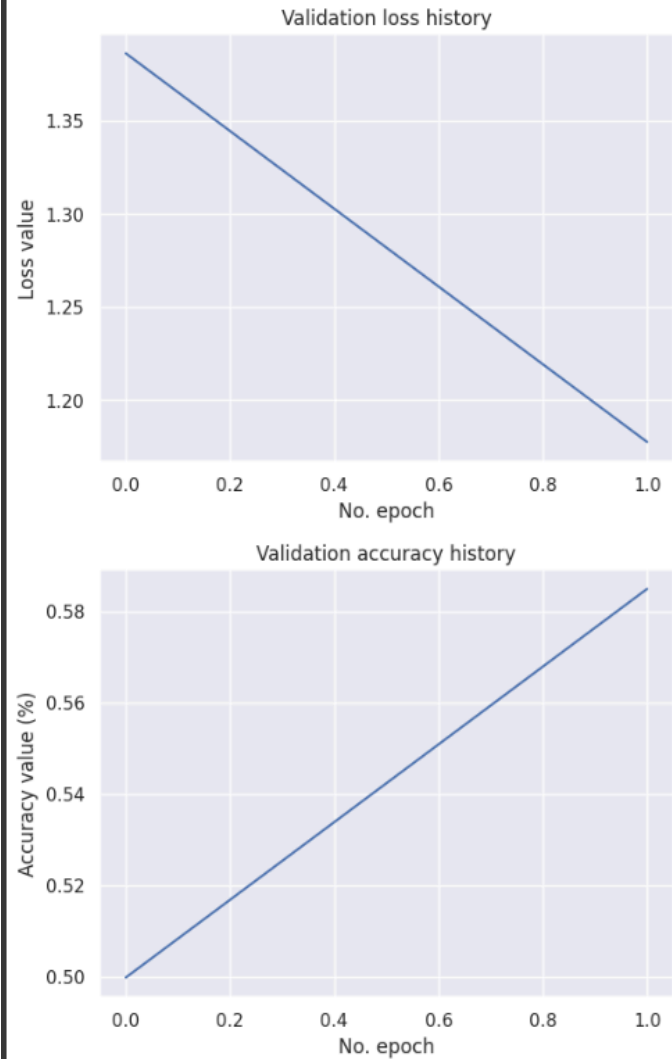
The script displays a subset of images from the test dataset along with their actual and predicted labels using Matplotlib.

RESULTS/GRAPHS:

```
plt.plot(history.history["loss"])  
plt.plot(history.history["val_loss"])  
plt.show()
```



Test loss: 1.1772819757461548 / Test accuracy: 0.5849000215530396



Actual	airplane	763	10	62	18	40	2	8	8	63	26
	automobile	29	818	3	10	8	2	17	0	13	100
	bird	81	4	496	74	226	37	38	21	16	7
	cat	38	4	80	506	138	104	64	30	16	20
	deer	21	3	48	52	776	18	15	61	4	2
	dog	9	1	65	240	112	490	25	52	3	3
	frog	15	7	44	85	145	15	670	3	7	9
	horse	15	1	38	54	104	42	1	725	6	14
	ship	85	35	16	20	8	4	8	7	793	24
	truck	50	50	5	20	8	3	2	19	18	825
		airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck
		Predicted									

