MACHINE LEARNING
(FLOWERS RECOGNITION)

Summer Internship Report Submitted in partial fulfillment of the requirement for undergraduate degree of

Bachelor of Technology

In

Computer Science Engineering

By

KURAKULA ESWAR SAI SUHAS REDDY

221710304030

Under the Guidance of

Assistant Professor

Department Of Computer Science and Engineering, GITAM School of Technology GITAM (Deemed to be University), Hyderabad-502329

July 2020

<u>DECLARATION</u>

I submit this industrial training work entitled “FLOWERS RECOGNITION” to GITAM (Deemed To Be University), Hyderabad in partial fulfilment of the requirements for the award of the degree of “Bachelor of Technology” in “Computer Science Engineering”. I declare that it was carried out independently by me under the guidance of Mr. Asst. Professor, GITAM (Deemed To Be University), Hyderabad, India.

The results embodied in this report have not been submitted to any other University or Institute for the award of any degree or diploma.

Place: HYDERABAD **KURAKULA ESWAR SAI SUHAS REDDY**

Date: **221710304030**

CERTIFICATE

This is to certify that the Industrial Training Report entitled "FLOWERS RECOGNITION" is being submitted by Kurakula Eswar Sai Suhas Reddy (221710304030) in partial fulfilment of the requirement for the award of Bachelor of Technology in Computer Science Engineering at GITAM (Deemed To Be University), Hyderabad during the academic year 2019-20.

It is faithful record work carried out by her at the Computer Science Engineering Department, GITAM University Hyderabad Campus
under my guidance and supervision.

Mr.	**Dr. S.Phani Kumar**
Assistant Professor	Professor and HOD
Department of CSE	Department of CSE

ACKNOWLEDGEMENT

Apart from my effort, the success of this internship largely depends on the encouragement and guidance of many others. I take this opportunity to express my gratitude to the people who have helped me in the successful competition of this internship.

I would like to thank respected **Dr. N. Siva Prasad**, Pro Vice Chancellor, GITAM Hyderabad and **Dr. N.Seetharamaiaha,** Principal, GITAM Hyderabad.

I would like to thank respected **Dr. S.Phani Kumar**, Head of the Department of Computer Science Engineering for giving me such a wonderful opportunity to expand my knowledge for my own branch and giving me guidelines to present a internship report. It helped me a lot to realize of what we study for.

I would like to thank the respected faculties **Mr.** who helped me to make this internship a successful accomplishment.

I would also like to thank my friends who helped me to make my work more organized and well-stacked till the end.

KURAKULA ESWAR SAI SUHAS REDDY

221710304030

ABSTRACT

In general, there are many different types of flowers. Presently, new flower species are being discovered on an ongoing basis, even though they are very similar in color, shape, texture, petals and features. Ordinary persons with limited Botanical knowledge would not know the exact type of a flower just by looking at it. In order to classify a flower correctly, it is important to provide enough important information including the flower’s name. In this report a Flower Identification System (FIS) is presented, one that that classifies and provides the information from an input random flower image.

The information includes the flower’s scientific name, botanical information and so on. The system was developed using Python and Convolution Neural Networks method. Prior to flower identification, Grab Cut was used to segment the background and the foreground from the input image. The identification of flower name from the input image is done based on RGB Histogram data. Given a database of 5 different types of flowers. The pictures are divided into five classes:chamomile,tulip,rose,sunflower,dandelion.
For each class there are about 800 photos. Photos are not high resolution, about 320x240 pixels. Photos are not reduced to a single size, they have different proportions

KURAKULA ESWAR SAI SUHAS REDDY

221710304030

Table of Contents

LIST OF FIGURES

1.MACHINE LEARNING

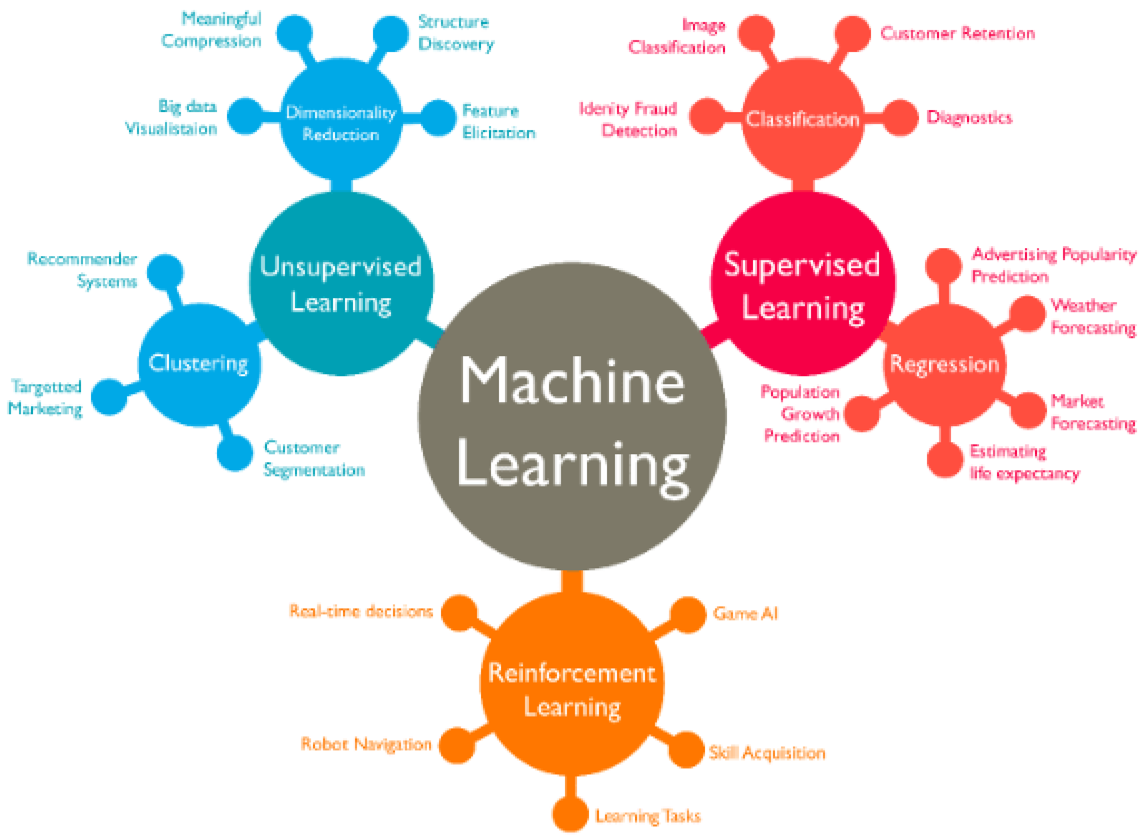
1.1 INTRODUCTION:

Machine learning (ML) is the study of computer algorithms that improve automatically through experience. It is seen as a subset of artificial intelligence. Machine learning algorithms build a mathematical model based on sample data, known as training data in order to make predictions or decisions without being explicitly programmed to do so. Machine learning algorithms are used in a wide variety of applications, such as email filtering and computer vision, where it is difficult or infeasible to develop conventional algorithms to perform the needed tasks.

1.2 IMPORTANCE OF MACHINE LEARNING:

Machine learning has several very practical applications that drive the kind of real business results such as time and money savings that have the potential to dramatically impact the future of your organization. At Interactions in particular, we see tremendous impact occurring within the customer care industry, whereby machine learning is allowing people to get things done more quickly and efficiently. Through Virtual Assistant solutions, machine learning automates tasks that would otherwise need to be performed by a live agent such as changing a password or checking an account balance.

This frees up valuable agent time that can be used to focus on the kind of customer care that humans perform best: high touch, complicated decision-making that is not as easily handled by a machine. At Interactions, we further improve the process by eliminating the decision of whether a request should be sent to a human or a machine unique Adaptive Understanding technology, the machine learns to be aware of its limitations, and bail out to humans when it has a low confidence in providing the correct solution.



Source: https://wordstream-files-prod.s3.amazonaws.com/s3fs-public/machine-learning.png

Figure 1.2.1. Machine Learning

1.3 APPLICATIONS OF MACHINE LEARNING:

The value of machine learning technology has been recognized by companies across several industries that deal with huge volumes of data. By leveraging insights obtained from this data, companies are able work in an efficient manner to control costs as well as get an edge over their competitors. This is how some sectors / domains are implementing machine learning -\

- Financial Services

Companies in the financial sector are able to identify key insights in financial data as well as prevent any occurrences of financial fraud, with the help of machine learning technology. The technology is also used to identify opportunities for investments and trade. Usage of cyber surveillance helps in identifying those individuals or institutions which are prone to financial risk, and take necessary actions in time to prevent fraud.

- Marketing and Sales Companies are using machine learning technology to analyze the purchase history of their customers and make personalized product recommendations for their next purchase. This ability to capture, analyze, and use customer data to provide a personalized shopping experience is the future of sales and marketing.
- Government

  Government agencies like utilities and public safety have a specific need for Machine Learning, as they have multiple data sources, which can be mined for identifying useful patterns and insights. For example sensor data can be analyzed to identify ways to minimize costs and increase efficiency. Furthermore, ML can also be used to minimize identity thefts and detect fraud.
- Healthcare

  With the advent of wearable sensors and devices that use data to access health of a patient in real time, ML is becoming a fast-growing trend in healthcare. Sensors in wearable provide real-time patient information, such as overall health condition, heartbeat, blood pressure and other vital parameters. Doctors and medical experts can use this information to analyze the health condition of an individual, draw a pattern from the patient history, and predict the occurrence of any ailments in the future. The technology also empowers medical experts to analyze data to identify trends that facilitate better diagnoses and treatment.
- Transportation

  Based on the travel history and pattern of traveling across various routes, machine learning can help transportation companies predict potential problems that could arise on certain routes, and accordingly advise their customers to opt for a different route. Transportation firms and delivery organizations are increasingly using machine learning technology to carry out data analysis and data modeling to make informed decisions and help their customers make smart decisions when they travel.

- Oil and Gas This is perhaps the industry that needs the application of machine learning the most. Right from analyzing underground minerals and finding new energy sources to streaming oil distribution, ML applications for this industry are vast and are still expanding.

1.4 TYPES OF MACHINE LEARNING ALGORITHMS:

There some variations of how to define the types of Machine Learning Algorithms but commonly they can be divided into categories according to their purpose and the main categories are the following:

1.4.1 SUPERVISED LEARNING

- I like to think of supervised learning with the concept of function approximation, where basically we train an algorithm and in the end of the process we pick the function that best describes the input data, the one that for a given X makes the best estimation of y (X -> y). Most of the time we are not able to figure out the true function that always make the correct predictions and other reason is that the algorithm rely upon an assumption made by humans about how the computer should learn and this assumptions introduce a bias, Bias is topic I'll explain in another post.
- Here the human experts acts as the teacher where we feed the computer with training data containing the input/predictors and we show it the correct answers (output) and from the data the computer should be able to learn the patterns.
- Supervised learning algorithms try to model relationships and dependencies between the target prediction output and the input features such that we can predict the output values for new data based on those relationships which it learned from the previous data sets

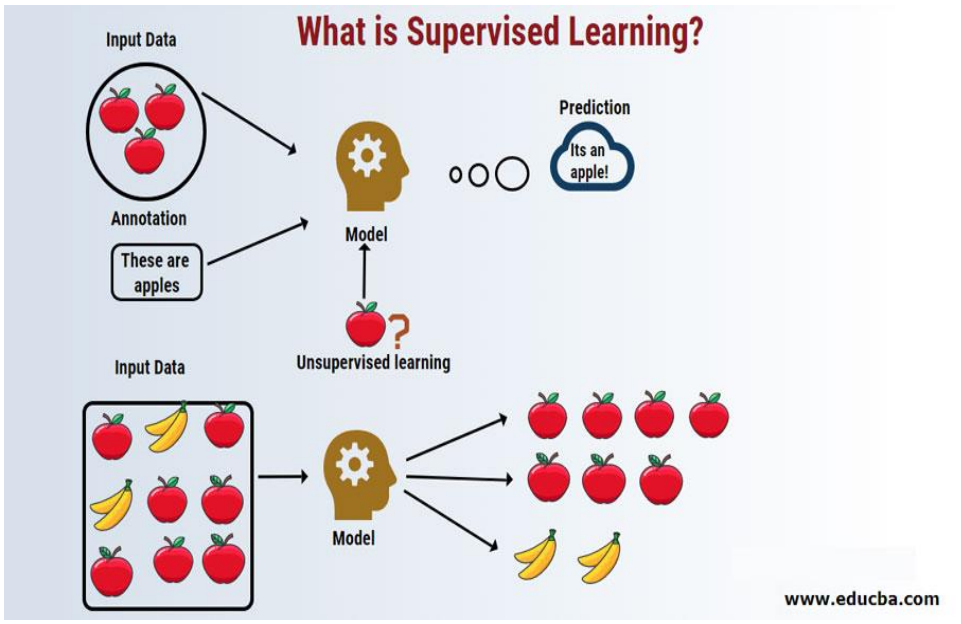


Figure 1.4.1.1: Supervised Learning

1.4.2 UNSUPERVISED LEARNING:

- The computer is trained with unlabeled data.
- Here there's no teacher at all, actually the computer might be able to teach you new things after it learns patterns in data, these algorithms a particularly useful in cases where the human expert doesn't know what to look for in the data.
- are the family of machine learning algorithms which are mainly used in pattern detection and descriptive modeling. However, there are no output categories or labels here based on which the algorithm can try to model relationships. These algorithms try to use techniques on the input data to mine for rules, detect patterns, and summarize and group the data points which help in deriving meaningful insights and describe the data better to the users.

1.4.3 SEMI SUPERVISED LEARNING:

In the previous two types, either there are no labels for all the observation in the dataset or labels are present for all the observations. Semi-supervised learning falls in between these two. In many practical situations, the cost to label is quite high, since it requires skilled human experts to do that. So, in the absence of labels in the majority of the observations but present in few, semi-supervised algorithms are the best candidates for the model building. These methods exploit the idea that even though the group memberships of the unlabeled data are unknown, this data carries important information about the group parameters.

1.4.4 Reinforcement Learning:

method aims at using observations gathered from the interaction with the environment to take actions that would maximize the reward or minimize the risk. Reinforcement learning algorithm (called the agent) continuously learns from the environment in an iterative fashion. In the process, the agent learns from its experiences of the environment until it explores the full range of possible states. Reinforcement Learning is a type of Machine Learning, and thereby also a branchofArtificialIntelligence.

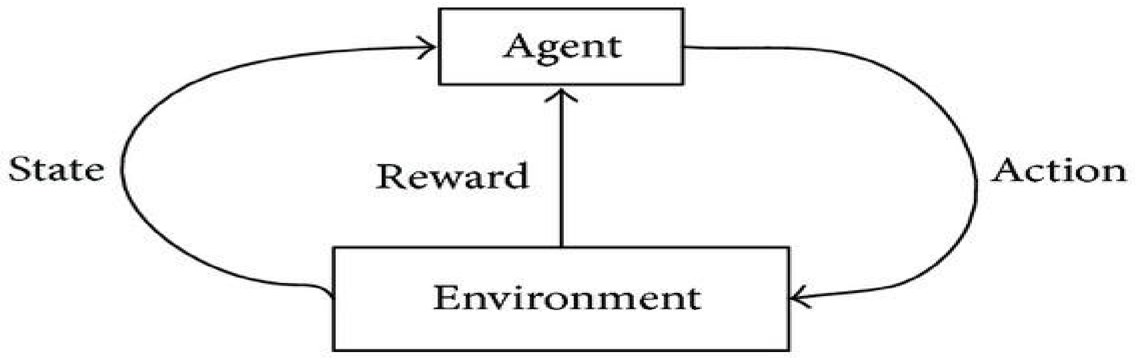


Figure 1.4.4.1: Reinforcement Learning

2. PYTHON

2.1 INTRODUCTION TO PYTHON:

Python is an interpreted, high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991, Python's design philosophy emphasizes code readability with its notable use of significant whitespace. Its language constructs and object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects

Python is dynamically typed and garbage-collected. It supports multiple programming paradigms, including structured (particularly, procedural), object-oriented, and functional programming. Python is often described as a "batteries included" language due to its comprehensive standard library

2.2 HISTORY OF PYTHON:

Python was conceived in the late 1980s by Guido van Rossum at Centrum Wiskunde & Informatica (CWI) in the Netherlands as a successor to the ABC language (itself inspired by SETL), capable of exception handling and interfacing with the Amoeba operating system. Python 2.0 was released on 16 October 2000 with many major new features, including a cycle-detecting garbage collector and support for Unicode. Python 3.0 was released on 3 December 2008. It was a major revision of the language that is not completely backward-compatible. Many of its major features were backported to Python 2.6.x and 2.7.x version series. Releases of Python 3 include the 2to3 utility, which automates (at least partially) the translation of Python 2 code to Python 3.Python 2.7's end-of-life date was initially set 2015 then postponed to 2020 out of concern that a large body of existing code could not easily be forward-ported to Python 3.

2.3 FEATURES OF PYTHON:

Python provides many useful features which make it popular and valuable from the other programming languages. It supports object-oriented programming, procedural programming approaches and provides dynamic memory allocation. We have listed below a few essential features.

Easy to Learn and Use

Python is easy to learn as compared to other programming languages. Its syntax is straightforward and much the same as the English language. There is no use of the semicolon or curly-bracket, the indentation defines the code block. It is the recommended programming language for beginners.

Expressive Language

Python can perform complex tasks using a few lines of code. A simple example, the hello world program you simply type print("Hello World"). It will take only one line to execute, while Java or C takes multiple lines. the Python community. The open-source means, "Anyone can download its source code without paying any penny."

Object-Oriented Language

Python supports object-oriented language and concepts of classes and objects come into existence. It supports inheritance, polymorphism, and encapsulation, etc. The object-oriented procedure helps to programmer to write reusable code and develop applications in less code.

Extensible

It implies that other languages such as C/C++ can be used to compile the code and thus it can be used further in our Python code. It converts the program into byte code, and any platform can use that byte code.

Large Standard Library

It provides a vast range of libraries for the various fields such as machine learning, web developer, and also for the scripting. There are various machine learning libraries, such as Tensor flow, Pandas, NumPy, Kera’s, and Pytorch, etc. Django, flask, pyramids are the popular framework for Python web development.

GUI Programming Support

Graphical User Interface is used for the developing Desktop application. PyQT5, Tkinter, Kivy are the libraries which are used for developing the web application.

Integrated

It can be easily integrated with languages like C, C++, and JAVA, etc. Python runs code line by line like C, C++ Java. It makes easy to debug the code.

Embeddable

The code of the other programming language can use in the Python source code. We can use Python source code in another programming language as well. It can embed other language into our code.

Dynamic Memory Allocation

In Python, we don't need to specify the data-type of the variable. When we assign some value to the variable, it automatically allocates the memory to the variable at run time. Suppose we are assigned integer value 15 to **x,** then we don't need to write **int x = 15.** Just write x = 15.

