

# **Detection of Diseases in Blueberry Leaves using Machine Learning**

*Summer Internship Report Submitted in partial fulfillment*

*of the requirement for undergraduate degree of*

**Bachelor of Technology**

**In**

**Computer Science and Engineering**

**By**

**K.Nanda Kiran(221710301029)**

**Y.Rohith Srikanth(221710301064)**

**B.Shanmuka Sai Datta(221710301005)**

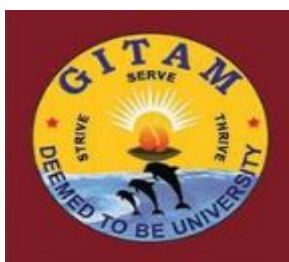
**G.Sushanth Reddy(221710301024)**

**Under the guidance of**

**Mr.Mujeeb Shaik Mohammed**

**Assistant Professor**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**



**GITAM  
(Deemed to be University)**

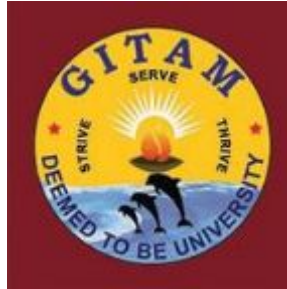
**HYDERABAD**

**December 2020**

# **DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

## **GITAM**

**(Deemed to be University)**



## **DECLARATION**

We hereby declare that the mini project entitled “**Detection of Diseases in Blueberry Leaves using Machine Learning**” is an original work done in the Department of Computer Science and Engineering, GITAM School of Technology, GITAM (Deemed to be University) submitted in partial fulfillment of the requirements for the award of the degree of B.Tech. in Computer Science and Engineering. The work has not been submitted to any other college or University for the award of any degree or diploma.

**Place:Hyderabad**

**Date:**

**K.Nanda Kiran(221710301029)**

**Y.Rohith Srikanth(221710301064)**

**B.Shanmukha Sai Datta(221710301005)**

**G.Sushanth Reddy(221710301024)**

# **DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**GITAM**

**(Deemed to be University)**



## **CERTIFICATE**

This is to certify that the Industrial Training Report entitled “**Detection of Diseases in Blueberry Leaves using Machine Learning**” is being submitted by **K. Nanda Kiran(221710301029), Y. Rohith Srikanth(221710301064), B. Shanmukha Sai Datta(221710301005), G. Sushanth Reddy(221710301024)** in partial fulfillment of the requirement for the award of Bachelor of Technology in Computer Science and Engineering at GITAM (Deemed To Be University), Hyderabad during the academic year 2017-21.

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

**Mr.Mujeeb Shaik Mohammed**

**Assistant Professor**

**Department of CSE**

**Dr.Phani Kumar**

**Professor and HOD**

**Department of CSE**

## **ACKNOWLEDGEMENT**

Our Mini Project would not have been successful without the help of several people. We would like to thank the personalities who were part of our seminar in numerous ways, those who gave us outstanding support from the birth of the seminar.

We are extremely thankful to our honorable Pro-Vice Chancellor, **Prof. N. Siva Prasad** for providing necessary infrastructure and resources for the accomplishment of our seminar.

We are highly indebted to **Prof. N. Seetharamaiah**, Principal, School of Technology, for his support during the tenure of the seminar.

We are very much obliged to our beloved **Prof. S. Phani Kumar**, Head of the Department of Computer Science & Engineering for providing the opportunity to undertake this seminar and encouragement in completion of this seminar.

We hereby wish to express our deep sense of gratitude to Dr. S Aparna, Assistant Professor, Department of Computer Science and Engineering, School of Technology and to Guide, Assistant Professor, Department of Computer Science and Engineering, School of Technology for the esteemed guidance, moral support and invaluable advice provided by them for the success of the Mini project.

We are also thankful to all the staff members of the Computer Science and Engineering department who have cooperated in making our Project a success.

**K.Nanda Kiran(221710301029)**

**Y.Rohith Srikanth(221710301064)**

**B.Shanmukha Sai Datta(221710301005)**

**G.Sushanth Reddy(221710301024)**

## **ABSTRACT**

Agriculture plays a major role in the present society. Developing effective technologies to improve agriculture, is important. One such technology is detection of plant diseases using leaf images. This project uses an algorithm to take the image of a leaf of a plant under study, analyzing the leaf image and then determining the disease in the plant, if there is any. The algorithm is based on the image template matching technique using Normalized Square Differences Based Image Matching formula.

The system has three modules. The DTD (Disease types module) is used to store the details of different types of plant diseases in the Database, the DA (Disease Analysis) module is used to analyze the disease of a given plant, using its leaf image and the IP (Image Processing) module is used to pinpoint the affected area of the leaf.

This Project uses PyQt tool to create the needed Graphical User Interfaces, PyUIC module to automatically generate the automated code .

The images of various types of disease affected leaves are stored in an image file system. The file system will be stored at a specific location in directory structure. The location will be fed as in input to the system through a GUI screen.

Algorithm Used: Normalized Square Differences method.

Outputs from the project:

- (1) A set of graphical User Interfaces to control project operations.
- (2) Three different leaf images with disease
- (3) Images showing the affected part of the leaves.

## Table of Contents

S .No	Title	Page .No
	Abstract.....	i
	List of tables.....	ii-iii
	List of Figures.....	iv
1	Introduction.....	1-3
	1.1.Advantages	1
	1.2.Problem definition	2
	1.3.Other Problems	2
2	Literature Survey.....	4-6
3	Problem Analysis.....	7-8
	3.1.Requirement Analysis	7
	3.1.1.Hardware Requirement	7
	3.1.2.Software Requirement	7
	3.2 .Feasibility Analysis	7
	3.2.1.Economical Feasibility	8
	3.2.2.Technical Feasibility	8
4	Design.....	9-28
	4.1.UML diagrams	9
	4.2.Tools Used	13
	4.2.1.Python Qt Designer	13
	4.2.2.Parameters	26

<b>5</b>	<b>Implementation.....</b>	<b>29-42</b>
	5.1.Languages	29
	5.2.Sample code	30
	5.3.Database Connectivity	37
	5.4.Outcome of above process	39
<b>6</b>	<b>Testing.....</b>	<b>43-46</b>
	6.1.Software Testing	43
	6.1.1.Types of testing	43
	6.1.2.Testing the model	45
<b>7</b>	<b>System Testing Results.....</b>	<b>47-49</b>
<b>8</b>	<b>Conclusion.....</b>	<b>50</b>
	References.....	51

## List of Figures

<b>Figures</b>	<b>Names</b>	<b>Page.No</b>
4.1	Data flow diagram.....	9
4.2	Use Case Diagrams.....	10
4.3	Sequence Diagram.....	11
4.4	Activity Diagram.....	12
4.5	Template of Qt Designer.....	14
4.6	Window Design attributes	14
5.1	Screenshot showing the files created during the project	39
5.2	Main window.....	40
5.3	Leaf type details window screenshot.....	41
5.4	Temperature details window screenshot.....	41
5.5	Moisture details window screenshot.....	42
5.6	Accept Leaf image.....	42
6.1	Screenshot of Leaf Type output.....	45
6.2	screenshot of the SQLite db leaf type.....	45
6.3	Screenshot of Temperature output.....	46
7.1	screenshot of the SQLite db temperature.....	47
7.2	Screenshot of Moisture Details output.....	47
7.3	screenshot of the SQLite db moisture details.....	48
7.4	Screenshot of Accept Leaf image acceptance.....	48
7.5	Final output of Disease affected area.....	49



# CHAPTER 1

## INTRODUCTION

Our project is “Detection of Diseases in Blueberry Leaves using Machine Learning.”, for this we have chosen python which is robust, easy to understand and we can minimize the code through various predefined functions. To develop this, we have used different tools like PYQT Designer, PYUIC and python. Each and every tool plays a major role. PYQT Designer tool is used to design user interfaces, PYUIC tool is helpful to generate python code for user interfaces. Detection of Diseases in Blueberry Leaves using Machine Learning. is developed using python, pyuic and pyqt designer. PyQt is a Python binding of the cross-platform GUI toolkit Qt, implemented as a Python plug-in.

PyQt implements different classes and methods including: classes for accessing SQL databases (ODBC, MySQL, PostgreSQL, Oracle, SQLite), Scintilla-based rich text editor widget, data aware widgets that are automatically populated from a database, an XML parser and SVG support. Scalable Vector Graphics (SVG) is an XML-based vector image format for graphics with support for interactivity. All the above mentioned features of PyQt, are extensively used in this project, to create the needed Graphical User Interfaces.

PyUic tool is used to automatically generate the code for the Front end user interfaces created by PyQt. All the front end python code is automatically generated by this tool, by converting the user interface(.ui) files into .py files.

### 1.1 ADVANTAGES

Following are the technical advantages:

- Using Python, which is chosen as the best programming language, by the Programming Community.
- More Functionality can be implemented with less no. of lines of code in Python.
- PyQt tool is used to create the Graphical User interfaces.
- All the Front end code is generated automatically by PyUIC.
- Following are the other project advantages:
- The project is very useful to the farmers, as they need not go for the plant specialist consultation towards the identification of the disease.
- The project is also useful to the plant specialists, as they need not remember the details about each and every plant disease.

This project finally leads to the enhancement of average plant life.

## **1.2 PROBLEM DEFINITION**

### **Vision:**

The project aims at developing a tool for the Detection of Diseases in Blueberry Leaves using Machine Learning, with all the above mentioned advantages.

### **Mission:**

This tool is developed by using Python along with its layout toolkit PyQt & PyUIC.

## **1.3 OTHER PROBLEMS**

Types of pests and insects

Following are different types of pests.

- Beneficial Insects
- Garden Pests
- Household Pests
- Houseplant Pests
- Landscape/ Lawn Pests
- Orchard & Tree Pests

### **Beneficial Insects**

Not all insects are pests. In fact, there are a great many good bugs for garden. Many growers recognize the help provided by beneficial insects as well as the injury done by harmful ones. As per C.B Huffaker "When we kill off the natural enemies of a pest, we inherit their work."

### **Garden Pests**

One bug does not make a problem! In nature, there are always some garden pests chewing on plants; that's just the way it is. However, not all pest damage is significant enough to warrant action. Even the healthiest gardens encounter bugs at one time or another, yet they still produce a beautiful harvest. As gardeners, they must each consider the level of pest activity that they are willing to tolerate.

### **Household Pests**

All homes occasionally run into problems with household pests. While most are merely a nuisance, some may bite, sting or transmit disease. A few may even cause serious structural damage which can impact the value of your house. While it may seem easier to reach for a can of bug spray, this may not be the best way to fix

the problem. Many homeowners today are leery of the harmful effects of chemical pesticides and are turning to safer, least-toxic solutions for protection.

### **Houseplant Pests**

Like all plants, houseplants will occasionally come under attack from pests. These insects can be just as voracious as their outdoor counterparts but have the added benefit of developing and reproducing in near ideal conditions. As a result, houseplant pests can multiply very quickly, so it has to be diligent about checking for symptoms. A plant that suddenly begins to look ill, take a closer look — chances are an insect is responsible. Infestations can be very severe and plants that have had more than half of their leaves damaged are probably not worth saving. For this reason, it is important to identify and control indoor plant pests as quickly as possible.

### **Landscape/ Lawn Pests**

Healthy lawns and landscapes are teeming with insect life. Yet, very few of the species you see in your backyard are damaging (it is estimated that less than 1% of all insect species are pests). The point here is that once lawn pests are discovered don't go crazy and spray all kinds of harmful chemicals to eradicate the problem. Broad-spectrum insecticides kill all insects, both good and bad.

### **Orchard & Tree Pests**

Trees and ornamental shrubs will eventually encounter a variety of bugs. The difficulty is determining when, or if, many of these problems will reach a level where something must be done about it. Some tree pests are present every few growing seasons and require immediate attention, while many others are found each year but cause little or no harm. Maintaining healthy, productive trees means knowing about common pests. Survey your backyard regularly to identify problems at an early stage. Identifying the problem and evaluating its severity will help decide if control is necessary and, if so, what management strategy to take. The tree pests listed below are some (of the many) that homeowners are likely to encounter.

## **CHAPTER 2**

### **LITERATURE SURVEY**

This paper explains how image processing techniques and Machine Learning algorithms were used, such as Support Vector Machine (SVM), Artificial Neural Networks (ANN) and Random Forest; and Deep Learning's technique Convolutional Neural Network (CNN) was also used so we can determine which is the best algorithm for the construction of a recognition model that detects whether a blueberry plant is being affected by a disease or pest, or if it is healthy. The images were processed with different filters such as medianBlur and gaussianblur for the elimination of noise, the add Weighted filter was used for the enhancement of details in the images.

