



UrbanAgro : Application to Support Sri Lankan Urban Farmers to Detect and Control Common Diseases in Tomato Plants

W.L.V. Fernando 16000471

H.A.D.D. Navodi 16000943

Supervisor: Dr. Thilina Halloluwa

Co-Supervisor: Ms. Hiruni Kegalle

March 2021

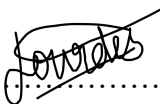
Submitted in partial fulfillment of the requirements of the
B.Sc(Hons) in Software Engineering 4th Year Project (SCS4123)



Declaration

I certify that this dissertation does not incorporate, without acknowledgment, any material previously submitted for a degree or diploma in any university and to the best of my knowledge and belief, it does not contain any material previously published or written by another person or myself except where due reference is made in the text. I also hereby give consent for my dissertation, if accepted, be made available for photocopying and for interlibrary loans, and for the title and abstract to be made available to outside organizations.

Candidate Name: Ms. W.L.V. Fernando

.....


Signature of Candidate

Date: 26.03.2021

Candidate Name: Ms. H.A.D.D. Navodi

.....


Signature of Candidate

Date: 26.03.2021

This is to certify that this dissertation is based on the work of Ms. W.L.V. Fernando and Ms. H.A.D.D. Navodi under my supervision. The dissertation has been prepared according to the format stipulated and is of the acceptable standard.

Supervisor's Name: Dr. Thilina Halloluwa

.....

Signature of Supervisor

Date:

Co-Supervisor's Name: Ms. Hiruni Kegalle

.....

Signature of Co-Supervisor

Date:

Abstract

Plant diseases cause many significant damages and losses in crops around the world. Some appropriate measures should be introduced on identification of plant diseases to prevent damages and minimize losses. With Covid-19 lockdowns many Urban dwellers are encouraged to grow their own foods. As most urban farmers do not tend to use pesticides in their farms there is a high chance for the crops to get caught of various diseases. Comparatively, identifying the plant diseases visually is expensive, difficult and inefficient. And also getting expertise knowledge is very expensive and practically impossible to reach them whenever they need. As such this might be a difficult task for urban farmers or newcomers to this field to decide which disease can be attached to the crops. Early detection of diseases helps in increasing the productivity of crops as well as in minimizing expenses. Technical approaches using machine learning and computer vision are actively researched to achieve intelligence farming by early detection on plant diseases. The accuracy of object detection and recognition systems has been drastically improved by the recent development in Deep Neural Networks. The system proposed presents a practical, applicable solution for the identification of the type and location of 5 different types of diseased and healthy leaves of tomato plant, which is a significant difference from the conventional methods for plant disease classification. In this context we have used YOLOv3 model which is a method based on transfer learning to diagnose tomato plant diseases using images taken in-place by camera devices on smartphones instead of using the procedure to collect, test and analyze physical samples (leaves, plants) in the laboratory. The trained model achieved an average accuracy of 92 percent, which is exceptional in comparison to previous studies in this context. The target group of users are urban farmers who request a quick diagnosis on common tomato leaf diseases at any time of the day as they lack knowledge on diseases that are attached with plants.

Acknowledgements

We take this opportunity to gratefully acknowledge the assistance and contribution supervisors and advisors who have been helping throughout the project.

First and foremost we would like to express gratitude to our Supervisor Dr. Thilina Halloluwa and our Co-supervisor Miss. Hiruni Kegalle for their guidance, encouragement and immense knowledge to do a better quality project. They were always there whenever we needed their help and ideas. We are really thankful for them who made our concepts clearer.

At last we would like to acknowledge all the assistance and contributions of the University of Colombo School of Computing for supporting us with all that is needed starting from the knowledge, and ending with the full care that it is provided with, to help us to be professionals in the field of Software Engineering.

Table of Contents

Declaration	i
Abstract	ii
Acknowledgements	iii
Table of Contents	iv
List of Figures	vii
List of Tables.....	ix
List of Acronyms.....	x
Chapter 1 - Introduction	1
1.1 Introduction and Motivation.....	1
1.2 Problem Statement	2
1.3 Aim.....	3
1.4 Objectives	3
1.5 Scope of the Project	3
1.6 Novelty.....	3
1.7 Limitations	4
1.8 Justification for a Product Based Project	4
1.9 Outline of the Dissertation	5
Chapter 2 - Literature Review.....	6
2.1 Pre-processing.....	6
2.2. Feature Extraction	6
2.3 Machine Learning	7
2.4 Deep Learning	8
2.4.1 Classification.....	9

2.4.2 Object Detection	9
Chapter 3 - Analysis and Design.....	14
3.1 System Overview	14
3.2 Use Case Diagram.....	14
3.3 Use Case Narratives	15
3.4 Activity Diagram	19
3.5 Architectural Pattern	20
3.6 Quality Attributes.....	20
Chapter 4 - Implementation	22
4.1 Methodology	22
4.2 Data Collection	22
4.3 Image Annotation.....	24
4.4 Data Augmentation	25
4.5 Deep Learning Model	25
4.5.1 YOLOv3 Design	26
4.5.2 Justification for Choosing a Object Detection Model Over a Classification Model	28
4.6 Model Training	28
4.7 Graphical User Interfaces (GUI).....	30
4.8 The Use of Design Patterns.....	35
4.9 Justification for Tools and Technologies	36
Chapter 5 - Results and Analysis.....	39
Chapter 6 - Evaluation and Testing.....	47
6.1 Evaluation of the Model.....	47
6.1.1 Model Results	51
6.1.2 Comparison of Our Results with Previous Works	52

6.2 Testing Plan	53
6.2.1 Test Items.....	53
6.2.2 Testing Approaches	53
6.3 Test Results	55
6.3.1 Automation Testing	55
6.3.2 Manual Testing	58
Chapter 7 - Conclusion.....	62
7.1 Shortcomings of the Study.....	62
7.2 Future Directions	63
Peer Evaluation.....	64
References	65

List of Figures

Figure 1.1: Common diseases in tomato plants	2
Figure 3.1: System overview	14
Figure 3.2: Use case diagram	14
Figure 3.3: Activity diagram	19
Figure 4.1: Methodology.....	22
Figure 4.2: Images of classes from PlantVillage dataset.....	23
Figure 4.3: Images of classes from PlantDoc dataset	24
Figure 4.4: Image annotation using Labellmg tool	24
Figure 4.5: Comparison with other object detectors [29]	26
Figure 4.6: YOLOv3 network architecture [30]	27
Figure 4.7: Launch screen	30
Figure 4.8: Interface of Home screen	31
Figure 4.9: Interface of Upload Image screen	32
Figure 4.10: Interface of Result screen: displaying the processed image and control instruction	33
Figure 4.11: Interface of zoomed in image and results showing for Late Blight disease	34
Figure 4.12: Interface of a Result screen showing results as healthy	35
Figure 5.1: Training Loss observed in Experiment 1	40
Figure 5.2: Validation Loss observed in Experiment 1.....	41
Figure 5.3: Training Loss observed in Experiment 2	42
Figure 5.4: Validation Loss observed in Experiment 2.....	43
Figure 5.5: Prediction of disease in Experiment 1	44
Figure 5.6: Prediction of disease in Experiment 1	44
Figure 5.7: Prediction of disease in Experiment 2	45
Figure 5.8: Prediction of disease in Experiment 2	45
Figure 6.1: Training loss	48

Figure 6.2: Validation loss	49
Figure 6.3: Prediction of the disease	50
Figure 6.4: Prediction of the disease	50
Figure 6.5: Prediction of the disease	51
Figure 6.6: Sample of unit testing results of python application	55
Figure 6.7: Sample of unit testing results of Flutter application	56
Figure 6.8: Sample of unit testing results of Flutter application	56
Figure 6.9: Sample of widget testing code of Flutter application	56
Figure 6.10: Sample of widget testing results of Flutter application	57
Figure 6.11: Sample of widget testing results of Flutter application	57
Figure 6.12: Sample of widget testing code of Flutter application	57
Figure 6.13: Sample of widget testing results of Flutter application	58
Figure 6.14: Sample of widget testing results of Flutter application	58

List of Tables

Table 2.1: Summary of some studies in plant disease classification	11
Table 3.1: Use case narrative for upload a photo.....	15
Table 3.2: Use case narrative for detect disease	16
Table 3.3: Use case narrative for suggest remedies	17
Table 3.4: Use case narrative for send results.....	17
Table 4.1: Summary of PlantVillage dataset	23
Table 4.2: Summary of PlantDoc dataset	23
Table 4.3: Parameter configuration during the training.....	29
Table 5.1: Results obtained from experiments.....	43
Table 6.1: Average Precisions (AP) on the test set for each class and mAP	51
Table 6.2: Comparison with previous work	52
Table 6.3: Manual testing performed	58

List of Acronyms

API	-	Application Programming Interface
CNN	-	Convolutional Neural Networks
DL	-	Deep Learning
GIoU	-	Generalized Intersection over Union
IoU	-	Intersection over Union
JSON	-	Javascript Object Notation
mAP	-	Mean Average Precision
R-CNN	-	Regions with Convolutional Neural Networks
R-FCN	-	Region-based Fully Convolutional Network
SSD	-	Single Shot Detector
SVM	-	Support Vector Machine
YOLO	-	You Only Looks Once

Chapter 1 - Introduction

1.1 Introduction and Motivation

COVID-19 lockdowns are forcing people to rethink their lifestyles, particularly in terms of food as they have seen how panic buying left many supermarket shelves empty. As a result, many urban dwellers have transformed their back yards or rooftops of their apartments into “Urban Farms” [1].

Tomato is a plant that is enjoyed both as a fruit and a vegetable worldwide. The temperature that is required for tomato cultivation varies from Centigrade 20 to 27 which is the temperature that exists in most areas of Sri Lanka throughout the year [2]. As such, since 2013, Sri Lanka has enjoyed a healthy production of approximately 80000 tons of tomato on average annually [3]. Further, it takes only 40 to 50 days after planting to produce fruits/vegetables [4]. Also, it is a plant which can be grown in both backyard pots as well as the highly commercial fields. Due to these reasons there is a tendency for people in urban areas to grow tomatoes in their “Urban Farms”.

When it comes to tomato farming, tomato crop diseases can not be ignored. They are attacked by many types of fungus, bacteria and virus. According to the experts, the most common diseases found in tomato farming are Late blight, Leaf Mold, Bacterial spot and Yellow leaf curl virus (Figure 1.1) [5]. These diseases need different types of control measures. A brief description on those diseases are given below;

Leaf mold: The initial signs are light green or yellowish points on the top surface of the leaf that eventually enlarge to become yellow. [6].

Bacterial spot: Small angular to irregular, water-soaked spots on the leaves are the symptoms. [7].

Late blight: Late blight appears first as water-soaked, gray-green spots on the lower, older leaves. These leaf spots will rapidly enlarge, and a white mold will appear at the affected area's margins on the lower surface of the leaves. [8].

Yellow leaf curl virus: Plants that affected this disease exhibit upward and inward rolling of the leaf margins and yellowing of leaflets. This is a destructive disease which causes severe loss in productivity [9].



Leaf Mold



Bacterial Spot



Late Blight



Yellow Leaf Curl
Virus

Figure 1.1: Common diseases in tomato plants

1.2 Problem Statement

For urban farmers disease detection and taking necessary control measures play a crucial role in tomato farming. As most urban farmers do not tend to use pesticides in their farms there is a high chance for the crops to get caught of various diseases. As such this might be a difficult task for urban farmers or newcomers to this field to decide which disease can be attached to the crops.

The symptoms of tomato plant diseases are conspicuous in different parts of a plant such as leaves, stems, fruits [10]. But the manual detection of a tomato plant disease using leaf images is a challenging task. Hence an automatic disease detection technique can be advantageous specially for newcomers to this field in order to detect a plant disease and to take preventive measures and control plant diseases at a very initial stage.

The detection and recognition of plant diseases based on deep learning provide hints for early stage detection of diseases. Comparatively, identifying the plant diseases visually is expensive, difficult and inefficient. Also it requires a higher level of expertise of trained agricultural experts and botanists.

1.3 Aim

The aim of our project is to develop a practical, reliable and inexpensive real time application to support urban farmers by detecting tomato plant diseases and guiding them to take necessary control measures to enhance the productivity of the tomato plants.

1.4 Objectives

- Reviewing existing similar solutions on detecting plant diseases.
- Exploring real time technologies that are available to detect plant diseases.
- Find an effective and accurate approach to detect diseases on tomato plants.
- Find a simple and user friendly way to reach the urban farmers through an application which can detect diseases on tomato plants.

1.5 Scope of the Project

- Classification & localization model that is designed to detect 5 different types of diseased and healthy leaves of tomato plant.
- Identification of the control measures and mapping them with the respective disease.
- Implementation of the mobile application to communicate the identified disease and the control measures to the farmer effectively.

1.6 Novelty

This project marks its novelty by successfully detecting 5 different types of diseased and healthy leaves of tomato plant in heterogeneous background using one of the best object detection models recently introduced which gives more accurate results in real time than the previous works carried out in this field.

In addition to that our application provides following facilities when compared to most of the traditional approaches used in the context of plant disease detection.

- Our application uses images of tomato plant diseases taken on site rather than collecting and analyzing samples in the laboratory.
- The possibility is considered that more than one disease can be affected by a single plant at the same time.
- It offers a convenient real-time application that can be used in the field without the use of costly and complicated technologies.
- Our approach uses input images captured by various mobile phone camera devices with different resolutions.
- With differing sizes of objects and changes of background contained in the plant environment, the system results are not impacted negatively.
- It offers the end-user the remedies for the tomato plant diseases detected.

1.7 Limitations

- Detects only 5 different types of diseased and healthy leaves of tomato plants.
- The control measures are given considering only the type of disease in the plant such that the control measures are not given accordingly to the severity of the disease.
- Identification of diseases affecting the parts other than leaves in tomato plants is out of scope.

1.8 Justification for a Product Based Project

Software engineering projects focus on the development of software products with the main intention of doing an innovation rather than an invention. According to the facts described in the previous section this project consists of an innovative research component of successfully detecting 5 different types of diseased and healthy leaves of tomato plant in real time with a mean average accuracy of 92 percent. The goal is building a mobile application that could be used by the urban tomato farmers to detect

the diseases in their farms neither having to place expensive devices in their farms nor having to reach expertise consultancy.

Having considered the overall characteristics and capabilities of the members and the time constraints, an iterative and incremental method was adopted as the software engineering process model. The application was divided into components and these components were completed and improved in subsequent iterations gradually, before they were combined into a system. Rather than delivering the whole project at once, the application is developed with the feedback of the supervisors to do necessary changes on the application on time.

We considered the best practices in software engineering such as adapting quality assurance principles and design patterns which will ensure the quality of the system and enhance the system's maintainability, flexibility and adaptability. Version controlling and code reviews (peer reviews) was carried throughout the project to maintain best coding structures and increase the quality of coding.

From the above facts it points to the conclusion that this is a software engineering project and to be more specific; this is a product-based software engineering project.

1.9 Outline of the Dissertation

The dissertation contains six main chapters each dedicated to an important aspect of the system. Chapter 1 provides an overview of the application from problem definition to the solution identified. Chapter 2 discusses the related work carried out regarding plant disease detection. Chapter 3 gives a detailed description of problem analysis and system design along with UML diagrams. Chapter 4 describes the approach used to solve the problem identified and the tools and technologies used in the implementation process. Chapter 5 depicts two approaches adapted to train the model and compares the findings of those approaches. Chapter 6 explains how the testing was carried out and how the system was evaluated. Chapter 7 provides the conclusion as the final chapter of the body of the dissertation.

Chapter 2 - Literature Review

2.1 Pre-processing

Various preprocessing techniques are applied to the images to process raw images and then to get useful images for further processing and analysis. For instance, Vetal et al [11], uses Kurtosis and skewness filters to smoothen images. Then performed the image segmentation using the inverse difference method to part disease affected area from the leaf. After this stage, two images were available, one with only diseases and one with disease extracted images.

In order to enhance the quality of the images and upgrade the feature extraction phase, Mokhtar et al [12] applied leaf image isolation, image resizing, and background removing as the preprocessing techniques. They manually cropped the image to separate every leaf in the image and since the input images are in different sizes, they resized the images to 512x512 resolution to make them identical in size so that the storage capacity is utilized and the computational time is pulled down in later phases. Although the leaf image is isolated and extracted at an early phase still there can be small parts and shadows remaining disturbing the feature extraction phase. In order to overcome that issue, the background of each image was removed using the background subtraction technique with some morphological operations.

