

# INTRODUCTION

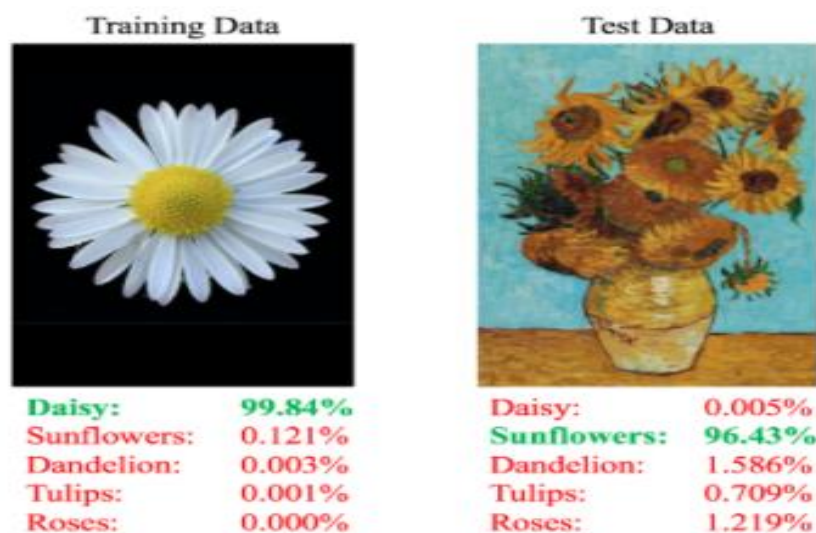
## 1. INTRODUCTION

Flower recognition is a very tedious task. Being able to identify a flower without the need of an expert botanist would be of great help for industries including pharmaceuticals and cosmetics. Due to this a lot of research has been done on this topic. The two main common features of flowers are their color and shape. Those features can be used to train the model such that it can later identify an unknown flower.

### 1.1 INTRODUCTION

Unlike simple object classification such as distinguishing cats from dogs, flower recognition and classification is a challenging task due to the wide range of flower classes that share similar features. Several flowers from different types share similar color, shape and appearance. Furthermore, images of different flowers usually contain similar surrounding objects. Deep learning techniques, especially convolutional neural networks (CNNs), can be chosen as the best option for the training process because it produced a high percentage of accuracy especially in image classification technology.

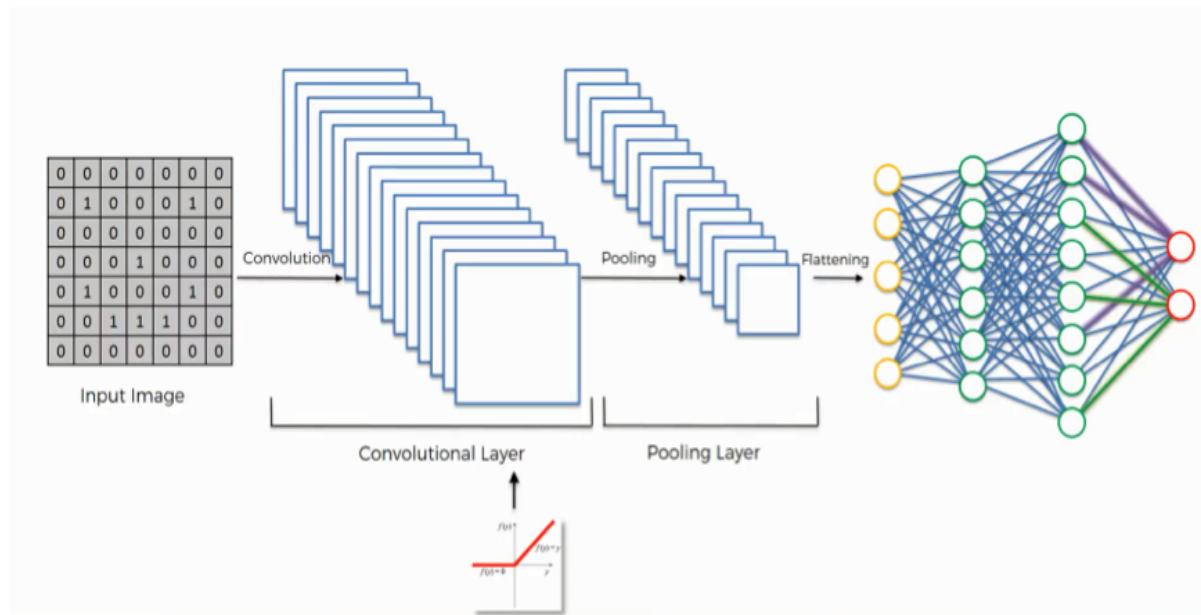
CNN is a type of deep learning model for processing data that has a grid pattern, such as images. The CNN includes convolution layers, pooling layers, and fully connected layers. A typical architecture consists of repetitions of a stack of several convolution layers and a pooling layer, followed by one or more fully connected layers.



**Figure 1.1: Flower Detection**

A convolution layer is a fundamental component that performs feature extraction. A pooling layer provides down sampling operation which reduces the in-plane dimensionality of the feature maps.

When you input an image into a ConvNet, each of its layers generates several activation maps. Activation maps highlight the relevant features of the image. Each of the neurons takes a patch of pixels as input, multiplies their color values by its weights, sums them up, and runs them through the activation function.



**Figure 1.2: Structure of CNN**

The first (or bottom) layer of the CNN usually detects basic features such as horizontal, vertical, and diagonal edges. The output of the first layer is fed as input of the next layer, which extracts more complex features, such as corners and combinations of edges. As you move deeper into the convolutional neural network, the layers start detecting higher-level features such as objects, faces, and more.

Each layer of the neural network will extract specific features from the input image. The operation of multiplying pixel values by weights and summing them is called “convolution” (hence the name convolutional neural network). A CNN is usually composed of several convolution layers, but it also contains other components. The final layer of a CNN is a classification layer, which takes the output of the final convolution layer as input (remember, the higher convolution layers detect complex objects).

Based on the activation map of the final convolution layer, the classification layer outputs a set of confidence scores (values between 0 and 1) that specify how likely the image is to belong to a “class.” For instance, if you have a ConvNet that detects cats, dogs, and horses, the output of the final layer is the possibility that the input image contains any of those animals.

## 1.2 MOTIVATION

There are many types of flower species. It might be hard to identify their names and properties just by looking. The flower classification model will help in immediately identifying the properties of a flower. Development of the recognition of plant species will be advantageous in the fields such as the pharmaceutical industry, botany, agricultural, and trade activities.

## 1.3 PROBLEM DEFINITION:

In recent years, flower classification and detection has been of considerable interest to the computer vision community. The flower detection problem can be defined as a labeling problem based on models of known flowers. Formally, if given an image consisting of a flower, the system should be able to predict correct type of flower in the image with a good accuracy.

## 1.4 OBJECTIVE OF THE PROJECT

The main objective of '**Flower Detection**' is to identify the type of flower based on the analysis different flower images. This can thereby help botanist, agriculturists, pharmaceutical industries etc in their research. The project aims to build a machine learning model with high accuracy such that the flower type can be detected immediately and accurately.

## 1.5 LIMITATIONS OF THE PROJECT

The dataset used for training the model has only five types of flowers i.e. Rose, Sunflower, Daisy, Tulip, and Dandelion. If any other flower image is given for prediction the model will not be able to identify the flower type.

### **1.6 ORGANIZATION OF THE PROJECT:**

The first chapter deals with introduction of flower detection and CNN, motivation for developing this project, objective of the project, limitations of the project. The second chapter deals with the system specifications required for developing the project. It includes hardware & software specifications. The third chapter gives you the preview about literature survey which includes the information about existing system, disadvantages of existing system and proposed system. The fourth chapter is for analysis of the project which includes module organization, feasibility study. The fifth chapter deals with design of the project. The sixth chapter deals with the description of key parameters and functions, implementation of the project which includes source code and result analysis of the project. The seventh chapter tells about the testing and validations. Finally, eight chapter deals with the conclusion and future enhancements.

# **SYSTEM SPECIFICATIONS**

## **2. SYSTEM SPECIFICATIONS:**

Software Requirement Specification is the starting point of the software developing activity. As system grew complex it became evident that the goal of the entire system cannot be easily comprehended. Hence the need for the requirement phase arose. The software is initiated by the client's needs. The SRS is the means of translating the ideas of the minds of the clients (the input) into a formal document (the output of the requirement phase).

### **2.1 SOFTWARE SPECIFICATION**

- Python
- Jupyter Notebook
- Scikitlearn
- Pandas
- Matplotlib
- Numpy
- Seaborn
- OpenCV
- Keras
- Tensorflow

### **PYTHON:**

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse.

### **JUPYTER NOTEBOOK:**

The Jupyter Notebook is an open source web application that you can use to create and share documents that contain live code, equations, visualizations, and text. It is maintained by the people at Project Jupyter. These are a spin-off project from the IPython project, which used to have an IPython Notebook project itself. The name, Jupyter, comes from the core supported programming languages that it supports: Julia, Python, and R.

### **SCIKITLEARN:**

Sklearn is a Python module integrating classical machine learning algorithms in the tightly-knit world of scientific Python packages (numpy, scipy, matplotlib). It aims to provide simple and efficient solutions to learning problems that are accessible to everybody and reusable in various contexts: machine-learning as a versatile tool for science and engineering. The sklearn library contains a lot of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction.

### **PANDAS:**

Pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series. It is free software released under the three-clause BSD license. Pandas is mainly used for data analysis. Pandas allows importing data from various file formats such as comma-separated values, JSON, SQL, Microsoft Excel. Pandas allows various data manipulation operations such as merging, reshaping, selecting, as well as data cleaning, and data wrangling features.

### **NUMPY:**

Numpy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays. It can be utilized to perform a number of mathematical operations on arrays such as trigonometric, statistical, and algebraic routines.

### **MATPLOTLIB:**

Matplotlib is a plotting library for the Python programming language and it has numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose Graphical User Interface toolkits like Tkinter, wxPython, Qt, or GTK+. matplotlib. Pyplot is a collection of command style functions that make matplotlib work like MATLAB. Each pyplot function makes some change to a figure: e.g., creates a figure, creates a plotting area in a figure, plots some lines in a plotting area, decorates the plot with labels. Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python.



### **SEABORN:**

Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics. Seaborn helps you explore and understand your data. Its plotting functions operate on data frames and arrays containing whole datasets and internally perform the necessary semantic mapping and statistical aggregation to produce informative plots. Its dataset-oriented, declarative API lets you focus on what the different elements of your plots mean, rather than on the details of how to draw them.

### **KERAS:**

Keras is an open-source software library that provides a Python interface for artificial neural networks. Keras acts as an interface for the TensorFlow library. Keras contains numerous implementations of commonly used neural-network building blocks such as layers, objectives, activation functions, optimizers, and a host of tools to make working with image and text data easier to simplify the coding necessary for writing deep neural network code.

### **TENSORFLOW:**

TensorFlow is an open source library for fast numerical computing. It was created and is maintained by Google and released under the Apache 2.0 open source license. The API is nominally for the Python programming language, although there is access to the underlying C++ API.

## 2.2 HARDWARE SPECIFICATIONS

The most common set of requirements defined by any operating system or software application is the physical computer resources, also known as hardware. A hardware requirements list is often accompanied by a hardware compatibility list (HCL), especially in case of operating systems. An HCL lists tested, compatible, and sometimes incompatible hardware devices for a particular operating system or application. The following sub-sections discuss the various aspects of hardware requirements. They are peripherals, secondary storage, processing power, memory.

### Requirements

- Processor                    -        Pentium-IV
- Speed                        -        1.1 GHZ
- RAM                         -        512 MB(min)
- Hard Disk                  -        20 GB

# LITERATURE SURVEY

## **3. LITERATURE SURVEY**

Literature survey includes the papers referred for building the system.

### **3.1 PAPERS REFERRED**

- A study on Image Classification based on Deep Learning and Tensorflow
- Image Augmentation on the fly using Keras ImageDataGenerator
- Understanding of a Convolutional Neural Network

### **3.2 PAPER ON IMAGE CLASSIFICATION**

#### **3.2.1 Introduction**

Recently, image classification is growing and becoming a trend among technology developers especially with the growth of data in different parts of industry such as e-commerce, automotive, healthcare, and gaming. One of the dominant approaches for this technology is deep learning. Deep learning falls under the category of Artificial Intelligence where it can act or think like a human. Normally, the system itself will be set with hundreds or maybe thousands of input data in order to make the ‘training’ session to be more efficient and fast. It starts by giving some sort of ‘training’ with all the input data.

Machine learning is also the frequent system that has been applied towards image classification. However, there are still parts that can be improved within machine learning. Therefore, image classification is going to be occupied with deep learning system. Machine Vision has its own context when it comes with Image Classification. The ability of this technology is to recognize people, objects, places, action and writing in images. The combination of artificial intelligence software and machine vision technologies can achieve the outstanding result of image classification.