The images were compiled by the authors of this work, since there was no accessible database of this specific kind of fruit, for which we visited Valle and Pampa farm so we could take pictures of different blueberry leaves, labeled in three different tags: diseased, plagued and healthy. The extraction of characteristics was done with algorithms such as HOG (Histogram of oriented gradients) and LBP (Local binary patterns), both normalized and not normalized. The results of the model showed an 84% accuracy index using Deep Learning, this model was able to classify whether the blueberry plant was being affected or not. The result of this work provides a solution to a constant problem in the agricultural sector that affects the production of blueberries, because pests as well as diseases are constant problems in this sector.

The investment in blueberries is a matter of great care because the cultivation of these fruits is highly complex, in addition to needing as much agronomic knowledge as possible. The companies that venture for the first time in this field can end into failure if they do not have all the important knowledge to start. The approximate investment per hectare is 29 thousand dollars and this doesn't include any irrigation system so in the end the investment ends being higher. Considering this, it is very important to invest in all the necessary care so the investment in this crop is not lost, but blueberries like other fruits are susceptible to certain risks, such as diseases and pests that can affect the blueberry plantations, which can end in loss of hectares of crops that they were ready to harvest.

The pests and diseases that attack blueberries not only affect them during the growth of these, but can also affect them just weeks before starting the harvest, in some cases the plants reduce the amount of produced fruit, in others the quality of the fruits, harvesting very small fruits that were expected, in others the plant does not produce anything until it is free of the disease or plague and in other cases, the plant dies thus losing the fruits it produced and the investment that was done in the

purchase of each plant. This happens many times because the control that is made to combat these pests or diseases is not enough, in addition sometimes the disease is not detected in its initial phase or the type of pest or disease is not determined correctly causing the plant not to heal and to infect those that are nearby, causing a spread of the disease or plague, and as a result financial losses to the company.

According to Mr. César Torres, head of crops at Bayer, who exhibited at the Berries seminar organized by Sierra Exportadora, he explained the main pests and diseases that attack blueberries in Peru, as follows: One of the important problems that is being presented in the blueberry is the anomalous insect sp that is a white beetle whose larvae mainly affect the roots, they eat rootlets and can even cause the death of the plant. It has come to report up to 30% of death of the plant, this pest attacks all year and there is still no effective control. It usually happens when organic material that is not well decomposed is used [2].

A plague that has bothered the blueberry in the north of Peru, accurately in Trujillo, is the *Prodiplosis longifila* an insect that mainly affects the asparagus and has a very high reproduction capacity, therefore the ability to do harm is constant. Another plague that is occurring particularly in the blueberry production areas is the *Heliothis*, a fruit-boring worm belonging to the *Noctuidae* family, the larvae perforate the fruits, the damaged fruits rot and fall, causing defoliation of shoots, terminals and fruits, produces up to 40% damage on the total population of plants

For this work, other research was taken as a reference with topics related to those used in this research. In the thesis "Assessment of Internal and External Quality of Blueberries using images", addresses situations related to how to improve the quality of blueberries in order to increase performance and improve the marketing of these, this thesis uses machine learning techniques with images.

They were used for training different classifiers in order to determine which of them gives a better classification, some classifiers we used were: discriminant linear analysis, vector support machine and probabilistic neural network being these also reported with better performance after training and pattern recognition.

Another investigation taken into account was the thesis "Evaluation of Classifiers for Automatic Disease Detection in Citrus leaves using machine vision", this investigation was intended to evaluate an adequate control for the diseases presented in the citrus industry because this industry is important in the agricultural economy of Florida. In this thesis, the study that was implemented investigated the use of artificial vision and image processing techniques in the classification of diseased leaves of citrus fruits. In addition, algorithms based on image processing techniques were used to extract characteristics. The classifiers used in this study were: statistical classifier using the Mahalanobis minimum distance method, neural network based on the use of the back propagation algorithm and neural network using radial

basis functions. His study determined that such classification methods are suitable for the classification of citrus leaves

The proposal is to implement a recognition model of the status of a blueberry plant and identify if it is being affected by a disease or pest, or if it's healthy.

This proposal aims to solve a number of problems in blueberry crops because reducing the time of analysis or recognition that was normally done in laboratories, would make it immediately with only an instant capture or a short video of the plant. At the end of the investigation, a disease and pest recognition system will be obtained and will be able to detect the disease. In addition people can use this system in a mobile device or a mechanical device that monitors the showings of blueberries all the time so the accuracy of the system is greater; if it is used in a mechanical device it would have to be go around the fields for several hours and send alerts to the operations center when finding certain anomalies in the blueberry plants.

This work concludes that the recognition model has a good level of prediction, with an 84% prediction it can correctly classify a healthy leaf and one sick or affected by a pest. The advantage of developing a database of images of blueberry plants was that we had the segmented image ready to work, but one disadvantage was the time of collection of these images because we had to take pictures of different blueberry leaves to classify them. The work in the future is to continue collecting more images of blueberry leaves to increase the database. It is also expected to build a model that classifies diseased blueberry leaves by type of disease or plague.

In the future we will work with other features extraction algorithms such as sift (Scale-invariant feature transform) or surf(Speeded-Up Robust Features). The idea is to develop a mobile application and that it can be used by anyone in this sector, the person would take a photo of the affected plant and the system will hint what the disease is and what the solution would be. Another idea would be to build a small robot that monitors cranberry fields and if it finds an affected plant, then it will send an alarm to the supervisor.

Finally after finishing this project, we want to support other people who have an interest in the area of artificial vision by providing them with the image database that was built in this project.

## **CHAPTER 3**

### **PROBLEM ANALYSIS**

#### **3.1 Requirement Analysis**

##### **3.1.1 Hardware Requirements**

1. It requires a minimum of 2.16 GHz processor.
2. It requires a minimum of 4 GB RAM.
3. It requires 64-bit architecture.
4. It requires a minimum storage of 500GB.

##### **3.1.2 Software Requirements**

1. It requires a 64-bit Ubuntu Operating System.
2. Python Qt Designer for designing user interfaces.
3. MY SQL server for storing database Entities.
4. Pyuic for converting the layout designed user interface (UI) to python code.
5. Python language for coding.

#### **3.2 Feasibility Analysis**

As the name implies, a feasibility study is used to determine the viability of an idea, such as ensuring a project is legally and technically feasible as well as economically justifiable. It tells us whether a project is worth the investment—in some cases, a project may not be doable. There can be many reasons for this, including requiring too many resources, which not only prevents those resources from performing other tasks but also may cost more than an organization would earn back by taking on a project that isn't profitable.

### **3.2.1 Economical Feasibility**

This assessment typically involves a cost/ benefits analysis of the project, helping organizations determine the viability, cost, and benefits associated with a project before financial resources are allocated. It also serves as an independent project assessment and enhances project credibility—helping decision makers determine the positive economic benefits to the organization that the proposed project will provide. Our project is economically feasible because in this we have used “UBUNTU”, “PYTHON”, “PYQT” designer tool and “PYUIC” which are all available as an open source.

### **3.2.2 Technical Feasibility**

This assessment focuses on the technical resources available to the organization. It helps organizations determine whether the technical resources meet capacity. Technical feasibility also involves evaluation of the hardware, software, and other technology requirements of the proposed system. A prototype of the tool was developed to verify the technical feasibility. The prototype is working successfully and hence the project is feasible.



## CHAPTER 4

### DESIGN

#### 4.1 UML Diagrams

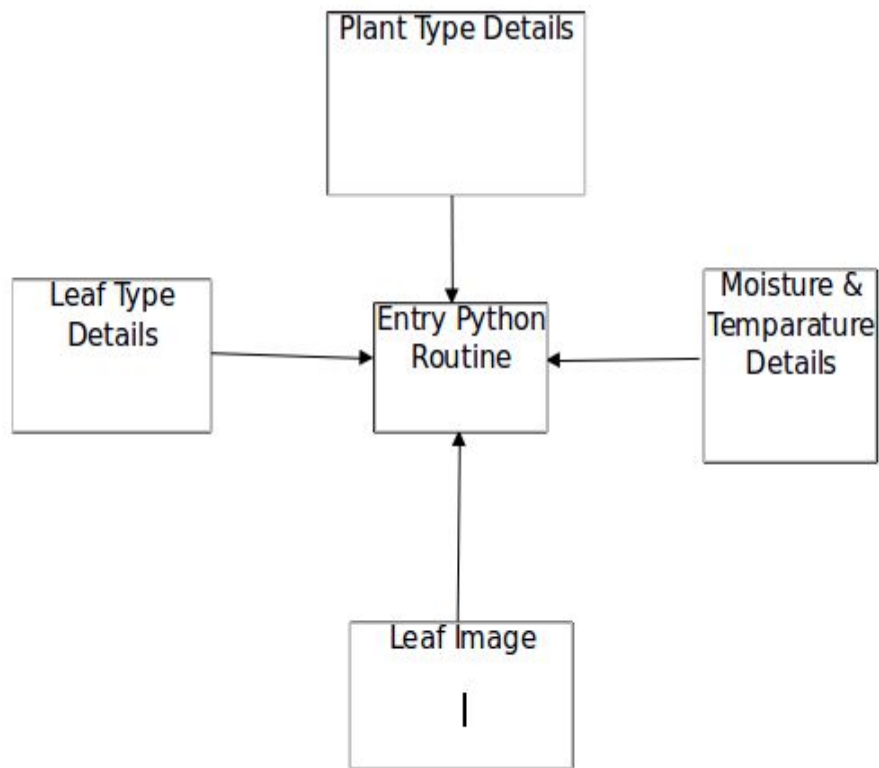


Fig 4.1: Data Flow Diagram

Details like plant type, leaf type, moisture and temperature are to be provided as Input to the system, using the corresponding user interfaces. The following section describes the screenshots of these interfaces.

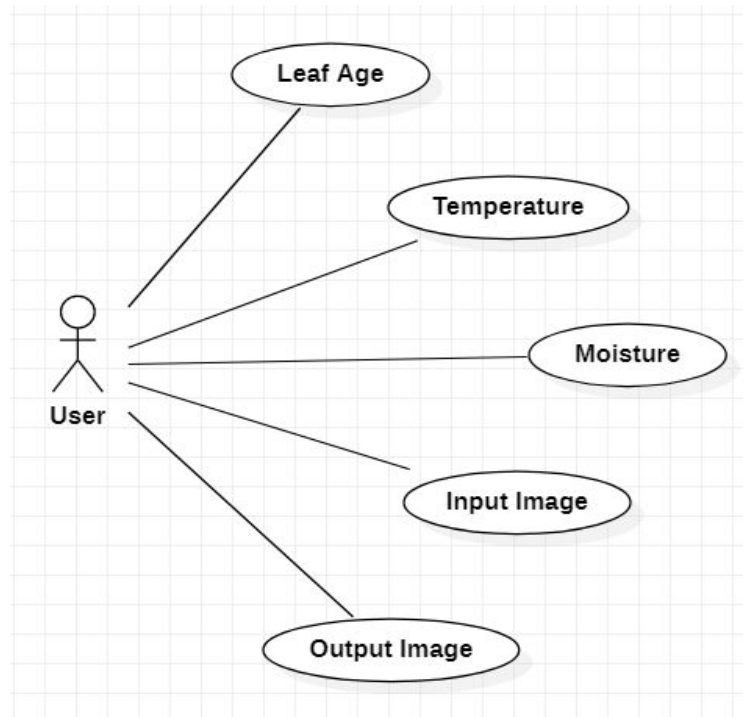


Fig 4.2: USE CASE DIAGRAM

### Description of Use case diagram

Use case diagrams are usually referred to as behaviour diagrams used to describe a set of actions (use cases) that some system or systems (subject) should or can perform in collaboration with one or more external users of the system (actors). A use case diagram at its simplest is a representation of a user's interaction with the system that shows the relationship between the user and the different use cases in which the user is involved.

As we can see the user is interacting with the system by a UI through which the customer can perform above mentioned operations like entering the plant type details, leaf type details, moisture details & Temperature details.

