Lab Manual of Machine Learning [CS-602]

B. Tech. VI Semester

Jan - June 2024

Department of Computer Science and Information Technology

Submitted to:
Prof. Shilpa Bhalerao
HOD, CSIT Dept.

Submitted By:
Janvi Garg
0827CI211087

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 BTech III year syllabus prescribed by the RGPV was done by Ms. Janvi Garg BTech VI semester CI in the Machine Learning Laboratory of this institute during the academic year Jan-June 2024.

Signature of Faculty

INDEX PAGE

Programs to be uploaded on Github
Github Link:

Experiment No.	Title	Sign of faculty
1	How to setup 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, Mat plotlib 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**	

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

Experiment- 1

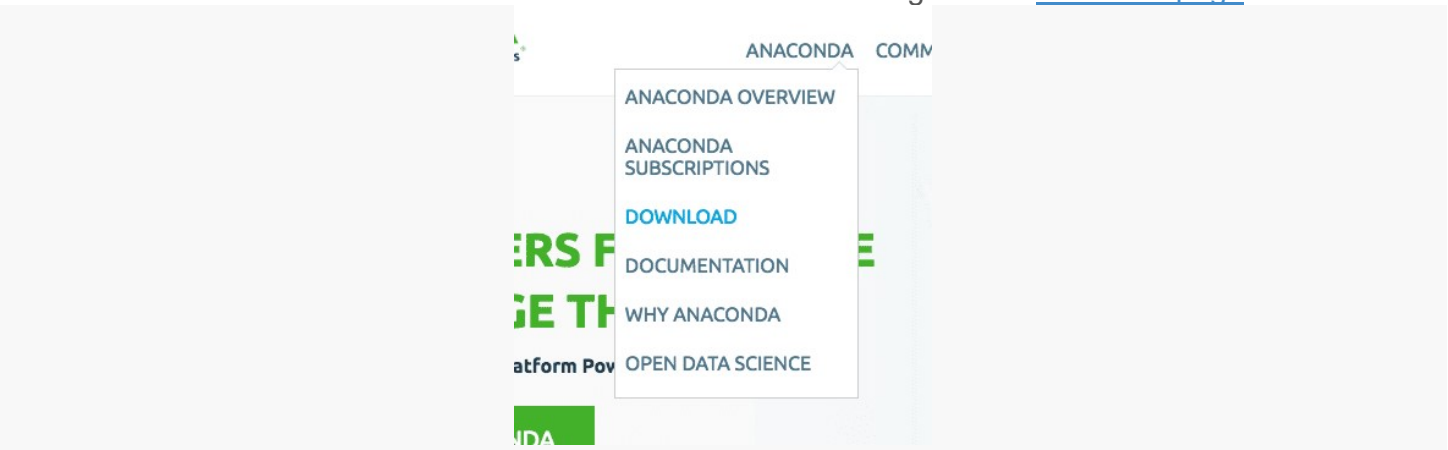
Aim: Setting up a Python environment for Machine Learning and Deep Learning with Anaconda .

1. Download Anaconda

In this step, we will download the Anaconda Python package for your platform.

Anaconda is a free and easy-to-use environment for scientific Python.

- 1. Visit the Anaconda homepage.
- 2. Click “Anaconda” from the menu and click “Download” to go to the download page.



-

This will download the Anaconda Python package to your workstation.

I’m on OS X, so I chose the OS X version. The file is about 426 MB.

You should have a file with a name like:

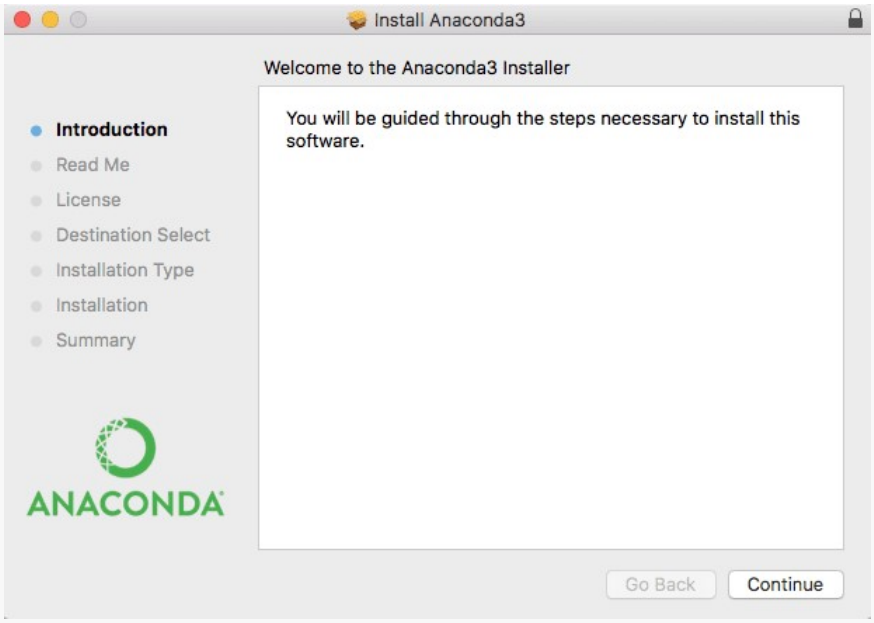
1Anaconda3-4.2.0-MacOSX-x86_64.pkg

2. Install Anaconda

In this step, we will install the Anaconda Python software on your system.

This step assumes you have sufficient administrative privileges to install software on your system.

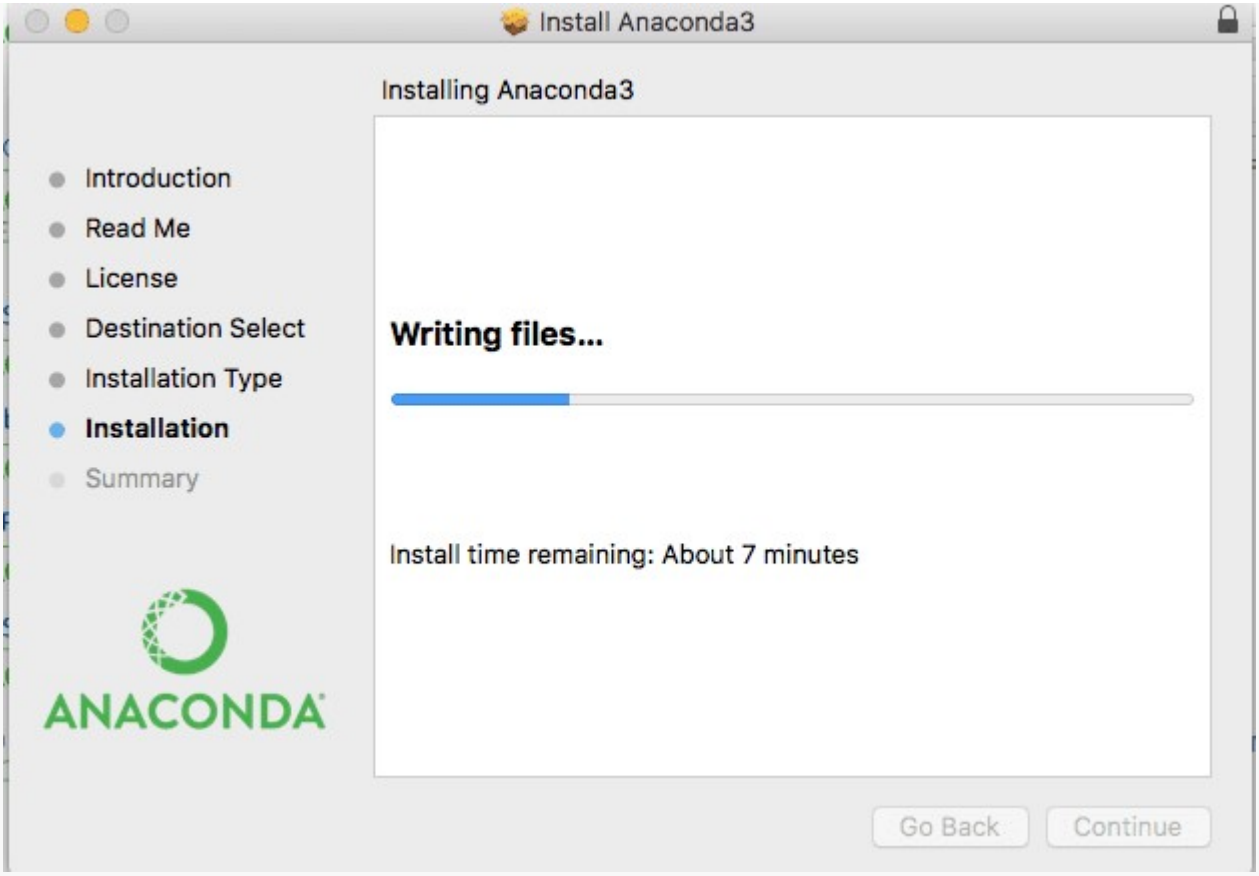
- 1. Double click the downloaded file.
- 2. Follow the installation wizard.



Anaconda Python Installation Wizard

Installation is quick and painless.

There should be no tricky questions or sticking points.