Interpreted Language

Python is an interpreted language; it means the Python program is executed one line at a time. The advantage of being interpreted language, it makes debugging easy and portable.

Cross-platform Language

Python can run equally on different platforms such as Windows, Linux, UNIX, and Macintosh, etc. So, we can say that Python is a portable language. It enables programmers to develop the software for several competing platforms by writing a program only once.

Free and Open Source

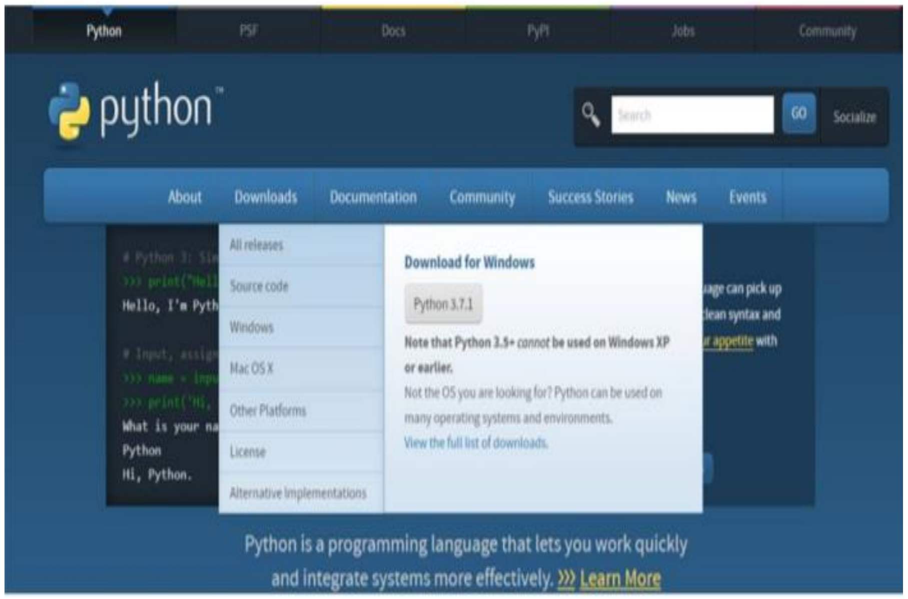
Python is freely available for everyone. It is freely available on its official website www.python.org. It has a large community across the world that is dedicatedly working towards make new python modules and functions. Anyone can contribute to the Python community. The open-source means, "Anyone can download its source code without paying any penny."

2.4 HOW TO SETUP PYTHON:

- Python is available on a wide variety of platforms including Linux and Mac OS X. Let's understand how to set up our Python environment.
- The most up-to-date and current source code, binaries, documentation, news, etc., is available on the official website of Python.

2.4.1 Installation (using python IDLE):

- Installing python is generally easy, and nowadays many Linux and Mac OS distributions include a recent python.
- Download python from www.python.org
- When the download is completed, double click the file and follow the instructions to install it.
- When python is installed, a program called IDLE is also installed along with it. It provides a graphical user interface to work with python



Note that Python 3.5+ *cannot* be used on Windows XP or earlier.

Not the OS you are looking for? Python can be used on many operating systems and environments.

View the full list of downloads.

Python is a programming language that lets you work quickly and integrate systems more effectively. >>> Learn More

Figure 2.4.1.1 : Python download

2.4.2 Installation (using Anaconda):

- Python programs are also executed using Anaconda.
- Anaconda is a free open source distribution of python for large scale data processing, predictive analytics and scientific computing.
- Conda is a package manager quickly installs and manages packages.
- In WINDOWS:
- Step 1: Open Anaconda.com/downloads in a web browser.
- Step 2: Download python 3.4 version for (32-bitgraphic installer/64 -bit graphic installer)
- Step 3: select installation type (all users)
- Step 4: Select path (i.e. add anaconda to path & register anaconda as default python 3.4) next click install and next click finish
- Step 5: Open jupyter notebook (it opens in default browser)

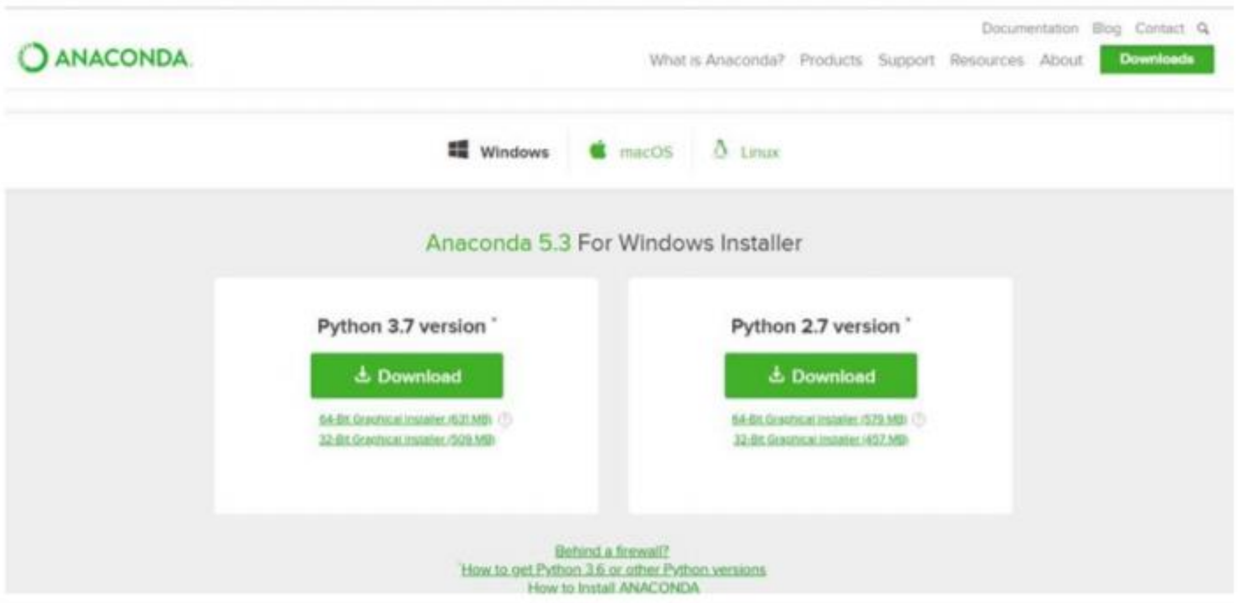

ANACONDA
Documentation Blog Contact
What is Anaconda? Products Support Resources About Downloads
Windows macOS Linux
Anaconda 5.3 For Windows Installer
Python 3.7 version
Download
64-Bit Graphical Installer (631 MB)
32-Bit Graphical Installer (509 MB)
Python 2.7 version
Download
64-Bit Graphical Installer (579 MB)
32-Bit Graphical Installer (457 MB)
Behind a firewall?
How to get Python 3.6 or other Python versions
How to Install ANACONDA

Figure 2.4.2.1 : Anaconda download

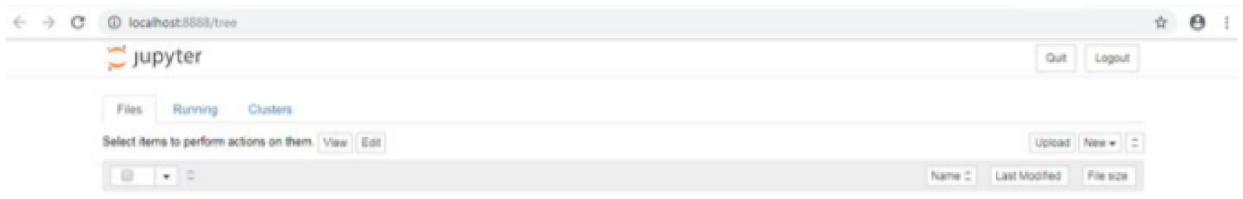

localhost:8888/tree
jupyter
Quit Logout
Files Running Clusters
Select items to perform actions on them. View Edit
Upload New
Name Last Modified File size

Figure 2.4.2.2: Jupyter notebook

2.5 PYTHON VARIABLE TYPES:

- Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in memory.
- Variables are nothing but reserved memory locations to store values.
- Based on the data type of a variable, the interpreter allocates memory and decides what can be stored in the reserved memory.
- Python variables do not need explicit declaration to reserve memory space. The declaration happens automatically when you assign a value to a variable.
- Python has various standard data types that are used to define the operations possible on them and the storage method for each of them.
  - Python has five standard data types –

- Numbers
- Strings
- Lists
- Tuples
- Dictionary

2.5.1 Python Numbers:

- Number data types store numeric values. Number objects are created when you assign a value to them.
- Python supports four different numerical types − int (signed integers) long (long integers, they can also be represented in octal and hexadecimal) float (floating point real values) complex (complex numbers).

2.5.2 Python Strings:

- Strings in Python are identified as a contiguous set of characters represented in the quotation marks.
- Python allows for either pairs of single or double quotes.
- Subsets of strings can be taken using the slice operator ([] and [:] sss) with indexes starting at 0 in the beginning of the string and working their way from -1 at the end.
- The plus (+) sign is the string concatenation operator and the asterisk (*) is the repetition operator.

2.5.3 Python Lists:

- Lists are the most versatile of Python's compound data types.
- A list contains items separated by commas and enclosed within square brackets ([]).
- To some extent, lists are similar to arrays in C. One difference between them is that all the items belonging to a list can be of different data type.
- The values stored in a list can be accessed using the slice operator ([] and [:]) with indexes starting at 0 in the beginning of the list and working their way to end -1.
- The plus (+) sign is the list concatenation operator, and the asterisk (*) is the repetition operator.

2.5.4 Python Tuples:

- A tuple is another sequence data type that is similar to the list.
- A tuple consists of a number of values separated by commas. Unlike lists, however, tuples are enclosed within parentheses.
- The main differences between lists and tuples are: Lists are enclosed in brackets ([]) and their elements and size can be changed, while tuples are enclosed in parentheses (()) and cannot be updated.
- Tuples can be thought of as read-only lists.
- For example − Tuples are fixed size in nature whereas lists are dynamic. In other words, a tuple is immutable whereas a list is mutable. You can't add elements to a tuple. Tuples have no append or extend method. You can't remove elements from a tuple. Tuples have no remove or pop method.

2.5.5 Python Dictionary:

- Python's dictionaries are kind of hash table type. They work like associative arrays or hashes found in Perl and consist of key-value pairs. A dictionary key can be almost any Python type, but are usually numbers or strings. Values, on the other hand, can be any arbitrary Python object.
- Dictionaries are enclosed by curly braces ({}) and values can be assigned and accessed using square braces ([]).
- You can use numbers to "index" into a list, meaning you can use numbers to find out what's in lists. You should know this about lists by now, but make sure you understand that you can only use numbers to get items out of a list.
- What a dict does is let you use anything, not just numbers. Yes, a dict associates one thing to another, no matter what it is.

2.6 PYTHON FUNCTION:

2.6.1 Defining a Function:

You can define functions to provide the required functionality. Here are simple rules to define a function in Python. Function blocks begin with the keyword def followed by the function name and parentheses (i.e. ()).

Any input parameters or arguments should be placed within these parentheses. You can also define parameters inside these parentheses. The code block within every function starts with a colon (:) and is indented. The statement returns [expression] exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as return None.

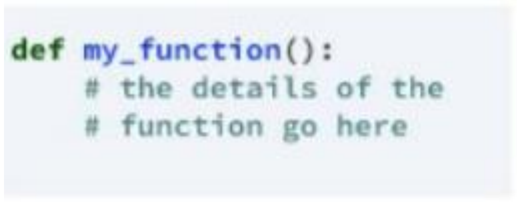
2.6.2 Calling a Function:

Defining a function only gives it a name, specifies the parameters that are to be included in the function and structures the blocks of code. Once the basic structure of a function is finalized, you can execute it by calling it from another function or directly from the Python prompt.

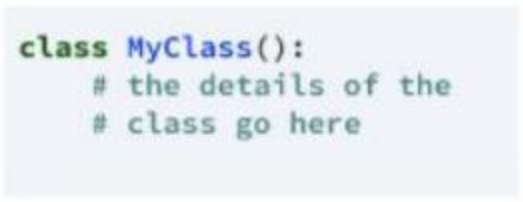
2.7 PYTHON USING OOP'S CONCEPTS:

2.7.1 Class:

- Class: A user-defined prototype for an object that defines a set of attributes that characterize any object of the class. The attributes are data members (class variables and instance variables) and methods, accessed via dot notation.
- Class variable: A variable that is shared by all instances of a class. Class variables are defined within a class but outside any of the class's methods. Class variables are not used as frequently as instance variables are.
- Data member: A class variable or instance variable that holds data associated with a class and its objects.
- Instance variable: A variable that is defined inside a method and belongs only to the current instance of a class.
- Defining a Class: o We define a class in a very similar way how we define a function. o Just like a function; we use parentheses and a colon after the class name (i.e. () :) when we define a class. Similarly, the body of our class is indented like a function body is.



```
def my_function():
    # the details of the
    # function go here
```



```
class MyClass():
    # the details of the
    # class go here
```

Figure 2.7.1.1: Defining a Class

2.7.2 __init__ method in Class:

- The init method — also called a constructor — is a special method that runs when an instance is created so we can perform any tasks to set up the instance.
- The init method has a special name that starts and ends with two underscores: _init__ ()

3.CASE STUDY

3.1 PROBLEM STATEMENT:

Recognition of flowers in environments such as forests and mountains is necessary to know whether they are extinct or not. While search engines assist in searching for a flower, they lack the robustness because of the intra-class variation among millions of flower species. The primary objective of the project is to identify the category of the given image.

3.2 DATA SET:

This dataset contains 4242 images of flowers. The data collection is based on the data Flickr, google images, Yandex images. You can use this dataset to recognize flowers from the photo.

The pictures are divided into five classes: daisy, tulip, rose, sunflower, dandelion. For each class there are about 800 photos. Photos are not high resolution, about 320x240 pixels. Photos are not reduced to a single size; they have different proportions. The data collection is based on scraped data from Flickr, google images, and Yandex images

.3.3 OBJECTIVE OF THE CASE STUDY:

- The primary objective of the project is to identify the category of the given image.
- Splitting of data into train and validation sets and all the images in the data should be of

  same size i.e. is 150*150
- Improving the accuracy of validation and training Accuracy

4.MODEL BUILDING

4.1 PREPROCESSING OF THE DATA:

Preprocessing the data actually involves the following steps.

4.1.1 GETTING THE DATASET:

We can get the data set from the database or we can get the data from the google or Yandex

4.1.2 IMPORTING THE LIBRARIES:

We have imported the libraries as per the requirement of the algorithm.

The **OS module in python** provides functions for interacting with the operating system. **OS**, comes under **Python's** standard utility modules. This **module** provides a portable way of using operating system dependent functionality.

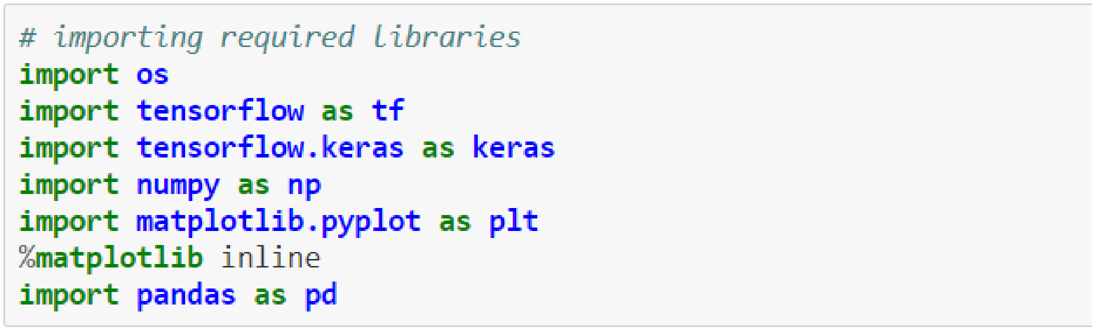
TensorFlow is a **Python** library for fast numerical computing created and released by Google. It is a foundation library that can be used to create Deep Learning models directly or by using wrapper libraries that simplify the process built on top of **TensorFlow.**

Keras is an open-source neural-network library written in Python. It is capable of running on top of TensorFlow, Microsoft Cognitive Toolkit, R, Theano, or Plaid ML. Designed to enable fast experimentation with deep neural networks, it focuses on being user-friendly, modular, and extensible. It was developed as part of the research effort of project ONEIROS (Open-ended Neuro-Electronic Intelligent Robot Operating System),[5] and its primary author and maintainer is François Cholet, a Google engineer. Cholet also is the author of the Exception deep neural network model.

NumPy is a **python** library used for working with arrays. It also has functions for working in domain of linear algebra, Fourier transform, and matrices. **NumPy** was created in 2005 by Travis Oliphant. It is an open source project and you can use it freely.

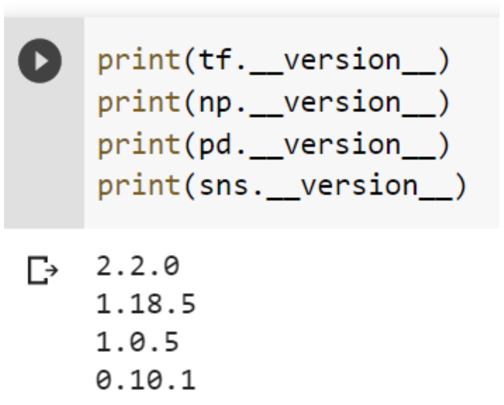
Matplotlib is a plotting library for the **Python** programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, python, Qt, or GTK+. SciPy makes use of **Matplotlib**.

Pandas is a high-level data manipulation tool developed by Wes McKinney. It is built on the NumPy package and its key data structure is called the Data Frame. Data Frames allow you to store and manipulate tabular data in rows of observations and columns of variables.



```
# importing required libraries
import os
import tensorflow as tf
import tensorflow.keras as keras
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import pandas as pd
```

Figure 4.1.2.1: Importing Libraries



```
print(tf.__version__)
print(np.__version__)
print(pd.__version__)
print(sns.__version__)
```

```
2.2.0
1.18.5
1.0.5
0.10.1
```

Figure 4.1.2.2: Versions of libraries

4.1.3 IMPORTING THE DATA-SET:

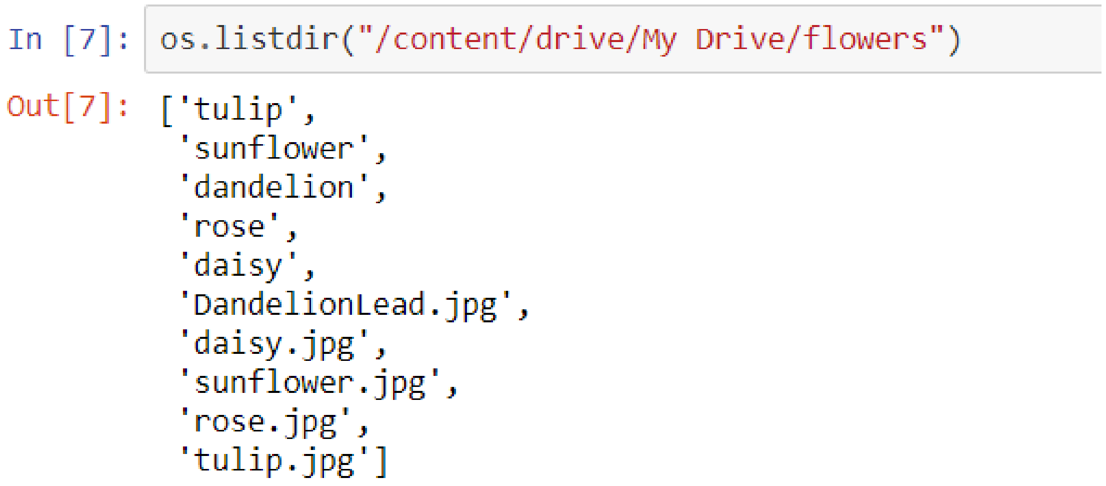
Python method **list dir** () returns a list containing the names of the entries in the directory given by path. The list is in arbitrary order. It does not include the special entries '. ' and '..' even if they are present in the directory.