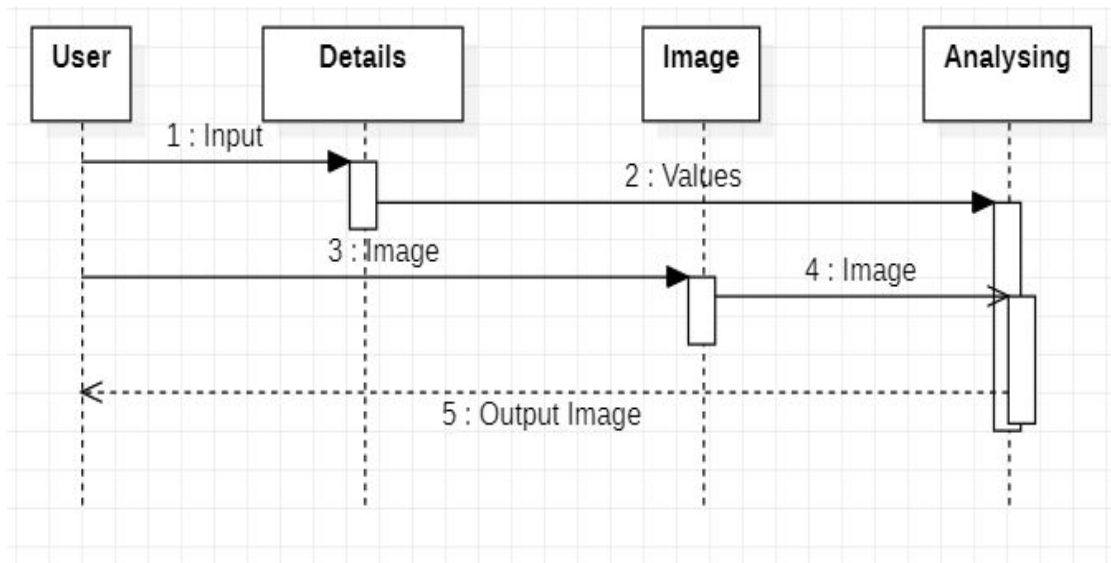


Fig 4.3: SEQUENCE DIAGRAM

#### Description of sequence diagram

A sequence diagram is an interaction diagram that shows how objects operate with one another and in what order. It is a construct of a message sequence chart. A sequence diagram shows object interactions arranged in time sequence.

From the above mentioned sequence diagram we have to go in sequence: Enter the needed details as shown in the above figure, Provide the leaf image as Input, Analyze the image & Identify the disease.

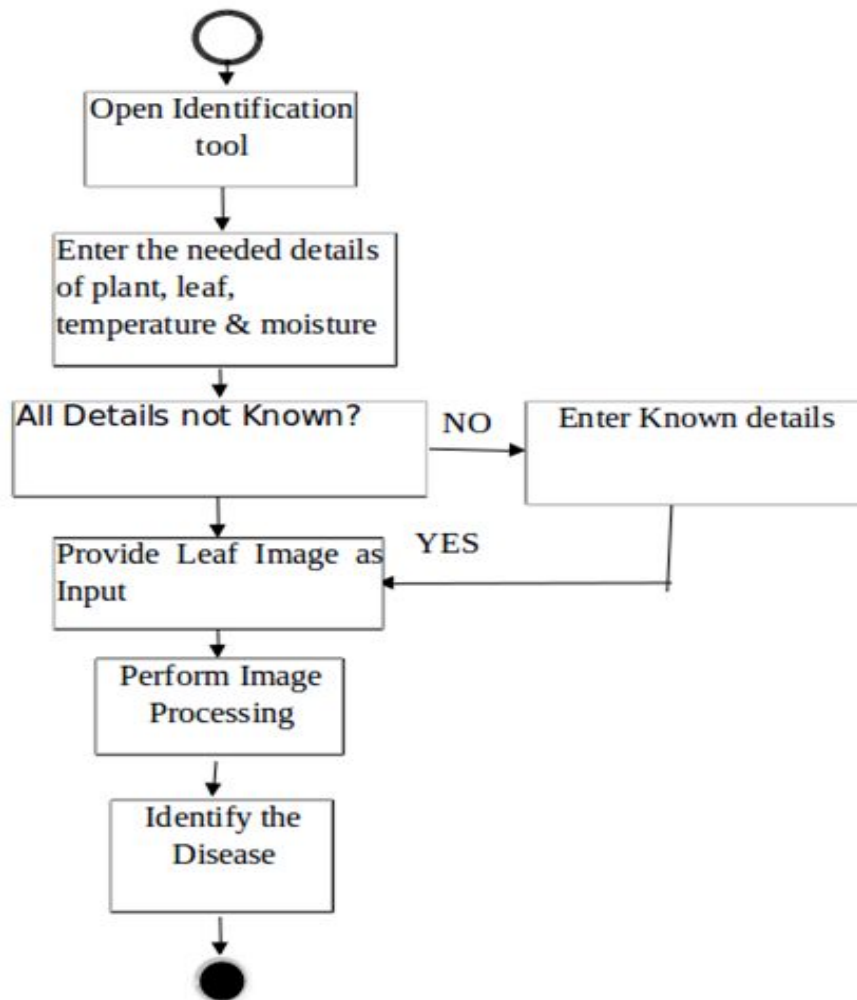


Fig 4.4: ACTIVITY DIAGRAM

### Description of Activity diagram

Activity diagram is another important diagram in UML to describe dynamic aspects of the system. Activity diagram is basically a flowchart to represent the flow from one activity to another activity. The activity can be described as an operation of the system. So, the control flow is drawn from one operation to another. In the activity diagram we can see that first we have to open the disease identification tool and then we will check whether all the needed details are there or not. If Yes, go to the next step else enter the needed details. After successfully entering the details the customer needs to provide the leaf image as input, and then click on 'Analyze Leaf Image' button which will lead to the analysis and identification of the disease.

## **4.2 Tools Used**

### **4.2.1. Python Qt Designer**

Qt is designed for developing applications and user interfaces once and deploying them across several desktop and mobile operating systems.

The easiest way to start application development with Qt is to download and install Qt 5. It contains Qt libraries, examples, documentation, and the necessary development tools, such as the Qt Creator integrated development environment (IDE).

Qt Creator provides tools for accomplishing tasks throughout the whole application development life-cycle, from creating a project to deploying the application on the target platforms. Qt Creator automates some tasks, such as creating projects, by providing wizards that guide step-by-step through the project creation process, create the necessary files, and specify settings. Also, it speeds up some tasks, such as writing code, by offering semantic highlighting, checking code syntax, code completion, refactoring actions, and other useful features.

### **Python Qt Designer**

The PyQt installer comes with a GUI builder tool called Qt Designer. Using its simple drag and drop interface, a GUI interface can be quickly built without having to write the code. It is however, not an IDE such as Visual Studio. Hence, Qt Designer does not have the facility to debug and build the application.

Creation of a GUI interface using Qt Designer starts with choosing a top-level window for the application

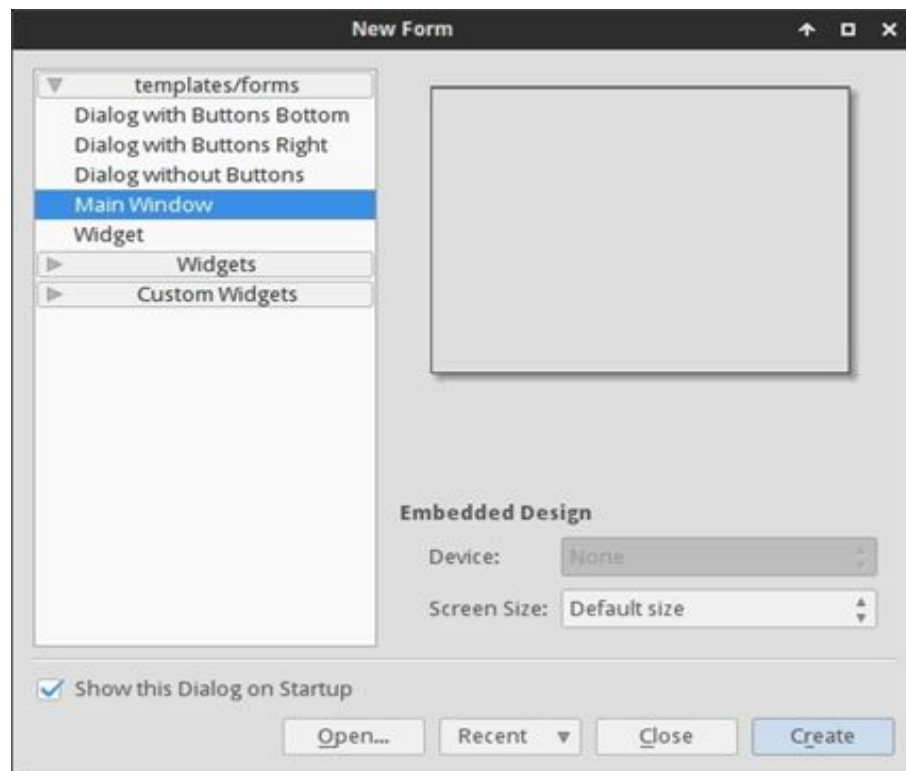


Fig 4.5: Template of Qt Designer

Drag and drop required widgets from the widget box on the left pane. Also assign value to properties of the widget laid on the form.

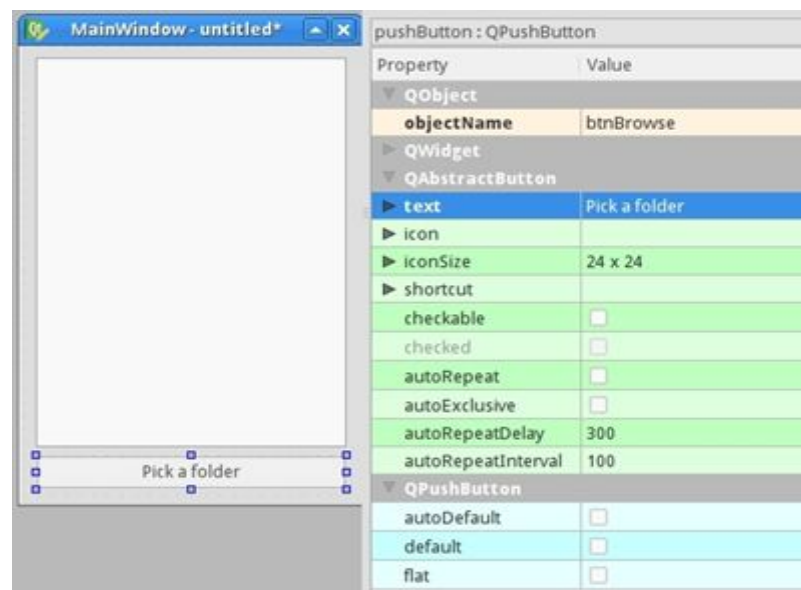


Fig 4.6: Window design attributes

The designed form is saved as demo.ui. This ui file contains XML representation of widgets and their properties in the design. This design is translated

into Python equivalent by using pyuic4 command line utility. This utility is a wrapper for the ui module. The usage of pyuic4 is as follows

### **Widgets & Description**

1. QLabel: A QLabel object acts as a placeholder to display non-editable text or image, or a movie of animated GIF. It can also be used as a mnemonic key for other widgets.
2. QLineEdit: QLineEdit object is the most commonly used input field. It provides a box in which one line of text can be entered. In order to enter multi-line text, QTextEdit object is required.
3. QPushButton: In PyQt API, the QPushButton class object presents a button which when clicked can be programmed to invoke a certain function.

### **QPushButton**

The QPushButton widget provides a command button.

The push button, or command button, is perhaps the most commonly used widget in any graphical user interface. Push (click) a button to command the computer to perform some action, or to answer a question. Typical buttons are OK, Apply, Cancel, Close, Yes, No and Help.

A command button is rectangular and typically displays a text label describing its action. A shortcut key can be specified by preceding the preferred character with an ampersand in the text. For example:

```
QPushButton *button = new QPushButton("&Download", this);
```

In this example the shortcut is Alt+D. See the QShortcut documentation for details (to display an actual ampersand, use '&&').

Push buttons display a textual label, and optionally a small icon. These can be set using the constructors and changed later using setText() and setIcon(). If the button is disabled the appearance of the text and icon will be manipulated with respect to the GUI style to make the button look "disabled".

A push button emits the signal clicked() when it is activated by the mouse, the Spacebar or by a keyboard shortcut. Connect to this signal to perform the button's action. Push buttons also provide less commonly used signals, for example, pressed() and released().

Command buttons in dialogs are by default auto-default buttons, i.e. they become the default push button automatically when they receive the keyboard input

focus. A default button is a push button that is activated when the user presses the Enter or Return key in a dialog. Note that auto-default buttons reserve a little extra space which is necessary to draw a default-button indicator. If not, call `setAutoDefault(false)`.