Sabrol et al [13], resized and standardized the images to a fixed size of 256×256 . Then recognized the tomato plant diseases by classifying them using a decision tree having applied Otsu's segmentation to convert intensity-based feature extraction.

2.2. Feature Extraction

Feature extraction is performed on images to extract features for classification. For instance, Vetal et al [11], considered the color and texture of the segmented disease-only images to extract the unique features from the image. Color features were extracted by converting the RGB image of a leaf into HIS color space. Texture features that were

extracted are Energy, Entropy, correlation, and homogeneity. A minimum of 80 sample images per disease was chosen to extract features.

Measuring certain features or properties like texture and color in order to reduce the original dataset was described as the motive of the feature extraction phase by Mokhtar et al [12]. They extracted the textural pattern of tomato leaves using Gabor wavelet transformation and extracted 402 such texture-based features and represented them in a database as vector values.

The digital image represents using digital information that contains the color intensity values of each pixel. The perception of a color is a combination of three primary colors red, green, and blue. In the study of Sabrol et al [13], the pattern of the disease symptoms are recognized by the intensity of each disease infected tomato plant image. A total of ten intensity-based statistical features were computed for each plant disease. Then ten color descriptors were computed for three colors. Finally, the extracted features were submitted to three different classifiers.

2.3 Machine Learning

In the context of plant disease detection traditional machine learning approaches such as SVM and decision trees are more applicable in the identification of plant images in uniform-background where they are captured in an ideal laboratory environment. The reason behind this is in diagnosing plant diseases using traditional machine learning approaches efficiency is reduced because images should undergo some complex image preprocessing and feature extraction steps.

After the phases of pre-processing and feature extraction, machine learning approaches are used to determine which disease is present in the leaf. For instance, Vetal et al [11], used a multi-class SVM algorithm to identify four key diseases in tomato plants namely Early Blight, Septoria Leaf Spot, Bacterial Spot, and Iron Chlorosis using 320 images. The training dataset was prepared by extracting the co-occurrence features for the leaves with analogous feature values. The result reported better classification accuracies for all the four diseases and the percentage accuracy is 93.75%.

Mokhtar et al [12] used SVM to classify 2 tomato plant diseases, Powdery Mildew and Early Blight using 200 images. It was trained and tested using different kernel functions such as Cauchy kernel, Invmult Kernel and Laplacian Kernel comparing different results yielded. They achieved 99.5% of accuracy.

Sabrol et al [13], proposed a solution to detect 5 diseases of tomato plants using a decision tree using 670 images of disease infected areas of the plant and achieved 78% accuracy. They have proposed to combine some statistical and geometric features with more other classifiers. The decision tree based classifier follows the criteria of if-then rules. They had considered five types of tomato plant diseases and healthy images for the experiment. The proposed algorithm computed ten statistical normalized features and submitted to a classifier based decision tree. The recognition accuracy of bacterial canker: 84.6%, bacterial leaf spot:69.2%, late blight: 80.7%, septoria leaf spot: 92.3%, leaf curl:70% and healthy recognized with 70%. The overall recognition accuracy yielded 78% and that was quite satisfactory.

2.4 Deep Learning

The main incentive using deep learning for computer vision is without having to use hand-crafted features it can exploit the image directly. This is because CNNs can extract features automatically and directly with no need of complex preprocessing on images. But it requires a high machine configuration, large amounts of data and relies on large scale datasets. To solve that problem transfer learning comes to play. Transfer learning enables us to use models trained on one computer vision task with a large number of labelled images to use in another task. It removes the need for a huge dataset and a lot of computational power to train a model from scratch. In comparison to models trained from scratch, transfer learning experiments are much faster and more accurate[14][15].

2.4.1 Classification

Image classification involves assigning a class label to an image. This decides, using the classification model, which disease is present in the leaf. This model should be trained with labeled data using learning algorithms.

Mohanty et al. [14] used AlexNet and GoogLeNet to detect 26 diseases from 14 crop species. They used 54206 images of diseased and healthy plant leaves to train the model. They achieved an overall accuracy of 99.35% on the held-out test set.

Brahimi et al. [15] did a comparison between AlexNet and GoogLeNet architectures for tomato plant diseases by using 14828 images of healthy and diseased leaves from Plant Village dataset in which GoogLeNet performed better than AlexNet. They achieved 98.66% accuracy with Alexnet and achieved 99.17% accuracy on GoogLeNet. They had achieved this accuracy by getting a portion of the same dataset they used for training as the test set.

The 3 deep learning models Densnet161, DensNet121 and VGG16 with transfer learning were taken into consideration by Ouhami et al. [16] to find the machine learning model for tomato-growing disorders in standard RGB images. The study was based on images of infected leaves of plants divided into six kinds of infections, pest attacks and plant conditions. They used a 666-image dataset and divided the data set into 80% for training and 20% for evaluation. They achieved accuracy of 95.65% for DensNet161, 94.93% for DensNet121 and 90.58% for VGG16.

2.4.2 Object Detection

Unlike image classification, in object detection it is required to detect and locate certain multiple objects from the image. Few deep learning algorithms were developed for the purposes of object detection which are primarily divided into two categories. One approach is for the algorithm to generate a series of candidate frames as samples, and then classify the samples using CNN, such as RCNN [17], which was one of the first algorithms in the context of object detection through CNN, faster RCNN [18], and R-

FCN [19]. The other transforms the problem of object bounding box location directly into a regression problem, which does not require the generation of candidate boxes. SSD [20] and YOLO [21] are two examples for such algorithms. There have been very few studies that have used object detection algorithms in the context of plant disease recognition.

Fuentes et al. [22] did a comparison between three network structures, Faster R-CNN, SSD and R-FCN with different feature extractors, AlexNet, ZFNet, GoogLeNet, VGG-16, ResNet-50, ResNet-101 and ResNetXt-101 for the recognition of tomato plant pests and diseases. Dataset of 2823 images that contain 9 classes of pests and diseases were used in this work. Data augmentation was applied due to the small number of images in the dataset. Dataset was divided into 80% for the training set, 10% for the validation set and 10% for the testing set. Study states that some classes with high variation in patterns are often confused with others due to lack of sample numbers, leading to false positives or low average precision. It states that R-FCN with the ResNet-50 feature extractor gave best results with a mean AP of 85.98%.

Cynthia et al. [23] presents a method that detects 5 different diseases from plant leaf images (Blast of Rice, Sigatoka Leaf Spot of banana, Black Spot of Rose, White Rust of Mustard, Grey Spot of Mustard) using Tensorflow, and the model was trained using a faster R-CNN method. They used a dataset containing only 236 images. The difference between R-CNN and faster R-CNN is that faster R-CNN does not require selective search and allows the network to generate ideas for regional proposals. However, this algorithm does not use selective search, and another network is used to predict region proposals. They divided the entire dataset into two parts: 80 percent of the total images were taken for training samples, and 20 percent of the total images were taken for test samples. From that, they achieved the accuracy value of 67.34%.

Jiang et coll. [24] suggested an enhanced deep learning method of CNN-based to detect Apple Leaf and Insect Pests in real time. Primarily, an apple leaf disease data set was composed using laboratory images and complex images in real-world conditions by means of data expansion and images annotation technology. A new method has therefore been suggested by introducing GoogleNet inception structure and rainbow

concatenation. Finally, five common insect pests and apple leaf disease images were trained in the proposed INAR-SSD (SSD with a perception module and rainbow condition). The model obtained 78.80% mAP according to experimental results.

The summary of some studies in plant disease classification is shown in Table 2.1. These studies show some principal issues. Such as a small number of images in datasets, accuracy results which are relatively low and to fail to detect diseases from the images taken under conditions different from the images used for training.

Table 2.1: Summary of some studies in plant disease classification

Study	Approach	Diseases	Dataset	Accuracy	Drawback
Vetal et al [11]	Multi-class SVM	Early blight, Septoria leaf spot, Bacterial spot, Iron chlorosis	320 images	93.75%	Images in the dataset were very less
Mokhtar et al [12]	SVM	Powdery mildew, Early Blight	200 images	99.5%	Although the accuracy is 99.5%, the used dataset contains only 200 images that belong to 2 classes.
Sabrol et al [13]	Decision Tree	Bacterial canker, Bacterial leaf spot, Late blight, Septoria leaf spot, Leaf curl	670 images	78%	Accuracy yielded for Bacterial spot and Leaf Curl was not satisfactory.
Mohanty et al [14]	AlexNet GoogLeNet	26 diseases from 14 crop species	54206 images	99.35%	Accuracy fell to 31.4% when tested on another verified dataset of 121 images collected

					from real life scenario.
Brahimi et al [15]	AlexNet GoogLeNet	Early blight, Tomato leaf curl virus, Septoria spot, Tomato Mosaic Virus, Target spot, Bacterial spot, spider mites, leaf mold, Late blight	14828 images	98.66% 99.17%	When tested on a set of images taken under conditions different from the images used for training, the model's accuracy reduced substantially.
Ouhami et al [16]	DensNet161 DensNet121 VGG16	Early blight, Late blight, Powdery mildew, Leaf miner flies, Thrips, Tuta absoluta	666 images	95.65% 94.93% 90.58%	Images in the dataset were very less
Fuentes et al [22]	Faster R-CNN SSD R-FCN	Leaf mold, Gray mold, Canker, plague, Miner, Low temperature, powdery mildew, Whitefly, Nutritional excess	2823 images	85.98%	Due to the lacking number of images some classes with high pattern variation tend to be confused with others which resulted in lower average precision.
Cynthia et al [23]	Faster R-CNN	Blast of Rice, Sigatoka Leaf Spot of banana, Black Spot of Rose, White Rust of Mustard, Grey Spot of Mustard	236 images	67.34%	Overall accuracy obtained was relatively low. Images in the dataset were very less.

Jiang et al [24]	INAR-SSD (SSD with perception module and rainbow condition) model	Alternaria leaf spot, Brown spot, Mosaic, Grey spot and Rust (Apple leaf diseases)	26,377 images	78.80%	Accuracy obtained was relatively low.
------------------	---	--	---------------	--------	---------------------------------------

Chapter 3 - Analysis and Design

3.1 System Overview

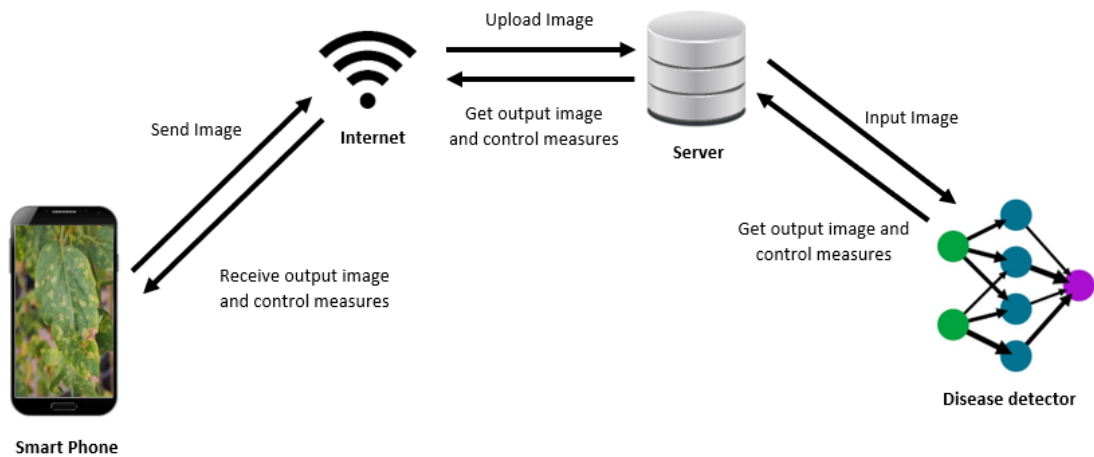


Figure 3.1: System overview

3.2 Use Case Diagram

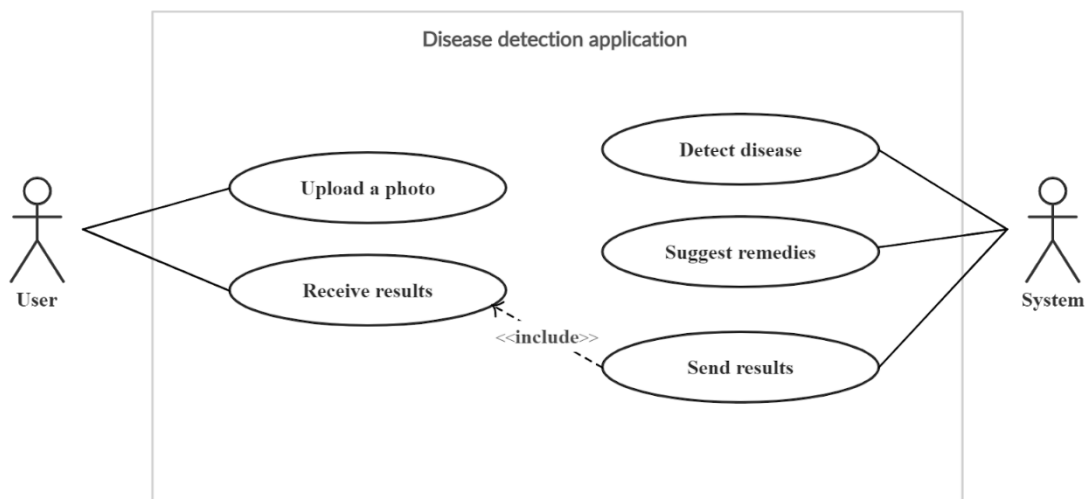


Figure 3.2: Use case diagram

3.3 Use Case Narratives

Use case 01: Upload a photo

Table 3.1: Use case narrative for upload a photo

Use Case ID	01
Use Case Name	Upload a photo
Description	Upload a photo of leaves of the tomato plant to detect whether the leaves are healthy or not. If the leaves have disease then to know which disease is attached to it.
Primary Actors	User
Secondary Actors	System
Precondition	None
Trigger	This use case is triggered when the user want to upload a photo
Scenario	<ol style="list-style-type: none">1. Launch the application.2. Select the upload method “From Camera” or “From Gallery”<ol style="list-style-type: none">2.1. User select the “From Camera”<ol style="list-style-type: none">2.1.1. System checks for the permission to access camera<ol style="list-style-type: none">2.1.1.1. If the camera is launched from this application for first time then the application ask for permission2.1.1.2. If the user hasn't given the permission previous time then the application asks for permission again.2.1.2. If the permission was given then the camera is launched and then the user can capture the photo. Else the user is returned back to home screen.