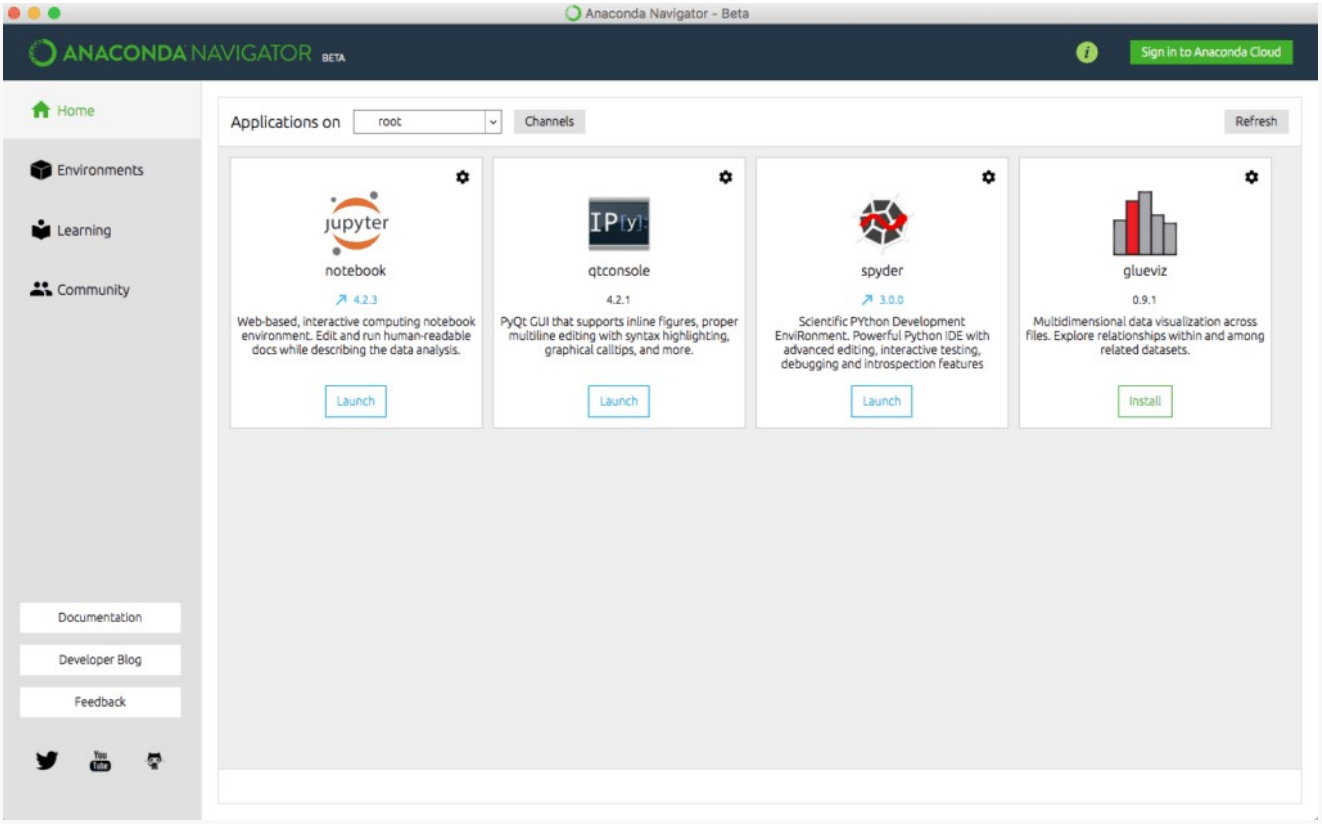
Anaconda Python Installation Wizard Writing Files

The installation should take less than 10 minutes and take up a little more than 1 GB of space on your hard drive.

3. Start and Update Anaconda

In this step, we will confirm that your Anaconda Python environment is up to date.

Anaconda comes with a suite of graphical tools called Anaconda Navigator. You can start Anaconda Navigator by opening it from your application launcher.



Anaconda Navigator GUI

You can learn all about the Anaconda Navigator here.
You can use the Anaconda Navigator and graphical development environments later; for now, I recommend starting with the Anaconda command line environment called conda.
Conda is fast, simple, it's hard for error messages to hide, and you can quickly confirm your environment is installed and working correctly.

- 1. Open a terminal (command line window).
- 2. Confirm conda is installed correctly, by typing:

```
conda -V
```

You should see the following (or something similar):

```
conda 4.2.9
```

- 3. Confirm Python is installed correctly by typing:

```
python -V
```

You should see the following (or something similar):

```
Python 3.5.2 :: Anaconda 4.2.0 (x86_64)
```

Confirm Conda and Python are Installed

If the commands do not work or have an error, please check the documentation for help for your platform.

See some of the resources in the “Further Reading” section.

- 4. Confirm your conda environment is up-to-date, type:

```
conda update conda
conda update anaconda
```

You may need to install some packages and confirm the updates.

- 5. Confirm your SciPy environment.

The script below will print the version number of the key SciPy libraries you require for machine learning development, specifically: SciPy, NumPy, Matplotlib, Pandas, Statsmodels, and Scikit-learn.

You can type “python” and type the commands in directly. Alternatively, I recommend opening a text editor and copy-pasting the script into your editor.

```
# scipy
import scipy
print('scipy: %s' % scipy.__version__)
# numpy
import numpy
print('numpy: %s' % numpy.__version__)
# matplotlib
import matplotlib
print('matplotlib: %s' % matplotlib.__version__)
# pandas
import pandas
print('pandas: %s' % pandas.__version__)
# statsmodels
import statsmodels
print('statsmodels: %s' % statsmodels.__version__)
# scikit-learn
import sklearn
print('sklearn: %s' % sklearn.__version__)
```

Save the script as a file with the name: *versions.py*.

On the command line, change your directory to where you saved the script and type:

```
python versions.py
```

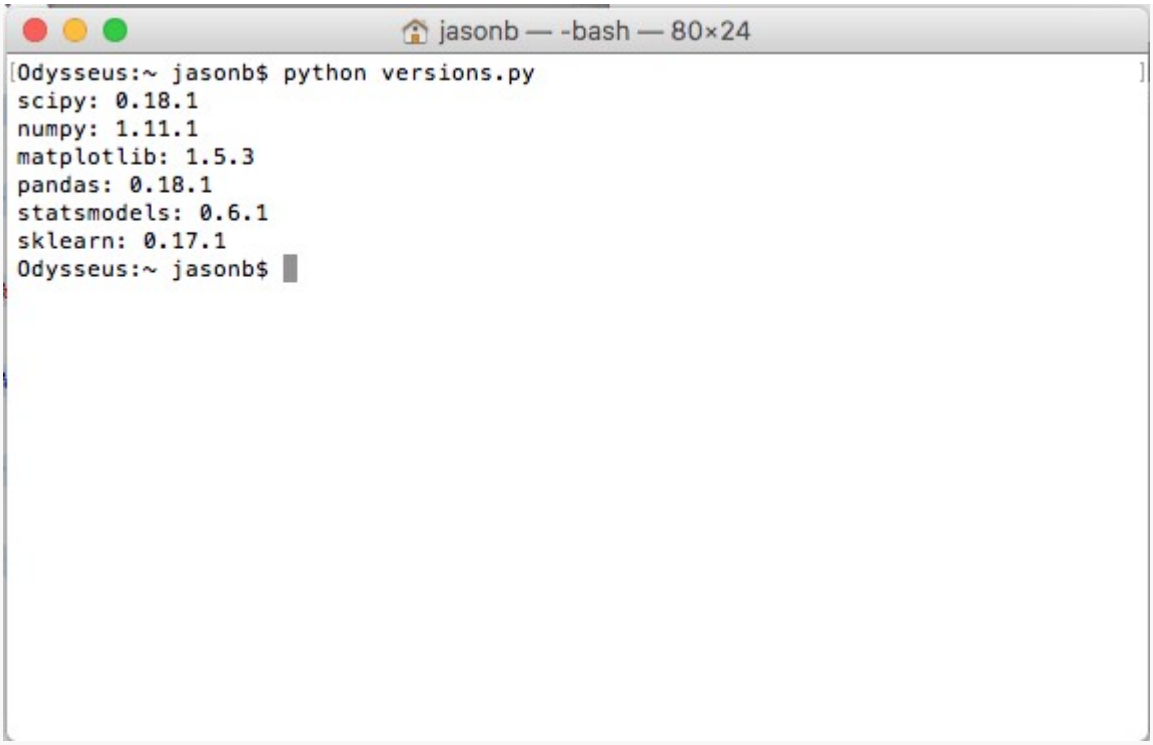
You should see output like the following:

```
scipy: 0.18.1
numpy: 1.11.1
matplotlib: 1.5.3
pandas: 0.18.1
statsmodels: 0.6.1
```

```
sklearn: 0.17.1
```

What versions did you get?

Paste the output in the comments below.



Confirm Anaconda SciPy environment

4. Update scikit-learn Library

In this step, we will update the main library used for machine learning in Python called scikit-learn.

- 1. Update scikit-learn to the latest version.

At the time of writing, the version of scikit-learn shipped with Anaconda is out of date (0.17.1 instead of 0.18.1). You can update a specific library using the conda command; below is an example of updating scikit-learn to the latest version.

At the terminal, type:

```
conda update scikit-learn
```

Update scikit-learn in Anaconda

Alternatively, you can update a library to a specific version by typing:

```
conda install -c anaconda scikit-learn=0.18.1
```

Confirm the installation was successful and scikit-learn was updated by re-running the *versions.py* script by typing:

```
python versions.py
```

You should see output like the following:

```
scipy: 0.18.1
numpy: 1.11.3
matplotlib: 1.5.3
pandas: 0.18.1
statsmodels: 0.6.1
sklearn: 0.18.1
```

What versions did you get?

Paste the output in the comments below.

You can use these commands to update machine learning and SciPy libraries as needed.

Try a scikit-learn tutorial, such as:

- Your First Machine Learning Project in Python Step-By-Step

5. Install Deep Learning Libraries

In this step, we will install Python libraries used for deep learning, specifically: Theano, TensorFlow, and Keras.

NOTE: I recommend using Keras for deep learning and Keras only requires one of Theano or TensorFlow to be installed. You do not need both! There may be problems installing TensorFlow on some Windows machines.