Being so central, the button widget has grown to accommodate a great many variations in the past decade. The Microsoft style guide now shows about ten different states of Windows push buttons and the text implies that there are dozens more when all the combinations of features are taken into consideration.

The most important modes or states are:

- Available or not (grayed out, disabled).
- Standard push button, toggling push button or menu button.
- On or off (only for toggling push buttons).
- Default or normal. The default button in a dialog can generally be "clicked" using the Enter or Return key.
- Auto-repeat or not.
- Pressed down or not.

As a general rule, use a push button when the application or dialog window performs an action when the user clicks on it (such as Apply, Cancel, Close and Help) and when the widget is supposed to have a wide, rectangular shape with a text label. Small, typically square buttons that change the state of the window rather than performing an action (such as the buttons in the top-right corner of the `QFileDialog`) are not command buttons, but tool buttons. Qt provides a special class (`QToolButton`) for these buttons.

If need toggle behavior (see `setCheckable()`) or a button that auto-repeats the activation signal when being pushed down like the arrows in a scroll bar (see `setAutoRepeat()`). When in doubt, use a tool button.

A variation of a command button is a menu button. These provide not just one command, but several, since when they are clicked they pop up a menu of options. Use the method `setMenu()` to associate a popup menu with a push button.

Other classes of buttons are option buttons (see `QRadioButton`) and check boxes (see `QCheckBox`).



In Qt, the `QAbstractButton` base class provides most of the modes and other API, and `QPushButton` provides GUI logic. See `QAbstractButton` for more information about the API.

## **QLineEdit**

The `QLineEdit` widget is a one-line text editor.

A line edit allows the user to enter and edit a single line of plain text with a useful collection of editing functions, including undo and redo, cut and paste, and drag and drop.

By changing the `echoMode()` of a line edit, it can also be used as a "write-only" field, for inputs such as passwords.

The length of the text can be constrained to `maxLength()`. The text can be arbitrarily constrained using a `validator()` or an `inputMask()`, or both. When switching between a validator and an input mask on the same line edit, it is best to clear the validator or input mask to prevent undefined behaviour.

A related class is `QTextEdit` which allows multi-line, rich text editing.

You can change the text with `setText()` or `insert()`. The text is retrieved with `text()`; the displayed text (which may be different, see `EchoMode`) is retrieved with `displayText()`. Text can be selected with `setSelection()` or `selectAll()`, and the selection can be `cut()`, `copy()`ied and `paste()`d. The text can be aligned with `setAlignment()`.

When the text changes the `textChanged()` signal is emitted; when the text changes other than by calling `setText()` the `textEdited()` signal is emitted; when the cursor is moved the `cursorPositionChanged()` signal is emitted; and when the Return or Enter key is pressed the `returnPressed()` signal is emitted.

When editing is finished, either because the line edit lost focus or Return/Enter is pressed the `editingFinished()` signal is emitted.

Note that if there is a validator set on the line edit, the `returnPressed()/editingFinished()` signals will only be emitted if the validator returns `QValidator.Acceptable`.

By default, `QLineEdit`s have a frame as specified by the Windows and Motif style guides; turn it off by calling `setFrame(false)`.

The default key bindings are described below. The line edit also provides a context menu (usually invoked by a right mouse click) that presents some of these editing options.

## **QMainWindow**

The QMainWindow class provides a main application window.

### **Qt Main Window Framework**

A main window provides a framework for building an application's user interface. Qt has QMainWindow and its related classes for main window management. QMainWindow has its own layout which can add QToolBars, QDockWidgets, a QMenuBar, and a QStatusBar. The layout has a centre area that can be occupied by any kind of widget.

Note: Creating a main window without a central widget is not supported. Need to have a central widget even if it is just a placeholder.

#### **Creating Main Window Components**

A central widget will typically be a standard Qt widget such as a QTextEdit or a QGraphicsView. Custom widgets can also be used for advanced applications. Can be set as the central widget with setCentralWidget().

Main windows have either a single (SDI) or multiple (MDI) document interface.

We will now examine each of the other widgets that can be added to a main window. We give examples on how to create and add them.

#### **Creating Menus**

Qt implements menus in QMenu and QMainWindow keeps them in a QMenuBar. QActions are added to the menus, which display them as menu items.

Add new menus to the main window's menu bar by calling menuBar(), which returns the QMenuBar for the window, and then add a menu with QMenuBar::addMenu().

#### **Creating Main Window Components**

A central widget will typically be a standard Qt widget such as a QTextEdit or a QGraphicsView. Custom widgets can also be used for advanced applications.

Main windows have either a single (SDI) or multiple (MDI) document interface.

We will now examine each of the other widgets that can be added to a main window. We give examples on how to create and add them.

An example of how to create menus follows:

```
void MainWindow.createMenus()

{

fileMenu = menuBar()->addMenu(tr("&File"));

fileMenu->addAction(newAct);

fileMenu->addAction(openAct);

fileMenu->addAction(saveAct);
```

The `createPopupMenu()` function creates popup menus when the main window receives context menu events. The default implementation generates a menu with the checkable actions from the dock widgets and toolbars. Also can reimplement `createPopupMenu()` for a custom menu.

## **Creating Toolbars**

Toolbars are implemented in the `QToolBar` class. Add a toolbar to a main window with `addToolBar()`. Control the initial position of toolbars by assigning them to a specific `Qt.ToolBarArea`. It can split an area by inserting a toolbar break - think of this as a line break in text editing - with `addToolBarBreak()` or `insertToolBarBreak()`. Also restrict placement by the user with `QToolBar.setAllowedAreas()` and `QToolBar.setMovable()`.

The size of toolbar icons can be retrieved with `iconSize()`. The sizes are platform dependent. Also can alter the appearance of all tool buttons in the toolbars with `setToolButtonStyle()`.

An example of toolbar creation follows:

```
void MainWindow.createToolBars()

{

fileToolBar = addToolBar(tr("File"));
```

```
fileToolBar->addAction(newAct);
```

Function `isWidgetType()` returns whether an object is actually a widget. It is much faster than `inherits("QWidget")`.

Some `QObject` functions, e.g. `children()`, `objectTrees()` and `queryList()` return a `QObjectList`. A `QObjectList` is a `QPtrList` of `QObject`s. `QObjectList`s support the same operations as `QPtrList`s and have an iterator class, `QObjectListIt`. To convert the design file to python code saved as `design.py`, use `cd` command to change to the directory holding the `design.ui` file and simply run:

```
$ pyuic4 design.ui -o design.py
```

If you want to specify full path for either input or output file you can do that like this:

```
$ pyuic4 path/to/design.ui -o output/path/to/design.py
```

Writing the code

Now that we have the `design.py` file with the necessary design part of the application we can create our main application code and logic.

Create a file `main.py` in the same folder as your `design.py` file.

Using the design

For the application we'll need the following python modules imported:

```
from PyQt4 import QtGui
```

```
import sys
```

We also need the design code we created in the previous steps so add this too:

```
import design
```

Since the design file will be completely overwritten each time we change something in the design and recreate it we will not be writing any code in it, instead we'll create a new class e.g. `ExampleApp` that we'll combine with the design code so that we can use all of its features, like this:

```
class ExampleApp(QtGui.QMainWindow, design.Ui_MainWindow):
```

```

def __init__(self, parent=None):

super(ExampleApp, self).__init__(parent)

self.setupUi(self)

```

In that class we'll interact with the GUI elements, add connections and everything else we need. But first we'll need to initialize that class on our code startup, we'll handle the class instance creation and other stuff in our main() function:

```

def main():

app = QtGui.QApplication(sys.argv)

form = ExampleApp()

form.show()

```

```

app.exec_()

```

And to execute that main function:

```

if __name__ == '__main__':

main()

```

In the end our whole main.py file looks like this (with short explanations of the code):

```

from PyQt4 import QtGui    # Import the PyQt4 module we'll need

import sys                # We need sys so that we can pass argv to QApplication

import design              # This file holds our MainWindow and all design related things

# it also keeps events etc that we defined in Qt Designer

class ExampleApp(QtGui.QMainWindow, design.Ui_MainWindow):

def __init__(self):

super(self.__class__, self).__init__()

self.setupUi(self)        # This is defined in design.py file automatically

# It sets up layout and widgets that are defined

```

```

def main():

app = QtGui.QApplication(sys.argv)          # A new instance of QApplication

form = ExampleApp()                        # We set the form to be our ExampleApp (design)

form.show()                               # Show the form

app.exec_()                                # and execute the app

if __name__ == '__main__':                 # if we're running file directly and not importing
it                                         it

main()                                     # run the main function

```

Running that will bring up our app running completely from python code!

But clicking the button isn't doing anything, so we need to implement those features ourselves.

Implementing functions

(All of the following code is written inside the ExampleApp class)

Let's start with the "Pick a folder" button.

To connect a button event, such as clicked, to a function use the following code:

```
self.btnBrowse.clicked.connect(self.browse_folder)
```

And add it to the `__ini__` method of our ExampleApp class so that it's set up when the application starts.

### **Code Explanation:**

`self.btnBrowse` - `btnBrowse` is the name of the object we defined in Qt Designer. `self` is self explanatory and means that it belongs to current class.

`clicked` - the event we want to connect. Various elements have various events, for example list widgets have `itemSelectionChanged` etc.

`connect()` - used to specify with what to connect it with. In our example:

`self.browse_folder` - simply a function name that we'll write inside our ExampleApp class:

For getting the directory browser dialog we can use the built in `QtGui.QFileDialog.getExistingDirectory` method like this:

```
directory = QtGui.QFileDialog.getExistingDirectory(self,"Pick a folder")
```

If the user picks a directory the directory variable will be equal to absolute path of the selected directory, otherwise it's `None`. To avoid running our code any further if the user cancels the folder browse dialog use `if directory:` statement.

To list the directory contents, need to add `os` to our imports:

```
import os
```

and to get the current file list use `os.listdir(path)`.

For adding items to the `listWidget` we use `addItem()` method on it, and to clear all existing items simply use `self.listWidget.clear()`

In the end our `browse_folder` function looks something like this:

```
def browse_folder(self):
```

```
    self.listWidget.clear()
```

```
    directory = QtGui.QFileDialog.getExistingDirectory(self,"Pick a folder")
```

```
    if directory:
```

```
        for file_name in os.listdir(directory):
```

```
            self.listWidget.addItem(file_name)
```

Now run the app by typing `python main.py` and get the layout one has designed and picking the folder will populate the list with folder items.

Finished `main.py`

```
from PyQt4 import QtGui    # Import the PyQt4 module we'll need
```

```
import sys                # We need sys so that we can pass argv to QApplication
```

```
import design              # This file holds our Main Window and all design related things
```

```
# it also keeps events etc. that we defined in Qt Designer
```

```

import os      # For listing directory methods

class ExampleApp(QtGui.QMainWindow, design.Ui_MainWindow):

    def __init__(self):

        super(self.__class__, self).__init__()

        self.setupUi(self) # This is defined in design.py file automatically

        # It sets up layout and widgets that are defined

        self.btnBrowse.clicked.connect(self.browse_folder) # When the button is pressed

        # Execute browse_folder function

        def browse_folder(self):

            self.listWidget.clear() # In case there are any existing elements in the list

            directory = QtGui.QFileDialog.getExistingDirectory(self,

                "Pick a folder")

            # execute getExistingDirectory dialog and set the directory variable to be equal

            # to the user selected directory

            if directory: # if user didn't pick a directory don't continue

                for file_name in os.listdir(directory): # for all files, if any, in the directory

                    self.listWidget.addItem(file_name) # add file to the listWidget

        def main():

            app = QtGui.QApplication(sys.argv) # A new instance of QApplication

            form = ExampleApp() # We set the form to be our ExampleApp (design)

            form.show() # Show the form

            app.exec_() # and execute the app

            if __name__ == '__main__': # if we're running file directly and not importing it

```



```
main() # run the main function
```

That's the basic logic of using Qt Designer and PyQt to design and develop a GUI application.

