# Deficiency Identification of Greenhouse Lettuce using Explainable AI

A.R.S.P. Rodrigo

2020

# Deficiency Identification of Greenhouse Lettuce using Explainable AI

**A dissertation submitted for the Degree of Master of Science in Computer Science**

**A.R.S.P. Rodrigo**

**University of Colombo School of Computing**

**2020**

# DECLARATION

The thesis is my original work and has not been submitted previously for a degree at this or any other university/institute.

To the best of my knowledge it does not contain any material published or written by another person, except as acknowledged in the text.

Student Name: ARSP Rodrigo

Registration Number: 2017/MCS/070

Index Number: 17440704

[illegible]

14/11/2020

Signature: Date:

This is to certify that this thesis is based on the work of

Mr./~~Ms.~~ ARSP Rodrigo

under my supervision. The thesis has been prepared according to the format stipulated and is of acceptable standard.

Certified by,

Supervisor Name: Dr. DD Karunaratne

Signature: Date

# ABSTRACT

The Convolutional Neural Network (CNN) based solutions are used to identify the nutrient deficiencies of the crops based on the color variances of the leaves. However, one of the major problems in the CNN based solutions are the lack of ability to explain the results obtained. This research is focused on overcoming this challenge by combining the results obtained from CNN with TensorFlow Inference Engine to provide humanely understandable results for deficiency identification of crops. Therefore, greenhouse lettuce is selected as the crop for the study. Greenhouse farming became popular with the technological evolution in the last few decades. This aims to provide an optimal nutrient composition to the corps, to protect the crops from pests without applying pesticides, and to provide the optimal environmental conditions to the corps such as temperature, humidity, etc. Lettuce is one of the mostly consumed vegetable crops among the green house crops but facing a yield loss due the nutrient deficiencies. Therefore, the early identification of deficiencies becomes crucial. A custom test bed is created to gather data/images and using those data/images YOLOv3 object detection model was trained to detect Calcium, Nitrogen, and Magnesium nutrient deficiencies of greenhouse lettuce. The results demonstrate a mean average precision of 94.38% on training data and 75.53% on custom data. The trained weights were combined with the TensorFlow Inference Engine to provide explainable results using a local knowledge base of deficiencies.

# ACKNKOWLEDGEMENTS

My deepest gratitude first and foremost, goes to Dr. Damith Karunarathne, my supervisor at University of Colombo School of Computing (UCSC), for friendly and open-minded supervision and enabling work on this thesis at the UCSC. Your support and understanding over these past 12 months have been tremendous and I would like to pay my gratitude for the guidance provided. Furthermore, I would like to thank Dr. Chamath Keppitiyagama for his early engagement and the rest of my research evaluation committee for the insightful comments and hard questions.

My sincere appreciation also extends to Dr. Subha Fernando, my advisor at CodeGen Internation (Pvt.) Ltd. for the continuous support, guidance, and motivation. I hope that I have managed to address some of the many precious points that she has raised during the discussions.

This research covered an area where I have very little knowledge, to begin with, so I would like to recognize the invaluable assistance provided by Mr. Chamil Chinthaka, the Agronomist at CodeGen International (Pvt.) Ltd. Thank you very much for the assistance given with the AiGROW greenhouse plant setup, maintenance, and immense knowledge.

It is whole-heartedly appreciated that the resources provided by CodeGen International (Pvt.) Ltd. and AiGROW for proved monumental support towards the success of this study.

Finally, I wish to show my gratitude towards my colleagues, friends, and family for the ideas, encouragement, and endless support given to make this study a success.

# TABLE OF CONTENTS

# TABLE OF FIGURES

# LIST OF TABLES

# Chapter 1 : Introduction

## 1.1 Introduction

Horticulture has become more than farming with technological evolution in the last few decades and Controlled Environment Architecture (CEA) became a popular approach to produce yield productively compared to open field production. CEA is defined as a combination of engineering, plant science, and computer managed greenhouse control technologies used to optimize plant growing systems, plant quality, and production efficiency. The greenhouse is one such CEA structure vastly used in the horticultural field with the aim of providing an optimal nutrient composition to the corps, to protect the crops from pests without applying pesticides, and to provide the optimal environmental conditions to the corps as in temperature, humidity, etc. Technavio's market research analysts have predicted that the greenhouse horticulture market will register a Compound Annual Growth Rate (CAGR) of more than 11% by 2022 and currently, it is at 10.79% [1].

Among all the greenhouse crops, lettuce is the most popular salad vegetable crop around the world and consists of vitamins, minerals, and the taste that attract people [2]. Greenhouse lettuce is produced in soil-less culture. This procedure refers to the techniques of Hydroponics and Aeroponics mainly. In Hydroponics, plants grow in containers with mineral nutrient solutions, without soil. The greenhouse associated with this research uses a deep flow technique to produce lettuce where a consistent nutrient environment for the roots is provided by the flowing solution culture [3].

## 1.2 Problem

Even though the greenhouse is a controlled environment and lettuce is produced in a nutrient solution, there can be a production loss due to diseases and nutrient deficiencies and toxicities. Some of the diseases and deficiencies generate manifestation in the visible spectrum and some have not. Diseases or deficiencies without any visible symptoms can be identified with electromagnetic analysis, microscopic analysis, etc. Due to their complexity, and to the extent of the literature review of this research, they will not be addressed under this research. The diseases with visible symptoms can be identified with remote sensing techniques [4]. Remote sensing techniques for disease identification in the visible spectrum will be discussed in the next sections.

Symptoms of nutrient deficiencies of hydroponic lettuce are available in the visible spectrum [5]. The identification of nutrient deficiencies using machine vision is a research area where researchers developed various methodologies to detect the defected plants [4], [6]–[8] which will be discussed in the literature review section. The methodologies discussed in sited projects are not developed in a point of serving the stakeholders of the horticultural society but in a theoretical manner where they cannot be applied in a real-life scenario.

## 1.3 Problem Domain

This study is affiliated with some key areas in computer science, including both trending, and non-trending. Convolutional Neural Networks, Object Detection, and Explainable AI are some of them. This subsection will explain in brief on the mentioned areas to compose the background of this study.

### 1.3.1 Convolutional Neural Networks

Artificial Neural Networks (ANN) are developed under the inspiration of the operations of biological neural networks. It contains a large number of interconnected operational units called neurons. They work collectively and interconnectedly in a distributed manner to learn from the input to process and develop the final output. The architecture contains input layers that take the input as a vector and distribute it to the second level. The second level contains the hidden layers, which take the decisions by learning from the outputs of previous layers to improve the final output [9]. Convolutional Neural Networks (CNNs) are similar in fashion to ANNs where both categorized under Deep Neural Networks. But the CNNs are developed by reducing the number of parameters in ANNs but the neurons of both are self-optimize through learning. In contrast to ANNs, CNN’s architecture there are three types of layers as Convolutional, Pooling, and Fully connected. Among these three, convolutional layers are the most important and take most of the time within the network, but they are also capable of reducing the complexity of the network. Currently, CNNs are mainly used in pattern recognition within images, videos, and voice. There are a different kinds of implementations of CNNs and they are introduced down the line of this research [10].

### 1.3.2 Object Detection

Object detection is one of the fundamental computer vision problems and the task of identifying the presence, location (object localization), and type (object classification) of a given object within a given image. Object detection is related to many applications like image classification, face recognition, human behavior analysis, obstacle avoidance, autonomous driving, etc. [11], [12] There are three types of object detection models. Namely, information region selection, feature extraction, and classification. Information region selection is the task of finding the possible positions of the objects, feature extraction is the reduction of the number of features in an image by combining the existing features and generating new features [13] and the classification is separating the given object from all the other objects in the given image and representing the information. The evolution of CNNs is closely coupled with object detection by improving the performance with leaning complex features within the images. R-CNN, Faster R-CNN are models that optimize the classification and bounding box generation [14]. Another promising model is YOLO and it detects the objects using fixed-grid regression [15]. Both of these models provide real-time and accurate object detection [16]. YOLOv3 is the current version of YOLO and will be discussed more in this article [17].

### 1.3.3 Explainable Artificial Intelligence (XAI)

Artificial Intelligence is introduced way back in the 19th century but, the true use and importance of AI are started only a few decades ago. The AI methods are currently achieving higher levels of performance in solving complex computational tasks and the AI powered systems are evolved to an extent that the humane intervention is almost in no need in the development and deployment. But the decisions taken and suggested by those AI powered systems eventually affect the humans' lives so, the need of understanding the given and taken decisions is made a right [18]. Even though the early stages of AI powered models can be explained easily, the latest models developed with systems like Convolutional Neural Networks are complex and not interpretable. This makes the system a black box [19]. Explainable AI is developed to open up this backbox and allow humans to understand, trust, and manage the emerging artificial intelligent models. This explanation is given by Dr. Gunning in [20].

According to DARPA [21] (Defense Advanced Research Projects Agency), the term XAI refers to the actions to make sure that the AI models are transparent in their actions for the given purpose. It also refers to the ability to understand the work logic behind the AI algorithms. So the idea behind this concept is that the Artificial Intelligence programs and technologies should not be back box systems that people with expert knowledge in that specific field can only understand as discussed in the previous paragraph but explaining why and how the model came into that decision.

## 1.4 Research Contribution

### 1.4.1 Goal

To develop an approach that can identify and explain a set of predefined nutrient deficiencies of greenhouse lettuce plants in real-time using image processing techniques and explainable AI theories.