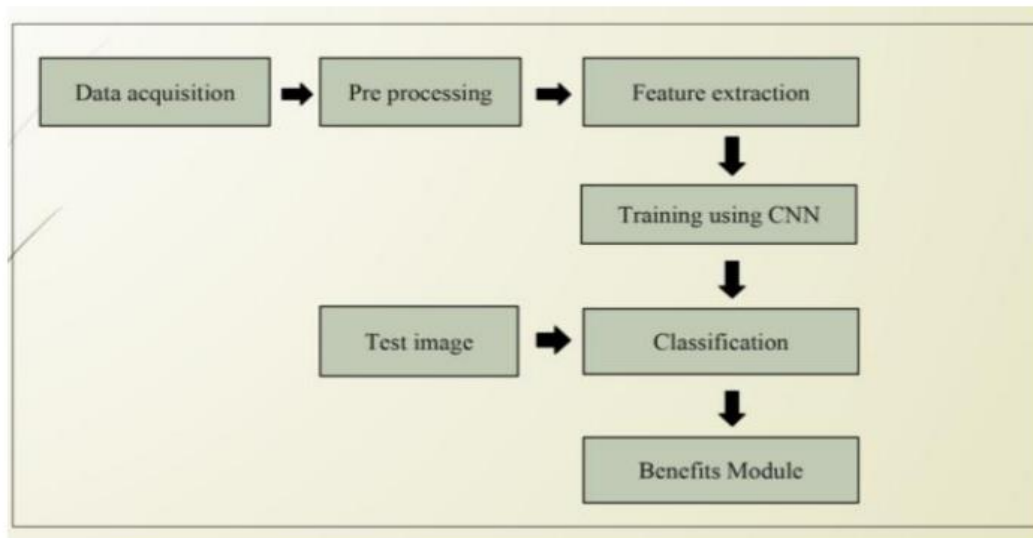
The fundamental task of image classification is to make sure all the images are categorized according to its specific sectors or groups. It consists of unidentified patterns compared to detecting an object as it should be classified to the proper categories. Image classification has become a major challenge in machine vision and has a long history with it. The challenge includes a broad intra-class range of images caused by color, size, environmental conditions and shape.

Deep learning is a subfield of machine learning that is a set of algorithms that is inspired by the structure and function of the brain. TensorFlow is used to design, build, and train deep learning models. TensorFlow library can be used to do numerical computations,

which in itself doesn't seem all too special, but these computations are done with data flow graphs. In these graphs, nodes represent mathematical operations, while the edges represent the data, which usually are multidimensional data arrays or tensors that are communicated between these edges.

### 3.2.2 Methodology

The figure shows the framework of image classification where deep neural networks are also applied. These are the main phases throughout this process and each of the phases will be discussed. Each of the phases includes TensorFlow as the open source software and Python as its programming language.



**Figure 3.1: Block diagram for Image classification**

### 3.2.3 Training data

Input data for this paper mainly uses thousands of images. All of these images are taken from ImageNet. ImageNet also was known as Large Scale Visual Recognition Challenge where it is a competition about detecting and classified thousands of object into its categories. It is free. It meant to be used by the researcher or engineers. This research paper solely focuses on classify flowers into each of its categories. There are thousands of flower images and it has five types of flowers here. Each type of flowers contains hundreds of images with different side and also colors. The total number for all of these flowers is 4323 images as shown in the figure

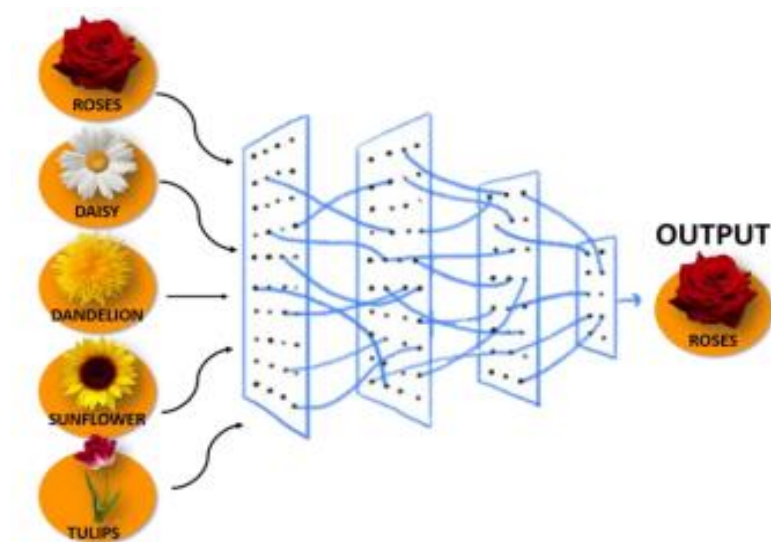
## FLOWER DETECTION



**Figure 3.2: Number of images according to the type of flower**

### 3.2.4 Implementation Deep Neural Network (DNN)

As shown in the figure, it consists of five (5) data inputs (five types of different flowers) and undergoes training with multiple hidden layers. The inputs are also set with fixed-size of the 150x150 RGB image. The convolution process is configured with multiple layers as it produces an efficient convolution neural network.



**Figure 3.3: CNN process for flower images**

The roles of epochs in CNN is able to control accuracy and also prevent any problems such as over fitting. An epoch means training the neural network with all the training data for one cycle.

Implementation of deep learning by using framework TensorFlow gives good results as it is able to simulate well. Python can be used as the programming language since it comes together with TensorFlow framework.

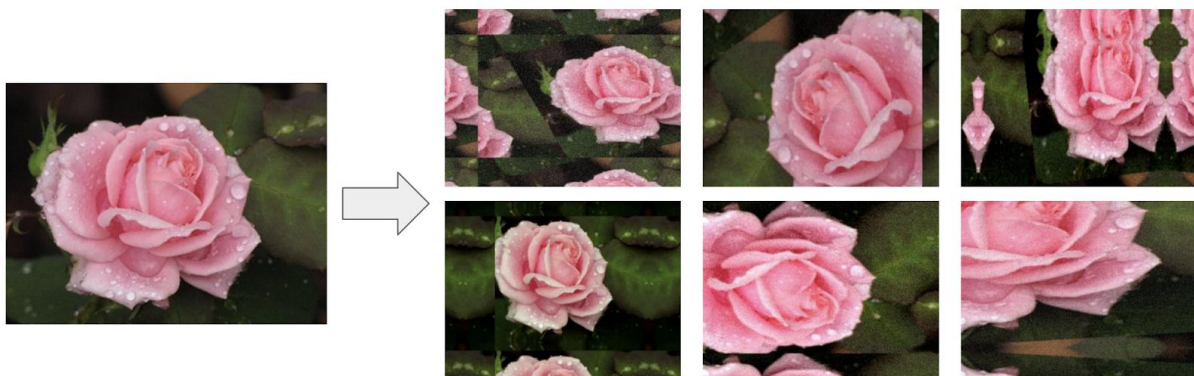
### 3.3 PAPER ON IMAGE AUGMENTATION

#### 3.3.1 Image augmentation

Image augmentation is a technique of applying different transformations to original images which results in multiple transformed copies of the same image. Each copy, however, is different from the other in certain aspects depending on the augmentation techniques you apply like shifting, rotating, flipping, etc.

Applying these small amounts of variations on the original image does not change its target class but only provides a new perspective of capturing the object in real life. And so, we use it is quite often for building deep learning models.

These image augmentation techniques not only expand the size of your dataset but also incorporate a level of variation in the dataset which allows your model to generalize better on unseen data. Also, the model becomes more robust when it is trained on new, slightly altered images.



**Figure 3.4: Image augmentation**

### 3.3.2 Image augmentation in Keras

Keras ImageDataGenerator class provides a quick and easy way to augment your images. It provides a host of different augmentation techniques like standardization, rotation, shifts, flips, brightness change, and many more.

However, the main benefit of using the Keras ImageDataGenerator class is that it is designed to provide real-time data augmentation. Meaning it is generating augmented images on the fly while your model is still in the training stage.

ImageDataGenerator class ensures that the model receives new variations of the images at each epoch. But it only returns the transformed images and does not add it to the original corpus of images.

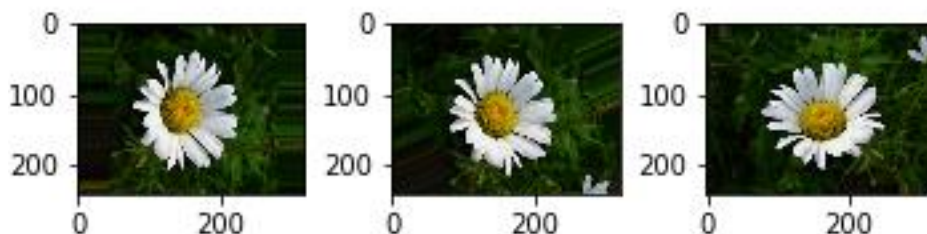
Another advantage of ImageDataGenerator is that it requires lower memory usage. This is so because without using this class, we load all the images at once. But on using it, we are loading the images in batches which save a lot of memory.

### 3.3.3 Image augmentation techniques

#### Random Rotations

Image rotation is one of the widely used augmentation techniques and allows the model to become invariant to the orientation of the object. ImageDataGenerator class allows you to randomly rotate images through any degree between 0 and 360 by providing an integer value in the rotation\_range argument.

When the image is rotated, some pixels will move outside the image and leave an empty area that needs to be filled in. You can fill this in different ways like a constant value or nearest pixel values, etc. This is specified in the fill\_mode argument and the default value is “nearest” which simply replaces the empty area with the nearest pixel values.



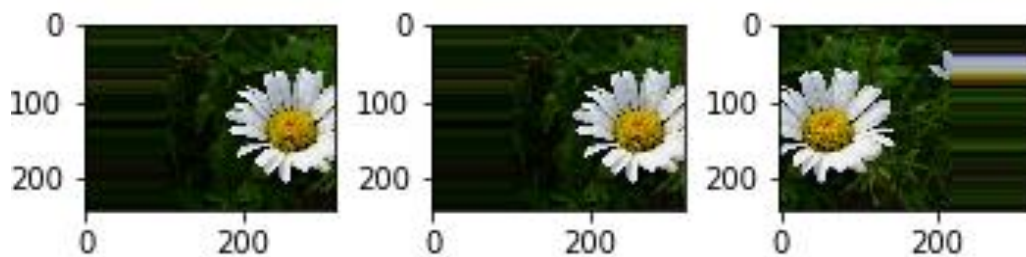
**Figure 3.5: Augmented images generated with a random rotation**



### Random Shifts

It may happen that the object may not always be in the center of the image. To overcome this problem we can shift the pixels of the image either horizontally or vertically; this is done by adding a certain constant value to all the pixels.

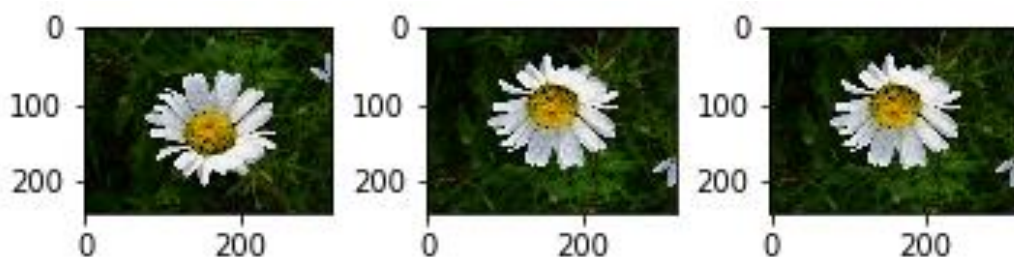
ImageDataGenerator class has the argument `height_shift_range` for a vertical shift of image and `width_shift_range` for a horizontal shift of image. If the value is a float number, that would indicate the percentage of width or height of the image to shift. Otherwise, if it is an integer value then simply the width or height are shifted by those many pixel values.



**Figure 3.6: Augmented images generated with a random horizontal shift**

### Random Flips

ImageDataGenerator class has parameters `horizontal_flip` and `vertical_flip` for flipping along the vertical or the horizontal axis. However, this technique should be according to the object in the image. For example, vertical flipping of a car would not be a sensible thing compared to doing it for a symmetrical object like football or something else. Having said that, I am going to flip my image in both ways just to demonstrate the effect of the augmentation.

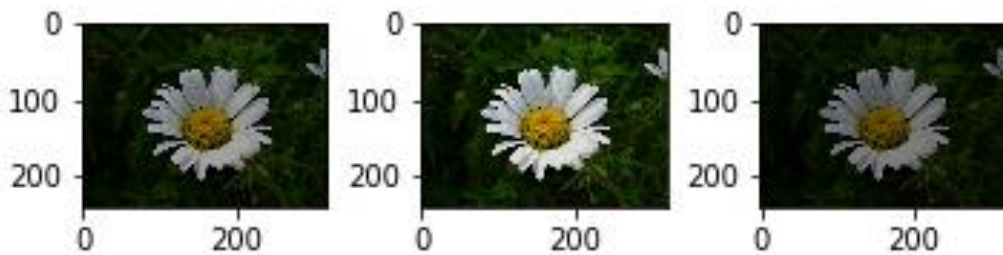


**Figure 3.7: Augmented images generated with a random vertical flip**

### Random Brightness

It randomly changes the brightness of the image. It is also a very useful augmentation technique because most of the times our object will not be under perfect lighting condition. So, it becomes imperative to train our model on images under different lighting conditions.