	<p>2.2. User select the “From Gallery”</p> <p>2.2.1. System checks for the permission to access the gallery.</p> <p>2.2.1.1. If the gallery is launched from this application for first time then the application ask for permission</p> <p>2.2.1.2. If the user hasn't given the permission previous time then the application asks for permission again.</p> <p>2.2.2. If the permission was given then the gallery is launched and then the user can select the photo. Else the user is returned back to home screen.</p>
Post Condition	Photo is uploaded to the application and begins the detection process

Use case 02: Detect disease

Table 3.2: Use case narrative for detect disease

Use Case ID	02
Use Case Name	Detect disease
Description	Detect leaves with diseases and output bounding boxes with the class label attached to each bounding box
Primary Actors	System
Secondary Actors	None
Precondition	Image should be uploaded to the application
Trigger	This use case is triggered when the image is uploaded
Scenario	<ol style="list-style-type: none"> 1. Input uploaded image to the model 2. Output the bounding boxes with the disease name attached to it as a label for each diseased leaf

Post Condition	<p>If diseases are found, then those diseases should be sent to suggest remedies else start the processing of results.</p> <p>Details of bounding boxes with class names should be sent to process the results.</p>
----------------	---

Use case 03: Suggest remedies

Table 3.3: Use case narrative for suggest remedies

Use Case ID	03
Use Case Name	Suggest remedies
Description	Suggest remedies for the detected disease
Primary Actors	System
Secondary Actors	None
Precondition	Diseases should be found in the uploaded image
Trigger	This use case is triggered when the diseases are found in the uploaded image
Scenario	<ol style="list-style-type: none"> 1. Input the diseases that are found in the uploaded image 2. Output remedies for the disease.
Post Condition	Start processing the result

Use case 04: Send results

Table 3.4: Use case narrative for send results

Use Case ID	04
Use Case Name	Send results

Description	Send the results to display to the user
Primary Actors	System
Secondary Actors	User
Precondition	Results from the detect disease and suggest remedies use cases.
Trigger	This use case is triggered when there are no diseases found or after suggesting the remedies
Scenario	<ol style="list-style-type: none"> 1. Send the image with bound boxes around the each diseased leaves with their disease name and the remedies for the disease
Post Condition	Results should be displayed to user

3.4 Activity Diagram

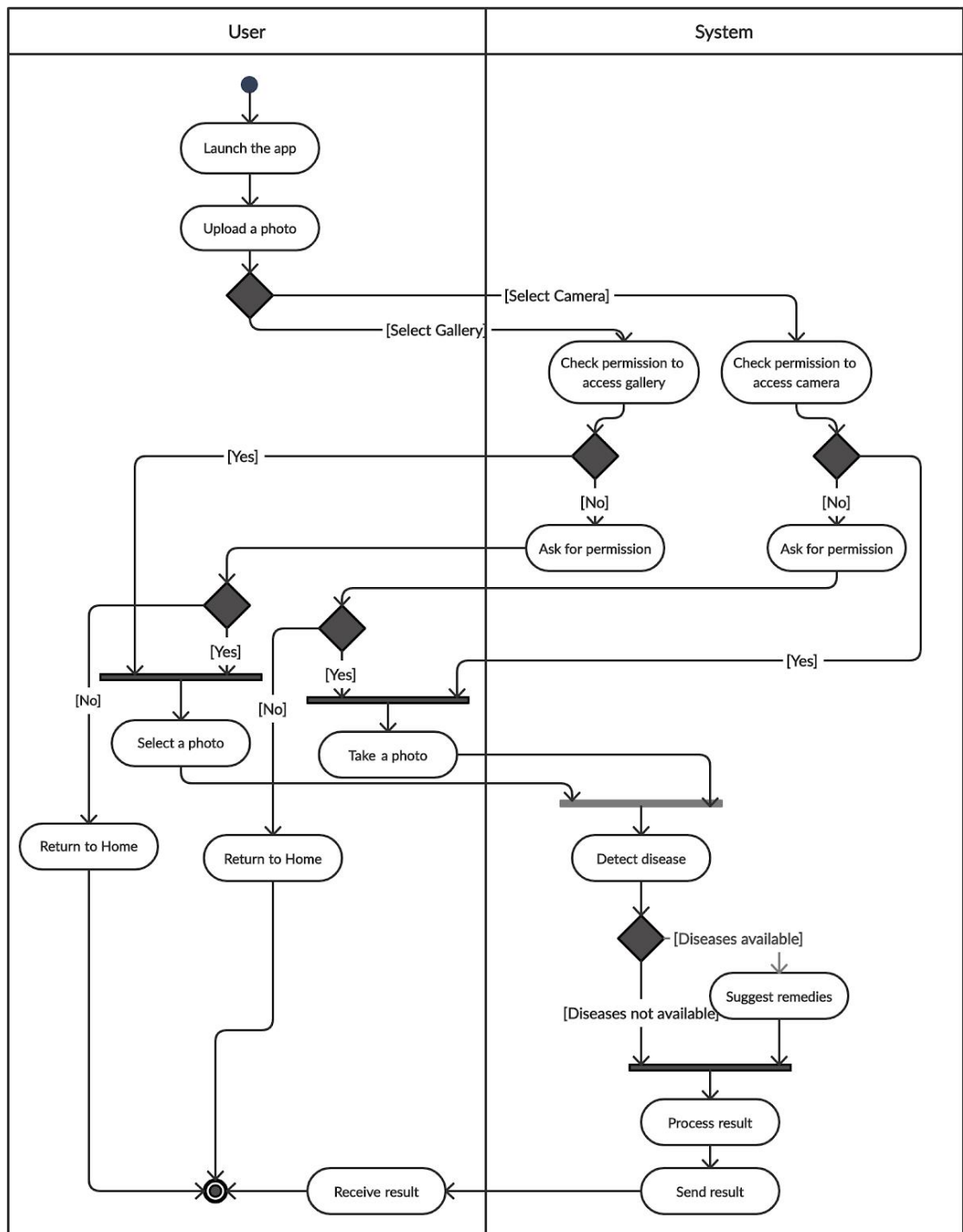


Figure 3.3: Activity diagram

3.5 Architectural Pattern

MVVM (Model View ViewModel) was used as the architectural pattern which builds a view model that can represent the data through a view.

Model in the MVVM is used to fetch the data from the python backend where the input image is sent for the detection and receive the processed image and diseases and to interact with the ViewModel.

ViewModel in the MVVM is used as the mediator between View and Model. It takes all the user events and sends requests with the input image for the Model for data. When the model receives detection details from the python backend, it returns to ViewModel and then ViewModel sends processed images and diseases to View.

View in the MVVM contains the widget that the user interacts with the application. This interacts with the ViewModel, when the user events occur.

3.6 Quality Attributes

Accuracy

With the achieved detection accuracy of 92 percent of the trained model the application is able to predict the correct disease a tomato plant is affected with and provide the necessary control measures to overcome those diseases as that is the main objective of this project to support urban farmers in detecting tomato plant diseases. Although accuracy is not critical in this context because wrongly detected diseases may not cause severe damage but in order to accomplish the goal of this project the accuracy should be there to some extent.

Usability

As the end users of this application are urban farmers, usability was treated as a critical factor. The system features were made easy to learn by making the interfaces familiar to

users for instance the icons like camera, gallery were chosen such that the end user can identify them quickly. The application consists of an efficient navigation system and distinct views that the user feels comfortable using. In addition, if a user error occurs, the user can easily return to the previous state. Thus the usability quality attribute is ensured in our application.

Reliability

Software reliability is the ability of a computer program to perform its intended functions and operations in a system's environment, without experiencing failure. In this context reliability factor is ensured as the farmers are able to use this application whenever they need it.

Performance

Performance requirements describe response time, resources required and the survivability. Therefore our system was made interactive and the delays involved were made minimal by making the application simpler, reducing the number of event sources and avoiding complex data arrival patterns and thereby enhancing the user experience so that users may not get bored waiting for the application to respond.

Chapter 4 - Implementation

4.1 Methodology

A general overview of the methodology adopted is presented as follows.

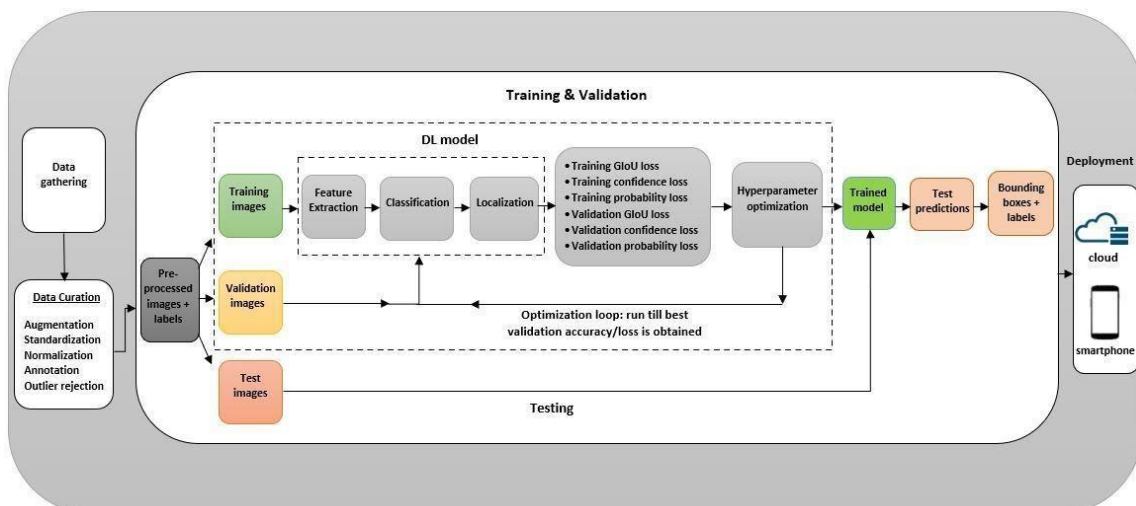


Figure 4.1: Methodology

4.2 Data Collection

Two benchmarked datasets are used in this work. One is PlantVillage [25] dataset. It contains 54309 images for 14 crops and 38 classes which are both healthy and diseased categories. As mentioned in the [25] images are taken in laboratory setups such that they collected leaves by removing them from the plant and then placed against a paper sheet that provided a grey or black background. Table 3.1 and Figure 3.2 give a summary of the PlantVillage dataset.

Second one is PlantDoc [26] dataset which contains 2569 images with 13 plant species and 30 classes of both healthy and diseased. This dataset contains images with heterogeneous backgrounds where the images are taken in natural environments. Table 3.2 and Figure 3.3 give a summary of the PlantDoc dataset.

From these two datasets only images of 5 different types of diseased and healthy leaves of tomato plant were extracted in this work.

Table 4.1: Summary of PlantVillage dataset

Classes	No of images
Leaf Mold	952
Bacterial Spot	2127
Late Blight	1910
Yellow Leaf Curl Virus	5357
Healthy	1592



Figure 4.2: Images of classes from PlantVillage dataset

Table 4.2: Summary of PlantDoc dataset

Classes	No of images
Leaf Mold	91
Bacterial Spot	110
Late Blight	111
Yellow Leaf Curl Virus	76

Healthy	63
---------	----



Figure 4.3: Images of classes from PlantDoc dataset

4.3 Image Annotation

To train a model for the object detection task we need to input images with specifying where the objects are located in the image. LabellImg tool is used to draw the bounding box around the leaf and to annotate the class manually in all images. The image can have multiple leaves or a combination of healthy and diseased leaves in real scenarios. Therefore explicitly with their particular classes we label the diseased leaves in the image and make sure that the entire leaf is presented inside the box. Output of this step is an xml file corresponding to each image that contains information about all the coordinates of the bounding boxes of different sizes with their respective class labels.

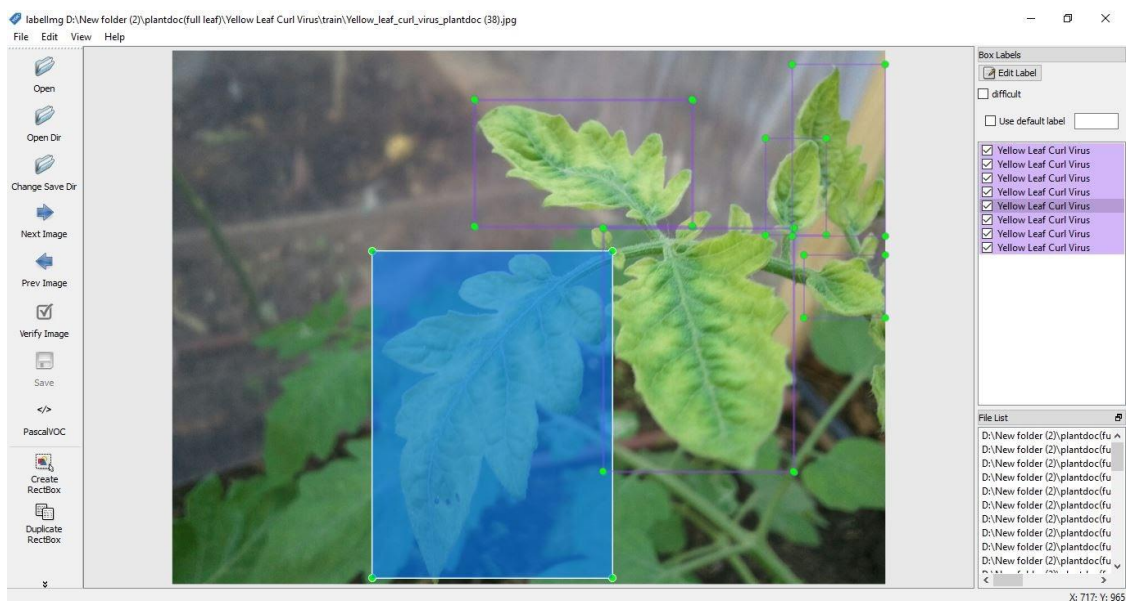


Figure 4.4: Image annotation using LabellImg tool

4.4 Data Augmentation

Data augmentation must be followed if the data set does not contain sufficient number of images. Although we collected a considerable number of images, image augmentation was applied to further improve the learning of the model. We used several techniques that basically increase the number of images of our dataset. These techniques consist of geometrical transformations such as random horizontal flipping, random crop and random translation.

Data augmentation can overcome the problem of overfitting in deep neural network systems training. The problem of overfitting occurs when there is random noise or errors. With more images following expansion using techniques for data augmentation, the model can learn as many irrelevant patterns as possible during the training process, thus preventing overfitting and improving the performance.

4.5 Deep Learning Model

The YOLOv3 [27] model is used as the object detection model in our work. YOLO (You Only Look Once) is a faster object detection algorithm that uses convolution neural networks for detecting objects of various sizes. It is a sliding window and classification approach where you look at the image and classify it for every window which is called one-stage detection or one shot detection. In a region proposal network which is used in R-CNN and Fast R-CNN, the image is looked at in two steps. First is to identify where the objects might be and the second one is to classify it. Drawback of this region proposal network is taking more processing time and chances of occurring false positives are more. So YOLO was coming into existence to overcome all these problems. YOLO treats the problem of detection as a regression problem not as a classification problem.

YOLOv3 is an improvement over previous YOLO detection networks of YOLOv1 [21] and YOLOv2 [28]. It's features are multi scale detection, stronger feature extractor network and some changes in the loss function.

The below graph is sufficient to demonstrate that YOLOv3 has achieved a very high accuracy rate under the premise of ensuring speed. Although it shows high accuracy rates in models like RetinaNet-101 and FPN FRCN the detection time is very high.

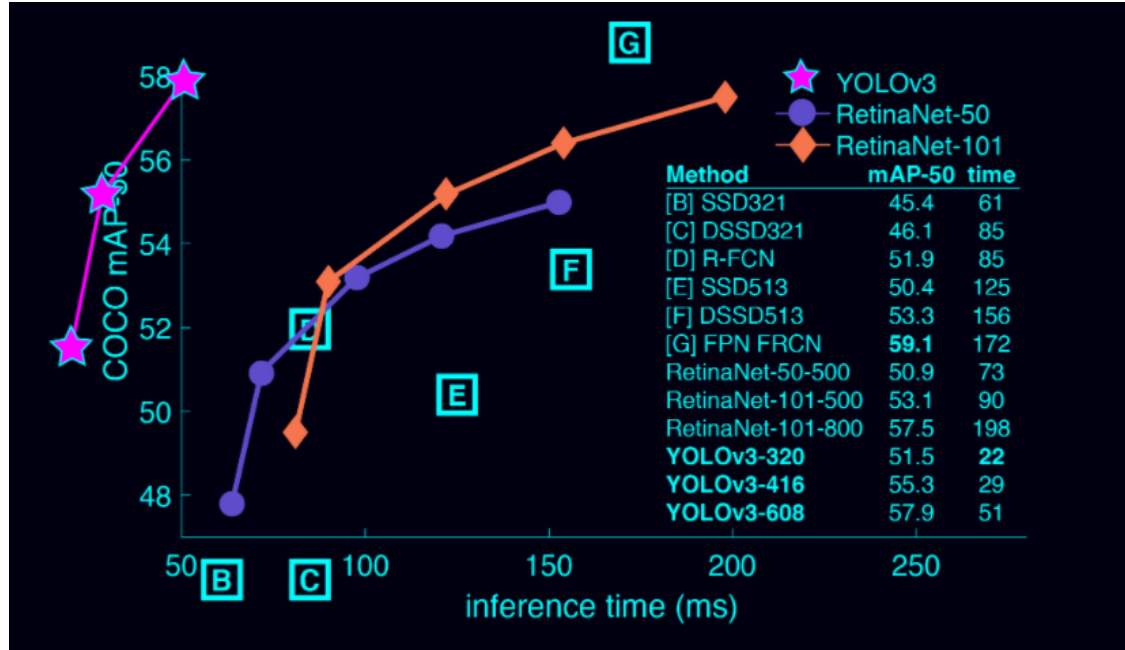


Figure 4.5: Comparison with other object detectors [29]

4.5.1 YOLOv3 Design

The YOLO algorithm treats the object detection problem as a regression problem and divides the image into an $S \times S$ grid. The grid that is responsible for detecting the target object is discovered by tracing to which grid the center of the target is fallen into. Each grid yields a bounding box containing 4 values consisting of cartesian values of the center of the bounding box and width and height of the bounding box, a confidence indicating the probability of containing objects in this prediction box and a class probability map containing the class probability of the object.