### **Image Template matching algorithm:**

Template Matching is a method for searching and finding the location of a template image in a larger image. OpenCV comes with a function `cv2.matchTemplate()` for this purpose. It simply slides the template image over the input image (as in 2D convolution) and compares the template and patch of input image under the template image. Several comparison methods are implemented in OpenCV.

```
import cv2

import numpy as np

from matplotlib import pyplot as plt

img = cv2.imread('messi5.jpg',0)

img2 = img.copy()

template = cv2.imread('template.jpg',0)

w, h = template.shape[::-1]

# All the 6 methods for comparison in a list

methods = ['cv2.TM_CCOEFF', 'cv2.TM_CCOEFF_NORMED', 'cv2.TM_CCORR',

           'cv2.TM_CCORR_NORMED', 'cv2.TM_SQDIFF',

           'cv2.TM_SQDIFF_NORMED']

for meth in methods:

    img = img2.copy()

    method = eval(meth)

    # Apply template Matching

    res = cv2.matchTemplate(img,template,method)

    min_val, max_val, min_loc, max_loc = cv2.minMaxLoc(res)
```

```

# If the method is TM_SQDIFF or TM_SQDIFF_NORMED, take minimum
if method in [cv2.TM_SQDIFF, cv2.TM_SQDIFF_NORMED]:

    top_left = min_loc

else:

    top_left = max_loc

bottom_right = (top_left[0] + w, top_left[1] + h)

cv2.rectangle(img,top_left, bottom_right, 255, 2)

plt.subplot(121),plt.imshow(res,cmap = 'gray')

plt.title('Matching Result'), plt.xticks([], plt.yticks([]))

plt.subplot(122),plt.imshow(img,cmap = 'gray')

plt.title('Detected Point'), plt.xticks([], plt.yticks([]))

plt.suptitle(meth)

plt.show()

```

Methods used in template matching:

`cv2.matchTemplate(image, templ, method[, result]) → result`

#### 4.2.2.Parameters:

- `image` – Image where the search is running. It must be 8-bit or 32-bit floating-point.
- `templ` – Searched template. It must be not greater than the source image and have the same data type.
- `result` – Map of comparison results. It must be single-channel 32-bit floating-point. If `image` is `src` and `templ` is `tem`, then `result` is `src - tem`.
- `method` – Parameter specifying the comparison method (see below).

The function slides through `image`, compares the overlapped patches of size `templ` against `templ` using the specified method and stores the comparison results in `result`. Here are the formulae for the available comparison methods (`src` denotes `image`, `tem` template, `result`). The summation is done over template and/or the image patch:

1. `method=CV_TM_SQDIFF`

$$R(x, y) = \sum_{x', y'} (T(x', y') - I(x + x', y + y'))^2$$

2. method=CV\_TM\_SQDIFF\_NORMED

$$R(x, y) = \frac{\sum_{x', y'} (T(x', y') - I(x + x', y + y'))^2}{\sqrt{\sum_{x', y'} T(x', y')^2 \cdot \sum_{x', y'} I(x + x', y + y')^2}}$$

3. method=CV\_TM\_CCORR

$$R(x, y) = \sum_{x', y'} (T(x', y') \cdot I(x + x', y + y'))$$

4. method=CV\_TM\_CCORR\_NORMED

$$R(x, y) = \frac{\sum_{x', y'} (T(x', y') \cdot I(x + x', y + y'))}{\sqrt{\sum_{x', y'} T(x', y')^2 \cdot \sum_{x', y'} I(x + x', y + y')^2}}$$

5. method=CV\_TM\_CCOEFF

$$R(x, y) = \sum_{x', y'} (T'(x', y') \cdot I'(x + x', y + y'))$$

where

$$T'(x', y') = T(x', y') - 1/(w \cdot h) \cdot \sum_{x'', y''} T(x'', y'')$$

$$I'(x + x', y + y') = I(x + x', y + y') - 1/(w \cdot h) \cdot \sum_{x'', y''} I(x + x'', y + y'')$$

6. method=CV\_TM\_CCOEFF\_NORMED

$$R(x, y) = \frac{\sum_{x', y'} (T'(x', y') \cdot I'(x + x', y + y'))}{\sqrt{\sum_{x', y'} T'(x', y')^2 \cdot \sum_{x', y'} I'(x + x', y + y')^2}}$$

After the function finishes the comparison, the best matches can be found as global minimums(when CV\_TM\_SQDIFF was used) or maximums(when CV\_TM\_CCORR or CV\_TM\_CCOEFF was used) using the minMaxLoc() function. In case of a color image, template summation in the numerator and each sum in the denominator is done over all of the channels and separate mean values are used for each channel. That is,

the function can take a color template and a color image. The result will still be a single-channel image, which is easier to analyze.

# **CHAPTER 5**

## **IMPLEMENTATION**

### **5.1 Languages**

#### **PYTHON**

Python was conceived in the late 1980s, and its implementation began in December 1989 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.

Python is an interpreted high-level programming language for general-purpose programming. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace. It provides constructs that enable clear programming on both small and large scales. Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object oriented, imperative, functional and procedural, and has a large and comprehensive standard library.

Python interpreters are available for many operating systems. CPython, the reference implementation of Python, is open source software and has a community-based development model, as do nearly all of its variant implementations.

#### **FEATURES AND PHILOSOPHY**

Python is a multi-paradigm programming language. Object-oriented programming and structured programming are fully supported, and many of its features support functional programming and aspect-oriented programming (including meta programming and meta objects (magic methods)). Many other paradigms are supported via extensions, including design by contract and logic programming.

Python uses dynamic typing, and a combination of reference counting and a cycle detecting garbage collector for memory management. It also features dynamic name resolution (late binding), which binds method and variable names during program execution.

## 5.2 Sample Code

### **idplf1.py**

```
import sys
import os
from idplf import *
from PyQt5 import QtWidgets, QtGui, QtCore

class MyForm(QtWidgets.QMainWindow):
    def __init__(self, parent=None):
        QtWidgets.QWidget.__init__(self, parent)
        self.ui = Ui_MainWindow()
        self.ui.setupUi(self)
        self.ui.pushButton.clicked.connect(self.ldetails)
        self.ui.pushButton_2.clicked.connect(self.tempdetails)
        self.ui.pushButton_3.clicked.connect(self.moistdetails)
        self.ui.pushButton_4.clicked.connect(self.limage)
        self.ui.pushButton_5.clicked.connect(self.pdetails)

    def ldetails(self):
        os.system("python ltype1.py")

    def pdetails(self):
        os.system("python ptype1.py")

    def tempdetails(self):
        os.system("python temprt1.py")

    def moistdetails(self):
        os.system("python moist1.py")

    def limage(self):
        os.system("python limage1.py")

if __name__ == "__main__":
    app = QtWidgets.QApplication(sys.argv)
    myapp = MyForm()
    myapp.show()
    sys.exit(app.exec_())
```

### **Disease1.py**

```
import sys
import os
from disease import *
from PyQt5 import QtWidgets, QtGui, QtCore

import sqlite3
con = sqlite3.connect('idplf1')

class MyForm(QtWidgets.QMainWindow):
    def __init__(self,parent=None):
        QtWidgets.QWidget.__init__(self,parent)
        self.ui = Ui_MainWindow()
        self.ui.setupUi(self)
        self.ui.pushButton.clicked.connect(self.leafimage)

    def leafimage(self):
        os.system("python limage1.py")

if __name__ == "__main__":
    app = QtWidgets.QApplication(sys.argv)
    myapp = MyForm()
    myapp.show()
    sys.exit(app.exec_())
```

### **Ima\_match1.py**

```
import cv2
import numpy as np
small_image = cv2.imread('eblight1.jpg')
large_image = cv2.imread('eblight.jpg')

result = cv2.matchTemplate(small_image, large_image,
cv2.TM_SQDIFF_NORMED)

mn,_,mnLoc,_ = cv2.minMaxLoc(result)

MPx,MPy = mnLoc

trows,tcols = small_image.shape[:2]

cv2.rectangle(large_image, (MPx,MPy),(MPx+tcols,MPy+trows),(0,0,255),2)
```

```
cv2.imshow('output',large_image)
```

```
cv2.waitKey(7000)
```

### **Limage1.py**

```
import sys
```

```
import os
```

```
from limage import *
```

```
import sqlite3
```

```
con = sqlite3.connect('idplf1')
```

```
class MyForm(QtWidgets.QMainWindow):
```

```
    def __init__(self,parent=None):
```

```
        QtWidgets.QWidget.__init__(self,parent)
```

```
        self.ui = Ui_MainWindow()
```

```
        self.ui.setupUi(self)
```

```
        self.ui.pushButton.clicked.connect(self.analyse)
```

```
    def analyse(self):
```

```
        s1 = str(self.ui.lineEdit.text())
```

```
        if (s1=='eblight.jpg'):
```

```
            os.system("python ima_match1.py")
```

```
        elif (s1=='anthracnose.jpg'):
```

```
            os.system("python ima_match2.py")
```

```
        elif (s1=='downy.jpg'):
```

```
            os.system("python ima_match3.py")
```

```
        elif (s1=='lblight.jpg'):
```

```
            os.system("python ima_match4.py")
```

```
        elif (s1=='lspot.jpg'):
```

```
            os.system("python ima_match5.py")
```

```
if __name__ == "__main__":
```

```
    app = QtWidgets.QApplication(sys.argv)
```

```
    myapp = MyForm()
```

```
    myapp.show()
```

```
    sys.exit(app.exec_())
```

### **Ltype1.py**

```
import sys
```

```
import os
```



```

from ltype import *
from PyQt5 import QtWidgets, QtGui, QtCore

import sqlite3
con = sqlite3.connect('idplf1')

class MyForm(QtWidgets.QMainWindow):
    def __init__(self,parent=None):
        QtWidgets.QWidget.__init__(self,parent)
        self.ui = Ui_MainWindow()
        self.ui.setupUi(self)
        self.ui.pushButton.clicked.connect(self.insertvalues)

    def insertvalues(self):
        with con:
            cur = con.cursor()
            s4 = str(self.ui.lineEdit_4.text())
            s5 = str(self.ui.lineEdit_5.text())
            s6 = str(self.ui.lineEdit_6.text())
            s7 = str(self.ui.lineEdit_7.text())
            if (s4=='Y'):
                cur.execute('INSERT INTO ltype(leaftype) VALUES("unknown")')
            elif (s5=='Y'):
                cur.execute('INSERT INTO ltype(leaftype) VALUES("Younger")')
            elif (s6=='Y'):
                cur.execute('INSERT INTO ltype(leaftype) VALUES("Mid_Aged")')
            elif (s7=='Y'):
                cur.execute('INSERT INTO ltype(leaftype) VALUES("older")')
            con.commit()

    def temprtdetails(self):
        os.system("python temprt1.py")

if __name__ == "__main__":
    app = QtWidgets.QApplication(sys.argv)
    myapp = MyForm()
    myapp.show()
    sys.exit(app.exec_())

```

### **Moist1.py**

```

import sys
import os
from moist import *

```

```

from PyQt5 import QtWidgets, QtGui, QtCore

import sqlite3
con = sqlite3.connect('idplf1')

class MyForm(QtWidgets.QMainWindow):
    def __init__(self,parent=None):
        QtWidgets.QWidget.__init__(self,parent)
        self.ui = Ui_MainWindow()
        self.ui.setupUi(self)
        self.ui.pushButton.clicked.connect(self.insertvalues)

    def insertvalues(self):
        with con:
            cur = con.cursor()
            s4 = str(self.ui.lineEdit_4.text())
            s5 = str(self.ui.lineEdit_5.text())
            s6 = str(self.ui.lineEdit_6.text())
            s7 = str(self.ui.lineEdit_7.text())
            if (s4=='Y'):
                cur.execute('INSERT INTO moist(moisture) VALUES("unknown")')
            elif (s5=='Y'):
                cur.execute('INSERT INTO moist(moisture) VALUES("Low")')
            elif (s6=='Y'):
                cur.execute('INSERT INTO moist(moisture) VALUES("Medium")')
            elif (s7=='Y'):
                cur.execute('INSERT INTO moist(moisture) VALUES("High")')
            con.commit()

    def imagedetails(self):
        os.system("python disease1.py")

if __name__ == "__main__":
    app = QtWidgets.QApplication(sys.argv)
    myapp = MyForm()
    myapp.show()
    sys.exit(app.exec_())