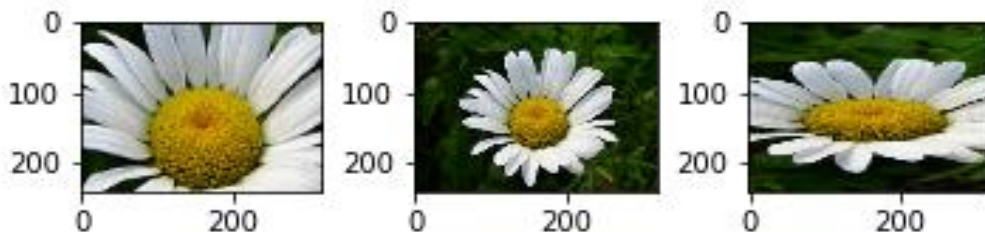
Brightness can be controlled in the ImageDataGenerator class through the brightness\_range argument. It accepts a list of two float values and picks a brightness shift value from that range. Values less than 1.0 darken the image, whereas values above 1.0 brighten the image.



**Figure 3.8: Augmented images generated with a random brightness**

### Random Zoom

The zoom augmentation either randomly zooms in on the image or zooms out of the image. ImageDataGenerator class takes in a float value as input for zooming in the zoom\_range argument. You could provide a list with two values specifying the lower and the upper limit. Else, if you specify a float value, then zoom will be done in the range  $[1 - \text{zoom\_range}, 1 + \text{zoom\_range}]$ . Any value smaller than 1 will zoom in on the image. Whereas any value greater than 1 will zoom out on the image.

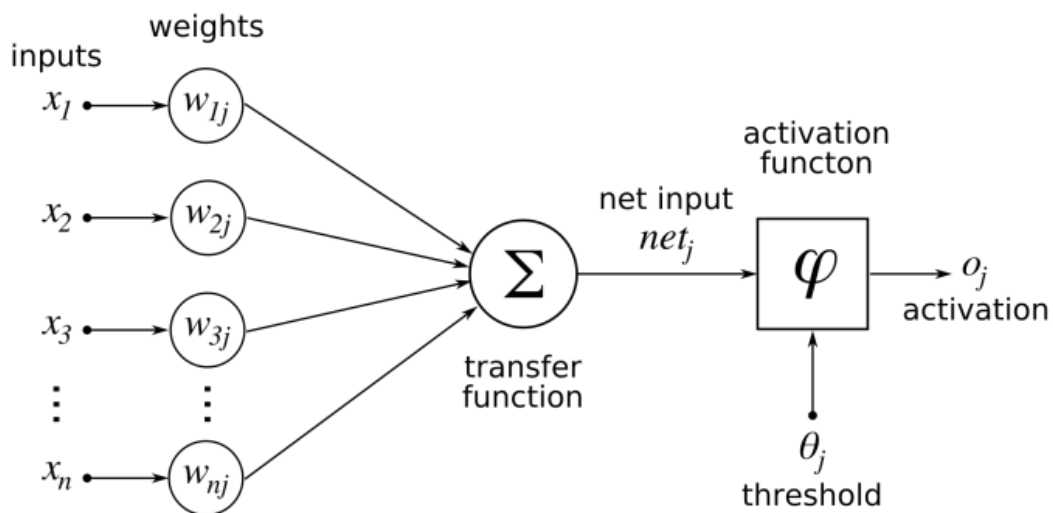


**Figure 3.9: Augmented images generated with a random zoom**

## 3.4 PAPER ON CONVOLUTIONAL NEURAL NETWORKS

### 3.4.1 Working of CNN

Convolutional neural networks are composed of multiple layers of artificial neurons. Artificial neurons, a rough imitation of their biological counterparts, are mathematical functions that calculate the weighted sum of multiple inputs and output an activation value. The behavior of each neuron is defined by its weights. When fed with the pixel values, the artificial neurons of a CNN pick out various visual features.

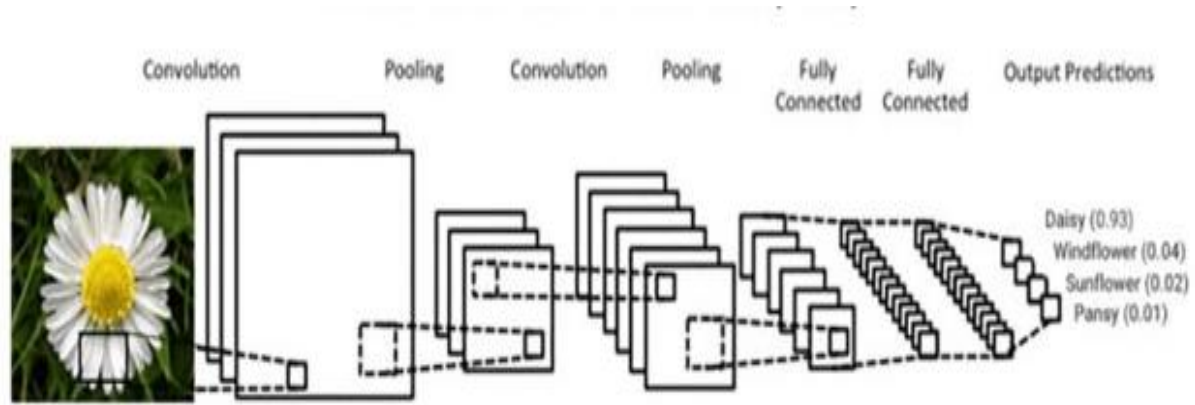


**Figure 3.10: Structure of artificial neuron**

It is a Deep Learning algorithm which can take in an input image, assign importance to various aspects in the image and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics.

A Classic CNN consists of:

1. Convolutional Layer.
2. Activation operation following each convolutional layer.
3. Pooling layer especially Max Pooling layer and also others based on the requirement.
4. Finally Fully Connected Layer.



**Figure 3.11: Layers in CNN**

The most important layer in CNN is convolution layer which takes most of the time within the network. Important aspects related to Convolutional Neural Network (CNN) and the effect each parameter such as filters, padding, strides etc on performance of network.

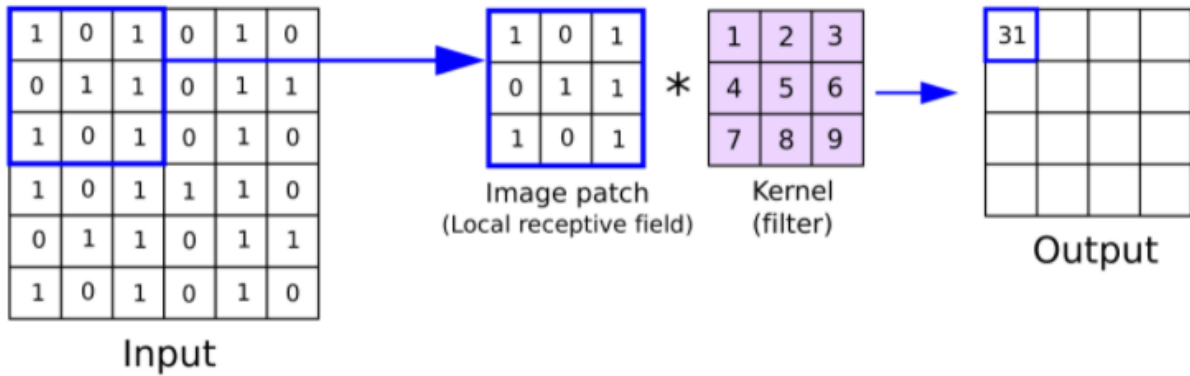
### 3.4.2 Filters in Conv2D

A filter or a kernel in a conv2D layer has a height and a width. They are generally smaller than the input image and so we move them across the whole image. The area where the filter is on the image is called the receptive field. Conv2D filters extend through the three channels in an image (Red, Green, and Blue). The filters may be different for each channel too. After the convolutions are performed individually for each channel, they are added up to get the final convoluted image. The output of a filter after a convolution operation is called a feature map.

Each filter in this layer is randomly initialized to some distribution (Normal, Gaussian, etc.). By having different initialization criteria, each filter gets trained slightly differently. They eventually learn to detect different features in the image. If they were all initialized similarly, then the chances of two filters learning similar features increase dramatically. Random initialization ensures that each filter learns to identify different features.

Since each conv2D filter learns a separate feature, we use many of them in a single layer to identify different features. The best part is that every filter is learnt automatically.

## FLOWER DETECTION



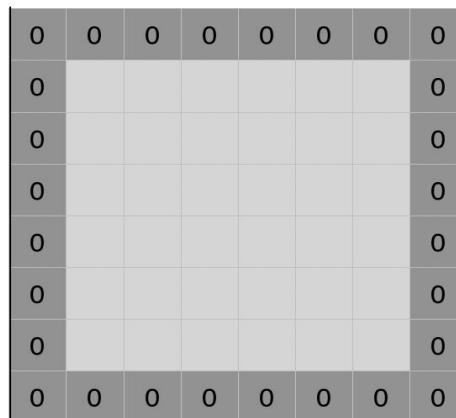
**Figure 3.12: Filter in Conv2D layer**

Each of these filters are used as inputs to the next layer in the neural network. If there are 8 filters in the first layer and 32 in the second, then each filter in the second layer sees 8 filter inputs. Meaning that we get 32X8 feature maps in the second layer. Each of the 8 feature maps of a single filter are added to get a single output from each layer.

### 3.4.3 Padding in Conv2D

Padding is simply a process of adding layers of zeros to our input images so as to avoid the problems mentioned above. This prevents shrinking as, if  $p$  = number of layers of zeros added to the border of the image, then our  $(n \times n)$  image becomes  $(n + 2p) \times (n + 2p)$  image after padding. So, applying convolution-operation (with  $(f \times f)$  filter) outputs  $(n + 2p - f + 1) \times (n + 2p - f + 1)$  images.

For example, adding one layer of padding to an  $(8 \times 8)$  image and using a  $(3 \times 3)$  filter we would get an  $(8 \times 8)$  output after performing convolution operation.



Zero-padding added to image

**Figure 3.13: Padding in CNN**

This increases the contribution of the pixels at the border of the original image by bringing them into the middle of the padded image. Thus, information on the borders is preserved as well as the information in the middle of the image.

### Types of Padding

1. Valid Padding: It implies no padding at all. The input image is left in its valid/unaltered shape.
2. Same Padding: In this case, we add 'p' padding layers such that the output image has the same dimensions as the input image.

### 3.4.4 Strides in CNN

Stride is a parameter of the neural network's filter that modifies the amount of movement over the image or video. For example, if a neural network's stride is set to 1, the filter will move one pixel, or unit, at a time. The size of the filter affects the encoded output volume, so stride is often set to a whole integer, rather than a fraction or decimal.

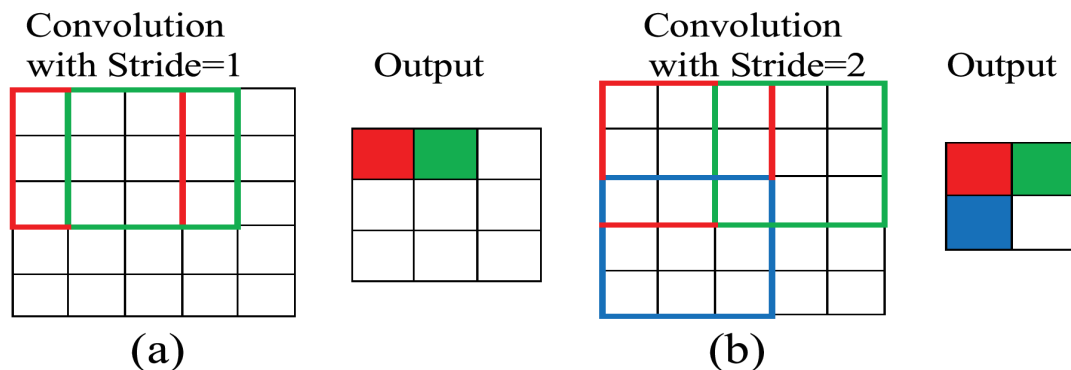
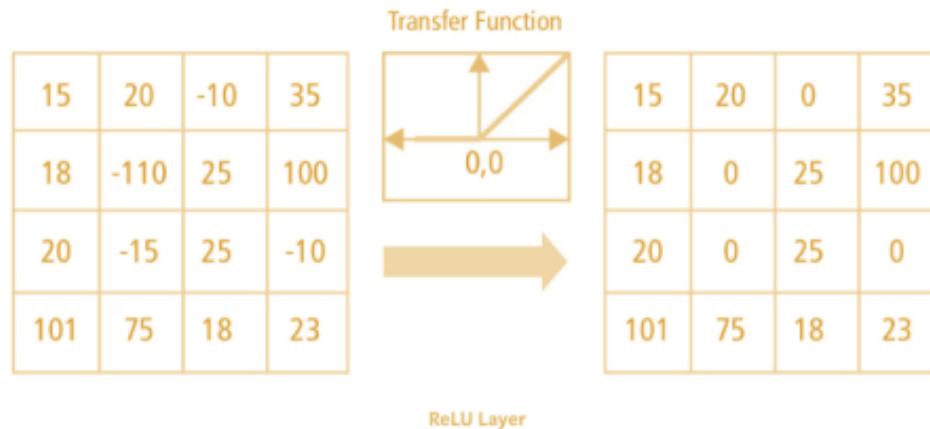


Figure 3.14: Strides in CNN

### 3.4.5 Non Linearity (ReLU)

ReLU stands for Rectified Linear Unit for a non-linear operation. The output is  $f(x) = \max(0, x)$ . ReLU's purpose is to introduce non-linearity in our ConvNet. Since, the real world data would want our ConvNet to learn would be non-negative linear values. There are other non linear functions such as tanh or sigmoid that can also be used instead of ReLU. Most of the data scientists use ReLU since performance wise ReLU is better than the other two.