Darknet 53 is used as the feature extractor in the YOLOv3 which contains 53 convolution layers. It is mainly composed of convolutional and residual structures. By implementing a residual unit it improved the depth of the network to protect against

gradient disappearance. After every convolutional layer, batch normalization and dropout operations are added to prevent overfitting.

YOLOv3 makes 3 scales detection to adapt the size of different objects using 32, 16 and 8 strides. YOLOv3 downsamples the input image to 13 x 13 and predicts at the 82nd layer in the first scale. YOLOv3 then applies one convolutional layer to the feature map from the 79th layer before upsampling it by a factor of two and concatenating it with the feature map from layer 61. The combined feature map is then passed through to some more convolutional layers up to the second detection scale which produces a 3-D tensor at layer 94. The diagram below depicts the detection procedure of YOLOv3.

To predict the third scale, the same design is repeated once more. The feature map from layer 91 is concatenated with a feature map from layer 36 after being passed through one convolutional layer. The final prediction layer is completed at layer 106, resulting in a 3-D tensor. In short, what simply happens in this case is that YOLOv3 predicts 3 different detection scales. For example, if we feed an image of size 416x416, we get three different output shape tensors: 13 x 13 x 255, 26 x 26 x 255, and 52 x 52 x 255. The diagram below depicts the detection procedure of YOLOv3.

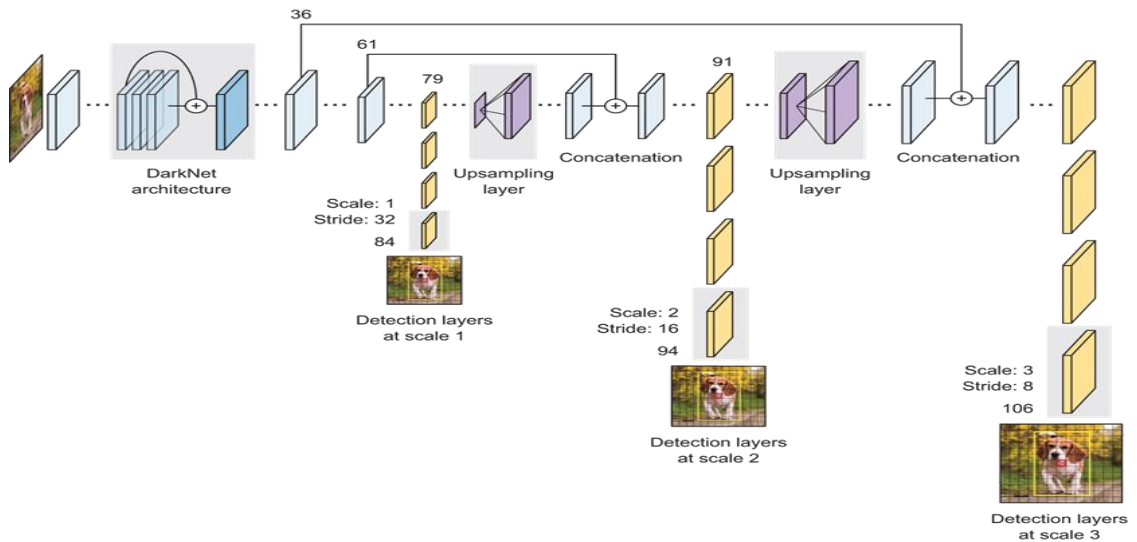


Figure 4.6: YOLOv3 network architecture [30]

4.5.2 Justification for Choosing a Object Detection Model Over a Classification Model

There are several researches conducted on plant disease detection. But those researches used classification models to detect the disease. Hence those models are by giving a dataset that contains images of a single leaf with a homogenous background and doing the predictions with the images taken under the same conditions. But our task is to detect the diseased leaves from the images that contain single leaf as well as multiple leaves in the heterogeneous backgrounds. So to do that the best choice is to go for an object detection model which acts as a combination of image classification and object localization. It uses an input image and generates an or several bounding boxes with the class label attached to each bounding box.

4.6 Model Training

The YOLOv3 model is pre-trained for detection on the COCO dataset that consists of 80 different classes. Inorder to accomplish the goal in our study we retrained the model on our preprocessed tomato plant diseases dataset using transfer learning technique so that we can reuse the already pre-trained model on our new problem. Thus it is possible to transfer the weights that a network has learned at the initial training process to apply on a new task.

Dataset of 3150 images of tomato leaves which contains 630 images for each class was used in this work. 630 images consists of all of the images from the PlantDoc dataset (The number of images per class in the PlantDoc dataset is mentioned in Table 4.2) and the remainder from the PlantVillage dataset. Dataset was divided into 80% for the training set, 10% for the validation set and 10% for the testing set.

The weight parameters provided in the official website of YOLOv3 were therefore used in order to initialize network training and the images in the annotated tomato disease dataset were randomly used for network parameters in the training, so that we can have a good detection result for the entire model. The parameter configuration of our YOLOv3 model during the training is shown in the following table.

Table 4.3: Parameter configuration during the training

Parameter	Value
Batch size	4
Learning Rate	Initial learning rate - $1e-4$ Ending learning rate - $1e-6$
Epochs	100
Match Threshold	0.3

4.7 Graphical User Interfaces (GUI)

Launch Screen

Launch screen, also known as splash screen is added to improve the user experience instead of having a blank screen until the app initializes.

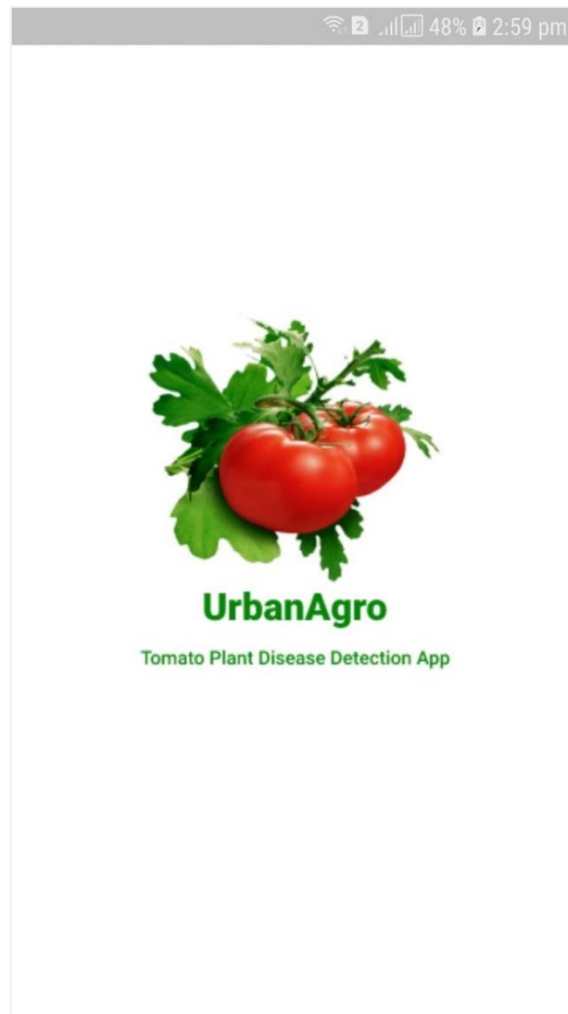


Figure 4.7: Launch screen

Home Screen

The Home Screen has two options for selecting an image. One is from the device camera and the other is from the gallery. After taking or selecting an image user is navigated to Upload Image screen.

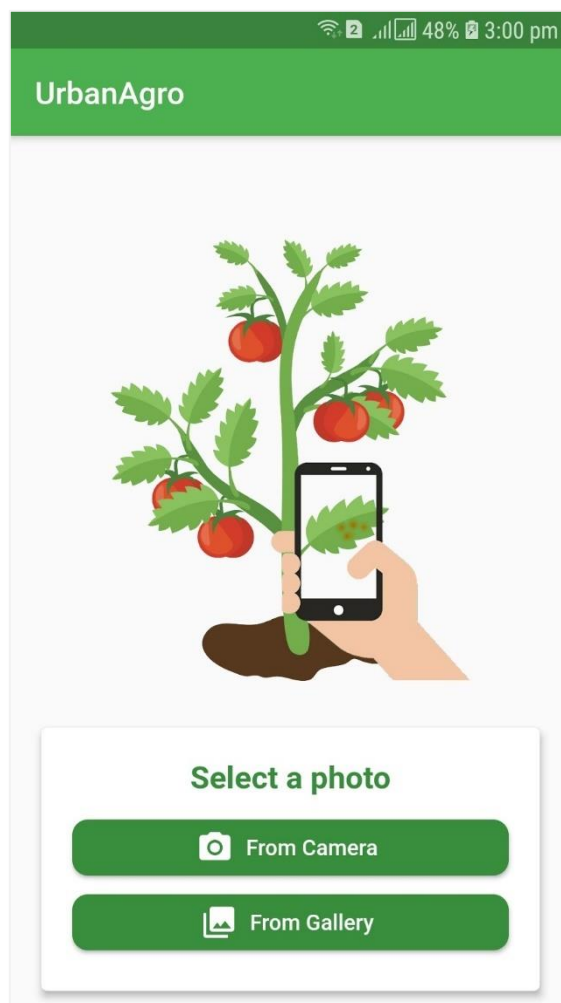


Figure 4.8: Interface of Home screen

Upload Image Screen

If the user is satisfied with the image selected, then the user can upload the image by tapping the upload image button. If not, the user can go back to the Home screen and select another image. Internet connection is required to upload the image. Therefore when tapping the upload image button it will first check the Internet connection and if the Internet connection is not available, the user will be informed that Internet connection is not available. If the Internet connection is available it will upload the image (and detect the diseases using the trained model and send results back to the application) then navigate to the Result screen.

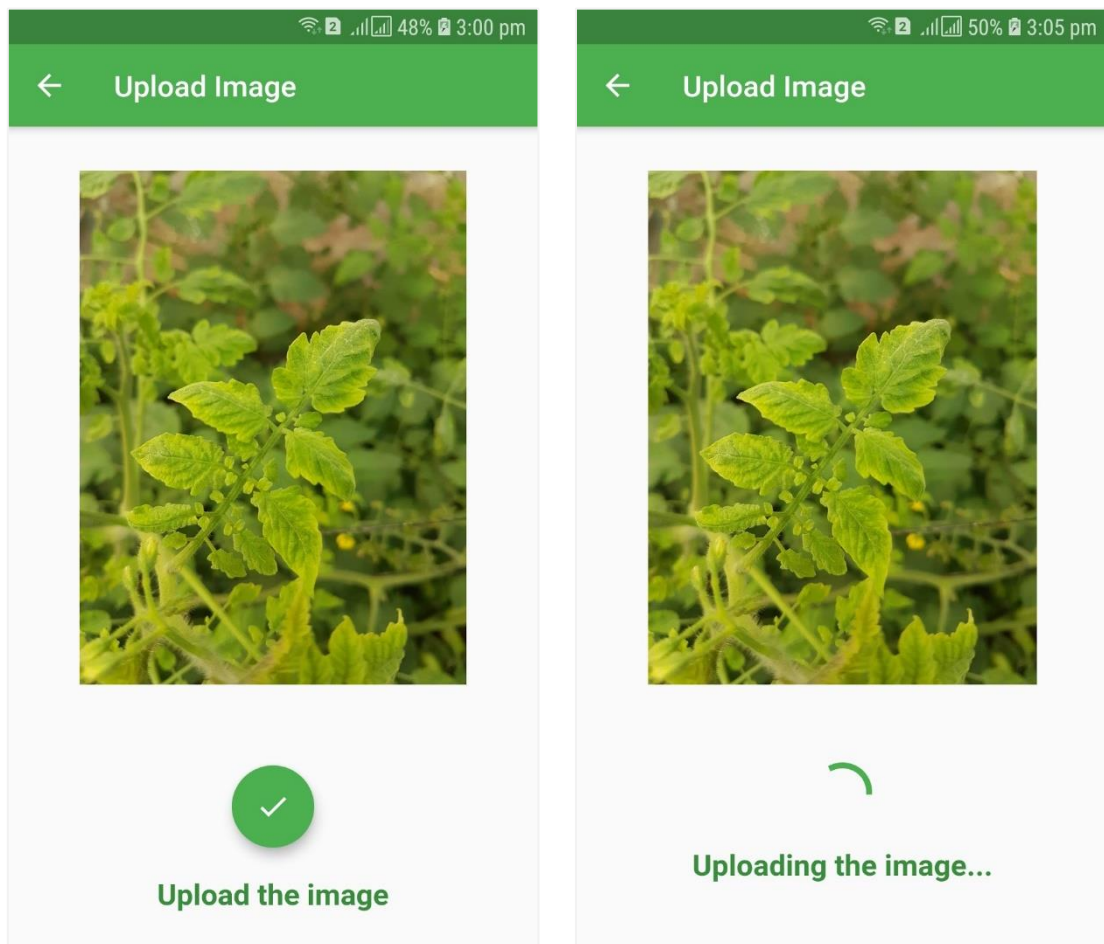


Figure 4.9: Interface of Upload Image screen

Result Screen

The processed image and disease control measures are displayed on the Result screen. The processed image is displayed on the Result screen, along with bounding boxes of diseased and healthy leaves, as well as their respective probabilities. Users can zoom in the image for better inspection.

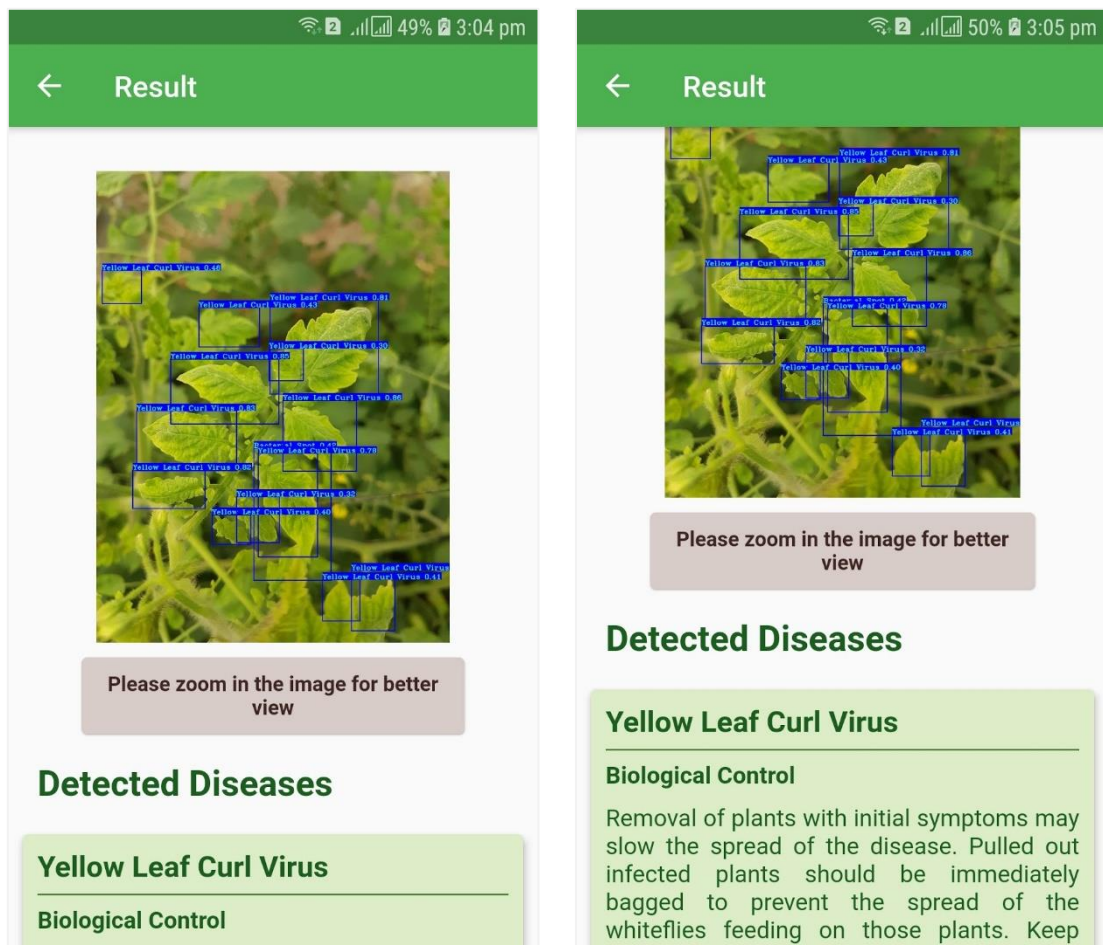


Figure 4.10: Interface of Result screen: displaying the processed image and control instruction