Syntax: os. listdir(path)

Parameters*:* path (optional): path of the directory

Return Type: This method returns the list of all files and directories in the specified path. The return type of this method is *list*.

Here In the code the categories and the files which are present in the dataset are shown using the list dir function and also no of files or images present in each category is also shown. Here you an see the categories and files which are present in the dataset. There are five categories in the dataset and five sample images which are present in the dataset are also shown here.

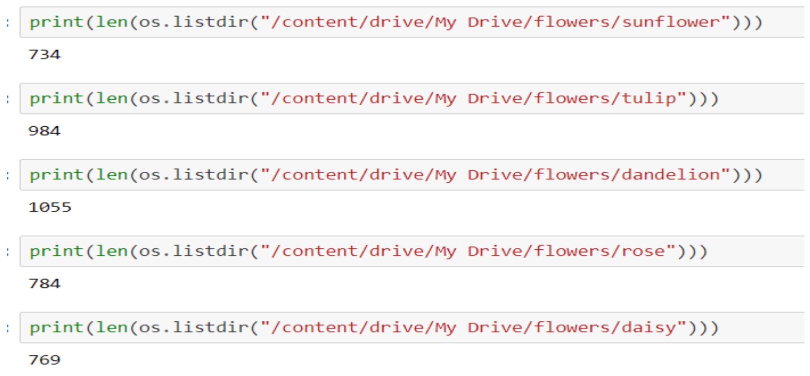


```
In [7]: os.listdir("/content/drive/My Drive/flowers")

Out[7]: ['tulip',
         'sunflower',
         'dandelion',
         'rose',
         'daisy',
         'DandelionLead.jpg',
         'daisy.jpg',
         'sunflower.jpg',
         'rose.jpg',
         'tulip.jpg']
```

Figure 4.1.3.1: Reading the dataset

4.1.4 LISTING THE LENGTH OF CATEGORIES IN THE DATASET:

The length of each category is also using list dir function where you can see there are :734 images in sunflower,784 images in rose, 984 images in tulip,769 images in daisy,1055 images in dandelion

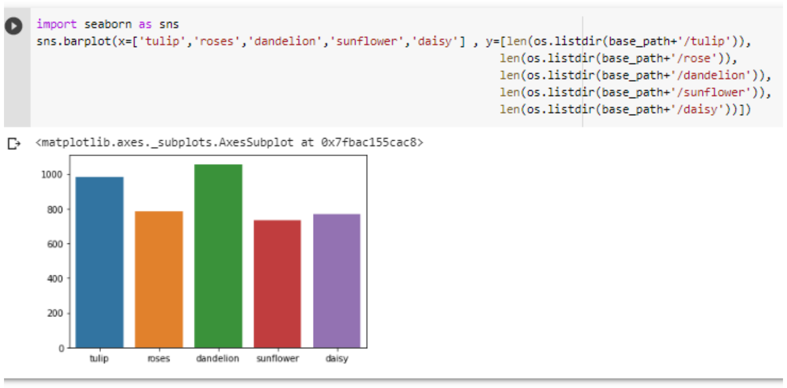


```
print(len(os.listdir("/content/drive/My Drive/flowers/sunflower")))
734
print(len(os.listdir("/content/drive/My Drive/flowers/tulip")))
984
print(len(os.listdir("/content/drive/My Drive/flowers/dandelion")))
1055
print(len(os.listdir("/content/drive/My Drive/flowers/rose")))
784
print(len(os.listdir("/content/drive/My Drive/flowers/daisy")))
769
```

Figure 4.1.4.1: Length of categories

4.1.5 VISUALIZATION OF IMAGE CATEGORIES:

seaborn: statistical data visualization. **Seaborn** is a **Python** data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics. For a brief introduction to the ideas behind the library, you can read the introductory notes.



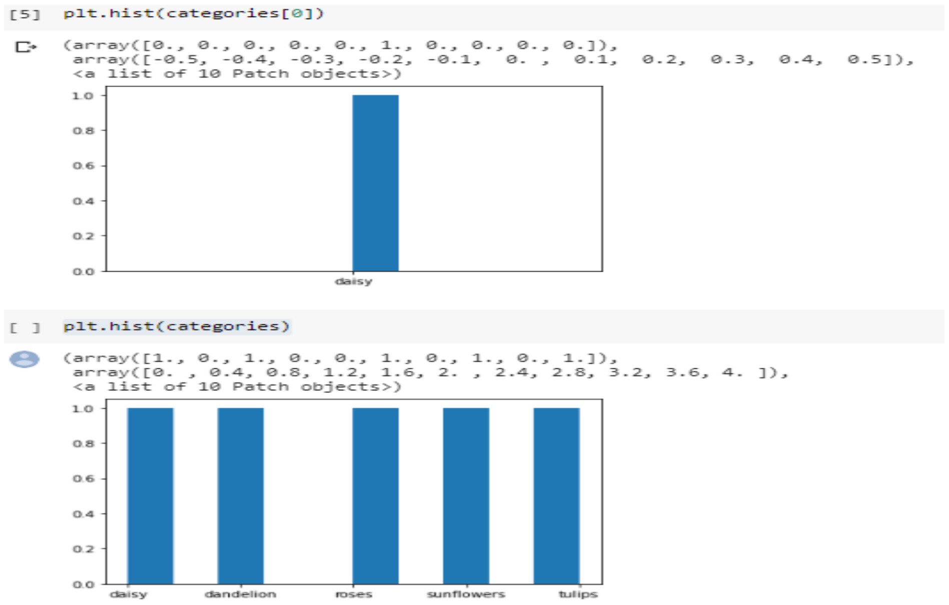
```
import seaborn as sns
sns.barplot(x=['tulip','roses','dandelion','sunflower','daisy'] , y=[len(os.listdir(base_path+'/tulip')),
                                                                      len(os.listdir(base_path+'/rose')),
                                                                      len(os.listdir(base_path+'/dandelion')),
                                                                      len(os.listdir(base_path+'/sunflower')),
                                                                      len(os.listdir(base_path+'/daisy'))])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fbac155cac8>
```

Figure 4.1.5.1: visualization of image categories

Here using seaborn library as sns length of the category in y axis and name of the category in the x axis. There are 984 images in tulip category ,784 images in roses category ,769 images in daisy category, 1055 images in dandelion category ,734 images in rose category. **histogram** representation of color information helps the **CNN** to exploit color information in the original image. This means that we can minimize the size of the basic feature detectors (i.e. layer 1 of the **CNN**). The proposed architecture is introduced in the following section. Color histograms are widely used to compare images despite the simplicity of this method. It has been proven to have good performance on image indexing with relatively small datasets.

Color histograms are trivial to compute and tend to be robust against small changes to camera viewpoint, which makes them a good compact image descriptor. It was also reported in that the performance of a histogram-based classifier was improved when the higher-level classifier was a support vector machine. However, when applied to large dataset, histogram-based classifiers tend to give poor performance because of high variances within the same category. It is also observed that images with different labels may share similar histograms. In this work, we propose a novel architecture that combines the histogram-based classification method with CNN. The histogram representation of color information helps the CNN to exploit color information in the original image.



```
[5]  plt.hist(categories[0])

     (array([0., 0., 0., 0., 0., 1., 0., 0., 0., 0.]),
      array([-0.5, -0.4, -0.3, -0.2, -0.1,  0. ,  0.1,  0.2,  0.3,  0.4,  0.5]),
      <a list of 10 Patch objects>)

[ ]  plt.hist(categories)

     (array([1., 0., 1., 0., 0., 1., 0., 1., 0., 1.]),
      array([0. , 0.4, 0.8, 1.2, 1.6, 2. , 2.4, 2.8, 3.2, 3.6, 4. ]),
      <a list of 10 Patch objects>)
```

Figure 4.1.5.2: visualization of image categories using histograms

4.2 PREPROCESSING THE IMAGE:

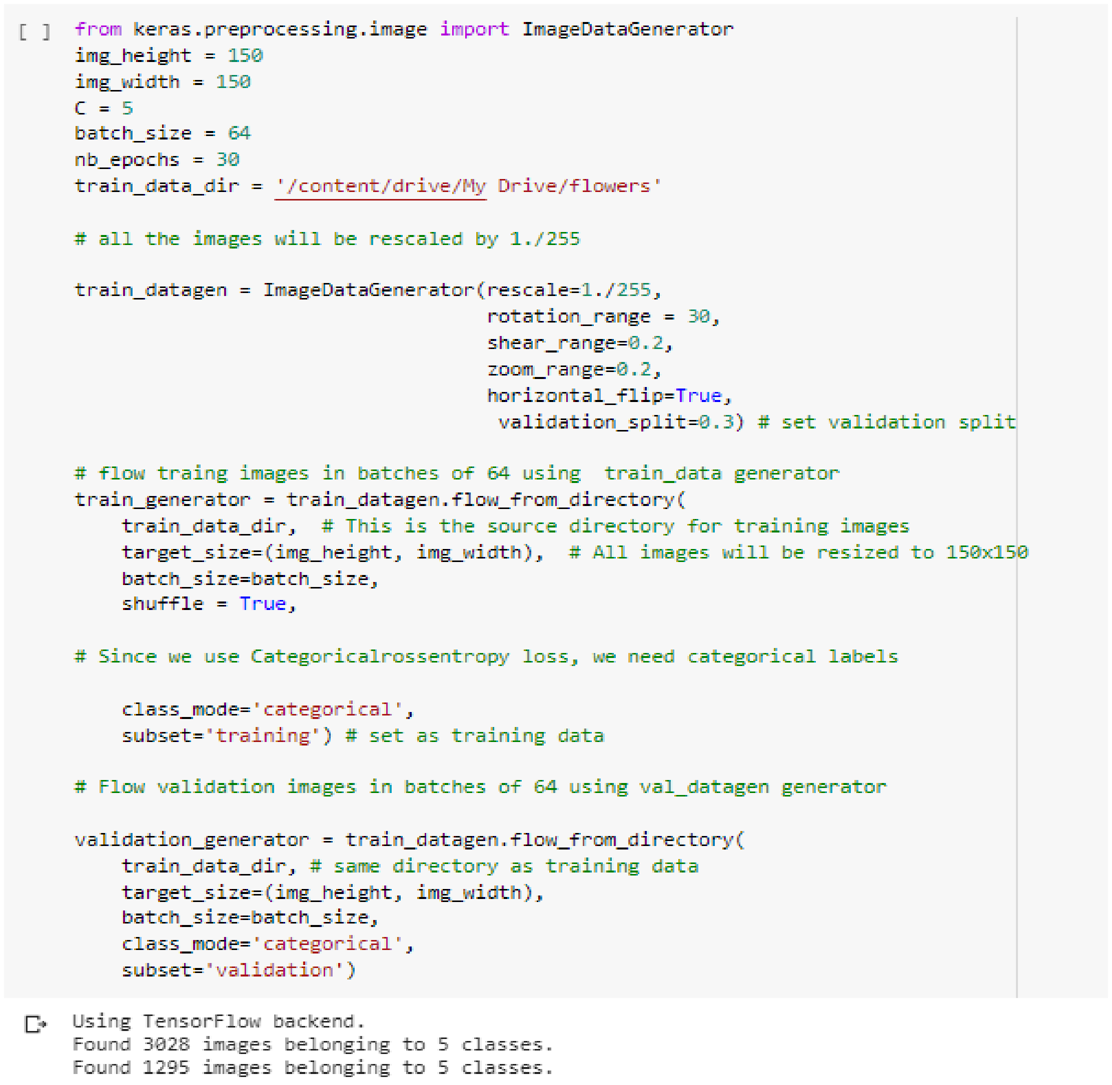
- rotation range is a value in degrees (0-180), a range within which to randomly rotate pictures
- width shift and height shift are ranges (as a fraction of total width or height) within which to randomly translate pictures vertically or horizontally
- rescale is a value by which we will multiply the data before any other processing. Our original images consist in RGB coefficients in the 0-255, but such values would be too high for our

models to process (given a typical learning rate), so we target values between 0 and 1 instead by scaling with a 1/255. factor.

- shear range is for randomly applying shearing transformations
- zoom range is for randomly zooming inside pictures
- horizontal flip is for randomly flipping half of the images horizontally --relevant when there are no assumptions of horizontal asymmetry (e.g. real-world pictures).
- **batch size**: Size of the batches of data. Default: 32.
- **image size**: Size to resize images to after they are read from disk. Defaults to (256, 256). Since the pipeline processes batches of images that must all have the same size, this must be provided.
- **shuffle**: Whether to shuffle the data. Default: True. If set to False, sorts the data in alphanumeric order.
- **seed**: Optional random seed for shuffling and transformations.
- **validation split**: Optional float between 0 and 1, fraction of data to reserve for validation.

The Keras Image Data Generator class is not an "additive" operation. It's not taking the original data, randomly transforming it, and then returning both the original data and transformed data. Instead, the Image Data Generator accepts the original data, randomly transforms it, and returns only the new, transformed data. Here using the Image Data Generator.

The images are converted into size 150X150 and the images are rotated by 30 degrees, horizontal flipping is done and the images are zoomed by 20% and the splitting of images is done into training and validation split in the ratio of 7:3.



```
[ ] from keras.preprocessing.image import ImageDataGenerator
    img_height = 150
    img_width = 150
    C = 5
    batch_size = 64
    nb_epochs = 30
    train_data_dir = '/content/drive/My Drive/flowers'

    # all the images will be rescaled by 1./255

    train_datagen = ImageDataGenerator(rescale=1./255,
                                       rotation_range = 30,
                                       shear_range=0.2,
                                       zoom_range=0.2,
                                       horizontal_flip=True,
                                        validation_split=0.3) # set validation split

    # flow traing images in batches of 64 using  train_data generator
    train_generator = train_datagen.flow_from_directory(
        train_data_dir,  # This is the source directory for training images
        target_size=(img_height, img_width),  # All images will be resized to 150x150
        batch_size=batch_size,
        shuffle = True,

    # Since we use Categoricalrossentropy loss, we need categorical labels

        class_mode='categorical',
        subset='training') # set as training data

    # Flow validation images in batches of 64 using val_datagen generator

    validation_generator = train_datagen.flow_from_directory(
        train_data_dir, # same directory as training data
        target_size=(img_height, img_width),
        batch_size=batch_size,
        class_mode='categorical',
        subset='validation')
```

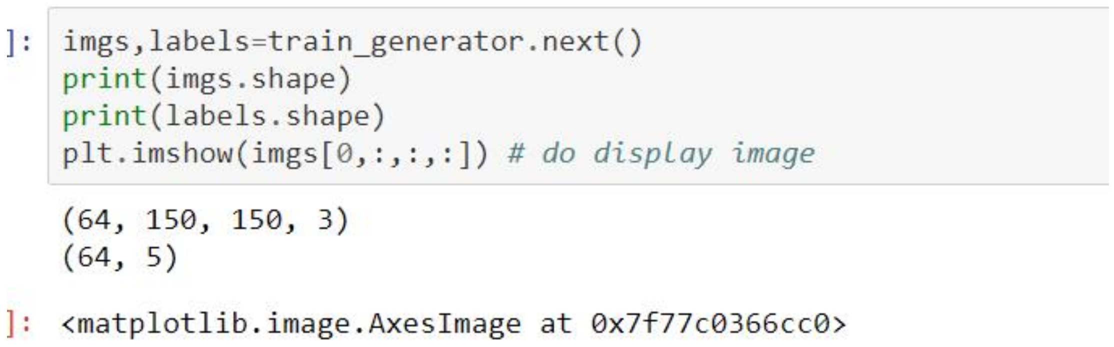
```
Using TensorFlow backend.
Found 3028 images belonging to 5 classes.
Found 1295 images belonging to 5 classes.
```

Figure 4.2.1: Image preprocessing

4.3 GENERATING THE LABELS OF IMAGES:

Here in the labeling is done because labeling is essential to send the images through convolution layer. Here you can see as the image is color it contains an array of 3d and labels are added to each image.