- 1. Install the Theano deep learning library by typing:

```
conda install theano
```

- 2. Install the TensorFlow deep learning library (all except Windows) by typing:

```
conda install -c conda-forge tensorflow
```

Alternatively, you may choose to install using pip and a specific version of tensorflow for your platform.

See the installation instructions for tensorflow.

- 3. Install Keras by typing:

```
pip install keras
```

- 4. Confirm your deep learning environment is installed and working correctly.

Experiment 2

Aim : 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.

Data Structures are a way of organizing data so that it can be accessed more efficiently depending upon the situation. Data Structures are fundamentals of any programming language around which a program is built. Python helps to learn the fundamental of these data structures in a simpler way as compared to other programming languages.

Lists

Python Lists are just like the arrays, declared in other languages which is an ordered collection of data. It is very flexible as the items in a list do not need to be of the same type.

Example:

```
List = [1, 2,  3, "GFG", 2.3]
print(List)
```

Tuple

Python Tuple is a collection of Python objects much like a list but Tuples are immutable in nature i.e. the elements in the tuple cannot be added or removed once created. Just like a List, a Tuple can also contain elements of various types.
In Python, tuples are created by placing a sequence of values separated by 'comma' with or without the use of parentheses for grouping of the data sequence.

Dictionary

Python dictionary is like hash tables in any other language with the time complexity of O(1). It is an unordered collection of data values, used to store data values like a map, which, unlike other Data Types that hold only a single value as an element, Dictionary holds the key:value pair. Key-value is provided in the dictionary to make it more optimized.

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 ( ).

Strings:
Definition: Strings are sequences of characters. They are immutable like tuples.
Initialization: Strings are created by enclosing characters inside single ' ' or double " " quotes.

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.

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.

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."
```

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.

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.

Experiment- 4

Aim: Basic of Numpy, Pandas and Matplotlib

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()
```

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

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

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.

$$y = \theta_0+\theta_1x_1+\theta_2x_2+...+\theta_nx_n$$

Where:

- y **is the dependent variable.**
- $x_1 , x_2,...,x_n$ **are the independent variables.**
- $\theta_0,\theta_1,...,\theta_n$ **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 ith 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.

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
```

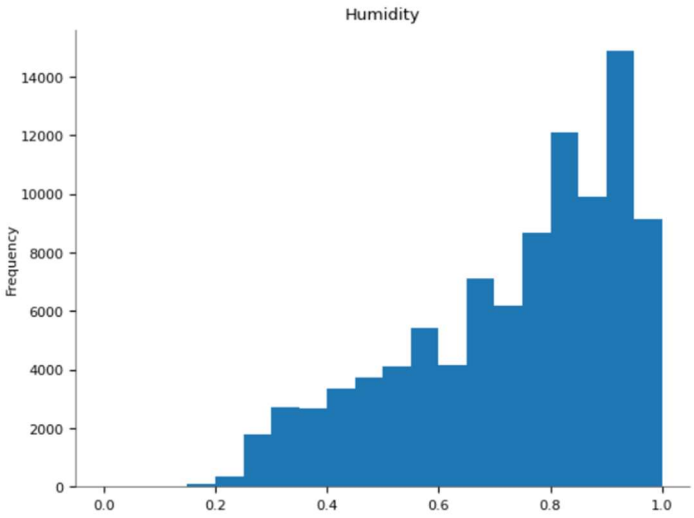
```
1157436
```

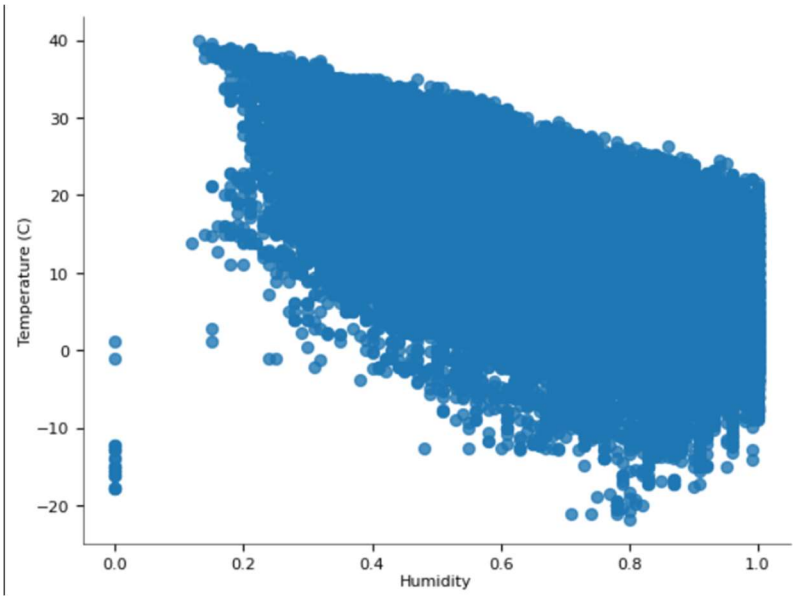
2. **Data Preprocessing:** Check for any missing values or outliers in the data. If there are missing values, you may need to handle them by imputing or removing the respective rows. Outliers may also need to be treated appropriately depending on their impact on the model.

```
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





a.

3. **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

4. **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.
5. **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.
6. **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.

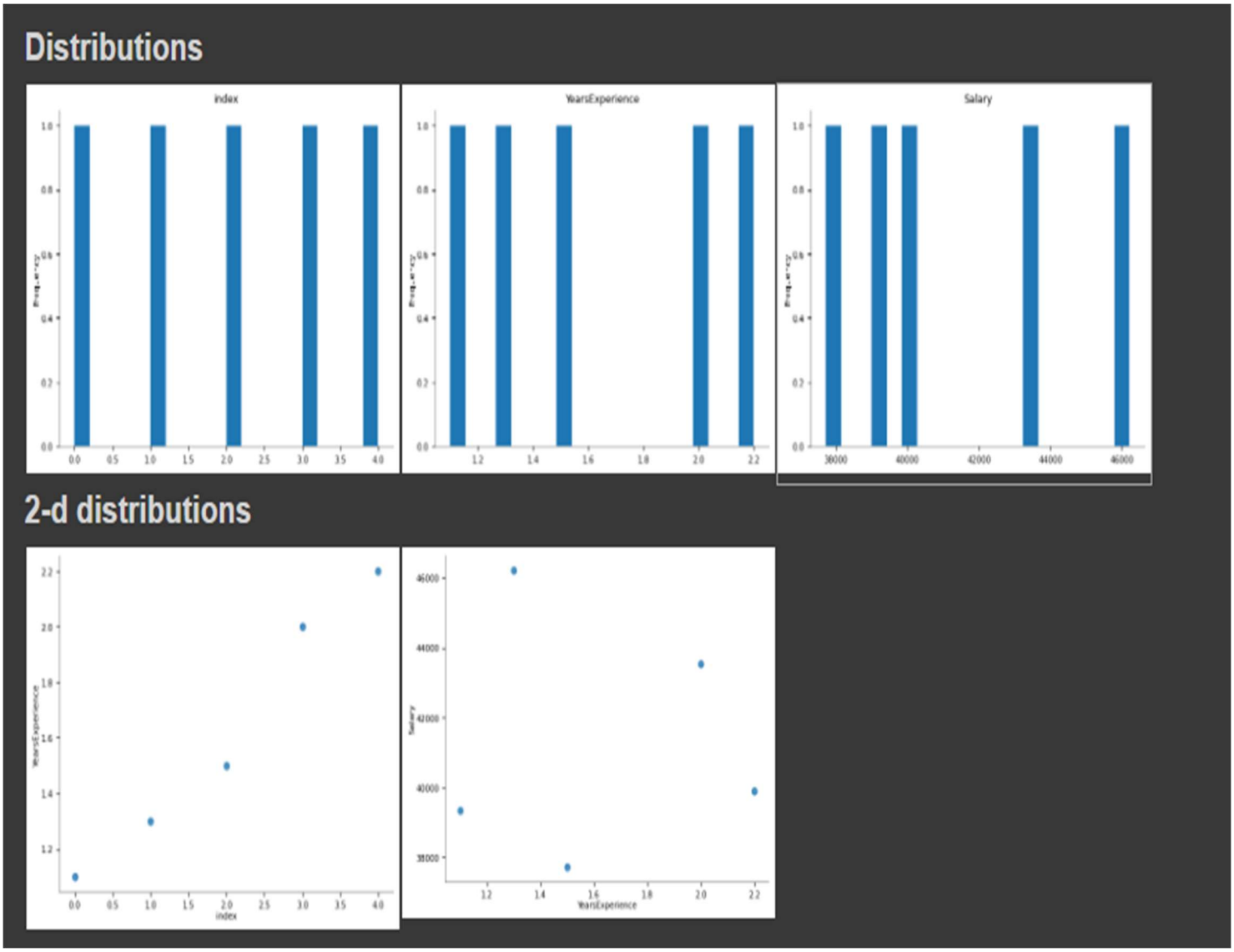
Experiment- 7

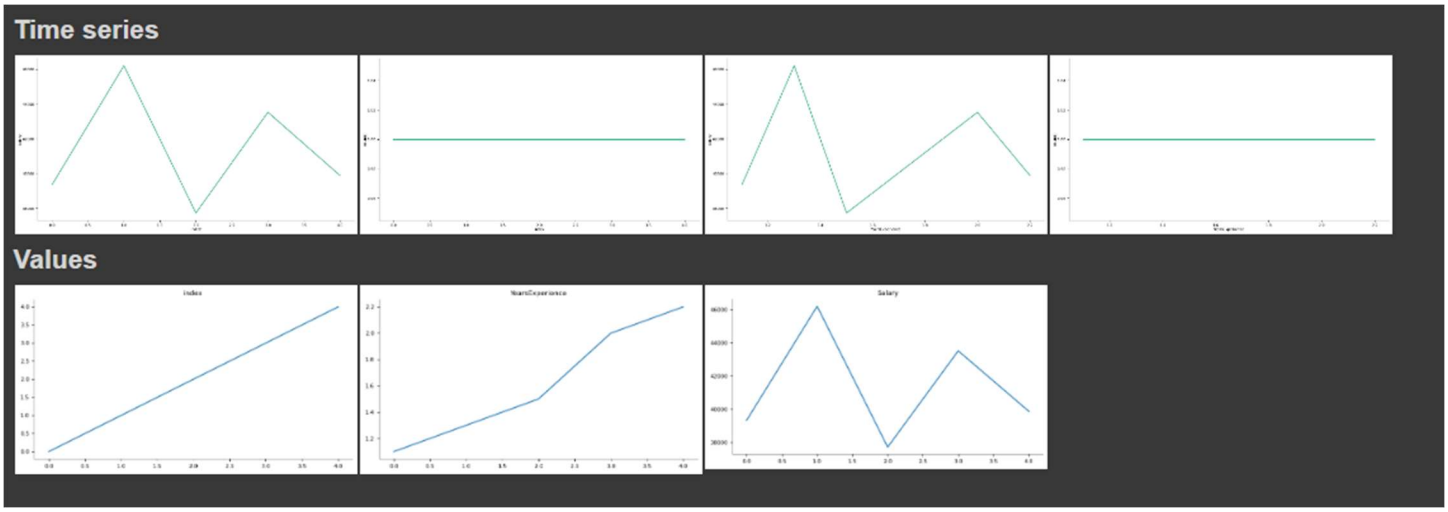
Aim: For a given set of training data examples stored in a .CSV file, implement and demonstrate the Implementation of Linear Regression Algorithm Linear Regression using Python library (for any given CSV dataset) salary.csv.

Description:

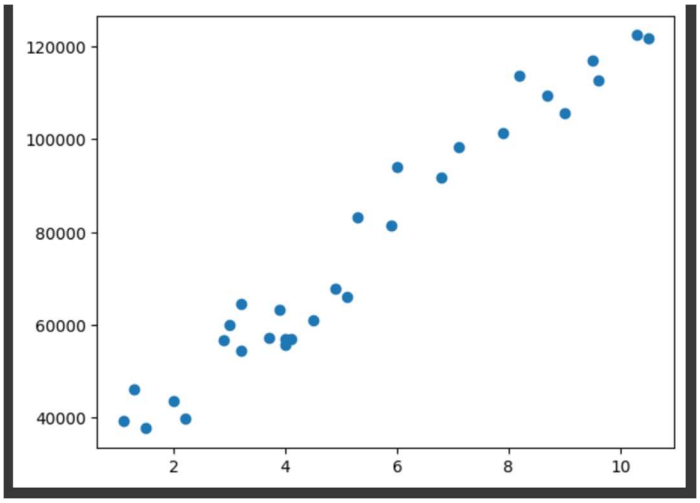
1. **Load the Data:** Read the salary.csv file to extract the training data. Each row represents a data example, with Years of experience and salary as features, and the target variable could be something like precipitation, temperature at a certain time, or any other-related metric.

	YearsExperience	Salary
0	1.1	39343.0
1	1.3	46205.0
2	1.5	37731.0
3	2.0	43525.0
4	2.2	39891.0





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.



3. **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
```

	YearsExperience	Salary
YearsExperience	1.000000	0.978242
Salary	0.978242	1.000000

4. **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.
5. **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.
6. **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)

    0.9549236946181227

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

[ ] score.mean()

    4659.518797475633

[ ] test_x.shape

    (20, 1)

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

    4472.274104679314
```

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 modelling tasks.

Experiment- 8

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 and KNN. Compare the accuracy of both algorithms 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 = b0 + b1x1 + b2x2 + ... + bn*xn
The coefficients b0, b1, b2, ..., bn are estimated using maximum likelihood estimation.

Implementation Steps:

1. **Importing Libraries:** The script starts by importing the necessary libraries, including NumPy, Pandas, Seaborn, Matplotlib, and various modules from Scikit-learn.

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split,cross_val_score
from sklearn.metrics import accuracy_score,confusion_matrix
from sklearn.linear_model import LogisticRegression
```

2. **Loading and Exploring the Dataset:** The script reads the "TrainData.csv" file from the user's Google Drive and stores it in the DF1 DataFrame. It then creates a copy of the dataset in the Data DataFrame and performs the following actions:Checks for missing values using

```
DF1=pd.read_csv("/content/drive/MyDrive/TrainData.csv")
Data=DF1.copy()
Data.isnull().sum()
```

```
OilQual        0
EnginePerf     0
NormMileage    0
TyreWear       0
HVACwear       0
Service        0
dtype: int64
```

3. **Handling Missing Values:** The script identifies the rows with missing values using Data[Data.isnull().any(axis=1)] and stores them in the missing DataFrame. It then creates a new DataFrame data2 by dropping the rows with missing values using Data.dropna(axis=0).

```
Data.isnull().sum()
missing=Data[Data.isnull().any(axis=1)]
```

```
data2=Data.dropna(axis=0)
data2.describe()
data2.isnull().sum()
```

```
OilQual        0
EnginePerf     0
NormMileage    0
TyreWear       0
HVACwear       0
Service        0
dtype: int64
```

4. **Crosstabulation:** The script creates a crosstabulation of the "OilQual" and "EnginePerf" columns, normalizing the values by row using pd.crosstab(index=data2['OilQual'], columns=data2['EnginePerf'], margins=True, normalize="index").

```
age_pricesat=pd.crosstab(index= data2['OilQual'], columns=data2['EnginePerf'],margins=True, normalize= "index")
print(age_pricesat[0:3])
```

```
EnginePerf   1.891003    2.841003    2.891003    3.021003    3.461003    \
OilQual
0.987185029        0.0         0.0         0.0         0.0         0.0
2.027185029        0.0         0.0         0.0         0.0         0.0
2.987185029        0.0         0.0         0.0         0.0         0.0

EnginePerf   3.891003    4.271003    4.511003    4.711003    4.891003    ...  \
OilQual                                                                  ...
0.987185029        0.0         0.0         0.0         0.0         1.0  ...
2.027185029        0.0         0.0         0.0         0.0         1.0  ...
2.987185029        0.0         0.0         0.0         0.5         0.5  ...

EnginePerf   104.384032  104.454032  104.474032  104.514032  104.664032  \
OilQual
0.987185029        0.0         0.0         0.0         0.0         0.0
2.027185029        0.0         0.0         0.0         0.0         0.0
2.987185029        0.0         0.0         0.0         0.0         0.0

EnginePerf   104.694032  104.744032  105.414032  105.474032  105.744032
OilQual
0.987185029        0.0         0.0         0.0         0.0         0.0
2.027185029        0.0         0.0         0.0         0.0         0.0
2.987185029        0.0         0.0         0.0         0.0         0.0

[3 rows x 150 columns]
```

5. **Data Preprocessing:** The script maps the "Yes" and "No" values in the "Service" column to 1 and 0, respectively, using data2['Service']=data2['Service'].map({'Yes':1,'No':0}). It then creates a new DataFrame newdata by applying one-hot encoding to the categorical variables using pd.get_dummies(data2, drop_first=True).

```
data2['Service']=data2['Service'].map({'Yes':1,'No':0})
print(data2['Service'])
#data2.describe(include='O')
```

```
0      0
1      1
2      1
3      0
4      0
      ..
310    0
311    0
312    1
313    1
314    1
Name: Service, Length: 315, dtype: int64
```

6. **Feature Selection**: The script creates a list of feature columns by subtracting the "Service" column from the list of all columns in newdata.

```
column_list=list(newdata.columns)
feature=list(set(column_list)-set(['Service']))
feature
```

```
['NormMileage', 'HVACwear', 'TyreWear', 'OilQual', 'EnginePerf']
```

7. **Train-Test Split:** The script splits the data into training and testing sets using train_test_split(x, y, test_size=0.2, random_state=0), where x represents the feature matrix and y represents the target variable.
8. **Logistic Regression Model:** The script creates a Logistic Regression model using LogisticRegression(), fits the model to the training data, and prints the coefficients and intercept of the model.

```
y=newdata['Service'].values
x=newdata[feature].values
trainx,testx,trainy,testy=train_test_split(x,y,test_size=0.2,random_state=0)
Logistic=LogisticRegression()
Logistic.fit(trainx,trainy)
Logistic.coef_
Logistic.intercept_
predict=Logistic.predict(testx)
print(predict)
```

```
[0 1 1 0 1 0 0 0 0 1 0 1 0 1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 1 0
 1 0 0 1 0 0 1 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 1 0 0 0]
```

9. Model Evaluation: The script uses the trained model to make predictions on the test set and prints the confusion matrix using confusion_matrix(testy, predict).

```
confusionm=confusion_matrix(testy,predict)
print(confusionm)
```

```
[[44  3]
 [ 1 15]]
```

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.

Implementation Steps:

1. **Importing Libraries**: Importing necessary libraries like pandas, numpy, seaborn, sklearn for data manipulation, visualization, and machine learning.

```
[1] import pandas as pd
    import numpy as np
    import seaborn as sns
    from sklearn.model_selection  import train_test_split
    from sklearn.linear_model import LogisticRegression
```

```
[2] from sklearn.metrics import accuracy_score,confusion_matrix
```

2. **Reading Data:** Reading a CSV file (TrainData.csv) from Google Drive using pandas and making a copy of the DataFrame.

```
DF1=pd.read_csv("/content/drive/My Drive/TrainData.csv")
Data=DF1.copy()
Data.isnull().sum()  #idebtify the null values in data set
```

```
OilQual        0
EnginePerf     0
NormMileage    0
TyreWear       0
HVACwear       0
Service        0
dtype: int64
```

3. **Identifying Missing Values:** Checking for missing values in the DataFrame (Data) using the isnull() method and then summing the missing values to identify the count of missing values in each column.

```
Data.isnull().sum()
missing=Data[Data.isnull().any(axis=1)]
missing
```

	OilQual	EnginePerf	NormMileage	TyreWear	HVACwear	Service

4. **Handling Missing Values**: Dropping rows with missing values using the dropna() method and creating a new DataFrame (data2) without missing values.

5. Data Exploration: Describing the data (data2) using the describe() method to get statistical information about the numerical columns and then checking for missing values again to confirm that there are no missing values in the new DataFrame.

```
[34] data2=Data.dropna(axis=0)
     data2.describe()
     data2.isnull().sum()
```

```
OilQual        0
EnginePerf     0
NormMileage    0
TyreWear       0
HVACwear       0
Service        0
dtype: int64
```

6. **Data Transformation:** Mapping categorical values ('Yes', 'No') in the 'Service' column to numerical values (1, 0) using the map() method. Using one-hot encoding (get_dummies()) to convert categorical variables into numerical format for machine learning.

```
data2['Service']=data2['Service'].map({'Yes':1,'No':0})
print(data2['Service'])
#data2.describe(include='O')
```

```
0      0
1      1
2      1
3      0
4      0
      ..
310    0
311    0
312    1
313    1
314    1
Name: Service, Length: 315, dtype: int64
```

7. **Splitting Data:** Splitting the data into training and testing sets using train_test_split() method from sklearn. This step divides the data into 80% training and 20% testing sets.
8. **Logistic Regression:** Creating a Logistic Regression model, fitting the model to the training data, and then making predictions on the test data using LogisticRegression() from sklearn.
9. **Model Evaluation:** Calculating the confusion matrix and accuracy score to evaluate the performance of the Logistic Regression model on the test data.

```
y=newdata['Service'].values
x=newdata[feature].values
trainx,testx,trainy,testy=train_test_split(x,y,test_size=0.2,random_state=0)
Logistic=LogisticRegression()
Logistic.fit(trainx,trainy)
Logistic.coef_
Logistic.intercept_
predict=Logistic.predict(testx)
print(predict)
```

```
[0 1 1 0 1 0 0 0 0 1 0 1 0 1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 1 0
 1 0 0 1 0 0 1 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 1 0 0 0]
```

```
confusionm=confusion_matrix(testy,predict)
print(confusionm)
```

```
[[44  3]
 [ 1 15]]
```

10. **K-Nearest Neighbors (KNN):** Creating a KNN classifier with k=3 neighbors, fitting the model to the training data, and then making predictions on the test data using KNeighborsClassifier() from sklearn.
11. **Model Evaluation (KNN):** Calculating the confusion matrix to evaluate the performance of the KNN classifier on the test data.

```
[24] accuracyscore=accuracy_score(testy,predict)
     print(accuracyscore)
     print((testy!=predict).sum())
```

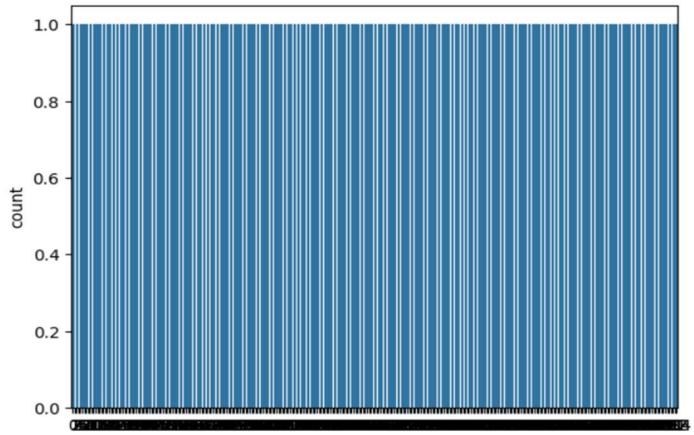
```
0.9365079365079365
4
```

```
from sklearn.neighbors import KNeighborsClassifier
import matplotlib.pyplot as plt
knn_classifier=KNeighborsClassifier(n_neighbors=3)
knn_classifier.fit(trainx, trainy)
kpredict=knn_classifier.predict(testx)
kpredict
confusionm=confusion_matrix(testy,kpredict)
print(confusionm)
```

```
[[47  0]
 [ 0 16]]
```

12. **Visualization:** Using seaborn's countplot() method to visualize the distribution of 'TyreWear' in the dataset.

```
plot1=sns.countplot(data2['TyreWear'])
#plot2=sns.distplot(data2['TyreWear'],bins=10, kde='False')
```



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.

Implementation Steps:

Data Preparation: Load and preprocess the MNIST dataset
Import necessary libraries including TensorFlow and load the MNIST dataset.
Normalize pixel values of the images to be between 0 and 1.

```
[1]  import numpy as np
     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
     from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
     import matplotlib.pyplot as plt
```

```
[ ]  # Load MNIST dataset
     (x_train, y_train), (x_test, y_test) = mnist.load_data()

     # Normalize pixel values to range [0, 1]
     x_train = x_train / 255.0
     x_test = x_test / 255.0
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 0s 0us/step
```

Model Architecture:
Design the ANN architecture with input, hidden, and output layers
Flatten the input images from 28x28 pixels to a 1D array of 784 values.
Define the ANN model with specified hidden layers and activation functions.\

```
     # Flatten the images
     x_train_flat = x_train.reshape(x_train.shape[0], -1)
     x_test_flat = x_test.reshape(x_test.shape[0], -1)
```

```
[ ]  # Convert labels to one-hot encoding
     y_train = tf.keras.utils.to_categorical(y_train, 10)
     y_test = tf.keras.utils.to_categorical(y_test, 10)
```

```
[ ]  # Define the ANN model
     model = Sequential([
         Dense(512, activation='relu', input_shape=(784,)),
         BatchNormalization(),  # Batch normalization layer
         Dropout(0.2),          # Dropout layer to prevent overfitting
         Dense(256, activation='relu'),
         BatchNormalization(),
         Dropout(0.2),
         Dense(128, activation='relu'),
         BatchNormalization(),
         Dropout(0.2),
         Dense(10, activation='softmax')
     ])
```

Batch Normalization:
Add batch normalization layers after the activation function in each hidden layer
Add BatchNormalization() layers after Dense layers to normalize the activations.

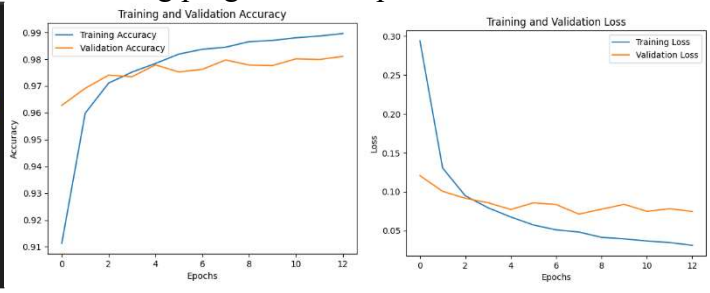
Dropout: Add dropout layers after the activation function in each hidden layer
Add Dropout() layers after BatchNormalization() to prevent overfitting by randomly dropping a fraction of input units during training.

Training: Train the model using the Backpropagation algorithm with early stopping
Compile the model specifying optimizer, loss function, and metrics.
Use EarlyStopping callback to monitor validation loss and stop training if no improvement after a certain number of epochs.

```
[ ]  # Evaluate the model
     test_loss, test_accuracy = model.evaluate(x_test_flat, y_test)
     print(f'Test Accuracy: {test_accuracy}')

     313/313 [==============================] - 1s 3ms/step - loss: 0.0717 - accuracy: 0.9810
     Test Accuracy: 0.9810000061988831
```

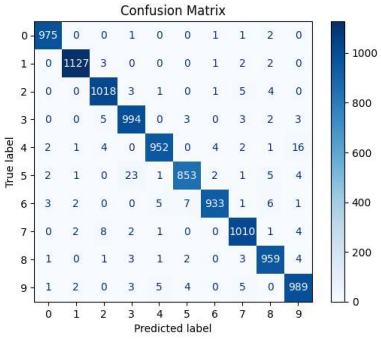
Plotting Training History
Plot training and validation loss to visualize the model's learning progress over epochs.