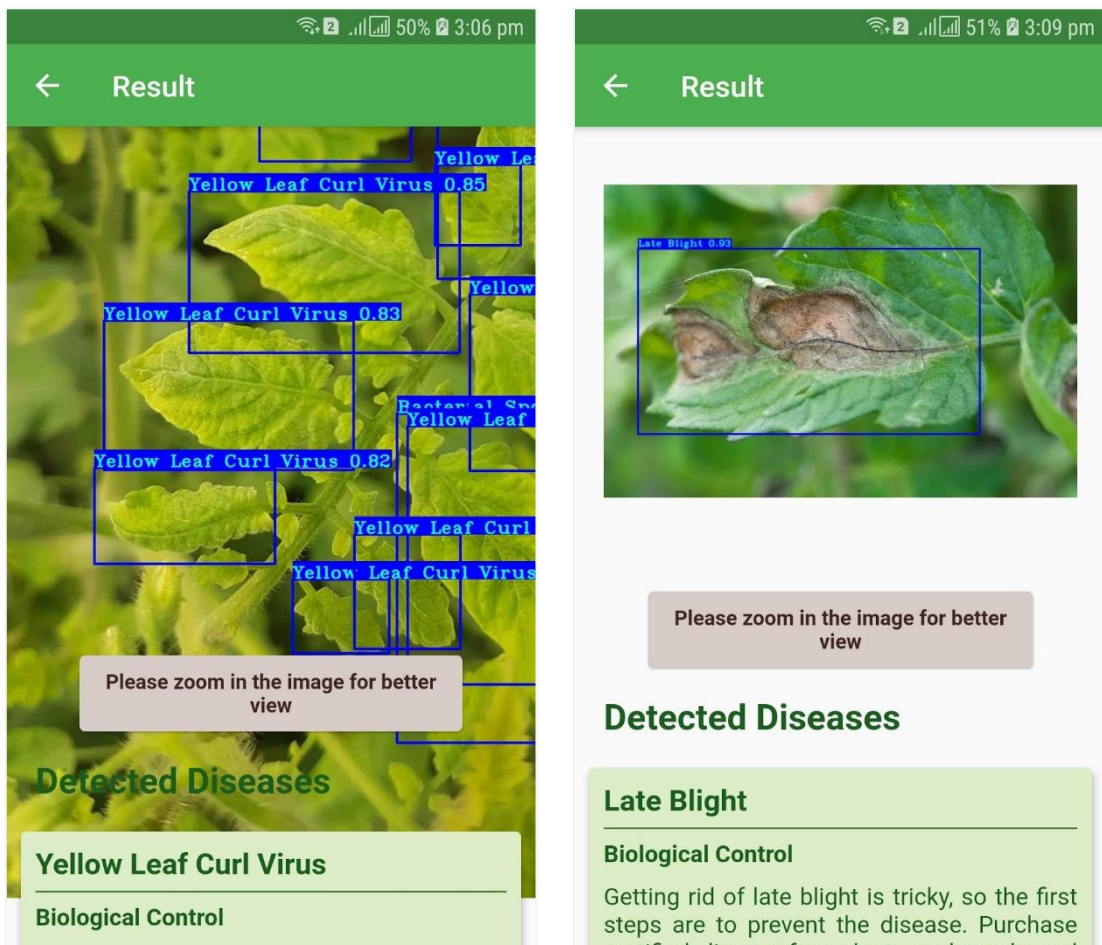


Figure 4.11: Interface of zoomed in image and results showing for Late Blight disease

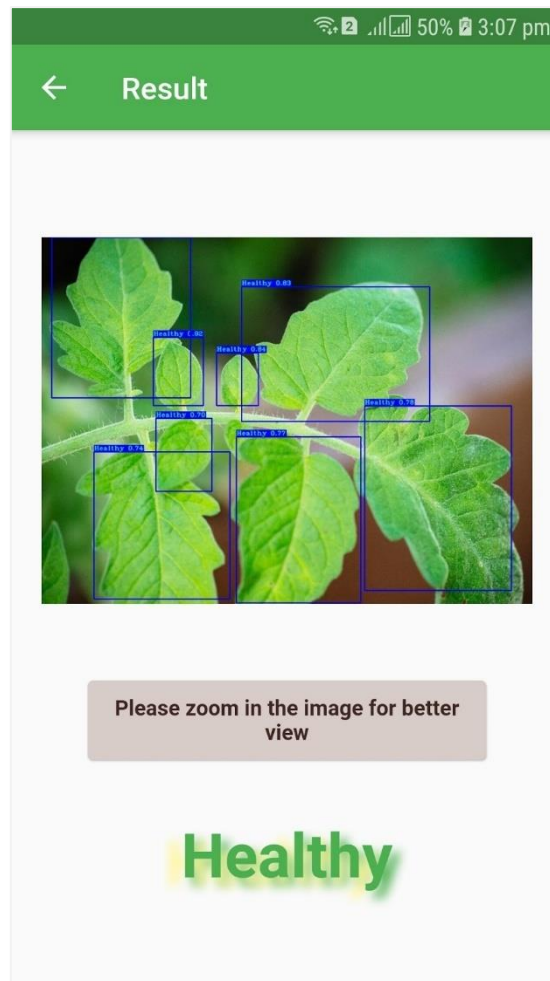


Figure 4.12: Interface of a Result screen showing results as healthy

4.8 The Use of Design Patterns

State design pattern

The State design pattern which is a behavioural design pattern was used in our Flutter application to make it easier to change the appearance of the widgets in our application according to different internal state changes. We could easily add new states by encapsulating each state and its implementations in a different class, and also could change the existing states independently of each other. For an instance in the process of image uploading in our app when we click on the check button to upload the image the progress indicator should be displayed replacing the check button along with the text saying “Uploading the Image”.

Adapter design pattern

Adapter design pattern was used to exchange data between our frontend Flutter application with the backend Python application so that these incompatible interfaces can collaborate. For instance when data is passed between the two applications we wrap the data into a JSON object.

Facade design pattern

Facade is a structural design pattern that provides a simplified interface to a complicated subsystem. For example in our python application we used the Flask framework to listen to http requests and in the flutter application flutter packages were used to access the camera and gallery without coding from scratch. We could use the methods of those readily available functions off the shelf without the complex implementation behind them.

4.9 Justification for Tools and Technologies

Python

Python 3.6 was used for implementing the neural network and for image processing. There are many libraries that exist in python for image processing and neural networks to facilitate the development process.

Tensorflow

Tensorflow is a free and open source framework for developing and using Neural networks. It consists of neural network models and functions necessary for operation in neural networks development. So Tensorflow was used as the machine learning platform.

Github

Github was used as the version controlling application because it was familiar and easy handling.

Labellmg

Labellmg is a tool for annotating graphical images. It's written in Python and has a graphical user interface built with Qt. It was used to annotate our dataset by drawing bounding boxes around the diseased leaves.

Flutter

Flutter is a mobile development framework that has become quite popular across the world. It has all the elements from cross-platform and native development models to build robust applications in minimal time. In contrast to a native mobile development approach, Flutter allows the creation of a single code base that works for both iOS and Android devices. Unlike React Native which requires a “bridge” from the JavaScript (JS) code to the device’s native environment, Flutter, whose programming language is Dart, can access native features of the mobile device directly without any additional interlayers. Thus it compiles quicker and consumes fewer resources to execute the code. That results in the instant start-up of the application, faster performance, and the ability to process multiple threads and complicated animations with less load on the device. Hence Flutter was chosen over other mobile development platforms to develop the mobile app.

Keras

Keras is one of the leading high-level neural networks APIs as it was created to be user friendly, modular, easy to extend, and to work with Python [31]. It is written in Python and supports a variety of neural network computation engines on the back end. In our model we used Keras as the library that provides the Python interface for the neural network.

Flask

Flask is a web application framework written in Python. In the common situations the equivalent Web application Flask is more explicit than other web frameworks like Django[32]. Flask is also easy to begin as a newbie, as there's a small boilerplate code to run a simple app. So Flask was used as the web framework in our python backend so that our Flutter frontend can send http requests and retrieve prediction results.

Pytest

Pytest is a framework that makes it simple to create simple and scalable tests. Tests are expressive and readable, with no need for boilerplate code. Due to different capabilities of pytest such as fast test mechanism, highly customizable simple scripting mechanism pytest was chosen over other testing tools such as 'nose' and 'unittest'.

Chapter 5 - Results and Analysis

The main intention of this study is to aid urban tomato farmers in detecting tomato plant diseases that are commonly turning up in tomato crops, and providing them with necessary control measures so that the plants can be treated accordingly even though the farmers are in shortfall of mastery in the area.

The YOLOv3 object detection model was chosen as the detection model over other object detection models as it is more accurate and faster. To train the model to obtain the most promising detection results, we explored training the model with different blending of the dataset.

The experiments were undertaken only for the two diseases, Late Blight and Bacterial Spot, to examine the perfect blending of the annotated data that should be fed into the model so that the results produced are optimal.

We explored two strategies using about total of 400 images of both the aforementioned diseases by annotating them as explained below;

Experiment 1 : Annotation of the images was done by drawing bounding boxes to the full leaf in the images taken from the PlantVillage and PlantDoc datasets.

Experiment 2 : Annotation of the images was done drawing bounding boxes only around the diseased area in the images taken from the PlantVillage and PlantDoc datasets.

loss

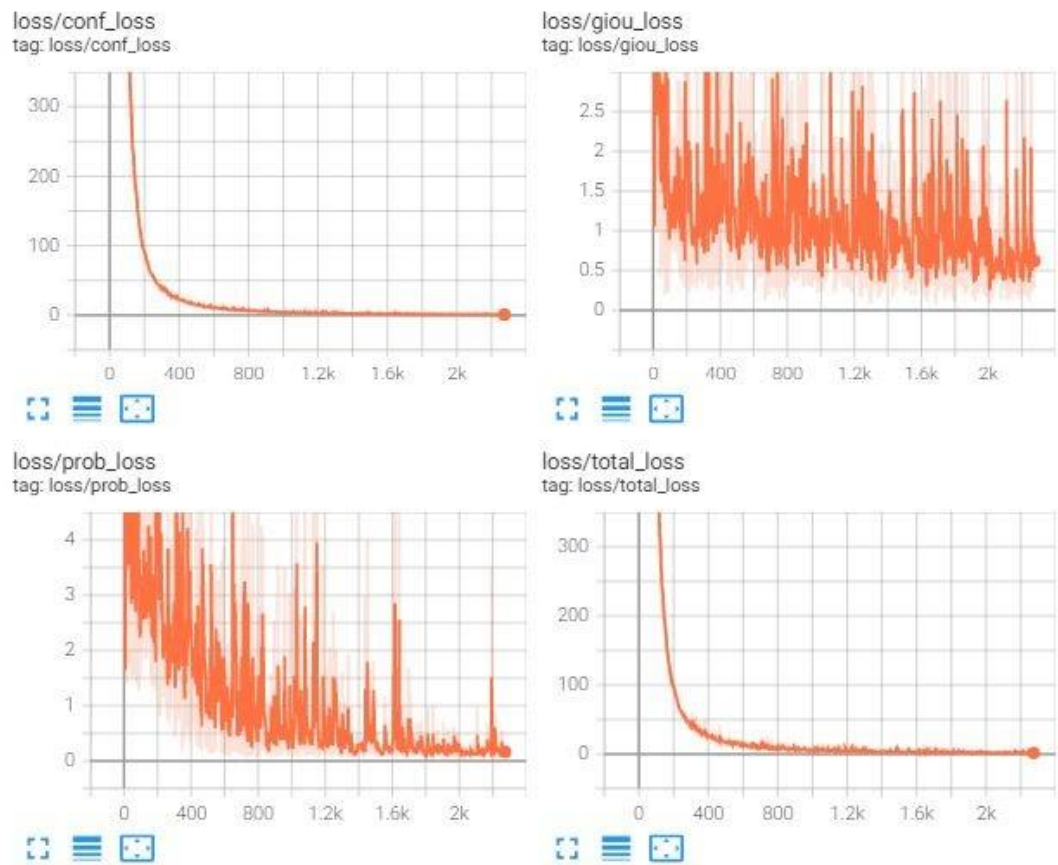
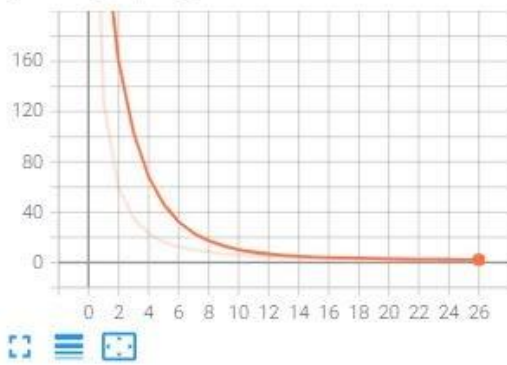


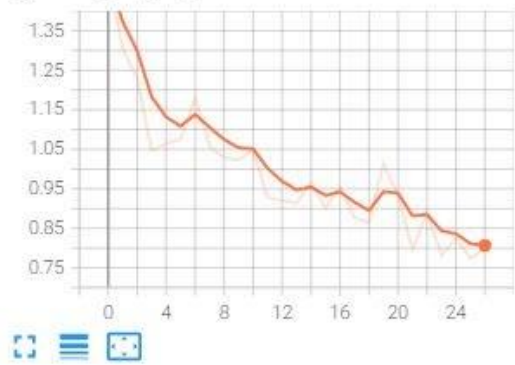
Figure 5.1: Training Loss observed in Experiment 1