- **labels**: Either "inferred" (labels are generated from the directory structure), or a list/tuple of integer labels of the same size as the number of image files found in the directory. Labels should be sorted according to the alphanumeric order of the image file paths (obtained via os. Walk(directory) in Python).



```
]:  imgs,labels=train_generator.next()
    print(imgs.shape)
    print(labels.shape)
    plt.imshow(imgs[0,:,:,:]) # do display image

    (64, 150, 150, 3)
    (64, 5)

]:  <matplotlib.image.AxesImage at 0x7f77c0366cc0>
```

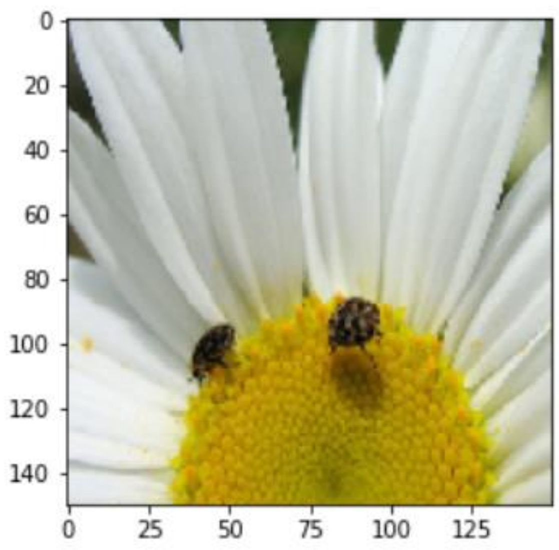
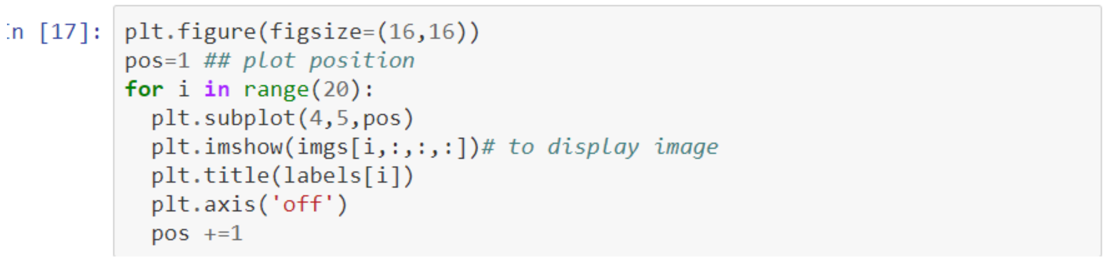


Figure 4.3.1: Label of a sample image

Some of the random images are shown below i.e,16 random images of size 150*150 are shown here were labels above the image represent the label array of each category. Where you can see array [1,0,0,0,0] represents the category of the daisy flowers and [0,0,0,0,1] Represents the category of flowers tulips similarly all the flowers images are shown with their categories.



```
In [17]: plt.figure(figsize=(16,16))
         pos=1 ## plot position
         for i in range(20):
           plt.subplot(4,5,pos)
           plt.imshow(imgs[i,:,:,:])# to display image
           plt.title(labels[i])
           plt.axis('off')
           pos +=1
```

Figure 4.3.2: Label generator

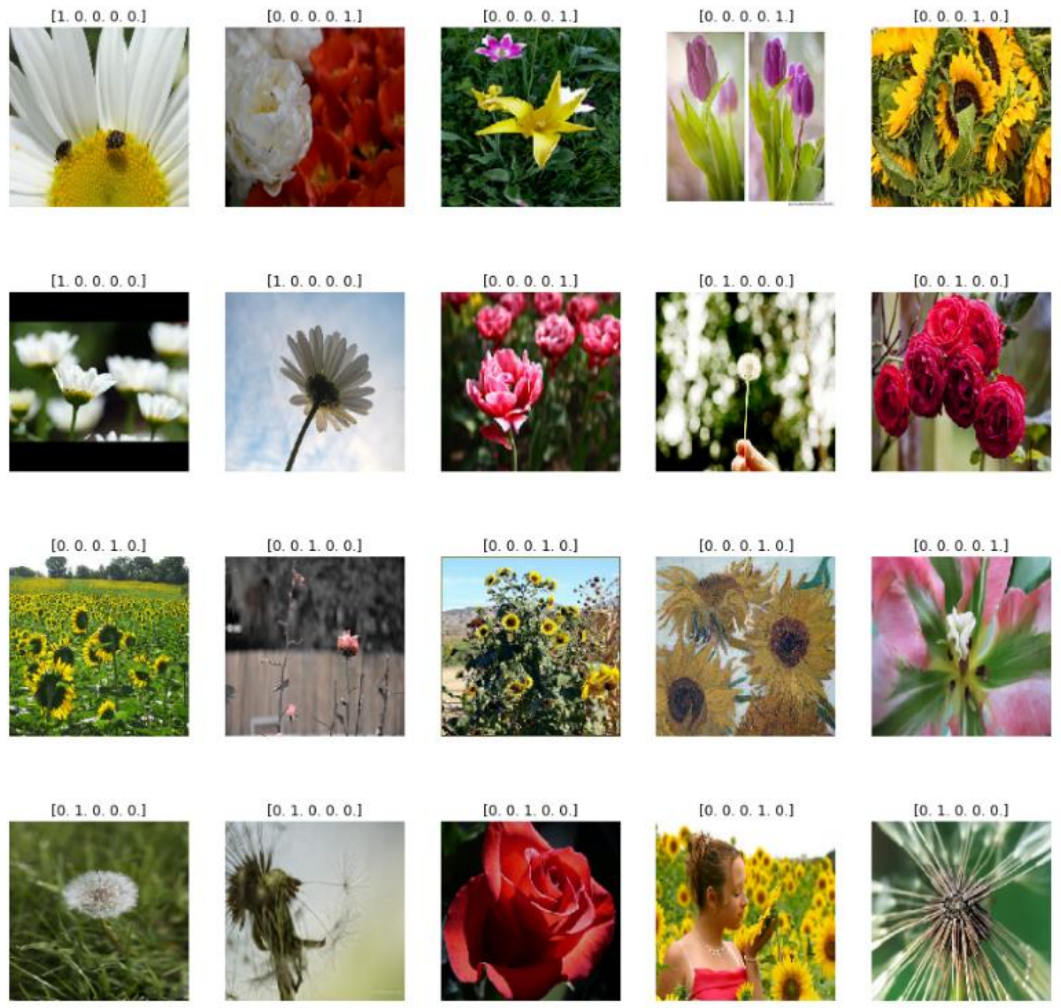


Figure 4.3.3: Labels of images belonging to various classes.

4.4 IMPLEMENTING THE CNN MODEL 1:

1. Convolution

A convolution extracts tiles of the input feature map, and applies filters to them to compute new features, producing an output feature map, or convolved feature (which may have a different size and depth than the input feature map).

2. ReLU

Following each convolution operation, the CNN applies a Rectified Linear Unit (ReLU) transformation to the convolved feature, in order to introduce nonlinearity into the model. The ReLU function, F(x)=max (0, x), returns x for all values of x > 0, and returns 0 for all values of x ≤ 0.

3. Pooling

After ReLU comes a pooling step, in which the CNN down samples the convolved feature (to save on processing time), reducing the number of dimensions of the feature map, while still preserving the most critical feature information. A common algorithm used for this process is called max pooling.

4. SoftMax

The SoftMax function is a function that turns a vector of K real values into a vector of K real values that sum to 1. The input values can be positive, negative, zero, or greater than one, but the SoftMax transforms them into values between 0 and 1, so that they can be interpreted as probabilities.

5.Strides

Strides, in general, define an overlap between applying operations. In the case of conv2d, it specifies what is the distance between consecutive applications of convolutional filters. The value of 1 in a specific dimension means that we apply the operator at every row/col, the value of 2 means every second, and so on.

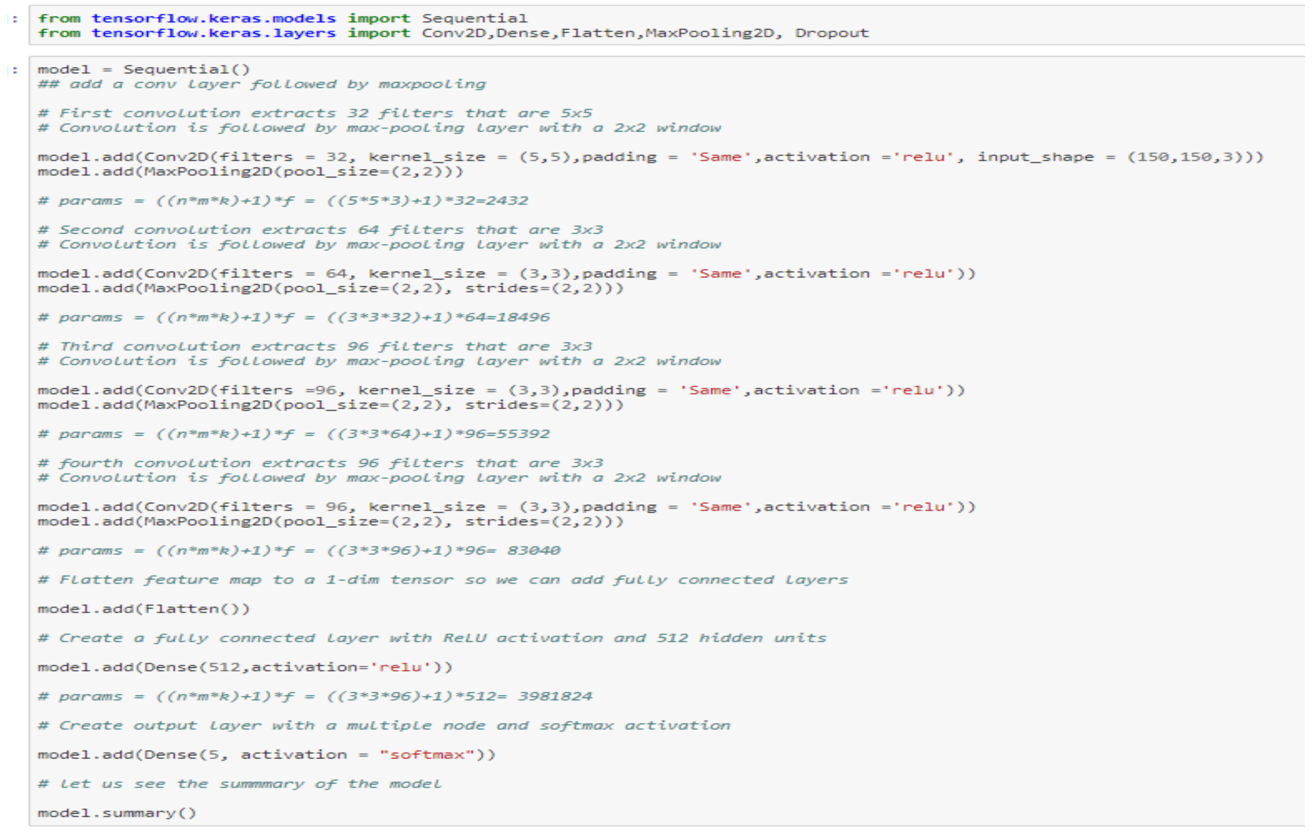
6.Padding

Padding is a term relevant to convolutional neural networks as it refers to the number of pixels added to an image when it is being processed by the kernel of a **CNN**. For example, if the **padding** in a **CNN** is set to zero, then every pixel value that is added will be of value zero.

A Sequential model is appropriate for a plain stack of layers where each layer has exactly one input tensor and one output tensor

Sequential model is not appropriate when:

- Your model has multiple inputs or multiple outputs
- Any of your layers has multiple inputs or multiple outputs
- You need to do layer sharing
- You want non-linear topology



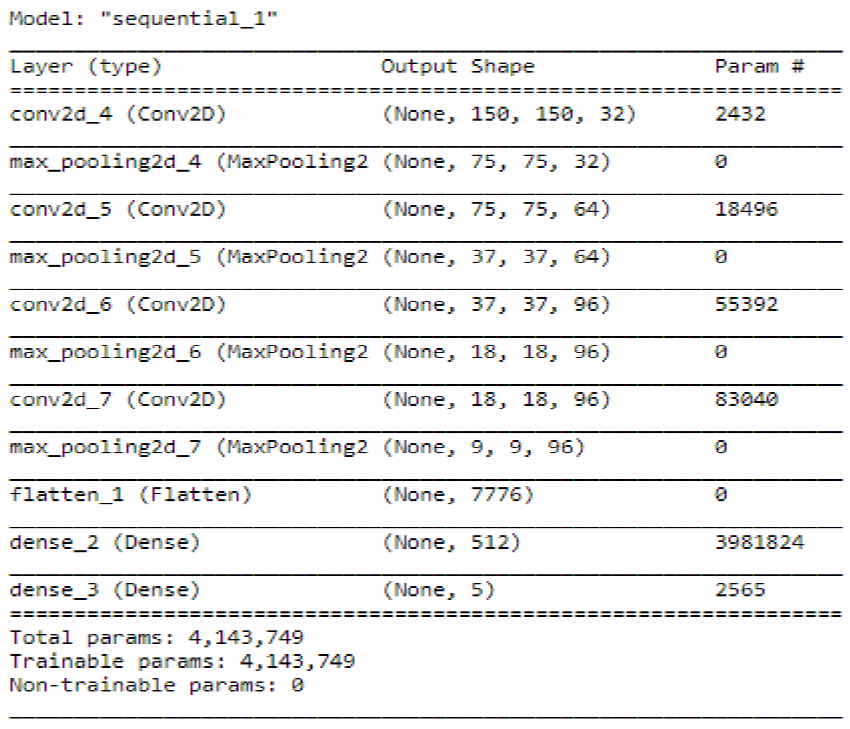
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D,Dense,Flatten,MaxPooling2D, Dropout
```

```
model = Sequential()
## add a conv layer followed by maxpooling

# First convolution extracts 32 filters that are 5x5
# Convolution is followed by max-pooling layer with a 2x2 window

model.add(Conv2D(filters = 32, kernel_size = (5,5),padding = 'Same',activation ='relu', input_shape = (150,150,3)))
model.add(MaxPooling2D(pool_size=(2,2)))

# params = ((n*m*k)+1)*f = ((5*5*3)+1)*32=2432

# Second convolution extracts 64 filters that are 3x3
# Convolution is followed by max-pooling layer with a 2x2 window

model.add(Conv2D(filters = 64, kernel_size = (3,3),padding = 'Same',activation ='relu'))
model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))

# params = ((n*m*k)+1)*f = ((3*3*32)+1)*64=18496

# Third convolution extracts 96 filters that are 3x3
# Convolution is followed by max-pooling with a 2x2 window

model.add(Conv2D(filters =96, kernel_size = (3,3),padding = 'Same',activation ='relu'))
model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))

# params = ((n*m*k)+1)*f = ((3*3*64)+1)*96=55392

# fourth convolution extracts 96 filters that are 3x3
# Convolution is followed by max-pooling layer with a 2x2 window

model.add(Conv2D(filters = 96, kernel_size = (3,3),padding = 'Same',activation ='relu'))
model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))

# params = ((n*m*k)+1)*f = ((3*3*96)+1)*96= 83040

# Flatten feature map to a 1-dim tensor so we can add fully connected layers

model.add(Flatten())

# Create a fully connected layer with ReLU activation and 512 hidden units

model.add(Dense(512,activation='relu'))

# params = ((n*m*k)+1)*f = ((3*3*96)+1)*512= 3981824

# Create output layer with a multiple node and softmax activation

model.add(Dense(5, activation = "softmax"))

# let us see the summmary of the model

model.summary()
```

Figure 4.4.1: Building the model 1

There are four convolution layers in the model. The first layer consists of input shape i.e., is 150 x 150 x 3 and the maxpooling of size 2x2 and the number of filters is 32 and the size of window is 5x5. The second layer consists of maxpooling of size 2x2 and the number of filters is 64 and the size of window is 3x3. The third and fourth layer consists of maxpooling of size 2x2 and the number of filters are 96 and the size of window is 3x3.Activation function for all the layers is ReLu and activation function for the output layer is SoftMax because it contains multiple nodes and 512 nodes are present in fully connected layer.

The parameters present in each layer are shown in the summary of the layers where you can see there are 2432 in the first layer and 18946 in the second layer and 55392 and 83040 in the third and fourth layer. There will no params for the maxpooling layers and the params in the fully connected layer and the dense layer are 3981824 and 2565.



```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_4 (Conv2D)            (None, 150, 150, 32)      2432
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 75, 75, 32)        0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 75, 75, 64)        18496
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 37, 37, 64)        0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 37, 37, 96)        55392
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 18, 18, 96)        0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 18, 18, 96)        83040
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 9, 9, 96)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 7776)              0
_________________________________________________________________
dense_2 (Dense)              (None, 512)               3981824
_________________________________________________________________
dense_3 (Dense)              (None, 5)                 2565
=================================================================
Total params: 4,143,749
Trainable params: 4,143,749
Non-trainable params: 0
_________________________________________________________________
```

Figure 4.4.2: Model 1 Summary

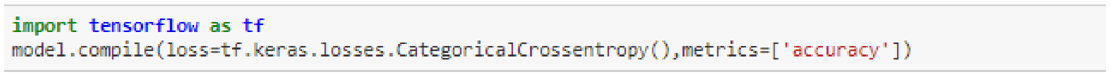
4.4.1 COMPILING THE MODEL:

- Compiling of the model is done using categorical cross entropy because there multiple categories in the dataset and the metrics used is accuracy
- Categorical cross entropy is a loss function that is used in multi-class classification tasks. These are tasks where an example can only belong to one out of many possible categories, and the model must decide which one.

- Adam is an optimization algorithm that can be used instead of the classical stochastic gradient descent procedure to update network weights iterative based in training data. ... The algorithm is called Adam.

Metrics

Classification Accuracy is what we usually mean, when we use the term accuracy. It is the ratio of number of correct predictions to the total number of input samples. It works well only if there are equal number of samples belonging to each class.



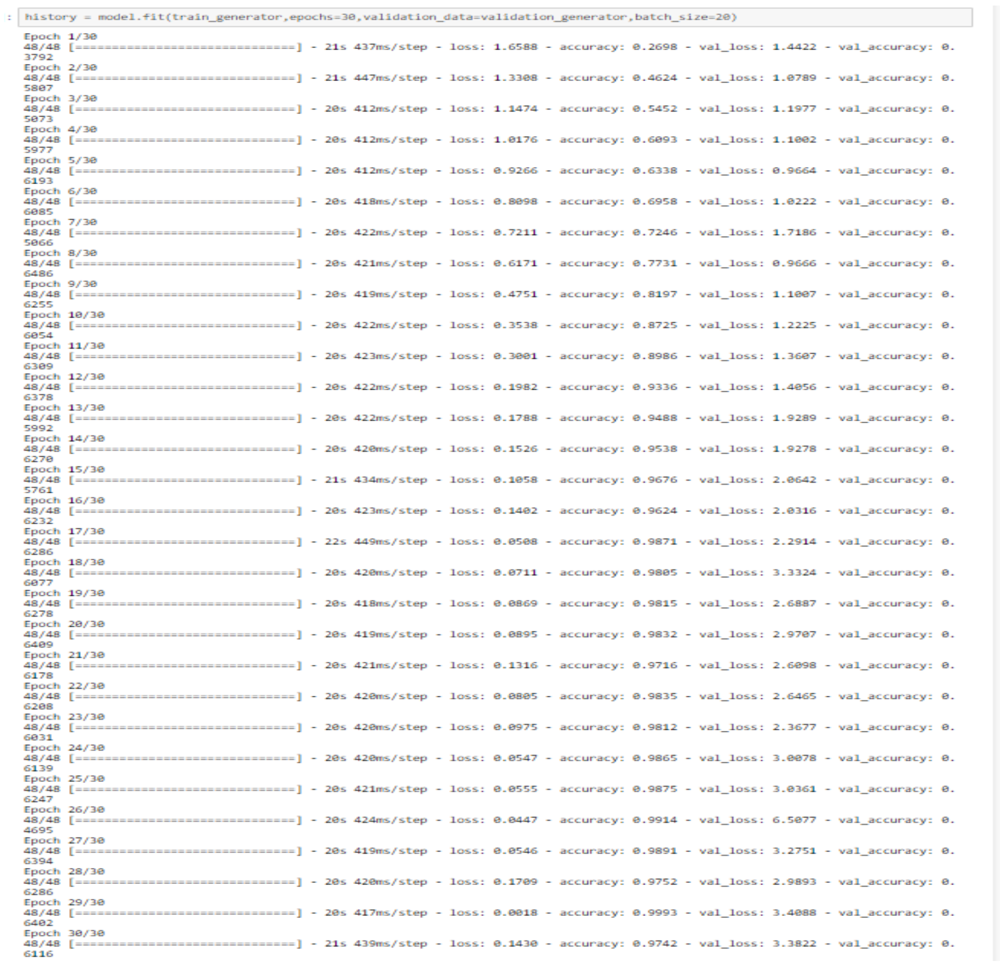
```
import tensorflow as tf
model.compile(loss=tf.keras.losses.CategoricalCrossentropy(),metrics=['accuracy'])
```

Figure 4.4.1.1: Compiling the model

4.4.2 FITTING THE MODEL:

One **Epoch** is when an ENTIRE dataset is passed forward and backward through the neural network only ONCE. Since one **epoch** is too big to feed to the computer at once we divide it in several smaller batches.

The **Batch size** is a number of samples processed before the model is updated. The number of epochs is the number of complete passes through the training dataset. The size of a batch must be more than or equal to one and less than or equal to the number of samples in the training dataset. The **forward propagation** process, we randomly initialized the weights, biases and filters. These values are treated as parameters from the convolutional neural network algorithm. In the backward **propagation** process, the model tries to update the parameters such that the overall predictions are more accurate We can use **classification** performance **metrics** such as Log-Loss, Accuracy, AUC (Area under Curve) etc. Another example of **metric** for **evaluation** of machine learning algorithms is precision, recall, which can be used for sorting algorithms primarily used by search engines



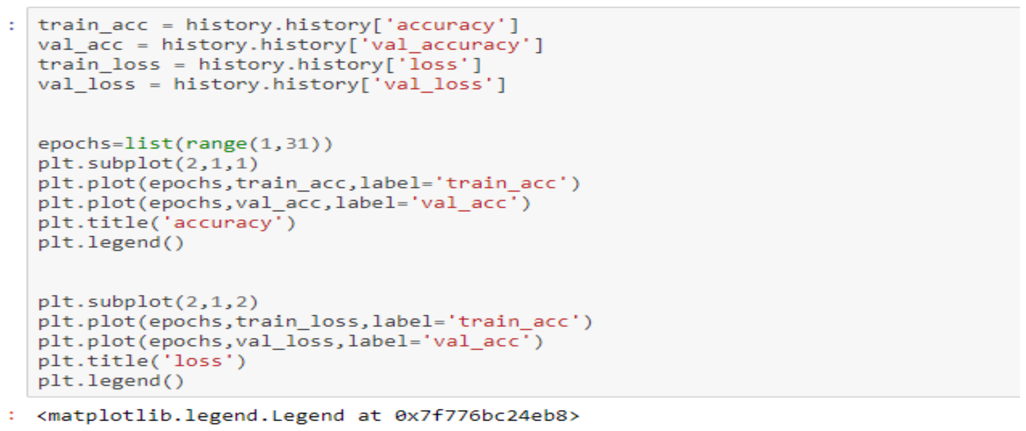
```
: history = model.fit(train_generator,epochs=30,validation_data=validation_generator,batch_size=20)