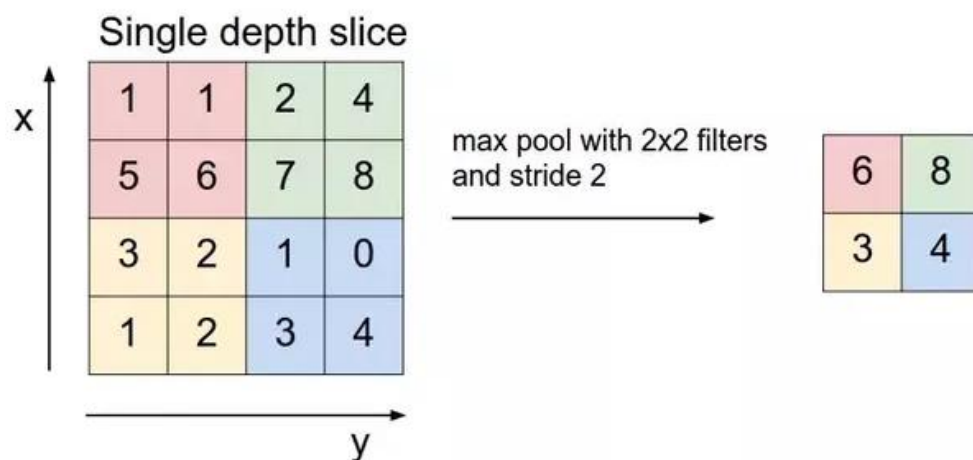
**Figure 3.15: ReLU operation**

### 3.4.6 Pooling Layer

Pooling layers section would reduce the number of parameters when the images are too large. Spatial pooling also called sub sampling or down sampling which reduces the dimensionality of each map but retains important information. Spatial pooling can be of different types:

1. Max Pooling
2. Average Pooling
3. Sum Pooling

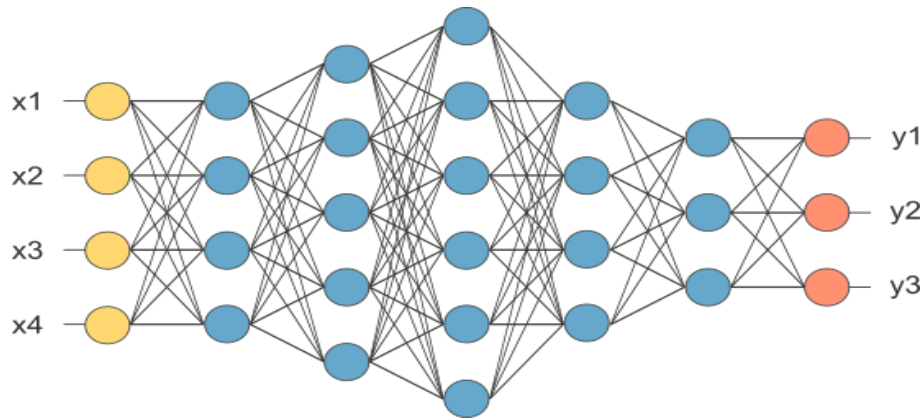
Max pooling takes the largest element from the rectified feature map. Taking the largest element could also take the average pooling. Sum of all elements in the feature map call as sum pooling.



**Figure 3.16: Max Pooling**

### 3.4.7 Fully Connected Layer

The layer we call as FC layer, we flattened our matrix into vector and feed it into a fully connected layer like a neural network.



**Figure 3.17: Fully Connected Layer**

In the above diagram, the feature map matrix will be converted as vector ( $x_1, x_2, x_3, \dots$ ). With the fully connected layers, we combined these features together to create a model. Finally, we have an activation function such as softmax or sigmoid to classify the outputs.

Network performance also depends on the number of levels within the network. But in the other hand as the number of levels increases the time required to train and test the network. Today the CNN consider as power full tool within machine learning for a lot of application such as face detection and image , video recognitions and voice recognition.



# ANALYSIS

## **4. ANALYSIS**

Analysis involves how the system is being programmed? What are the specific requirements for the designing of system? What all problems exist in the present system? What must be done to solve the problem in a efficient way? module organization, requirement analysis, feasibility study.

### **4.1 MODULE ORGANISATION**

- Importing the required packages into our python environment.
- Loading the data
- Processing the data to our needs
- Exploratory data Analysis
- Building the Sequential model and adding layers
- Data Split and Feature Selection using CNN layers
- Compiling and fitting the model
- Making predictions
- Evaluating the models using the evaluation metric

### **4.2 FEASIBILITY STUDY**

Before jumping right into your new modeling task, consider performing a feasibility study. A feasibility study of a predictive model will answer key questions that can help you decide if the modeling task is likely to succeed. A **feasibility study** is an assessment of the practicality of a proposed project or system.

#### **Questions to Ask During a Feasibility Study**

The following template can be used while doing a feasibility study for a predictive model.

1. Training Data — Does training data need to be collected? If so, how much time and money will it cost?
2. Predictive Features — According to domain experts, what factors are likely to predict the target variable? Is that data accessible to you?
3. Data Sources — What data sources will you need to gain access to? If internal, do you have support from data engineers? If external, how much will vendor data cost?

4. Production — what is the level of effort to develop, deploy, and maintain your model in production.

### **Training Data**

The data used for training the model need not be collect manually. It is already available in Kaggle website. As reusing a data set that already exists will not cost time or money.

### **Predictive Features**

The features to predict the type of the image are the color, edges, curves related to the particular image. All these can be detected from the images in the dataset. So, we have all the predictive features and as we are using neural networks these will be automatically extracted by the algorithm.

### **Production**

The main problem in production can be the amount of time taken for training the algorithm. To avoid this waiting time a model is pre trained with all the images and using in real time the already trained model is loaded to make the predictions.

### **Market Demand**

Various cosmetic industries and pharmaceutical industries depend on various types of flowers. The model can be used by these industries for choosing the correct type of flower according to their need. The model would be useful for agriculturists or botanists for their research.

So, It is feasible to build a flower detection model.

# DESIGN

## **5. DESIGN**

### **5.1 INTRODUCTION**

Design is an early phase of the project where a project's key features, structure, criteria for success, and major deliverables are all planned out. The aim is to develop one or more designs that can be used to achieve the desired project goals.

Unified Modeling Language (UML) is a general purpose modeling language. The main aim of UML is to define a standard way to visualize the way a system has been designed. It is quite similar to blueprints used in other fields of engineering.

UML is not a programming language; it is rather a visual language. We use UML diagrams to portray the behavior and structure of a system. UML helps software engineers, businessmen and system architects with modeling, design and analysis.

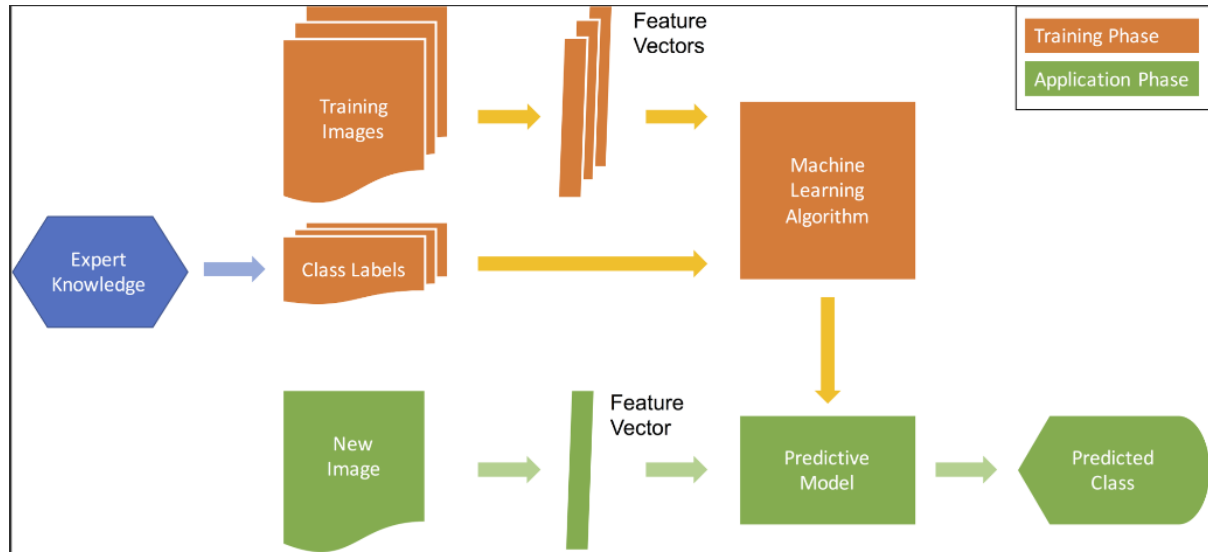
Data flow diagrams are used to graphically represent the flow of data in a business information system. DFD describes the processes that are involved in a system to transfer data from the input to the file storage and reports generation.

DFD graphically representing the functions, or processes, which capture, manipulate, store, and distribute data between a system and its environment and between components of a system. The visual representation makes it a good communication tool between User and System designer. Structure of DFD allows starting from a broad overview and expands it to a hierarchy of detailed diagrams.

Entity relationship modeling enables us to describe business information (data) and its inherent structure. The entity relationship model is represented as a diagram, known as the entity relationship diagram (ERD). This diagram is used to show our understanding of data and is one of the components of the information architecture. This model is also used in later stages of business system design and development

## 5.2 UML DIAGRAMS

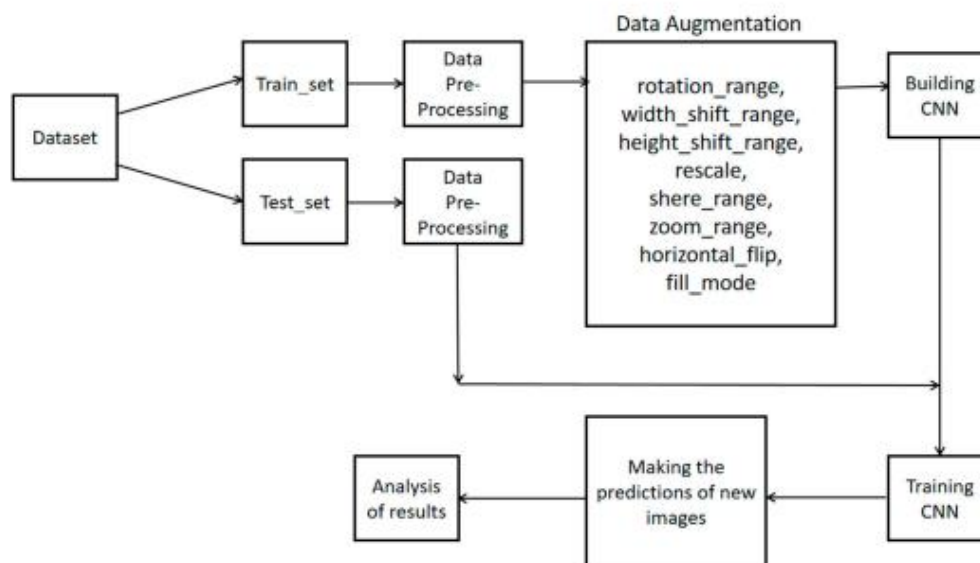
From a deep learning perspective, flower identification is a supervised classification problem, as outlined in Figure 5.1.



**Figure 5.1: Supervised classification structure for flower classification.**

## Framework

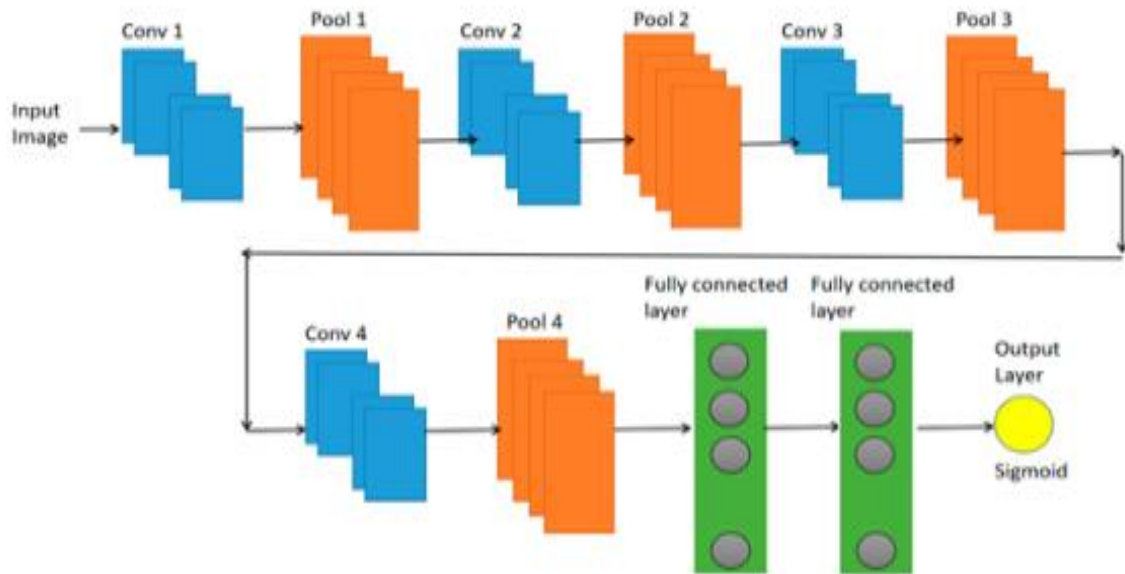
The construction of the model and the framework of the model is shown in Figure 5.2



**Figure 5.2: The overall framework of building the model.**

## Algorithm

We applied a Deep Learning technique called Convolutional Neural Networks (CNN). We have run the code on Convolutional Neural Networks with and without regularization techniques. The model is shown in Figure 5.3. For CNN without regularization, we used four convolutional layers, four pooling layers. Data augmentation is used to avoid over fitting. Two activation functions, ReLU and Softmax, are used. ReLU is used in the inner layers of CNN and Softmax is used in the output layer of CNN. Adaptive momentum estimation (Adam) is used for optimization and Max-pooling is used in the pooling layer. 2. For CNN with regularization, we used five CNN layers and five pooling layers. ReLU and Softmax activation functions are used. Adam Optimizer, Max-pooling, and L2 regularization with different weight functions are used for the evaluation. We calculate the accuracy every time we run the code and the best accuracy on the test set is taken into the account.