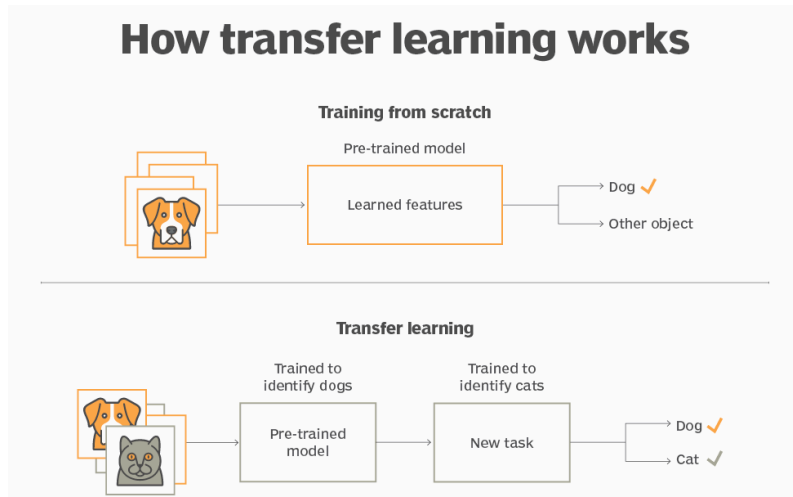
Conclusion

The results of the experiment should be interpreted based on the training loss convergence and test set accuracy. These metrics provide insights into the learning progress and generalization ability of the CNN model on a challenging multiclass image classification task like CIFAR-100. Adjustments to the model architecture, hyperparameters, or training strategy may be needed based on the observed results to further enhance performance.

EXPERIMENT 14

AIM: Implementation of Transfer Learning.

Transfer learning is a machine learning technique where a model trained on one task is re-purposed on a second related task. Instead of starting the learning process from scratch, you start from patterns learned from solving a related task. It's a popular approach in deep learning, especially in scenarios where you don't have enough data to train a model from scratch or when training a model from scratch is computationally expensive.



- **Pre-trained Model:** Start with a model that has previously been trained for a certain task using a large set of data. Frequently trained on extensive datasets, this model has identified general features and patterns relevant to numerous related jobs.
- **Base Model:** The model that has been pre-trained is known as the base model. It is made up of layers that have utilized the incoming data to learn hierarchical feature representations.
- **Transfer Layers:** In the pre-trained model, find a set of layers that capture generic information relevant to the new task as well as the previous one. Because they are prone to learning low-level information, these layers are frequently found near the top of the network.
- **Fine-tuning:** Using the dataset from the new challenge to retrain the chosen layers. We define this procedure as fine-tuning. The goal is to preserve the knowledge

from the pre-training while enabling the model to modify its parameters to better suit the demands of the current assignment.

TensorFlow is an open-source framework that is used for Machine Learning. It provides a range of functions to achieve complex functionalities with single lines of code.

Import required libraries and the MNIST dataset, a dataset of handwritten digits often used for training and testing machine learning models.

Using TensorFlow's Keras API, this code builds a convolutional neural network (CNN). Layers for reshaping, convolution, pooling, flattening, and fully connected operations are included. Dropout is used to achieve regularisation. Using softmax activation, the model, which is ideal for image classification like MNIST, generates class probabilities. The design achieves a compromise between feature extraction and categorization, allowing for successful learning and generalization.

Advantages of transfer learning:

- Speed up the training process: By using a pre-trained model, the model can learn more quickly and effectively on the second task, as it already has a good understanding of the features and patterns in the data.
- Better performance: Transfer learning can lead to better performance on the second task, as the model can leverage the knowledge it has gained from the first task.
- Handling small datasets: When there is limited data available for the second task, transfer learning can help to prevent overfitting, as the model will have already learned general features that are likely to be useful in the second task.

Disadvantages of transfer learning:

- **Domain mismatch:** The pre-trained model may not be well-suited to the second task if the two tasks are vastly different or the data distribution between the two tasks is very different.

- **Overfitting:** Transfer learning can lead to overfitting if the model is fine-tuned too much on the second task, as it may learn task-specific features that do not generalize well to new data.
- **Complexity:** The pre-trained model and the fine-tuning process can be computationally expensive and may require specialized hardware.

Output:

```
# print the classification
print('%s (%.2f%%)' % (label[1], label[2]*100))
```

⇒ 1/1 [=====] - 1s 983ms/step
 Downloading data from https://storage.googleapis.com/download.tensorflow.org/images/tiger_cat.jpg
 35363/35363 [=====] - 0s 0us/step
 tiger_cat (30.23%)

Total Parameter:

```
=====
Total params: 138357544 (527.79 MB)
Trainable params: 138357544 (527.79 MB)
Non-trainable params: 0 (0.00 Byte)
```