  Epoch 1/30
  48/48 [==============================] - 21s 437ms/step - loss: 1.6588 - accuracy: 0.2698 - val_loss: 1.4422 - val_accuracy: 0.
  3792
  Epoch 2/30
  48/48 [==============================] - 21s 447ms/step - loss: 1.3308 - accuracy: 0.4624 - val_loss: 1.0789 - val_accuracy: 0.
  5807
  Epoch 3/30
  48/48 [==============================] - 20s 412ms/step - loss: 1.1474 - accuracy: 0.5452 - val_loss: 1.1977 - val_accuracy: 0.
  5073
  Epoch 4/30
  48/48 [==============================] - 20s 412ms/step - loss: 1.0176 - accuracy: 0.6093 - val_loss: 1.1002 - val_accuracy: 0.
  5977
  Epoch 5/30
  48/48 [==============================] - 20s 412ms/step - loss: 0.9266 - accuracy: 0.6338 - val_loss: 0.9664 - val_accuracy: 0.
  6193
  Epoch 6/30
  48/48 [==============================] - 20s 418ms/step - loss: 0.8098 - accuracy: 0.6958 - val_loss: 1.0222 - val_accuracy: 0.
  6085
  Epoch 7/30
  48/48 [==============================] - 20s 422ms/step - loss: 0.7211 - accuracy: 0.7246 - val_loss: 1.7186 - val_accuracy: 0.
  5066
  Epoch 8/30
  48/48 [==============================] - 20s 421ms/step - loss: 0.6171 - accuracy: 0.7731 - val_loss: 0.9666 - val_accuracy: 0.
  6486
  Epoch 9/30
  48/48 [==============================] - 20s 419ms/step - loss: 0.4751 - accuracy: 0.8197 - val_loss: 1.1007 - val_accuracy: 0.
  6255
  Epoch 10/30
  48/48 [==============================] - 20s 422ms/step - loss: 0.3538 - accuracy: 0.8725 - val_loss: 1.2225 - val_accuracy: 0.
  6054
  Epoch 11/30
  48/48 [==============================] - 20s 423ms/step - loss: 0.3001 - accuracy: 0.8986 - val_loss: 1.3607 - val_accuracy: 0.
  6309
  Epoch 12/30
  48/48 [==============================] - 20s 422ms/step - loss: 0.1982 - accuracy: 0.9336 - val_loss: 1.4056 - val_accuracy: 0.
  6378
  Epoch 13/30
  48/48 [==============================] - 20s 422ms/step - loss: 0.1788 - accuracy: 0.9488 - val_loss: 1.9289 - val_accuracy: 0.
  5992
  Epoch 14/30
  48/48 [==============================] - 20s 420ms/step - loss: 0.1526 - accuracy: 0.9538 - val_loss: 1.9278 - val_accuracy: 0.
  6270
  Epoch 15/30
  48/48 [==============================] - 21s 434ms/step - loss: 0.1058 - accuracy: 0.9676 - val_loss: 2.0642 - val_accuracy: 0.
  5761
  Epoch 16/30
  48/48 [==============================] - 20s 423ms/step - loss: 0.1402 - accuracy: 0.9624 - val_loss: 2.0316 - val_accuracy: 0.
  6232
  Epoch 17/30
  48/48 [==============================] - 22s 449ms/step - loss: 0.0508 - accuracy: 0.9871 - val_loss: 2.2914 - val_accuracy: 0.
  6286
  Epoch 18/30
  48/48 [==============================] - 20s 420ms/step - loss: 0.0711 - accuracy: 0.9805 - val_loss: 3.3324 - val_accuracy: 0.
  6077
  Epoch 19/30
  48/48 [==============================] - 20s 418ms/step - loss: 0.0869 - accuracy: 0.9815 - val_loss: 2.6887 - val_accuracy: 0.
  6278
  Epoch 20/30
  48/48 [==============================] - 20s 419ms/step - loss: 0.0895 - accuracy: 0.9832 - val_loss: 2.9707 - val_accuracy: 0.
  6409
  Epoch 21/30
  48/48 [==============================] - 20s 421ms/step - loss: 0.1316 - accuracy: 0.9716 - val_loss: 2.6098 - val_accuracy: 0.
  6178
  Epoch 22/30
  48/48 [==============================] - 20s 420ms/step - loss: 0.0805 - accuracy: 0.9835 - val_loss: 2.6465 - val_accuracy: 0.
  6208
  Epoch 23/30
  48/48 [==============================] - 20s 420ms/step - loss: 0.0975 - accuracy: 0.9812 - val_loss: 2.3677 - val_accuracy: 0.
  6031
  Epoch 24/30
  48/48 [==============================] - 20s 420ms/step - loss: 0.0547 - accuracy: 0.9865 - val_loss: 3.0078 - val_accuracy: 0.
  6139
  Epoch 25/30
  48/48 [==============================] - 20s 421ms/step - loss: 0.0555 - accuracy: 0.9875 - val_loss: 3.0361 - val_accuracy: 0.
  6247
  Epoch 26/30
  48/48 [==============================] - 20s 424ms/step - loss: 0.0447 - accuracy: 0.9914 - val_loss: 6.5077 - val_accuracy: 0.
  4695
  Epoch 27/30
  48/48 [==============================] - 20s 419ms/step - loss: 0.0546 - accuracy: 0.9891 - val_loss: 3.2751 - val_accuracy: 0.
  6394
  Epoch 28/30
  48/48 [==============================] - 20s 420ms/step - loss: 0.1709 - accuracy: 0.9752 - val_loss: 2.9893 - val_accuracy: 0.
  6286
  Epoch 29/30
  48/48 [==============================] - 20s 417ms/step - loss: 0.0018 - accuracy: 0.9993 - val_loss: 3.4088 - val_accuracy: 0.
  6402
  Epoch 30/30
  48/48 [==============================] - 21s 439ms/step - loss: 0.1430 - accuracy: 0.9742 - val_loss: 3.3822 - val_accuracy: 0.
  6116
```

Figure 4.4.2.1: fitting the model

Here in Model 1 30 epoch are taken and the batch size of 20. After the 30 epochs the training accuracy we got is 0.9742 and the validation accuracy is 0.6116 and the losses are 0.1430 and 3.3822

4.4.3 VISUALIZATION OF LOSS AND ACCURACY OF MODEL1:

Whenever fit () is called, it returns a History object that can be used to visualize the training history. It contains a dictionary with loss and metric values at each epoch calculated both for training and validation datasets.



```
: train_acc = history.history['accuracy']
  val_acc = history.history['val_accuracy']
  train_loss = history.history['loss']
  val_loss = history.history['val_loss']


  epochs=list(range(1,31))
  plt.subplot(2,1,1)
  plt.plot(epochs,train_acc,label='train_acc')
  plt.plot(epochs,val_acc,label='val_acc')
  plt.title('accuracy')
  plt.legend()


  plt.subplot(2,1,2)
  plt.plot(epochs,train_loss,label='train_acc')
  plt.plot(epochs,val_loss,label='val_acc')
  plt.title('loss')
  plt.legend()
: <matplotlib.legend.Legend at 0x7f776bc24eb8>
```

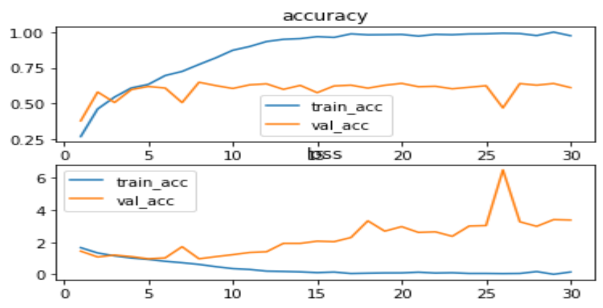


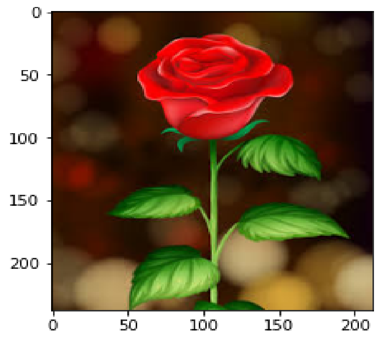
Figure 4.4.3.1: Metrics of the model 1

From the above plot we can understand that the model 1 accuracy has overfit. The above two plots the relations between validation loss and training loss and validation accuracy and the training accuracy. all the 30 epochs are taken to plot. The y axis represents the accuracy and losses and the x axis represent the no of epochs.

4.4.4 PREPROCESSING AND PREDICTING THE ROSE:

```
from tensorflow.keras.preprocessing import image
import numpy as np
img = image.load_img('/content/drive/My Drive/flowers/rose.jpg')
print(type(img))
plt.imshow(img)
#print(img.shape)

img = tf.keras.preprocessing.image.img_to_array(img)
print(img.shape)
print(type(img))
img = tf.image.resize(img,(150,150))

#scaling
img = img/255
print(img.shape)
img = np.expand_dims(img,axis=0)
print(img.shape)
```

```
<class 'PIL.JpegImagePlugin.JpegImageFile'>
(238, 212, 3)
<class 'numpy.ndarray'>
(150, 150, 3)
(1, 150, 150, 3)
```



```
model.predict(img).round(3)
```

```
array([[0.038, 0.   , 0.898, 0.   , 0.064]], dtype=float32)
```

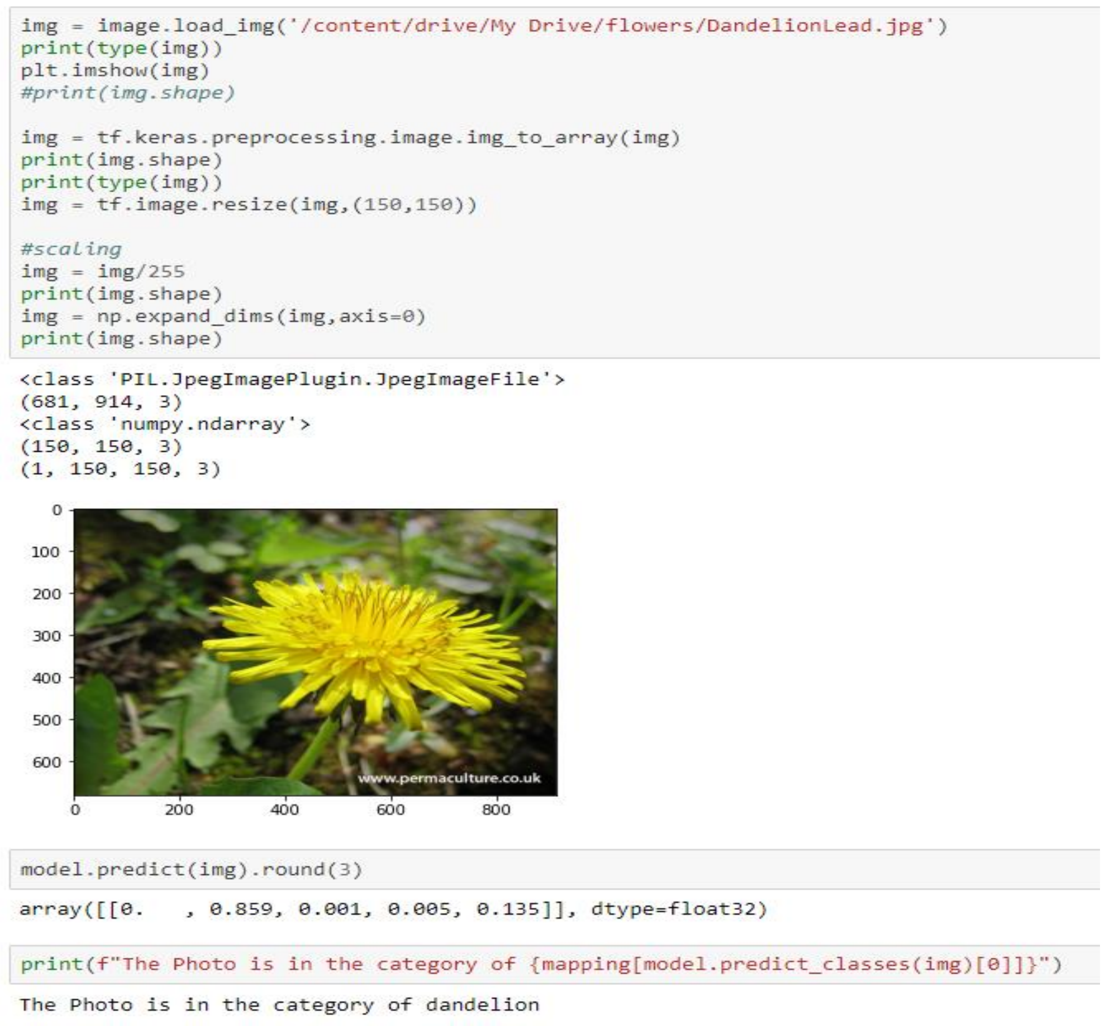
```
print(f"The Photo is in the category of {mapping[model.predict_classes(img)[0]]}")
```

```
The Photo is in the category of roses
```

Figure 4.4.4.1: Predicting the rose.

A rose image is taken from the google and the preprocessing steps such as Scaling and labelling is done to the image. After applying the preprocessing steps the image is tested with model with predict () method and the category of the flower present image are shown.

4.4.5 PREPROCESSING AND PREDICTING THE DANDELION:



```
img = image.load_img('/content/drive/My Drive/flowers/DandelionLead.jpg')
print(type(img))
plt.imshow(img)
#print(img.shape)

img = tf.keras.preprocessing.image.img_to_array(img)
print(img.shape)
print(type(img))
img = tf.image.resize(img,(150,150))

#scaling
img = img/255
print(img.shape)
img = np.expand_dims(img,axis=0)
print(img.shape)
```

```
<class 'PIL.JpegImagePlugin.JpegImageFile'>
(681, 914, 3)
<class 'numpy.ndarray'>
(150, 150, 3)
(1, 150, 150, 3)
```

```
model.predict(img).round(3)
```

```
array([[0.   , 0.859, 0.001, 0.005, 0.135]], dtype=float32)
```

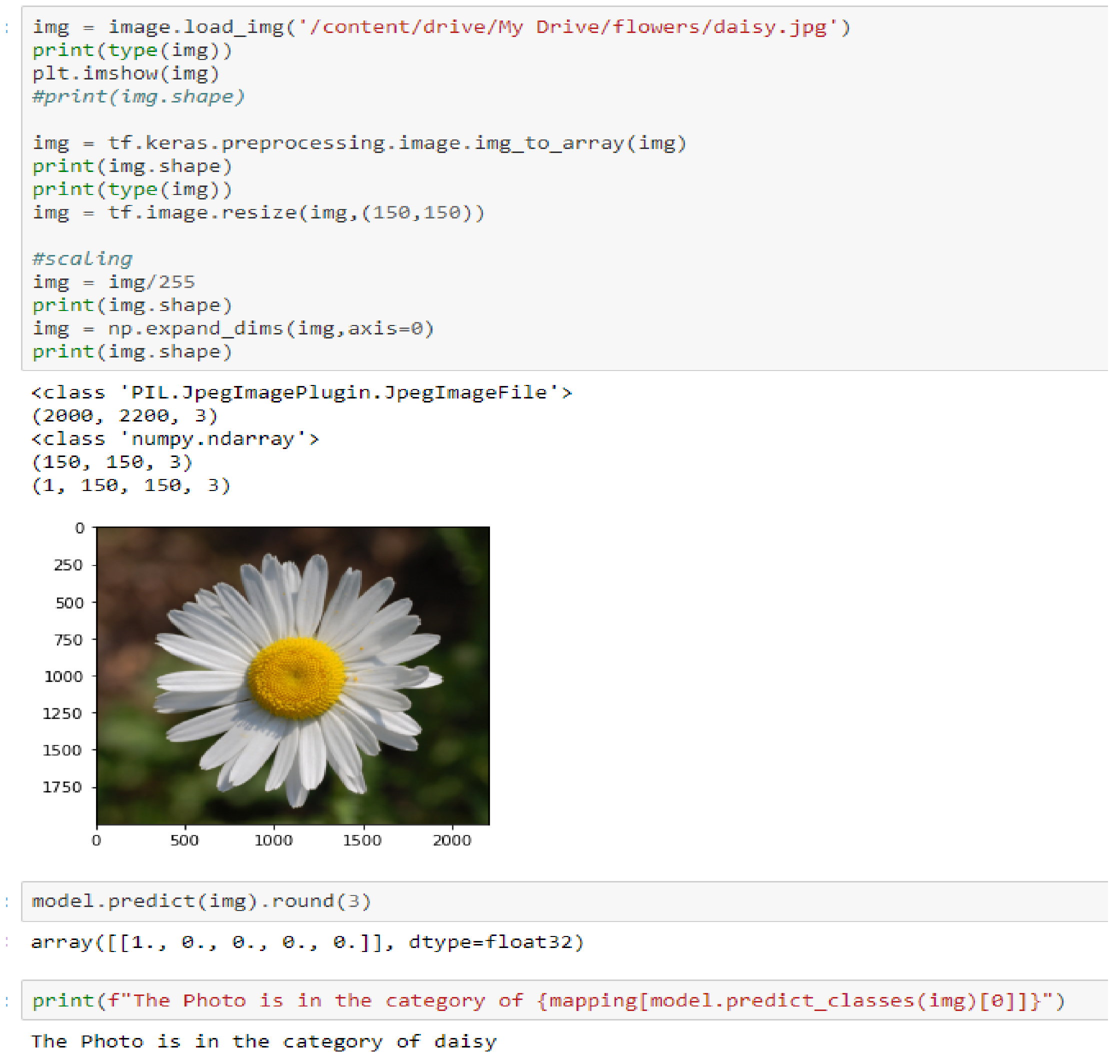
```
print(f"The Photo is in the category of {mapping[model.predict_classes(img)[0]]}")
```