**Figure 5.3: Convolutional Neural Network**

## 5.3 MODULE DESIGN and ORGANISATION

### Dataset Collection

Kaggle supports a variety of dataset publication formats. The flower images are taken from Kaggle. The dataset has five types of flower images:

- Rose
- Dasiy
- Tulip
- Sunflower
- Dandelion

### Data Preprocessing

Data preprocessing is a data mining technique which is used to transform the raw data in a useful and efficient format. Data preprocessing is an important step to prepare the data to form a CNN model. There are many steps in data preprocessing, such as data cleaning, data transformation, and feature selection. Data cleaning and transformation are methods used to remove outliers and standardize the data so that they take a form that can be easily used to create a model.

### Data Exploration

Data exploration is an approach similar to initial data analysis, whereby a data analyst uses visual exploration to understand what is in a dataset and the characteristics of the data, rather than through traditional data management systems. These characteristics can include size or amount of data, completeness of the data, correctness of the data, possible relationships amongst data elements or files/tables in the data.

### Feature Selection

Feature Selection is the process where you automatically or manually select those features which contribute most to your prediction variable or output in which you are interested in. Having irrelevant features in your data can decrease the accuracy of the models and make your model learn based on irrelevant features.



## **Data Split**

The train-test split is a technique for evaluating the performance of a machine learning algorithm. It can be used for classification or regression problems and can be used for any supervised learning algorithm. The procedure involves taking a dataset and dividing it into two subsets. The first subset is used to fit the model and is referred to as the training dataset. The second subset is not used to train the model; instead, the input element of the dataset is provided to the model, then predictions are made and compared to the expected values. This second dataset is referred to as the test dataset. Train Dataset: Used to fit the machine learning model. Test Dataset: Used to evaluate the fit machine learning model. The objective is to estimate the performance of the machine learning model on new data: data not used to train the model.

## **Building Flower detection Systems**

Build a sequential CNN model and add layers to the model. The first layer should be Convolutional layer and input should be given to this layer. The second layer will be Max pooling layer to reduce the amount of computation. Next we add a stack of same layers. Then we add Flatten and Dense layers. Compile the model using Adam optimizer. Fit the model using train data. Predict using test data and a new image. Performance is evaluated using confusion matrix and accuracy score.

# **IMPLEMENTATION AND RESULT ANALYSIS**

## **6. IMPLEMENTATION AND RESULT ANALYSIS**

### **6.1 INTRODUCTION**

In order to implement flower classification system, the Flower Recognition Dataset from Kaggle is used. The dataset consists of five folders namely Rose, Tulip, Daisy, Dandelion, Sun flower, which are used in Flower Detection Project. This data consists of 4323 flower images. For each type of flower there are about 800 photos. Photo resolution is about 320x240 pixels.

### **6.2 DESCRIPTION OF KEY PARAMETES AND FUNCTIONS**

#### **Sequential ( )**

It allows you to build a model layer by layer. We use the 'add ()' function to add layers to our model. A Sequential model is appropriate for a plain stack of layers where each layer has exactly one input tensor and one output tensor.

#### **Conv2D ( )**

Keras Conv2D is 2D Convolution Layer; this layer creates a convolution kernel that is wind with layers input which helps produce a tensor of outputs. In image processing kernel is a convolution matrix or masks which can be used for blurring, sharpening, embossing, edge detection, and more by doing a convolution between a kernel and an image. Important aspects related to Convolutional Neural Network (CNN) and the effect each parameter such as filters, padding, strides etc on performance of network.

#### **MaxPooling2D ( )**

It is a pooling operation that selects the maximum element from the region of the feature map covered by the filter. The output would be a feature map containing the most prominent features of the previous feature map. It performs dimensionality reduction.

#### **Flatten ( )**

Flatten is the function that converts the pooled feature map to a single column that is passed to the fully connected layer. It serves as a connection between the convolution and dense layer.

### **Activation ( )**

The activation function is a node that is put at the end of or in between Neural Networks. They help to decide if the neuron would fire or not. We have different types of activation functions; we will be on Rectified Linear Unit (ReLU).

### **Dense ( )**

Dense is a standard output layer type that is most commonly used in many cases for neural networks. In the dense layer we will have 5 nodes which is our output layer, one for each possible outcome (0–4). The activation is ‘softmax’. Softmax makes the output sum up to 1 so the output can be interpreted as probabilities. The model will then make its prediction based on which option has the highest probability.

### **ImageDataGenerator ( )**

The image augmentation technique is a great way to expand the size of your dataset. You can generate new transformed images from the old dataset. Keras *ImageDataGenerator* allows to augment images in real-time while model is getting trained. Random transformations can be applied on each training image as it is passed to the model. This will not only make your model robust but will also save up on the overhead memory.

### **Confusion matrix**

A Confusion matrix is an  $N \times N$  matrix used for evaluating the performance of a classification model, where  $N$  is the number of target classes. The matrix compares the actual target values with those predicted by the machine learning model.

### **Accuracy Score**

Accuracy is one metric for evaluating classification models. Informally, accuracy is the fraction of predictions our model got right. Formally, accuracy has the following definition:  $\text{Accuracy} = \text{Number of correct predictions} / \text{Total number of predictions}$ . The accuracy of the model is 80%.

## 6.3 METHOD OF IMPLEMENTATION

### Attaining of images

Flower images are taken from “Flowers Recognition” dataset in Kaggle. These flower images are converted into matrices using Numpy.

### Assigning labels

Each image is loaded from its path and reshaped to 150X150 using cv2. The reshaped image is converted into numpy array and stored. The corresponding label for that image is also stored.

### Label encoding

Label Encoding refers to converting the labels into numeric form so as to convert it into the machine-readable form. Machine learning algorithms can then decide in a better way on how those labels must be operated. It is an important pre-processing step for the structured dataset in supervised learning. Label encoding convert the data in machine readable form, but it assigns a unique number(starting from 0) to each class of data.

### Normalizing input features

The input features are pixel values of each image. As the images used are in RGB format, the pixel values will be in the range of 0 to 255. So, dividing all the values of input by 255 will normalize the values into the range of 0 to 1

### Convolutional Neural Network

CNN is a type of deep learning model for processing data that has a grid pattern, such as images. The CNN includes convolution layers, pooling layers, and fully connected layers. A typical architecture consists of repetitions of a stack of several convolution layers and a pooling layer, followed by one or more fully connected layers. A convolution layer is a fundamental component that performs feature extraction. A pooling layer provides down sampling operation which reduces the in-plane dimensionality of the feature maps.

## 6.4 SOURCE CODE

### Printing all the folder names in the data set

```
import os

#Getting all the folders (types of flowers)

labels= os.listdir('D:/Major Project/flowers')

print(labels)
```

### Declaring necessary variables and constants

```
X=[] #list of features of each image

Z=[] #list of corresponding lable

IMG_SIZE=150

#Getting directory paths of each type of flower

FLOWER_DAISSY_DIR='D:/Major Project/flowers/daisy'

FLOWER_SUNFLOWER_DIR='D:/Major Project/flowers/sunflower'

FLOWER_TULIP_DIR='D:/Major Project/flowers/tulip'

FLOWER_DANDI_DIR='D:/Major Project/flowers/dandelion'

FLOWER_ROSE_DIR='D:/Major Project/flowers/rose'
```

### Function for data preprocessing

```
#can process images and videos

import cv2

#For converting image to array of pixels

import numpy as np

#allows to display progress bar with elapsed time and estimated remaining time

from tqdm import tqdm
```

## FLOWER DETECTION

---

#takes the flower type and its directory

def make\_train\_data(flower\_type,DIR):

    for img in tqdm(os.listdir(DIR)): #iterating over each flower by getting name

        path = os.path.join(DIR,img) #getting path

        img = cv2.imread(path,cv2.IMREAD\_COLOR) #getting image from the path

        img = cv2.resize(img, (IMG\_SIZE,IMG\_SIZE)) #resizing to 150X150

        X.append(np.array(img)) #adding image

        Z.append(str(flower\_type)) #adding label of the flower

### **Preprocessing an image and storing it as an array of pixels and assigning labels**

#storing each image as the array of its pixels and assigning labels

make\_train\_data('Daisy',FLOWER\_DAISSY\_DIR)

print(len(X))

make\_train\_data('Sunflower',FLOWER\_SUNFLOWER\_DIR)

print(len(X))

make\_train\_data('Tulip',FLOWER\_TULIP\_DIR)

print(len(X))

make\_train\_data('Dandelion',FLOWER\_DANDI\_DIR)

print(len(X))

make\_train\_data('Rose',FLOWER\_ROSE\_DIR)

print(len(X))

### **Plotting 10 random images from the data set**

import matplotlib.pyplot as plt #For plotting

from matplotlib import style

%matplotlib inline

style.use('fivethirtyeight')

## FLOWER DETECTION

---

```
import random as rn      #For getting a random flower

fig,ax=plt.subplots(5,2)

fig.set_size_inches(15,15)

#getting 10 random images and printing them

for i in range(5):

    for j in range (2):

        l=mn.randint(0,len(Z))    #getting image randomly

        ax[i,j].imshow(X[l])      #Displaying the image

        ax[i,j].set_title('Flower: '+Z[l]) #Setting the lable as images title

plt.tight_layout()
```

### Changing lables of images from String to Integer

```
from sklearn.preprocessing import LabelEncoder #Encode target labels with value between 0
and n_classes-1.
```

```
from keras.utils import to_categorical
```

```
le=LabelEncoder()    #changing lables to 0 to n-1 ie 4
```

```
Y=le.fit_transform(Z)
```

```
Y=to_categorical(Y,5)
```

```
X=np.array(X)        #Normalizing the pixel values
```

```
X=X/255
```

### Split the data into train data and test data

```
from sklearn.model_selection import train_test_split      #for splitting the data
```

```
#splitting the test data and train data
```

```
x_train,x_test,y_train,y_test=train_test_split(X,Y,test_size=0.20,random_state=42)
```



### **Adding layers to Sequential model**

```
from keras.models import Sequential

from keras.layers import Conv2D, MaxPooling2D

from keras.layers import Flatten, Activation

from keras.layers import Dense

# modelling using a CNN.

model = Sequential() #using Sequential model

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

model.add(MaxPooling2D(pool_size=(2,2)))

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(Conv2D(filters =96, kernel_size = (3,3),padding = 'Same',activation ='relu'))

model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))

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(Flatten())

model.add(Dense(512))

model.add(Activation('relu'))

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

### **Image Augmentation**

```
from keras.preprocessing.image import ImageDataGenerator
```

```
datagen = ImageDataGenerator(

    # set input mean to 0 over the dataset

    featurewise_center=False,
```

```
# set each sample mean to 0
samplewise_center=False,

# divide inputs by std of the dataset
featurewise_std_normalization=False,

# divide each input by its std
samplewise_std_normalization=False,

# apply ZCA whitening
zca_whitening=False,

# randomly rotate images in the range (degrees, 0 to 180)
rotation_range=10,

# Randomly zoom image
zoom_range = 0.1,

# randomly shift images horizontally (fraction of total width)
width_shift_range=0.2,

# randomly shift images vertically (fraction of total height)
height_shift_range=0.2,

# randomly flip images
horizontal_flip=True,

# randomly flip images
vertical_flip=False)

datagen.fit(x_train)
```

### **Fitting the model with augmented data**

```
History = model.fit_generator(datagen.flow(x_train,y_train, batch_size=batch_size),
                             epochs = epochs, validation_data = (x_test,y_test),
                             verbose = 1, steps_per_epoch=x_train.shape[0] // batch_size)
```

### Storing the model in JSON format

```
model_json = History.to_json()
with open("History.json", "w") as json_file:
    json_file.write(History_json)
model.save_weights("History.h5")
```

### Loading the stored model

```
from keras.models import model_from_json
#loading stored model
json_file= open('model.json','r')
loaded_model_json= json_file.read()
json_file.close()
model = model_from_json(loaded_model_json)
model.load_weights("model.h5")
print("Loaded")
```

### Compiling the model

```
from keras.optimizers import Adam
model.compile(optimizer=Adam(lr=0.001),loss='categorical_crossentropy',metrics=['accuracy'])
```

### Summary of the model