validate_loss

validate_loss/conf_val
tag: validate_loss/conf_val



validate_loss/giou_val
tag: validate_loss/giou_val



validate_loss/prob_val
tag: validate_loss/prob_val



validate_loss/total_val
tag: validate_loss/total_val

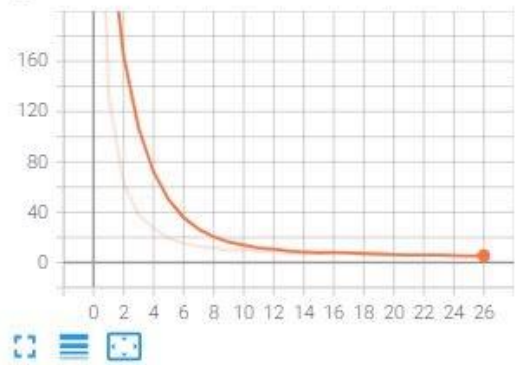


Figure 5.2: Validation Loss observed in Experiment 1

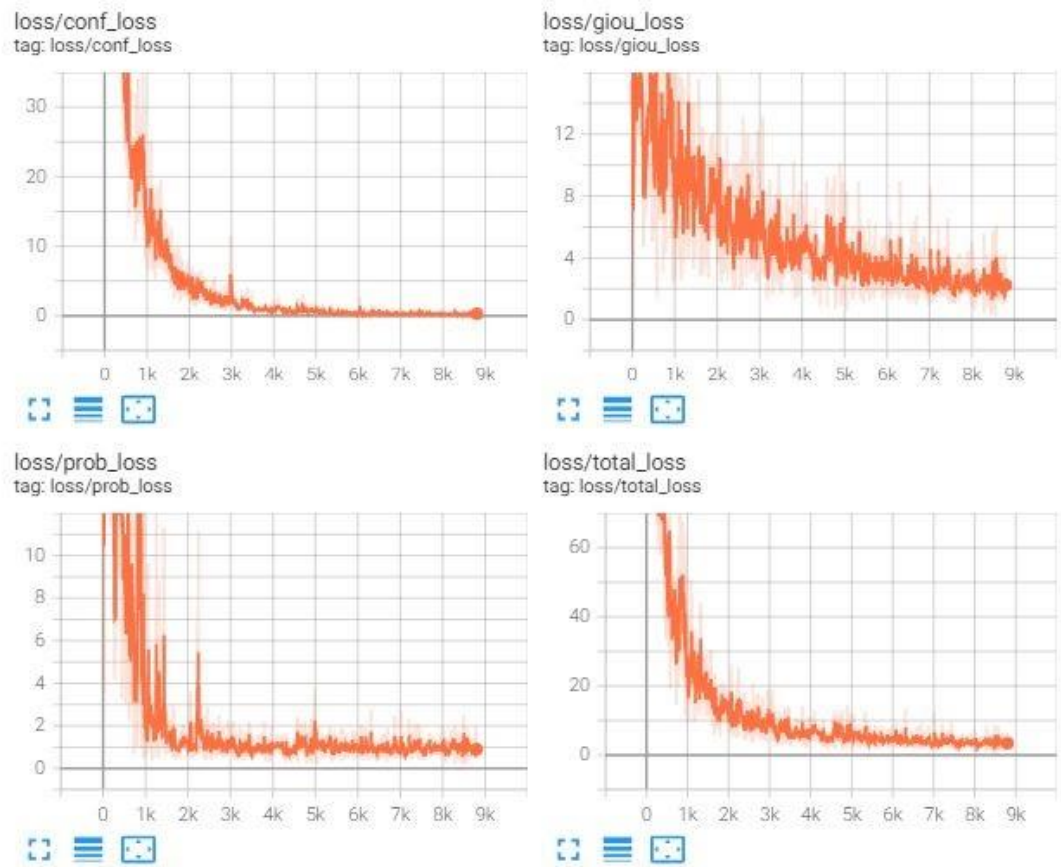


Figure 5.3: Training Loss observed in Experiment 2

validate_loss

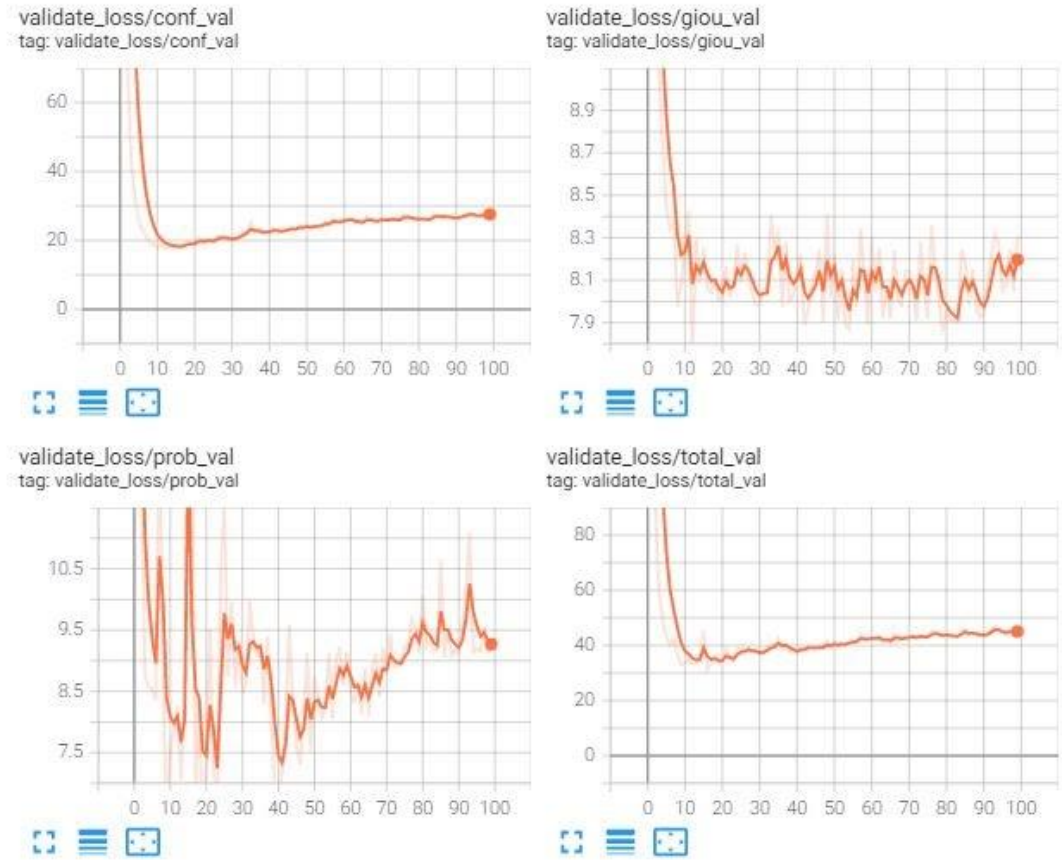


Figure 5.4: Validation Loss observed in Experiment 2

Illustration of the results observed

Table 5.1: Results obtained from experiments

Strategy	Average Precision
Experiment 1	Bacterial Spot : 84.683% Late Blight : 94.101% Mean Average Precision : 89.392%
Experiment 2	Bacterial Spot : 16.492% Late Blight : 48.940% Mean Average Precision : 32.716%

Some predictions from Experiment 1 are shown in Figure 5.5 and Figure 5.6.

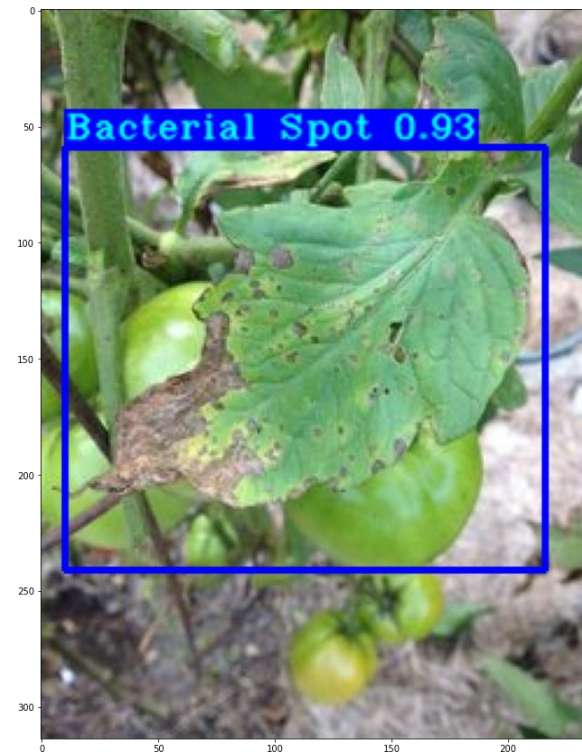


Figure 5.5: Prediction of disease in Experiment 1



Figure 5.6: Prediction of disease in Experiment 1

Some predictions from Experiment 2 are shown in Figure 5.7 and Figure 5.8.

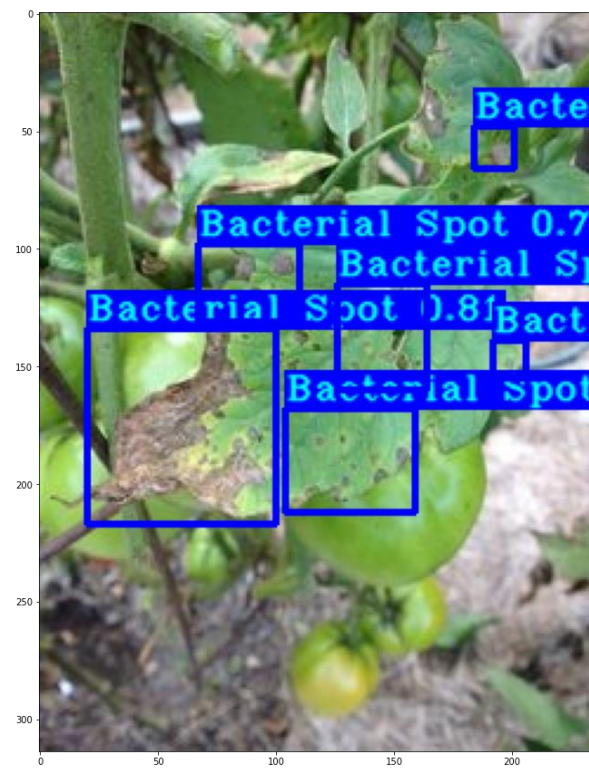


Figure 5.7: Prediction of disease in Experiment 2

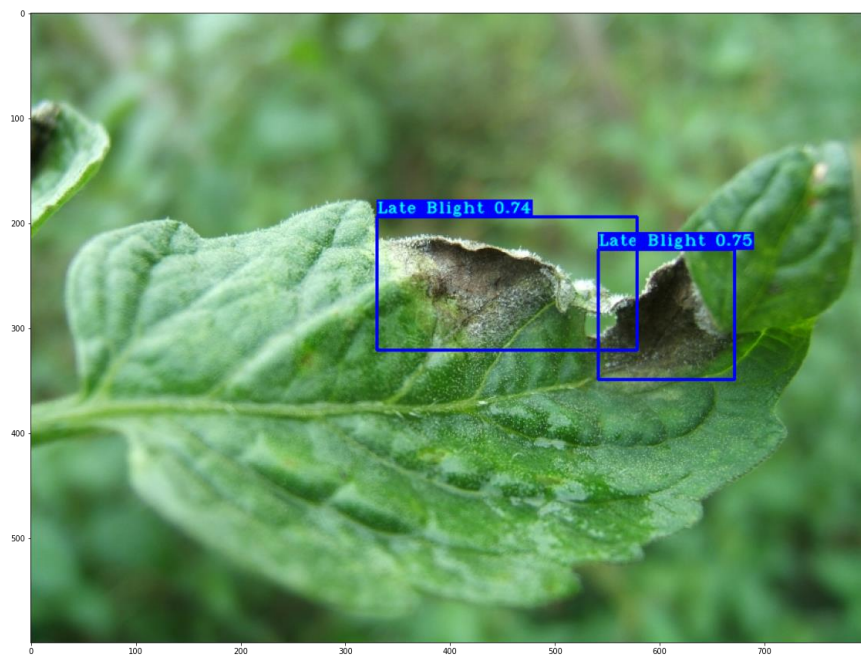


Figure 5.8: Prediction of disease in Experiment 2

Possible reasons for the deviation of the above results were assumed as;

The features for learning the characteristics of the diseases were more distinguishable when the images were annotated to be examined by the model for the leaf as a whole rather than they were annotated region wise only for the diseased area.

Locating the diseased area in the test set so that they overlap with the manually annotated bounding boxes becomes more methodical in Experiment 1 than Experiment 2 as in Experiment 1 it is just a matter of locating the leaf.

Thus it explains the causes for the high average precision values in experiment 1. Observing the above facts it was obvious that we should adapt the strategy in Experiment 1 in order to anticipate optimal results in our study.

Chapter 6 - Evaluation and Testing

6.1 Evaluation of the Model

There are various deep learning architectures available for image classification. These classification models are evaluated in terms of accuracy and parameters. The metrics mAP and IOU are used to assess classification and localization performance.

IoU is used to assess the accuracy of the object detector on the dataset. The definition of Intersection Over Union (IoU) is as follows:

$$IoU = \frac{\text{Area of overlap}}{\text{Area of union}}$$

The intersection of the ground truth bounding box and the predicted bounding box is defined in the above equation as the area of overlap. The area of union is defined as the area bounded by both the ground truth bounding box and the predicting bounding box.

Mean average precision (mAP) is used in determining whether or not the predicted object is correct. It is also the most common assessment method. The detection is then marked as correct or incorrect after calculating the IoU threshold between the ground truth and the anchors for that result. Once the true positive and false positive value is achieved, the precision recall curve is calculated.

$$mAP = \frac{1}{N} \sum_{r=0}^1 (\max (p(\bar{r})))$$

$$\bar{r} \geq r$$

where p is precision and r is recall.

Dataset of 3150 images of tomato leaves which contains 630 images for each class was used in this work. Dataset was divided into 80% for the training set, 10% for the validation set and 10% for the testing set and achieved 92.2% of mAP on the test set. The training loss and validation loss are shown in Figure 4.1 and Figure 4.2. Average Precisions (AP) achieved on the test set for each class are presented on Table 6.1.

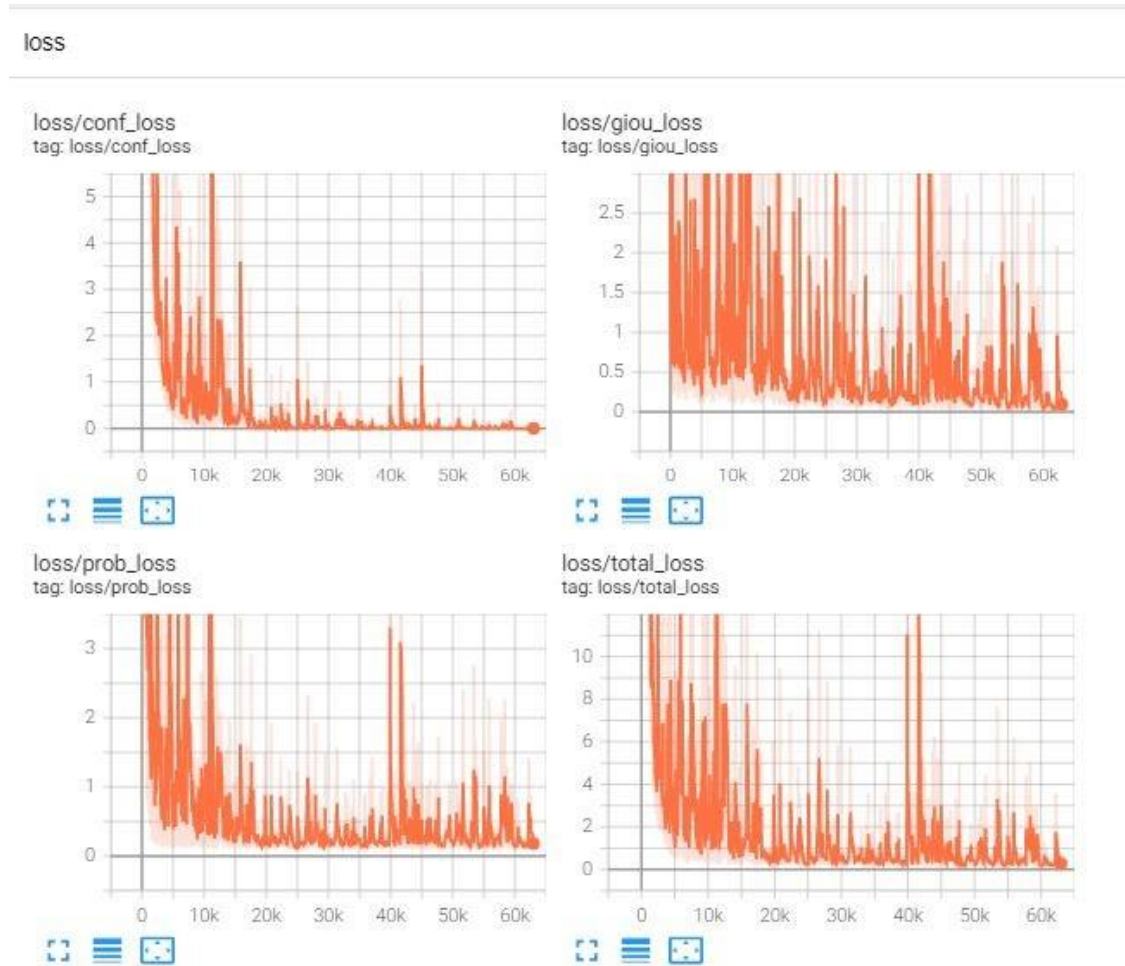


Figure 6.1: Training loss

validate_loss

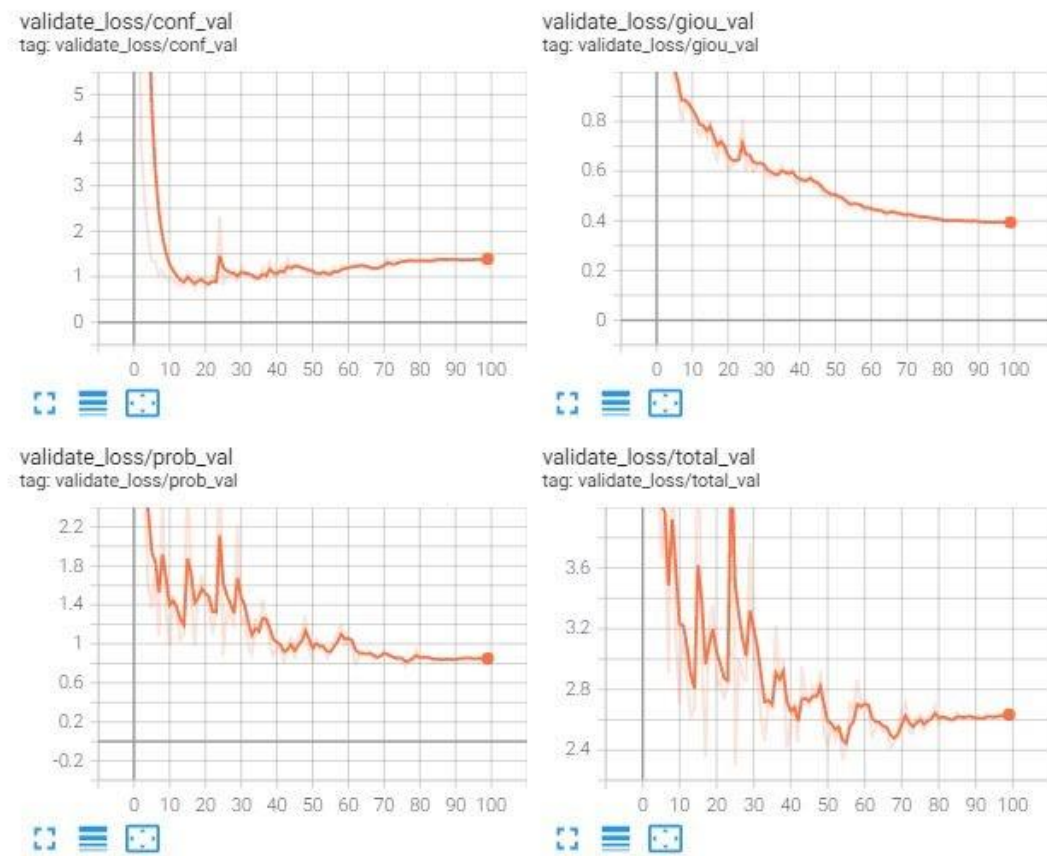


Figure 6.2: Validation loss

Visualization of the bounding boxes and class labels predicted by our trained model are shown in Figure 6.3, Figure 6.4 and Figure 6.5.