```
# Plot training and validation loss
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Training and Validation Loss')
plt.legend()
plt.show()
```



Confusion Matrix
Generate a confusion matrix to evaluate the model's performance on the test set

```
# Confusion matrix
y_pred = np.argmax(model.predict(x_test_flat), axis=-1)
cm = confusion_matrix(np.argmax(y_test, axis=-1), y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=np.arange(10))
disp.plot(cmap=plt.cm.Blues)
plt.title('Confusion Matrix')
plt.show()
```



Conclusion
The implementation demonstrated training a robust neural network on the MNIST dataset using TensorFlow and Keras. The process involved loading and preprocessing the data, designing an ANN with batch normalization and dropout for regularization, and training the model with early stopping to optimize performance. Evaluation showed a high test accuracy of approximately 98.1%. Visualizations such as training history plots and a confusion matrix provided insights into model performance.

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.

Implementation Steps:

Importing Necessary Libraries:

Import required libraries including TensorFlow, Keras, and relevant modules for dataset loading, model building, and evaluation.

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.datasets import cifar100
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Dropout, BatchNormalization
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt
```

Loading CIFAR-100 Dataset: Load the CIFAR-100 dataset using TensorFlow's dataset loader.
Normalize pixel values to be within the range [0, 1].
Label Encoding: Convert labels (y_train and y_test) to one-hot encoded format suitable for multi-class classification.

```
[2] # Load CIFAR-100 dataset
    (x_train, y_train), (x_test, y_test) = cifar100.load_data()

    # Normalize pixel values to range [0, 1]
    x_train = x_train / 255.0
    x_test = x_test / 255.0

    # Convert labels to one-hot encoding
    y_train = tf.keras.utils.to_categorical(y_train, 100)
    y_test = tf.keras.utils.to_categorical(y_test, 100)

    Downloading data from https://www.cs.toronto.edu/~kriz/cifar-100-python.tar.gz
    169001437/169001437 [==============================] - 6s 0us/step
```

Defining the Neural Network Model: Design the architecture of the neural network model using Keras Sequential API.

```
# Define the ANN model
model = Sequential([
    Flatten(input_shape=(32, 32, 3)),
    Dense(512, activation='relu'),
    BatchNormalization(),  # Batch normalization layer
    Dropout(0.2),          # Dropout layer to prevent overfitting
    Dense(256, activation='relu'),
    BatchNormalization(),
    Dropout(0.2),
    Dense(128, activation='relu'),
    BatchNormalization(),
    Dropout(0.2),
    Dense(100, activation='softmax')
])
```

Compiling the Model:

Compile the model specifying optimizer, loss function, and metrics for training.

```
[4]  # Compile the model
     model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

Early Stopping:
Define an early stopping callback to monitor validation loss and stop training if no improvement is observed.

```
✓ 0s  [5]  # Early stopping callback
           early_stopping = EarlyStopping(monitor='val_loss', patience=5)
```

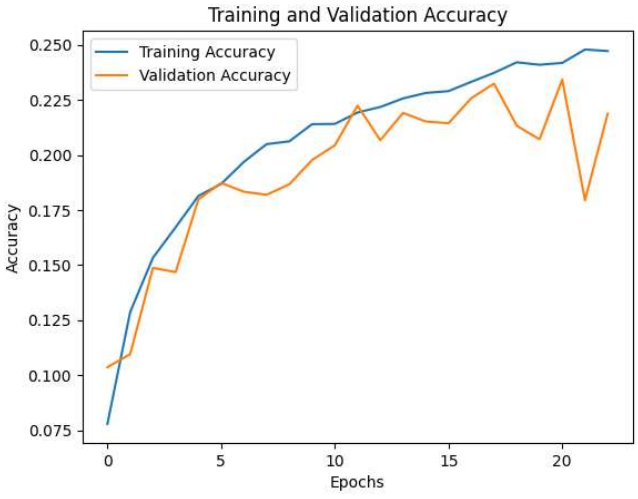
Training the Model:
Train the compiled model using the training data.

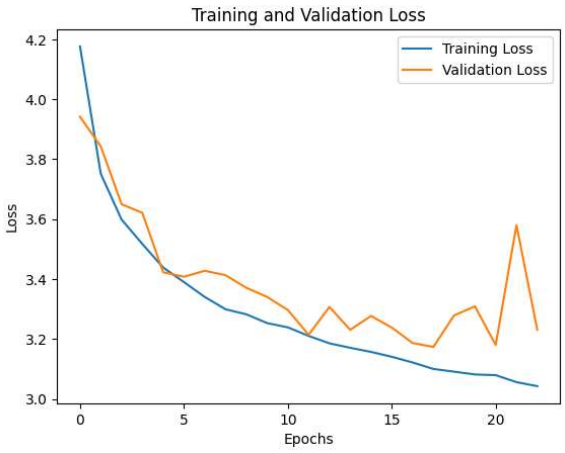
```
# Plot training and validation loss
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Training and Validation Loss')
plt.legend()
plt.show()
```

Model Evaluation:
Evaluate the trained model on the test set to measure its performance.

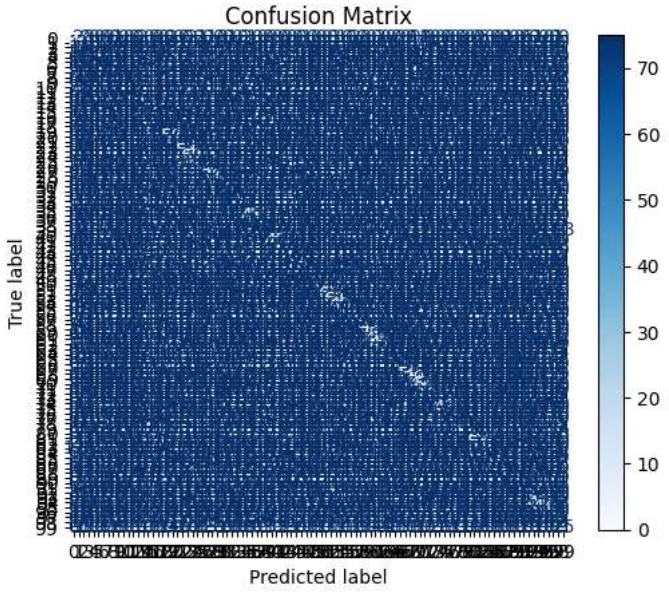
```
✓ 21s  [7]  # Evaluate the model
            test_loss, test_accuracy = model.evaluate(x_test, y_test)
            print(f'Test Accuracy: {test_accuracy}')

       313/313 [==============================] - 3s 9ms/step - loss: 3.2284 - accuracy: 0.2268
       Test Accuracy: 0.22679999470710754
```

Plotting Training History:
Plot training and validation loss to visualize the model's learning progress.





Confusion Matrix:
Generate a confusion matrix to evaluate the model's performance on the test set.




```
# Confusion matrix
y_pred = np.argmax(model.predict(x_test), axis=-1)
cm = confusion_matrix(np.argmax(y_test, axis=-1), y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=np.arange(100))
disp.plot(cmap=plt.cm.Blues)
plt.title('Confusion Matrix')
plt.show()
```

Conclusion:

This implementation demonstrates a complete workflow for training, evaluating, and analyzing a neural network model for image classification on the CIFAR-100 dataset using TensorFlow and Keras, including data loading, preprocessing, model definition, training, evaluation, and result visualization. The code also incorporates techniques like batch normalization, dropout regularization, early stopping, and model performance visualization using plots and confusion matrix.

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])
```

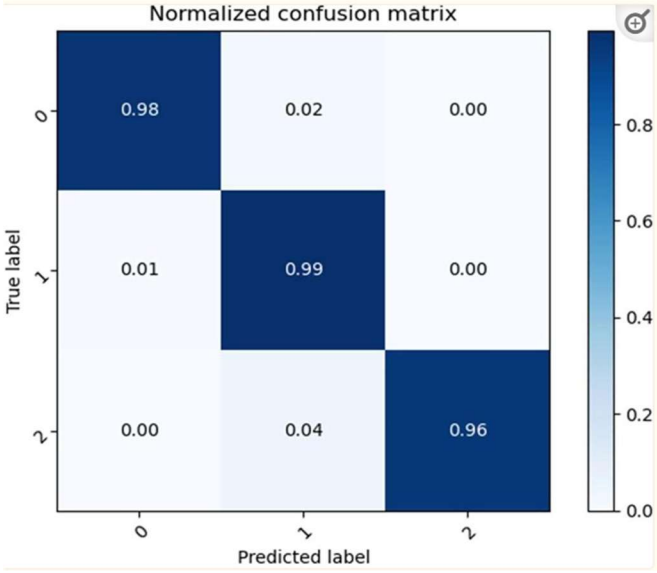
In this example, we're creating an early stopping callback that stops training when the validation loss hasn't improved for 3 epochs. We then compile the model and train it with the early stopping callback.

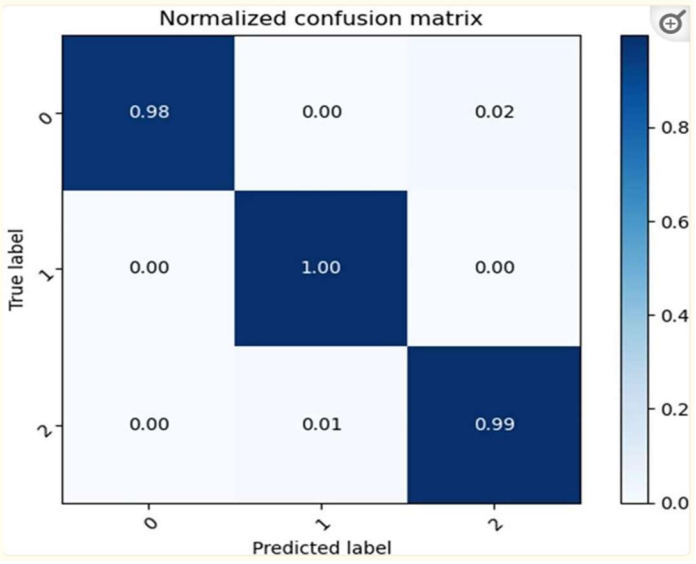
➔ **Dropout**

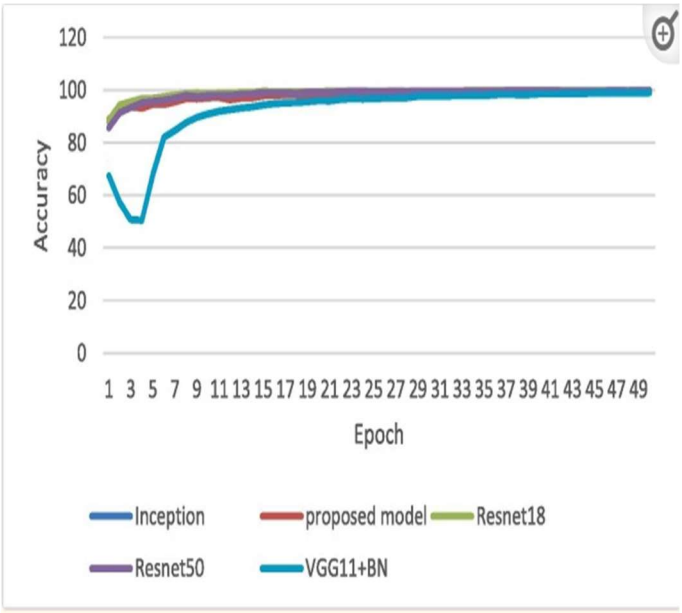
Dropout is a technique that randomly drops out neurons during training, which can help prevent overfitting. Here's an example of how you might implement dropout in Keras:

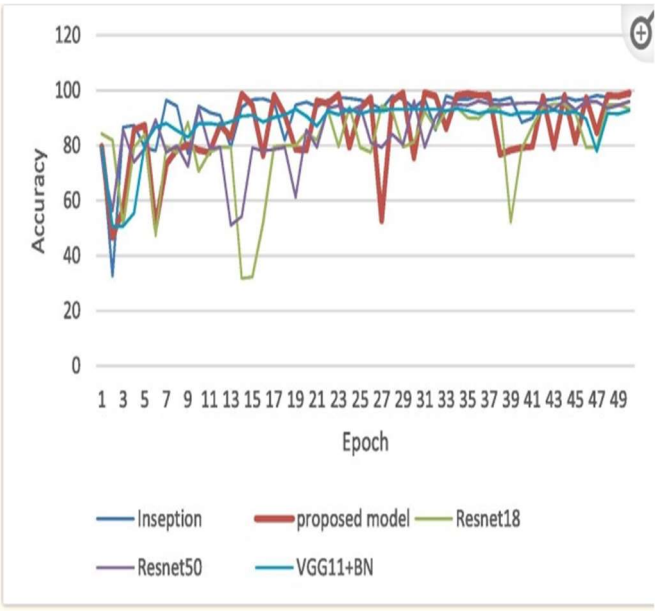
```
from keras.layers import Dropout

# 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))
```

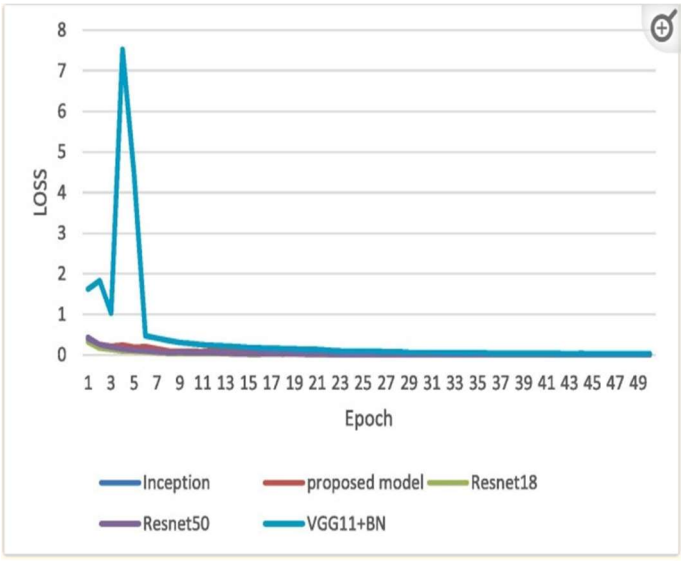
In this example, we're adding a dropout layer after each convolutional layer. The dropout rate is set to 0.25 for the first two layers and 0.5 for the last layer. This means that 25% and 50% of the neurons will be randomly dropped out during training, respectively.By combining these techniques, you can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

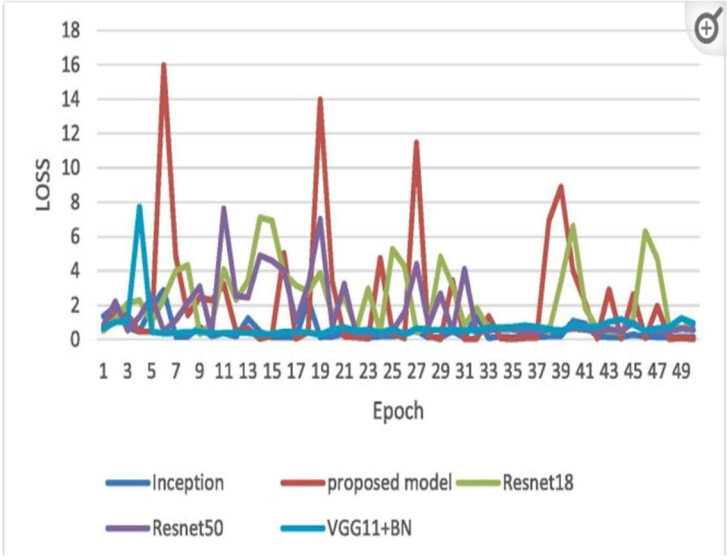





Accuracy
Epoch
Inception
proposed model
Resnet18
Resnet50
VGG11+BN


Accuracy
Epoch
Inception
proposed model
Resnet18
Resnet50
VGG11+BN


LOSS
Epoch
Inception
proposed model
Resnet18
Resnet50
VGG11+BN


LOSS
Epoch
Inception
proposed model
Resnet18
Resnet50
VGG11+BN

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.

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.

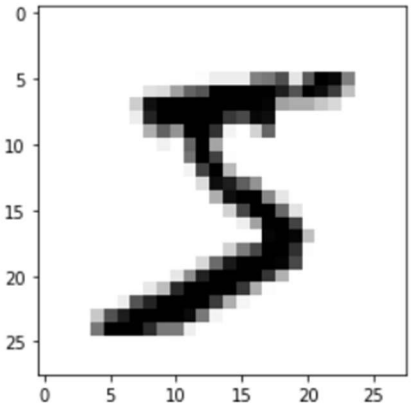
The MNIST dataset is a popular choice for training machine learning models because it is relatively small and easy to work with, while still being large enough to train effective models. Additionally, the MNIST dataset is well-labeled, which makes it easy to evaluate the performance of machine learning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

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.

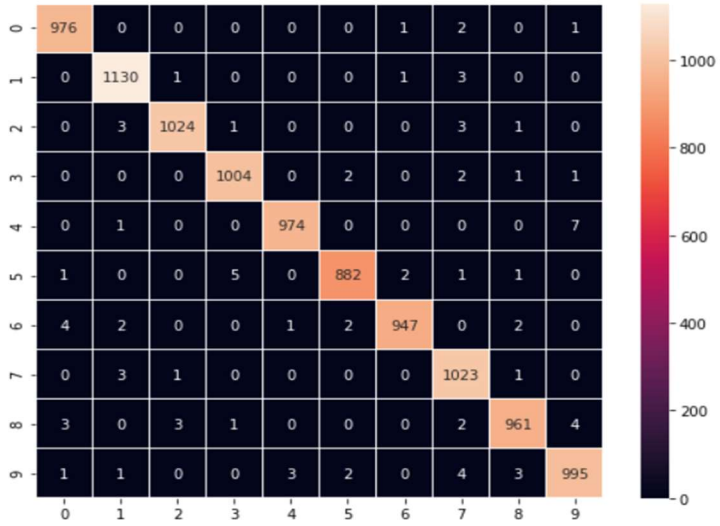
RESULTS/GRAPHS:

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

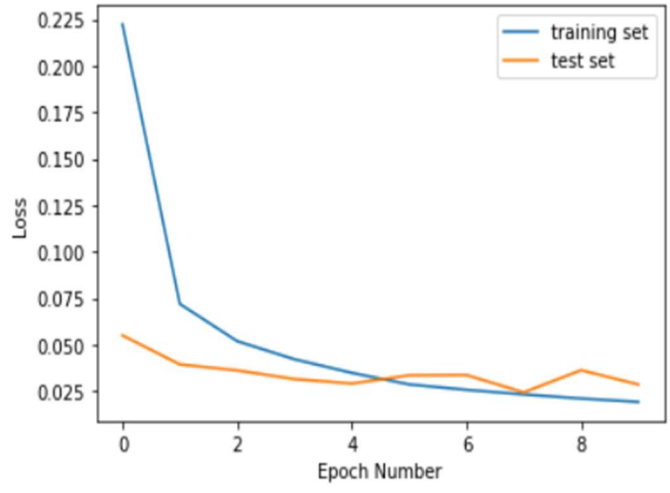


```
model.summary()
```

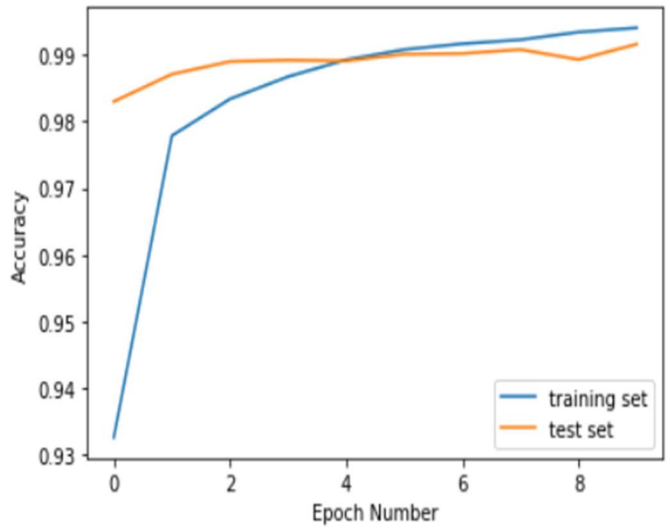
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 24, 24, 8)         208
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 12, 12, 8)         0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 8, 8, 16)          3216
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (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>



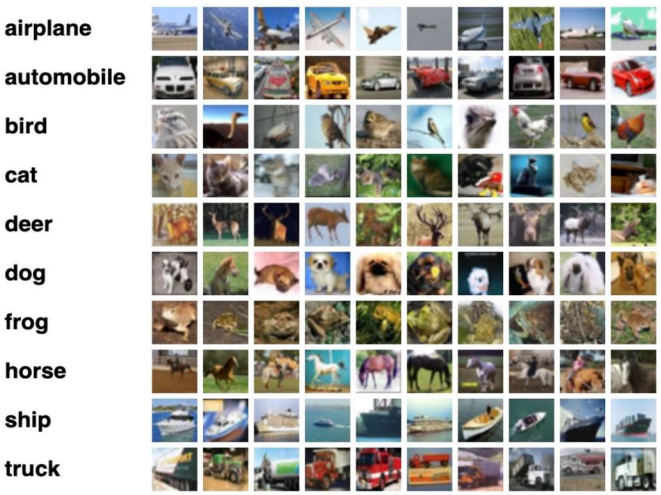
<matplotlib.legend.Legend at 0x1551c77d0>



Experiment- 13

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

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 **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.