```
model.summary()
```

### Making predictions on Test data

```
pred=model.predict(x_test)
pred_digits=np.argmax(pred,axis=1)
```

### Storing 8 properly classified and 8 misclassified images

```
i=0

prop_class=[]

mis_class=[]

for i in range(len(y_test)):

    if(np.argmax(y_test[i])==pred_digits[i]):

        prop_class.append(i)

        if(len(prop_class)==8):

            break

i=0

for i in range(len(y_test)):

    if(not np.argmax(y_test[i])==pred_digits[i]):

        mis_class.append(i)

        if(len(mis_class)==8):

            break
```

### Displaying 10 properly classified images

```
count=0

fig,ax=plt.subplots(4,2)

fig.set_size_inches(15,15)

for i in range (4):

    for j in range (2):

        ax[i,j].imshow(x_test[prop_class[count]])

        ax[i,j].set_title("Predicted Flower :
"+str(le.inverse_transform([pred_digits[prop_class[count]]]))+"\n"+"Actual Flower :
"+str(le.inverse_transform([np.argmax(y_test[prop_class[count]]))]))

        plt.tight_layout()

        count+=1
```

### Displaying 10 misclassified images

```
count=0

fig,ax=plt.subplots(4,2)

fig.set_size_inches(15,15)

for i in range (4):

    for j in range (2):

        ax[i,j].imshow(x_test[mis_class[count]])

        ax[i,j].set_title("Predicted Flower :
"+str(1e.inverse_transform([pred_digits[mis_class[count]]]))+"\n"+"Actual Flower :
"+str(1e.inverse_transform([np.argmax(y_test[mis_class[count]]))]))

        plt.tight_layout()

        count+=1
```

### Predicting on a new image

```
from keras.preprocessing import image

test_image = image.load_img('D:/Major Project/Image1.jpg', target_size = (150,150))

test_image = image.img_to_array(test_image)

#test_image.reshape(1,64,64,3)

test_image = np.expand_dims(test_image, axis = 0)

test_image=test_image/255

result =model.predict(test_image)

print(result)

print(labels[result.argmax()])
```

### Confusion matrix

```
import seaborn as sns

sns.set(style='whitegrid',color_codes=True)

from sklearn.metrics import confusion_matrix
```

```
Y_pred = model.predict(x_test)
Y_pred_classes = np.argmax(Y_pred,axis = 1)
Y_true = np.argmax(y_test,axis = 1)
matrix_labels = labels
confusion_mtx = confusion_matrix(Y_true,Y_pred_classes)
f,ax = plt.subplots(figsize = (8,8))
sns.heatmap(confusion_mtx,annot=True,linewidths = 0.01,cmap="Reds",
            xticklabels=labels,yticklabels=labels,
            linecolor = "gray",fmt = ".2f",ax=ax
            )
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)
plt.xlabel("predicted label")
plt.ylabel("True Label")
plt.title("confusion matrix")
plt.show()
```

### **Accuracy score**

```
def acc(y_true, y_pred):
    return np.equal(np.argmax(y_true, axis=-1), np.argmax(Y_pred, axis=-1)).mean()
print("accuracy: " + str(acc(y_test, Y_pred)))
```

## 6.5 OUTPUT SCREENS

Displaying folders of the dataset:

```
import os
#Getting all the folders (types of flowers)
labels= os.listdir('D:/Major Project/flowers')
print(labels)

['daisy', 'dandelion', 'rose', 'sunflower', 'tulip']
```

Figure 6.1: Displaying all the folders present in the dataset

Data Preprocessing:

```
make_train_data('Daisy',FLOWER_DAISY_DIR)
print(len(X))
```



100% | 769/769 [00:06<00:00, 117.02it/s]

769

```
make_train_data('Sunflower',FLOWER_SUNFLOWER_DIR)
print(len(X))
```



100% | 734/734 [00:06<00:00, 108.61it/s]

1503

```
make_train_data('Tulip',FLOWER_TULIP_DIR)
print(len(X))
```



100% | 984/984 [00:08<00:00, 117.13it/s]

2487

```
make_train_data('Dandelion',FLOWER_DANDI_DIR)
print(len(X))
```



100% | 1052/1052 [00:08<00:00, 121.58it/s]

3539

```
make_train_data('Rose',FLOWER_ROSE_DIR)
print(len(X))
```



100% | 784/784 [00:05<00:00, 131.14it/s]

4323

Figure 6.2: Resizing the images and storing image in array format

## FLOWER DETECTION

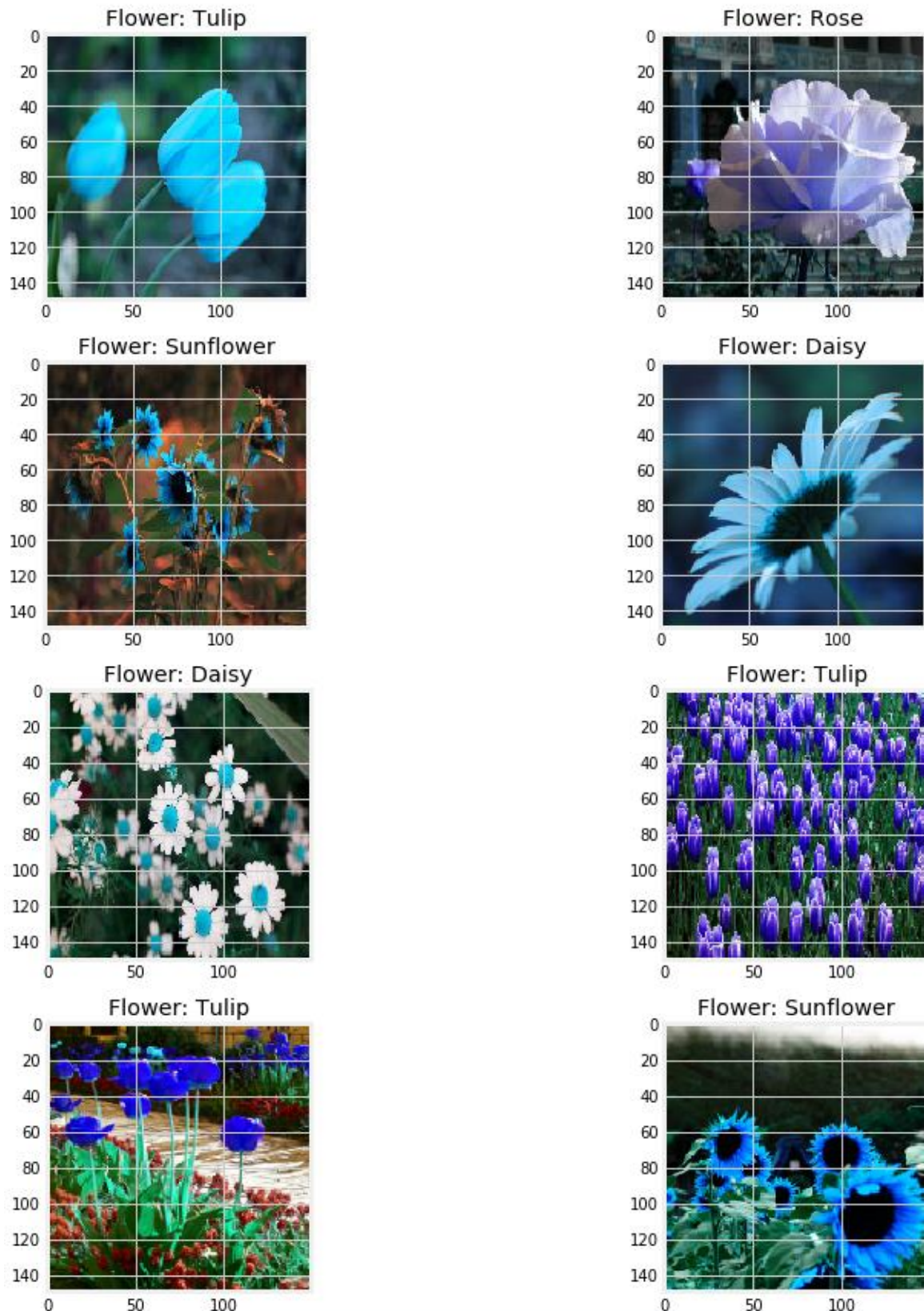


Figure 6.3: Displaying random images from the dataset



## Building the model:

```
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 150, 150, 32)	2432
max_pooling2d (MaxPooling2D)	(None, 75, 75, 32)	0
conv2d_1 (Conv2D)	(None, 75, 75, 64)	18496
max_pooling2d_1 (MaxPooling2D)	(None, 37, 37, 64)	0
conv2d_2 (Conv2D)	(None, 37, 37, 96)	55392
max_pooling2d_2 (MaxPooling2D)	(None, 18, 18, 96)	0
conv2d_3 (Conv2D)	(None, 18, 18, 96)	83040
max_pooling2d_3 (MaxPooling2D)	(None, 9, 9, 96)	0
flatten (Flatten)	(None, 7776)	0
dense (Dense)	(None, 512)	3981824
activation (Activation)	(None, 512)	0
dense_1 (Dense)	(None, 5)	2565

Total params: 4,143,749  
 Trainable params: 4,143,749  
 Non-trainable params: 0

Figure 6.4: Summary of the model

## Splitting the data set:

```
from sklearn.model_selection import train_test_split #for splitting the data

#splitting the test data and train data
x_train,x_test,y_train,y_test=train_test_split(X,Y,test_size=0.20,random_state=42)
```

Figure 6.5: Split the data set as Train and Test data

# FLOWER DETECTION

## Training the model:

```
History = model.fit_generator(datagen.flow(x_train,y_train, batch_size=batch_size),
                              epochs = epochs, validation_data = (x_test,y_test),
                              verbose = 1, steps_per_epoch=x_train.shape[0] // batch_size)
```

Epoch 1/50

25/25 [=====] - ETA: 1:40 - loss: 1.6057 - accuracy: 0.20 - ETA: 58s - loss: 1.7378 - accuracy: 0.2148 - ETA: 53s - loss: 1.7513 - accuracy: 0.221 - ETA: 40s - loss: 1.7530 - accuracy: 0.227 - ETA: 42s - loss: 1.7467 - accuracy: 0.231 - ETA: 41s - loss: 1.7385 - accuracy: 0.234 - ETA: 40s - loss: 1.7309 - accuracy: 0.235 - ETA: 38s - loss: 1.7237 - accuracy: 0.235 - ETA: 36s - loss: 1.7169 - accuracy: 0.236 - ETA: 34s - loss: 1.7107 - accuracy: 0.237 - ETA: 32s - loss: 1.7052 - accuracy: 0.237 - ETA: 30s - loss: 1.7000 - accuracy: 0.238 - ETA: 28s - loss: 1.6953 - accuracy: 0.238 - ETA: 26s - loss: 1.6910 - accuracy: 0.239 - ETA: 23s - loss: 1.6868 - accuracy: 0.240 - ETA: 21s - loss: 1.6827 - accuracy: 0.241 - ETA: 19s - loss: 1.6788 - accuracy: 0.243 - ETA: 16s - loss: 1.6749 - accuracy: 0.244 - ETA: 14s - loss: 1.6708 - accuracy: 0.245 - ETA: 12s - loss: 1.6666 - accuracy: 0.247 - ETA: 9s - loss: 1.6626 - accuracy: 0.248 - ETA: 7s - loss: 1.6585 - accuracy: 0.25 - ETA: 4s - loss: 1.6545 - accuracy: 0.25 - ETA: 2s - loss: 1.6504 - accuracy: 0.25 - ETA: 0s - loss: 1.6463 - accuracy: 0.25 - 71s 3s/step - loss: 1.6425 - accuracy: 0.2567 - val\_loss: 1.3741 - val\_accuracy: 0.4440

Epoch 2/50

25/25 [=====] - ETA: 1:13 - loss: 1.3735 - accuracy: 0.46 - ETA: 1:01 - loss: 1.3985 - accuracy: 0.44 - ETA: 56s - loss: 1.3901 - accuracy: 0.4466 - ETA: 53s - loss: 1.3760 - accuracy: 0.449 - ETA: 50s - loss: 1.3680 - accuracy: 0.445 - ETA: 47s - loss: 1.3618 - accuracy: 0.443 - ETA: 45s - loss: 1.3557 - accuracy: 0.440 - ETA: 42s - loss: 1.3522 - accuracy: 0.438 - ETA: 36s - loss: 1.3491 - accuracy: 0.436 - ETA: 35s - loss: 1.3452 - accuracy: 0.435 - ETA: 33s - loss: 1.3416 - accuracy: 0.435 - ETA: 31s - loss: 1.3375 - accuracy: 0.435 - ETA: 29s - loss: 1.3336 - accuracy: 0.434 - ETA: 26s - loss: 1.3297 - accuracy: 0.434 - ETA: 24s - loss: 1.3264 - accuracy: 0.434 - ETA: 22s - loss: 1.3238 - accuracy: 0.434 - ETA: 19s - loss: 1.3210 - accuracy: 0.434 - ETA: 17s - loss: 1.3184 - accuracy: 0.434 - ETA: 15s - loss: 1.3157 - accuracy: 0.434 - ETA: 12s - loss: 1.3130 - accuracy: 0.435 - ETA: 10s - loss: 1.3106 - accuracy: 0.436 - ETA: 7s - loss: 1.3082 - accuracy: 0.436 - ETA: 5s - loss: 1.3059 - accuracy: 0.43 - ETA: 2s - loss: 1.3034 - accuracy: 0.43 - ETA: 0s - loss: 1.3010 - accuracy: 0.43 - 69s 3s/step - loss: 1.2988 - accuracy: 0.4395 - val\_loss: 1.1487 - val\_accuracy: 0.4884

