

**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. Shilpa Bhalerao Nehal Jain

HOD, CSIT Dept. 0827CI211120

**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. Nehal Jain 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 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.

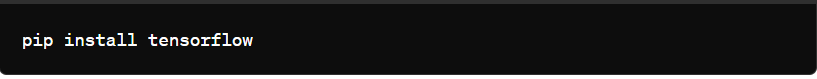
**Step 2: Create a New Environment:**

* + 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:

**Step 4 : Install Additional Packages:**

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

**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 notebookin 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**

**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.

**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 behaviour through methods.

**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.

**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."

**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**
   1. Try to use vectorized operations for data manipulation and extraction. This is often much faster than iterating through a DataFrame or Series.
   2. 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.
   3. Use the .describe() method to get summary statistics of numeric columns.
   4. Use the .isnull() method to check for missing values.
   5. Use the .groupby() method to aggregate and filter data.
2. **Numpy**
   1. Use boolean masks instead of explicit loops for vectorized operations.
   2. Use the .reshape() method to manipulate the shape of arrays.
   3. Use the .concatenate() method to combine multiple arrays.
   4. Use the .stack() method to convert a 2-dimensional array into a 1-dimensional array.
   5. 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++**](https://www.geeksforgeeks.org/cpp-tutorial/)**,** [**Java**](https://www.geeksforgeeks.org/java/)**,** [**JavaScript**](https://www.geeksforgeeks.org/javascript/)**,** [**Go**](https://www.geeksforgeeks.org/golang-tutorial-learn-go-programming-language/)**,** [**Swift**](https://www.geeksforgeeks.org/swift-tutorial/)**, 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](https://www.geeksforgeeks.org/matplotlib-tutorial/), [Pandas](https://www.geeksforgeeks.org/pandas-tutorial/), 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= *θ*0+*θ*1*x*1​+*θ*2*x*2​+...+*θn*​*xn***

**Where:**

* ***y* is the dependent variable.**
* ***x*1 , *x*2,...,*xn* are the independent variables.**
* **θ0,θ1,...,θnare 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:



Where:

* *m* is the number of training examples.
* *hθ*(*x*) is the hypothesis function, which predicts the value of *y* given *x*.
* **

### 

### 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:

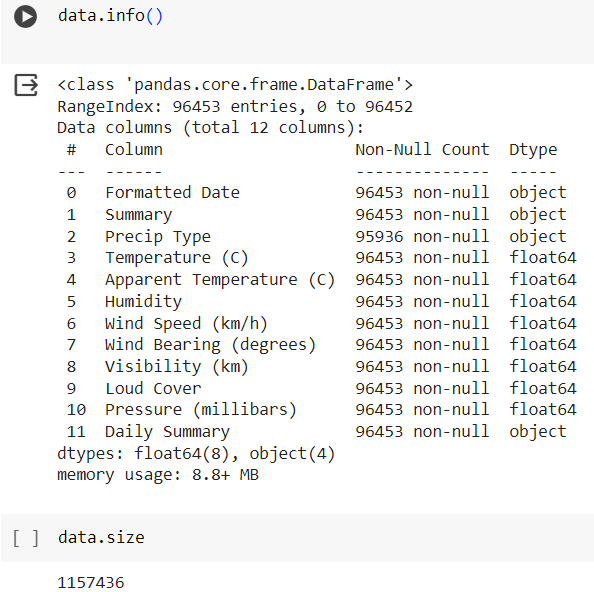


Where:

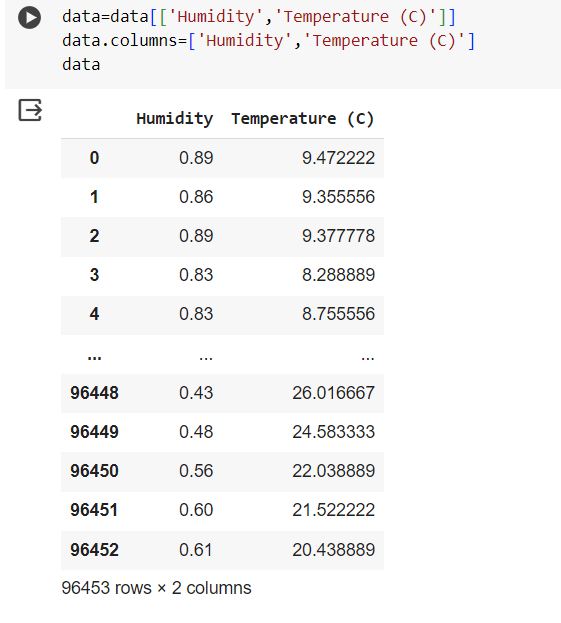
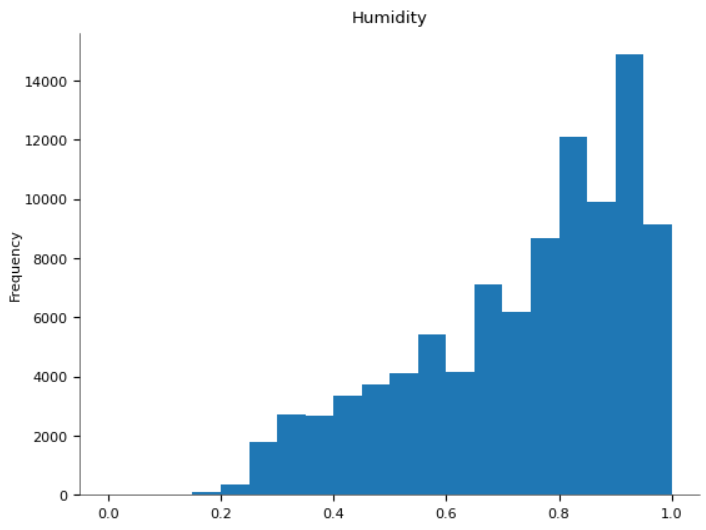
* *α* is the learning rate, determining the step size of each iteration.

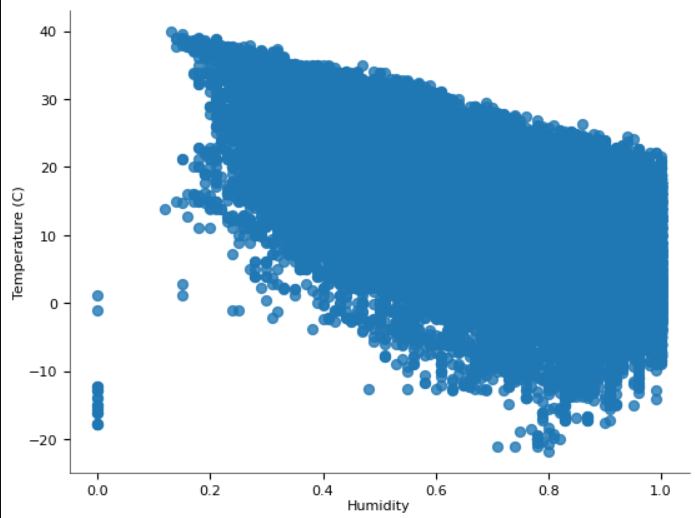
### Description:

### 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.

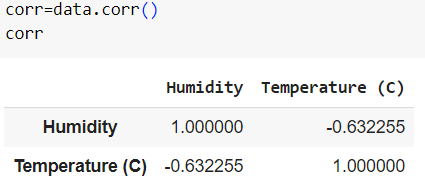


### Data Preprocessing: Check for any missing values or outliers in the data. If there are missing values, you may need to handle them by imputing or removing the respective rows. Outliers may also need to be treated appropriately depending on their impact on the model.



* 1. 