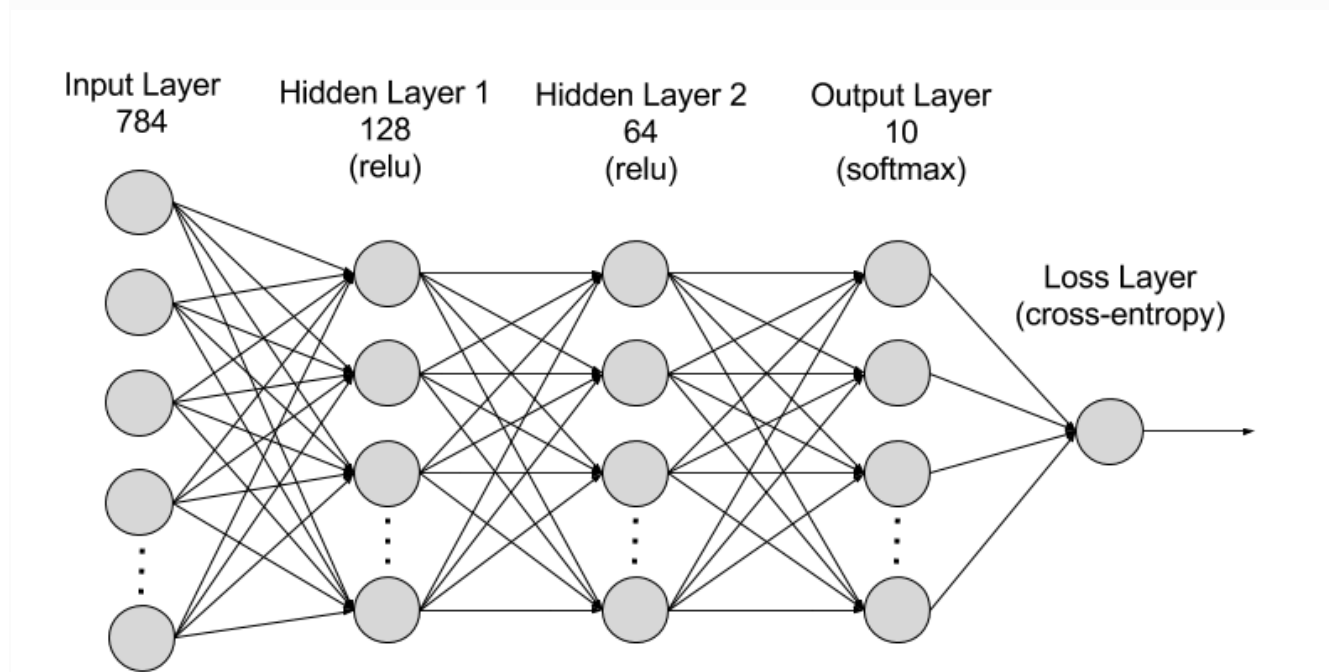
EXPERIMENT 15

AIM: Implementation of RNN

Recurrent Neural Network(RNN) is a type of Neural Network where the output from the previous step is fed as input to the current step. In traditional neural networks, all the inputs and outputs are independent of each other. Still, in cases when it is required to predict the next word of a sentence, the previous words are required and hence there is a need to remember the previous words. Thus RNN came into existence, which solved this issue with the help of a Hidden Layer. The main and most important feature of RNN is its **Hidden state**, which remembers some information about a sequence. The state is also referred to as the *Memory State* since it remembers the previous input to the network. It uses the same parameters for each input as it performs the same task on all the inputs or hidden layers to produce the output. This reduces the complexity of parameters, unlike other neural networks.

The following image shows a diagram of an RNN.:





RNNs are made of neurons: data-processing nodes that work together to perform complex tasks. The neurons are organized as input, output, and hidden layers. The input layer receives the information to process, and the output layer provides the result. Data processing, analysis, and prediction take place in the hidden layer.

Hidden layer

RNNs work by passing the sequential data that they receive to the hidden layers one step at a time. However, they also have a self-looping or *recurrent* workflow: the hidden layer can remember and use previous inputs for future predictions in a short-term memory component. It uses the current input and the stored memory to predict the next sequence.