```

### **Ptype1.py**

```

import sys
import os
from ptype import *
from PyQt5 import QtWidgets, QtGui, QtCore

```

```

import sqlite3
con = sqlite3.connect('idplf1')

class MyForm(QtWidgets.QMainWindow):
    def __init__(self,parent=None):
        QtWidgets.QWidget.__init__(self,parent)
        self.ui = Ui_MainWindow()
        self.ui.setupUi(self)
        self.ui.pushButton.clicked.connect(self.insertvalues)

    def insertvalues(self):
        with con:
            cur = con.cursor()
            s4 = str(self.ui.lineEdit_4.text())
            s5 = str(self.ui.lineEdit_5.text())
            s6 = str(self.ui.lineEdit_6.text())
            s7 = str(self.ui.lineEdit_7.text())
            s8 = str(self.ui.lineEdit_8.text())
            s9 = str(self.ui.lineEdit_9.text())
            s10 = str(self.ui.lineEdit_10.text())
            s11 = str(self.ui.lineEdit_11.text())
            s12 = str(self.ui.lineEdit_12.text())
            s13 = str(self.ui.lineEdit_13.text())
            if (s4=='Y'):
                cur.execute('INSERT INTO ptype(planttype) VALUES("unknown")')
            elif (s5=='Y'):
                cur.execute('INSERT INTO ptype(planttype) VALUES("Blue Berries")')
            elif (s6=='Y'):
                cur.execute('INSERT INTO ptype(planttype) VALUES("Goji Berries")')
            elif (s7=='Y'):
                cur.execute('INSERT INTO ptype(planttype) VALUES("Black Berries")')
            elif (s8=='Y'):
                cur.execute('INSERT INTO ptype(planttype) VALUES("Straw Berries")')
            elif (s9=='Y'):
                cur.execute('INSERT INTO ptype(planttype) VALUES("Rasp Berries")')
            elif (s10=='Y'):
                cur.execute('INSERT INTO ptype(planttype) VALUES("Bil Berries")')
            elif (s11=='Y'):
                cur.execute('INSERT INTO ptype(planttype) VALUES("Acai Berries")')
            elif (s12=='Y'):
                cur.execute('INSERT INTO ptype(planttype) VALUES("Grape Berries")')
            elif (s13=='Y'):
                cur.execute('INSERT INTO ptype(planttype) VALUES("CranBerries")')
            con.commit()

```

```

def leafdetails(self):
    os.system("python ltype1.py")

if __name__ == "__main__":
    app = QtWidgets.QApplication(sys.argv)
    myapp = MyForm()
    myapp.show()
    sys.exit(app.exec_())

```

### **Temprrt1.py**

```

import sys
import os
from temprrt import *
from PyQt5 import QtWidgets, QtGui, QtCore

import sqlite3
con = sqlite3.connect('idplf1')

class MyForm(QtWidgets.QMainWindow):
    def __init__(self,parent=None):
        QtWidgets.QWidget.__init__(self,parent)
        self.ui = Ui_MainWindow()
        self.ui.setupUi(self)
        self.ui.pushButton.clicked.connect(self.insertvalues)

    def insertvalues(self):
        with con:
            cur = con.cursor()
            s4 = str(self.ui.lineEdit_4.text())
            s5 = str(self.ui.lineEdit_5.text())
            s6 = str(self.ui.lineEdit_6.text())
            s7 = str(self.ui.lineEdit_7.text())
            if (s4=='Y'):
                cur.execute('INSERT INTO temprrt(temperature) VALUES("unknown")')
            elif (s5=='Y'):
                cur.execute('INSERT INTO temprrt(temperature) VALUES("Low")')
            elif (s6=='Y'):
                cur.execute('INSERT INTO temprrt(temperature) VALUES("Medium")')
            elif (s7=='Y'):
                cur.execute('INSERT INTO temprrt(temperature) VALUES("High")')
            con.commit()

```

```

def moistdetails(self):
    os.system("python moist1.py")

if __name__ == "__main__":
    app = QtWidgets.QApplication(sys.argv)
    myapp = MyForm()
    myapp.show()
    sys.exit(app.exec_())

```

### 5.3 Database Connectivity

SQLite is a C library that implements an SQL database engine. It is a Relational Database Management System (or RDBMS). Most of the SQL databases work with the client/server model. Take MySQL for an example. To enter and receive data from a MySQL database to send a request to the MySQL server, which on reception will provide the appropriate response. Contrary to MySQL, SQLite databases are operated directly from the disk. There is no need to create requests to the server.

#### Meta Commands

Meta Commands are used to define output format for tables, examine databases and for other administrative operations. They always start with a *dot*. Even *.help* is a meta command. Here are some that will frequently come in handy:

Command	Description
<b>.show</b>	<b>Displays current settings for various parameters</b>
<b>.databases</b>	<b>Provides database names and files</b>
<b>.quit</b>	<b>Quit sqlite3 program</b>
<b>.tables</b>	<b>Show current tables</b>
<b>.schema</b>	<b>Display schema of table</b>
<b>.header</b>	<b>Display or hide the output table header</b>

<b>.mode</b>	<b>Select mode for the output table</b>
<b>.dump</b>	<b>Dump database in SQL text format</b>

## Standard Commands

Let us go through the standard commands in sqlite3. Meta commands are issued to examine a database. Standard SQL commands are issued to operate on a database. Standard Commands can be classified into three groups:

- **Data Definition Language:** It provides the storage structure and methods to access data from the database system.
  - CREATE
  - ALTER
  - DROP
- **Data Manipulation Language:** It enables users to manipulate (add/ modify/ delete) data.
  - INSERT
  - UPDATE
  - DELETE
- **Data Query Language:** It enables users to retrieve required data from the database.
  - SELECT

## 5.4.Outcome of above process:

Screenshots and description of files:

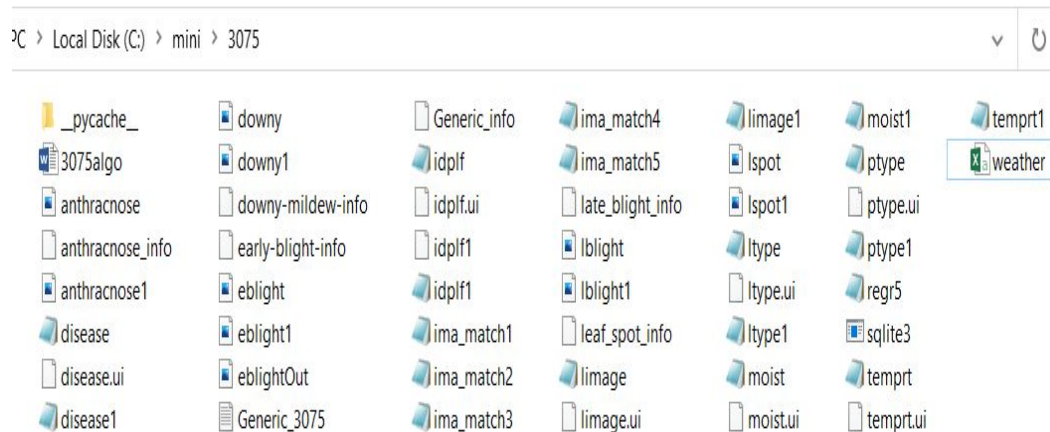


Fig 5.1 : Screenshot showing the files created during the project

The above screen shot shows different files created during this project. There are four different types of files: (1) .jpg files (2) .ui files (3).py files and (4).txt files

.jpg files contains the needed images like leaf and disease images that are needed in this project.

.ui files are the user interface files, created by using PyQt layout editor

.py files are python program files created either manually, or automatically. For instance, each .ui file has a corresponding .py file that is created automatically by using the PyUIC tool. .txt files contains the generic useful information about the project.

Idplf1.py, is the entry program for this project. Execution of this python program leads to the entry screen as follows:

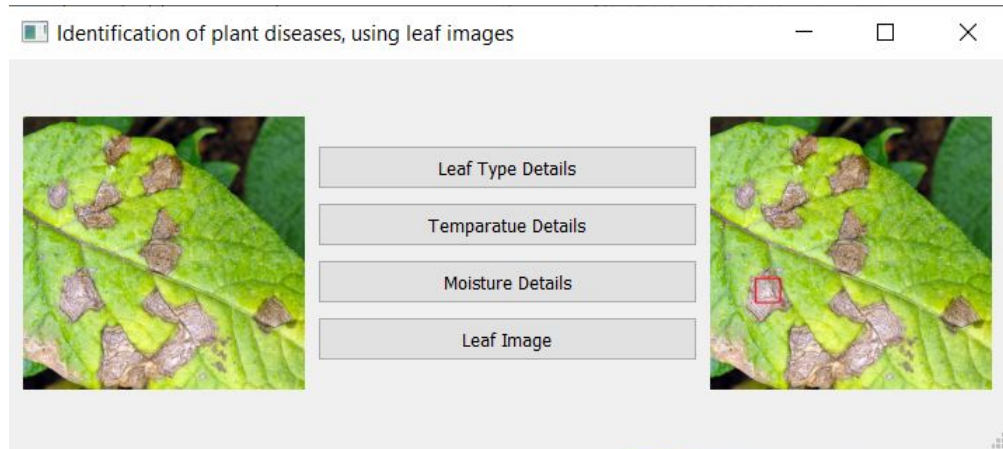


Fig 5.2: User interface screen

This entry screen consists of five push buttons.

The first button leads to the invoking of ltype1.py program, which results in a screen, where the user can enter the details of the leaves.

Upon clicking the Second button, temp1.py program is instantiated resulting in a screen , by using which the user can enter the temperature details.

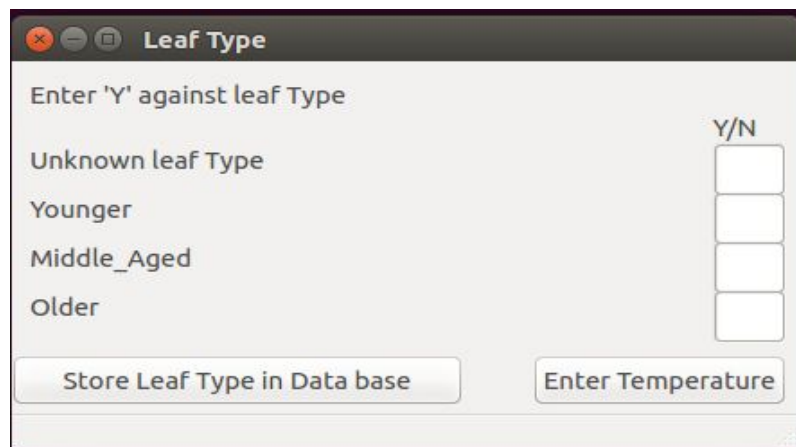
Upon clicking the Third button, moist1.py program is instantiated resulting in a screen , by using which the user can enter the moisture details.

The Fourth button leads to the invoking of ltype1.py program, which results in a screen, where the user can enter the details of the leaves.

The user can enter the leaf details like whether the leaf is young/ middle\_aged/ old/ unknown, by using the following screen.



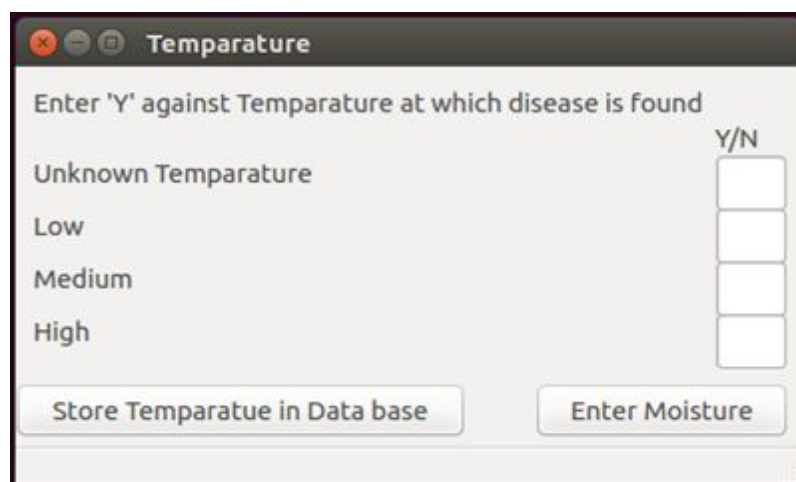
The user can enter the leaf details like whether the leaf is young/ middle\_aged/ old/ unknown, by using the following screen.



The screenshot shows a window titled "Leaf Type" with a standard OS title bar (red, yellow, and green buttons). The main content area has a light gray background. At the top, it says "Enter 'Y' against leaf Type". Below this, there are four labels: "Unknown leaf Type", "Younger", "Middle\_Aged", and "Older". To the right of these labels is a vertical column of four empty text input boxes, with the label "Y/N" positioned above the top box. At the bottom of the window, there are two buttons: "Store Leaf Type in Data base" on the left and "Enter Temperature" on the right.

Fig 5.3:Leaf Type

The temperature details can be entered and stored in the system, by using the following screen.