### 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.



### 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.

### 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.

### 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.



### 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.

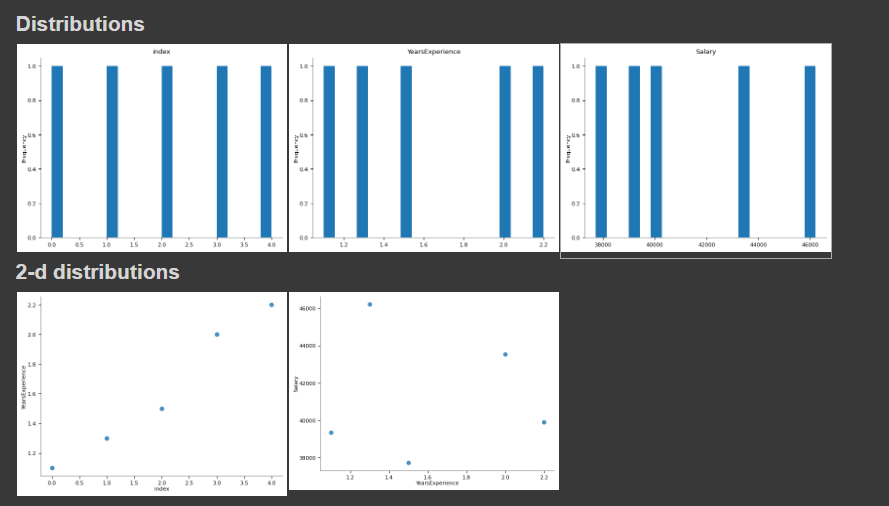
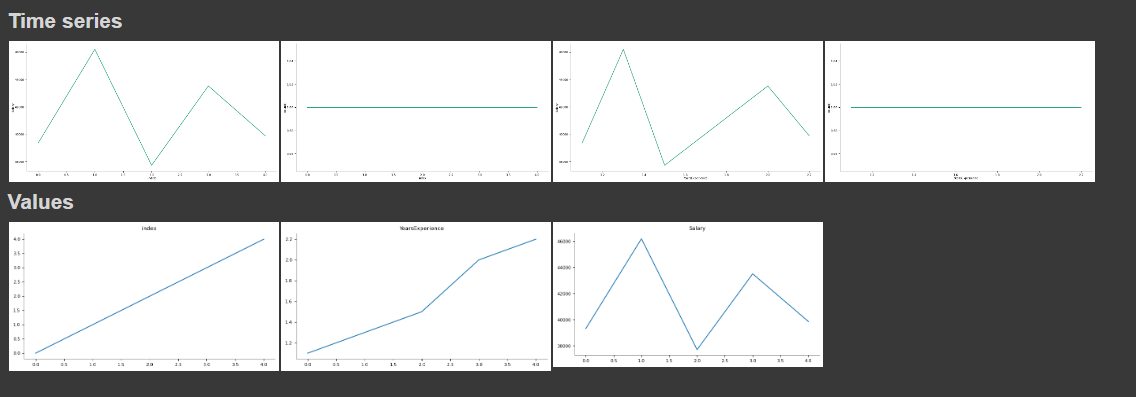
**Experiment- 7**

**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.**

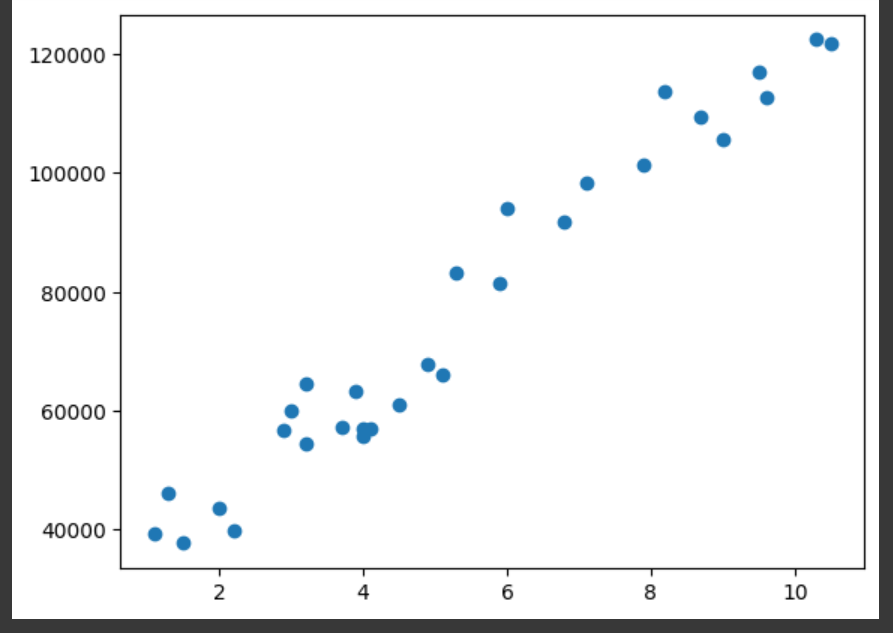


**Description:**

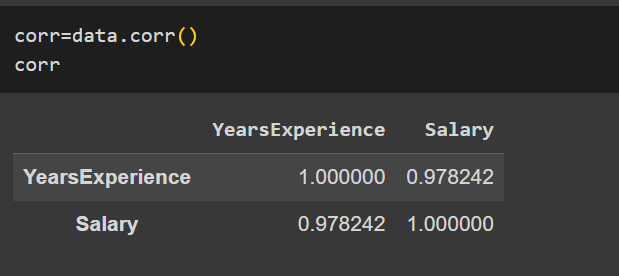
1. **Load the Data:** Read the salary.csv file to extract the training data. Each row represents a data example, with Years of experience and salary as features, and the target variable could be something like precipitation, temperature at a certain time, or any other-related metric.

****

### 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.

****

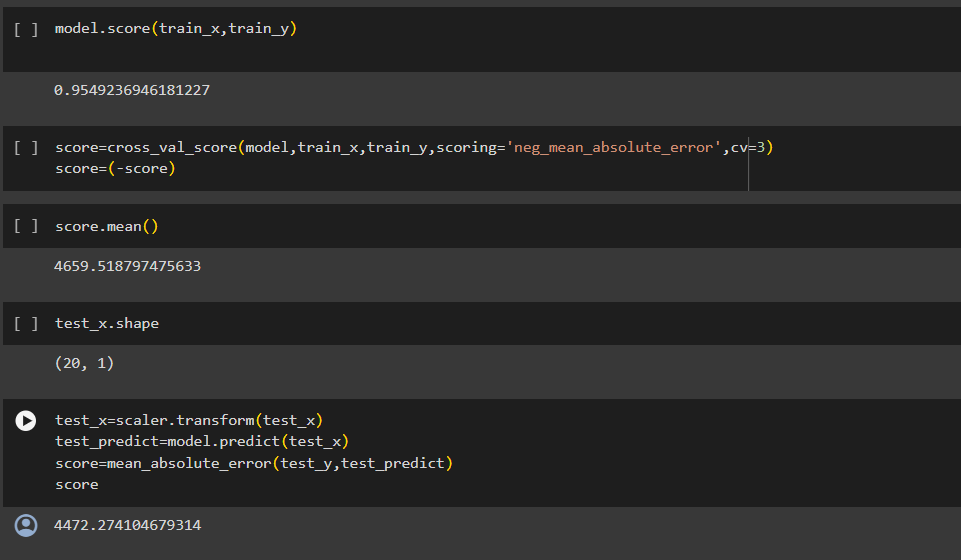
1. **Split the Data:** Divide the dataset into training and testing sets. The training set is used to train the model, while the testing set is used to evaluate its performance.

****

### 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.

### 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.

### 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.



**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:

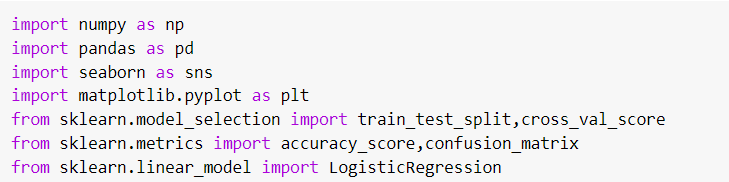
P(Y=1) = 1 / (1 + e^(-z))

where z = b0 + b1x1 + b2x2 + ... + bn\*xn

The coefficients b0, b1, b2, ..., bn are estimated using maximum likelihood estimation.