Figure 6.3: Prediction of the disease

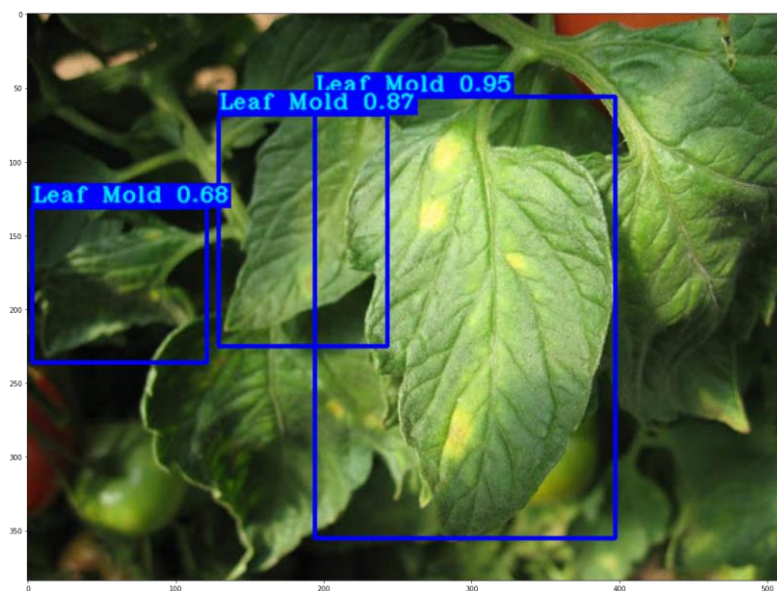


Figure 6.4: Prediction of the disease

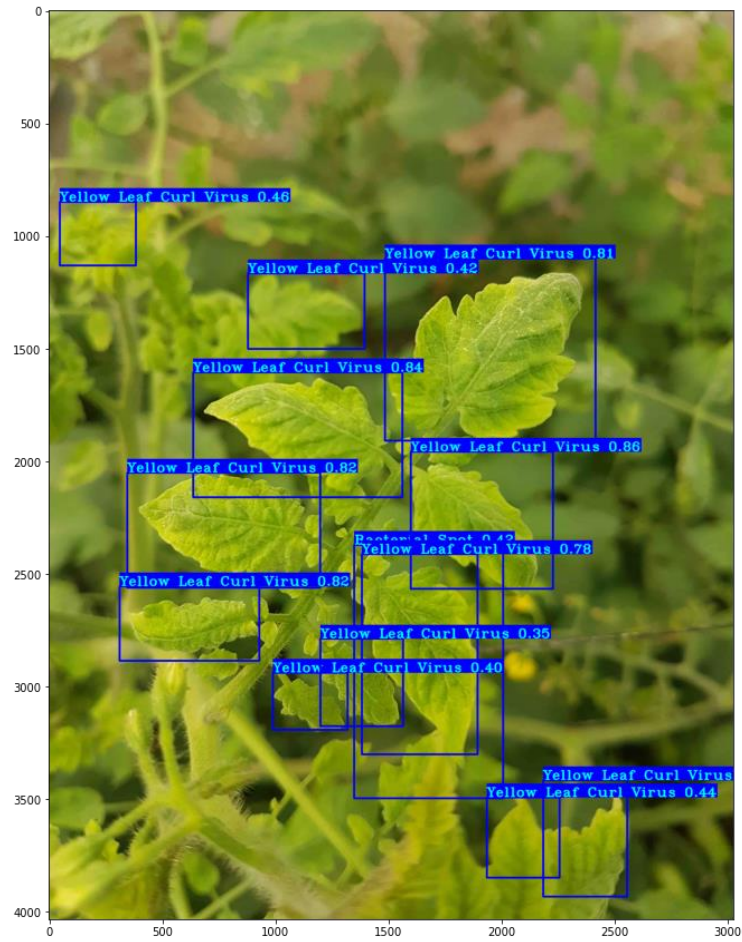


Figure 6.5: Prediction of the disease

6.1.1 Model Results

Table 6.1: Average Precisions (AP) on the test set for each class and mAP

Class	Average Precision
Leaf Mold	93.970%
Bacterial Spot	86.066%
Late Blight	94.717%
Yellow Leaf Curl Virus	91.913%
Healthy	94.334%
Mean Average Precision (mAP)	92.200%

6.1.2 Comparison of Our Results with Previous Works

A comparison of our work with the previous works carried out with object detection models is presented in Table 6.2.

Table 6.2: Comparison with previous work

Study	Classes	Dataset	Approach	mAP
Fuentes et al. [22]	Gray mold, Canker, plague, Miner, Leaf mold, Low temperature, powdery mildew, Whitefly, Nutritional excess	2823 images with 9 classes of tomato plant	Faster R-CNN with VGG-16 SSD with ResNet-50 R-FCN with ResNet-50	83% 82.53% 85.98%
Cynthia et al [23]	Blast of Rice, Sigatoka Leaf Spot of banana, Black Spot of Rose, White Rust of Mustard, Grey Spot of Mustard	236 images with 5 classes of different types of plants	Faster R-CNN	67.34%
Jiang et al [24]	Alternaria leaf spot, Brown spot, Mosaic, Grey spot and Rust (Apple leaf diseases)	26,377 images with 5 classes of apple plant	INAR-SSD (SSD with perception module and rainbow condition) model	78.80%
This work	Leaf Mold, Bacterial Spot, Late Blight, Yellow Leaf Curl Virus, Healthy	3150 images with 5 classes of tomato plant	YOLOv3	92.20%

6.2 Testing Plan

6.2.1 Test Items

Annotating images

Annotated images are double checked manually in case there are missed leaves or improperly annotated leaves in the images when the annotation is done in the first phase. Improperly annotated images may lead to faults in the neural network because we feed these annotated images as the input to the neural network to abstract features.

Neural Network training

The training of the neural network can be visualized using the tensorboard.

Using trained network for predictions

The tests in the training process can be examined by using the trained network for predictions.

Visualizing the predicted class labels and bounding boxes

Captured images/Selected images from the gallery are predicted correctly. Remedies are suggested to the user in accordance to the predicted disease.

6.2.2 Testing Approaches

Unit Testing

Individual functionalities were tested and the issues of those functionalities were discussed among the group. Feedback of the supervisors was taken into consideration and necessary changes were made for the individual components before integration.

Widget Testing

Widgets are UI building blocks of Flutter applications. Widget testing was carried out to ensure that the widget's UI looks and interacts with other widgets as expected which is a unique testing approach for flutter.

Integration Testing

Each individually refined component from the unit testing phase was integrated one by one and tested the dependencies between those components. Dependency issues among components were resolved by performing integrated tests before integrating them together so that the integrated components function properly.

System Testing

To test the complete application, system testing was used. After integrating all the system components and performing integrating tests the whole system was tested as one and checked whether the system fulfills the functional requirements intended by the requirements and fulfill Quality Standards.

Regression Testing

This was done in order to test the issues in the system that are caused due to bug fixes and changes. So that these tests ensure that the system is not prone to bugs due to those changes.

User Acceptance Testing

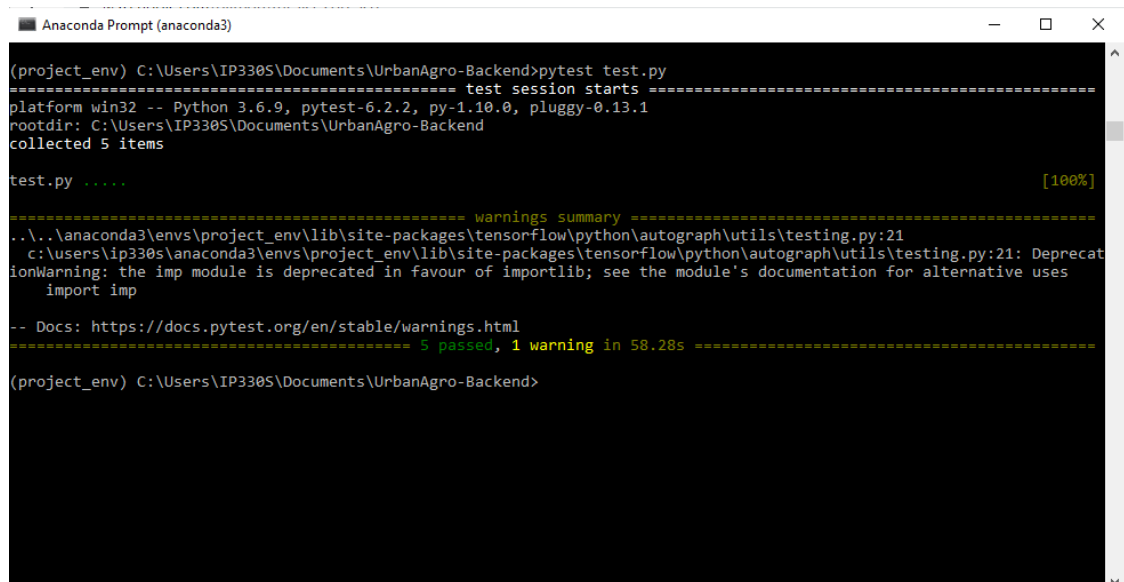
User acceptance of the application is the key factor for the success of our application. Acceptance testing was held with our supervisor Dr. Thilina Halloluwa and co-supervisor Ms. Hiruni Kegalle by getting their feedback.

6.3 Test Results

6.3.1 Automation Testing

The unit tests in our python application were carried out using Pytest. Pytest unit testing ensures that our tests are stateless, makes repetitive tests more comprehensible, runs subsets of codes by name or custom groups, and creates and maintains reusable testing utilities.

The test results of the python scripts written to perform unit testing inorder to ensure that we get the exact results as we expected from the functions are shown in Figure 6.6.



```
Anaconda Prompt (anaconda3)

(project_env) C:\Users\IP330S\Documents\UrbanAgro-Backend>pytest test.py
===== test session starts =====
platform win32 -- Python 3.6.9, pytest-6.2.2, py-1.10.0, pluggy-0.13.1
rootdir: C:\Users\IP330S\Documents\UrbanAgro-Backend
collected 5 items

test.py ..... [100%]

===== warnings summary =====
..\..\anaconda3\envs\project_env\lib\site-packages\tensorflow\python\autograph\utils\testing.py:21
  c:\users\ip330s\anaconda3\envs\project_env\lib\site-packages\tensorflow\python\autograph\utils\testing.py:21: Deprecat
ionWarning: the imp module is deprecated in favour of importlib; see the module's documentation for alternative uses
    import imp

-- Docs: https://docs.pytest.org/en/stable/warnings.html
===== 5 passed, 1 warning in 58.28s =====

(project_env) C:\Users\IP330S\Documents\UrbanAgro-Backend>
```

Figure 6.6: Sample of unit testing results of python application

Unit testing in the Flutter application was carried out using the test package in flutter. Following are the samples of results obtained after unit testing.

```

DEBUG CONSOLE  ...  Filter (e.g. text, !exclude)  Flutter
Connecting to VM Service at http://127.0.0.1:56904/EBY6Q_LX2cc=/ws
✓ Upload Unit Tests isUploading should start with false
✓ Upload Unit Tests uploadingState should start with string-Upload the image
✓ Upload Unit Tests colorState should start with green
Exited

```

Figure 6.7: Sample of unit testing results of Flutter application

```

PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE  1: dart
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS E:\4th year project\flutter projects\UrbanAgro> flutter test test/upload_test.dart
00:01 +3: All tests passed!
PS E:\4th year project\flutter projects\UrbanAgro>

```

Figure 6.8: Sample of unit testing results of Flutter application

Widget testing in the Flutter application was carried out using the `flutter_test` package. Following is a sample test code of Home screen widget testing to test whether `ScrollView` shows up.

```

5  Widget createHomeScreen() => MaterialApp(home: HomeScreen());
   Run | Debug
6  void main() {
   Run | Debug
7    group('Home Screen Widget Tests', () {
8
   Run | Debug
9      testWidgets('Testing if ScrollView shows up', (tester) async {
10         await tester.pumpWidget(createHomeScreen());
11         expect(find.byType(SingleChildScrollView), findsOneWidget);
12       });
13
   Run | Debug
14     testWidgets('Testing if texts are available', (tester) async {

```

Figure 6.9: Sample of widget testing code of Flutter application

Following are the samples of results obtained after Home Screen widget testing.

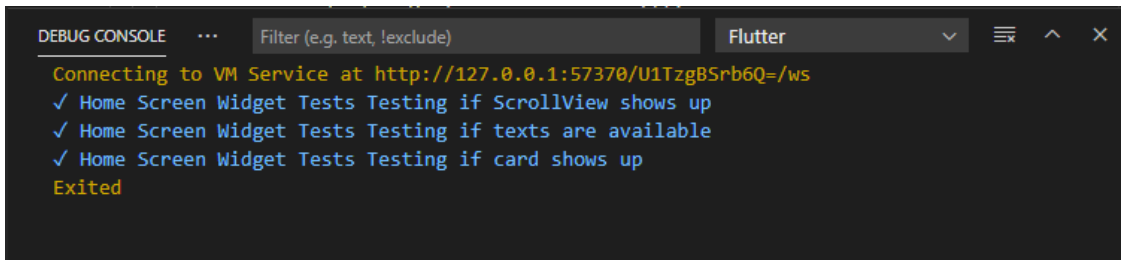


Figure 6.10: Sample of widget testing results of Flutter application

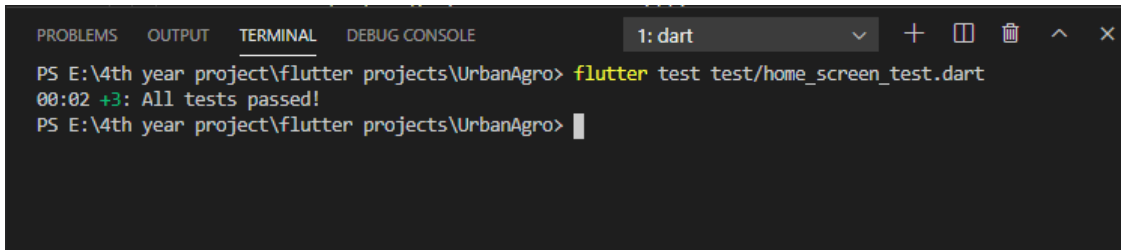


Figure 6.11: Sample of widget testing results of Flutter application

Following is a sample test code of Upload screen widget testing to test whether the progress indicator and the text “Uploading the image...” shows up instead of the upload button and the text “Upload the image” when tap the upload button.