```
The Photo is in the category of dandelion
```

Figure 4.4.5.1: Predicting the Dandelion.

A dandelion image is taken from the google and the preprocessing steps such as Scaling and Labelling is done to the image. After applying the preprocessing steps the image is tested with model with predict () method and the category of the flower present image are shown.

4.4.6 PREPROCESSING AND PREDICTING THE DAISY:



```
img = image.load_img('/content/drive/My Drive/flowers/daisy.jpg')
print(type(img))
plt.imshow(img)
#print(img.shape)

img = tf.keras.preprocessing.image.img_to_array(img)
print(img.shape)
print(type(img))
img = tf.image.resize(img,(150,150))

#scaling
img = img/255
print(img.shape)
img = np.expand_dims(img,axis=0)
print(img.shape)
```

```
<class 'PIL.JpegImagePlugin.JpegImageFile'>
(2000, 2200, 3)
<class 'numpy.ndarray'>
(150, 150, 3)
(1, 150, 150, 3)
```

```
model.predict(img).round(3)
```

```
array([[1., 0., 0., 0., 0.]], dtype=float32)
```

```
print(f"The Photo is in the category of {mapping[model.predict_classes(img)[0]]}")
```

```
The Photo is in the category of daisy
```

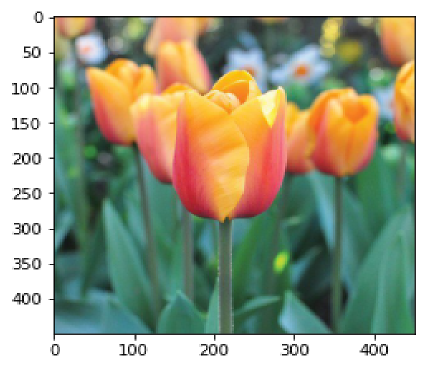
Figure 4.4.6.1: Predicting Daisy6

A daisy image is taken from the google and the preprocessing steps such as Scaling and Labelling is done to the image. After applying the preprocessing steps the image is tested with model with predict () method and the category of the flower present image are shown.

4.4.7 PREPROCESSING AND PREDICTING THE TULIPS:

```
img = image.load_img('/content/drive/My Drive/flowers/tulip.jpg')
print(type(img))
plt.imshow(img)
#print(img.shape)

img = tf.keras.preprocessing.image.img_to_array(img)
print(img.shape)
print(type(img))
img = tf.image.resize(img,(150,150))

#scaling
img = img/255
print(img.shape)
img = np.expand_dims(img,axis=0)
print(img.shape)
```

```
<class 'PIL.JpegImagePlugin.JpegImageFile'>
(450, 450, 3)
<class 'numpy.ndarray'>
(150, 150, 3)
(1, 150, 150, 3)
```



```
model.predict(img).round(3)
```

```
array([[0., 0., 0., 0., 1.]], dtype=float32)
```

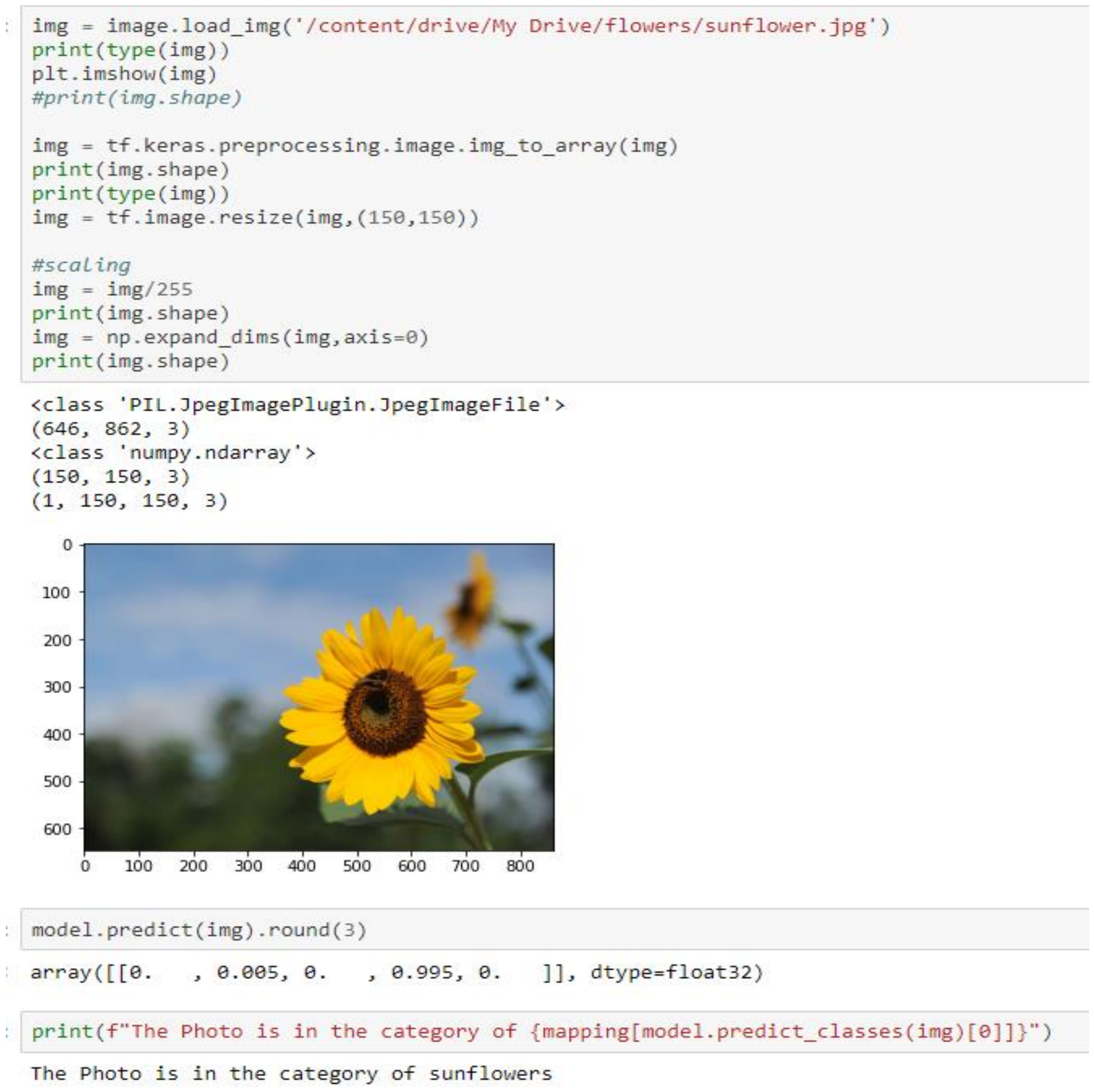
```
print(f"The Photo is in the category of {mapping[model.predict_classes(img)[0]]}")
```

```
The Photo is in the category of tulips
```

Figure 4.4.7.1: Predicting the tulips.

A tulips image is taken from the google and the preprocessing steps such as Scaling and Labelling is done to the image. After applying the preprocessing steps the image is tested with model with predict () method and the category of the flower present image are shown.

4.4.8 PREPROCESSING AND PREDICTING THE SUNFLOWER:



```
img = image.load_img('/content/drive/My Drive/flowers/sunflower.jpg')
print(type(img))
plt.imshow(img)
#print(img.shape)

img = tf.keras.preprocessing.image.img_to_array(img)
print(img.shape)
print(type(img))
img = tf.image.resize(img,(150,150))

#scaling
img = img/255
print(img.shape)
img = np.expand_dims(img,axis=0)
print(img.shape)
```

```
<class 'PIL.JpegImagePlugin.JpegImageFile'>
(646, 862, 3)
<class 'numpy.ndarray'>
(150, 150, 3)
(1, 150, 150, 3)
```

```
model.predict(img).round(3)
```

```
array([[0.   , 0.005, 0.   , 0.995, 0.   ]], dtype=float32)
```

```
print(f"The Photo is in the category of {mapping[model.predict_classes(img)[0]]}")
```

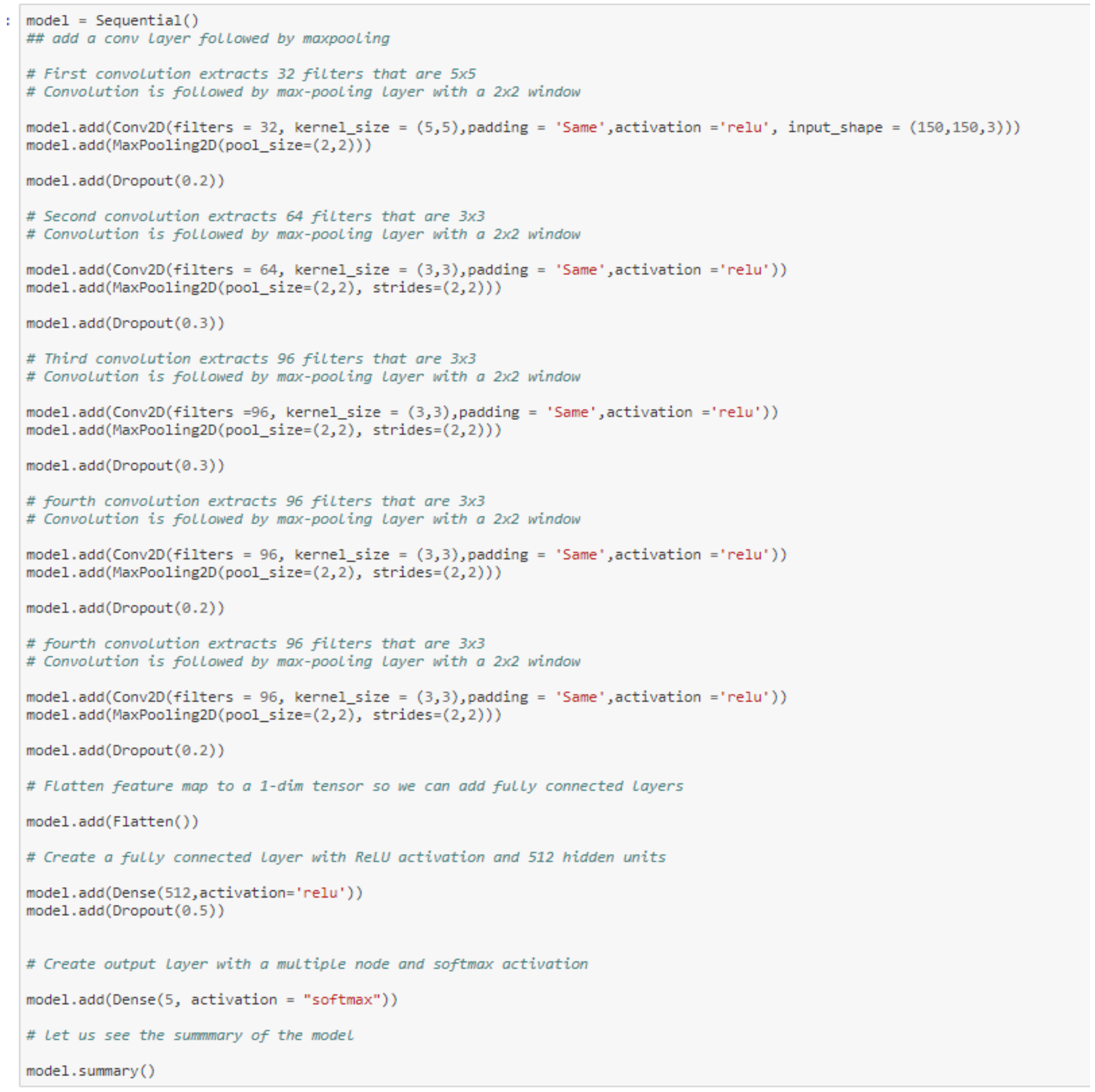
```
The Photo is in the category of sunflowers
```

Figure 4.4.8.1: Predicting the Sunflower.

- A sunflower image is taken from the google and the preprocessing steps such as Scaling and Labeling is done to the image. After applying the preprocessing steps the image is tested with model with predict () method and the category of the flower present image are shown.

4.5 IMPLEMENTING THE CNN MODEL 2:

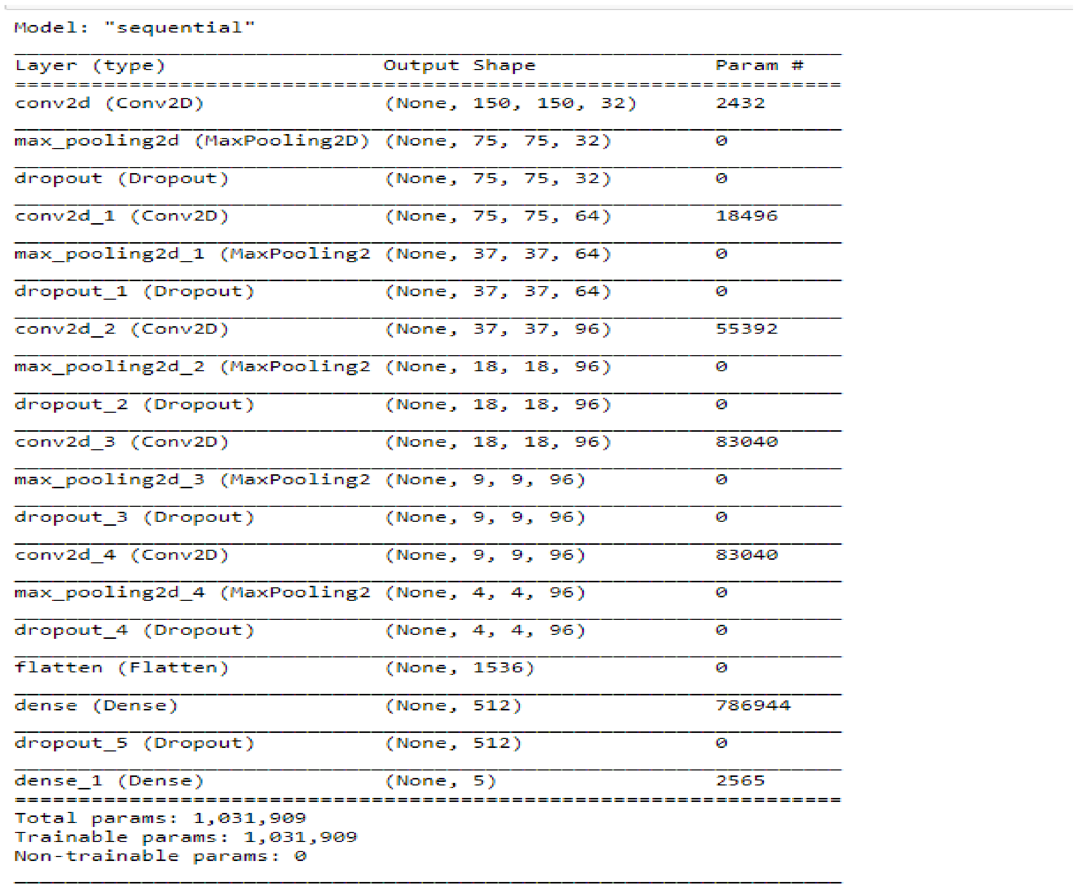
Dropout is a technique where randomly selected neurons are ignored during training. They are “dropped-out” randomly. This means that their contribution to the activation of downstream neurons is temporally removed on the forward pass and any weight updates are not applied to the neuron on the backward pass.