### Implementation Steps:

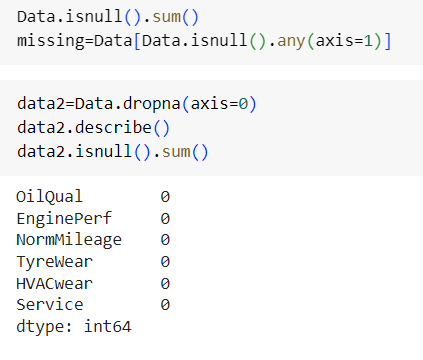
1. **Importing Libraries:** The script starts by importing the necessary libraries, including NumPy, Pandas, Seaborn, Matplotlib, and various modules from Scikit-learn.



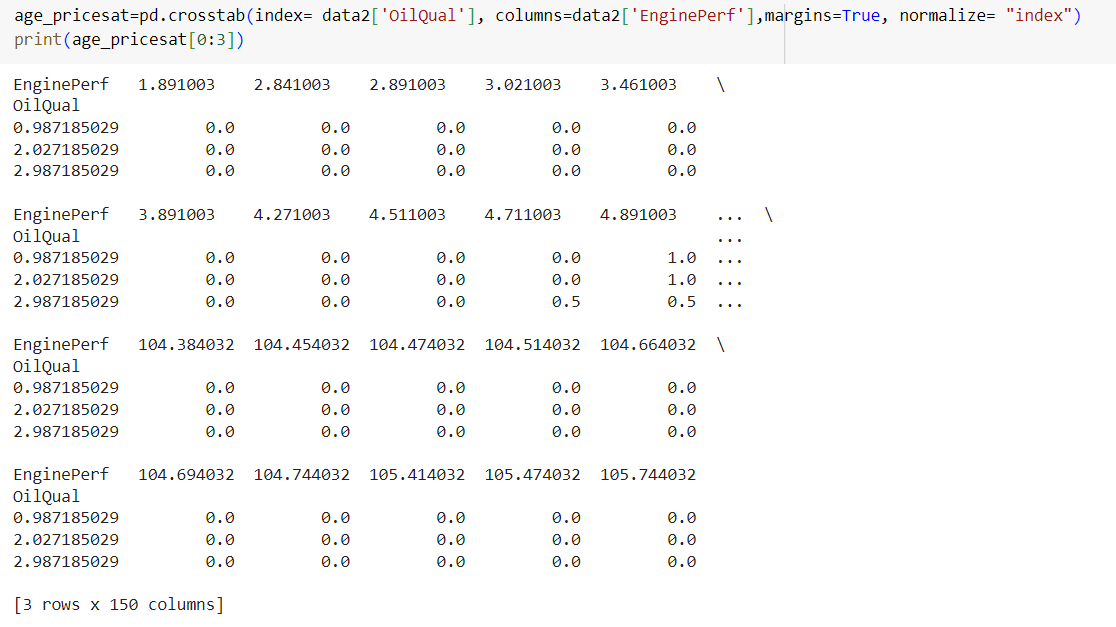
1. **Loading and Exploring the Dataset:** The script reads the "TrainData.csv" file from the user's Google Drive and stores it in the DF1 DataFrame. It then creates a copy of the dataset in the Data DataFrame and performs the following actions:Checks for missing values using



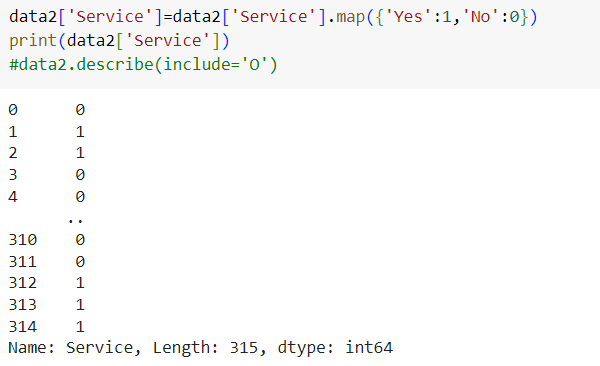
1. **Handling Missing Values:** The script identifies the rows with missing values using Data[Data.isnull().any(axis=1)] and stores them in the missing DataFrame. It then creates a new DataFrame data2 by dropping the rows with missing values using Data.dropna(axis=0).



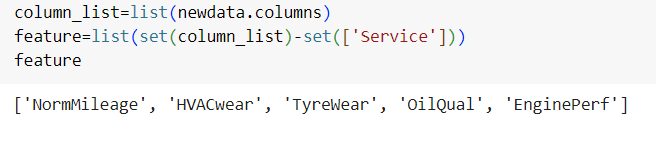
1. **Crosstabulation:** The script creates a crosstabulation of the "OilQual" and "EnginePerf" columns, normalizing the values by row using pd.crosstab(index=data2['OilQual'], columns=data2['EnginePerf'], margins=True, normalize="index").



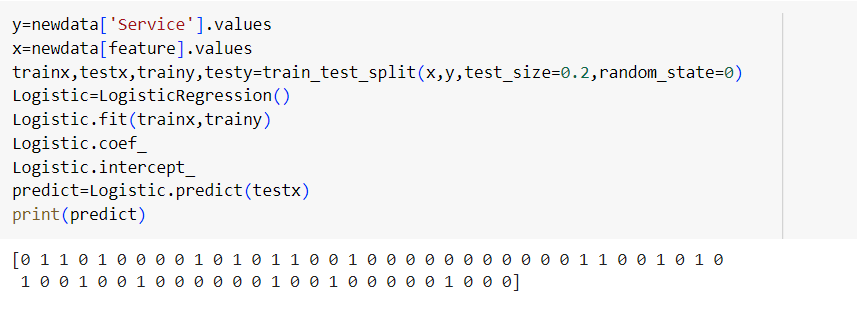
1. **Data Preprocessing:** The script maps the "Yes" and "No" values in the "Service" column to 1 and 0, respectively, using data2['Service']=data2['Service'].map({'Yes':1,'No':0}). It then creates a new DataFrame newdata by applying one-hot encoding to the categorical variables using pd.get\_dummies(data2,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



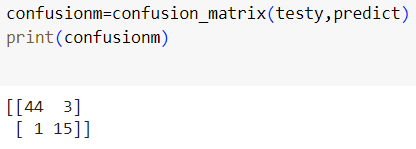
1. **Feature Selection**: The script creates a list of feature columns by subtracting the "Service" column from the list of all columns in newdata.



1. **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.
2. **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.



1. 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).

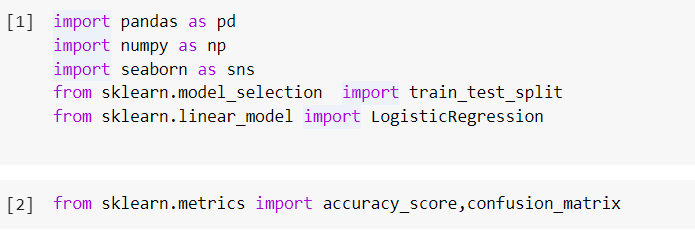


**KNN Theory:**

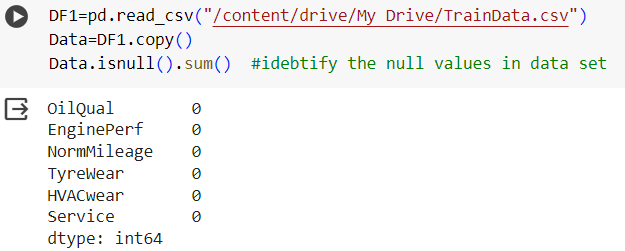
KNN (K-Nearest Neighbors) is a simple, instance-based learning algorithm used for classification and regression. The idea behind KNN is to find the 'k' nearest training samples to a test sample and classify the test sample based on the majority class of these 'k' nearest neighbors.