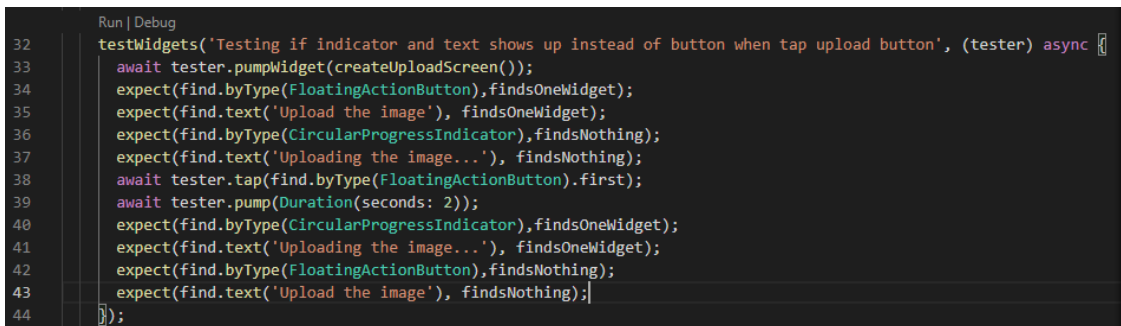


Figure 6.12: Sample of widget testing code of Flutter application

Following are the samples of results obtained after Upload Screen widget testing.

```

DEBUG CONSOLE  ...  Filter (e.g. text, exclude)  Flutter
Connecting to VM Service at http://127.0.0.1:62343/V067t1TaqN0=/ws
✓ Upload Screen Widget Tests Testing if ScrollView shows up
✓ Upload Screen Widget Tests Testing if initial text is available
✓ Upload Screen Widget Tests Testing if button shows up in beginning
✓ Upload Screen Widget Tests Testing if indicator shows up in beginning
✓ Upload Screen Widget Tests Testing if indicator and text shows up instead of button when t
ap upload button
Exited

```

Figure 6.13: Sample of widget testing results of Flutter application

```

PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE  1: dart
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS E:\4th year project\flutter projects\UrbanAgro> flutter test test/upload_screen_test.dart
00:03 +5: All tests passed!
PS E:\4th year project\flutter projects\UrbanAgro>

```

Figure 6.14: Sample of widget testing results of Flutter application

6.3.2 Manual Testing

In addition to the automation tests performed as described above to check whether our system functions properly as expected the manual testing was also carried out throughout the implementation process and after the whole system was built to ensure that our system is up and running according to the requirements established at the outset.

Table 6.3: Manual testing performed

Test Case #	Test Case Description	Test Steps	Expected Results	Actual Results	Pass / Fail
1	Taking a photo from the camera for the first time.	1.Launch the app. 2.Tap on the “From Camera” button.	System should ask the permission to access the camera	As expected	Pass

2	Selecting a photo from the gallery for the first time.	1.Launch the app. 2.Tap on the “From Gallery” button.	System should ask the permission to access the gallery	As expected	Pass
3	Granting the permission to access the camera for the first time.	1.Launch the app. 2.Tap on the “From Camera” button. 3.Clicking on “Allow” button when the permission is requested by the system to access the camera	The user should be directed to the camera allowing him/her to take a photo.	As expected	Pass
4	Granting the permission to access the gallery for the first time.	1.Launch the app. 2.Tap on the “From Gallery” button. 3.Clicking on “Allow” button when the permission is requested by the system to access the gallery	The user should be directed to the gallery allowing him/her to select a photo.	As expected	Pass
5	Denying the permission to access the camera.	1.Launch the app. 2.Tap on the “From Camera” button. 3.Clicking on “Deny” button when the permission is requested by the system to access the camera	The user should be returned to the home screen.	As expected	Pass
6	Denying the permission to access the gallery.	1.Launch the app. 2.Tap on the “From Gallery” button. 3.Clicking on “Deny” button when the	The user should be returned to the home screen.	As expected	Pass

		permission is requested by the system to access the gallery			
7	Taking a photo from the camera after permission is granted.	1.Launch the app. 2.Tap on the “From Camera” button.	The user should be directed to the camera allowing him/her to take a photo.	As expected	Pass
8	Selecting a photo from the gallery after permission is granted.	1.Launch the app. 2.Tap on the “From Gallery” button.	The user should be directed to the gallery allowing him/her to select a photo.	As expected	Pass
9	Uploading an image without the internet connection.	1.Launch the app. 2.Tap on the “From Gallery”/ “From Camera” button. 3.Taking a photo from the camera or selecting a photo from the gallery. 4.Click on the upload button on the upload image screen.	A message should be displayed saying “No internet connection”	As expected	Pass
10	Uploading an random image which is not an image of a leaf providing the internet connection.	1.Launch the app. 2.Tap on the “From Gallery”/ “From Camera” button. 3.Taking a random photo from the camera	A message should be displayed saying “Please upload a valid image”	As expected	Pass

		<p>or selecting a random photo from the gallery.</p> <p>4.Click on the upload button on the upload image screen.</p>			
11	Uploading a diseased tomato plant leaf image providing the internet connection.	<p>1.Launch the app.</p> <p>2.Tap on the “From Gallery”/ “From Camera” button.</p> <p>3.Taking a photo of diseased tomato plant leaves from the camera or selecting a photo of diseased tomato plant leaves from the gallery.</p> <p>4.Click on the upload button on the upload image screen.</p>	The user should be directed to the result screen displaying the photo with detected diseases with bounding boxes and necessary control measures for the identified diseases.	As expected	Pass
12	Uploading a healthy tomato plant leaf image providing the internet connection.	<p>1.Launch the app.</p> <p>2.Tap on the “From Gallery”/ “From Camera” button.</p> <p>3.Taking a photo of healthy tomato plant leaves from the camera or selecting a photo of healthy tomato plant leaves from the gallery.</p> <p>4.Click on the upload button on the upload image screen.</p>	The user should be directed to the result screen displaying the photo detected as healthy with bounding boxes around healthy leafs and notifying the user that the plant is healthy.	As expected	Pass

Chapter 7 - Conclusion

We developed a robust deep-learning-based detector for real-time tomato disease detection, in this study. It introduces a practical and applicable solution for detecting the class and location of diseases in tomato plants, which in fact represents a significant comparable difference with other methods for plant disease classification. The experimental results show that the YOLOv3 model's detection accuracy reached 92 percent. As a result, for the task of detecting tomato diseases, the YOLOv3 algorithm that was proposed can not only maintain a high detection rate, but also meet real-time detection requirements, and accurately and quickly detect the location and category of diseased tomato leaves.

In comparison to the previous studies conducted in this field our system, which employs the YOLOv3 Model, has strong robustness for detecting different object sizes and resolution images in a complex environment, as well as high detection and positioning accuracy, and can meet the needs of tomato disease detection in complex environments using the end user's smartphone camera. Further we have trained our detector to detect diseases that have not been considered in the previous studies.

The discoveries and the outcomes achieved by this study make a significant impact to make the field of plant disease detection move forward. We explored different deep learning models that are available for the purpose of object detection and finally came to a conclusion that YOLOv3 outstands all the other object detectors available such as R-CNN, Faster R-CNN and SSD, not only by accuracy but also by performance. Moving forward YOLOv3 model can be used to make excellence advancements in the field of crop cultivation by mitigating the losses and damages to the crops.

7.1 Shortcomings of the Study

Several limitations of the system should be acknowledged. Our system only detects 5 different classes of diseased and healthy leaves of the tomato plants. The control measures are given considering only the type of disease in the plant such that the

control measures are not given accordingly to the severity of the disease. Identification of diseases affecting the parts other than the leaves in tomato plants is beyond the scope of this study. The detection accuracy could be further improved by training the model on more images with the heterogeneous background. Due to this pandemic situation it was impossible to collect images through field visits. Hence all the images were collected using internet sources.

7.2 Future Directions

It is recommended that this research can be extended to other crops, having improved the detection accuracy of plant diseases in order to mitigate the losses due to plant diseases in the agricultural sector. The system can be improved further by training the model to identify different states of the disease based on the view of the disease in the plant and providing remedies accordingly. In addition the application can be further improved by making a platform where the farmers can post their problems to a forum so that the other farmers who had similar problems can discuss how they overcame such conditions. Thus these facts can be considered when making further improvements to the system.

Peer Evaluation

In accordance with the problem definition in this dissertation, both the team members contributed equally in finding an approach to achieve the proposed goal. Some components in the project were implemented separately and some components were implemented collectively by the two of us.

Contributions made collectively

The tasks that we both collaborated collectively are; researching and reviewing the previous work that has been carried out in the field of plant disease detection, exploring a suitable method to solve the identified research problem and choosing YOLO v3 model from the available deep learning models and writing the dissertation of the study.

Contributor 1: H.A.D.D. Navodi

Annotating images of the diseases Leaf Mold, Yellow Leaf Curl Virus and for the healthy leaf. Training the YOLOv3 model for different numbers of images and different model parameters to compare results and determine the optimal number of images and model parameters to train to achieve the optimal accuracy. In the front end flutter application the user interfaces Launching screen, Home screen and the Image upload screen that sends the input image to the backend for processing the desired result was developed. And the automation testing scripts for testing the frontend application was written and tested the front end application functions as expected.

Contributor 2: W.L.V. Fernando

Annotating images of the diseases Bacterial Spot, Late Blight and the healthy leaf. Training the YOLOv3 model using the images that were annotated only for the diseased area of the leaves and examined whether it improves the accuracy of the detector. In the front end application the user interface Result screen was developed displaying the resultant image from the back end along with the remedies for the identified diseases. And the automation testing scripts for testing the backend application was written and tested the back end application functions as expected.

References

- [1] R. Chandran. (2020, Apr. 09). “Grow your own: Urban farming is flourishing during the coronavirus lockdowns”[Online]. Available: <https://www.weforum.org/agenda/2020/04/grow-your-own-urban-farming-flourishes-in-coronavirus-lockdowns/> [Accessed 29 May 2020]
- [2] Daily News. (2017, Jan. 30). “A toast for tomatoes!”[Online]. Available: <http://www.dailynews.lk/2017/01/30/features/106060/toast-tomatoes> [Accessed 27 May 2020]
- [3] TILASTO. “Sri Lanka: Tomatoes, production quantity (tons)”[Online]. Available: <https://www.tilasto.com/en/country/sri-lanka/geography-and-agriculture/tomatoes-production-quantity> [Accessed 28 May 2020]
- [4] L. Rose. (2018, Dec. 15). “Plant Starts to Grow Tomatoes?”[Online]. Available: <https://homeguides.sfgate.com/long-before-plant-starts-grow-tomatoes-59989.html> [Accessed 27 May 2020]
- [5] “Tomato”[Online]. Available: <https://plantvillage.psu.edu/topics/tomato/infos> [Accessed 28 May 2020]
- [6] H.M. Griffiths. (2014, Mar. 26). “Tomato Leaf Mold”[Online]. Available: <https://www.growertalks.com/Article/?articleid=20665> [Accessed 27 May 2020]
- [7] G. McAvoy. (2019, Oct. 17). “Tactics to Tackle Bacterial Spot of Tomato”[Online]. Available: <https://www.growingproduce.com/vegetables/tactics-to-tackle-bacterial-spot-of-tomato/> [Accessed 27 May 2020]

- [8] Planet Natural. "Late Blight"[Online]. Available:
<https://www.planetnatural.com/pest-problem-solver/plant-disease/late-blight/>
[Accessed 27 May 2020]
- [9] UC IPM. "Tomato Yellow Leaf Curl"[Online]. Available:
<https://www2.ipm.ucanr.edu/agriculture/tomato/Tomato-Yellow-Leaf-Curl/>
[Accessed 27 May 2020]
- [10] HGIC. (2020,Jul. 23). "Tomato Diseases & Disorders"[Online]. Available:
<https://hgic.clemson.edu/factsheet/tomato-diseases-disorders/> [Accessed 27
May 2020]
- [11] S. Vetral and R.S. Khule, "Tomato Plant Disease Detection using Image
Processing", 2017 International Journal of Advanced Research in Computer
and Communication Engineering (IJARCCE), 2017, vol. 6, pp. 293-297.
- [12] U. Mokhtar, M. A. S. Ali, A. E. Hassenian and H. Hefny, "Tomato leaves
diseases detection approach based on Support Vector Machines," 2015 11th
International Computer Engineering Conference (ICENCO), Cairo, 2015, pp.
246-250, doi: 10.1109/ICENCO.2015.7416356.
- [13] H. Sabrol and S. Kumar, "Intensity based feature extraction for tomato plant
disease recognition by classification using decision tree," International Journal
of Computer Science and Information Security (IJCSIS), 2016, vol. 14, no. 9,
pp. 622-626.
- [14] S. Mohanty, D. Hughes and S. Marcel, "Using deep learning for image-based
plant disease detection," 2016.
- [15] M. Brahimi, K. Boukhalfa and A. Moussaoui, "Deep Learning for Tomato
Diseases: Classification and Symptoms Visualization", Applied Artificial
Intelligence, 2017, vol. 31, no. 4, pp. 299-315.

- [16] M. Ouhami, Y. Es-Saady, M. E. Hajji, A. Hafiane, R. Canals and M. E. Yassa, "Deep Transfer Learning Models for Tomato Disease Detection", ICISP 2020, LNCS, 2020, vol. 12119, pp. 65-73.
- [17] R. Girshick, J. Donahue, T. Darrell and J. Malik, "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation," 2014 IEEE Conference on Computer Vision and Pattern Recognition, Columbus, OH, 2014, pp. 580-587, doi: 10.1109/CVPR.2014.81.
- [18] S. Ren, K. He, R. Girshick and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 39, no. 6, pp. 1137-1149, 2015, doi: 10.1109/TPAMI.2016.2577031.
- [19] J. Dai, Y. Li, K. He and J. Sun, "R-FCN: Object Detection via Region-based Fully Convolutional Networks," 2016.
- [20] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C. Fu and A. C. Berg, "SSD: Single Shot MultiBox Detector," ECCV 2016. Lecture Notes in Computer Science, 2016, vol. 9905, pp. 21-37.
- [21] J. Redmon and A. Farhadi, "YOLO9000: Better, Faster, Stronger," 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, 2017, pp. 6517-6525, doi: 10.1109/CVPR.2017.690.
- [22] A. Fuentes, S. Yoon, S. C. Kim and D. S. Park, "A Robust Deep-Learning-Based Detector for Real-Time Tomato Plant Diseases and Pests Recognition", Sensors, 2017, vol. 17, no. 9, p. 2022.
- [23] S. T. Cynthia, K. M. Shahrukh Hossain, M. N. Hasan, M. Asaduzzaman and A. K. Das, "Automated Detection of Plant Diseases Using Image Processing and Faster R-CNN Algorithm," 2019 International Conference on Sustainable

Technologies for Industry 4.0 (STI), Dhaka, Bangladesh, 2019, pp. 1-5, doi: 10.1109/STI47673.2019.9068092.

- [24] P. Jiang, Y. Chen, B. Liu, D. He and C. Liang, "Real-Time Detection of Apple Leaf Diseases Using Deep Learning Approach Based on Improved Convolutional Neural Networks," in IEEE Access, vol. 7, pp. 59069-59080, 2019, doi: 10.1109/ACCESS.2019.2914929.
- [25] D. P. Hughes and M. Salathe, "An open access repository of images on plant health to enable the development of mobile disease diagnostics", 2015.
- [26] D. Singh, N. Jain, P. Jain, P. Kayal, S. Kumawat and N. Batra, "PlantDoc: A Dataset for Visual Plant Disease Detection", 2020.
- [27] J. Redmon and A. Farhadi, "YOLOv3: An Incremental Improvement", 2018.
- [28] J. Redmon, S. Divvala, R. Girshick and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, 2016, pp. 779-788, doi: 10.1109/CVPR.2016.91.
- [29] "YOLO: Real-Time Object Detection"[Online]. Available: <https://pjreddie.com/darknet/yolo/> [Accessed 05 Jan 2020]
- [30] R. Balsys. "Yolo v3 with TensorFlow 2" Python Lessons.[Online]. Available: <https://pylessons.com/YOLOv3-TF2-introduction/> [Accessed 05 Jan 2020]
- [31] M. Heller. (2019, Jan. 28). "What is Keras? The deep neural network API explained"[Online]. Available: <https://www.infoworld.com/article/3336192/what-is-keras-the-deep-neural-network-api-explained.html> [Accessed 29 May 2020]

- [32] M. Makai. “Flask”[Online]. Available:
<https://www.fullstackpython.com/flask.html> [Accessed 29 May 2020]