```
model = Sequential()
## add a conv layer followed by maxpooling

# First convolution extracts 32 filters that are 5x5
# Convolution is followed by max-pooling layer with a 2x2 window

model.add(Conv2D(filters = 32, kernel_size = (5,5),padding = 'Same',activation ='relu', input_shape = (150,150,3)))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Dropout(0.2))

# Second convolution extracts 64 filters that are 3x3
# Convolution is followed by max-pooling layer with a 2x2 window

model.add(Conv2D(filters = 64, kernel_size = (3,3),padding = 'Same',activation ='relu'))
model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))

model.add(Dropout(0.3))

# Third convolution extracts 96 filters that are 3x3
# Convolution is followed by max-pooling layer with a 2x2 window

model.add(Conv2D(filters =96, kernel_size = (3,3),padding = 'Same',activation ='relu'))
model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))

model.add(Dropout(0.3))

# fourth convolution extracts 96 filters that are 3x3
# Convolution is followed by max-pooling layer with a 2x2 window

model.add(Conv2D(filters = 96, kernel_size = (3,3),padding = 'Same',activation ='relu'))
model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))

model.add(Dropout(0.2))

# fourth convolution extracts 96 filters that are 3x3
# Convolution is followed by max-pooling layer with a 2x2 window

model.add(Conv2D(filters = 96, kernel_size = (3,3),padding = 'Same',activation ='relu'))
model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))

model.add(Dropout(0.2))

# Flatten feature map to a 1-dim tensor so we can add fully connected layers

model.add(Flatten())

# Create a fully connected layer with ReLU activation and 512 hidden units

model.add(Dense(512,activation='relu'))
model.add(Dropout(0.5))


# Create output layer with a multiple node and softmax activation

model.add(Dense(5, activation = "softmax"))

# let us see the summmary of the model

model.summary()
```

Figure 4.5.1: Building the model 2

- There are four convolution layers in the model. The first layer consists of input shape i.e., is 150 x 150 x 3 and the maxpooling of size 2x2 and the number of filters is 32 and the size of window is 5x5. The second layer consists of maxpooling of size 2x2 and the number of filters is 64 and the size of window is 3x3. The third, fourth and fifth layer consists of maxpooling of size 2x2 and the number of filters is 96 and the size of window is 3x3.
- Activation function for all the layers is ReLu and activation function for the output layer is SoftMax because it contains multiple nodes and 512 nodes are present in fully connected layer.
- The parameters present in each layer are shown in the summary of the layers where you can see there are 2432 in the first layer and 18946 in the second layer and 55392, 83040 and 83040 in the third and fourth layer.
- There will no params for the maxpooling layers and the params in the fully connected layer and the dense layer are 3981824 and 2565.The dropout regularization function is also added to the model the first layer dropout is 0.2 and the second- and third-layer dropout size is 0.3 and the fourth and fifth layer the dropout size is 0.2. The dropout is added to the layer because it increases the accuracy.

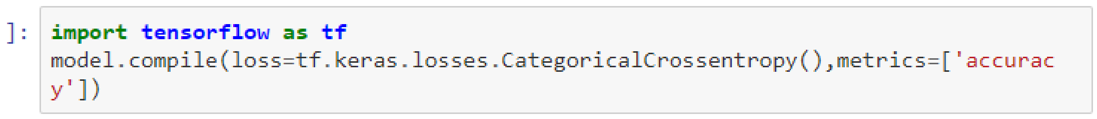


```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 150, 150, 32)      2432
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 75, 75, 32)        0
_________________________________________________________________
dropout (Dropout)            (None, 75, 75, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 75, 75, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 37, 37, 64)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 37, 37, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 37, 37, 96)        55392
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 18, 18, 96)        0
_________________________________________________________________
dropout_2 (Dropout)          (None, 18, 18, 96)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 18, 18, 96)        83040
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 9, 9, 96)          0
_________________________________________________________________
dropout_3 (Dropout)          (None, 9, 9, 96)          0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 9, 9, 96)          83040
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 4, 4, 96)          0
_________________________________________________________________
dropout_4 (Dropout)          (None, 4, 4, 96)          0
_________________________________________________________________
flatten (Flatten)            (None, 1536)              0
_________________________________________________________________
dense (Dense)                (None, 512)               786944
_________________________________________________________________
dropout_5 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 5)                 2565
=================================================================
Total params: 1,031,909
Trainable params: 1,031,909
Non-trainable params: 0
_________________________________________________________________
```

Figure 4.5.2: Model 2 Summary

4.5.1 COMPILING THE MODEL:

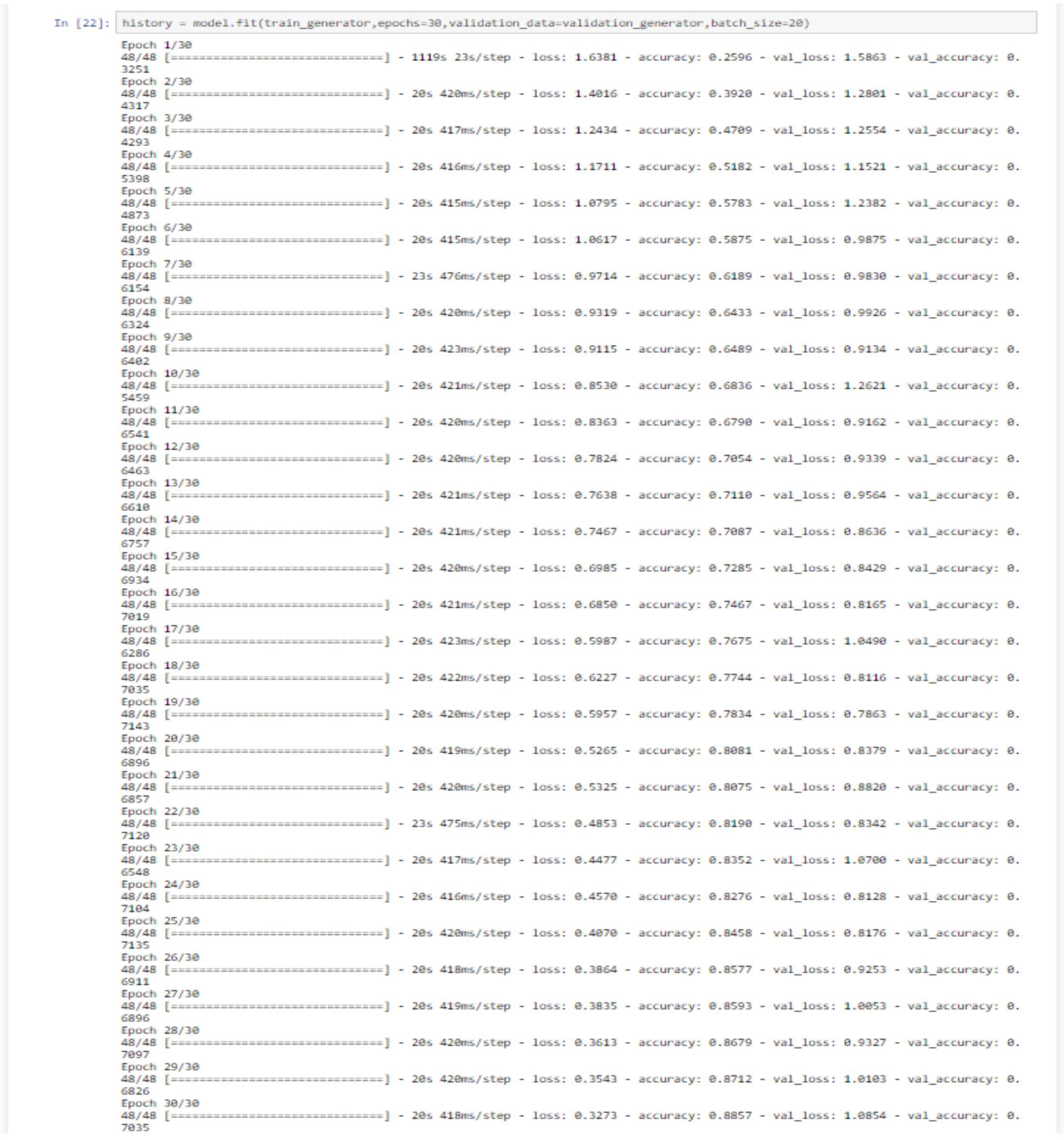
Compiling of the model is done using categorical cross entropy because there are multiple categories in the dataset and the metrics used is accuracy



```
]: import tensorflow as tf
model.compile(loss=tf.keras.losses.CategoricalCrossentropy(),metrics=['accurac
y'])
```

Figure 4.5.1.1: Compiling the model

4.5.2 FITTING THE MODEL:



```
In [22]: history = model.fit(train_generator,epochs=30,validation_data=validation_generator,batch_size=20)

         Epoch 1/30
         48/48 [==============================] - 1119s 23s/step - loss: 1.6381 - accuracy: 0.2596 - val_loss: 1.5863 - val_accuracy: 0.
         3251
         Epoch 2/30
         48/48 [==============================] - 20s 420ms/step - loss: 1.4016 - accuracy: 0.3920 - val_loss: 1.2801 - val_accuracy: 0.
         4317
         Epoch 3/30
         48/48 [==============================] - 20s 417ms/step - loss: 1.2434 - accuracy: 0.4709 - val_loss: 1.2554 - val_accuracy: 0.
         4293
         Epoch 4/30
         48/48 [==============================] - 20s 416ms/step - loss: 1.1711 - accuracy: 0.5182 - val_loss: 1.1521 - val_accuracy: 0.
         5398
         Epoch 5/30
         48/48 [==============================] - 20s 415ms/step - loss: 1.0795 - accuracy: 0.5783 - val_loss: 1.2382 - val_accuracy: 0.
         4873
         Epoch 6/30
         48/48 [==============================] - 20s 415ms/step - loss: 1.0617 - accuracy: 0.5875 - val_loss: 0.9875 - val_accuracy: 0.
         6139
         Epoch 7/30
         48/48 [==============================] - 23s 476ms/step - loss: 0.9714 - accuracy: 0.6189 - val_loss: 0.9830 - val_accuracy: 0.
         6154
         Epoch 8/30
         48/48 [==============================] - 20s 420ms/step - loss: 0.9319 - accuracy: 0.6433 - val_loss: 0.9926 - val_accuracy: 0.
         6324
         Epoch 9/30
         48/48 [==============================] - 20s 423ms/step - loss: 0.9115 - accuracy: 0.6489 - val_loss: 0.9134 - val_accuracy: 0.
         6402
         Epoch 10/30
         48/48 [==============================] - 20s 421ms/step - loss: 0.8530 - accuracy: 0.6836 - val_loss: 1.2621 - val_accuracy: 0.
         5459
         Epoch 11/30
         48/48 [==============================] - 20s 420ms/step - loss: 0.8363 - accuracy: 0.6790 - val_loss: 0.9162 - val_accuracy: 0.
         6541
         Epoch 12/30
         48/48 [==============================] - 20s 420ms/step - loss: 0.7824 - accuracy: 0.7054 - val_loss: 0.9339 - val_accuracy: 0.
         6463
         Epoch 13/30
         48/48 [==============================] - 20s 421ms/step - loss: 0.7638 - accuracy: 0.7110 - val_loss: 0.9564 - val_accuracy: 0.
         6610
         Epoch 14/30
         48/48 [==============================] - 20s 421ms/step - loss: 0.7467 - accuracy: 0.7087 - val_loss: 0.8636 - val_accuracy: 0.
         6757
         Epoch 15/30
         48/48 [==============================] - 20s 420ms/step - loss: 0.6985 - accuracy: 0.7285 - val_loss: 0.8429 - val_accuracy: 0.
         6934
         Epoch 16/30
         48/48 [==============================] - 20s 421ms/step - loss: 0.6850 - accuracy: 0.7467 - val_loss: 0.8165 - val_accuracy: 0.
         7019
         Epoch 17/30
         48/48 [==============================] - 20s 423ms/step - loss: 0.5987 - accuracy: 0.7675 - val_loss: 1.0490 - val_accuracy: 0.
         6286
         Epoch 18/30
         48/48 [==============================] - 20s 422ms/step - loss: 0.6227 - accuracy: 0.7744 - val_loss: 0.8116 - val_accuracy: 0.
         7035
         Epoch 19/30
         48/48 [==============================] - 20s 420ms/step - loss: 0.5957 - accuracy: 0.7834 - val_loss: 0.7863 - val_accuracy: 0.
         7143
         Epoch 20/30
         48/48 [==============================] - 20s 419ms/step - loss: 0.5265 - accuracy: 0.8081 - val_loss: 0.8379 - val_accuracy: 0.
         6896
         Epoch 21/30
         48/48 [==============================] - 20s 420ms/step - loss: 0.5325 - accuracy: 0.8075 - val_loss: 0.8820 - val_accuracy: 0.
         6857
         Epoch 22/30
         48/48 [==============================] - 23s 475ms/step - loss: 0.4853 - accuracy: 0.8190 - val_loss: 0.8342 - val_accuracy: 0.
         7120
         Epoch 23/30
         48/48 [==============================] - 20s 417ms/step - loss: 0.4477 - accuracy: 0.8352 - val_loss: 1.0700 - val_accuracy: 0.
         6548
         Epoch 24/30
         48/48 [==============================] - 20s 416ms/step - loss: 0.4570 - accuracy: 0.8276 - val_loss: 0.8128 - val_accuracy: 0.
         7104
         Epoch 25/30
         48/48 [==============================] - 20s 420ms/step - loss: 0.4070 - accuracy: 0.8458 - val_loss: 0.8176 - val_accuracy: 0.
         7135
         Epoch 26/30
         48/48 [==============================] - 20s 418ms/step - loss: 0.3864 - accuracy: 0.8577 - val_loss: 0.9253 - val_accuracy: 0.
         6911
         Epoch 27/30
         48/48 [==============================] - 20s 419ms/step - loss: 0.3835 - accuracy: 0.8593 - val_loss: 1.0053 - val_accuracy: 0.
         6896
         Epoch 28/30
         48/48 [==============================] - 20s 420ms/step - loss: 0.3613 - accuracy: 0.8679 - val_loss: 0.9327 - val_accuracy: 0.
         7097
         Epoch 29/30
         48/48 [==============================] - 20s 420ms/step - loss: 0.3543 - accuracy: 0.8712 - val_loss: 1.0103 - val_accuracy: 0.
         6826
         Epoch 30/30
         48/48 [==============================] - 20s 418ms/step - loss: 0.3273 - accuracy: 0.8857 - val_loss: 1.0854 - val_accuracy: 0.
         7035
```

Figure 4.5.2.1: fitting the model

Here 30 epochs are taken and the batch size of 20. After the 30 epochs the training accuracy we got is 0.8857 and the validation accuracy is 0.7035 and the losses are 0.3273 and 1.0854.

4.5.3 VISUALIZATION OF LOSS AND ACCURACY OF MODEL2:

```
train_acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
train_loss = history.history['loss']
val_loss = history.history['val_loss']


epochs=list(range(1,31))
plt.subplot(2,1,1)
plt.plot(epochs,train_acc,label='train_acc')
plt.plot(epochs,val_acc,label='val_acc')
plt.title('accuracy')
plt.legend()


plt.subplot(2,1,2)
plt.plot(epochs,train_loss,label='train_acc')
plt.plot(epochs,val_loss,label='val_acc')
plt.title('loss')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7fd7e4498c88>
```

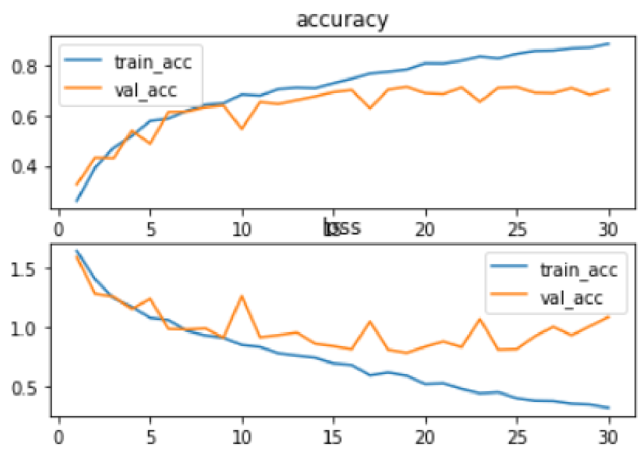
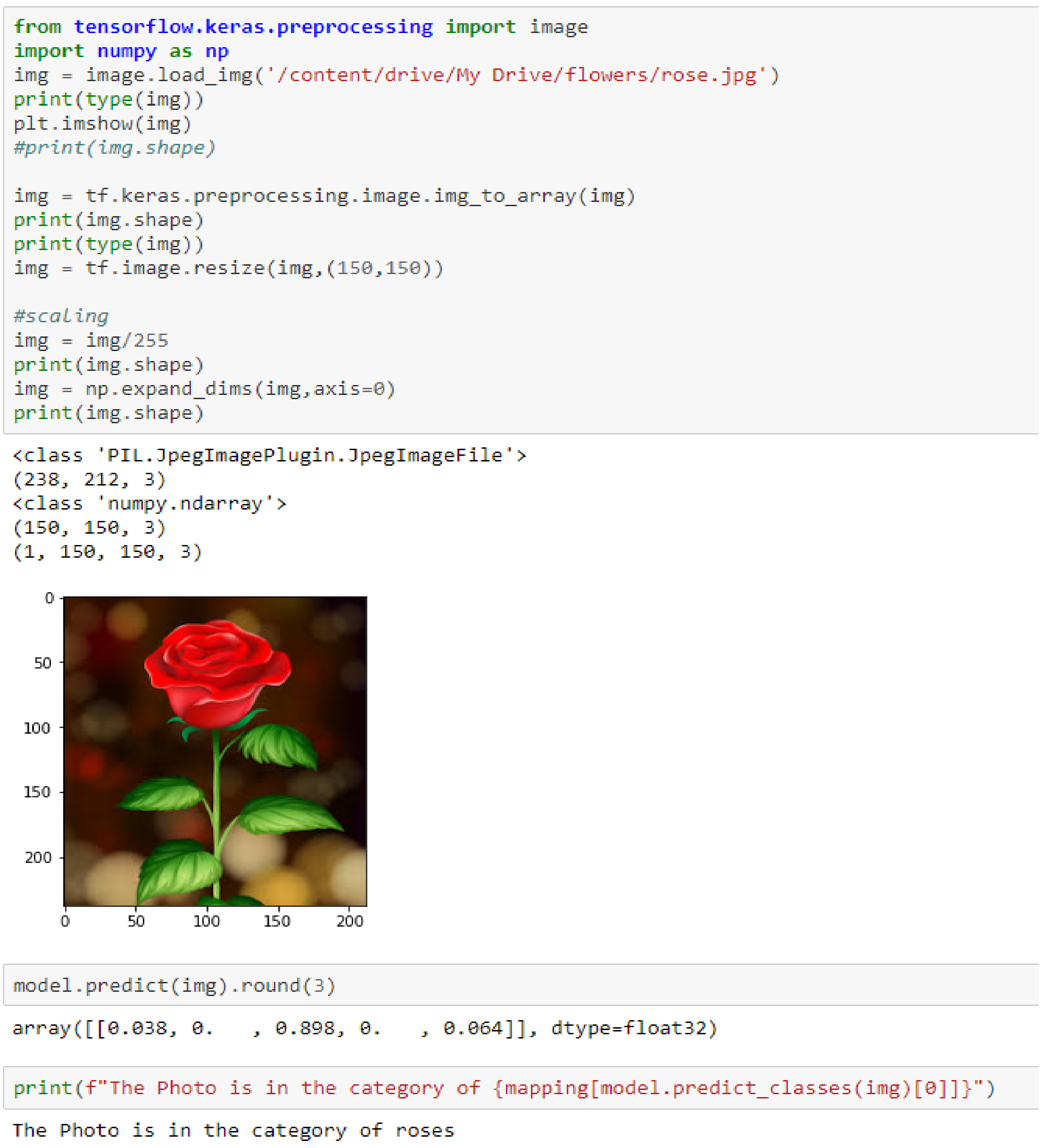


Figure 4.5.3.1: Metrics of the model 2

From the above plot we can understand that the model 2 accuracy has overfit. The above two plots the relations between validation loss and training loss and validation accuracy and the training accuracy. all the 30 epochs are taken to plot. The y axis represents the accuracy and losses and the x axis represent the no of epochs.

4.5.4 PREPROCESSING AND PREDICTING THE ROSE



```
from tensorflow.keras.preprocessing import image
import numpy as np
img = image.load_img('/content/drive/My Drive/flowers/rose.jpg')
print(type(img))
plt.imshow(img)
#print(img.shape)

img = tf.keras.preprocessing.image.img_to_array(img)
print(img.shape)
print(type(img))
img = tf.image.resize(img,(150,150))

#scaling
img = img/255
print(img.shape)
img = np.expand_dims(img,axis=0)
print(img.shape)
```

```
<class 'PIL.JpegImagePlugin.JpegImageFile'>
(238, 212, 3)
<class 'numpy.ndarray'>
(150, 150, 3)
(1, 150, 150, 3)
```

```
model.predict(img).round(3)
```

```
array([[0.038, 0.   , 0.898, 0.   , 0.064]], dtype=float32)
```

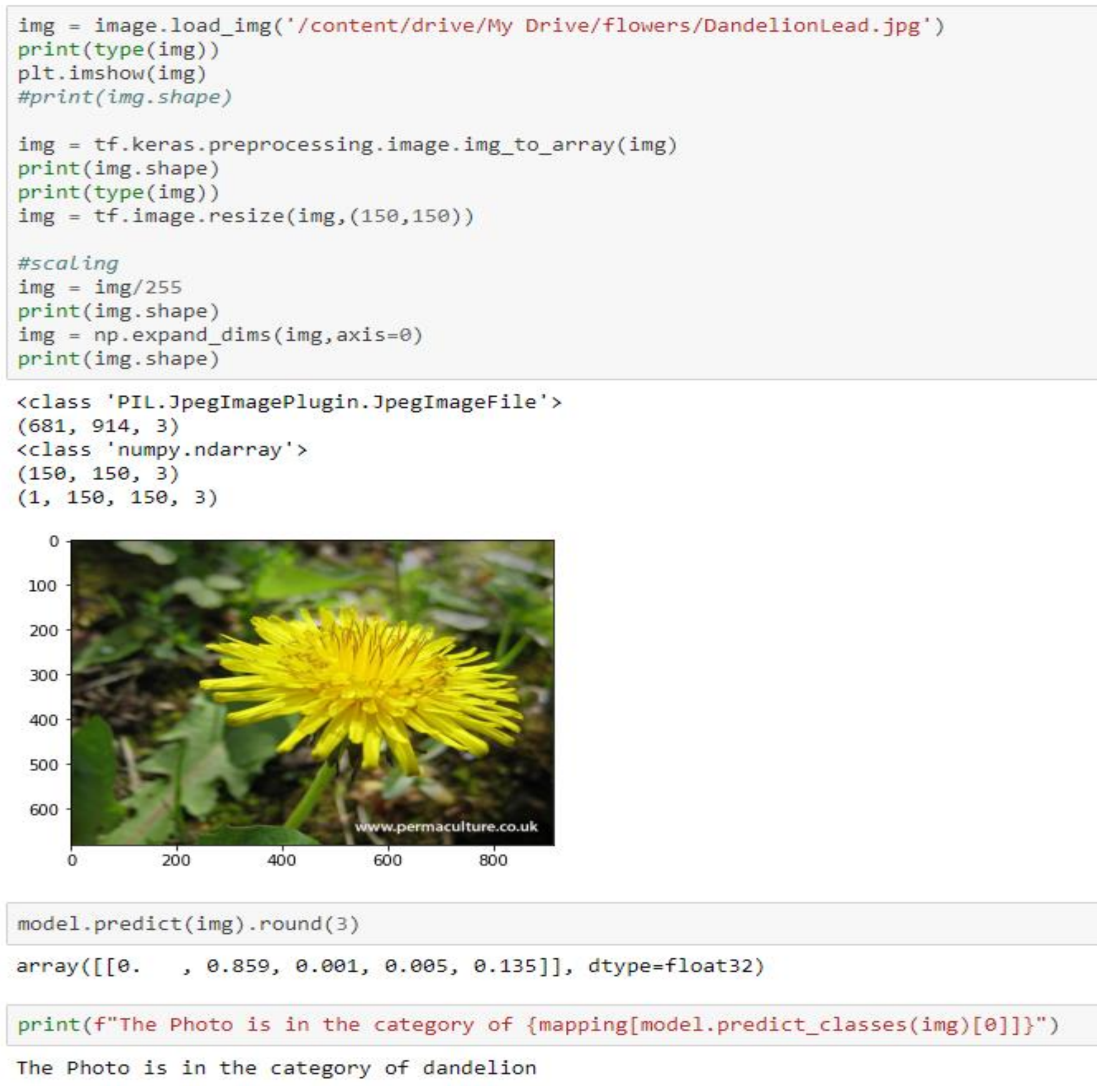
```
print(f"The Photo is in the category of {mapping[model.predict_classes(img)[0]]}")
```

```
The Photo is in the category of roses
```

Figure 4.5.4.1: Predicting the rose.

- ➢ A rose image is taken from the google and the preprocessing steps such as Scaling and Labelling is done to the image. After applying the preprocessing steps the image is tested with model with predict () method and the category of the flower present image are shown.

4.5.5 PREPROCESSING AND PREDICTING THE DANDELION:



```
img = image.load_img('/content/drive/My Drive/flowers/DandelionLead.jpg')
print(type(img))
plt.imshow(img)
#print(img.shape)

img = tf.keras.preprocessing.image.img_to_array(img)
print(img.shape)
print(type(img))
img = tf.image.resize(img,(150,150))

#scaling
img = img/255
print(img.shape)
img = np.expand_dims(img,axis=0)
print(img.shape)
```

```
<class 'PIL.JpegImagePlugin.JpegImageFile'>
(681, 914, 3)
<class 'numpy.ndarray'>
(150, 150, 3)
(1, 150, 150, 3)
```

```
model.predict(img).round(3)
```

```
array([[0.   , 0.859, 0.001, 0.005, 0.135]], dtype=float32)
```

```
print(f"The Photo is in the category of {mapping[model.predict_classes(img)[0]]}")
```

```
The Photo is in the category of dandelion
```

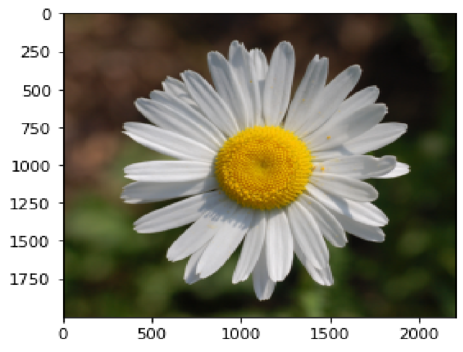
Figure 4.5.5.1: Predicting the Dandelion.

- A dandelion image is taken from the google and the preprocessing steps such as Scaling and Labeling is done to the image. After applying the preprocessing steps the image is tested with model with predict () method and the category of the flower present image are shown.

4.5.6 PREPROCESSING AND PREDICTING THE DAISY:

```
img = image.load_img('/content/drive/My Drive/flowers/daisy.jpg')
print(type(img))
plt.imshow(img)
#print(img.shape)

img = tf.keras.preprocessing.image.img_to_array(img)
print(img.shape)
print(type(img))
img = tf.image.resize(img,(150,150))

#scaling
img = img/255
print(img.shape)
img = np.expand_dims(img,axis=0)
print(img.shape)
```

```
<class 'PIL.JpegImagePlugin.JpegImageFile'>
(2000, 2200, 3)
<class 'numpy.ndarray'>
(150, 150, 3)
(1, 150, 150, 3)
```



```
model.predict(img).round(3)
```

```
array([[1., 0., 0., 0., 0.]], dtype=float32)
```

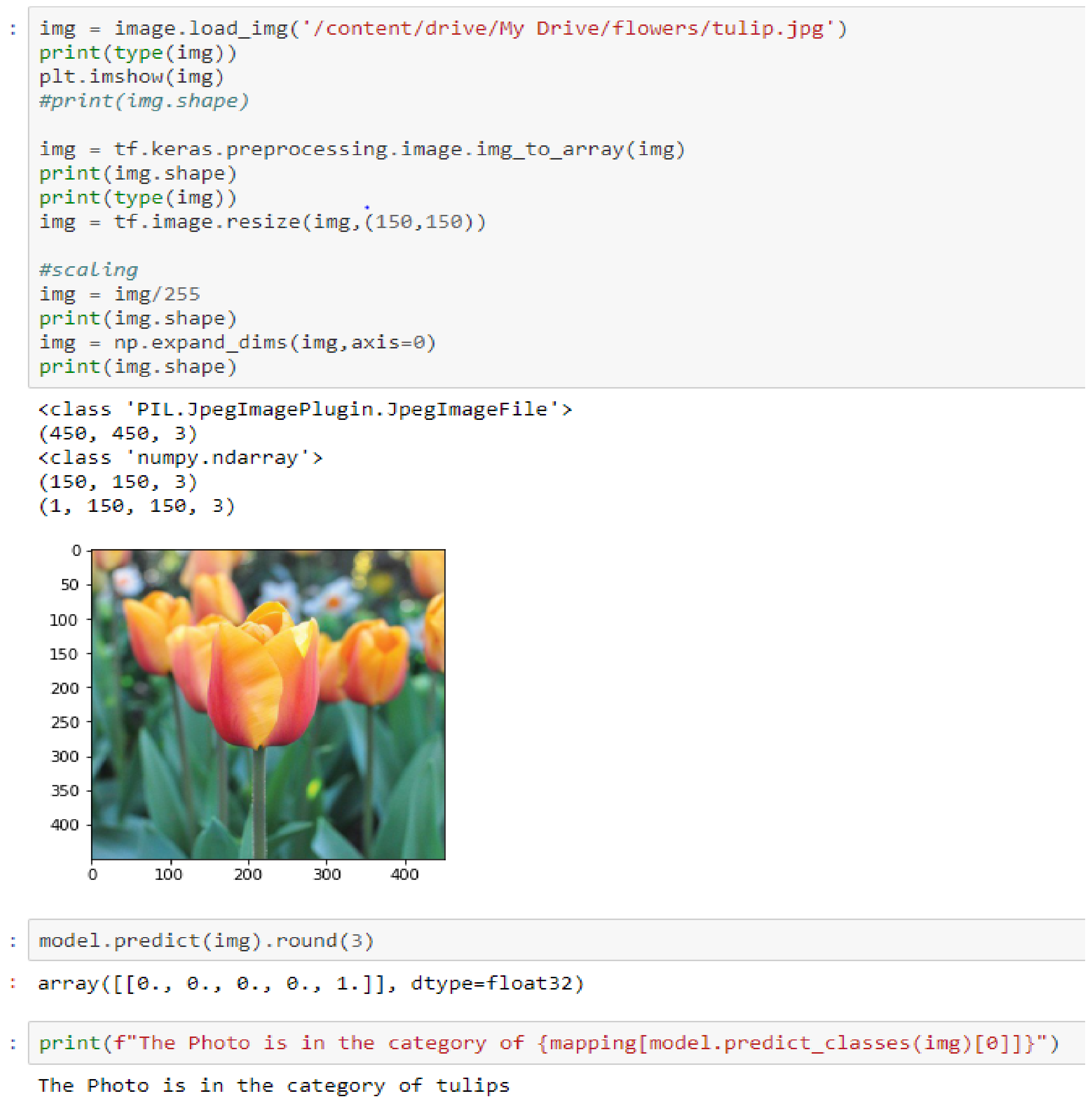
```
print(f"The Photo is in the category of {mapping[model.predict_classes(img)[0]]}")
```

```
The Photo is in the category of daisy
```

Figure 4.5.6.1: Predicting Daisy

- A rose image is taken from the google and the preprocessing steps such as Scaling and labelling is done to the image. After applying the preprocessing steps the image is tested with model with predict () method and the category of the flower present image are shown.

4.5.7 PREPROCESSING AND PREDICTING THE TULIPS:



```
img = image.load_img('/content/drive/My Drive/flowers/tulip.jpg')
print(type(img))
plt.imshow(img)
#print(img.shape)

img = tf.keras.preprocessing.image.img_to_array(img)
print(img.shape)
print(type(img))
img = tf.image.resize(img,(150,150))

#scaling
img = img/255
print(img.shape)
img = np.expand_dims(img,axis=0)
print(img.shape)
```

```
<class 'PIL.JpegImagePlugin.JpegImageFile'>
(450, 450, 3)
<class 'numpy.ndarray'>
(150, 150, 3)
(1, 150, 150, 3)
```

```
model.predict(img).round(3)
```

```
array([[0., 0., 0., 0., 1.]], dtype=float32)
```

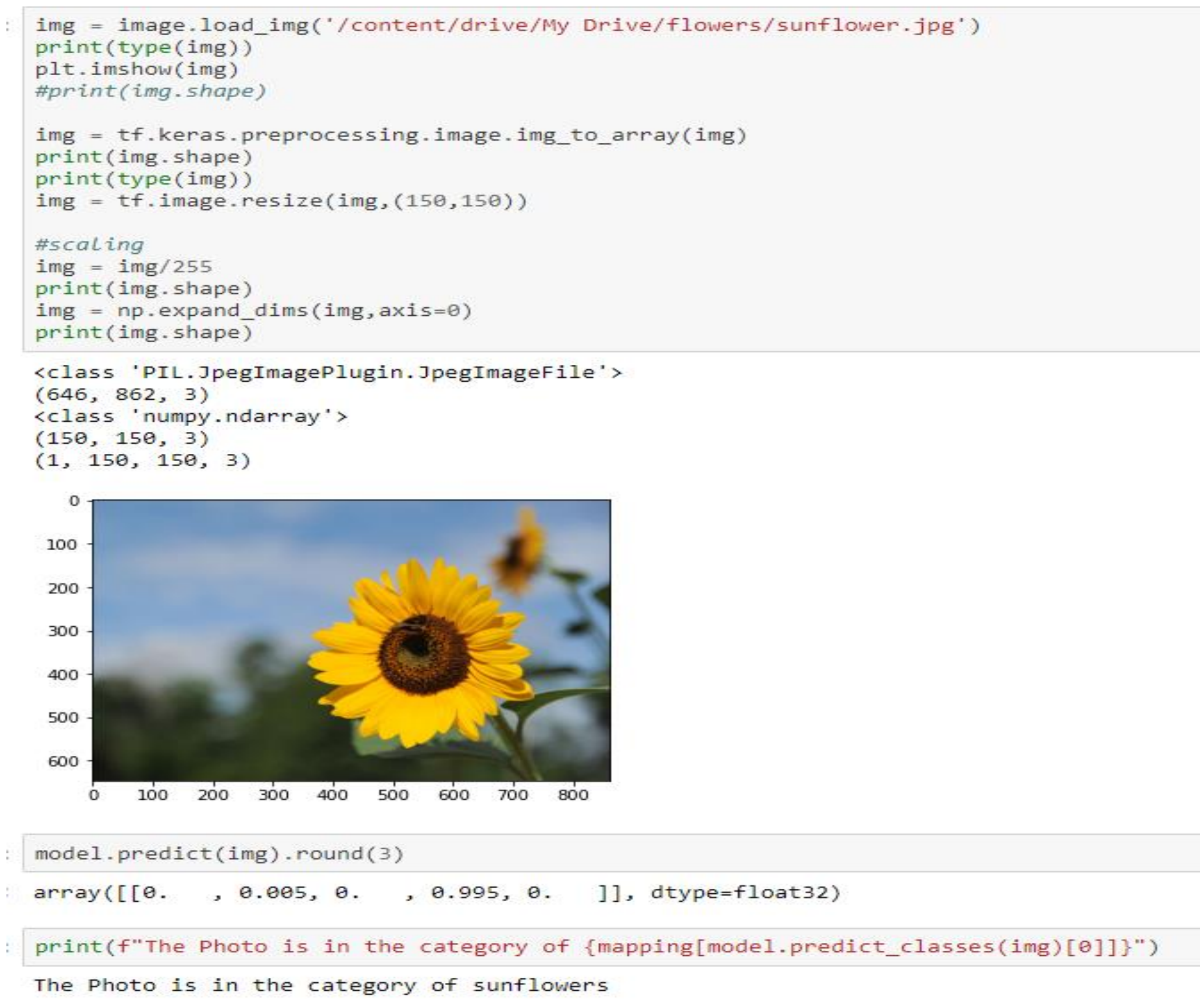
```
print(f"The Photo is in the category of {mapping[model.predict_classes(img)[0]]}")
```

```
The Photo is in the category of tulips
```

Figure 4.5.7.1: Predicting the tulips.

- ➢ A tulips image is taken from the google and the preprocessing steps such as Scaling and Labelling is done to the image. After applying the preprocessing steps the image is tested with model with predict () method and the category of the flower present image are shown.

4.5.8 PREPROCESSING AND PREDICTING THE SUNFLOWER:



```
img = image.load_img('/content/drive/My Drive/flowers/sunflower.jpg')
print(type(img))
plt.imshow(img)
#print(img.shape)

img = tf.keras.preprocessing.image.img_to_array(img)
print(img.shape)
print(type(img))
img = tf.image.resize(img,(150,150))

#scaling
img = img/255
print(img.shape)
img = np.expand_dims(img,axis=0)
print(img.shape)
```

```
<class 'PIL.JpegImagePlugin.JpegImageFile'>
(646, 862, 3)
<class 'numpy.ndarray'>
(150, 150, 3)
(1, 150, 150, 3)
```

```
model.predict(img).round(3)
```

```
array([[0.    , 0.005, 0.    , 0.995, 0.    ]], dtype=float32)
```

```
print(f"The Photo is in the category of {mapping[model.predict_classes(img)[0]]}")
```

```
The Photo is in the category of sunflowers
```

Figure 4.5.8.1: Predicting the Sunflower.

➢ A sunflower image is taken from the google and the preprocessing steps such as Scaling and Labeling is done to the image. After applying the preprocessing steps the image is tested with model with predict () method and the category of the flower present image are shown.

5.CONCLUSION

The dataset consists of 4232 images each of different pixel values. Each of the image can be classified into either of 5 types-> 'Daisy', 'Rose', 'Sunflower', 'Tulip' or 'Dandelion'. I have trained Convolutional Neural Network written in Keras to predict the type of flower on the validation set. Also used Image Data Generator to augment the training set and avoid overfitting problem. After applying the image preprocessing steps and labeling the images are trained and validated with two models in the first model dropout is not used after using drop out the accuracy is increased. In model 2 30 epoch are taken and the batch size of 20.

After the 30 epochs the training accuracy we got is 0.8857 and the validation accuracy is 0.7035 and the losses are 0.3273 and 1.0854. Where as in Model 1 30 epoch are taken and the batch size of 20. After the 30 epochs the training accuracy we got is 0.9742 and the validation accuracy is 0.6116 and the losses are 0.1430 and 3.3822. From the above 2 models we can conclude that model 2 has high accuracy metric when compared to model 1 which shows better classification of flowers. Even though the accuracy is improved by playing with layers further.

6.REFERENCES

- https://towardsdatascience.com/supervised-machine-learning-model-validation-a-step-by-step-approach-771109ae0253
- https://en.wikipedia.org/wiki/Machine_learning
- https://en.wikipedia.org/wiki/Convolutional_neural_network
- https://www.tensorflow.org/tutorials/keras/classification
- https://github.com/eswarsaisuhas/Flowers-Recognition