### Implementation Steps:

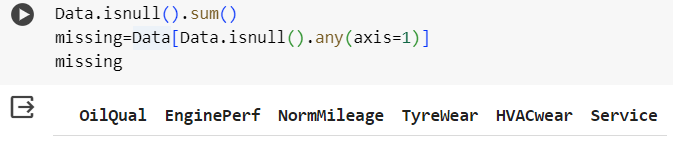
1. **Importing Libraries**: Importing necessary libraries like pandas, numpy, seaborn, sklearn for data manipulation, visualization, and machine learning.



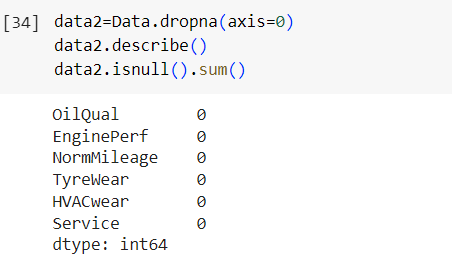
1. **Reading Data:** Reading a CSV file (TrainData.csv) from Google Drive using pandas and making a copy of the DataFrame.



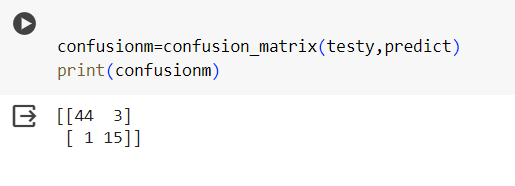
1. **Identifying Missing Values:** Checking for missing values in the DataFrame (Data) using the isnull() method and then summing the missing values to identify the count of missing values in each column.

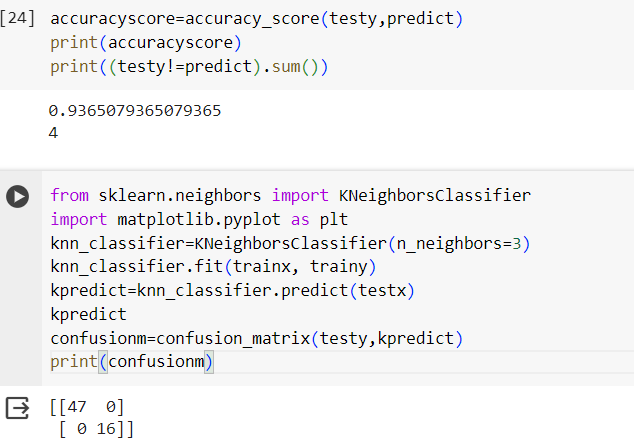


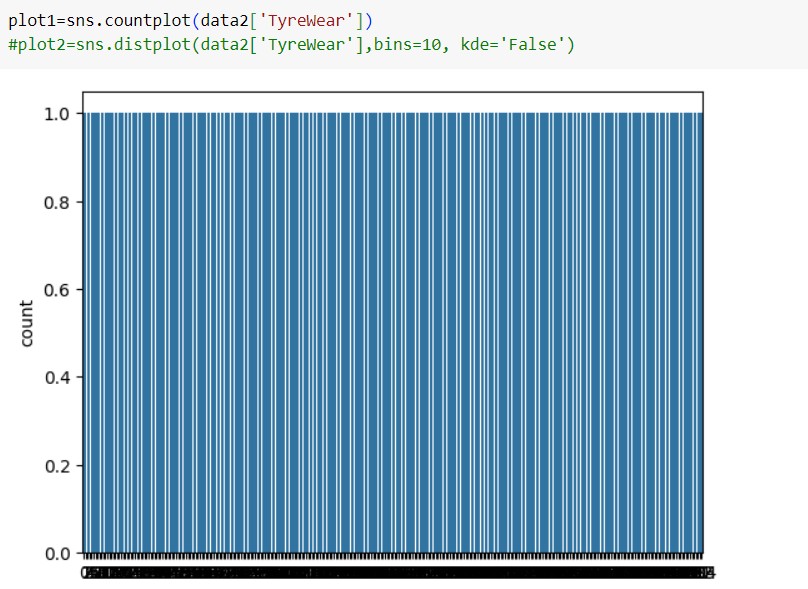
1. **Handling Missing Values**: Dropping rows with missing values using the dropna() method and creating a new DataFrame (data2) without missing values.



1. 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.
2. **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.
3. **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.
4. **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.
5. **Model Evaluation:** Calculating the confusion matrix and accuracy score to evaluate the performance of the Logistic Regression model on the test data.



1. **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.
2. **Model Evaluation (KNN):** Calculating the confusion matrix to evaluate the performance of the KNN classifier on the test data.
3. **Visualization:** Using seaborn's countplot() method to visualize the distribution of 'TyreWear' in the dataset.



**Experiment- 9**

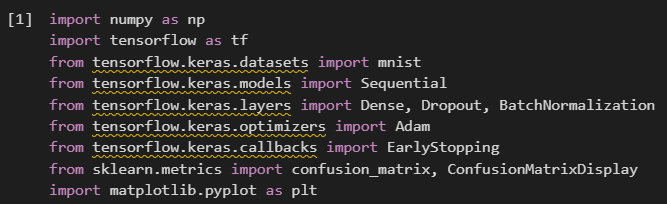
**Aim: Build an Artificial Neural Network (ANN) by implementing the Backpropagation algorithm and test the same using MNIST Handwritten Digit Multiclass classification data sets with use of batch normalization, early stopping and drop out.**

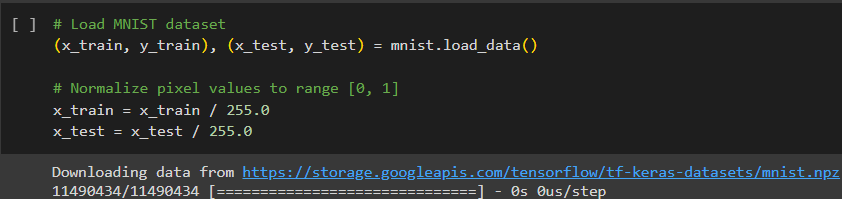
**Implementation Steps:**

**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.



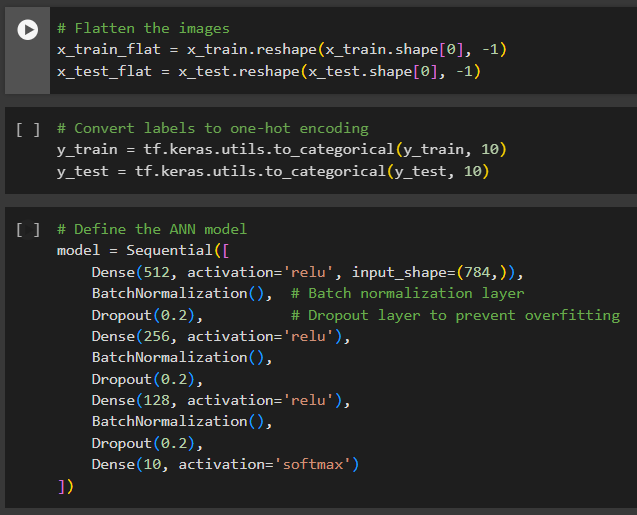


**Model Architecture:**

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.\



**Batch Normalization:**

Add batch normalization layers after the activation function in each hidden layer

Add BatchNormalization() layers after Dense layers to normalize the activations.

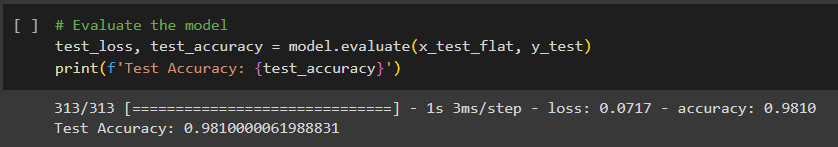
**Dropout:** Add dropout layers after the activation function in each hidden layer

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

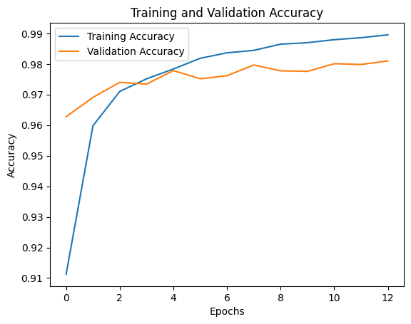
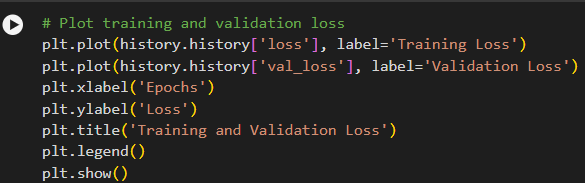
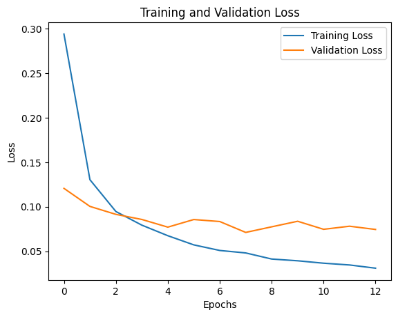
Compile the model specifying optimizer, loss function, and metrics.

Use EarlyStopping callback to monitor validation loss and stop training if no improvement after a certain number of epochs.



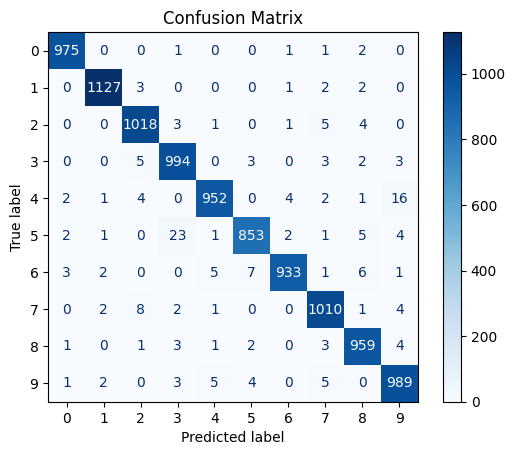
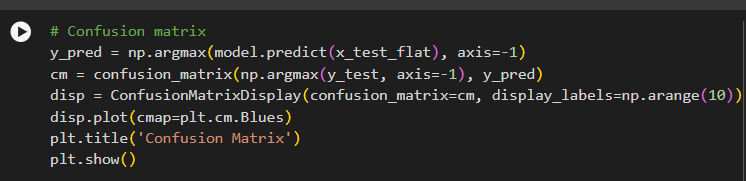
**Plotting Training History**

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

****

**Confusion Matrix**

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



**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.

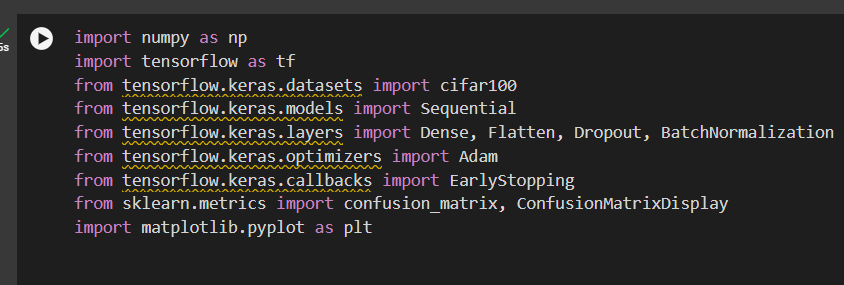
**Experiment- 10**

**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.



**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.



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

****

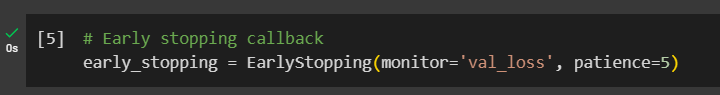
**Compiling the Model:**

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



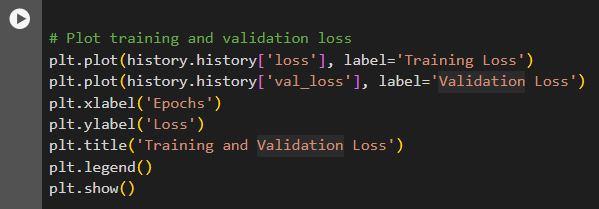
**Early Stopping:**

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



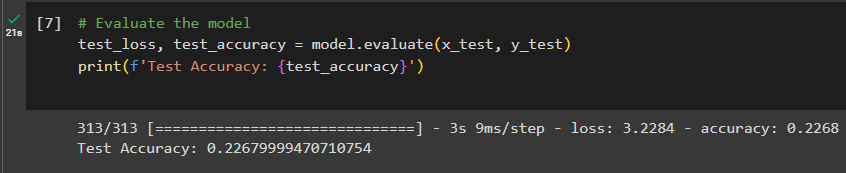
**Training the Model:**

Train the compiled model using the training data.



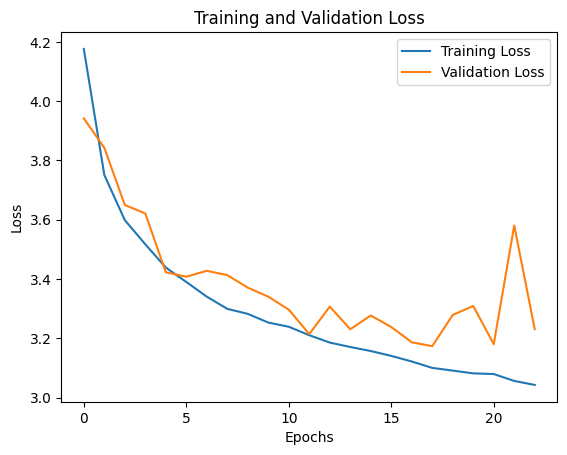
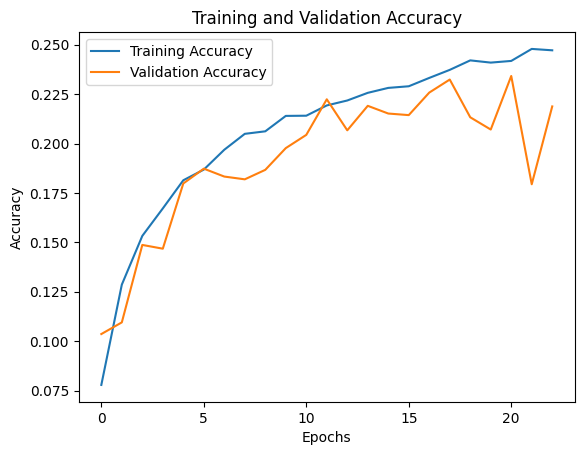
**Model Evaluation:**

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

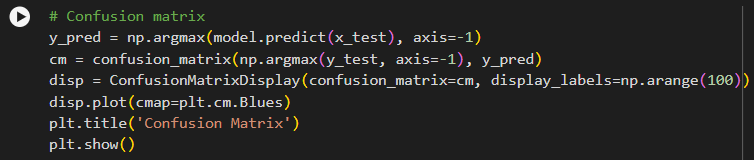
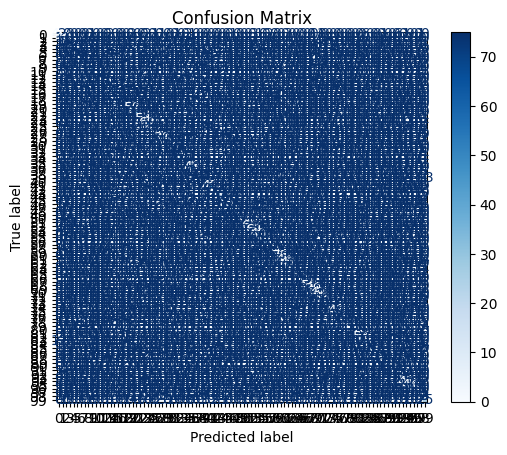


**Plotting Training History:**

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.

**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))**

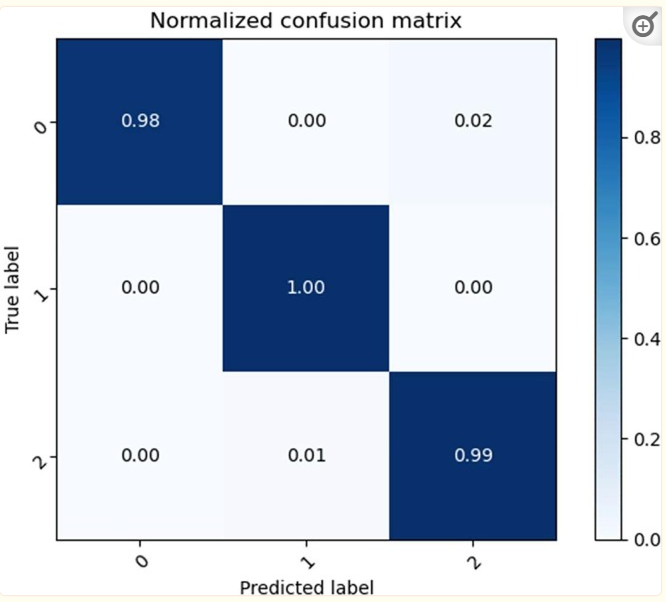
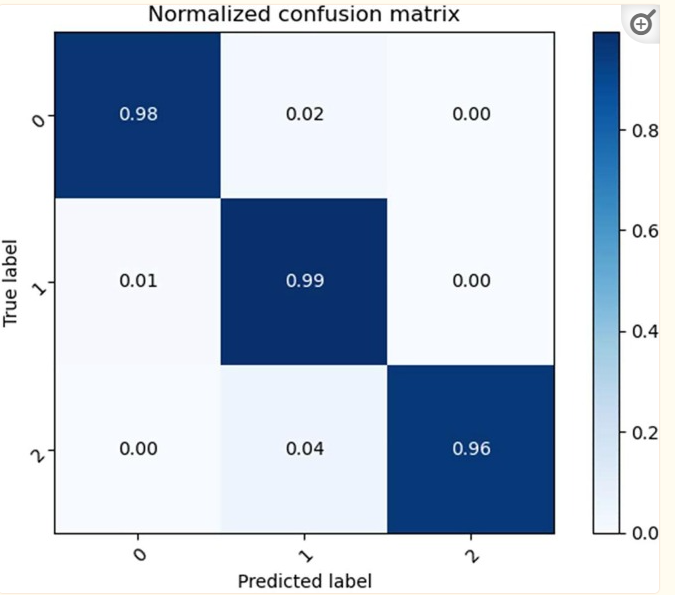
**model.add(Conv2D(64, (3, 3), activation='relu'))**

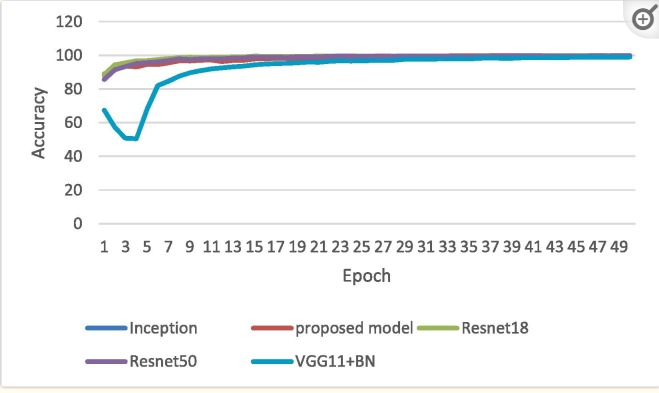
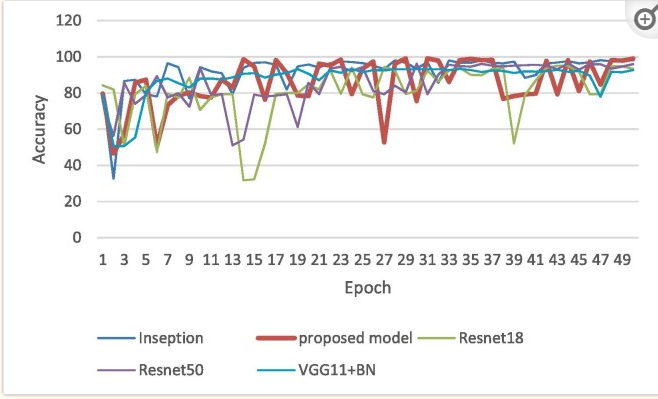
**model.add(BatchNormalization())**

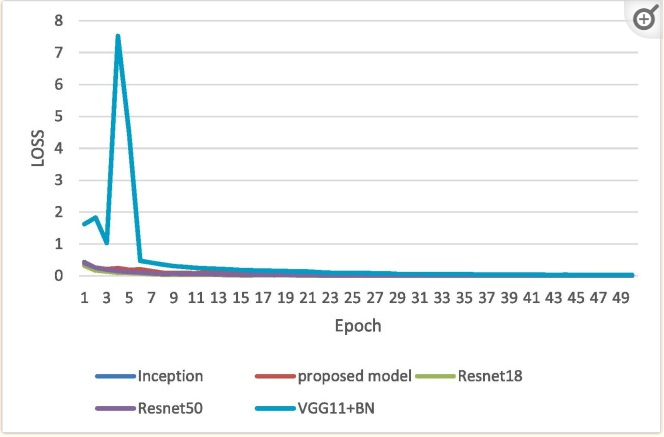
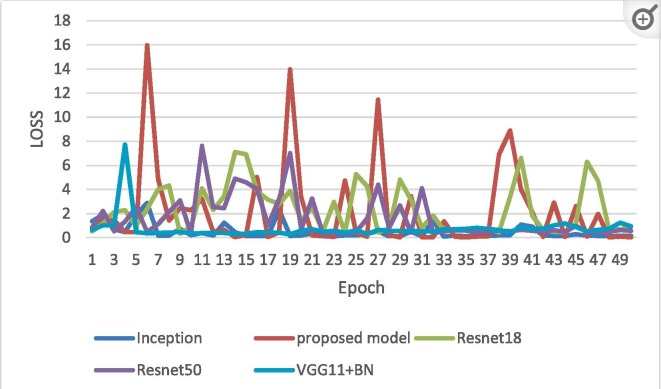
**model.add(Flatten())**

**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.



**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.

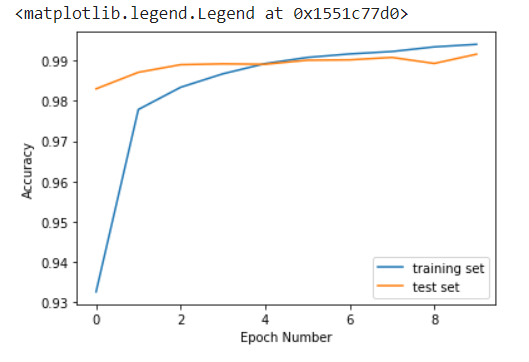
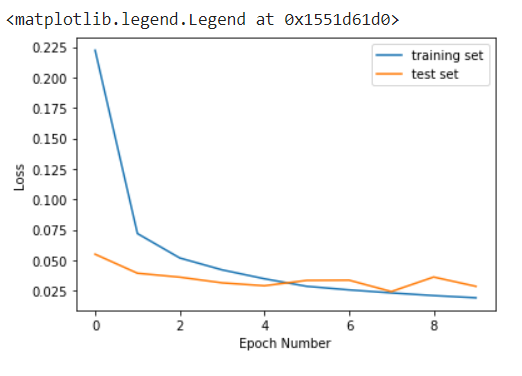
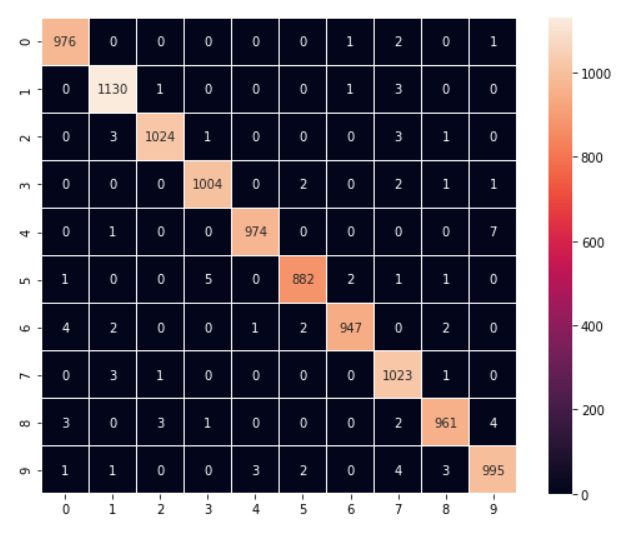
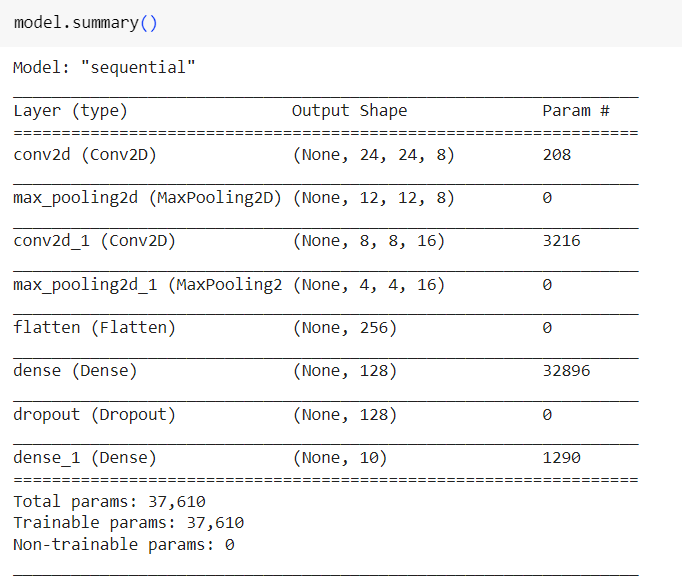
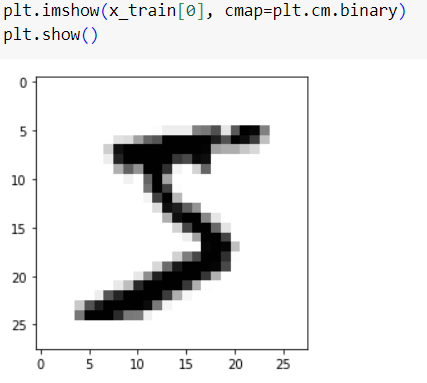
The **MNIST** dataset is a large database of handwritten digits that is commonly used for training various image processing systems. The dataset includes 60,000 training images and 10,000 test images. Each image is a 28x28 grayscale image of a single handwritten digit, from 0 to 9.

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:**

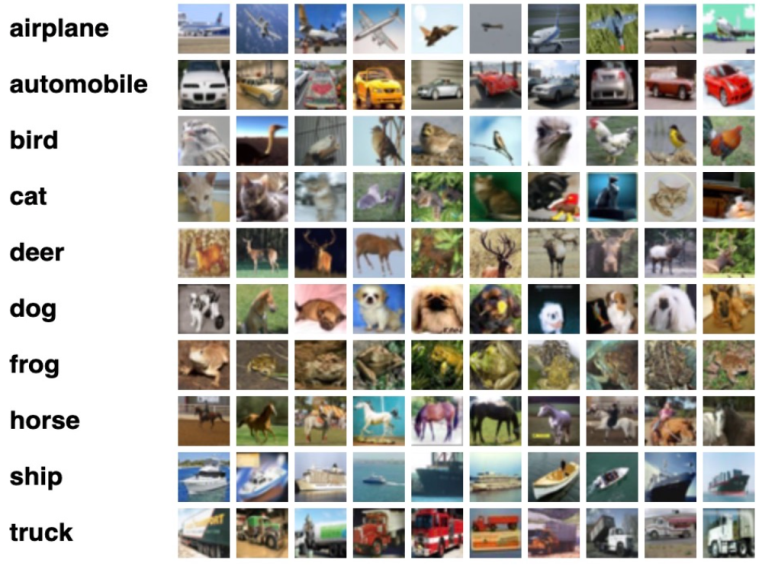
1. **Import Dependencies:** Necessary libraries are imported including TensorFlow, Matplotlib, Seaborn, NumPy, Pandas, and datetime.
2. **Load the Data:** MNIST dataset containing 60,000 training images and 10,000 test images of handwritten digits from 0 to 9 is loaded using TensorFlow's mnist.load\_data().
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:**



**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.

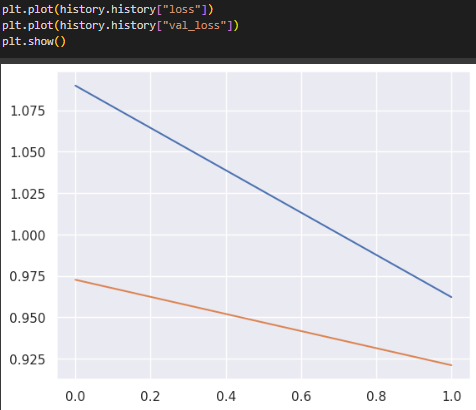
**Visualizing Model Performance:**

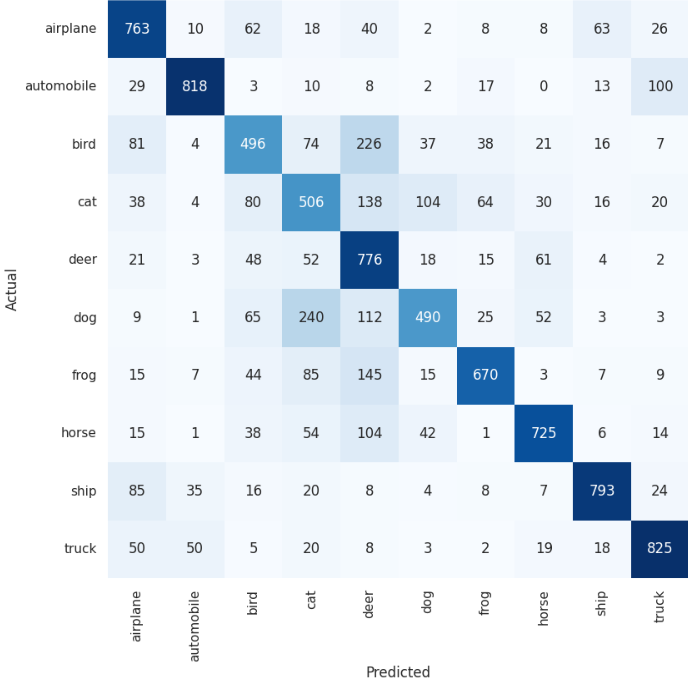
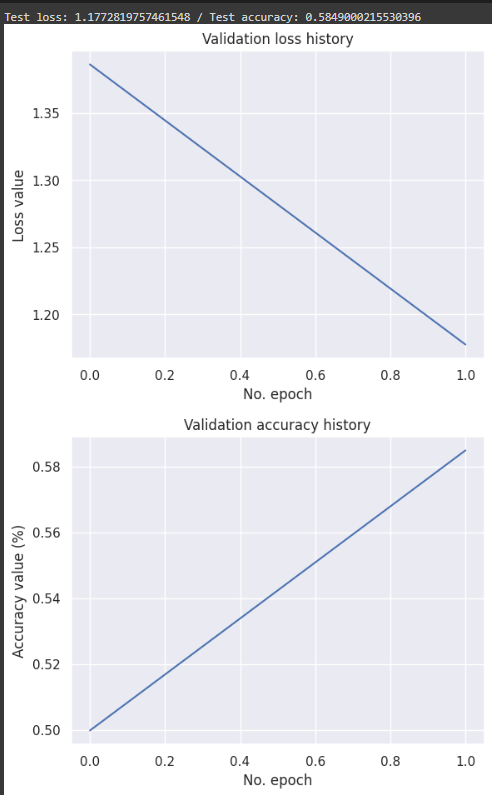
The script plots the validation loss and validation accuracy history using Matplotlib to visualize the model's performance during training.

**Visualizing Predictions:**

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

**RESULTS/GRAPHS:**





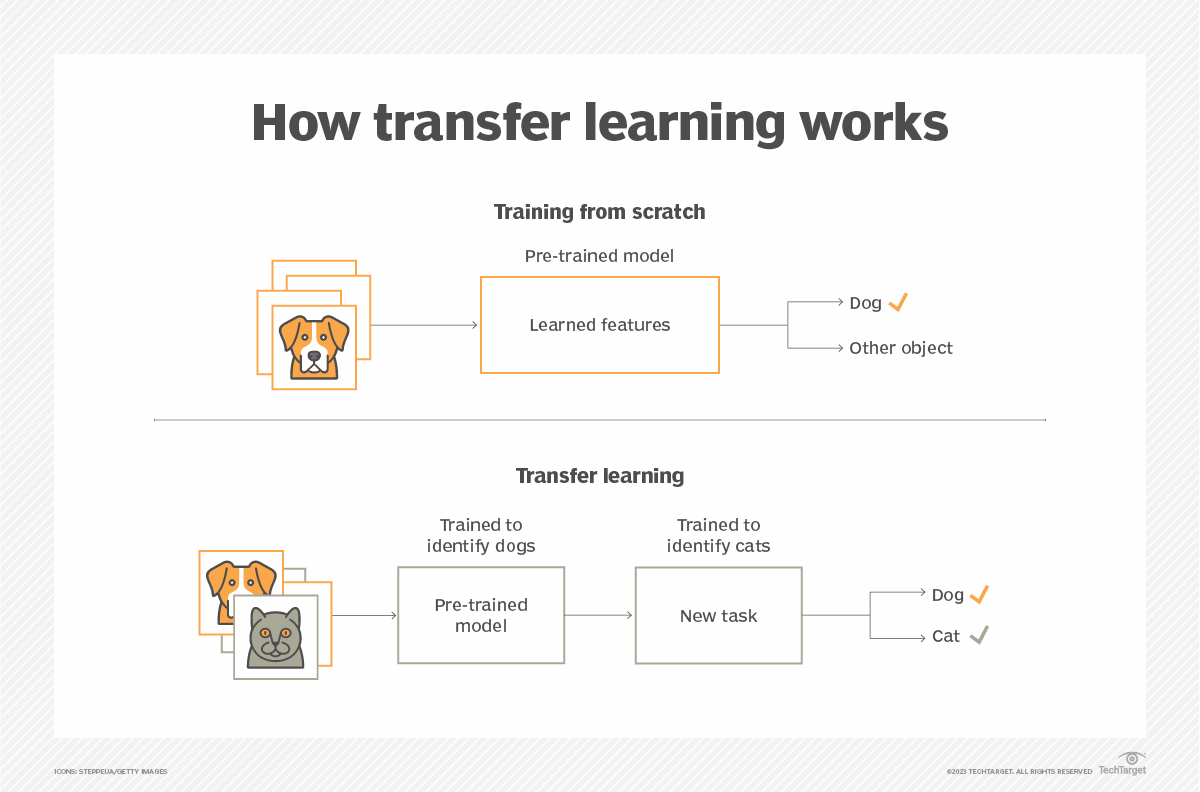
**Conclusion**

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](https://www.geeksforgeeks.org/how-to-install-python-tensorflow-in-windows/) 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.

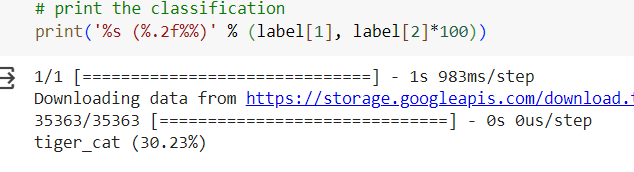
### Advantages of transfer learning:

* 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.

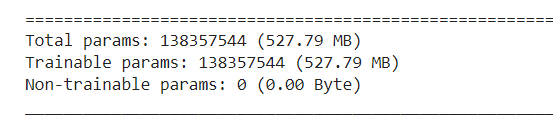
### Disadvantages of transfer learning:

* **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:**



Total Parameter:

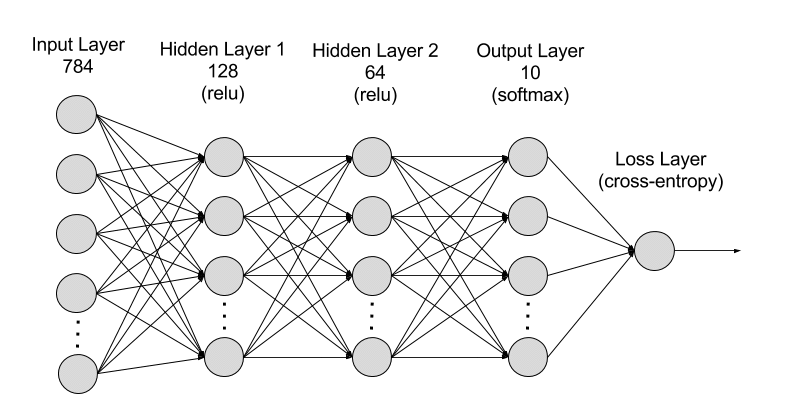


**Experiment- 15**

**Aim: Implementation of RNN**

Recurrent Neural Network (RNN) is a type of [Neural Network](https://www.geeksforgeeks.org/tag/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.

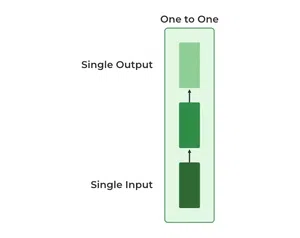
The following image shows a diagram of an RNN.:



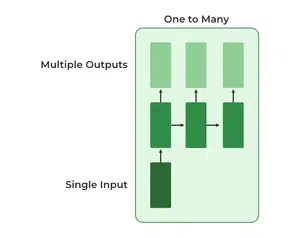
RNNs are made of neurons: data-processing nodes that work together to perform complex tasks. The neurons are organized as input, output, and hidden layers. The input layer receives the information to process, and the output layer provides the result. Data processing, analysis, and prediction take place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

**Types Of RNN**

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

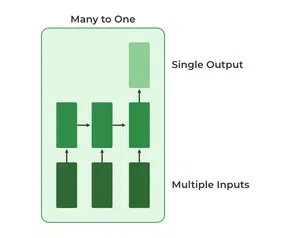
* **One to One**

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.



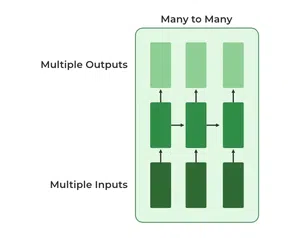
* **One To Many**

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.

* **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.

RESULTS:

