AROGYA - AN INTELLIGENT AYURVEDIC HERB MANAGEMENT PLATFORM

Final Report for Plant parts Detection and Segmentation of a Complex Image

Project ID: 2020-112

R.A.M.Nithmali

IT17089500

Bachelor of Science (Hons) in Information Technology Specialized in Software Engineering

Department of Software Engineering

09th September 2020

AROGYA - AN INTELLIGENT AYURVEDIC HERB MANAGEMENT PLATFORM

Project ID : 2020-112

R.A.M.Nithmali

IT17089500

Supervisor's name – Mrs. Lokesha Weerasinghe

Department of Information Technology

16th September 2019

Declaration

I declare that this is my own work and this dissertation does not incorporate without acknowledgement any material previously submitted for a degree or diploma in any other university or Institute of higher learning and to the best of my knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgement is made in the text. Also, I hereby grant to Sri Lanka Institute of Information Technology the non-exclusive right to reproduce and distribute my dissertation in whole or part in print, electronic or other medium. I retain the right to use this content in whole or part in future works.

Signature: Date: 09th of September 2020

Signature of the Supervisor: Date: 09th of September 2020

Abstract

Herbal plants are used widely in local medication. Especially in Sri Lanka we have our own set of rare ayurvedic herbs which have been utilized by generations as medicinal treatments for a variety of diseases. But now a day, an ordinary person has little knowledge of these herbs and he could not easily recognize these herbals. And some people want to recognize approximately more info of the plants and some human beings has expert know-how about plants. They need to proportion their understanding about plants with others who preference to know about them. So that sort of people like a social medial platform which only allows to percentage Ayurvedic plants associated things. With these main concerns regarding Ayurveda we propose a system called Arogya is an intelligent herbal management platform which classifies and identifies plant species using automatic visual recognition and tracks the locations of Ayurvedic plants in Sri Lanka through a single unit of mobile application. The app identifies the plant from gallery or photograph of its leaf or digested root. To identify a particular plant accurately, the very first step is that detecting the leaf or digested root correctly, even if it is in the complex background. As a first step of computer based recognition of herbs, an analysis has been made to identify the best method for segmenting object from its background. Several image segmentation techniques are available for detecting the objects based on global and local features of an image. And also object detection is done in real time. When user captures an image with complex background, existing several applications failed to accurately identify the particular objects. Thus, the end result will be inaccurate and those applications take much time for detecting objects when the objects are in complex background. And also they failed to identify overlapping objects, so that gives the end result inaccurately. Thus, we experimented several object detection technologies such as YOLO and marker based watershed algorithm to select the best technology for our application. According to experimental results, the proposed Marker-based Watershed algorithm selected as the best plant parts detection algorithm in a complex background.

Keywords: Ayurvedic Leaf Segmentation and classification, Computer vision, Convolutional Neural Network, Machine learning, transfer learning,

ACKNOWLEDGEMENT

The work described here was carried out as my 4th year research project for the subject Comprehensive Design Analysis Project. The completed final project is the result of combining all the hard work of the group members and the encouragement, support and guidance given by many others. Therefore, it is our duty to express our gratitude to all who gave us the support to complete this major task.

We are deeply indebted to our supervisor Mrs. Lokesha Weerasinghe and our co-supervisor Mrs. Ishara Weerathunge and Lecturers of Sri Lanka Institute of Information Technology whose suggestions, constant encouragement and support in development of this research, particularly for the many stimulating and instructive discussions. We are also extremely grateful to Dr. Darshana Kasthurirathna for helping us by providing necessary information to carry out our research and also I would like to thank senior lecturer and lecture in-charge for Comprehensive Design and Analysis Projects module, Mr. Janaka for the valuable assistance given to complete this project.

Last but not the least, we would like to thank all others whose names are not listed here but have given their utmost encouragement and support in every possible manner.

TABLE OF CONTENT

Declaration i
Abstract ii
ACKNOWLEDGEMENT iii
TABLE OF CONTENT iv
LIST OF FIGURES vi
LIST OF TABLES vii
1. INTRODUCTION 1
1.1 Background Literature 3
1.2 Research Gap 6
1.3 Research Problem 9
1.4 Research Objectives 10
2. METHODOLOGY 12
2.1 Understanding the key pillars of the research domain 12
2.2 Object detection 12
2.3 Research Methodology 12
2.3.1 YOLO (You Only Look Once) 13
2.3.2 Marker Based watershed algorithm 19
2.4 Commercialization aspects 22
2.5 Project Requirements that have been achieved 23
2.5.1 Functional Requirements 23
2.5.2 Non-Functional Requirements 24
2.6 Consideration of the aspects of the system 24
2.6.1 Social aspects 25
2.6.2 Security aspects 25
2.6.3 Ethical aspects 26
2.6.4 Limitations 26
2.7 Testing and Implementation 27
2.7.1 Implementation 27
2.7.2 Testing 39
3. RESULTS AND DISCUSSION 45

3.1 Results 45
3.2 Research findings 51
3.3 Discussion 53
4. CONCLUSION 54
5. REFERENCES 55
6. Appendices 62

LIST OF FIGURES

Figure 2.3.1.1-High level architecture diagram for leaf detection.....13
Figure 2.3.1.1-YOLO makes SxS predictions with B boundary boxes 14
Figure 2.3.1.2-Data Visualization.....16
Figure 2.3.1.3-Data annotation using LabelImg tool 16
Figure 2.3.1.4-YOLO V3 Architecture 17
Figure 2.3.1.5-Import dataset to Roboflow 18
Figure 2.3.1.6-Data augmentation in Roboflow 19
Figure 2.3.2.1-high level architecture for watershed segmentation.....20
Figure 2.3.2.2-Summary of the proposed marker based watershed segmentation technique 21
Figure 2.7.1.1-Android Studio IDE 27
Figure 2.7.1.2-Spyder IDE 28
Figure 2.7.1.3- Dataset training in Google Colab.....30
Figure 2.7.1.4- Import libraries.....30
Figure 2.7.1.5-Import YOLO weight file and configuration file 30
Figure 2.7.1.6-Get out put layer of the image 31
The figure 2.7.1.7 shows the object which I'm going to detect and the set the path which has the test data set. And then get the output Layer which needs to display the final result on the screen.....31
Figure 2.7.1.8-Dataset training in YOLO V3.....31
Figure 2.7.1.9-Object detection with bounding boxes 32
Figure 2.7.1.10-import data to Roboflow.....33
Figure 2.7.1.11-Number of classes 33
Figure 2.7.1.12-Model configuration and architecture 33
Figure 2.7.1.13-Training dataset in YOLO v5 34
Figure 2.7.1.14-End result of the training in YOLO V5.....34
Figure 2.7.1.15-Ground truth training data 34
Figure 2.7.1.16-Ground truth augmented training data.....34
Figure 2.7.1.17-Display inference on all test images 34
Figure 2.7.1.18-Imported packages for watershed algorithm.....35

Figure 2.7.1.19-Extracting green color range area from the background 35
Figure 2.7.1.20-Resize image 36
Figure 2.7.1.21-Remove white, black and blue color from green range mask.....36
Figure 2.7.1.22-Extracting green color range area from the background 37
Figure 2.7.1.23-convert green color mask to grey color mask 37
Figure 2.7.1.24-Check whether the image contains multiple leaves or single leaf 38
Figure 2.7.1.25-Brown color range area for digested root detection 38
Figure 2.7.1.26-Merge overlaps bounding boxes 38
Figure 2.7.2.1-Mobile UIs 44
Figure 2.7.2.1-Visualize ground truth training data.....47
Figure 2.7.2.2-Visualize ground augmented training data 48
Figure 2.7.2.3-Performance Metrics of YOLO V5 48
Figure 2.7.2.4-Leaf in a colored background 49
Figure 2.7.2.5-Leaf in a white background 50
Figure 2.7.2.6-Multiple leaves in a complex background.....50
Figure 2.7.2.7-Multiples leaves in a dark background.....50
Figure 2.7.2.8-Digested root detection in a simple background.....51
Figure 2.7.2.9-Digested root detection in a complex background.....51

LIST OF TABLES

Table 1.2.1-Comparing with existing system 8
Table 2.3.1-Summary of the methodology.....22
Table 2.7.1-Test Cases 42
Table 3.1.1-Compare results of YOLO V3 and YOLO V5 45

1. INTRODUCTION

Ayurveda is an ancient medicine which first appeared about 5000 years ago. It's in fact some of world's oldest health systems and it has demonstrated beneficial effects on many health issues. It helped many people maintain more balanced physical conditions and mental wellbeing and in addressing some severe health issues. There are masses of benefits of the usage of Ayurveda in preference to the use of western medicines. Some of them are completely natural, Positive influence on mental health, prevent inflammations, boosts the metabolism and thus helps us slim down, more affordable and so on. Specially, Sri Lanka is well self-sufficient country with ayurvedic plants.

However, the past 200 years of human activity have contributed to the disappearance of hundreds of plant species. Shielding of plant variety is essential. The key move for further protection is the identification of the plant species. Most of the current identification techniques, however, rely primarily on scientist or specialist knowledge. Therefore, a fully automated identification process for plant species is required. The identification system based on mobile devices has attracted a lot of attention with the attention given to geometric distribution of plants. The first mobile plant species identification application was developed by Kumar et al. [1], namely "Leafsnap".

Every plant on earth has a certain medicinal value according to Ayurveda and therefore it is essential to preserve the plant and recognize its medicinal properties. Studies have shown that the use of too many allopathic medicines can lead to side effects, as it performs many chemical reactions inside the body [30]. A basic truth of Allopathy is that once taken it involves taking another drug to treat the side effects that have arisen because of the previous medication. In general, the consumption process of medicinal products will not end. Allopathic therapies are intended to treat disease symptoms while Ayurveda treat the disease center [2].

Another of Ayurveda's main benefits is that it has no side effects as it is solely herbal, which is important in this most modern period when infectious threats

emerge as a result of changing lifestyles and modified diets. So it is important for every human being to return back to Ayurveda. Almost all general diseases can be cured through Ayurveda using the shrubs and herbs that are around us. Thus returning to Ayurveda is important to every human being. Using the shrubs and herbs around us, almost all general diseases can be cured through Ayurveda. It really is important for everyone to protect the environment, as plants and trees are being destroyed more rapidly in today's world because of rapid urbanization.

In several areas, machine learning has been extensively used for classification and recognition tasks, particularly in the biological fields, with the advancement of science and technology. Methods of computer vision and image processing should fill the gap between the lack of expert taxonomists and future requirements in medicinal plant identification and classification. Leaf, herb, root, and branch characteristics are usually used for classification purposes by the taxonomists [4]. There are existing applications that can identify plants with low predictive accuracy, according to many research performed regarding this problem. However, these applications are based on data sets from foreign plants that do not include valuable herbs and medicinal-grade shrubs.

The most of those researches have been considered shape, color, and textures as the important spatial and morphological features to detect plant leaf [5]. Because the leaves are available at all periods, most researchers have decided that it's the best solution to identify leaves from plants. Taxonomists use the variations in feature characteristics of the leaves of medicinal plants as a tool for identification. A lot of research has been done in this area. But the high level of intra class equivalence in characteristics of shape, color and texture still makes this problem a complicated and unsolved one. Therefore a fully automated system is inevitable at this point in time to properly detect the plant leaf.

Arogya is an herbal management platform which classifies, identifies and tracks the locations of ayurvedic plants in Sri Lanka through a single unit of mobile application. Since it is a kind of information sharing social media hub which can manipulates all four functionalities object detection, classification, location

mapping, summary generation and visualization, in a single component. This report will describe leaf detection and segmentation of a complex background.

1.1 Background Literature

Biodiversity is slowly decreasing across the globe. The current rate of extinction is largely the result of direct and indirect human activities [6]. It is necessary for the future survival of biodiversity to develop accurate knowledge of the identity and geographic distribution of plants [7]. Therefore, for efficient biodiversity research and management, rapid and precise identification of plants is important.

As in the introductory part, Researchers tried a number of methodologies to automatically extract the features and identify the plant leaf. According to previous work done, there are existing applications which can detect leaf with low detection accuracies.

However, existing applications are based on data from foreign plants, which do not include valuable herbs and shrubs of medicinal quality. Additionally, most of these methods make use of the combination of many parameters like color, shape, texture, features etc.

In a manually identification process, botanist use distinctive plant characteristics as identity keys, which might be used sequentially and adaptively to discover plant species. In essence, a person of an identification key's answering a chain of questions about one or more attributes of an unknown plant continuously focusing at the most discriminating traits and narrowing down the set of candidate species. This series of answered questions leads ultimately to the desired species. However, the determination of plant species requires a tremendous botanical expertise. Sometimes botanists themselves species identification is often a tough task. The situation is further exacerbated by means of the increasing scarcity of professional taxonomists [8]. The declining and partially nonexistent taxonomic knowledge within the standard public has been termed “taxonomic crisis” [9].And other hand, it decreases user

satisfaction about the application. Because these applications allow users to identify the plant using traits such as flower structure, leaf shape, flower color or fruit color. Sometimes users would not prefer to go several steps to identify the plant.

To overcome these issues, we are going to use object detections methods. So that user would not be unsatisfied about the product because user just has to take an image of a picture. After capturing the image, using image segmentation we can detect the object with its shape, color and with other features. Because our application gives deepest attention to identify the objects (leaf or digested root) accurately reason is why identifying objects correctly increases the application accuracy, performance and user satisfaction. Otherwise, it will give incorrect results at the end.

Shape is the most common feature, whether it be manual or automatic plant identification, used in plant identification [10]. So that we are going to use image segmentation techniques rather than using object detection technique. The reason is that Object detection [11] builds a bounding box corresponding to each class in the image. But for each object in the image, image segmentation creates a pixel wise mask. This technique provides us a much more granular idea of the object(s) in the image. So we can get clear idea of the shape of leaf, flower and etc.

Researchers have planned and suggested several segmentation techniques, but there is no general technique that can be used for all images. However, Keri Wood [12] suggested that good image segmentation need to meet the subsequent requirements:

Every pixel in the image must belong to a place and each place must be homogeneous with admire to a chosen feature, which might be syntactic e.g. color, depth or texture or the function primarily based on semantic interpretation.