For example, consider the sequence: *Apple is red*. You want the RNN to predict *red* when it receives the input sequence *Apple is*. When the hidden layer processes the word *Apple*, it stores a copy in its memory. Next, when it sees the word *is*, it recalls *Apple* from its memory and understands the full sequence: *Apple is* for context. It can then predict *red* for improved accuracy. This makes RNNs useful in speech recognition, machine translation, and other language modeling tasks.

Training

Machine learning (ML) engineers train deep neural networks like RNNs by feeding the model with training data and refining its performance. In ML, the neuron's weights are signals to determine how influential the information learned during training is when predicting the output. Each layer in an RNN shares the same weight.

ML engineers adjust weights to improve prediction accuracy. They use a technique called backpropagation through time (BPTT) to calculate model error and adjust its weight accordingly. BPTT rolls back the output to the previous time step and recalculates the error rate. This way, it can identify which hidden state in the sequence is causing a significant error and readjust the weight to reduce the error margin.

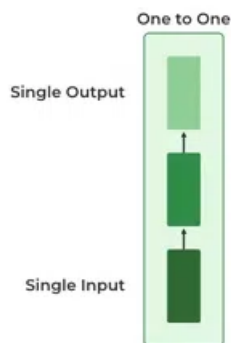
Types Of RNN

There are four types of RNNs based on the number of inputs and outputs in the network.

- One to One
- One to Many
- Many to One
- Many to Many

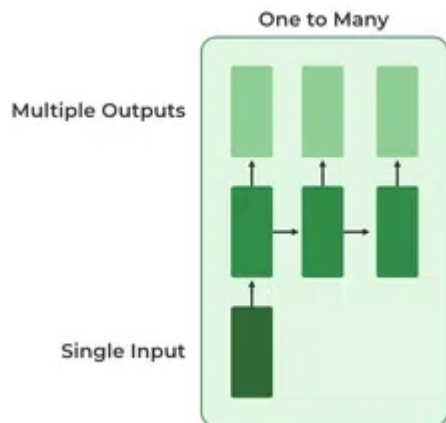
One to One

This type of RNN behaves the same as any simple Neural network it is also known as Vanilla Neural Network. In this Neural network, there is only one input and one output.



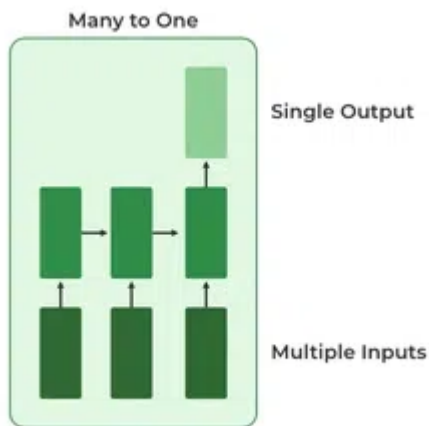
One To Many

In this type of RNN, there is one input and many outputs associated with it. One of the most used examples of this network is Image captioning where given an image we predict a sentence having Multiple words.



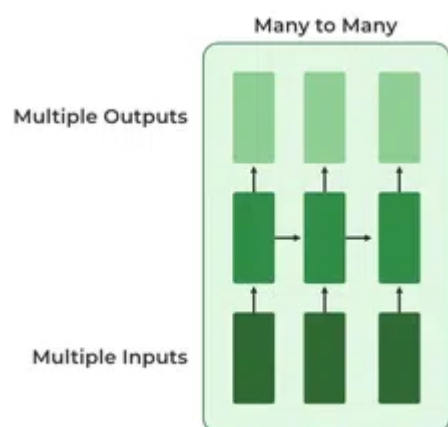
Many to One

In this type of network, Many inputs are fed to the network at several states of the network generating only one output. This type of network is used in the problems like sentimental analysis. Where we give multiple words as input and predict only the sentiment of the sentence as output.



Many to Many

In this type of neural network, there are multiple inputs and multiple outputs corresponding to a problem. One Example of this Problem will be language translation. In language translation, we provide multiple words from one language as input and predict multiple words from the second language as output.



PROGRAM/ IPYNB FILE:

RESULTS/ GRAPHS: