**Lab Manual of Machine Learning [CSIT-602]**

**B. Tech. VI Semester**

**Jan**

**-**

**June 202**

**3**

**Department of Computer Science and**

**Information Technology**

**Submitted to**

**Submitted By**

Prof. Shilpa Bhalerao

Jainam Singhai

Designation, CSIT Dept.

(0827CI201083)



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

**Department of Computer Science and Information Technology**

**Certificate**

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

Signature of Faculty

INDEX PAGE

Programs to be uploaded on Github.

https://github.com/kuldeeppanwar123/Machine-Learning

Link:

|  |  |  |  |
| --- | --- | --- | --- |
| Experiment No | Program | Commit date ( in  Github) | Sign of faculty |
| 1 | Python Basic Programming including Python  Data Structures such as List, Tuple, Strings, Dictionary, Lambda Functions, Python Classes and Objects and Python Libraries such as Numpy, Pandas, Mat plotlib etc. | 21/04/2023 |  |
| 2 | Python List Comprehension with examples | 21/04/2023 |  |
| 3 | Basic of Numpy, Pandas and Matplotlib | 21/04/2023 |  |
| 4 | Brief Study of Machine Learning Frameworks such as Open CV, Scikit Learn, Keras, Tensorflow etc. | 21/04/2023 |  |
| 5 | For a given set of training data examples stored in a .CSV file, implement and demonstrate the scratch Implementation of **Linear Regression Algorithm** | 22/04/2023 |  |
| 6 | For a given set of training data examples stored in a .CSV file, implement and demonstrate the Implementation of **Linear Regression Algorithm** Linear Regression using Python library (for any given CSV dataset ) | 22/04/2023 |  |
| 7 | For a given set of training data examples stored in a .CSV file, implement and demonstrate the scratch Implementation for binary classification using **Logistic Regression Algorithm** | 22/04/2023 |  |
| 8 | Build an **Artificial Neural Network**  **(ANN)** by implementing the  Backpropagation algorithm and test the same using MNIST **Handwritten Digit Multiclass classification**  data sets with use of use of batch normalization, early stopping and drop | 22/04/2023 |  |
|  | out |  |  |
| 9 | Build an **Artificial Neural Network** by implementing the Backpropagation algorithm and test the same using **CIFAR 100**  **Multiclass classification** data sets with use of use of batch normalization, early stopping and drop out | 22/04/2023 |  |
| 10 | ANN implementation use of batch  normalization, early stopping and drop out (For Image Dataset such as Covid Dataset) | 23/04/2023 |  |
| 11 | Build an **Convolutional Neural Network** by implementing the Backpropagation algorithm and test the same using **MNIST Handwritten Digit Multiclass classification**  data sets. | 23/04/2023 |  |
| 12 | Build an **Convolutional Neural Network** by implementing the Backpropagation algorithm and test the same using **CIFAR 100**  **Multiclass classification** data sets. | 23/04/2023 |  |
| 13 | Implementation of Transfer Learning (VGG 16) | 23/04/2023 |  |
| 14 | Implementation of RNN | 23/04/2023 |  |
|  |  |  |  |

**EXPERIMENT-1**

**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.

**What is Python** ?

Python is a high-level, general-purpose, interpreter and object-oriented programming language. The biggest strength of Python is huge collection of standard library which can be used for the following:

* [Machine Learning](https://www.geeksforgeeks.org/machine-learning/)
* GUI Applications (like [Kivy](https://www.geeksforgeeks.org/kivy-tutorial/" \t "_blank), Tkinter, PyQt etc. )
* Web frameworks like [Django](https://www.geeksforgeeks.org/django-tutorial/) (used by YouTube, Instagram, Dropbox)
* Image processing (like [OpenCV](https://www.geeksforgeeks.org/opencv-python-tutorial/), Pillow)
* Web scraping (like Scrapy, BeautifulSoup, Selenium)
* Test frameworks
* Multimedia
* Scientific computing
* Text processing

**Python Keywords**

Keywords are the reserved words in Python. We cannot use a keyword as a variable name, function name or any other identifier.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| [False](https://www.programiz.com/python-programming/keyword-list#true_false) | [await](https://www.programiz.com/python-programming/keyword-list#async_await) | [else](https://www.programiz.com/python-programming/keyword-list#if_else_elif) | [import](https://www.programiz.com/python-programming/keyword-list#from_import) | [pass](https://www.programiz.com/python-programming/keyword-list#pass) |
| [None](https://www.programiz.com/python-programming/keyword-list#none) | [break](https://www.programiz.com/python-programming/keyword-list#break_continue) | [except](https://www.programiz.com/python-programming/keyword-list#except_raise_try) | [in](https://www.programiz.com/python-programming/keyword-list#in) | [raise](https://www.programiz.com/python-programming/keyword-list#except_raise_try) |
| [True](https://www.programiz.com/python-programming/keyword-list#true_false) | [class](https://www.programiz.com/python-programming/keyword-list#class) | [finally](https://www.programiz.com/python-programming/keyword-list#finally) | [is](https://www.programiz.com/python-programming/keyword-list#is) | [return](https://www.programiz.com/python-programming/keyword-list#return) |
| [and](https://www.programiz.com/python-programming/keyword-list#and_or_not) | [continue](https://www.programiz.com/python-programming/keyword-list#break_continue) | [for](https://www.programiz.com/python-programming/keyword-list#for) | [lambda](https://www.programiz.com/python-programming/keyword-list#lambda) | [try](https://www.programiz.com/python-programming/keyword-list#except_raise_try) |
| [as](https://www.programiz.com/python-programming/keyword-list#as) | [def](https://www.programiz.com/python-programming/keyword-list#def) | [from](https://www.programiz.com/python-programming/keyword-list#from_import) | [nonlocal](https://www.programiz.com/python-programming/keyword-list#nonlocal) | [while](https://www.programiz.com/python-programming/keyword-list#while) |
| [assert](https://www.programiz.com/python-programming/keyword-list#assert) | [del](https://www.programiz.com/python-programming/keyword-list#del) | [global](https://www.programiz.com/python-programming/keyword-list#global) | [not](https://www.programiz.com/python-programming/keyword-list#and_or_not) | [with](https://www.programiz.com/python-programming/keyword-list#with) |
| [async](https://www.programiz.com/python-programming/keyword-list#asyn_await) | [elif](https://www.programiz.com/python-programming/keyword-list#if_else_elif) | [if](https://www.programiz.com/python-programming/keyword-list#if_else_elif) | [or](https://www.programiz.com/python-programming/keyword-list#and_or_not) | [yield](https://www.programiz.com/python-programming/keyword-list#yield) |

**Indentation in Python**

Whitespace is used for **indentation in Python**. Unlike many other programming languages which only serve to make the code easier to read, **Python indentation** is mandatory. One can understand it better by looking at an example of indentation in Python. A block is a combination of all these statements. Block can be regarded as the grouping of statements for a specific purpose. Most programming languages like C, C++, and Java use braces { } to define a block of code for indentation. One of the distinctive roles of Python is its use of indentation to highlight the blocks of code.

*# Jainam singhai (0827CI201083)*

*# indentation example*

branch = 'CSIT'

if branch == 'CSIT':

    print('Welcome to CSIT')

else:

    print('Other branch')

print('All set !')

**Comments in Python**

Python comments start with the hash symbol # and continue to the end of the line. [Comments in Python](https://www.geeksforgeeks.org/python-comments/) are useful information that the developers provide to make the reader understand the source code. It explains the logic or a part of it used in the code. Comments in Python are usually helpful to someone maintaining or enhancing your code when you are no longer around to answer questions about it. There are two types of comments:

### 1. **Single-line comment**

### 2**. multi-line comment**

*# Jainam SInghai*

print('This is python code')

*# this is single line comment*

"""

This is a multiline comment in Python that

spans several lines. This application is

a Computer Science portal for geeks.

"""

**If-else statement**

In [Python programming language](https://www.geeksforgeeks.org/python-programming-language/), the type of control flow statements are as follows:

1. [The if statement](https://www.geeksforgeeks.org/python3-if-if-else-nested-if-if-elif-statements/)
2. [The if-else statement](https://www.geeksforgeeks.org/python3-if-if-else-nested-if-if-elif-statements/)
3. [The nested-if statement](https://www.geeksforgeeks.org/python3-if-if-else-nested-if-if-elif-statements/)
4. [The if-elif-else ladder](https://www.geeksforgeeks.org/python3-if-if-else-nested-if-if-elif-statements/)

*# Jainam Singhai*

*# if else example*

*#if-else*

num = 100;

if num==100:

    print('num is 100')

else:

    print('num is not 100')

*#nested if-else*

if num<=100:

    if num<=50:

        print("number is between 0 to 50")

    else:

        print("number is between 50 to 100")

else:

    print('number is above 100')

*#if elif else*

if num==100:

    print("number is 100")

elif num==200:

    print('number is 200')

elif num==300:

    print('number is 300')

else:

    print('number something else')

**LIST**

**Python Lists**are just like dynamically sized arrays, declared in other languages (vector in C++ and ArrayList in Java). In simple language, a list is a collection of things, enclosed in [ ] and separated by commas. 

*#Jainam singhai (0827CI201083)*

*#List Data structure*

fruits = ['mango','apple',1000,'banana','orange',True,58.50]

print(fruits)     *#print list*

print('element at index 3 '+fruits[3])  *#access element using index value*

print('length of list ',len(fruits))    *#print length of list*

fruits.append(2020.55);                 *#append to the list*

fruits.insert(2,False);                 *#insert at index 2*

fruits.extend([8,'potato']);            *#insert at end*

fruits.reverse();

fruits.remove('apple');

fruits.pop(3);                          *#remove at index 3*

fruits.clear();

*# fruits.sort();                       #only work on same type of values in list*

print(fruits);

**Tuple**

**Tuple** is a collection of objects separated by commas. In some ways, a tuple is similar to a list in terms of indexing, nested objects, and repetition but a tuple is **immutable**, unlike **lists which are mutable.**

*#Jainam singhai (0827CI201083)*

*# myTuple = ('banana', 'apple', 'orange', 'pineapple');*

*#creating tuple using tuple constructor*

myTuple = tuple(('banana', 'apple', 'orange', 'pineapple'))

*print(myTuple)*

*print('length of tuple : ',len(myTuple));       #length of tuple*

*print('element at 2 ',myTuple[2]);              #access element using indexing*

*print(myTuple[1:3])                             #slicing from index 1 to 2*

*print(myTuple.count('banana'))                  #count occurrence of banana*

*print('banana' in myTuple)                      # print true if present*

mylist = ['banana', 'apple', 'orange', 'pineapple'];

convertedTuple = tuple(mylist);                     *#convert list into tuple*

**Strings in Python**

A string is a data structure in Python that represents a sequence of characters. It is an immutable data type, meaning that once you have created a string, you cannot change it.

*#Jainam singhai (0827CI201083)*

*#string data structure*

str = 'Welcome to the CSIT Department'

print(str[0])                   *#accessing using index*

print(str.capitalize())         *#capitalize the string*

print(''.join(reversed(str)))   *#reverse the string*

print(str[2:8])                 *#string slicing*

print(list(str))                *#convert string to list*

del str                         *#delete entire string*

String1 = "{1} {0} {2}".format('Geeks', 'For', 'Life')

print(str.count('to'))            *#count the word to*

print(str.split('the'))           *#split into list based on word 'the'*

print(str.upper())                *#convert into uppercase*

print(str.lower())                *#convert into lowercase*

print(str.find('the'))            *#return the starting index of 'the'*

print(str.index('to'))            *#return the index of 'to'*

**Dictionary in Python**

**Dictionary in Python** is a collection of keys values, used to store data values like a map, which, unlike other data types which hold only a single value as an element.

*#Jainam singhai (0827CI201083)*

*#Dictionary data structure*

myDict = {100:'jainam' , 200:'shahrukh',300:'jatin',400:'nikhilesh',500:'himanshu',600:'mahendra'};

print(myDict)

print(myDict.get(400))          *#get value using key*

print(myDict.values())          *#prints only values*

print(myDict.keys())            *#prints only keys*

print(myDict.pop(400))          *#pop item which has key 400*

print(myDict.popitem())         *#pop last item*

del(myDict[100])                *#delete item which has key 100*

myDict.update({10000:"ram"})      *#add new element at last*

print(myDict.\_\_sizeof\_\_())

**Lambda Functions**

**Python Lambda Functions** are anonymous function means that the function is without a name. As we already know that the *def* keyword is used to define a normal function in Python. Similarly, the *lambda* keyword is used to define an anonymous function in Python.

*#Jainam singhai (0827CI201083)*

*#lamda function*

square = lambda x:x\*2

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

newList = list(map(square , List))

print(newList)

**Class and Object**

A class is a user-defined blueprint or prototype from which objects are created. Classes provide a means of bundling data and functionality together.

Syntax:

class ClassName:

# Statement

An Object is an instance of a Class. A class is like a blueprint while an instance is a copy of the class with *actual values*.

Syntax:

Object = className()

*#Jainam Singhai (0827CI201083)*

*#class and object*

#declaring class student

class Student:

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

*self*.name = name

*self*.age = age

*self*.grade = grade

def details(self):

        print('name is : ',*self*.name)

        print('age is : ',*self*.age)

        print('grade is : ',*self*.grade)

    def get\_name(self):

        return *self*.name

    def get\_age(self):

        return *self*.age

    def get\_grade(self):

        return *self*.grade

    def set\_name(self, name):

*self*.name = name

    def set\_age(self, age):

*self*.age = age

    def set\_grade(self, grade):

*self*.grade = grade

#initializing object s1

s1 = Student('kuldeep',20,92)

s1.details()

**Python Libraries**

A Python library is a collection of related modules. It contains bundles of code that can be used repeatedly in different programs. It makes Python Programming simpler and convenient for the programmers. some of the commonly used libraries:

1. **TensorFlow:**This library was developed by Google in collaboration with the Brain Team. It is an open-source library used for high-level computations. It is also used in machine learning and deep learning algorithms. It contains a large number of tensor operations. Researchers also use this Python library to solve complex computations in Mathematics and Physics.
2. **Matplotlib:**This library is responsible for plotting numerical data. And that’s why it is used in data analysis. It is also an open-source library and plots high-defined figures like pie charts, histograms, scatterplots, graphs, etc.
3. **Pandas:**Pandas are an important library for data scientists. It is an open-source machine learning library that provides flexible high-level data structures and a variety of analysis tools. It eases data analysis, data manipulation, and cleaning of data. Pandas support operations like Sorting, Re-indexing, Iteration, Concatenation, Conversion of data, Visualizations, Aggregations, etc.
4. **Numpy:**The name “Numpy” stands for “Numerical Python”. It is the commonly used library. It is a popular machine learning library that supports large matrices and multi-dimensional data. It consists of in-built mathematical functions for easy computations. Even libraries like TensorFlow use Numpy internally to perform several operations on tensors. Array Interface is one of the key features of this library.
5. **SciPy:**The name “SciPy” stands for “Scientific Python”. It is an open-source library used for high-level scientific computations. This library is built over an extension of Numpy. It works with Numpy to handle complex computations. While Numpy allows sorting and indexing of array data, the numerical data code is stored in SciPy. It is also widely used by application developers and engineers.
6. **Scrapy:**It is an open-source library that is used for extracting data from websites. It provides very fast web crawling and high-level screen scraping. It can also be used for data mining and automated testing of data.
7. **Scikit-learn:**It is a famous Python library to work with complex data. Scikit-learn is an open-source library that supports machine learning. It supports variously supervised and unsupervised algorithms like linear regression, classification, clustering, etc. This library works in association with Numpy and SciPy.
8. **PyGame:**This library provides an easy interface to the Standard Directmedia Library (SDL) platform-independent graphics, audio, and input libraries. It is used for developing video games using computer graphics and audio libraries along with Python programming language.
9. **PyTorch:**PyTorch is the largest machine learning library that optimizes tensor computations. It has rich APIs to perform tensor computations with strong GPU acceleration. It also helps to solve application issues related to neural networks.
10. **PyBrain:**The name “PyBrain” stands for Python Based Reinforcement Learning, Artificial Intelligence, and Neural Networks library. It is an open-source library built for beginners in the field of Machine Learning. It provides fast and easy-to-use algorithms for machine learning tasks. It is so flexible and easily understandable and that’s why is really helpful for developers that are new in research fields.

**EXPERIMENT-2**

**AIM:** To study Python List Comprehension with examples.

A Python list comprehension consists of brackets containing the expression, which is executed for each element along with the for loop to iterate over each element in the [Python list](https://www.geeksforgeeks.org/python-list/). Python List comprehension provides a much more short syntax for creating a new list based on the values of an existing list.

### Advantages of List Comprehension

* More time-efficient and space-efficient than loops.
* Require fewer lines of code.
* Transforms iterative statement into a formula.

Syntax:

newList **=** **[** expression(element) **for** element **in** oldList **if** condition **]**

**if with List Comprehension**

*#Jainam singhai 0827CI20183*

*#List Comprehension*

number\_list = [ x for x in range(20) if x % 2 == 0]

print(number\_list)

**Lambda function with List Comprehension**

*#Jainam singhai 0827CI20183*

*#List Comprehension*

lettersList = list(map(lambda x: x, 'human'))

print(lettersList)

**Nested if with List Comprehension**

*#Jainam singhai 0827CI20183*

*#List Comprehension*

num\_list = [y for y in range(100) if y % 2 == 0 if y % 5 == 0]

print(num\_list)

**Nested if..else with List Comprehension**

*#Jainam singhai 0827CI20183*

*#List Comprehension*

obj = ["Even" if i%2==0 else "Odd" for i in range(10)]

print(obj)

**Note:**

* List comprehension is an elegant way to define and create lists based on existing lists.
* List comprehension is generally more compact and faster than normal functions and loops for creating list.
* However, we should avoid writing very long list comprehensions in one line to ensure that code is user-friendly.
* Remember, every list comprehension can be rewritten in for loop, but every for loop can’t be rewritten in the form of list comprehension.

**EXPERIMENT-3**

**Aim:** To study Basic of Numpy, Pandas and Matplotlib.

**Numpy:**

NumPy is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays. It is the fundamental package for scientific computing with Python. It is open-source software. It contains various features including these important ones:

* A powerful N-dimensional array object
* Sophisticated (broadcasting) functions
* Tools for integrating C/C++ and Fortran code
* Useful linear algebra, Fourier transform, and random number capabilities

**Basic Numpy Program**

#Jainam singhai 0827CI201083

import numpy as np

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

print(arr)

print(type(arr))

[1 2 3 4 5]

<class 'numpy.ndarray'>

**Pandas**

Pandas is an open-source library that is made mainly for working with relational or labeled data both easily and intuitively. It provides various data structures and operations for manipulating numerical data and time series. This library is built on top of the NumPy library. Pandas is fast and it has high performance & productivity for users.

### **Advantages**

* Fast and efficient for manipulating and analyzing data.
* Data from different file objects can be loaded.
* Easy handling of missing data (represented as NaN) in floating point as well as non-floating point data
* Size mutability: columns can be inserted and deleted from DataFrame and higher dimensional objects
* Data set merging and joining.
* Flexible reshaping and pivoting of data sets

#Jainam singhai 0827CI201083

import pandas as pd

df = pd.read\_csv('data.csv')

print(df.to\_string())

Duration Pulse Maxpulse Calories

0 60 110 130 409.1

1 60 117 145 479.0

2 60 103 135 340.0

3 45 109 175 282.4

4 45 117 148 406.0

5 60 102 127 300.5

**Matplotlib**

Matplotlib is an amazing visualization library in Python for 2D plots of arrays. Matplotlib is a multi-platform data visualization library built on NumPy arrays and designed to work with the broader SciPy stack. It was introduced by John Hunter in the year 2002. One of the greatest benefits of visualization is that it allows us visual access to huge amounts of data in easily digestible visuals. Matplotlib consists of several plots like line, bar, scatter, histogram etc.

#Jainam singhai 0827CI201083

import sys

import matplotlib

matplotlib.use('Agg')

import matplotlib.pyplot as plt

import numpy as np

xpoints = np.array([0, 6])

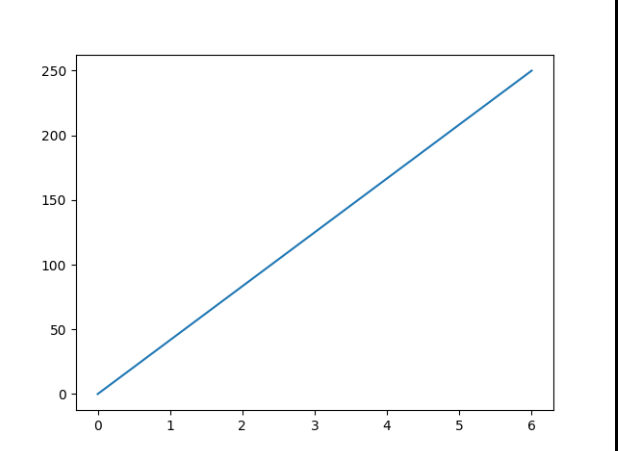
ypoints = np.array([0, 250])

plt.plot(xpoints, ypoints)

plt.show()

plt.savefig(sys.stdout.buffer)

sys.stdout.flush()



**EXPERIMENT-4**

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

**What is ML framework ?**

Machine learning relies on [algorithms](https://www.bmc.com/blogs/machine-learning-algorithms/). Unless you’re a [data scientist](https://www.bmc.com/blogs/data-engineer-vs-data-scientist/) or ML expert, these algorithms are very complicated to understand and work with. A machine learning framework, then, simplifies machine learning algorithms. An ML framework is any tool, interface, or library that lets you develop ML models easily, without understanding the underlying algorithms.There are a variety of machine learning frameworks, geared at different purposes. Nearly all ML the frameworks—those we discuss here and those we don’t—are written in Python. Python is [the predominant machine learning programming language](https://www.bmc.com/blogs/python-vs-java/).

**Open CV**

OpenCV is the huge open-source library for the computer vision, machine learning, and image processing and now it plays a major role in real-time operation which is very important in today’s systems. By using it, one can process images and videos to identify objects, faces, or even handwriting of a human. When it integrated with various libraries, such as NumPy, python is capable of processing the OpenCV array structure for analysis. To Identify image pattern and its various features we use vector space and perform mathematical operations on these features.

applications which are solved using OpenCV:

* face recognition
* Automated inspection and surveillance
* number of people – count (foot traffic in a mall, etc)
* Vehicle counting on highways along with their speeds
* Interactive art installations
* Anomaly (defect) detection in the manufacturing process (the odd defective products)
* Street view image stitching
* Video/image search and retrieval
* Robot and driver-less car navigation and control
* object recognition
* Medical image analysis
* Movies – 3D structure from motion
* TV Channels advertisement recognition

**OpenCV Functionality** 

* Image/video I/O, processing, display (core, imgproc, highgui)
* Object/feature detection (objdetect, features2d, nonfree)
* Geometry-based monocular or stereo computer vision (calib3d, stitching, videostab)
* Computational photography (photo, video, superres)
* Machine learning & clustering (ml, flann)
* CUDA acceleration (gpu)

**Scikit Learn**

Scikit-learn (Sklearn) is the most useful and robust library for machine learning in Python. It provides a selection of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction via a consistence interface in Python. This library, which is largely written in Python, is built upon NumPy, SciPy and Matplotlib.

**Uses**:

* Linear regression
* Decision tree regressions
* Random Forest regressions
* K-Nearest neighbor
* SVMs
* Stochastic Gradient Descent models

from sklearn import linear\_model

regr = linear\_model.LinearRegression()

regr.fit(diabetes\_X\_train, diabetes\_y\_train)

print(regr.coef\_)

Scikit provides model analysis tools like the [confusion matrix](https://www.bmc.com/blogs/confusion-precision-recall/) for assessing how well a model performed. Many times, you can start an ML job in scikit-learn and then move to another framework. For example, scikit-learn has excellent data pre-processing tools for one-hot encoding categorical data. Once the data is pre-processed through Scikit, you can move it into TensorFlow or PyTorch.

**Keras:**

Keras is an open-source high-level Neural Network library, which is written in Python is capable enough to run on Theano, TensorFlow, or CNTK. It was developed by one of the Google engineers, Francois Chollet. It is made user-friendly, extensible, and modular for facilitating faster experimentation with deep neural networks. It not only supports Convolutional Networks and Recurrent Networks individually but also their combination.

It cannot handle low-level computations, so it makes use of the **Backend** library to resolve it. The backend library act as a high-level API wrapper for the low-level API, which lets it run on TensorFlow, CNTK, or Theano.

Keras can be used with:

* Microsoft Cognitive Toolkit (CNTK)
* R
* Theano
* PlaidML

**Advantages of Keras**

Keras encompasses the following advantages, which are as follows:

* It is very easy to understand and incorporate the faster deployment of network models.
* It has huge community support in the market as most of the AI companies are keen on using it.
* It supports multi backend, which means you can use any one of them among TensorFlow, CNTK, and Theano with Keras as a backend according to your requirement.
* Since it has an easy deployment, it also holds support for cross-platform.

**Tensorflow**

TensorFlow was developed at Google Brain and then made into an open source project. TensorFlow can:

* Perform [regression](https://www.bmc.com/blogs/introduction-to-tensorflow-and-logistic-regression/), classification, neural networks, etc.
* Run on both CPUs and GPUs

TensorFlow is among the de facto machine learning frameworks used today, and it is free. (Google thinks the library can be free, but ML models use significant resources for production purposes, so they capitalize on selling the resources to run their tools.)

TensorFlow is a full-blown, ML research and production tool. It can be very complex—but it doesn’t have to be. Like an Excel spreadsheet, TensorFlow can be used simply or more expertly:

* TF is simple enough for the basic user who wants to return a prediction on a given set of data
* TF can also work for the advanced user who wishes to set up multiple [data pipelines](https://www.bmc.com/blogs/data-pipeline/), transform the data to fit their model, customize all layers and parameters of their model, and train on multiple machines while maintaining privacy of the user.

TensorFlow has a rich set of tools. For example, the activation functions for neural networks can do all the hard work of statistics. If we define deep learning as the ability to do neural networks, then TensorFlow does that. But it can also handle more everyday problems, like regression.

import tensorflow as tf

mnist = tf.keras.datasets.mnist

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

x\_train, x\_test = x\_train / 255.0, x\_test / 255.0

model = tf.keras.models.Sequential([

tf.keras.layers.Flatten(input\_shape=(28, 28)),

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

tf.keras.layers.Dropout(0.2),

tf.keras.layers.Dense(10, activation='softmax')

])

model.compile(optimizer='adam',

              loss='sparse\_categorical\_crossentropy',

              metrics=['accuracy'])

model.fit(x\_train, y\_train, epochs=5)

model.evaluate(x\_test, y\_test)

**EXPERIMENT-5**

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

**What is Linear Regression ?**

Linear Regression is a supervised machine learning algorithm used to predict a continuous output variable based on one or more input features. In other words, it tries to fit a line (or hyperplane) that best describes the relationship between the input features and the output variable.

**steps to implement Linear Regression Algorithm from scratch in Python:**

**Step 1: Load the data from the .CSV file :**

We will use the pandas library to read the data from the .CSV file and store it in a pandas DataFrame.

*#Jainam singhai 0827CI20183*

import pandas as pd

*# Load data from CSV file*

data = pd.read\_csv('data.csv')

*# Print first 5 rows of the data*

print(data.head())

**Step 2: Split the data into training and testing sets:**

We will use the train\_test\_split function from the sklearn library to split the data into training and testing sets.

*#Jainam singhai 0827CI20183*

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

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

X = data.iloc[:, :-1].values

y = data.iloc[:, -1].values

*# Split the data into training and testing sets*

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

**Step 3: Define the Linear Regression model:**

We will define the Linear Regression model as a class with two methods: fit and predict. The fit method will be used to train the model on the training data, and the predict method will be used to make predictions on new data.

*#Jainam singhai 0827CI201083*

class LinearRegression:

    def \_\_init\_\_(self):

*self*.coefficients = None

    def fit(self, X, y):

*# Add a column of ones to X for the bias term*

        X = np.hstack((np.ones((X.shape[0], 1)), X))

*# Calculate the coefficients using the normal equation*

*self*.coefficients = np.linalg.inv(X.T.dot(X)).dot(X.T).dot(y)

    def predict(self, X):

*# Add a column of ones to X for the bias term*

        X = np.hstack((np.ones((X.shape[0], 1)), X))

*# Make predictions using the coefficients*

        y\_pred = X.dot(*self*.coefficients)

        return y\_pred

**Step 4: Train the Linear Regression model on the training data:**

We will create an instance of the Linear Regression model and call the fit method to train it on the training data.

*#Jainam singhai 0827CI20183*

import numpy as np

*# Create an instance of the Linear Regression model*

lr = LinearRegression()

*# Train the model on the training data*

lr.fit(X\_train, y\_train)

**Step 5: Make predictions on the testing data:**

We will call the predict method of the Linear Regression model to make predictions on the testing data.

*#Jainam singhai 0827CI20183*

*# Make predictions on the testing data*

y\_pred = lr.predict(X\_test)

*# Print the predictions*

print(y\_pred)

**final code:**

*#Jainam singhai 0827CI20183*

import pandas as pd

import numpy as np

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

class LinearRegression:

    def \_\_init\_\_(self):

*self*.coefficients = None

    def fit(self, X, y):

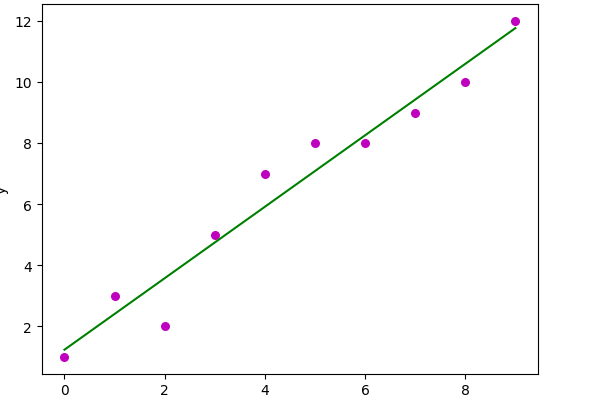
*# Add a column of ones to X for the bias term*

        X = np.hstack((np.ones((X.shape[0], 1)), X))

*# Calculate the coefficients using the normal equation*

*self*.coefficients = np.linalg.inv(X.T)

**Result:**



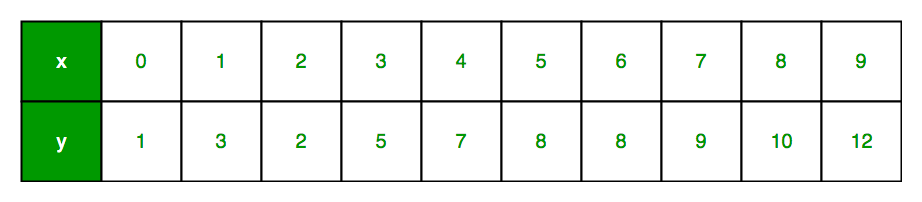
**EXPERIMENT-6**

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

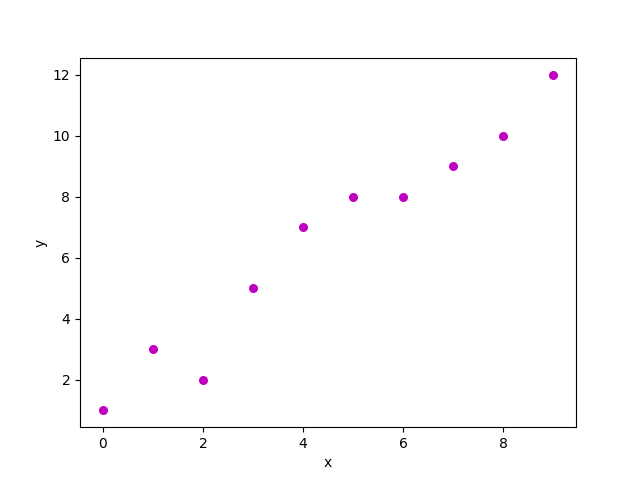
**Linear Regression**

Simple linear regression is an approach for predicting a **response** using a **single feature**. It is assumed that the two variables are linearly related. Hence, we try to find a linear function that predicts the response value(y) as accurately as possible as a function of the feature or independent variable(x).

Let us consider a dataset where we have a value of response y for every feature x:



For generality, we define:  
x as **feature vector**, i.e x = [x\_1, x\_2, …., x\_n],  
y as **response vector**, i.e y = [y\_1, y\_2, …., y\_n]  
for **n** observations (in above example, n=10).  
A scatter plot of the above dataset looks like:-



The equation of regression line is represented as:

h(xi) = B0+B1xi

Here,

* h(xi) represents the predicted response value for ith observation.
* b\_0 and b\_1 are regression coefficients and represent **y-**intercept and slope of regression line respectively.

To create our model, we must “learn” or estimate the values of regression coefficients b\_0 and b\_1. And once we’ve estimated these coefficients, we can use the model to predict responses!  
In this article, we are going to use the principle of  Least Squares.  
Here, e\_i is a residual error in ith observation.   
So, our aim is to minimize the total residual error.  
We define the squared error or cost function, J as:   
and our task is to find the value of b\_0 and b\_1 for which J(b\_0,b\_1) is minimum!  
Without going into the mathematical details, we present the result here:  
where SS\_xy is the sum of cross-deviations of y and x:   
and SS\_xx is the sum of squared deviations of x.

# Jainam singhai 0827CI201083

import numpy as np

import matplotlib.pyplot as plt

def estimate\_coef(x, y):

    # number of observations/points

    n = np.size(x)

    # mean of x and y vector

    m\_x = np.mean(x)

    m\_y = np.mean(y)

    # calculating cross-deviation and deviation about x

    SS\_xy = np.sum(y\*x) - n\*m\_y\*m\_x

    SS\_xx = np.sum(x\*x) - n\*m\_x\*m\_x

    # calculating regression coefficients

    b\_1 = SS\_xy / SS\_xx

    b\_0 = m\_y - b\_1\*m\_x

    return (b\_0, b\_1)

def plot\_regression\_line(x, y, b):

    # plotting the actual points as scatter plot

    plt.scatter(x, y, color = "m",

               marker = "o", s = 30)

    # predicted response vector

    y\_pred = b[0] + b[1]\*x

    # plotting the regression line

    plt.plot(x, y\_pred, color = "g")

    # putting labels

    plt.xlabel('x')

    plt.ylabel('y')

    # function to show plot

    plt.show()

def main():

    # observations / data

    x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

    y = np.array([1, 3, 2, 5, 7, 8, 8, 9, 10, 12])

    # estimating coefficients

    b = estimate\_coef(x, y)

    print("Estimated coefficients:\nb\_0 = {}  \

          \nb\_1 = {}".format(b[0], b[1]))

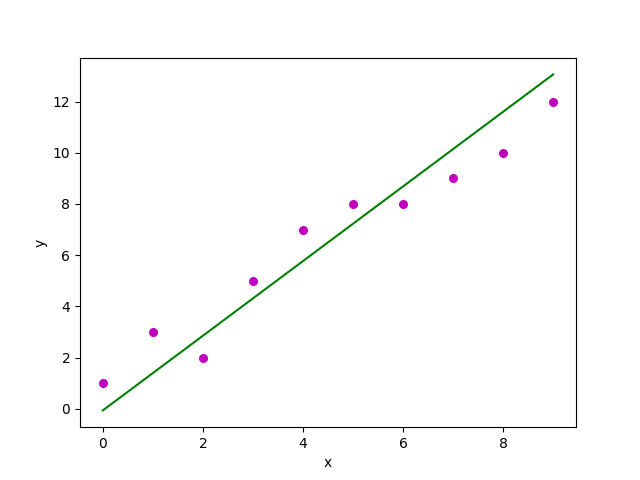
    # plotting regression line

    plot\_regression\_line(x, y, b)

if \_\_name\_\_ == "\_\_main\_\_":

    main()

**graph looks like:-**



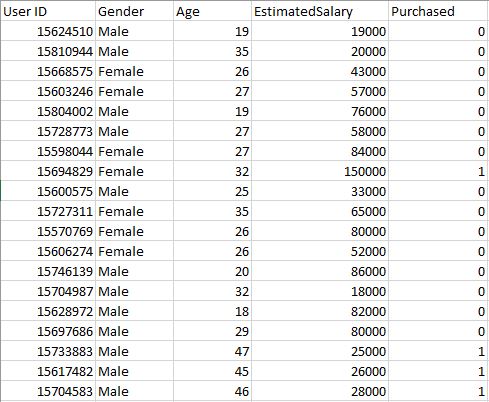
**EXPERIMENT-7**

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

**Logistic Regression Algorithm**

Logistic regression is one of the most popular Machine Learning algorithms, which comes under the Supervised Learning technique. It is used for predicting the categorical dependent variable using a given set of independent variables. Logistic regression predicts the output of a categorical dependent variable. Therefore the outcome must be a categorical or discrete value. It can be either Yes or No, 0 or 1, true or False, etc. but instead of giving the exact value as 0 and 1, it gives the probabilistic values which lie between 0 and 1.Logistic Regression is much similar to the Linear Regression except that how they are used. Linear Regression is used for solving Regression problems, whereas Logistic regression is used for solving the classification problems.

**Dataset:**



**Import Libraries:**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

**Read Data**

dataset **=** pd.read\_csv("User\_Data.csv")

**Split the Dataset**

**#** Jainam singhai0827CI201083

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

xtrain, xtest, ytrain, ytest **=** train\_test\_split(

    x, y, test\_size**=**0.25, random\_state**=**0)

it is very important to perform feature scaling here because Age and Estimated Salary values lie in different ranges. If we don’t scale the features then the Estimated Salary feature will dominate the Age feature when the model finds the nearest neighbor to a data point in the data space.

**#** Jainam singhai0827CI201083

**from** sklearn.preprocessing **import** StandardScaler

sc\_x **=** StandardScaler()

xtrain **=** sc\_x.fit\_transform(xtrain)

xtest **=** sc\_x.transform(xtest)

print (xtrain[0:10, :])

**Train the Model**

**#** Jainam singhai 0827CI201083

**from** sklearn.linear\_model **import** LogisticRegression

classifier **=** LogisticRegression(random\_state **=** 0)

classifier.fit(xtrain, ytrain)

**Evaluation Metrics**

**#** Jainam singhai0827CI201083

**from** sklearn.metrics **import** confusion\_matrix

cm **=** confusion\_matrix(ytest, y\_pred)

print ("Confusion Matrix : \n", cm)

**Visualizing the performance of Model.**

**#** Jainam singhai0827CI201083

**from** matplotlib.colors **import** ListedColormap

X\_set, y\_set **=** xtest, ytest

X1, X2 **=** np.meshgrid(np.arange(start **=** X\_set[:, 0].min() **-** 1,

                               stop **=** X\_set[:, 0].max() **+** 1, step **=** 0.01),

                     np.arange(start **=** X\_set[:, 1].min() **-** 1,

                               stop **=** X\_set[:, 1].max() **+** 1, step **=** 0.01))

plt.contourf(X1, X2, classifier.predict(

             np.array([X1.ravel(), X2.ravel()]).T).reshape(

             X1.shape), alpha **=** 0.75, cmap **=** ListedColormap(('red', 'green')))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

**for** i, j **in** enumerate(np.unique(y\_set)):

    plt.scatter(X\_set[y\_set **==** j, 0], X\_set[y\_set **==** j, 1],

                c **=** ListedColormap(('red', 'green'))(i), label **=** j)

plt.title('Classifier (Test set)')

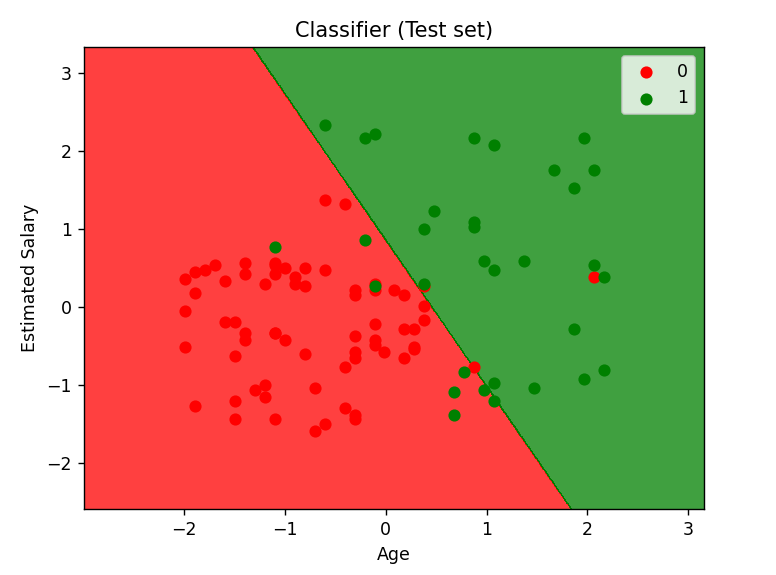
plt.xlabel('Age')

plt.ylabel('Estimated Salary')

plt.legend()

plt.show()

**OUTPUT:**



**EXPERIMENT-8**

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

**Backpropagation:**

The Backpropagation algorithm is a supervised learning method for multilayer feed-forward networks from the field of Artificial Neural Networks. The principle of the backpropagation approach is to model a given function by modifying internal weightings of input signals to produce an expected output signal. The system is trained using a supervised learning method, where the error between the system’s output and a known expected output is presented to the system and used to modify its internal state. Backpropagation can be used for both classification and regression problems.

**Implementation:**

**Import Libraries**

*#Jainam singhai 0827CI201083*

*# Import Libraries*

import numpy as np

import pandas as pd

from sklearn.datasets import load\_iris

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

import matplotlib.pyplot as plt

**Load Dataset**

*#Jainam singhai 0827CI201083*

*# Load dataset*

data = load\_iris()

*# Get features and target*

X=data.data

y=data.target

**Prepare Dataset**

*#Jainam singhai 0827CI201083*

*# Get dummy variable*

y = pd.get\_dummies(y).values

y[:3]

**Split dataset**

*#Jainam singhai 0827CI201083*

*#Split data into train and test data*

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=20, random\_state=4)

**Initialize Parameters and Weights**

*#Jainam singhai 0827CI201083*

*# Initialize variables*

learning\_rate = 0.1

iterations = 5000

N = y\_train.size

*# number of input features*

input\_size = 4

*# number of hidden layers neurons*

hidden\_size = 2

*# number of neurons at the output layer*

output\_size = 3

results = pd.DataFrame(columns=["mse", "accuracy"])

**Helper function**

*#Jainam singhai 0827CI201083*

def sigmoid(x):

    return 1 / (1 + np.exp(-x))

def mean\_squared\_error(y\_pred, y\_true):

    return ((y\_pred - y\_true)\*\*2).sum() / (2\*y\_pred.size)

def accuracy(y\_pred, y\_true):

    acc = y\_pred.argmax(axis=1) == y\_true.argmax(axis=1)

    return acc.mean()

**Backpropagation**

*#Jainam singhai 0827CI201083*

for itr in range(iterations):

*# feedforward propagation* *on hidden layer*

    Z1 = np.dot(x\_train, W1)

    A1 = sigmoid(Z1)

*# on output layer*

    Z2 = np.dot(A1, W2)

    A2 = sigmoid(Z2)

*# Calculating error*

    mse = mean\_squared\_error(A2, y\_train)

    acc = accuracy(A2, y\_train)

    results=results.append({"mse":mse, "accuracy":acc},ignore\_index=True )

*# backpropagation*

    E1 = A2 - y\_train

    dW1 = E1 \* A2 \* (1 - A2)

    E2 = np.dot(dW1, W2.T)

    dW2 = E2 \* A1 \* (1 - A1)

*# weight updates*

    W2\_update = np.dot(A1.T, dW1) / N

    W1\_update = np.dot(x\_train.T, dW2) / N

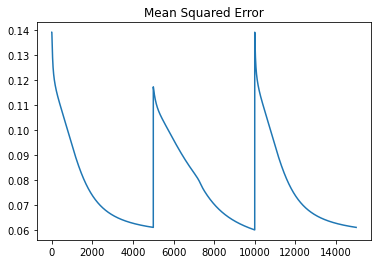
    W2 = W2 - learning\_rate \* W2\_update

    W1 = W1 - learning\_rate \* W1\_update

**Plot MSE**

*#Jainam singhai 0827CI201083*

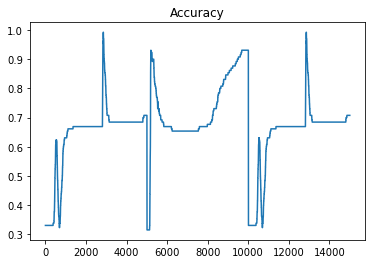
results.mse.plot(title="Mean Squared Error")



**Plot Accuracy**

*#Jainam singhai 0827CI201083*

results.accuracy.plot(title="Accuracy")



**Test the Performance**

*#Jainam singhai 0827CI201083*

Z1 = np.dot(x\_test, W1)

A1 = sigmoid(Z1)

Z2 = np.dot(A1, W2)

A2 = sigmoid(Z2)

acc = accuracy(A2, y\_test)

print("Accuracy: {}".format(acc))

**output:**

Accuracy: 0.8

**EXPERIMENT-9**

**Aim**: Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using CIFAR 100 Multiclass classification data sets with use of use of batch normalization, early stopping and drop out.

**Backpropagation:**

The Backpropagation algorithm is a supervised learning method for multilayer feed-forward networks from the field of Artificial Neural Networks. The principle of the backpropagation approach is to model a given function by modifying internal weightings of input signals to produce an expected output signal. The system is trained using a supervised learning method, where the error between the system’s output and a known expected output is presented to the system and used to modify its internal state. Backpropagation can be used for both classification and regression problems.

**Implementation:**

**Import Libraries**

*#Jainam singhai 0827CI201083*

*# Import Libraries*

import numpy as np

import pandas as pd

from sklearn.datasets import load\_iris

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

import matplotlib.pyplot as plt

**Load Dataset**

*#Jainam singhai 0827CI201083*

*# Load dataset*

data = load\_iris()

*# Get features and target*

X=data.data

y=data.target

**Prepare Dataset**

*#Jainam singhai 0827CI201083*

*# Get dummy variable*

y = pd.get\_dummies(y).values

y[:3]

**Split dataset**

*#Jainam singhai 0827CI201083*

*#Split data into train and test data*

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=20, random\_state=4)

**Initialize Parameters and Weights**

*#Jainam singhai 0827CI201083*

*# Initialize variables*

learning\_rate = 0.1

iterations = 5000

N = y\_train.size

*# number of input features*

input\_size = 4

*# number of hidden layers neurons*

hidden\_size = 2

*# number of neurons at the output layer*

output\_size = 3

results = pd.DataFrame(columns=["mse", "accuracy"])

**Backpropagation**

*#Jainam singhai 0827CI201083*

for itr in range(iterations):

*# feedforward propagation* *on hidden layer*

    Z1 = np.dot(x\_train, W1)

    A1 = sigmoid(Z1)

*# on output layer*

    Z2 = np.dot(A1, W2)

    A2 = sigmoid(Z2)

*# Calculating error*

    mse = mean\_squared\_error(A2, y\_train)

    acc = accuracy(A2, y\_train)

    results=results.append({"mse":mse, "accuracy":acc},ignore\_index=True )

*# backpropagation*

    E1 = A2 - y\_train

    dW1 = E1 \* A2 \* (1 - A2)

    E2 = np.dot(dW1, W2.T)

    dW2 = E2 \* A1 \* (1 - A1)

*# weight updates*

    W2\_update = np.dot(A1.T, dW1) / N

    W1\_update = np.dot(x\_train.T, dW2) / N

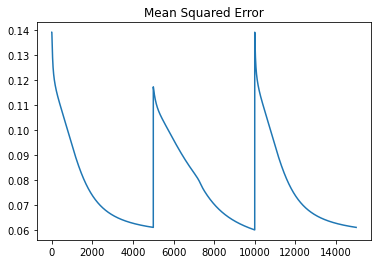
    W2 = W2 - learning\_rate \* W2\_update

    W1 = W1 - learning\_rate \* W1\_update

**Plot MSE**

*#Jainam singhai 0827CI201083*

results.mse.plot(title="Mean Squared Error")



**Testing using CIFAR 100 Multiclass Classification Dataset:**

*#Jainam singhai 0827CI201083*

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, Dropout, BatchNormalization

from tensorflow.keras.callbacks import EarlyStopping

*# Load the CIFAR-100 dataset*

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

*# Normalize the input images to have pixel values between 0 and 1*

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

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

*# Convert labels to one-hot encoded vectors*

num\_classes = 100

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

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

*# Define the model architecture*

model = Sequential([

*# Input layer*

    tf.keras.layers.Flatten(input\_shape=(32, 32, 3)),

*# Hidden layer 1*

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

    tf.keras.layers.BatchNormalization(),

    tf.keras.layers.Dropout(0.2),

*# Hidden layer 2*

    tf.keras.layers.Dense(256, activation='relu'),

    tf.keras.layers.BatchNormalization(),

    tf.keras.layers.Dropout(0.2),

*# Output layer*

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

])

*# Compile the model*

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

*# Define early stopping callback*

early\_stop = EarlyStopping(monitor='val\_loss', patience=5)

*# Train the model*

history = model.fit(x\_train, y\_train, epochs=50, batch\_size=64, validation\_split=0.2, callbacks=[early\_stop])

*# Evaluate the model on test set*

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

*# Print the test accuracy*

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



**EXPERIMENT-10**

**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 to each layer in the network to have zero mean and unit variance. It is used to speed up the training process and improve the performance of ANNs. By reducing the internal covariate shift, batch normalization helps to stabilize the distribution of the output of each layer, making the training process more efficient and reducing the risk of overfitting.

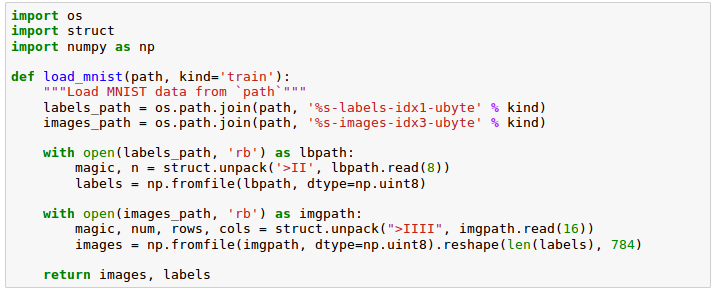
**Early stopping**

Early stopping is a technique that monitors the validation error during training and stops the training process when the validation error stops improving. This helps to prevent overfitting by stopping the training process before the model starts to fit the noise in the training data.

**Drop Out**

Drop out is a technique that randomly drops out some of the neurons in the network during training. This helps to prevent overfitting by forcing the remaining neurons to learn more robust features.

**Loading Images**



**Read image**

#Jainam singhai0827CI201083

magic, n = struct.unpack('>II', lbpath.read(8))

labels = np.fromfile(lbpath, dtype=np.uint8)

#Jainam singhai0827CI201083

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

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

my\_tuple = (6, 7, 8, 9, 10)

my\_string = "Hello, world!"

my\_dict = {'name': 'John', 'age': 30, 'city': 'New York'}

multiply = lambda x, y: x \* y

class Person:

    def \_init\_(self, name, age):

*self*.name = name

*self*.age = age

    def say\_hello(self):

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

person1 = Person("John", 30)

print(person1.name)

person1.say\_hello()

my\_array = np.array([1, 2, 3, 4, 5])

print(my\_array)

my\_dataframe = pd.DataFrame({'name': ['John', 'Mary', 'Adam'], 'age': [30, 25, 40]})

print(my\_dataframe)

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

y = np.sin(x)

plt.plot(x, y)

plt.show()

**Using Dropout and Batch Normalization**

from tensorflow import keras

from tensorflow.keras import layers

model = keras.Sequential([

    layers.Dense(1024, activation='relu', input\_shape=[11]),

    layers.Dropout(0.3),

    layers.BatchNormalization(),

    layers.Dense(1024, activation='relu'),

    layers.Dropout(0.3),

    layers.BatchNormalization(),

    layers.Dense(1024, activation='relu'),

    layers.Dropout(0.3),

    layers.BatchNormalization(),

    layers.Dense(1),

])

model.compile(

    optimizer='adam',

    loss='mae',

)

history = model.fit(

    X\_train, y\_train,

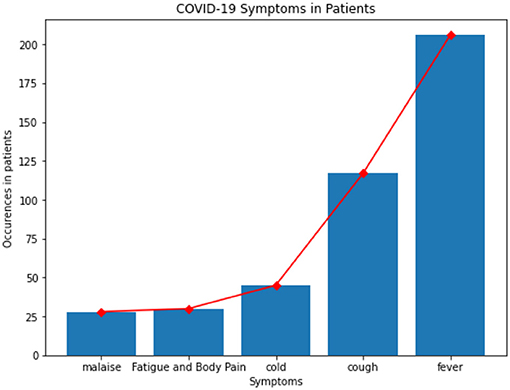
    validation\_data=(X\_valid, y\_valid),

    batch\_size=256,

    epochs=100,

    verbose=0,

)



**EXPERIMENT-11**

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

**Convolutional Neural Network**

A Convolutional Neural Network (CNN) is a type of deep learning algorithm that is particularly well-suited for image recognition and processing tasks. It is made up of multiple layers, including convolutional layers, pooling layers, and fully connected layers.The convolutional layers are the key component of a CNN, where filters are applied to the input image to extract features such as edges, textures, and shapes. The output of the convolutional layers is then passed through pooling layers, which are used to down-sample the feature maps, reducing the spatial dimensions while retaining the most important information. The output of the pooling layers is then passed through one or more fully connected layers, which are used to make a prediction or classify the image.

**Load MNIST Handwritten Digit Dataset**

*#Jainam singhai0827CI201083*

import tensorflow as tf

from tensorflow.keras.datasets import mnist

*# Load the MNIST dataset*

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

**Pre-processing Dataset**

*#Jainam singhai0827CI201083*

x\_train = x\_train / 255.0

x\_test = x\_test / 255.0

*# Reshape the input data to have a depth of 1*

x\_train = x\_train.reshape(x\_train.shape[0], 28, 28, 1)

x\_test = x\_test.reshape(x\_test.shape[0], 28, 28, 1)

**Define CNN Architecture**

*#Jainam singhai0827CI201083*

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

*# Define the CNN architecture*

model = Sequential([

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

    MaxPooling2D((2, 2)),

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

    MaxPooling2D((2, 2)),

    Flatten(),

    Dense(64, activation='relu'),

    Dense(10, activation='softmax')

])

**Train CNN using Backpropagation algorithm**

# Compile the model

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

# Train the model

model.fit(x\_train, tf.keras.utils.to\_categorical(y\_train), epochs=5, batch\_size=32)

**Testing Performance**

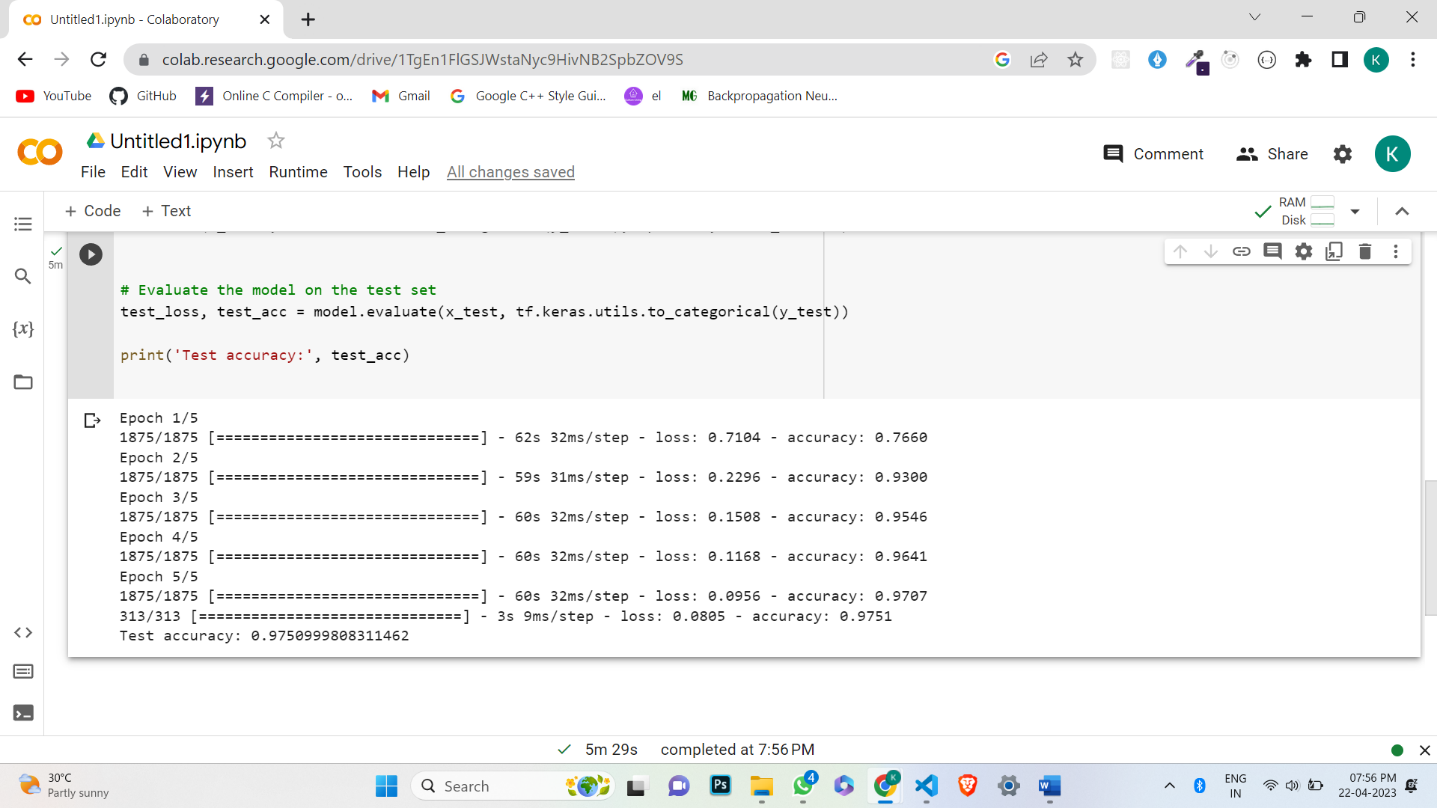
*#Jainam singhai0827CI201083*

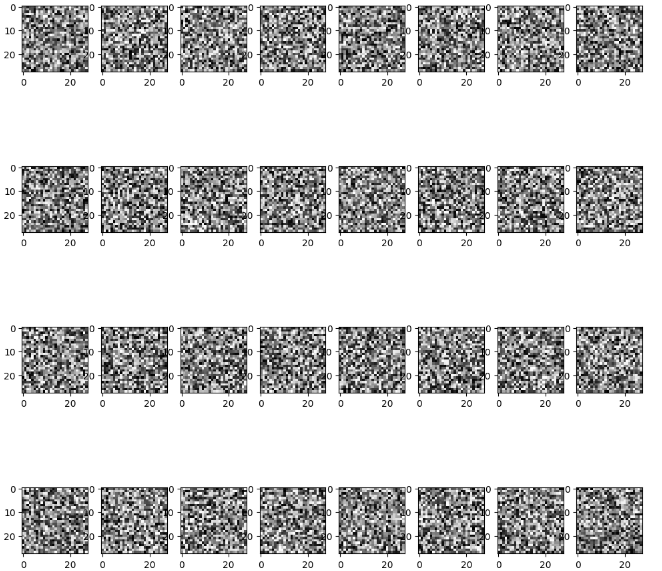
*# Evaluate the model on the test set*

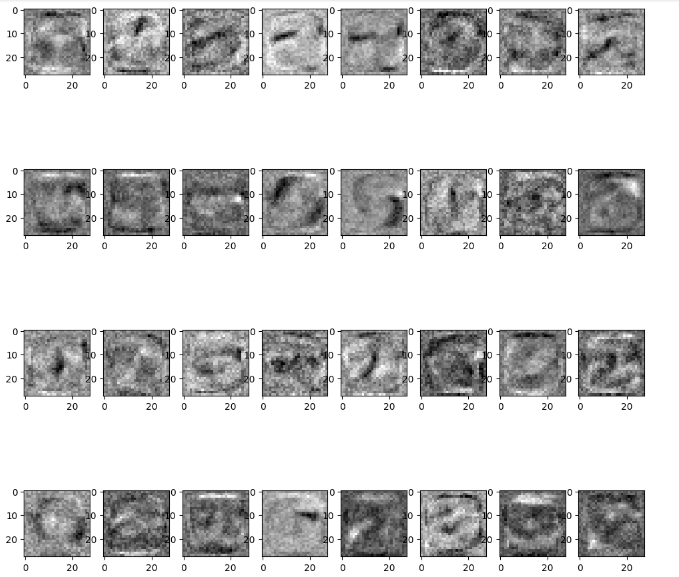
test\_loss, test\_acc = model.evaluate(x\_test, tf.keras.utils.to\_categorical(y\_test))

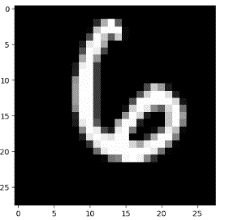
print('Test accuracy:', test\_acc)

**Output:**

**







**EXPERIMENT-12**

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

**Convolutional Neural Network**

**Convolutional Neural Network** is one of the main categories to do image classification and image recognition in neural networks. Scene labeling, objects detections, and face recognition, etc., are some of the areas where convolutional neural networks are widely used. CNN takes an image as input, which is classified and process under a certain category such as dog, cat, lion, tiger, etc. The computer sees an image as an array of pixels and depends on the resolution of the image. Based on image resolution, it will see as **h \* w \* d**, where h= height w= width and d= dimension.

**Steps for building a CNN with backpropagation:**

**Data preprocessing:** Load and preprocess the CIFAR 100 dataset. This may involve tasks such as splitting the data into training, validation, and test sets, normalization, and data augmentation.

**Model architecture:** Define the CNN architecture, including the number of layers, the type of layers (convolutional, pooling, dense), activation functions, and the number of filters.

**Forward propagation:** Implement the forward propagation algorithm to compute the output of the CNN for a given input.

**Loss function:** Define a suitable loss function to measure the error between the predicted output and the actual output.

**Backpropagation:** Implement the backpropagation algorithm to compute the gradients of the loss function with respect to the parameters of the CNN.

**Update parameters:** Use the computed gradients to update the parameters of the CNN using an optimization algorithm such as stochastic gradient descent.

**Training:** Train the CNN on the training dataset by iterating over the training examples, computing the loss, and updating the parameters using backpropagation.

**Evaluation:** Evaluate the performance of the trained CNN on the validation and test sets.

**Fine-tuning:** Fine-tune the CNN by adjusting the hyperparameters, such as the learning rate, batch size, and regularization, based on the performance on the validation set.

**Test**: Test the final CNN on the test set and report the performance metrics, such as accuracy, precision, and recall.

*#Jainam singhai0827CI201083*

import tensorflow as tf

from tensorflow import keras

from tensorflow.keras import layers

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

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

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

model = keras.Sequential([

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

    layers.MaxPooling2D((2, 2)),

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

    layers.MaxPooling2D((2, 2)),

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

    layers.MaxPooling2D((2, 2)),

    layers.Flatten(),

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

    layers.Dense(100)

])

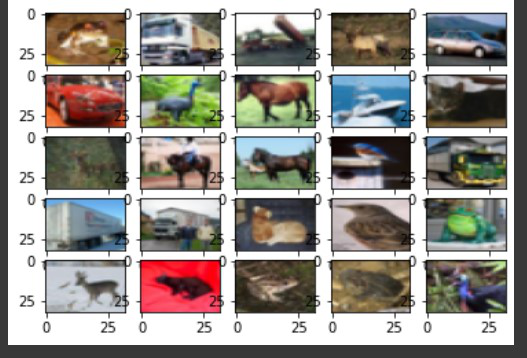
model.compile(optimizer='adam',

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

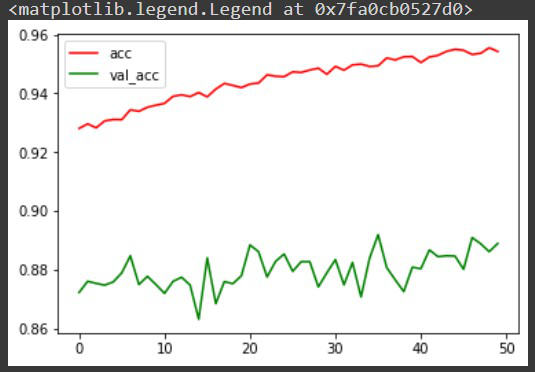
              metrics=['accuracy'])

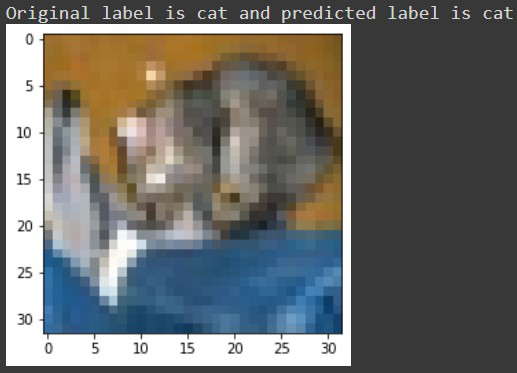
model.fit(x\_train, y\_train, epochs=1, validation\_data=(x\_test, y\_test))

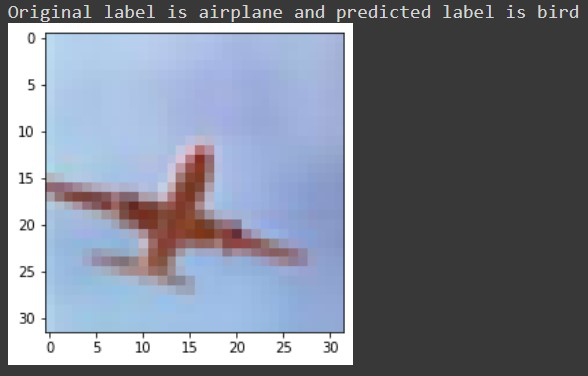
**Output**:











**EXPERIMENT-13**

**Aim**: Implementation of Transfer Learning (VGG 16).

**What is Transfer learning ?**

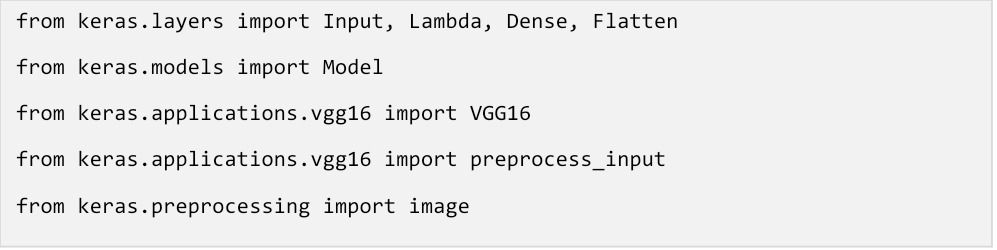
Transfer learning is a machine learning technique in which a pre-trained model developed for one task is used as a starting point for a different but related task. In transfer learning, a pre-trained model is adapted to a new problem domain by adjusting the weights of the model to fit the new data. This approach can save a significant amount of time and resources that would otherwise be required to train a model from scratch.

**What is VGG ?**

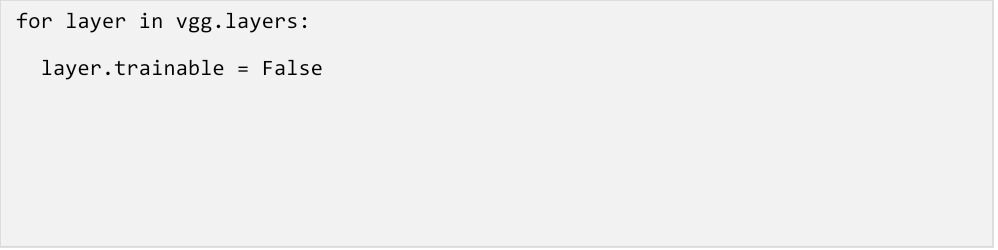
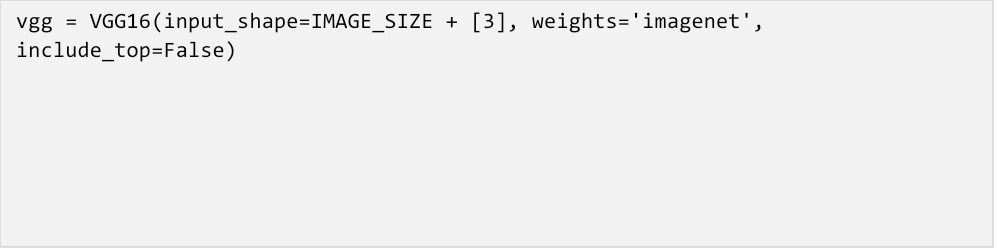
VGG 16 is a popular deep learning model for image recognition tasks that was developed by the Visual Geometry Group (VGG) at the University of Oxford. It consists of 16 layers and is trained on a large dataset of images called ImageNet. The VGG 16 model has achieved state-of-the-art performance on several image recognition benchmarks.

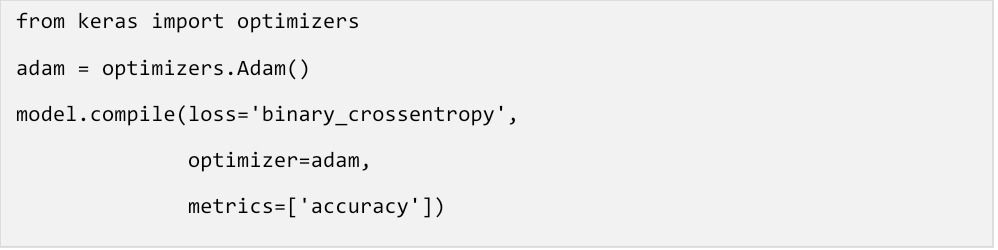
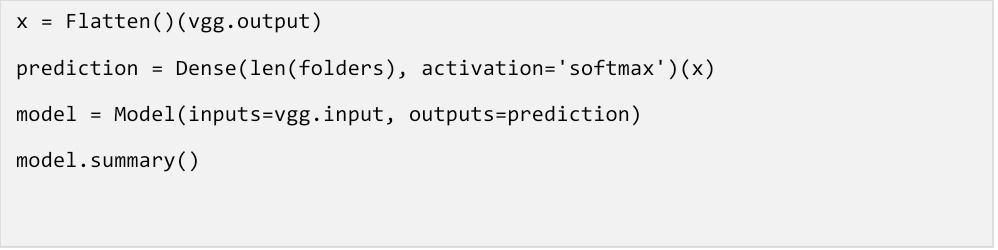
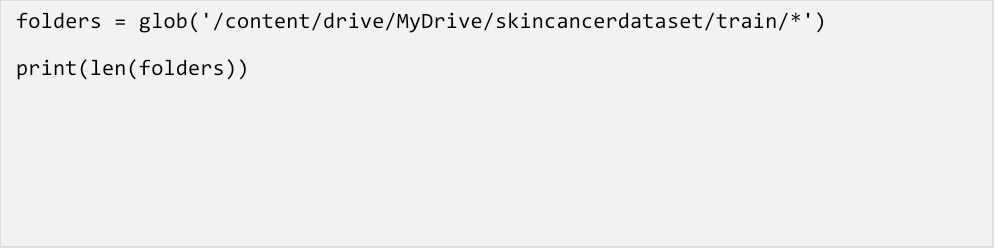
By using transfer learning with the VGG 16 model, one can start with the pre-trained weights of the VGG 16 model and fine-tune the model for a specific image recognition task. For example, if one wanted to build a model to recognize specific objects in medical images, one could use the pre-trained VGG 16 model as a starting point and fine-tune it on a smaller dataset of medical images. This can save a significant amount of time and resources compared to training a new model from scratch.

**importing Libraries**

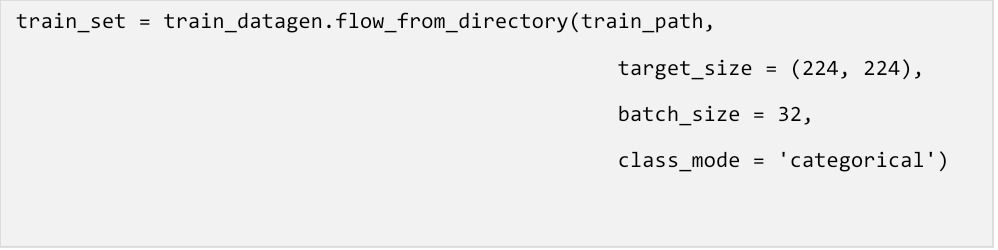
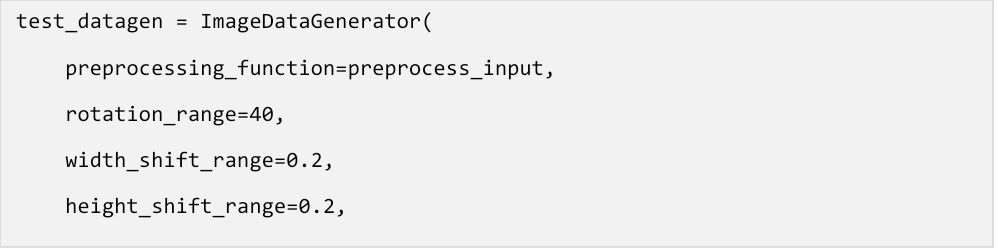
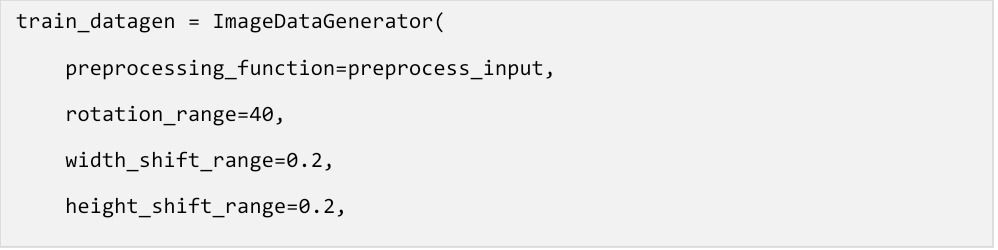


**Implementing Tranfer Learning**

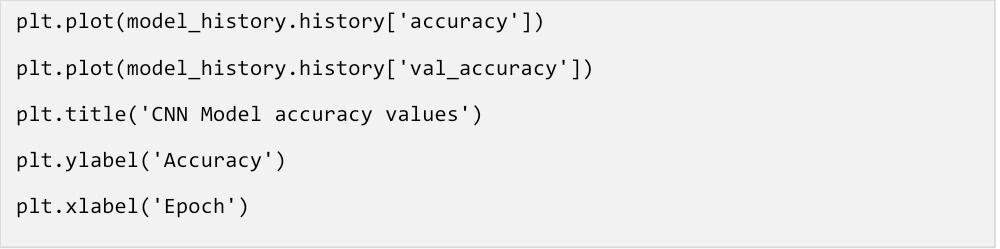




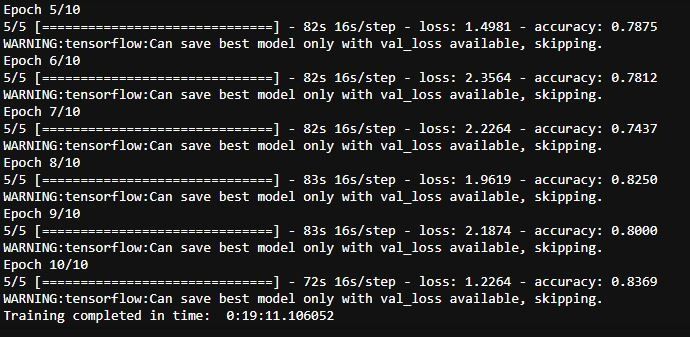
**Data augmentation**

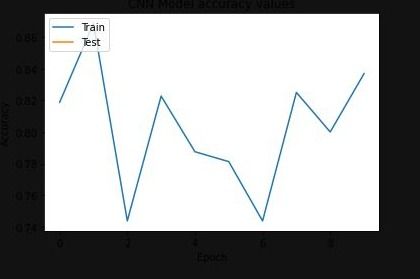


**Training Model**



**Output:**





**EXPERIMENT-14**

**Aim**: Implementation of RNN.

**What is RNN?**

RNN stands for Recurrent Neural Network. It is a type of artificial neural network that is designed to work with sequential data, such as time-series data or natural language processing (NLP) data. RNNs can be used for a variety of tasks such as language modeling, speech recognition, machine translation, and image captioning. The key feature of RNNs is that they have a recurrent connection that allows information to be passed from one step of the sequence to the next. This allows the network to maintain a memory of what it has seen earlier in the sequence and use it to make predictions at each step. The basic building block of an RNN is a cell, which takes an input and a hidden state as input and produces an output and a new hidden state as output.

*#Jainam singhai0827CI201083*

import numpy as np

import tensorflow as tf

from tensorflow import keras

from tensorflow.keras import layers

model = keras.Sequential()

*# Add an Embedding layer expecting input vocab of size 1000, and*

*# output embedding dimension of size 64.*

model.add(layers.Embedding(input\_dim=1000, output\_dim=64))

*# Add a LSTM layer with 128 internal units.*

model.add(layers.LSTM(128))

*# Add a Dense layer with 10 units.*

model.add(layers.Dense(10))

model.summary()

model = keras.Sequential()

model.add(layers.Embedding(input\_dim=1000, output\_dim=64))

*# The output of GRU will be a 3D tensor of shape (batch\_size, timesteps, 256)*

model.add(layers.GRU(256, return\_sequences=True))

*# The output of SimpleRNN will be a 2D tensor of shape (batch\_size, 128)*

model.add(layers.SimpleRNN(128))

model.add(layers.Dense(10))

model.summary()

encoder\_vocab = 1000

decoder\_vocab = 2000

encoder\_input = layers.Input(shape=(None,))

encoder\_embedded = layers.Embedding(input\_dim=encoder\_vocab, output\_dim=64)(

    encoder\_input

)

*# Return states in addition to output*

output, state\_h, state\_c = layers.LSTM(64, return\_state=True, name="encoder")(

    encoder\_embedded

)

encoder\_state = [state\_h, state\_c]

decoder\_input = layers.Input(shape=(None,))

decoder\_embedded = layers.Embedding(input\_dim=decoder\_vocab, output\_dim=64)(

    decoder\_input

)

*# Pass the 2 states to a new LSTM layer, as initial state*

decoder\_output = layers.LSTM(64, name="decoder")(

    decoder\_embedded, initial\_state=encoder\_state

)

output = layers.Dense(10)(decoder\_output)

model = keras.Model([encoder\_input, decoder\_input], output)

model.summary()

Model: "model"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param # Connected to

==================================================================================================

input\_1 (InputLayer) [(None, None)] 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

input\_2 (InputLayer) [(None, None)] 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

embedding\_2 (Embedding) (None, None, 64) 64000 input\_1[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

embedding\_3 (Embedding) (None, None, 64) 128000 input\_2[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

encoder (LSTM) [(None, 64), (None, 33024 embedding\_2[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

decoder (LSTM) (None, 64) 33024 embedding\_3[0][0]

encoder[0][1]

encoder[0][2]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_2 (Dense) (None, 10) 650 decoder[0][0]

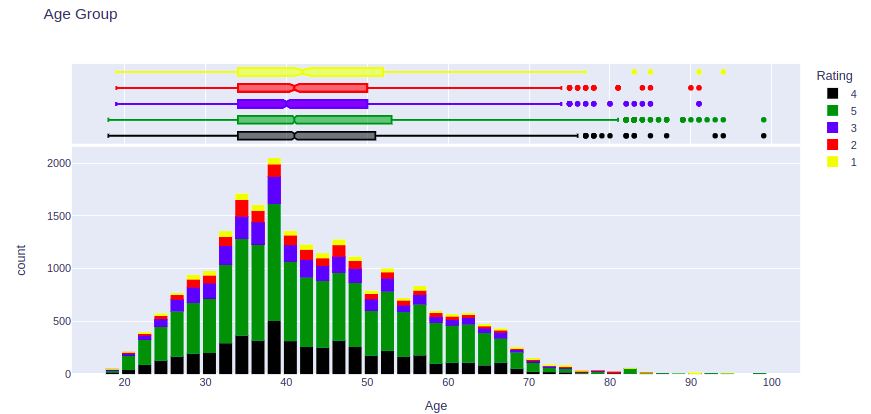
==================================================================================================

Total params: 258,698

Trainable params: 258,698

Non-trainable params: 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_



------------**Completed**--------------