1) Every region have to be linked and non- overlapping i.e. any pixels in a selected region need to be connected by a line that does depart the region.
2) It should not be viable to merge two adjoining regions to shape a single homogeneous region.

3) These characteristics also include in our segmentation process in order to get successful results.

Color images can increase the quality of segmentation but complexity of the problem is also increased. Segmentation of a colored image having different varieties of texture areas is a difficult problem, in particular if an exact texture subject is to be computed and a choice is to be made regarding optimum number of segments in the image. The problem will become further complicated if the image includes similar and/or "non-stationary texture fields. Each pixel of colored images is depicted using three component values that is red, green and blue and as such these are more complex as a long way as segmentation is concerned, than gray scale images which have a single intensity value for a pixel. Colored image segmentation can clear up many issues in medical imaging, bioinformatics, and material sciences [6]. It leads to lost lots of information of the leaf, flower or fruits. The lots of existing plant identifications applications regenerate the user entered pictures into greyscale images due to complexness of the color images. It results in lost voluminous info of the leaf, flower or fruits.

K.Deepak and A.N.Vinoth proposed to develop an application to identify plant species on android platform [13]. They used edge detection as there segmentation technique. This proposed method contain several major drawbacks. It converts the image into gray scale image. It ends with lost plenty of information approximately the detected object. It will be a major disadvantage when the detected object goes to classification model. And also in order to use this technique, there should be better contrast between objects. Otherwise, this technique isn't going to work and it doesn't detect the object accurately. It leads the application failed among the users. This indicates how the importance of detecting object accurately.

Lin-Hai Ma, Zhong-Qiu Zhao and Jing Wang designed plant identification system called APLeafis [14].They used threshold segmentation for the segmentation process. It is the simplest segmentation method. This application contains several drawbacks. In this application, one leaf image can be either a digital image from the existing leaf image database or a picture collected by a camera. The picture should be a single leaf

placed on a light and untextured background without other clutter. Those facts reduce the user satisfaction because before taking a picture, user has to make the background that they mentioned about otherwise this application not going to detect the particular leaf from the image. If the application doesn't select it accurately other all processing things are not going to work. The application is failed. Because now a days, modern people always do and buy things which their lives make easier.

Desta Sandya Prasvita and Yeni Herdiyeni proposed MedLeaf as a new mobile application for medicinal plants identification based on a leaf image [15]. Neeraj Kumar, Peter N. Belhumeur, Arijit Biswas and David W. Jacobs proposed a system called Leafsnap which is a mobile app that helps users identify trees from photographs of their leaves [16]. Above two authors proposed systems they identify the plant only using the leaf. The major drawback of that kind of system is that sometimes some leaves have similar features but they are different plants. Because Ayurvedic plants are used by people as their medicine. Thus, it affects to human health also. We have to identify those plants separately. So that our proposed system detect not only leaf but also flower and fruit also. It leads to get best accurate result.

As mentioned above several problems occur when detecting the objects (leaf, flower or fruit). Therefore our proposed method is going to use the most suitable image segmentation techniques to detect the objects even if the image background is complex. In computer vision, we cannot directly apply an algorithm by believing that matches well with our application. We have to do several experiments and get outputs of those and then we should perform some evaluations on final outcomes to select best approach.

1.2 Research Gap

Currently there is no existing system which identify the objects (leaf, digested root) if they are in a complex background. Although there are several systems available to detect the objects (leaf, digested root), detection of them in a

complex background is questionable. Some of the existing system takes much time for processing the result. That frustrates the user, if the app takes much time to load the results. However, some existing applications will not be capable of detecting the objects of the captured photograph of the plant contains multiple leaves. Therefore, those applications give instructions to user to place a single leaf on a light and untextured background without other clutter. At the moment user just wants to capture an image and get the result of the application, but if the applications give those kind of instructions, user might be frustrated. Moreover, there are several existing systems which can identify the plant correctly, but the user has to take the leaf in front of the white background. Assume at the instance, user will not be able to find that kind of background and he or she just needs to know about the plant. Thus, those sort of applications, frustrate the user and the user will uninstalled the application because it clearly doesn't assist the user when the user wants it. It leads to failed whole product also. When user capture an image with overlap leaves, existing methods are not going to work and show wrong result at the end reason is why at the begging those applications will not be capable of detecting the plant leaf or digested root accurately. Existing Systems used threshold, edge detection techniques for segmentation process in their applications. Those are simplest techniques that will not work above given situations. Accurately identifying the object is more important because classification and other processing things are going to use segmented object result. Thus, we can get granular idea how much of an impact that system has, if it doesn't identify the object correctly. Several current applications identify the plant only using plant's leaf, thus they only detect the leaf as their object. Sometime only detecting the leaf is not sufficient because in the classification phase if this segmented object matches to several plants, there may be no manner to identify the exact plant accurately.

Table 1.2.1-Comparing with existing system

References	Identify the objects in a complex background	Accuracy and performance	Experiment several segmentation techniques to select the best approach
Neeraj Kumar, Peter N Belhumeur, Arijit Biswas,David W Jacobs, W John Kress, Ida C Lopez, João VB Soares, "Leafsnap: A computer vision system for automatic plant species identification" [16]	✗	Medium	✗
K.Deepak, A.N.Vinoth," Leaf Detection Application For Android Operating System "[13]	✗	Low	✗
Research Gate, "MedLeaf: Mobile Application For Medicinal Plant	✗	Low	✗

Identification Based on Leaf Image" [15].			
Arogya our proposed System	✓	✓	✓

1.3 Research Problem

As a solution to the plant leaf or digested root identification, we can use any of the plant identifications systems available. However, none of these systems present a reliable, accurate and a high performance way to identify objects in a complex background. Existing systems works well when the object placed in a simple and light background. Moreover, most of the available systems have low accuracy. So, finding a way to optimize the results is more important.

But the real problem occurs when it comes to segmentation process. Detecting objects are difficult when the background and the objects do not have better contrast between them. Thresholding and edge based are segmentation techniques which identify the objects using the contrast between them. As an example, they convert the whole image into grayscale image and then check the difference of grey color values. If the image has significant contrast among the foreground and background, the techniques identify edges of the objects. But if not it is not going to detect the edges, Thus it change the segmented object shape. Shape is the most popular feature used in plant identification [6]. The segmented object is wrong. Thus, Final outcomes accuracy is much low.

1.4 Research Objectives

1.4.1 Main Objective

To develop an android mobile application that enables users to identify and classify a group of selected important Ayurvedic plant species in Sri Lanka based on photographs of the plant's leaf and other parts like digested root taken with a smartphone , detection of the leaf or digested root of the plant should be done accurately. Therefore, in here main purpose is to select the most accurate and the highest performance segmentation technique for detect the leaf or digested root from the input image even if the background of the image is complex.

1.4.2 Specific Objectives

1. A newly captured, prepared and annotated dataset of Sri Lankan Ayurvedic plants will be trained on YOLO object detection methods.

2. The noisy image set of medical leaves / digested roots will be analyzed using YOLO object detection technique.

3. Select best object detection technique among YOLO object detection method and Marker based watershed segmentation, which has highest performance, speed and accuracy of the detection. It will be applied effectively for our mobile application.

4. Remove the segmented objects from the background.

    - If the image contains the multiples leaves, all leaves will be detected. These detected all leaves will be segmented and send to the

classification process in order to do a best classification using multiple leaves.

5. The finalized model will be used to build our mobile application in the Android platform with TensorFlow-Lite which can detect the plant leaf or digest root from the captured image or gallery of the mobile.

- Ayurvedic plant leaf or digest root detection has done in real time and the time, it takes to detect the plant parts is less than the existing applications. User can takes images anywhere, so user does not need to worry about the background.

2. METHODOLOGY

2.1 Understanding the key pillars of the research domain

The object detections techniques, Data augmentation and Data annotation are the main key pillars of this research part.

2.2 Object detection

Object detection is one of the classic computer vision problems where you function to distinguish what and where-exactly what objects are within a given image and also where they are within the image. The object detection problem is more complicated than classification, which can also identify objects but does not indicate where the object is located in the image. Classification also doesn't work on photographs that contain more than one thing.

2.3 Research Methodology

The system provides a way to upload an image of the plant to be identified. Detecting the objects accurately is more important because it helps the classification process to do a better job using a leaf or digested root. In addition, detecting plant leaf or digested root from complex and noisy backgrounds should also be done accurately, especially in this scenario. Therefore, much attention was given to detect the specific object accurately at the beginning. In order to achieve this objective, it was experimented with YOLO and marker-based watershed algorithms and the most accurate one was selected based on the results. At the beginning backend application is developed using

python and to connect with the front end mobile part. Front end application is developed using Android.

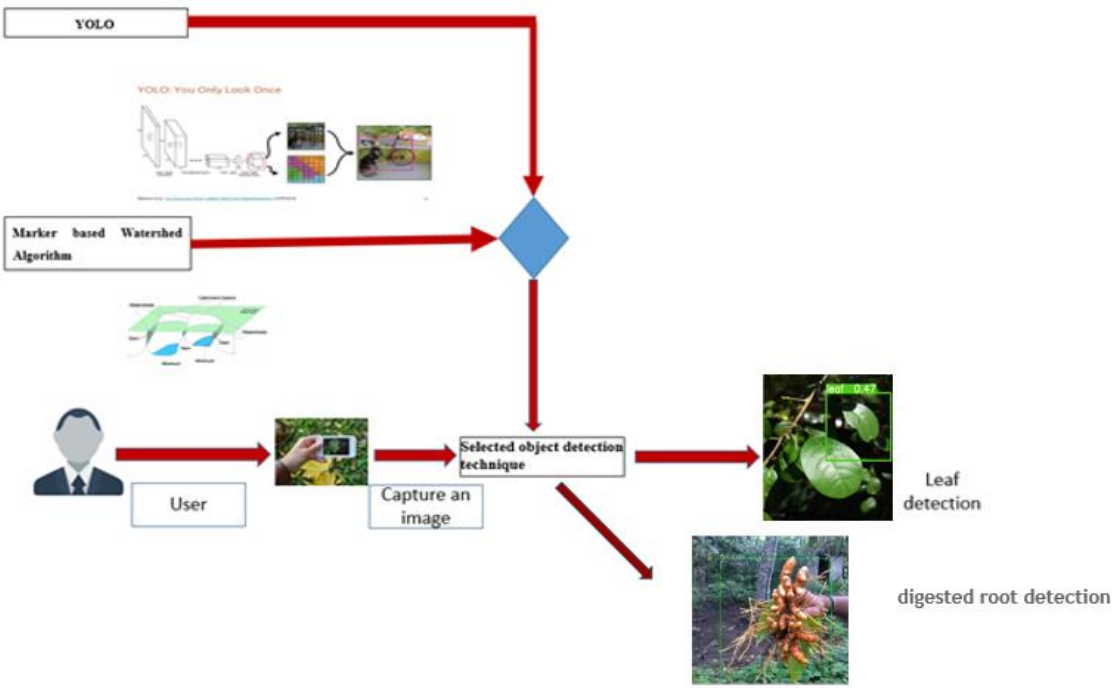


Figure 2.3.1.1-High level architecture diagram for leaf detection

2.3.1 YOLO (You Only Look Once)

YOLO is common as it achieves high precision while being able to run in real-time as well. The algorithm "only looks once" at the image in the sense that it requires only one forward propagation pass through the neural network to make predictions. After non-max suppression (which makes sure the object detection algorithm only detects each object once), it then outputs recognized objects together with the bounding boxes. YOLO divides the input image into an S×S grid. Each grid cell predicts only one object. Each grid cell predicts only one object and each grid cell predicts a fixed number of boundary boxes.

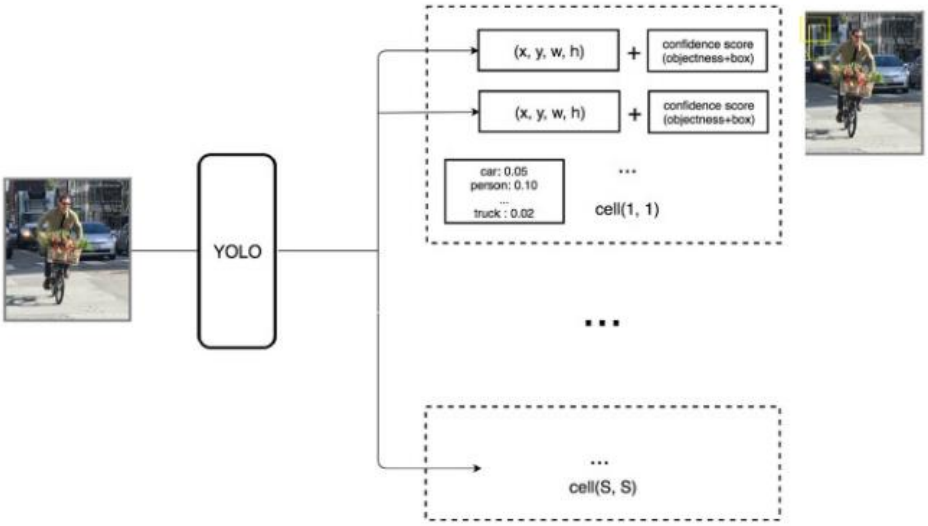


Figure 2.3.1.1-YOLO makes SxS predictions with B boundary boxes

The major concept of YOLO is to build a CNN network to predict a (7, 7, 30) tensor. YOLO has 24 convolutional layers followed by 2 fully connected layers (FC) .It uses a CNN network to reduce the spatial dimension to 7×7 with 1024 output channels at each location. YOLO performs a linear regression using two fully connected layers to make 7×7×2 boundary box predictions (the middle picture below). To make a final prediction, we keep those with high box confidence scores (greater than 0.25) as our final predictions.

With YOLO, a single CNN simultaneously predicts multiple bounding boxes and class probabilities for those boxes. YOLO trains on full images and directly optimizes detection performance. The YOLO network consists of three main pieces.

1) **Backbone -** A convolutional neural network that aggregates and forms image features at different granularities.

2) **Neck -** A series of layers to mix and combine image features to pass them forward to prediction.

3) **Head -** Consumes features from the neck and takes box and class prediction steps.

Dataset

The first important step, as with any deep learning step, is to prepare the dataset. The dataset is the fuel that runs every model of deep learning**.** Our Ayurvedic dataset was made using captured photos of the tree leaves/digested roots, web images, etc.). After the process of data annotation, labelling into 2 classes (leaf or digested root) for the purpose of annotation and totally pre-processing; the prepared dataset will be divided into 3 parts as:

1. Training dataset -70% of data for training the model
    - Training set: Data sample used for model fit. The real data set which we use to train the model (for a Neural Network, weights and biases). From this knowledge, the model sees and learns.

2. 15% of data for validating the model – Validation dataset

    - Validation set: The sample of data used to provide an impartial model assessment fits into the training dataset while hyper parameters are adjusted to the model. As knowledge on the validation dataset is integrated into the model setup, the assessment becomes more biased. Often called the Dev set or the Creation set, the validation set is also known. This makes sense because during the model's "growth" period, this dataset assists.

3. 15% of data for testing the model – Test dataset.

    - Test set: The sample of data used to provide an impartial assessment of a final model fits into the dataset of the training. The Test dataset provides the gold standard used for the model evaluation. It is only used once a model (using the trains and validation sets) is fully educated. In general, the test set is what is used to compare competing models. The validation set is often used as the test set, but it is not good practice. In general, the test collection is well selected. It contains

carefully sampled data which, when used in the real world, spans the various classes that the model will face.

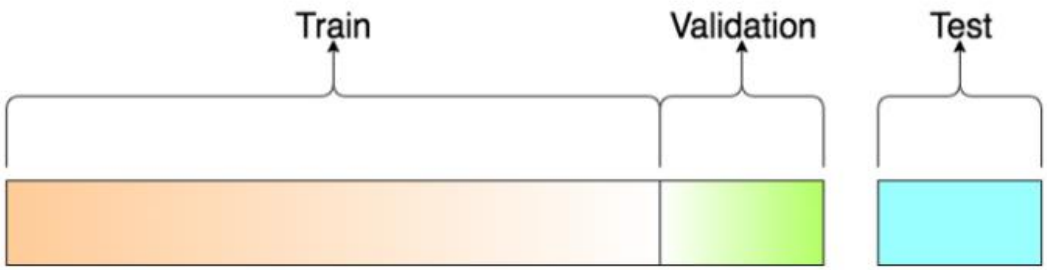


Figure 2.3.1.2-Data Visualization

Source: https://towardsdatascience.com/train-validation-and-test-sets-72cb40cba9e7

Data Annotation

In here I trained the **YOLO v3** version and **YOLO v5** version which are the latest versions of the YOLO algorithm using our annotated Ayurvedic dataset. To use the YOLO algorithm, first we annotate our dataset using LabelImg tool around 2000 images for 5 categories of plant species.

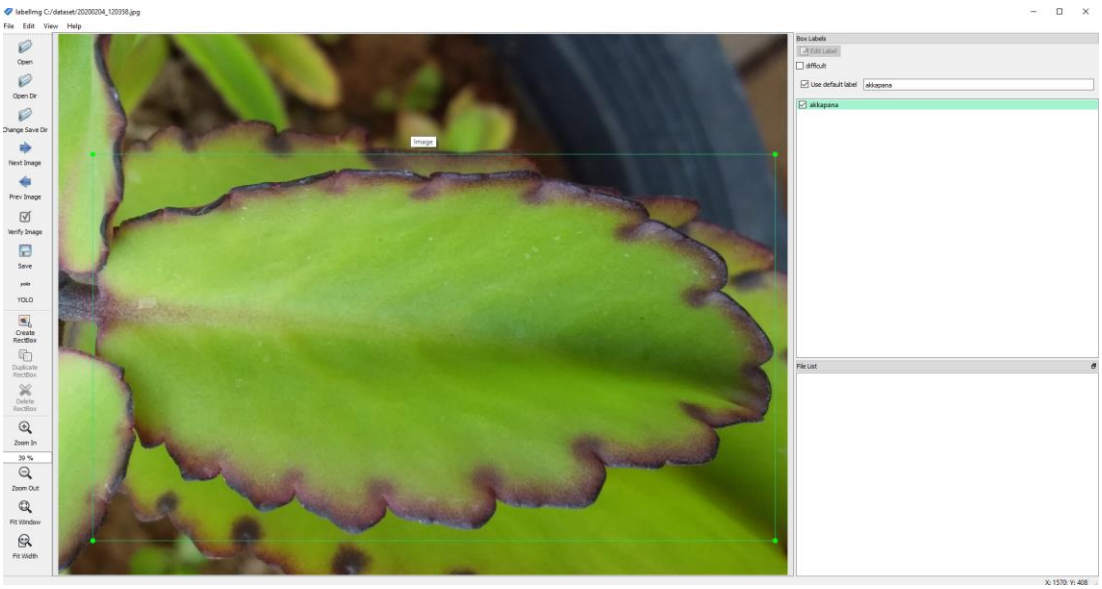


Figure 2.3.1.3-Data annotation using LabelImg tool

YOLO v3

YOLO is a completely convolutionary network, and by applying a 1 x 1 kernel on a feature map, its eventual output is generated.

The detection is achieved in YOLO v3 by applying 1 x 1 detection kernels to function maps of three different sizes at three different network locations. The most critical aspect of v3 is that it allows three distinct scales of detection [17].

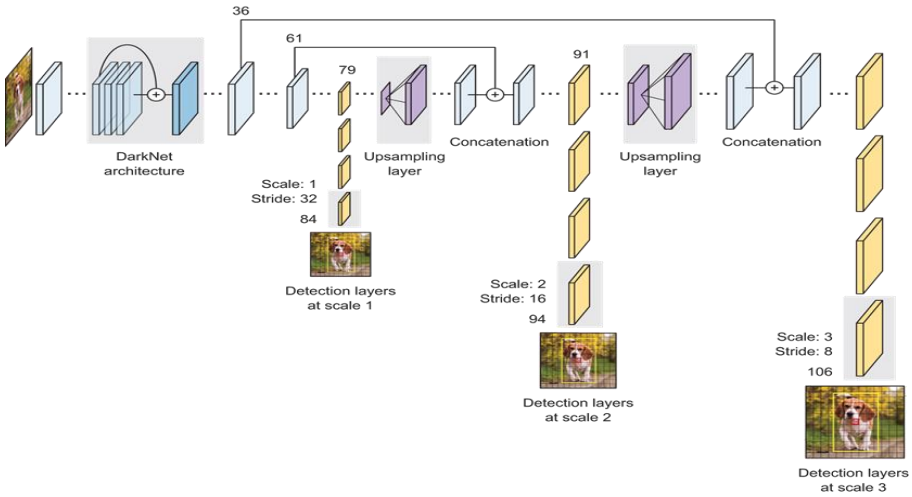


Figure 2.3.1.4-YOLO V3 Architecture

Source: https://towardsdatascience.com/yolo-v3-object-detection-53fb7d3bfe6b

On 3 different scales, YOLOv3 makes predictions:

1. In the last map layer of functions.
2. It then goes back back 2 layers and upsamples it by 2 layers. YOLOv3 then takes a higher resolution feature map and uses element-wise addition to combine it with the sampled feature map. YOLOv3 applies convolutionary filters to make the second set of predictions on the combined map.
3. Repeat 2 again so that the resulting map layer of features has strong high-level structure information and spatial information on object positions with good resolution.

YOLO V5

YOLOv5 is a recent release from the YOLO model family. Initially YOLO was implemented as the first model of object detection to merge bounding box prediction and object classification into a single end-to - end differentiable networkIt was written in a framework called Darknet and is preserved. The first of the YOLO models to be written in the PyTorch framework is YOLOv5 and it is much lighter and simpler to use. YOLOv5 is written in the Ultralytics PyTorch framework, which is very intuitive to use and inferences very fast. In fact, we and many others would often translate YOLOv3 and YOLOv4 Darknet weights to the Ultralytics PyTorch weights in order to inference faster with a lighter library. YOLOv5 is definitely easier to use and, based on our initial runs, it is very powerful on custom data. Therefore, YOLO V5 has more performance than YOLO V3.

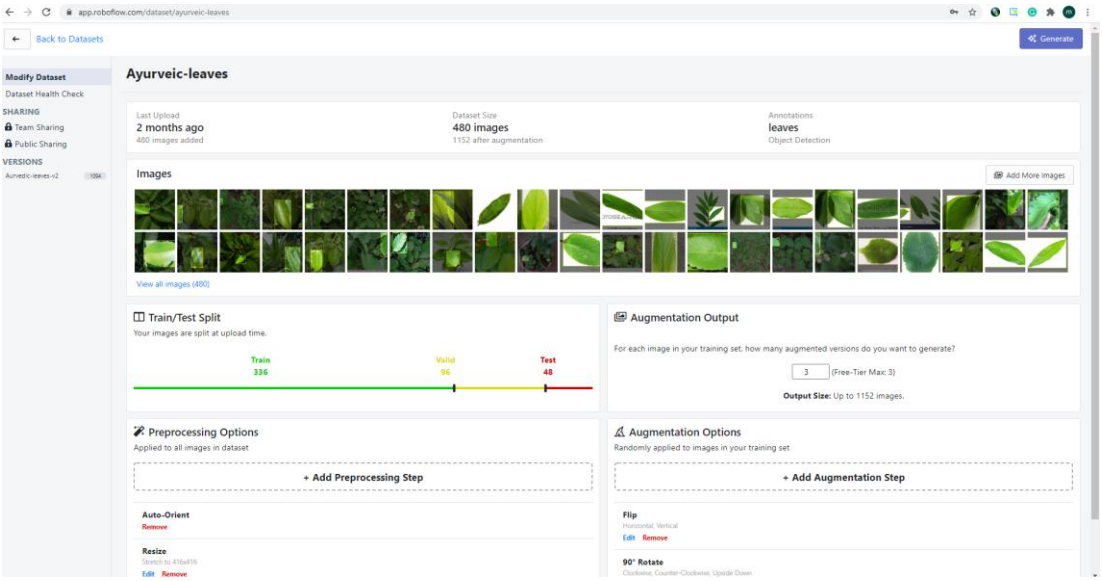


Figure 2.3.1.5-Import dataset to Roboflow

According to Figure 2.3.1.5, it is easier to import the dataset and it automatically removed problematic annotations of our annotated dataset. As shown in above figure, adding preprocessing steps and adding augmented steps are easier inside Roboflow.

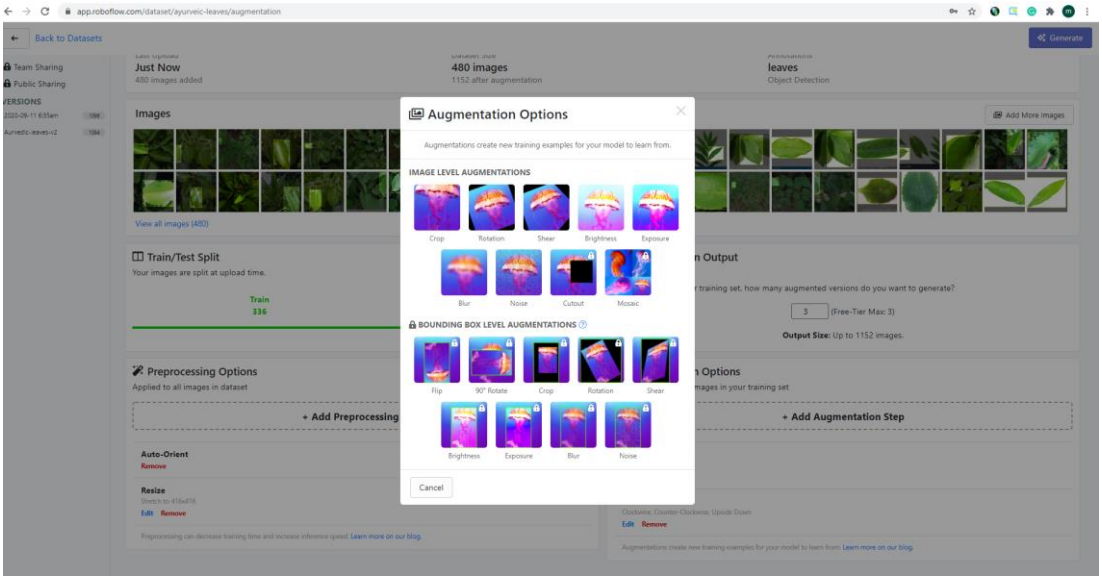


Figure 2.3.1.6-Data augmentation in Roboflow

As shown above figure 2.3.1.6, there are several augmentation options are available. Among them I added Flip and 90° Rotate augmentation options to the dataset. When uploading the dataset, pop up is shown to split the data into train, valid and test data.

Used the "YOLOv5 PyTorch" export format. That is the Ultralytics implementation calls for a YAML file defining where your training and test data is. The Roboflow export also writes this format for us.

2.3.2 Marker Based watershed algorithm

Watershed segmentation is region-based method that is regarded as a topological landscape with ridges and valleys. The elevation values of the landscape are typically defined by the gray values of the respective pixels or their gradient magnitude. Due to noise, direct application of the watershed segmentation caused over-segmentation. Markers were used as an approach to control over segmentation. One of the most complicated operations of image processing was to distinguish touching objects in image. The watershed segmentation is often applied to this issue. This image segmentation algorithm proposed here was capable of segmenting the images with

minimum limitations to under and over segmentation. Segmentation of the marker-controlled watershed has been shown to be a robust and versatile method for segmenting artifacts with closed contours, where the boundaries are represented as ridges.

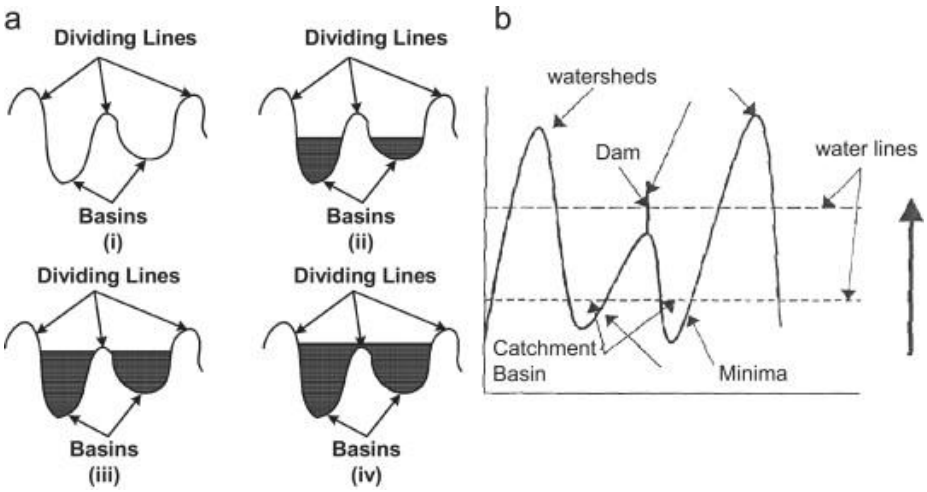


Figure 2.3.2.1-high level architecture for watershed segmentation

To identify a leaf, vein structure takes a prominent place. Vein is a vascular bundle within a leaf that frequently spreads from the middle of the leaf to the edge [3]. In this proposed method introduces a precise and fully automated image segmentation for complex context images of medicinal plant leaves. As the first step, green color range of the image is filtered. To get the wide green color range, at the begging of the process image is converted from RGB to HSV color format. By strictly applying the objective rule that the veins are linear and the accuracy of gradient angles of adjacent venous points is relatively high, the veins in the gradient magnitude images are further improved by the normal gradient angle deviation to obtain the vein enhancement image. Based on this image, a binary image taking the veins as the foreground is obtained using the OTSU process, and the main veins are identified from it. In the vein enhancement diagram, fine veins are found in other areas outside the main veins, and then attached to the main veins. Then a foreground marker image is obtained which targets the veins. In each component image of the color image, the background

marker image is segmented using the OTSU process, and then screened after a certain ratio is determined. The marker-controlled watershed approach is applied to obtain the outcome of binary segmentation, based on the foreground markers and the background markers. To identify the digested root from the image, yellow and brown color range area was extracted and then the same procedure was used which followed in leaf detection.

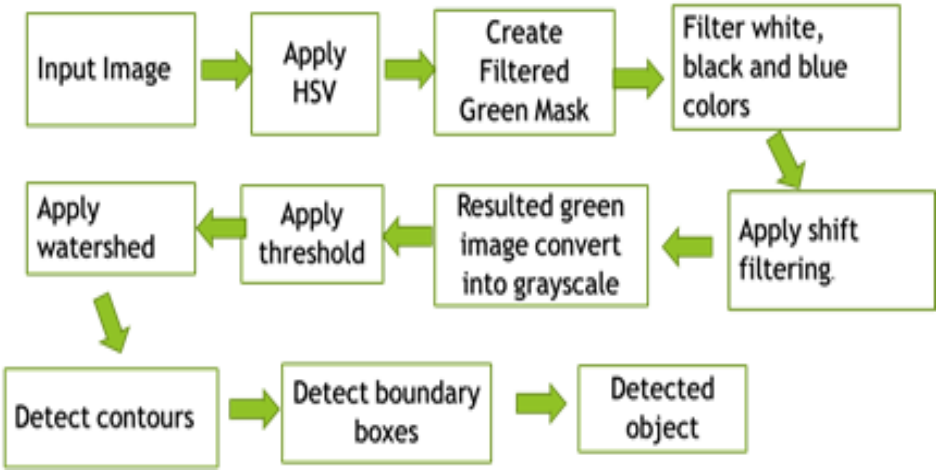


Figure 2.3.2.2-Summary of the proposed marker based watershed segmentation technique

Main purpose of the component is to detect the object (leaf) in real time even if it is in a complex background. This proposed marker-based watershed segmentation provided several advantages such as it required low computation time, and provided closed contours, the boundaries of the resulting regions always correspond to contours which appear in the image as obvious contours of objects, fast and simple. This proposed algorithm works well in our application as described and this is used as the finalized model.

Table 2.3.1-Summary of the methodology

Research	**Research Work**	**Methods and Algorithms to be used**	**Expected Accuracy**	**Remarks**
Leaf detection from the complex background	Use different object detection techniques on collected dataset.	YOLO architectures (Such as YOLO v3, YOLO v4, VGG-16, marker based watershed algorithm.) will be used to choose the best technique.	Higher Accuracy will be expected	Several existing algorithms will be analyzed, and accuracy will be tested in order to select the suitable algorithms to detect the leaf from the image accurately.

2.4 Commercialization aspects

7.1 Target Audience

- People who use ayurvedic treatment

• Researchers in the field of botany, medicine, chemical structure analysis, agriculture, ayurvedic medicinal practitioners, forest department officials, those who are involved in the preparation of ayurvedic medicines and others who are concerned with plant studies.

• Doctors, Students, locals and foreigners

• Ayurvedic plant sellers

7.2 Market Space

• No age limitations for the users

• No need of advance computer literacy

• No need of advance knowledge in Ayurveda field

7.3 Revenue Earning

• Through subscription fee

• Revenue via additional services

2.5 Project Requirements that have been achieved

2.5.1 Functional Requirements

1. Detection of the leaf from the input image even if image contains the complex background.

2. Selecting the best object detection technique with comparing several object detection techniques has been achieved.

3. Retraining the original model using new datasets to get the Maximum usage.

4. Request to re-train the existing model with new datasets by premium uses can be done.

5. Comparison of the accuracies and performance of these models with justification has been achieved.

2.5.2 Non-Functional Requirements

1. Results are given as fast as possible.

2. User friendly and attractive interfaces have been provided in order to increase customer satisfaction.

3. Feasible in order to understand the functionalities of the app.

4. High usability of the application.

5. High security of the application

6. Adding new functionality and updating the existing functionality can be done.

2.6 Consideration of the aspects of the system

Standards: We followed coding standards when doing our individual coding parts. Coding standards tell developers how they should write their codes. All the group members used object-oriented concepts to maintain coding standards. Inside the code, we commented important things. When writing reports and referencing, we followed IEEE format.

2.6.1 Social aspects

Plants are an important part of our environment, and Sri Lanka has a long tradition of using medicinal plants. The use of traditional medicine has diminished to a large degree since the introduction of modern allopathic medicine. However, in recent years, Ayurvedic plant have made a comeback for a variety of reasons like they are inexpensive, nontoxic and does not impact any side effect. There are various forms of Ayuvedic plant species present on earth, but the plant is very hard to classify. Scientists and urban people remain ignorant of the substantial knowledge gained by the villagers and tribals on plant medicine. This kind of knowledge is usually handed down through generations. Via the principles of machine learning, pattern recognition and computer vision, our immediate concern is to maintain this knowledge in digital form and build a smart user friendly mobile application. This mobile application can be used by any person who is not aware about but keen to experience Ayurveda medication worldwide. Researchers, taxonomists, agriculture, people who used ayurvedic medicines and treatment and many more can use this mobile application. They just need to know to use a smart mobile phone. So, this would be very beneficial for the whole society which would do a great service in uplifting our ancient Ayurveda, which is anyway better than modern medicine due to its naturality.

2.6.2 Security aspects

In order to use our mobile application, first user should register to the application. By only giving correct credentials, Users allow to log into the system. Thus, Security of

the application is high as well as all user credentials were stored in firebase with high security rules.

2.6.3 Ethical aspects

This application does not cause the user or his / her mobile to be affected, hurt, or disabled. In the implementation of this mobile application, no unethical actions, laws or values have been found. This does not, however, supersede the position of a plant taxonomist or an Ayurveda doctor. But it allows any person without any background knowledge of Ayurveda to categorize and hopefully understand the medicinal value of a particular Ayurveda plant. For almost all individuals who need to experience Ayurveda, this would be more of a learning tool.

2.6.4 Limitations

- Currently this application is only in English. In addition, this can be extended in native languages such as Sinhala, Tamil and in some of the other chosen languages.
- Although the system is designed exclusively as a mobile application, web apps with the same features and content will later be improvised.
- Currently this mobile application is developed only for Android users, so have to consider about other mobile platforms and operating systems as well.
- If the consumer starts having questions and clarifications about this method, this software could be supported with the aid of Ayurveda doctors in consultation.
- This mobile application is currently designed exclusively for Android users, but other mobile devices and operating systems need to be considered as well.

2.7 Testing and Implementation

2.7.1 Implementation

Arogya is a mobile application which is divided into sub sections and developed individual component wise by the Arogya research team members. Developing individual components includes designing, analyzing, coding and unit testing etc. Server-side implementation and all back-end related services implemented with the use of python and front-end is an android application.

Software interfaces

Since developing a mobile application, below indicated software to use for development purposes.

- Android studio

  This IDE is used for front-end development of the application.

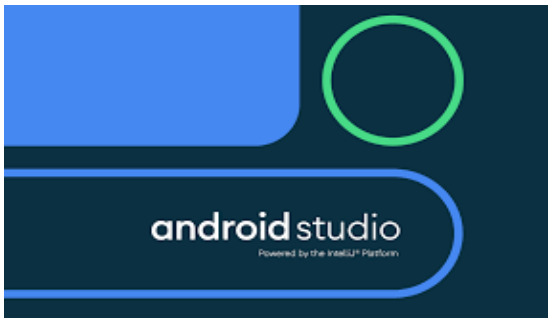


Figure 2.7.1.1-Android Studio IDE

- Spyder

  This IDE is used for front-end development of the application.

Figure 2.7.1.2-Spyder IDE

Used Technologies

- **Python 3.6**
- **OpenCV**
- **Keras**
- **TensorFlow**
- **TensorFlow Lite**
- **Android/Firebase for mobile**
- **Google Colab**

From the starting point of the project, we followed an agile model of development until the end of the project. During the implementation period, we used Github for the version controlling of our application. Clearly identified features and independent development of them in different branches allowed the features to be integrated to the main system successfully without major merge conflicts.

Following code snippet indicates the implementations of the research component.

YOLO (You Only Looks Once)

Reasons of using OpenCV for YOLO V3

1. Fast integration with an OpenCV program: If your program already uses OpenCV and you just want to use YOLOv3, you don't have to worry about compiling and creating the extra Darknet code..

2. The Processor version of OpenCV is 9x faster: the DNN module Device implementation of OpenCV is incredibly fast. For example, Darknet when used with OpenMP takes around 2 seconds on a CPU for inference on a single image. The implementation of OpenCV, by comparison, runs in just 0.22 seconds! Study the table below.

3. Support for Python: Darknet is written in C, and officially does not support Python. For comparison, OpenCV does. There are python ports available for Darknet though..

To load the YOLO V3, we need to have YOLO V3 training file and the YOLO V3 Configuration file. We can get the configuration file from the Darknet (framework used to run and train YOLO) repository. We use the free server provided by Google colab to train the image dataser. Google Colab is a free Google service that allows you to run python scripts and use machine learning libraries using their powerful hardware. To obtain the weight file we should train our model using our Ayurvedic dataset. The only downside is that it can be used for 12 hours in a row, from which it will be disconnected and our files will be removed. Therefore, I connected it with my Google drive. Thus, all files are saved if it is disconnected. It takes more than five hours to train the model using our custom dataset.

After setup Google Colab, start the training process. Output is given below.

Figure 2.7.1.3- Dataset training in Google Colab

It takes more than five hours to train our custom dataset. After completed the training, I got the weight file from my Google drive and configuration file from the Darknet.

```
import cv2
import numpy as np
import glob
import random
```

Figure 2.7.1.4- Import libraries

OpenCV is the immense open-source library for computer vision , machine learning, and image processing, and it now plays an essential role in real-time activity in today's systems. The glob module finds all the pathnames matching a specified pattern according to the rules used by the Unix shell, although results are returned in arbitrary order. Numpy is a general-purpose array-processing package.

```
net = cv2.dnn.readNet("yolov3_training_last.weights", "yolov3_testing.cfg")
```

Figure 2.7.1.5-Import YOLO weight file and configuration file

Load YOLO using weight and configuration file.

```
# Name custom object
classes = ["leaf"]

# Images path
images_path = glob.glob(r"C:\Users\ASUS\Desktop\test data\*.jpg");

layer_names = net.getLayerNames()
output_layers = [layer_names[i[0] - 1] for i in net.getUnconnectedOutLayers()]
colors = np.random.uniform(0, 255, size=(len(classes), 3))
```

Figure 2.7.1.6-Get out put layer of the image

The figure 2.7.1.7 shows the object which I'm going to detect and the set the path which has the test data set. And then get the output Layer which needs to display the final result on the screen.

```
random.shuffle(images_path)
```

Figure 2.7.1.8-Dataset training in YOLO V3

Figure 2.7.1.8 depicts recognizing the order of the images. Randomly, it takes the images from the test data folder.

```
# Showing informations on the screen
class_ids = []
confidences = []
boxes = []
for out in outs:
    for detection in out:
        scores = detection[5:]
        class_id = np.argmax(scores)
        confidence = scores[class_id]
        if confidence > 0.5:
            # Object detected
            print(class_id)
            center_x = int(detection[0] * width)
            center_y = int(detection[1] * height)
            w = int(detection[2] * width)
            h = int(detection[3] * height)

            # Rectangle coordinates
            x = int(center_x - w / 2)
            y = int(center_y - h / 2)

            boxes.append([x, y, w, h])
            confidences.append(float(confidence))
            class_ids.append(class_id)

indexes = cv2.dnn.NMSBoxes(boxes, confidences, 0.5, 0.4)
print(indexes)
font = cv2.FONT_HERSHEY_PLAIN
for i in range(len(boxes)):
    if i in indexes:
        x, y, w, h = boxes[i]
        label = str(classes[class_ids[i]])
        confidence = confidences[i]
        color = colors[class_ids[i]]
        cv2.rectangle(img, (x, y), (x + w, y + h), color, 2)
        cv2.putText(img, label +" "+ str(round(confidence,2)), (x, y + 30), font, 3, color, 2)

cv2.imshow("Image", img)
key = cv2.waitKey(0)

2.destroyAllWindows()
```

Figure 2.7.1.9-Object detection with bounding boxes

Figure 2.7.1.9 shows object detection with the rectangle bounding boxes and the confidence of the detected objects.

YOLO v5

Used the "YOLOv5 PyTorch" export format. That is the Ultralytics implementation calls for a YAML file defining where your training and test data is. The Roboflow export also writes this format for us.

```
%cd /content
!curl -L "https://app.roboflow.com/ds/3E8a5KsNyS?key=7KrOlFKurY" > roboflow.zip; unzip roboflow.zip; rm roboflow.zip
```

Figure 2.7.1.10-import data to Roboflow

This figure 2.7.1.10 shows the api key of exported Ayurvedic dataset from the roboflow.

```
train: ../train/images
val: ../valid/images

nc: 1
']
```

Figure 2.7.1.11-Number of classes

Figure 2.7.1.11 shows the YAML file Roboflow wrote to load into this notebook with our data.

```
# define number of classes based on YAML
import yaml
with open("data.yaml", 'r') as stream:
    num_classes = str(yaml.safe_load(stream)['nc'])
```

```
#this is the model configuration we will use for our tutorial
%cat /content/yolov5/models/yolov5s.yaml
```

```
#customize iPython writefile so we can write variables
from IPython.core.magic import register_line_cell_magic

@register_line_cell_magic
def writetemplate(line, cell):
    with open(line, 'w') as f:
        f.write(cell.format(**globals()))
```

```
%%writetemplate /content/yolov5/models/custom_yolov5s.yaml

# parameters
nc: {num_classes}  # number of classes
depth_multiple: 0.33  # model depth multiple
width_multiple: 0.50  # layer channel multiple

# anchors
anchors:
  - [10,13, 16,30, 33,23]  # P3/8
  - [30,61, 62,45, 59,119]  # P4/16
  - [116,90, 156,198, 373,326]  # P5/32
```

Figure 2.7.1.12-Model configuration and architecture

Figure 2.7.1.12 shows the model configuration and the architecture.

```
# train yolov5s on custom data for 100 epochs
# time its performance
%%time
%cd /content/yolov5/
!python train.py --img 416 --batch 16 --epochs 100 --data '../data.yaml' --cfg ./models/custom_yolov5s.yaml --weights '' --name yolov5s_results  --cache
```

Figure 2.7.1.13-Training dataset in YOLO v5

Figure 2.7.1.13 code describes training yolov5 on our dataset for 100 epochs and at the end it shows the wall time and the cpu times. data.yaml and custom_yolov5s.yaml files used to train.

```
     Epoch   gpu_mem       GIoU       obj       cls     total   targets  img_size
     97/99     1.54G   0.03983   0.04971         0   0.08954        28       416: 100% 59/59 [00:10<00:00,  5.84it/s]
               Class     Images     Targets          P          R      mAP@.5  mAP@.5:.95: 100% 6/6 [00:01<00:00,  5.34it/s]
                 all         96         100      0.225       0.37      0.196      0.0804

     Epoch   gpu_mem       GIoU       obj       cls     total   targets  img_size
     98/99     1.54G   0.03866   0.05101         0   0.08967        31       416: 100% 59/59 [00:09<00:00,  5.98it/s]
               Class     Images     Targets          P          R      mAP@.5  mAP@.5:.95: 100% 6/6 [00:01<00:00,  5.25it/s]
                 all         96         100      0.215       0.33      0.172      0.0691

     Epoch   gpu_mem       GIoU       obj       cls     total   targets  img_size
     99/99     1.54G   0.03979   0.05015         0   0.08994        36       416: 100% 59/59 [00:09<00:00,  5.94it/s]
               Class     Images     Targets          P          R      mAP@.5  mAP@.5:.95: 100% 6/6 [00:01<00:00,  3.42it/s]
                 all         96         100      0.189       0.37      0.167      0.0642
Optimizer stripped from runs/exp0_yolov5s_results/weights/last.pt, 14.7MB
Optimizer stripped from runs/exp0_yolov5s_results/weights/best.pt, 14.7MB
100 epochs completed in 0.331 hours.

CPU times: user 7.91 s, sys: 1.58 s, total: 9.49 s
Wall time: 20min 18s
```

Figure 2.7.1.14-End result of the training in YOLO V5

Figure 2.7.1.14 shows end result of the training YOLOv5

```
# first, display our ground truth data
print("GROUND TRUTH TRAINING DATA:")
Image(filename='./test_batch0_gt.jpg', width=900)
```

Figure 2.7.1.15-Ground truth training data

```
print("GROUND TRUTH AUGMENTED TRAINING DATA:")
Image(filename='./train_batch2.jpg', width=900)
```

Figure 2.7.1.16-Ground truth augmented training data

```
import glob
from IPython.display import Image, display

for imageName in glob.glob('/content/yolov5/inference/output/*.jpg'): #assuming JPG
    display(Image(filename=imageName))
    print("\n")
```

Figure 2.7.1.17-Display inference on all test images

Figure 2.7.1.17 displays inference on all test images and looks much better with longer training above.

Marker Based watershed algorithm

```
from __future__ import print_function
import cv2
import matplotlib.pyplot as plt
import numpy as np
from PIL import Image
from skimage.feature import peak_local_max
from skimage.morphology import watershed
from scipy import ndimage
```

Figure 2.7.1.18-Imported packages for watershed algorithm

Figure 2.7.1.18 shows the packages, which need to install. OpenCV-Python is a library of Python bindings designed to solve computer vision problems. Numpy is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays. pyplot is matplotlib's plotting framework. That specific import line merely imports the module "matplotlib.pyplot" and binds that to the name "plt". peak_local_max used for find peaks in an image as coordinate list or boolean mask.

```
def HSV(self):
    hsv = cv2.cvtColor(self.img, cv2.COLOR_BGR2HSV)
    #get green colr range
    mask = cv2.inRange(hsv, (36, 25, 25), (70, 255,255))

    #check whether mask  contaion 0
    mask_1 = mask >0

    #0 contain metrix
    Green_img = np.zeros_like(self.img,np.uint8)
    #Apply filtered green color mask t to  Green_img[mask_1]
    Green_img[mask_1] = self.img[mask_1]

    cv2.imwrite("Green_only.png", Green_img)
    plt.imshow(Green_img)
    cv2.waitKey()
```

Figure 2.7.1.19-Extracting green color range area from the background

The figure 2.7.1.19 illustrates extracting the green color range from the image and create green color mask. Before extracting the green color range, input image BGR format is converted into HSV color range. Because R, G, B in RGB are all co-related to the color luminance that is we can't distinguish luminance from color details. For separating image luminance from color details, HSV or Hue Saturation Value is used. This makes it clearer when we are working on the image / frame luminance or need it. In situations where color description plays an integral role, HSV is also used.

```
def resize_img(self,image):

    dims = (600,600)
    resize_imge = cv2.resize(image,dims)
    return resize_imge
```

Figure 2.7.1.20-Resize image

The figure 2.7.1.20 is for resize the input image.

```
#remove white color holes
def filter_white(self,image, M):

    filter_w = np.full((image.shape[0], image.shape[1]), True)
    filter_w[image[:, :, 0] <= 200] = False
    filter_w[image[:, :, 1] <= 220] = False
    filter_w[image[:, :, 2] <= 200] = False
    M[filter_w] = False

    return M


#remove black color holes
def filter_black(self,image, M):

    filter_b = np.full((image.shape[0], image.shape[1]), True)
    filter_b[image[:, :, 0] >= 50] = False
    filter_b[image[:, :, 1] >= 50] = False
    filter_b[image[:, :, 2] >= 50] = False
    M[filter_b] = False

    return M

  #remove blue color holes
def filter_blue(self,image, M):

    filter_bl = image[:, :, 0] > image[:, :, 1]
    M[filter_bl] = False
    return M
```

Figure 2.7.1.21-Remove white, black and blue color from green range mask

This figure 2.7.1.21 illustrates about removing white, black and blue colors. When we filter the green color range area, sometimes it contains those colors as well, in order to remove them, those logics are used.

```
#Remove dots
shifted = cv2.pyrMeanShiftFiltering(image_filtered, 21, 51)
cv2.imshow("Input", image_filtered)
cv2.waitKey()
```

Figure 2.7.1.22-Extracting green color range area from the background

Figure 2.7.1.22 code segment used to remove unwanted dots from the filtered green color mask.

```
#convert image into grey color
gray = cv2.cvtColor(shifted, cv2.COLOR_BGR2GRAY)

#apply treshold value
thresh = cv2.threshold(gray, 0, 255,cv2.THRESH_BINARY | cv2.THRESH_OTSU)[1]
cv2.imshow("Thresh", thresh)
cv2.waitKey()

#take the distance between ajacent leaves
ND_image = ndimage.distance_transform_edt(thresh)

localMax = peak_local_max(ND_image, indices=False, min_distance=20,labels=thresh)
markers = ndimage.label(localMax, structure=np.ones((3, 3)))[0]
#apply watershed algorithm
labels = watershed(-ND_image, markers, mask=thresh)
```

Figure 2.7.1.23-convert green color mask to grey color mask

According to Figure 2.7.1.23, filtered green color mask is converted to grey color mask. And then apply threshold OTSU and takes the distance between adjacent leaves. Then count the local maximum value for each and apply watershed segmentation.

```
        if self.leaf_type == 'Single':
            area = cv2.contourArea(c)
            if 10 < area and 10 < w and h > 5:
                contours_dict[(x, y, w, h)] = c

        else:
            cv2.rectangle(self.img, (x, y), (x+w, y+h), (0, 255, 0), 2)
            crp = self.orig.copy()[y:y+h, x:x+w]

            nx,ny,ch = crp.shape

            if nx >= 100 and ny >=100:
                cv2.imwrite("cropped/croped_{}.png".format(i),crp)
                i = i + 1
```

Figure 2.7.1.24-Check whether the image contains multiple leaves or single leaf

As shown above figure 2.7.1.24, if the image contains the single leaf, first if part is executed. If the image contains multiple leaves, second part is executed. The value for the area (area of the bounding boxes) is applied by trying different values to the code and get the most fitted value.

```
    if self.root == "Root":
        mask = cv2.inRange(hsv,(0, 48, 80),(20, 255, 255))
```

Figure 2.7.1.25-Brown color range area for digested root detection

As first step to identify the digested root, brown color range area is extracted.

```
#check overlap leaves
def is_overlapping_horizontally(self,box1, box2):
    x1, _, w1, _ = box1
    x2, _, _, _ = box2
    if x1 > x2:
        return self.is_overlapping_horizontally(box2, box1)
    return (x2 - x1) < w1

#merge
def merge(self,box1, box2):
    assert self.is_overlapping_horizontally(box1, box2)
    x1, y1, w1, h1 = box1
    x2, y2, w2, h2 = box2
    x = min(x1, x2)
    w = max(x1 + w1, x2 + w2) - x
    y = min(y1, y2)

    h = max(y1 + h1, y2 + h2) - y
    return (x, y, w, h)
```

Figure 2.7.1.26-Merge overlaps bounding boxes

Several bounding boxes are created around the detected leaf objects. In this figure 2.7.1.26, check all overlapping bounding boxes and merge them into single bounding box.

2.7.2 Testing

Software testing is the way toward getting to the functionality of the system and find whether it fulfills the requirements or not. Two key modes of testing are as Static Testing and Dynamic Testing.

1) Static Testing

Static testing is a form of test in which the code does not execute. It should be done by hand or with a collection of equipment. The code, requirement documents and design documents are reviewed by this kind of testing and survey comments are put on the job. With static testing, we endeavor to discover the errors, code defects and potentially malicious code in the application. Static testing should be done on work documents like source code, design documents, requirement specifications, test scripts and test cases etc. The Static test techniques are Inspection, Walkthrough, Technical Reviews, Informal Reviews and Static Code Review.

- **Inspection**

The main purpose of inspection is to find defects. This is a formal type of review where a checklist is prepared to review the work documents.

- **Walkthrough**

In walkthrough a meeting is led by initiator to describe the application. Members can make inquiries and a recorder is signed to make notes.

- **Technical Reviews**

A technical survey round is performed to verify whether the code is rendered according to technical requirements and standards in this form of static testing. For the most part the test plans, test procedure and test scripts are evaluated here.

- **Informal Reviews**

Informal Reviews is a static testing method in which the document is explored casually and informal remarks are given.

- **Static Code Review**

Static Code Review is the systematic review of the product source code without executing the code. It checks code syntax, coding standards, optimization of the code and so on. It is also known as white box testing. This testing could be done anytime during development.

1) Dynamic Testing

When the code is in operation mode we can do the dynamic testing. This testing is performed in runtime environment. At the point when the code being executed is input with a value, the output of the code is checked and compared with the expected output. In dynamic testing we can watch the functional behavior of the software. Types of Dynamic Testing techniques are Unit Testing, Integration Testing, System Testing and Acceptance Testing.

I. **Unit Testing**

In here one tests each unit individually to determine if they are properly implemented and fit for use. This is the smallest testable part of the system. Using the unit testing can find problems very early and at a considerably lower cost of detecting, identifying than correcting the bug later.

II. **Component Testing**

This tests each component in system separately. These tastings also finds the defects in the module and verifies the functioning of system.

III. **Integration Testing**

This tests how modules interact with each other. Combination of the modules are tested as a group. This assures the functional performance and reliability of the system.

IV. **System Testing**

This tests the overall functionality of the system. In here one tests the both functional and Non-Functional situations such as functionality, performance, scalability and reliability and compliance.

During testing we can find defects of the system. Then we can improve them to achieve the maximum out of our product. The main advantage of this testing is to enhance the quality of the system. To review the features of the functions we created a set of test cases. Test cases are a set of conditions with test case ID, set of test data, pre-conditions and expected results. Here is a sample of the test cases that has been applied to test the system.

Table 2.7.1-Test Cases

Test case No	**Description**	**Preconditions**	**Test Steps**	**Input**	**Expected output**
1	Register user With valid data	Previously used email cannot be used.	Enter username and password and then press login button in register interface	Username: m.nithmali@gmail.com Password: mithu1997	User should be registered to the system by creating a new account with the given credentials.
2	Register user With invalid data	Previously used email cannot be used.	Enter username and password and then press login button in register interface	Username: m.nithmali@gmail.com Password: mithu1997	User shouldbegivenan error message with 54real time validationas "enter a valid email address", and registration should be unsuccessful.

1	User Logs in using correct username and password.	Login details should be provided	Enter username and password and then press login button	Username: m.nithmali@gmail.com Password: mithu1997	User can login to the system.
2	User Logs in using incorrect credentials	Login details should be provided.	Enter username and password and then press login button	Username: m.nithmali12@gmail.com Password: mithu19972	System gives an error message.
3	Detect the leaf from input image	Login details should be provided and upload an image to the application	Upload Image and press upload button.	Input image:	Detected leaves.

After a successful development stage, following are the outcomes of above implementations in the UI point of view.

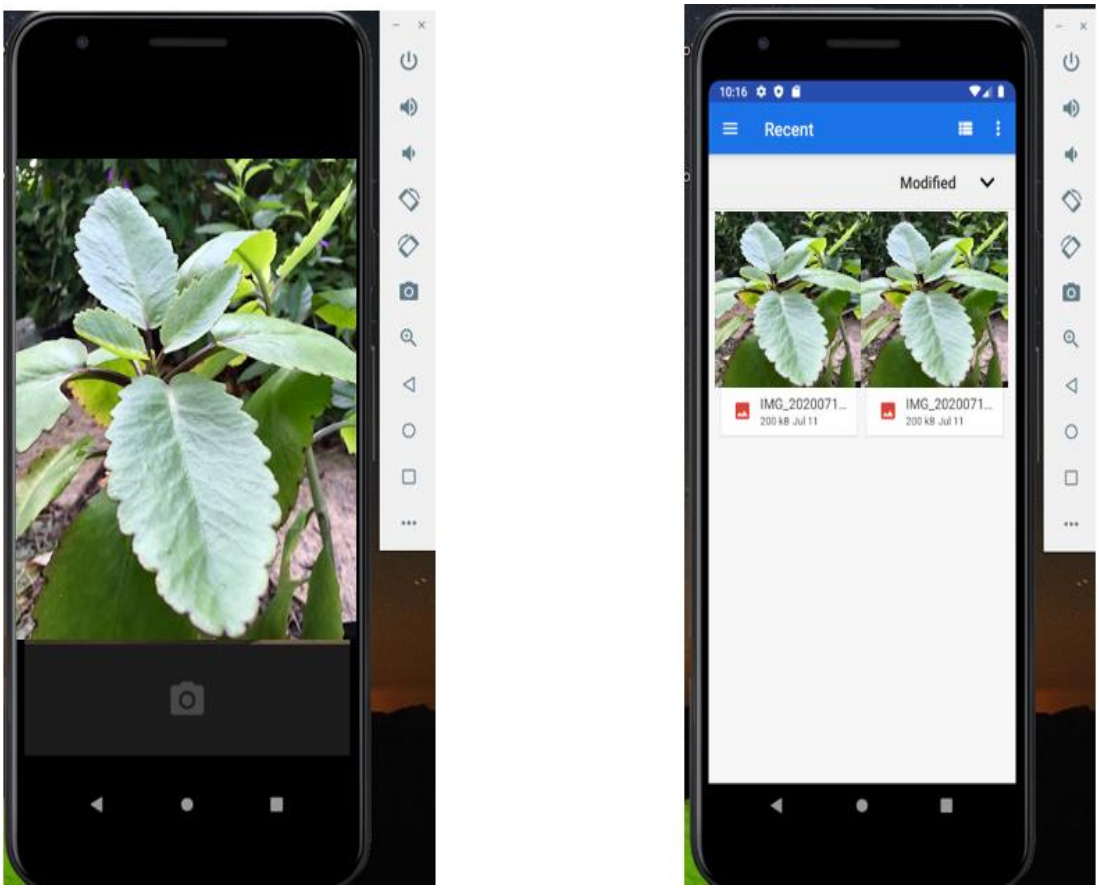


Figure 2.7.2.1-Mobile UIs

3. RESULTS AND DISCUSSION

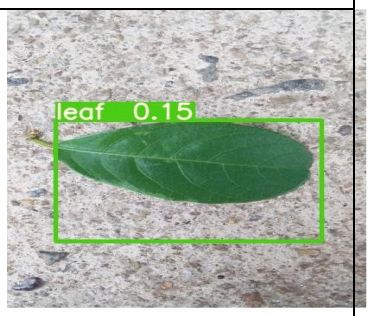
3.1 Results

1) YOLO

Same Annotated dataset is used in YOLO v3 and YOLO v5. To compare the result of these two versions of YOLO detection algorithm, same test data is used. Results of the test data for each version are given below.

Display inference on test images.

Table 3.1.1-Compare results of YOLO V3 and YOLO V5

Input image	YOLO v3	YOLO v4
01 [Leaf in a colored background] 		

02 [Leaf in a white background] 		
03 [leaves in a complex background] 		
04)[leaves in a dark background] 		

Visualize Ground truth training data

Figure 2.7.2.1-Visualize ground truth training data

Visualize Ground Augmented training data

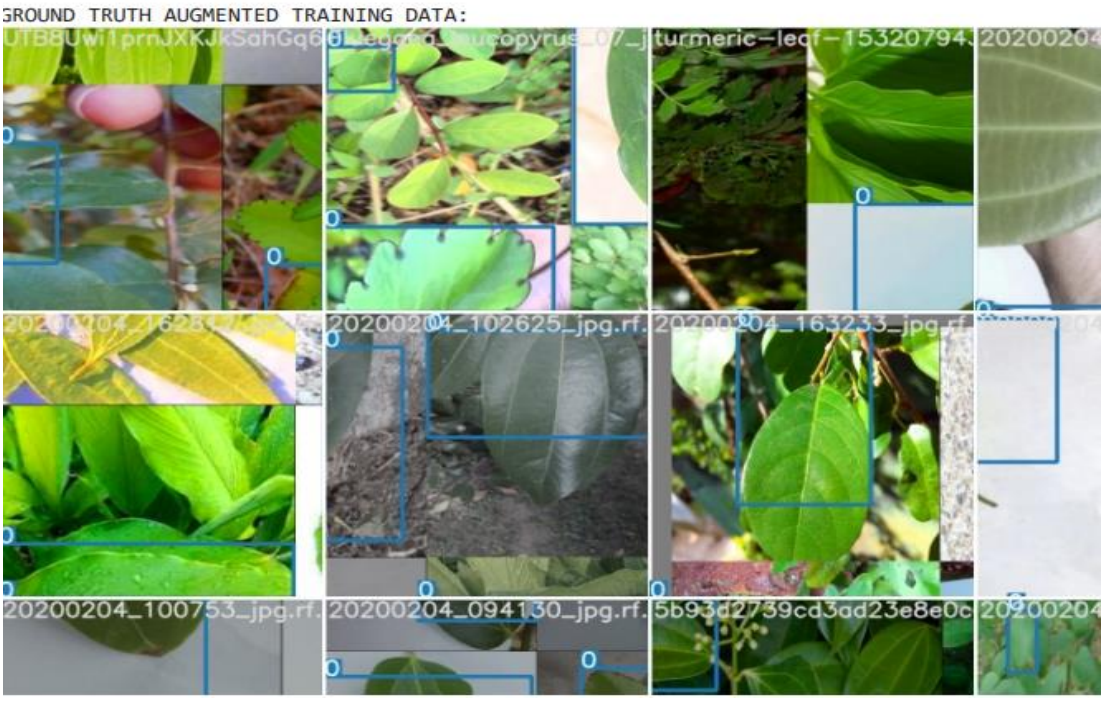


Figure 2.7.2.2-Visualize ground augmented training data

Evaluate Custom YOLOv5 Detector Performance

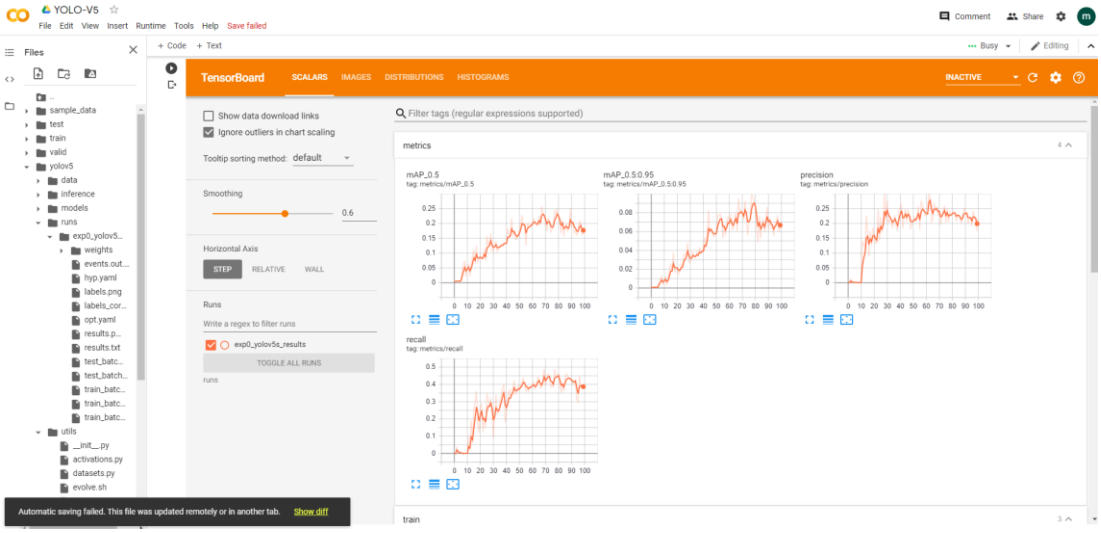


Figure 2.7.2.3-Performance Metrics of YOLO V5

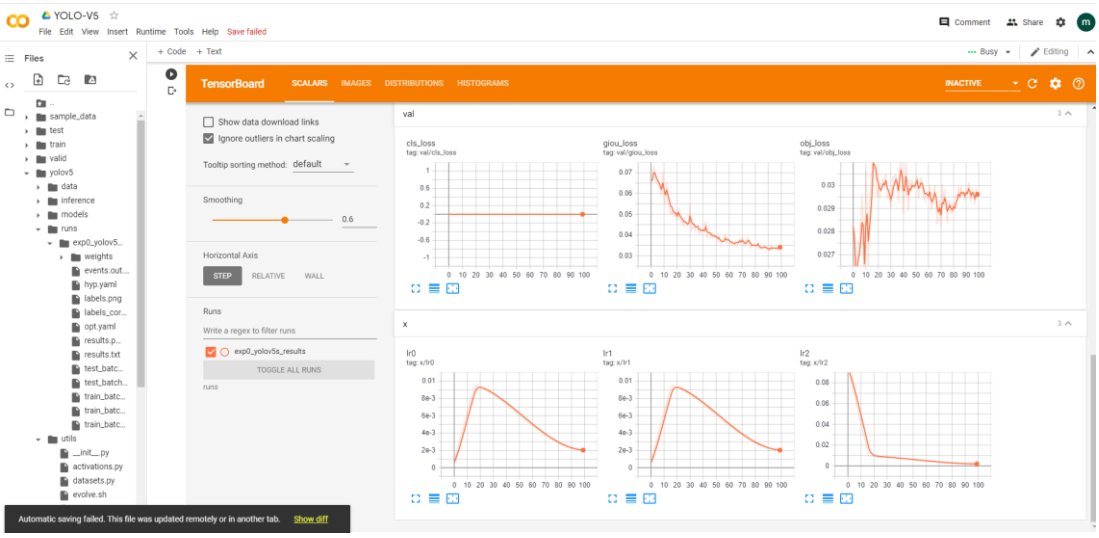


Figure 3.1.2-Validation performance of YOLO V5

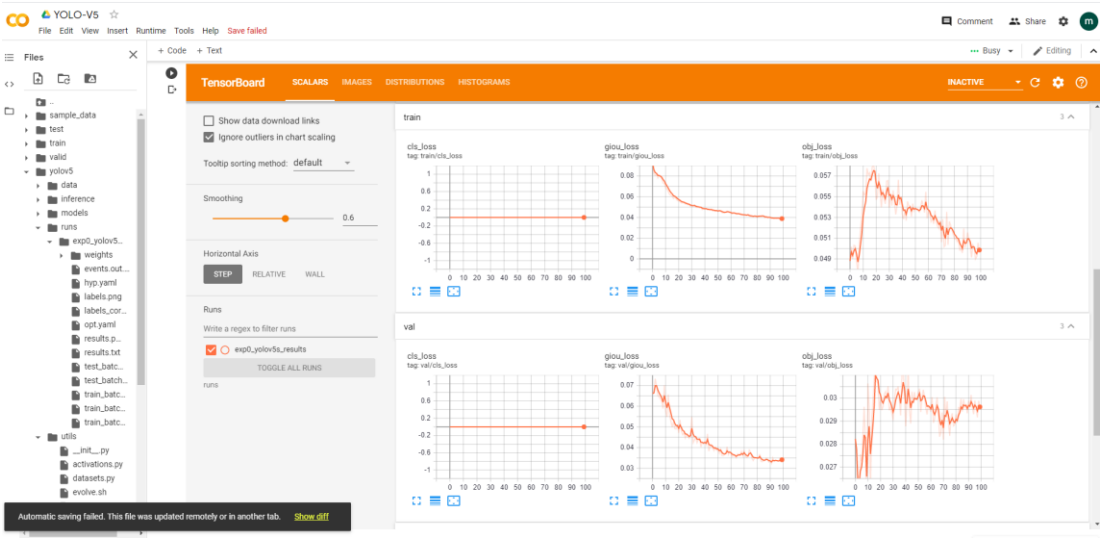


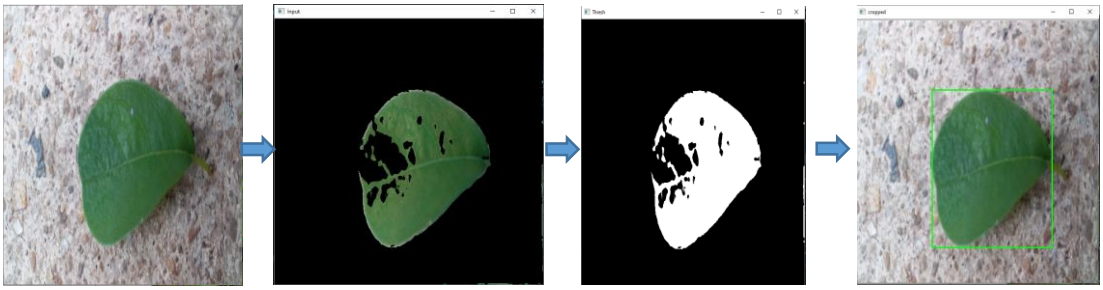
Figure 3.1.3-Training performance of YOLO V5

2) Proposed marker based watershed segmentation

Display inference on test images.

Above test data is used here also to compare the detection result.

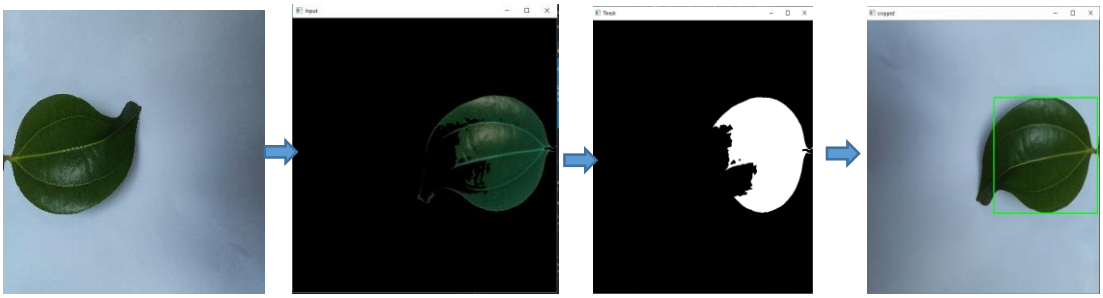
Input 1: [leaf in a colored background]



Input image green mask threshold detected object

Figure 2.7.2.4-Leaf in a colored background

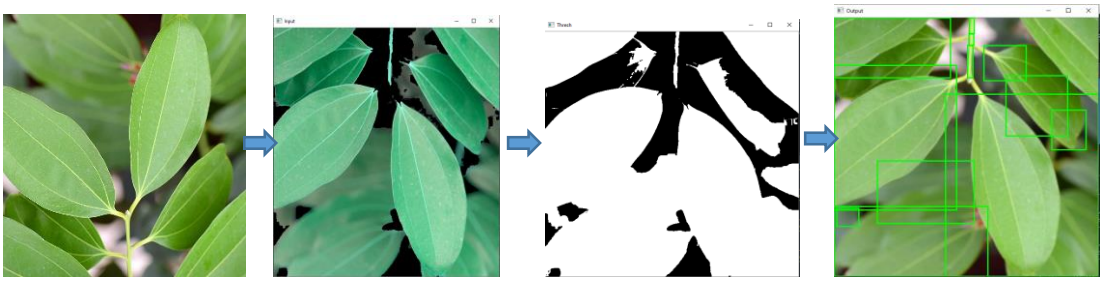
Input 2: [leaf in a white background]



Input image green mask threshold detected object

Figure 2.7.2.5-Leaf in a white background

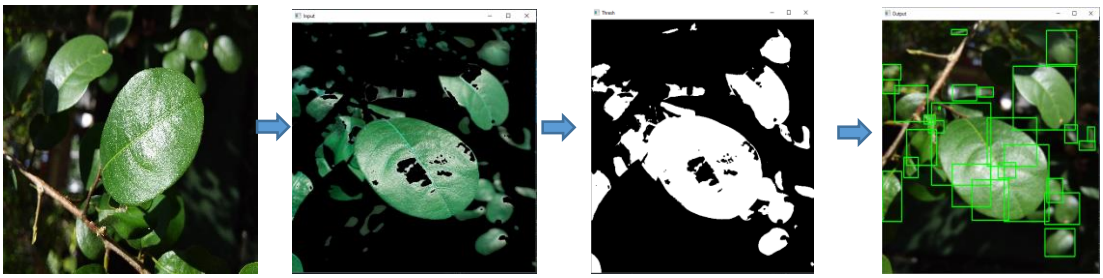
Input 3: [multiple leaves in a complex background]



Input image green mask threshold detected object

Figure 2.7.2.6-Multiple leaves in a complex background

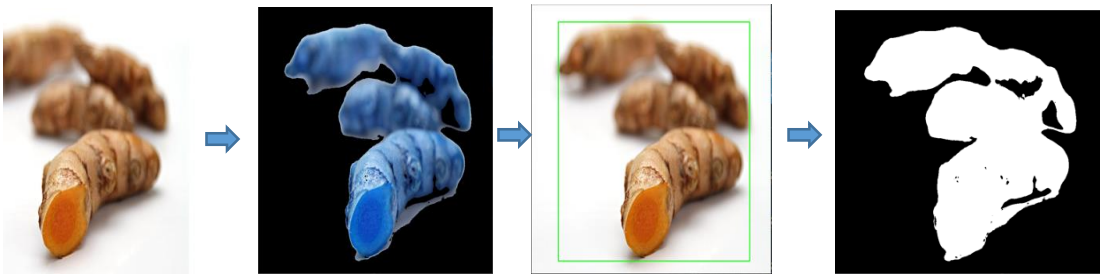
Input 4: [multiples leaves in a dark background]



Input image green mask threshold detected object

Figure 2.7.2.7-Multiples leaves in a dark background

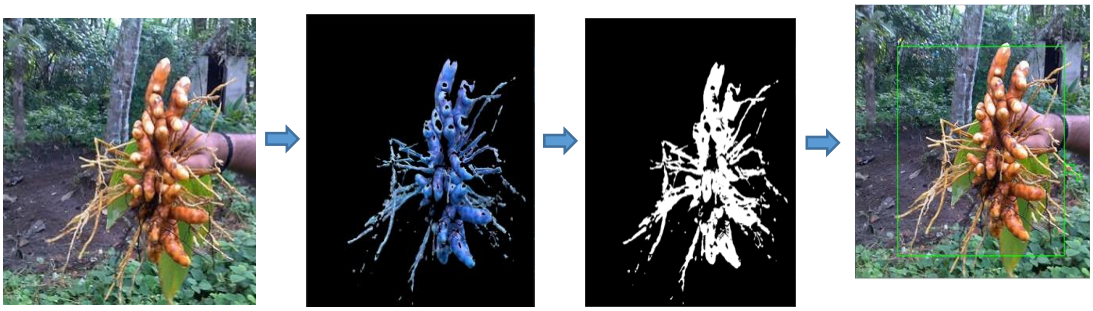
Input 5: [Digested root detection in a simple background]



Input image blue mask threshold detected object

Figure 2.7.2.8-Digested root detection in a simple background

Input 5: [Digested root detection in a complex background]



Input image blue mask threshold detected object

Figure 2.7.2.9-Digested root detection in a complex background

3.2 Research findings

This document proposes an innovative solution to help the Arogya users to find out the plant most accurately by simply uploading the picture of the plant. When doing the research many irrelevant topics have to be examined and filtered before the real research could be done. But all this relevant content had to be read over and over again for it to be made clear and to be used in the design stage of the development.

When developing object detection component, it was very difficult for us to find a proper objects detection technique at a glance for our proposed system because there are number of detection techniques are available. Therefore, to get a better outcome which means to apply a better detection model to our proposed system, we did some experiment with selected models. Comparison result of them is given above section. And finally select the most suitable model for our proposed mobile application.

From the generated results, proposed marker based watershed algorithm perform well for our proposed system. When compare YOLO v3 and YOLO v5, YOLO v5 performs well. According to the result, YOLO v5 works well even if the image has complex background rather than YOLO v3. And also to train the dataset, YOLO V5 takes less time than YOLO v3.Thus, YOLO v5 is superfast than YOLO v3.However, YOLO has some drawbacks. Sometimes, it struggles with small objects that appear in groups, it struggles to generalize to objects in new or unusual aspect ratios or configurations, to identify the object accurately, and at least it needs 1000 images per category. Thus, we need to annotate large datasets, and need to train the dataset and it takes time. Sometimes, it doesn't work on unseen data. However, according to the results comes from our proposed watershed, proposed marker-based watershed segmentation provided several advantages such as it required low computation time, and provided closed contours, the boundaries of the resulting regions always correspond to contours which appear in the image as obvious contours of objects, fast simple, no need to train the dataset .Thus no need to annotate data and saves time as well. And the accuracy of the final selected model is high. I selected 400 images for manual testing and 394 images were detected the plant parts. We can calculate the accuracy of the manual testing.

Accuracy of the proposed method = (394%400) * 100 = 98.5%

This proposed algorithm works well in our application as described and this is used as the finalized model.

3.3 Discussion

Main purpose of the component is to detect the object (plant parts) in real time even if image has a complex background. It was started by comparing and analyzing the existing systems. And also from the beginning, it was not an easy journey to get here. Several object detections techniques were studied and evaluated in order to get best result. Often more time had to be spent checking the consistency of the product to get the best results, rather than focusing on further implementation. As the very first step, Ayurvedic plant images were collected to make our dataset. Annotating and labeling the images took much time at the beginning. However, there were several ups and downs in the life cycle of the project, eventually we could really cover the scale of the project. Our Arogya mobile application was completed successfully as well as full filled the customer requirements and satisfaction.

.

4. CONCLUSION

The problems could not be solved by traditional object detection techniques when very small-shaped leaves or large numbers of strongly overlapped leaves appear in the images [18]. It takes less than a minute for the human brain to locate the position of the object within the image and recognize it as soon as it sees it; but to do the same job, the computer requires time and a large amount of data. However, this paper presents the survey on different object detection techniques used for plant leaf and digested root detection. We experimented with YOLO v3, YOLO v4 and our proposed watershed algorithm for the research problem. The experimental results prove that the proposed marker based segmentation can be successfully applied in leaf recognition and segmentation from a plant image. With very less computational efforts the optimum results were obtained, which also shows the efficiency of proposed algorithm in of the leaf or digested root.

5. REFERENCES

[1]Neeraj Kumar, Peter N Belhumeur, Arijit Biswas,David W Jacobs, W John Kress, Ida C Lopez, João VB Soares, "Leafsnap: A computer vision system for automatic plant species identification" in ECCV, Springer, pp. 502-517, 2012

[2] Pushpa, B. R., Anand, C., & Mithun Nambiar, P. (2016). 169Ayurvedic plant species recognition using statistical parameters on leaf images. International Journal of Applied Engineering Research, 11(7), 5142–5147.

[3] Pushpa, B. R., Anand, C., & Mithun Nambiar, P. (2016). 169Ayurvedic plant species recognition using statistical parameters on leaf images. *International Journal of Applied Engineering Research*, *11*(7), 5142–5147.

[4] Tan, J. W., Chang, S. W., Abdul-Kareem, S., Yap, H. J., & Yong, K. T. (2020). Deep Learning for Plant Species Classification Using Leaf Vein Morphometric. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, *17*(1), 82–90. https://doi.org/10.1109/TCBB.2018.2848653

[5] Chaki, J., Parekh, R., & Bhattacharya, S. (2015). Plant leaf recognition using texture and shape features with neural classifiers. *Pattern Recognition Letters*, *58*, 61–68. https://doi.org/10.1016/j.patrec.2015.02.010

[6] Pimm, S. L., Jenkins, C. N., Abell, R., Brooks, T. M., Gittleman, J. L., Joppa, L. N., Raven, P. H., Roberts, C. M., & Sexton, J. O. (2014). The biodiversity of species and their rates of extinction, distribution, and protection. *Science*, *344*(6187). https://doi.org/10.1126/science.1246752

[7] Joly, A., Müller, H., Goëau, H., Glotin, H., Spampinato, C., Rauber, A., Bonnet, P., Vellinga, W. P., & Fisher, B. (2014). LifeCLEF: Multimedia life species identification. *CEUR Workshop Proceedings*, *1222*(March), 7–13.

[8] De Luna, R. G., Baldovino, R. G., Cotoco, E. A., De Ocampo, A. L. P., Valenzuela, I. C., Culaba, A. B., & Gokongwei, E. P. D. (2017). Identification of philippine herbal medicine plant leaf using artificial neural network. *HNICEM 2017 - 9th International Conference on Humanoid, Nanotechnology, Information Technology, Communication and Control, Environment and Management*, *2018-January*, 1–8. https://doi.org/10.1109/HNICEM.2017.8269470

[9] Anami, B. S., Nandyal, S. S., & Govardhan, A. (2010). A Combined Color, Texture and Edge Features Based Approach for Identification and Classification of Indian Medicinal Plants. *International Journal of Computer Applications*, *6*(12), 45–51. https://doi.org/10.5120/1122-1471

[10] Jamil, N., Hussin, N. A. C., Nordin, S., & Awang, K. (2015). Automatic Plant Identification: Is Shape the Key Feature? *Procedia Computer Science*, *76*(Iris), 436–442. https://doi.org/10.1016/j.procs.2015.12.287

[11]https://www.analyticsvidhya.com/blog/2019/04/introductionimagesegmentationtechniques-python/

[12] Singh, V. (2019). Sunflower leaf diseases detection using image segmentation based on particle swarm optimization. *Artificial Intelligence in Agriculture*, *3*, 62–68. https://doi.org/10.1016/j.aiia.2019.09.002

[13] De Luna, R. G., Baldovino, R. G., Cotoco, E. A., De Ocampo, A. L. P., Valenzuela, I. C., Culaba, A. B., & Gokongwei, E. P. D. (2017). Identification of philippine herbal medicine plant leaf using artificial neural network. *HNICEM 2017 - 9th International Conference on Humanoid, Nanotechnology, Information Technology, Communication and Control, Environment and Management*, *2018-Janua*, 1–8. https://doi.org/10.1109/HNICEM.2017.8269470

[14] Ma, L. H., Zhao, Z. Q., & Wang, J. (2013). ApLeafis: An Android-based plant leaf identification system. *Lecture Notes in Computer Science (Including*

Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), *7995 LNCS*(61005007), 106–111. https://doi.org/10.1007/978-3-642-39479-9_13

[15] Prasvita, D. S., & Herdiyeni, Y. (2013). MedLeaf: Mobile Application for Medicinal Plant Identification Based on Leaf Image. *International Journal on Advanced Science, Engineering and Information Technology*, *3*(2), 103. https://doi.org/10.18517/ijaseit.3.2.287

[16] Barré, P., Stöver, B. C., Müller, K. F., & Steinhage, V. (2017). LeafNet: A computer vision system for automatic plant species identification. *Ecological Informatics*, *40*, 50–56. https://doi.org/10.1016/j.ecoinf.2017.05.005

[17] Redmon, J., & Farhadi, A. (2018). *YOLOv3: An Incremental Improvement*. http://arxiv.org/abs/1804.02767

[18] Xu, L., Li, Y., Sun, Y., Song, L., & Jin, S. (2018). Leaf instance segmentation and counting based on deep object detection and segmentation networks. *Proceedings - 2018 Joint 10th International Conference on Soft Computing and Intelligent Systems and 19th International Symposium on Advanced Intelligent Systems, SCIS-ISIS 2018*, *61501434*, 180–185. https://doi.org/10.1109/SCIS-ISIS.2018.00038

Appendices

1) YOLO V3

```
import cv2
import time
import tensorflow as tf
import tensorflow_hub as hub
```

```
import os
from tensorflow.keras.callbacks import LearningRateScheduler, ModelCheckpoint
from tensorflow.keras.layers import Dense, Flatten, Conv2D
from tensorflow.keras import Model
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import Adam
from tensorflow.keras import layers
import sklearn
#images are really high scale.Therefore to rescale the images


from PIL import Image
import numpy as np

confidence_limit = 0.2
threshold = 0.1




def leaf_detector(img):

    orig = img

#    Load YOLO
#    Wieght file for trained the model, the core algorithm to detect the object
#    cfg file for all configuration of the algorithm
    net = cv2.dnn.readNet("yolov3_training_last.weights", "yolov3-tiny-obj.cfg")

    np.random.seed(42)

    layer_names = net.getLayerNames()
```

```
    output_layers = [layer_names[i[0] - 1] for i in net.getUnconnectedOutLayers()]

    H, W, channels = img.shape

#We can't use the full image on the network,We should convert it into the blob.
    #detecting objects
    blob = cv2.dnn.blobFromImage(img, 1 / 255.0, (416, 416), swapRB=True, crop=True)

    #passed the image to the network(algorithm)
    net.setInput(blob)

#Outs is an array that conains all the informations about objects detected, their position and the confidence about the detection
    layerOutputs = net.forward(output_layers)

    boxes = []
    confidences = []
    classIDs = []

    for output in layerOutputs:

        for detection in output:
            scores = detection[5:]
            classID = np.argmax(scores)
            confidence = scores[classID]

            if confidence > confidence_limit:

                box = detection[0:4] * np.array([W, H, W, H])
                (centerX, centerY, width, height) = box.astype("int")
                x = int(centerX - (width / 2))
```

```
            y = int(centerY - (height / 2))
            boxes.append([x, y, int(width), int(height), centerX, centerY])
            confidences.append(float(confidence))
            classIDs.append(classID)

    idxs = cv2.dnn.NMSBoxes(boxes, confidences, confidence_limit, threshold)

    crop_boxes = []

    if len(idxs) > 0:

        for i in idxs.flatten():

            (x, y) = (boxes[i][0], boxes[i][1])
            (w, h) = (boxes[i][2], boxes[i][3])

            crop_boxes.append((x, y, x + w, y + h))

            cv2.rectangle(orig, (x, y), (x + w, y + h), (0,255,0), 2)

    return(crop_boxes)


def run_model(image_name):

    #loading the image
    img = cv2.imread(image_name)
    bounding_boxes = leaf_detector(img)

    i =0
    if (len(bounding_boxes) > 0):
        for box in bounding_boxes:
```

```
crop_img = img[box[1]:box[3], box[0]:box[2]]

crop_img = cv2.resize(crop_img, (256, 256))
cv2.imwrite("./detection/cropped-leaf_{}.jpg".format(i), crop_img)
i = i + 1
```

2)YOLO V5

```
# Export code snippet and paste here %cd /content !curl -L
    "https://app.roboflow.ai/ds/wnkGuPjBVD?key=3ak4Ufpdvd" > roboflow.zip;
    unzip roboflow

# train yolov5s on custom data for 50 epochs # time its performance %%time %cd
    /content/yolov5/ !python train.py --img 416 --batch 16 --epochs 100 --data
    '../data.yaml' --cfg ./models/custom

# define number of classes based on YAML import yaml with open("data.yaml", 'r')
    as stream: num_classes = str(yaml.safe_load(stream)['nc'])


%cd /content/
##write custom model .yaml
#you can configure this based on other YOLOv5 models in the models directory
with open('yolov5/models/custom_yolov5s.yaml', 'w') as f:
 # parameters
 f.write('nc: ' + num_classes + '\n')
 #f.write('nc: ' + str(len(class_labels)) + '\n')
 f.write('depth_multiple: 0.33' + '\n') # model depth multiple
( )
/content
custom model config written!
```

```
f.write('width_multiple: 0.50' + '\n') # layer channel multiple
f.write('\n')
f.write('anchors:' + '\n')
f.write(' - [10,13, 16,30, 33,23] ' + '\n')
f.write(' - [30,61, 62,45, 59,119]' + '\n')
f.write(' - [116,90, 156,198, 373,326] ' + '\n')
f.write('\n')
f.write('backbone:' + '\n')
f.write(' [[-1, 1, Focus, [64, 3]],' + '\n')
f.write(' [-1, 1, Conv, [128, 3, 2]],' + '\n')
f.write(' [-1, 3, Bottleneck, [128]],' + '\n')
f.write(' [-1, 1, Conv, [256, 3, 2]],' + '\n')
f.write(' [-1, 9, BottleneckCSP, [256]],' + '\n')
f.write(' [-1, 1, Conv, [512, 3, 2]], ' + '\n')
f.write(' [-1, 9, BottleneckCSP, [512]],' + '\n')
f.write(' [-1, 1, Conv, [1024, 3, 2]],' + '\n')
f.write(' [-1, 1, SPP, [1024, [5, 9, 13]]],' + '\n')
f.write(' [-1, 6, BottleneckCSP, [1024]],' + '\n')
f.write(' ]' + '\n')
f.write('\n')
f.write('head:' + '\n')
f.write(' [[-1, 3, BottleneckCSP, [1024, False]],' + '\n')
f.write(' [-1, 1, nn.Conv2d, [na * (nc + 5), 1, 1, 0]],' + '\n')
f.write(' [-2, 1, nn.Upsample, [None, 2, "nearest"]],' + '\n')

f.write(' [[-1, 6], 1, Concat, [1]],' + '\n')
f.write(' [-1, 1, Conv, [512, 1, 1]],' + '\n')
f.write(' [-1, 3, BottleneckCSP, [512, False]],' + '\n')
f.write(' [-1, 1, nn.Conv2d, [na * (nc + 5), 1, 1, 0]],' + '\n')

f.write(' [-2, 1, nn.Upsample, [None, 2, "nearest"]],' + '\n')
f.write(' [[-1, 4], 1, Concat, [1]],' + '\n')
```

```
 f.write(' [-1, 1, Conv, [256, 1, 1]],' + '\n')
 f.write(' [-1, 3, BottleneckCSP, [256, False]],' + '\n')
 f.write(' [-1, 1, nn.Conv2d, [na * (nc + 5), 1, 1, 0]],' + '\n')
 f.write('\n' )
 f.write(' [[], 1, Detect, [nc, anchors]],' + '\n')
 f.write(' ]' + '\n')
print('custom model config written!')
```

Start tensorboard

Launch after you have started training

first, display our ground truth data print("GROUND TRUTH TRAINING DATA:") Image(filename='./test_batch0_gt.jpg', width=900)

print out an augmented training example print("GROUND TRUTH AUGMENTED TRAINING DATA:") Image(filename='./train_batch2.jpg', width=900)

trained weights are saved by default in our weights folder

%ls weights/

%cd /content/yolov5/ !python detect.py --weights weights/last_yolov5s_results.pt --img 416 --conf 0.4 --source ../

#display inference on ALL test images import glob from IPython.display import Image, display

Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318

2) Proposed watershed segmentation

```
from __future__ import print_function
import cv2
import matplotlib.pyplot as plt
import numpy as np
from PIL import Image
from skimage.feature import peak_local_max
```

```
from skimage.morphology import watershed
from scipy import ndimage
class Detect_leaf:

    def __init__(self,img_path,leaf_type):
        print("first")

        self.img = self.resize_img(cv2.imread(img_path))
        self.orig = self.img.copy()
        self.img = cv2.GaussianBlur(self.img,(7,7),0)
        self.leaf_type = leaf_type

    #resize input image

    def resize_img(self,image):

        dims = (600,600)
        resize_imge = cv2.resize(image,dims)
        return resize_imge

    def HSV(self):
        hsv = cv2.cvtColor(self.img, cv2.COLOR_BGR2HSV)
        #get green colr range
        mask = cv2.inRange(hsv, (36, 25, 25), (70, 255,255))

        #check whether mask  contaion 0

        mask_1 = mask >0



        #0 contain metrix
        Green_img = np.zeros_like(self.img,np.uint8)
```

```
    #Apply filtered green color mask t to  Green_img[mask_1]
    Green_img[mask_1] = self.img[mask_1]

    cv2.imwrite("Green_only.png", Green_img)
    plt.imshow(Green_img)
    cv2.waitKey()

#remove white color holes
def filter_white(self,image, M):

    filter_w = np.full((image.shape[0], image.shape[1]), True)
    filter_w[image[:, :, 0] <= 200] = False
    filter_w[image[:, :, 1] <= 220] = False
    filter_w[image[:, :, 2] <= 200] = False
    M[filter_w] = False

    return M


#remove black color holes
def filter_black(self,image, M):



    filter_b = np.full((image.shape[0], image.shape[1]), True)
    filter_b[image[:, :, 0] >= 50] = False
    filter_b[image[:, :, 1] >= 50] = False
    filter_b[image[:, :, 2] >= 50] = False
    M[filter_b] = False

    return M
```

```
  #remove blue color holes
def filter_blue(self,image, M):



    filter_bl = image[:, :, 0] > image[:, :, 1]
    M[filter_bl] = False
    return M




def waterShed(self):

    image_wshd = cv2.imread("Green_only.png")
    marker = np.full((image_wshd.shape[0], image_wshd.shape[1]), True)
    plt.subplot(2,3,1);
    plt.imshow(marker)
    marker = self.filter_white(image_wshd,marker)
    plt.subplot(2,3,2);plt.imshow(marker)
    marker = self.filter_black(image_wshd,marker)
    plt.subplot(2,3,3);plt.imshow(marker)
    marker = self.filter_blue(image_wshd,marker)
    plt.subplot(2,3,4);plt.imshow(marker)

    new = np.zeros_like(self.orig,np.uint8)
    new[marker] = self.orig[marker]
    plt.imshow(new)


    testimg = Image.fromarray(new, 'RGB')
    testimg.save('filtered.png')
```

```
image_filtered = cv2.imread('filtered.png')
#Remove dots
shifted = cv2.pyrMeanShiftFiltering(image_filtered, 21, 51)
cv2.imshow("Input", image_filtered)
cv2.waitKey()

#convert image into grey color
gray = cv2.cvtColor(shifted, cv2.COLOR_BGR2GRAY)

#apply treshold value
thresh = cv2.threshold(gray, 0, 255,cv2.THRESH_BINARY | cv2.THRESH_OTSU)[1]
cv2.imshow("Thresh", thresh)
cv2.waitKey()

#take the distance between ajacent leaves
ND_image = ndimage.distance_transform_edt(thresh)

localMax = peak_local_max(ND_image, indices=False, min_distance=20,labels=thresh)
markers = ndimage.label(localMax, structure=np.ones((3, 3)))[0]
#apply watershed algorithm
labels = watershed(-ND_image, markers, mask=thresh)

contours_dict = dict()
i = 0
for label in np.unique(labels):
    if label == 0:
        continue
    mask = np.zeros(gray.shape, dtype="uint8")
```

```
        mask[labels == label] = 255
        #get the eges
        contours, heirarchy = cv2.findContours(mask.copy(),
cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)
        for c in contours:
            crp = self.img.copy()
            x,y,w,h = cv2.boundingRect(c)

            if self.leaf_type == 'Single':
                area = cv2.contourArea(c)
                if 10 < area and 10 < w and h > 5:
                    contours_dict[(x, y, w, h)] = c

            else:
                cv2.rectangle(self.img, (x, y), (x+w, y+h), (0, 255, 0), 2)
                crp = self.orig.copy()[y:y+h, x:x+w]

                nx,ny,ch = crp.shape

                if nx >= 100 and ny >=100:

                    cv2.imwrite("cropped/croped_{}.png".format(i),crp)

                    i = i + 1



    if self.leaf_type == "Single":

        img = self.orig.copy()

        #values sort
        contours_filtered = sorted(contours_dict.values(), key=cv2.boundingRect)
        boxes = self.windows(contours_filtered)
```

```
            for box in boxes:
                x, y, w, h = box
                img = cv2.rectangle(img, (x, y), (x + w, y + h), (0, 255, 0), 2)

cv2.imwrite("cropped/croped_{}.png".format(x+y),self.orig.copy()[y:y+h,x:x+w])
            cv2.imshow("cropped",img)
            cv2.waitKey()
            cv2.destroyAllWindows()
        else:
            cv2.imshow("Output", self.img)
            cv2.waitKey(0)
            cv2.destroyAllWindows()

    #check overlap leaves
    def is_overlapping_horizontally(self,box1, box2):
        x1, _, w1, _ = box1
        x2, _, _, _ = box2
        if x1 > x2:
            return self.is_overlapping_horizontally(box2, box1)
        return (x2 - x1) < w1



    #merge
    def merge(self,box1, box2):
        assert self.is_overlapping_horizontally(box1, box2)
        x1, y1, w1, h1 = box1
        x2, y2, w2, h2 = box2
        x = min(x1, x2)
        w = max(x1 + w1, x2 + w2) - x
        y = min(y1, y2)
```

```
        h = max(y1 + h1, y2 + h2) - y
        return (x, y, w, h)



    def windows(self,contours):

        boxes = []
        for cont in contours:
            box = cv2.boundingRect(cont)
            if not boxes:
                boxes.append(box)
            else:
                if self.is_overlapping_horizontally(boxes[-1], box):
                    last_box = boxes.pop()
                    merged_box = self.merge(box, last_box)
                    boxes.append(merged_box)
                else:
                    boxes.append(box)
        return boxes

    def main(self):
        self.HSV()
        self.waterShed()


'''
        # Detect_leaf("image path","Single") This is for single leaf image

        # Detect_leaf("image path","Multiple") This is for complex/multiple leaves image
```

```
'''

test = Detect_leaf("./samples/sample8.jpg","Multiple")
test.main()
```