Epoch 3/50

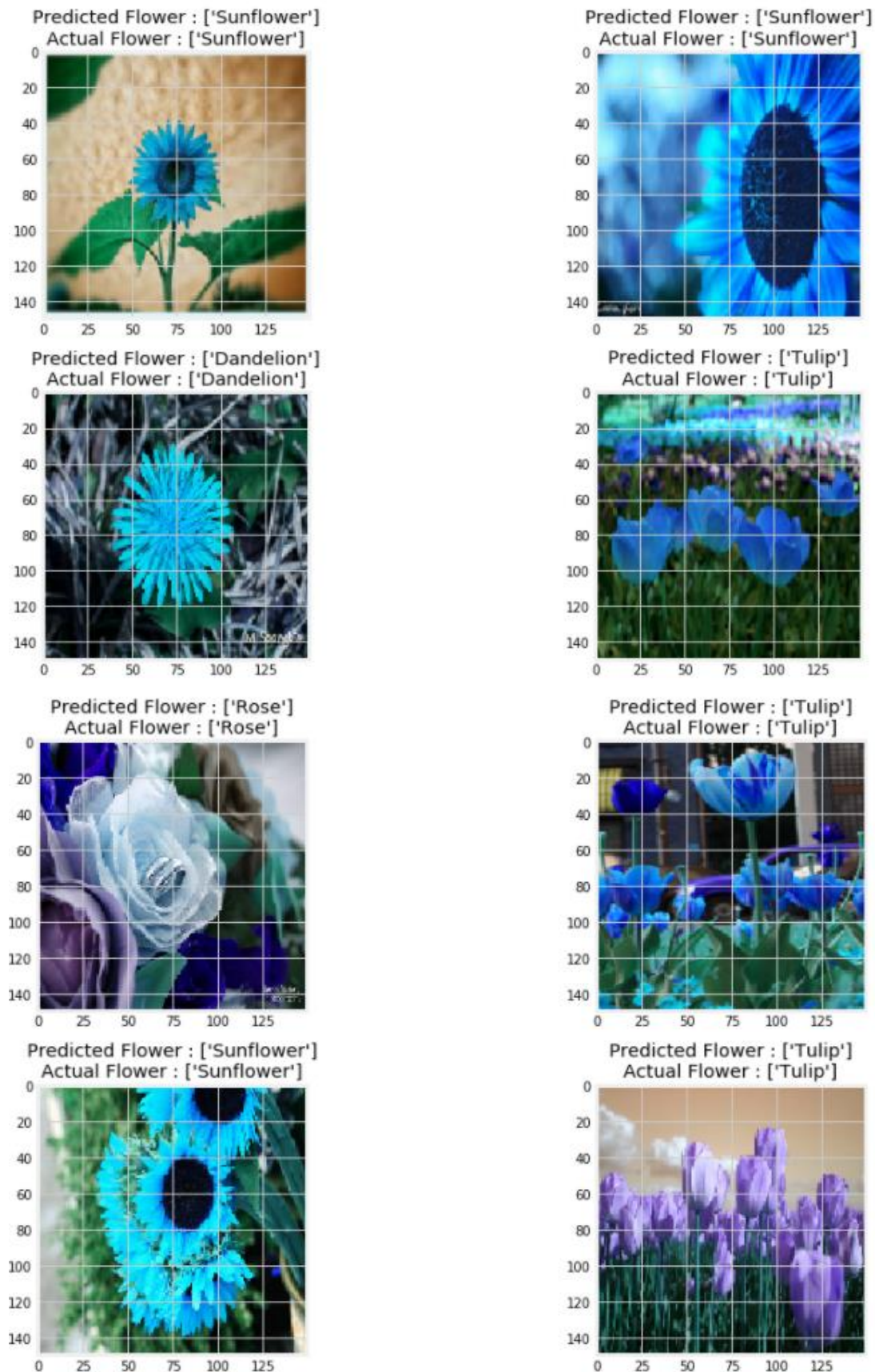
25/25 [=====] - ETA: 1:16 - loss: 1.1170 - accuracy: 0.49 - ETA: 55s - loss: 1.1059 - accuracy: 0.5117 - ETA: 55s - loss: 1.1051 - accuracy: 0.520 - ETA: 52s - loss: 1.1033 - accuracy: 0.523 - ETA: 50s - loss: 1.1028 - accuracy: 0.528 - ETA: 47s - loss: 1.1014 - accuracy: 0.532 - ETA: 45s - loss: 1.1001 - accuracy: 0.535 - ETA: 43s - loss: 1.0997 - accuracy: 0.538 - ETA: 40s - loss: 1.0994 - accuracy: 0.540 - ETA: 38s - loss: 1.0983 - accuracy: 0.542 - ETA: 35s - loss: 1.0989 - accuracy: 0.543 - ETA: 33s - loss: 1.0998 - accuracy: 0.543 - ETA: 30s - loss: 1.1008 - accuracy: 0.544 - ETA: 27s - loss: 1.1010 - accuracy: 0.544 - ETA: 24s - loss: 1.1015 - accuracy: 0.545 - ETA: 22s - loss: 1.1019 - accuracy: 0.545 - ETA: 19s - loss: 1.1023 - accuracy: 0.546 - ETA: 17s - loss: 1.1029 - accuracy: 0.546 - ETA: 14s - loss: 1.1032 - accuracy: 0.546 - ETA: 12s - loss: 1.1034 - accuracy: 0.547 - ETA: 10s - loss: 1.1036 - accuracy: 0.547 - ETA: 7s - loss: 1.1038 - accuracy: 0.547 - ETA: 5s - loss: 1.1041 - accuracy: 0.54 - ETA: 2s - loss: 1.1043 - accuracy: 0.54 - ETA: 0s - loss: 1.1043 - accuracy: 0.54 - 69s 3s/step - loss: 1.1042 - accuracy: 0.5474 - val\_loss: 1.1524 - val\_accuracy: 0.5365

Epoch 4/50

25/25 [=====] - ETA: 1:13 - loss: 1.2223 - accuracy: 0.48 - ETA: 59s - loss: 1.1979 - accuracy: 0.5059 - ETA: 55s - loss: 1.1822 - accuracy: 0.521 - ETA: 52s - loss: 1.1689 - accuracy: 0.525 - ETA: 42s - loss: 1.1628 - accuracy: 0.527 - ETA: 42s - loss: 1.1551 - accuracy: 0.530 - ETA: 41s - loss: 1.1531 - accuracy: 0.530 - ETA: 40s - loss: 1.1509 - accuracy: 0.530 - ETA: 38s - loss: 1.1473 - accuracy: 0.530 - ETA: 36s - loss: 1.1442 - accuracy: 0.531 - ETA: 34s - loss: 1.1416 - accuracy: 0.532 - ETA: 32s - loss: 1.1394 - accuracy: 0.533 - ETA: 30s - loss: 1.1368 - accuracy: 0.534 - ETA: 27s - loss: 1.1345 - accuracy: 0.535 - ETA: 25s - loss: 1.1328 - accuracy: 0.535 - ETA: 22s - loss: 1.1310 - accuracy: 0.536 - ETA: 20s - loss: 1.1292 - accuracy: 0.537 - ETA: 17s - loss: 1.1275 - accuracy: 0.538 - ETA: 15s - loss: 1.1259 - accuracy: 0.539 - ETA: 13s - loss: 1.1246 - accuracy: 0.539 - ETA: 10s - loss: 1.1234 - accuracy: 0.540 - ETA: 7s - loss: 1.1224 - accuracy: 0.541 - ETA: 5s - loss: 1.1214 - accuracy: 0.54 - ETA: 2s - loss: 1.1203 - accuracy: 0.54 - ETA: 0s - loss: 1.1192 - accuracy: 0.54 - 76s 3s/step - loss: 1.1181 - accuracy: 0.5439 - val\_loss: 1.0577 - val\_accuracy: 0.5541

Figure 6.6: Fitting the model on train data

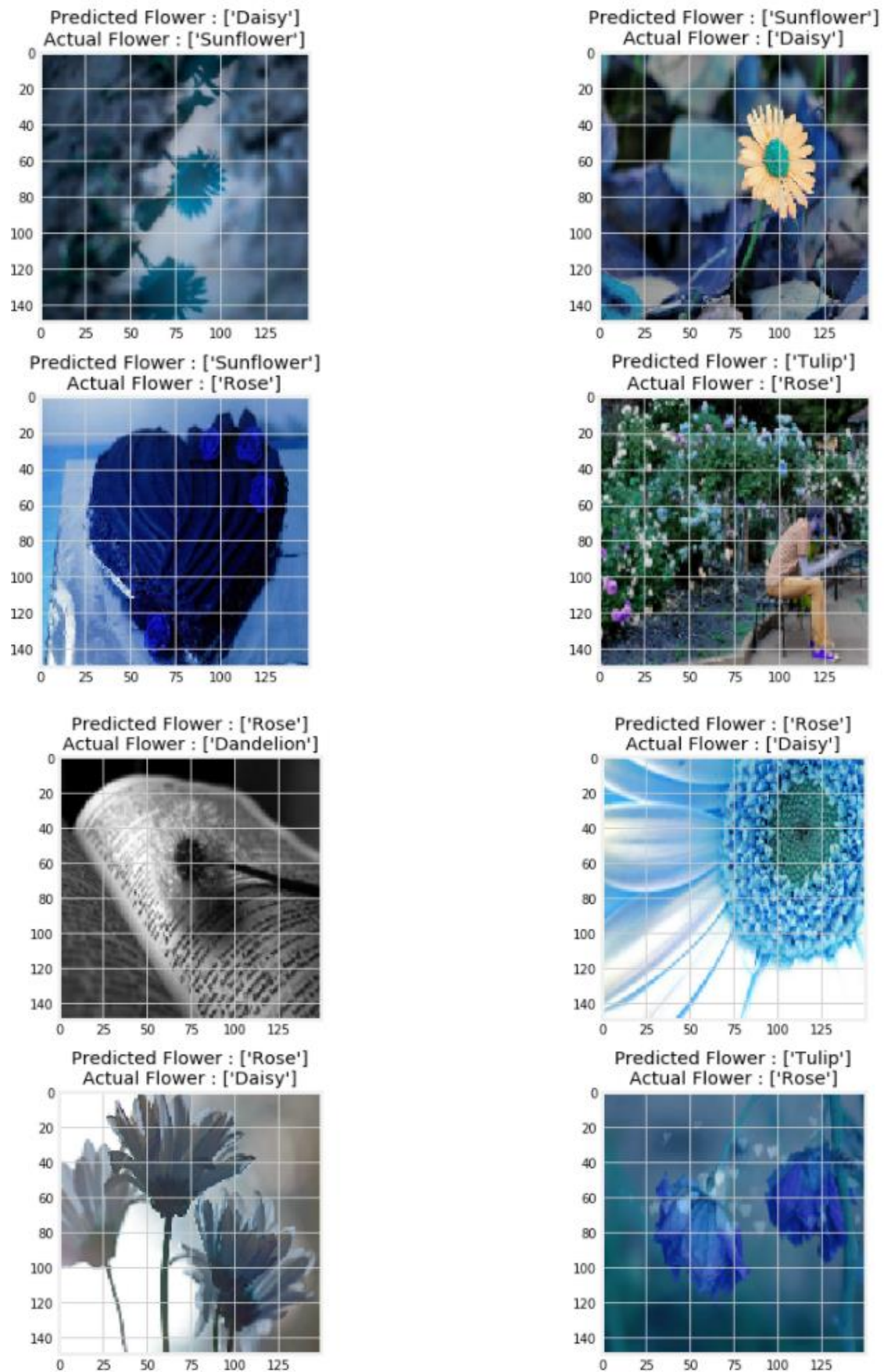
## FLOWER DETECTION



**Figure 6.7: Displaying correctly classified images from test data**



## FLOWER DETECTION



**Figure 6.8: Displaying misclassified images from test data**

```
test_image = image.load_img('C:/Users/DELL/Desktop/tulips.jpg', target_size = (150,150))
display(test_image)
test_image = image.img_to_array(test_image)
test_image = np.expand_dims(test_image, axis = 0)
test_image=test_image/255
result =model.predict(test_image)
print(result)
print(labels[result.argmax()])
```



```
[[0.03428503 0.00236131 0.25590205 0.02087361 0.686578  ]]
tulip
```

**Figure 6.9: Predicting on a new Tulip image**

```
test_image = image.load_img('D:/Major Project/Image1.jpg', target_size = (150,150))
display(test_image)
test_image = image.img_to_array(test_image)
test_image = np.expand_dims(test_image, axis = 0)
test_image=test_image/255
result =model.predict(test_image)
print(result)
print(labels)
print(labels[result.argmax()])
```



```
[[0.2229783 0.00591879 0.6507051 0.00872318 0.11167461]]
['daisy', 'dandelion', 'rose', 'sunflower', 'tulip']
rose
```

**Figure 6.10: Predicting on a new Rose image**

## FLOWER DETECTION

Performance measure:

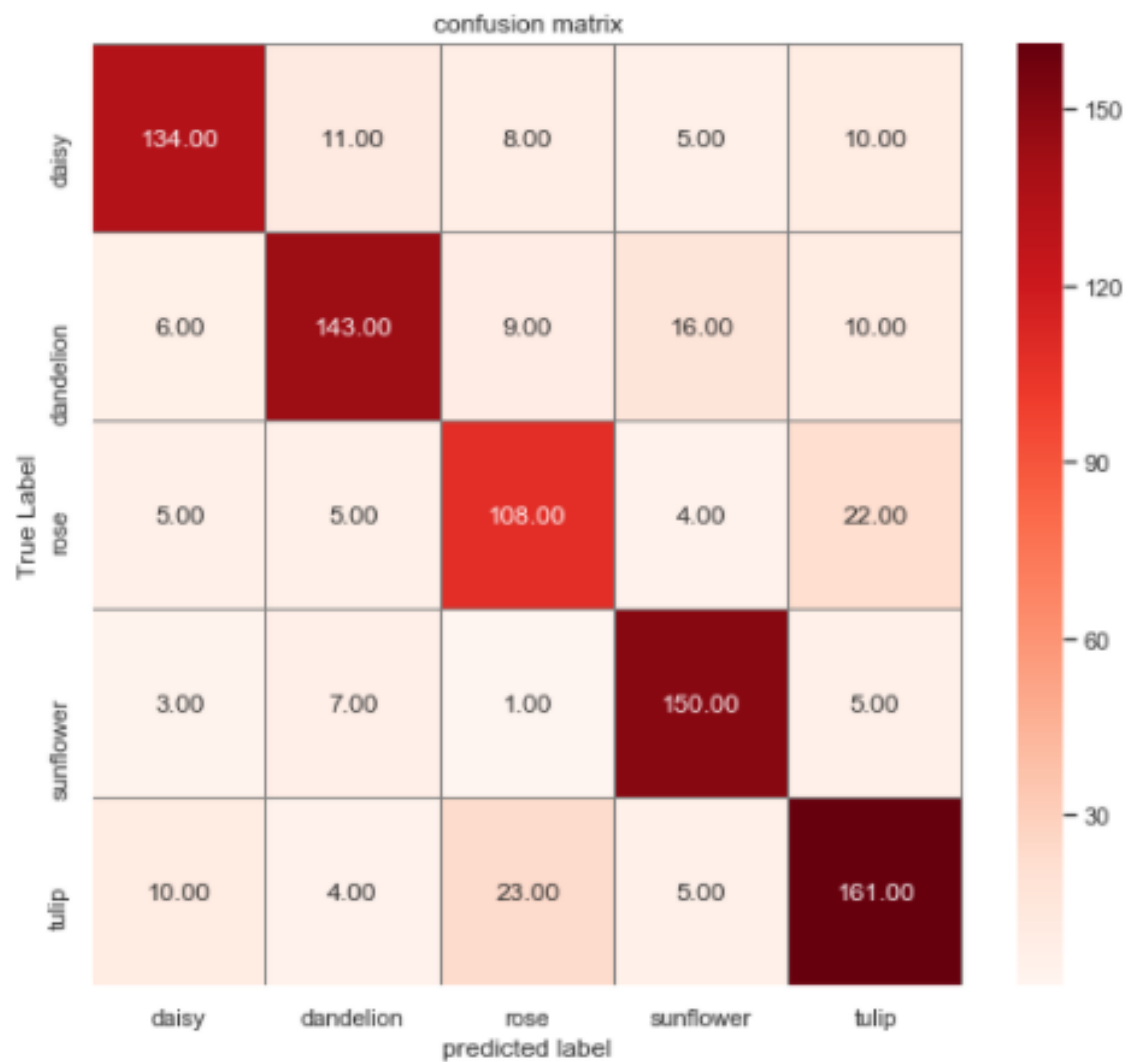


Figure 6.11: Confusion Matrix

```
def acc(y_true, y_pred):  
    return np.equal(np.argmax(y_true, axis=-1), np.argmax(Y_pred, axis=-1)).mean()  
print("accuracy: " + str(acc(y_test, Y_pred)))
```

accuracy: 0.8046242774566474

Figure 6.12: Accuracy Score

# TESTING AND VALIDATION

## **7. TESTING AND VALIDATIONS**

### **7.1 INTRODUCTION**

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub assemblies and/or a finished product. It is the process of exercising software with the intent of ensuring that the software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of tests. Each test type addresses a specific testing requirement.

### **7.2 TESTING STRATEGIES**

#### **Unit testing**

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program input produces valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application. It is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

#### **Functional test**

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals.

#### **Functional testing is centered on the following items:**

Valid Input : identified classes of valid input must be accepted.

Invalid Input : identified classes of invalid input must be rejected.

Functions : identified functions must be exercised.



**Output** : identified classes of application outputs must be exercised.

**Systems/Procedures** : interfacing systems or procedures must be invoked.

### **System Test**

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

### **Performance Test**

The performance test ensures that the output be produced within the time limits, and the time taken by the system for compiling, giving response to the users and request being send to the system for to retrieve the results.

### **Integration testing**

The task of integration test is to check that component or software applications, e.g. components in a software system or one set up software applications at the company level interact without error.

### **Acceptance Testing**

User Acceptance Testing is a critical phase of any project and requires significant participation by the end user,. It also ensures that the system meets the functional requirement.

### **Goal of ML testing**

In machine learning, a programmer usually inputs the data and the desired behavior, and the logic is elaborated by the machine. This is especially true for deep learning. Therefore, the purpose of machine learning testing is, first of all, to ensure that this learned logic will remain consistent, no matter how many times we call the program. Every ML model needs not only to be tested but evaluated. Your model should generalize well. Evaluation is needed to make sure that the performance is satisfactory.

For testing machine learning systems, we should run model evaluation and model tests.

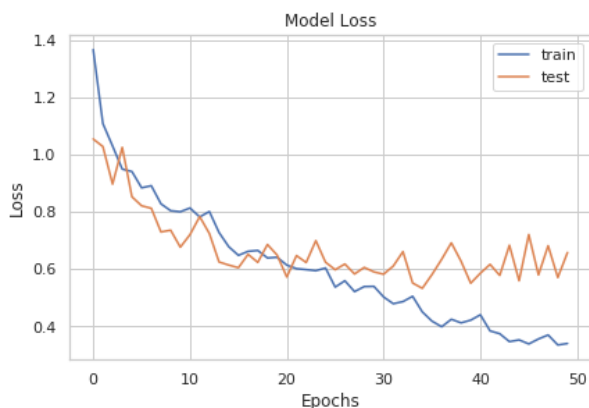
- **Model evaluation** covers metrics and plots which summarize performance on a validation or test dataset.
- **Model testing** involves explicit checks for behaviors that we expect our model to follow.

## 7.3 TESTING CNN MODEL

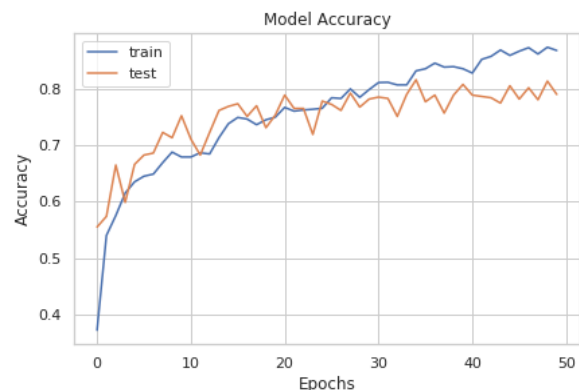
### 7.3.1 Model training history after each Epoch:

Keras provides the capability to register callbacks when training a deep learning model. One of the default callbacks that is registered when training all deep learning models is the History callback. It records training metrics for each epoch. This includes the loss and the accuracy.

The following plots are till 50 epochs



**Figure 7.1: Plot of loss**



**Figure 7.2: Plot of Accuracy**

From the plot of loss, we can see that the model has comparable performance on both train and validation datasets. As these start to depart consistently, it might be a sign to stop training at an earlier epoch.

From the plot of accuracy we can see that training can be stopped as the curves started departing. We can also see that the model has not yet over-learned the training dataset, showing comparable skill on both datasets.

From these plots we can conclude that the number of epochs chosen i.e. is the appropriate number.

### 7.3.2 Testing on images from dataset:

#### Test Data

The test data set is a set of observations used to evaluate the performance of the model using some performance metrics. If the test set does contain examples from the training set, it will be difficult to assess whether the algorithm has learned to generalize from the training set or has simply memorized it.

Initially the data set is divided into two parts i.e. train data and test data. Test data is 20% of the total data. The performance of how the model performs on test data is evaluated using confusion matrix and accuracy score.

The Fig 6.12 shows that the accuracy on the test data is 80%. The Fig 6.11 shows the confusion matrix on test data. The following conclusions can be drawn it.

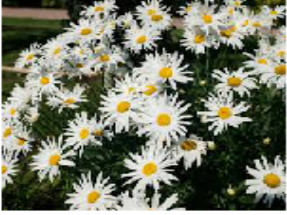




Type of flower	Daisy	Dandelion	Rose	Sunflower	Tulip
Total No of images	168	184	144	165	203
Correctly classified	134	143	108	150	161
Misclassified	34	41	36	15	42

**Figure 7.3: Conclusion from confusion matrix**

### 7.3.3 Testing on images not in dataset:

The images given in the real time will not same as the ones given during training or testing. In order to check further robustness of the model, a new image that is not present in the data set is given to the model for testing its performance. The following figure consolidates the accuracy of each type of image.

## FLOWER DETECTION

 <pre>[[0.89971644 0.00493891 0.08828656 0.00106212 0.00599598] ['daisy', 'dandelion', 'rose', 'sunflower', 'tulip'] daisy</pre>	<p style="text-align: center;"><b>Daisy is predicted with 0.89 probability</b></p>
 <pre>[[6.9067180e-02 9.2322290e-01 7.0821266e-03 1.6591517e-04 4.6183684e-04] ['daisy', 'dandelion', 'rose', 'sunflower', 'tulip'] dandelion</pre>	<p style="text-align: center;"><b>Dandelion is predicted with 0.92 probability</b></p>
 <pre>[[0.2229783 0.00591879 0.6507051 0.00872318 0.11167461] ['daisy', 'dandelion', 'rose', 'sunflower', 'tulip'] rose</pre>	<p style="text-align: center;"><b>Rose is predicted with 0.65 probability</b></p>
 <pre>[[0.03428503 0.00236131 0.25590205 0.02087361 0.686578] ['daisy', 'dandelion', 'rose', 'sunflower', 'tulip'] tulip</pre>	<p style="text-align: center;"><b>Tulip is predicted with 0.68 probability</b></p>
 <pre>[[0.06511386 0.09006584 0.2222677 0.39324984 0.22930282] ['daisy', 'dandelion', 'rose', 'sunflower', 'tulip'] sunflower</pre>	<p style="text-align: center;"><b>Sunflower is predicted with 0.39 probability</b></p>

**Figure 7.4: Testing on new images**

# **CONCLUSION AND FUTURE ENHANCEMENTS**

## **8. CONCLUSION AND FUTURE ENHANCEMENTS**

### **8.1 CONCLUSION**

Throughout this project we have explored the possibility of building a flower detection system using the Python in Jupyter note book. There are many flower species. It might be hard to identify their names and properties just by looking. The flower classification model will help in immediately identifying the properties of a flower.

Using CNN for classification of flowers can be implemented in real-time applications and can also be used to help botanists for their research for immediately identifying the properties of a flower. We are able to classify five types of flowers and achieved the accuracy of 85% on the test data.

### **8.2 FUTURE SCOPE**

The proposed approach is a faster way to train a Convolutional Neural Network (CNN) with a smaller dataset and limited computational resource such as CPU. This can be improved in such a way that, user can quickly take an image of the plant species and get all the information about the species. The crucial part in building such a system is having a huge training dataset.

As there are millions of flower species around the world, this system could be made adaptable by training more number of flower species images to recognize different species around the world. Thus, the future work would be to construct a larger database with not only flower images, but also with leaves, fruits, bark etc., collected from different sources around different parts of the world. This system would also be useful to identify plants for medicinal purposes.

## REFERENCES

References for the Project Development were taken from the following books , websites and papers.

### **Books Referred:**

- Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems, Book by Aurelien Geron
- Neural networks for pattern recognition, Book by Christopher Bishop
- Convolutional Neural Networks: Implement Advanced Deep Learning Models Using Python, Book by Md. Rezaul Karim, Mohit Sewak, and Pradeep Pujari

### **Websites Referred:**

- Data set: <https://www.kaggle.com/alxmamaev/flowers-recognition>
- <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>
- [https://www.tensorflow.org/api\\_docs/python/tf/keras/preprocessing/image/ImageDataGenerator](https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator)
- <https://en.wikipedia.org/wiki/Keras>

### **Papers Referred:**

- [1] Mohd Azlan Abu1 , Nurul Hazirah Indral, Izanoordina Ahmad: “**A study on Image Classification based on Deep Learning and Tensorflow**”, International Journal of Engineering Research and Technology. ISSN 0974-3154, Volume 12, pp. 563-569
- [2] Saad Albawi, Tareq Abed Mohammed: “**Understanding of a Convolutional Neural Network**”, The International Conference on Engineering and Technology 2017, At: Antalya, Turkey