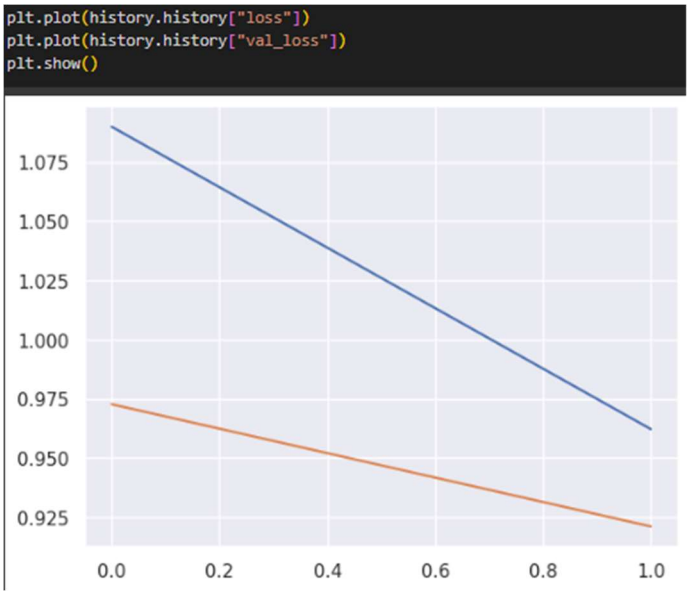
Visualizing Model Performance:

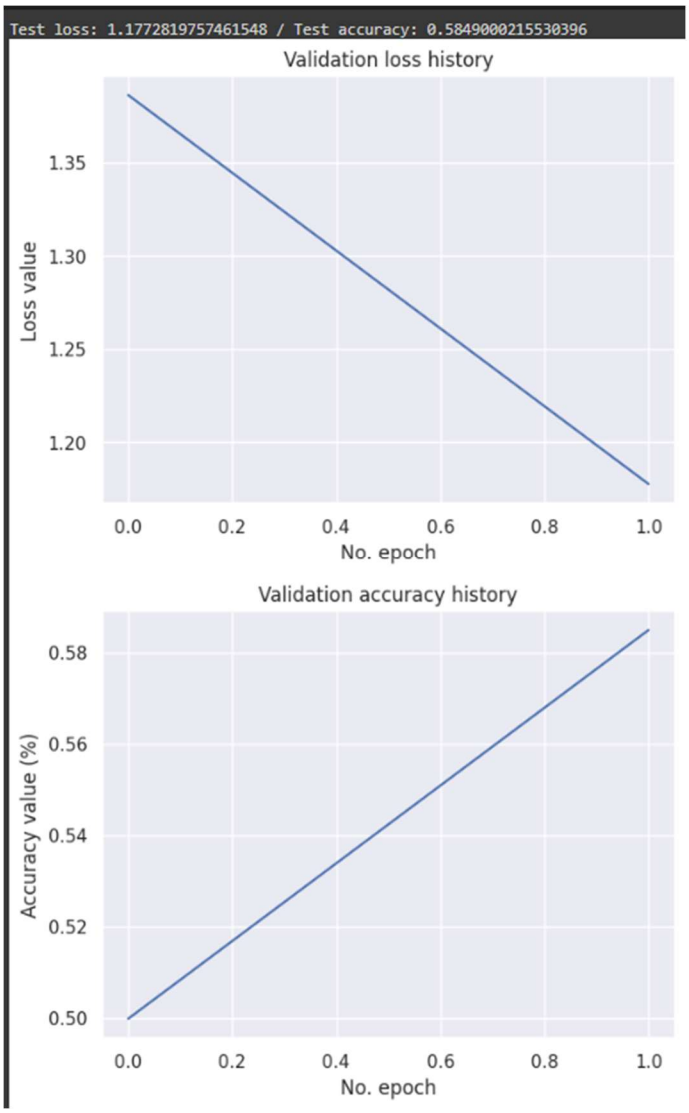
The script plots the validation loss and validation accuracy history using Matplotlib to visualize the model's performance during training.

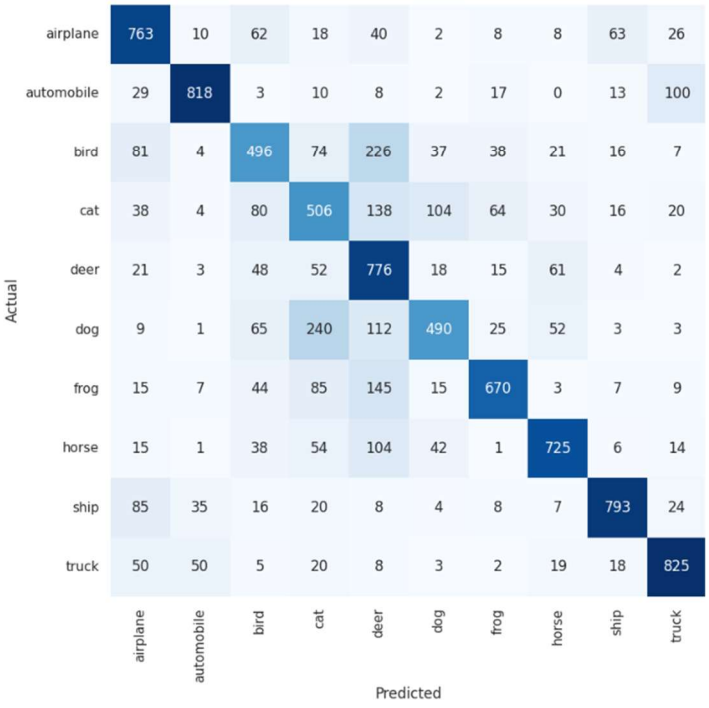
Visualizing Predictions:

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

RESULTS/GRAPHS:







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.t
35363/35363 [==============================] - 0s 0us/step
tiger_cat (30.23%)
```

Total Parameter:

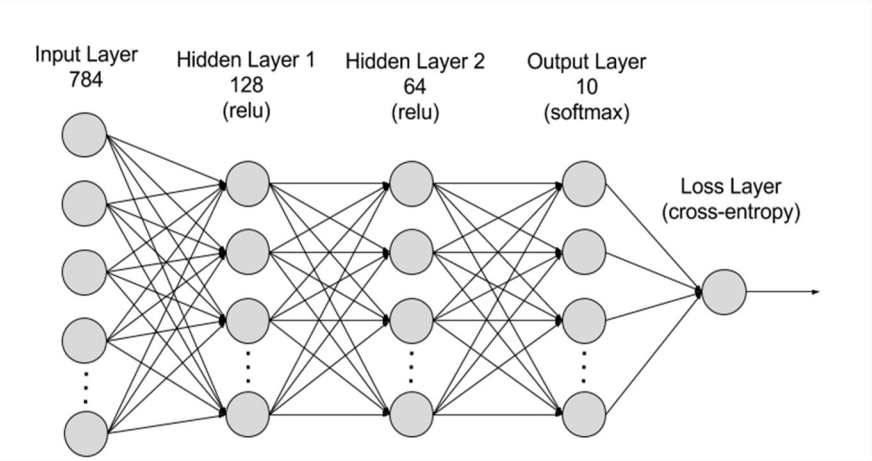
```
=================================================================
Total params: 138357544 (527.79 MB)
Trainable params: 138357544 (527.79 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

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.

Types Of RNN

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

- **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.

RESULTS:

```
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_2 (LSTM)               (None, 28, 128)           80384

 dropout_3 (Dropout)         (None, 28, 128)           0

 lstm_3 (LSTM)               (None, 128)               131584

 dropout_4 (Dropout)         (None, 128)               0

 dense_2 (Dense)             (None, 32)                4128

 dropout_5 (Dropout)         (None, 32)                0

 dense_3 (Dense)             (None, 10)                330

=================================================================
Total params: 216426 (845.41 KB)
Trainable params: 216426 (845.41 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
opt = tf.keras.optimizers.legacy.Adam(learning_rate=0.001, decay=1e-6)

# Compile model
model.compile(
    loss='sparse_categorical_crossentropy',
    optimizer=opt,
    metrics=['accuracy'],
)
```

```
model.fit(x_train,y_train,
          epochs=3,verbose=1)
```

```
Epoch 1/3
1875/1875 [==============================] - 158s 83ms/step - loss: 0.6443 - accuracy: 0.7915
Epoch 2/3
1875/1875 [==============================] - 155s 83ms/step - loss: 0.1642 - accuracy: 0.9554
Epoch 3/3
1875/1875 [==============================] - 156s 83ms/step - loss: 0.1126 - accuracy: 0.9692
<keras.src.callbacks.History at 0x7e0ed81f50c0>
```