The screenshot shows a window titled "Temperature" with a standard OS title bar. The main content area has a light gray background. At the top, it says "Enter 'Y' against Temperature at which disease is found". Below this, there are four labels: "Unknown Temperature", "Low", "Medium", and "High". To the right of these labels is a vertical column of four empty text input boxes, with the label "Y/N" positioned above the top box. At the bottom of the window, there are two buttons: "Store Temperatue in Data base" on the left and "Enter Moisture" on the right.

Fig 5.4:Temperature

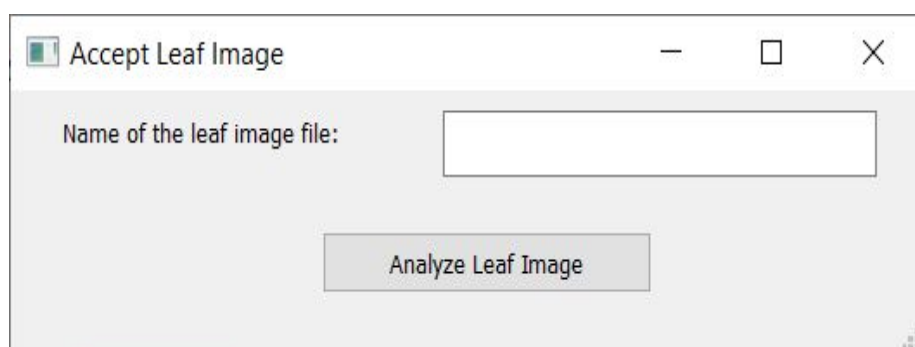
The following screen is used to store the moisture details in the Database.



A screenshot of a window titled "Moisture Details". The window has a dark title bar with standard OS controls. The main area contains the instruction "Enter 'Y' against Moisture at which disease is found". Below this, there are four labels: "Unknown Moisture", "Low", "Medium", and "High". To the right of these labels is a vertical column of four empty rectangular input boxes, with the label "Y/N" positioned above the top box. At the bottom of the window, there are two buttons: "Store Moisture in Data base" on the left and "Images Locations" on the right.

Fig 5.5:Moisture details window screenshot

The name and location of the leaf image file, can be provided as input to the system by using the following GUI screen.



A screenshot of a window titled "Accept Leaf Image". The window has a light gray background and standard OS controls in the title bar. It contains a label "Name of the leaf image file:" followed by a single-line text input field. Below the input field is a button labeled "Analyze Leaf Image".

Fig 5.6: Accept Leaf Image window screen shot

# CHAPTER 6

## TESTING

### 6.1 Software Testing

Software testing is the process of evaluating a software item to detect differences between given input and expected output. Testing assesses the quality of the product. Software testing is a process that should be done during the development process. In other words, software testing is a verification and validation process.

#### Verification

Verification is the process to make sure the product satisfies the conditions imposed at the start of the development phase. In other words, to make sure the product behaves the way it is meant to be.

#### Validation

Validation is the process to make sure the product satisfies the specified requirements at the end of the development phase. In other words, to make sure the product is built as per customer requirements.

#### Basics of software testing

There are two basics of software testing: Black box testing and white box testing.

#### Black box Testing

Black box testing is a testing technique that ignores the internal mechanism of the system and focuses on the output generated against any input and execution of the system. It is also called functional testing.

#### White box Testing

White box testing is a testing technique that takes into account the internal mechanism of a system. It is also called structural testing and glass box testing.

Black box testing is often used for validation and white box testing is often used for verification.

#### 6.1.1 Types of testing

There are many types of testing like

- Unit Testing
- Integration Testing

- Functional Testing
- System Testing
- Regression Testing etc.

### **Unit Testing**

Unit testing is the testing of an individual unit or group of related units. It falls under the class of white box testing. It is often done by the programmer to test that the unit he/she has implemented is producing expected output against given input.

### **Integration Testing**

Integration testing is testing in which a group of components are combined to produce output. Also, the interaction between software and hardware is tested in integration testing if software and hardware components have any relation. It may fall under both white box testing and black box testing.

### **Functional Testing**

Functional testing is the testing to ensure that the specified functionality required in the system requirements works. It falls under the class of black box testing.

### **System Testing**

System testing is the testing to ensure that by putting the software in different environments (e.g., Operating Systems) it still works. System testing is done with full system implementation and environment. It falls under the class of black box testing.

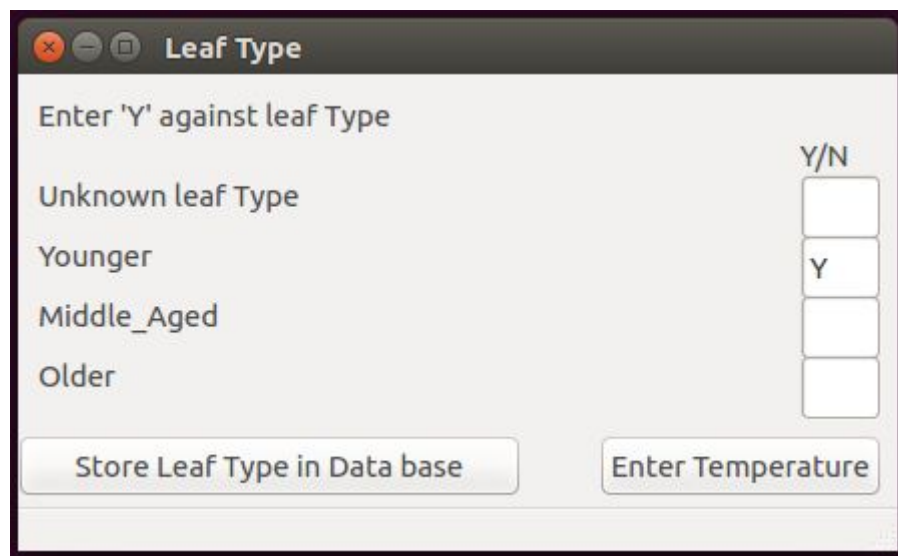
### **Regression Testing**

Regression testing is the testing after modification of a system, component, or a group of related units to ensure that the modification is working correctly and is not damaging or imposing other modules to produce unexpected results. It falls under the class of black box testing.

The project is thoroughly tested by testing the each and every text box and push buttons of the GUI Screens, and verifying the corresponding results in the Database.

### 6.1.2 Testing the model

Following is the leaf type screen along with Data.



Leaf Type

Enter 'Y' against leaf Type

Y/N

Unknown leaf Type ☐

Younger ☒ Y

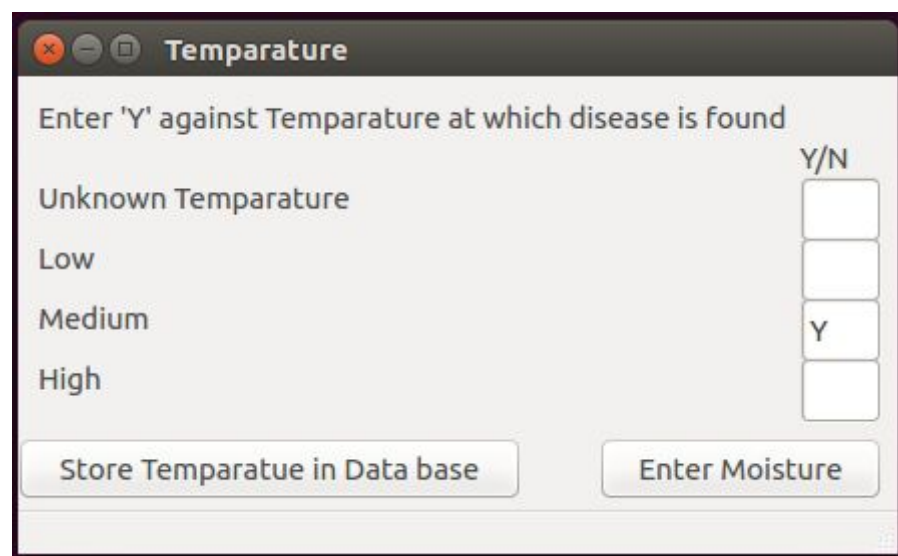
Middle\_Aged ☐

Older ☐

Store Leaf Type in Data base Enter Temperature

Fig 6.1:Leaf Type

Following is the temperature screen along with Data.



Temperature

Enter 'Y' against Temperature at which disease is found

Y/N

Unknown Temperature ☐

Low ☐

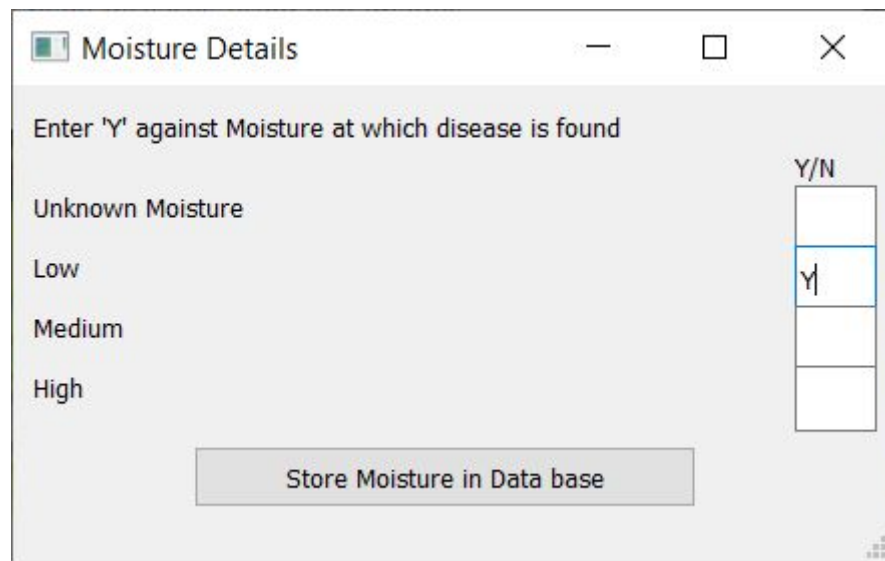
Medium ☒ Y

High ☐

Store Temperatue in Data base Enter Moisture

Fig 6.2:Temperature

Following is the Moisture type screen along with Data.



The image shows a software window titled "Moisture Details". Inside the window, there is a text prompt: "Enter 'Y' against Moisture at which disease is found". Below this prompt, there are four labels: "Unknown Moisture", "Low", "Medium", and "High". To the right of these labels is a vertical column of four input boxes, with the label "Y/N" positioned above them. The second input box, corresponding to "Low", contains the letter "Y". At the bottom center of the window is a button labeled "Store Moisture in Data base".

	Y/N
Unknown Moisture	
Low	Y
Medium	
High	

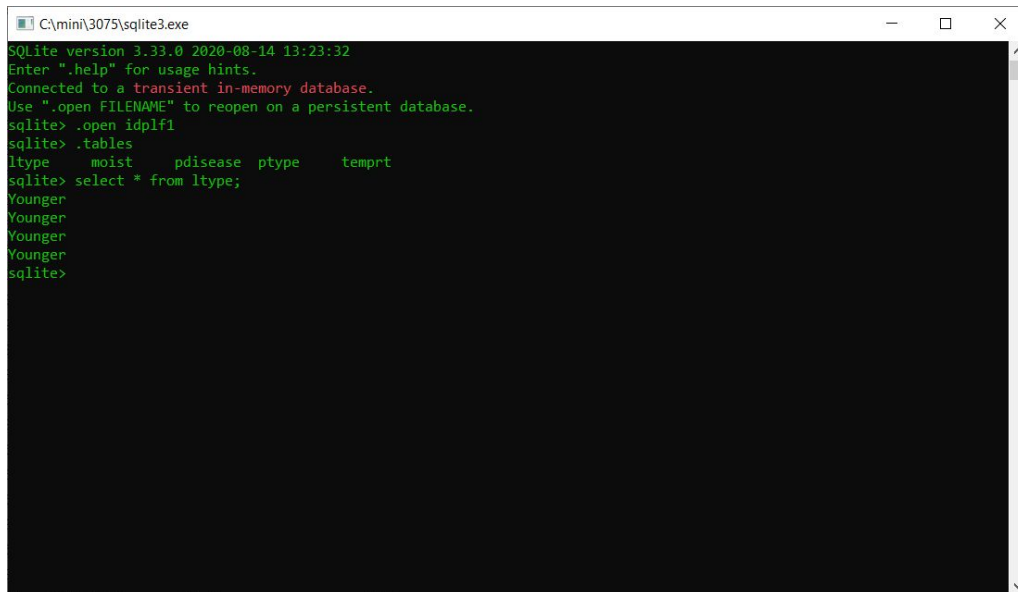
Store Moisture in Data base

Fig 6.3:Moisture Details

## CHAPTER 7

### RESULTS ANALYSIS

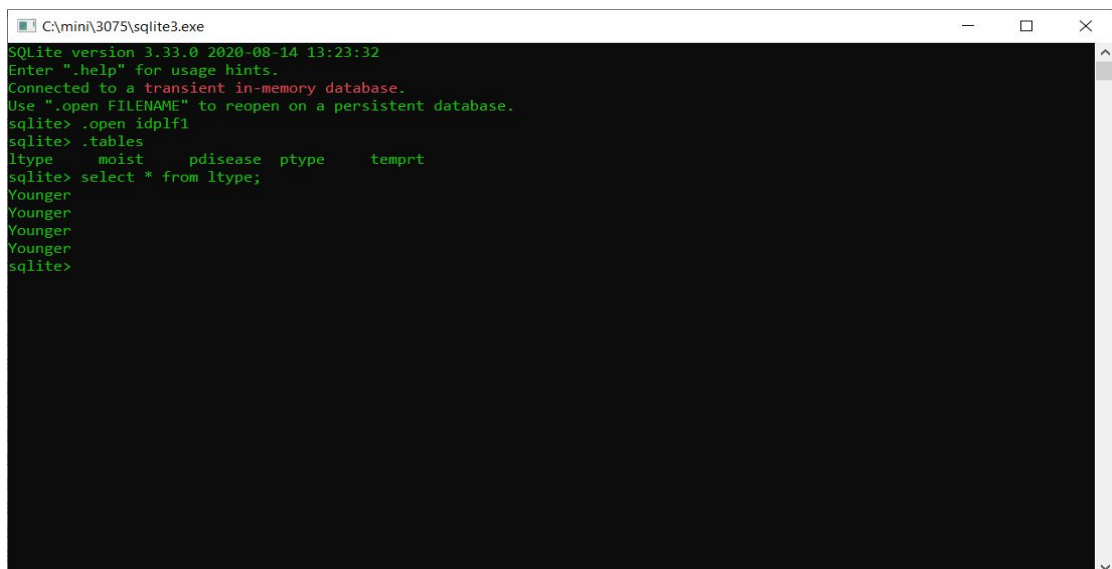
Following screenshot of the SQLite db confirms that the above data is stored in the Database.



```
C:\mini\3075\sqlite3.exe
SQLite version 3.33.0 2020-08-14 13:23:32
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> .open idplf1
sqlite> .tables
ltype      moist      pdisease  ptype      temprt
sqlite> select * from ltype;
Younger
Younger
Younger
Younger
sqlite>
```

Fig 7.1:screenshot of the SQLite db leaf type

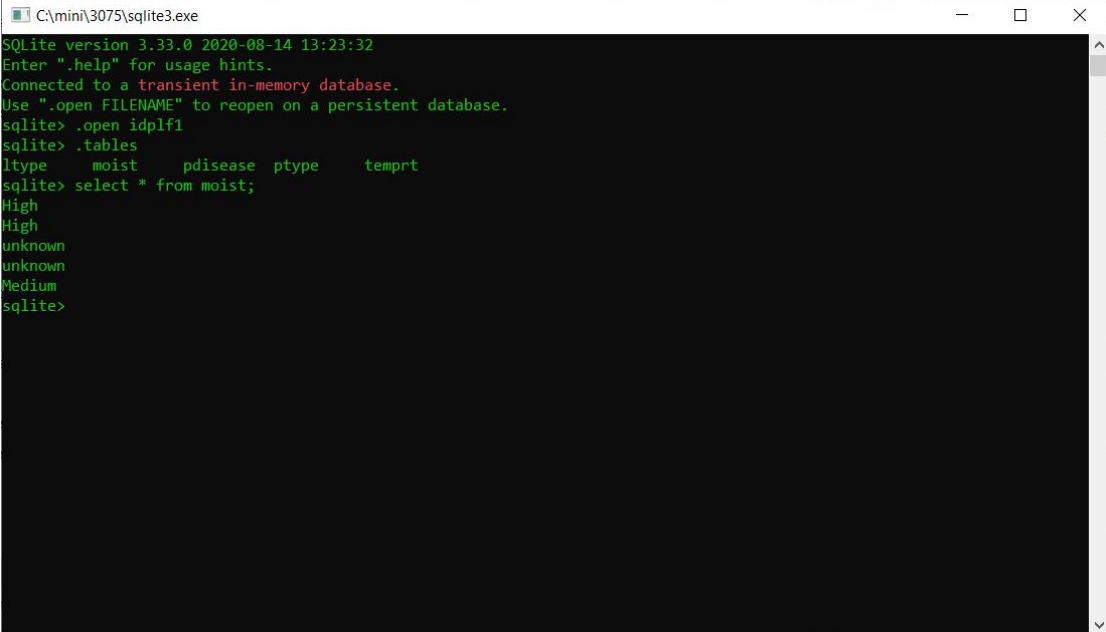
Following screen shot of the SQLite db confirms that the above data is stored in the Database.



```
C:\mini\3075\sqlite3.exe
SQLite version 3.33.0 2020-08-14 13:23:32
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> .open idplf1
sqlite> .tables
ltype      moist      pdisease  ptype      temprt
sqlite> select * from ltype;
Younger
Younger
Younger
Younger
sqlite>
```

Fig 7.2:screenshot of the SQLite db temperature

Following screen shot of the SQLite3 db confirms that the above data is stored in the Database.



```
C:\mini\3075\sqlite3.exe
SQLite version 3.33.0 2020-08-14 13:23:32
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> .open idplf1
sqlite> .tables
ltype      moist      pdisease  ptype      tempert
sqlite> select * from moist;
High
High
unknown
unknown
Medium
sqlite>
```

Fig 7.3:screenshot of the SQLite db moisture details

The leaf image is provided as input to the system, as shown in the following figure.

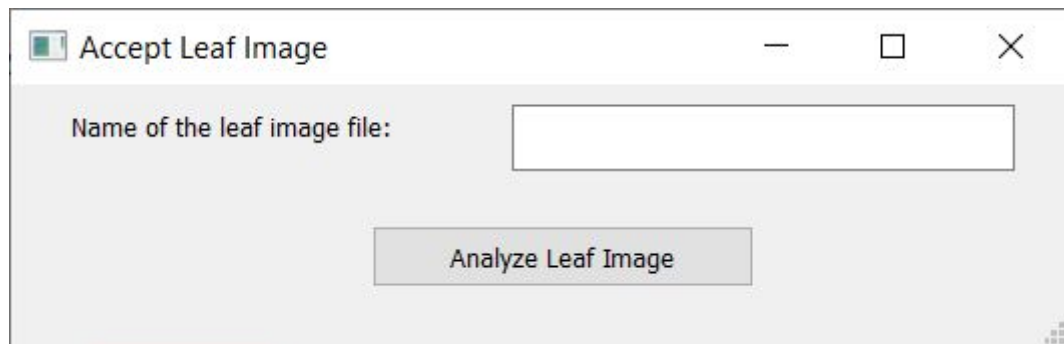


Fig 7.4:Accept Leaf image



The 'Analyze Leaf image' push button is used to call the python routine to analyze the image, and to pinpoint the disease affected area, as shown in the following figure.

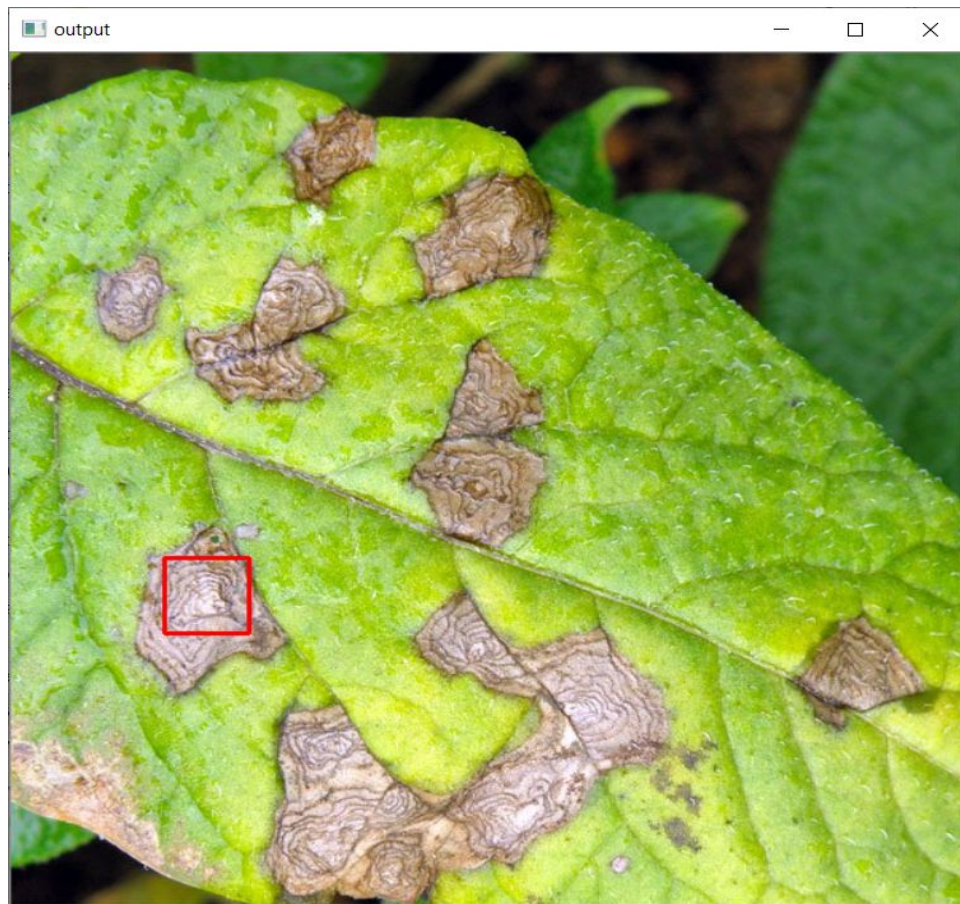


Fig 7.5: Final Output of Disease affected area

## **CHAPTER 8**

### **CONCLUSION**

#### **CONCLUSION**

This project entitled “Detection of Diseases in Blueberry Leaves using Machine Learning.” has presented an approach to identify the disease of a plant by processing the image of it’s affected leaf. This project is very useful to the farmers, as they need not go for the plant specialist consultation towards the identification of the disease. This project is also useful to the plant specialists, as they need not remember the details about each and every plant disease. This project finally leads to the enhancement of average plant life.

#### **FUTURE SCOPE**

This project can be enhanced further by implementing the measures to be taken for the disease affected plant.

## References

1. Base Paper : Cecilia Sullca, Carlos Molina, Carlos Rodríguez, and Thais Fernández: Diseases Detection in Blueberry Leaves using Computer Vision and Machine Learning Techniques, International Journal of Machine Learning and Computing  
<http://www.ijmlc.org/index.php?m=content&c=index&a=show&catid=98&id=990>.
2. N. R. Marroquin. (December 10, 2016). How to succeed in a blueberry project? gro Negocios Perú. [Online]. Available: <https://agronegociosperu.org/2016/12/10/como-tener-exito-en-unproyecto-de-arandanos/>
3. N. R. Marroquin. (November 20, 2015). Blueberries Peru. BlueberriesPeru.pe. [Online]. Available: <https://arandanosperu.pe/2015/11/20/principales-plagasyenfermedades-en-el-arandano-en-el-peru/>
4. R. Agrícola. (September 2017). Arándanos en Perú: Situación actual y perspectivas. RedAgrícola. [Online]. Available: <http://www.redagricola.com/cl/arandanos-en-peru-situacion-actual-y-perspectivas/>
5. G. Leiva, "Assessment of internal and external quality of blueberries using images," doctoral theses, Santiago of Chile: Pontifical Catholic University of Chile, 2013.
6. R. Pydipati, "Evaluation of classifiers for automatic disease detection in citrus leaves using machine vision," Thesis, University of Florida, 2014
7. <https://www.python.org/>
8. <https://github.com/baoboa/pyqt5/blob/master/pyuic/uic/pyuic.py>
9. <https://www.numpy.org/>
10. <https://riverbankcomputing.com/software/pyqt/intro>
11. <https://www.planetnatural.com/pest-problem-solver/plant-disease/>
12. Youtube Channel - Programming Knowledge
13. <https://stackoverflow.com/> - for clearing errors