### 1.4.2 Aims and Objectives

To achieve the above goal, there are some objectives that must be covered. These objectives can be listed as follows:

- Review of current methods of identifying the deficiencies of plants related to the problem domain.
- Create an image dataset of greenhouse lettuce plants which is grown on special recipes to visualize nutrient deficiencies.
- Preprocess the collected images with the help of an agronomist.
- Select deep learning architecture based on Convolutional Neural Networks based on the literature review to apply on the dataset.
- Develop an explainable deficiency identification approach using the selected model.

## 1.5 Scope of the study

Design and implementation of a methodology to automatically identify the nutrient deficiencies of greenhouse lettuce using image processing based on user input images and explain the solution using explainable AI-related theorems. The reason behind the selection of greenhouse lettuce in explains in the literature review section and this research only addresses three different nutrient deficiencies of lettuce. They can be explained as follows.

Calcium - Due to Calcium deficiency, the growth of lettuce is observed as reduced and leaves are wavier than normal. Brown or grey lesions has developed starting from leaf margins or tips of young leaves. This symptom is called as 'tip-burn'. When the Calcium -deficiency progresses, the leaves begin to die from tips and margins inwards. Subsequently, the persistent symptoms will spread over the older leaves.

Nitrogen - Older leaves have the symptoms of N deficiency at first with light green chlorosis. This moves to the head and will have light green chlorosis. No head is formed with severe Nitrogen deficiency and the growth is restricted. But the leave shape remains normal.

Magnesium - Older leaves have the symptoms of Mg deficiency at first with yellowing between veins and leaf margin discoloration of yellowish orange. If this continues, the yellow leaf zones will die but the veins remain green. Growth is incrementally restricted with the severity of the deficiency.

## 1.6 Summary

This research intends to overcome that problem by introducing a hybrid approach to convolutional neural networks and expert systems that can be applied to detect deficiencies of greenhouse lettuce. However, convolutional neural networks achieving the best performance in other research fields are not much applied in horticultural plant deficiency detection because of public datasets, but this will not be an issue in this research because of the collaboration of AIGrow [22] where the greenhouse has a dedicated sample set to acquire images.

The next chapter will give a critical review of the research around deficiency identification of greenhouse lettuce.

# Chapter 2 : Literature Review

## 2.1 Introduction

In Chapter 1 an overall picture of the research project was given, by showing the research problem, our hypothesis, and the solution. In this chapter, a critical review of the research is given in the area deficiency identification of greenhouse plants. For this purpose, the chapter has been structured as an early engagement of deficiency identification of lettuce, lettuce in horticulture, disease identification of greenhouse plants, a summary of the latest literature, and the XAI in fiend of research. The chapter also defines the research problem based on the literature review.

## 2.2 Early engagement of deficiency identification of lettuce

As mentioned in the previous section and according to the Medicinal Spices and Vegetables from Africa 2017, lettuce is produced worldwide, and is one of the most consumed green leafy vegetables in its raw form. It is used not only as a salad vegetable but as medicine [23] and research also indicates that lettuce consumption has positive effects on the reduction of cardiovascular disease and chronic conditions due to its rich nutrients such as vitamin A, beta-carotene, folate, and iron content [24]. Because of the importance of this plant, researches are conducted in earlier stages as well.

Identification of nutrient deficiencies of lettuce is discussed in earlier researches using the sand-culture experiments for open-field production. In 1955, Goodall, Grant Lipp, and Slater discussed the application of nitrogen, phosphorus, and potassium fertilizers to lettuce plant soil and monitored the deficiencies of plants for the different levels of nutrients [25]. In the 1970s, lettuce is started to grow in glasshouses where the crop production can be done throughout the year regardless of the seasons. This changes the research approach of deficiency identification. This was discussed by Roorda van Eysinga and Smilde K.W, on their research paper, Nutritional Disorders in Glasshouse Lettuce [26]. These research papers include soil-based lettuce nutrient deficiency and toxicity.

## 2.3 Lettuce in horticulture

With the emerging technology in agriculture and the vast reduction of per capita land availability, lettuce is started to grow in Controlled Environments to enable the productivity of the crop [3], [27]. Controlled Environment Agriculture (CEA) is a combination of science and engineering approaches to overcome the fragility of crop production due to fluctuating open field environments. A modern greenhouse operates as a system; therefore, it is also referred to as a Phytomation system, controlled environment plant production system (CEPPS), or controlled environment agriculture (CEA). This is elaborated by Shamshiri R. and his colleagues in their research paper on urban agriculture [28]. In greenhouses, lettuce is produced using a hydroponic system that offered around 11 times higher yield compared to conventionally produced lettuce. This was identified in research about lettuce production in Yuma, Arizona, USA, and published through "the School of Sustainable Engineering and the Built Environment, Arizona State University" [29].

Hydroponic systems can be developed in different manners, each setup provides a different approach to plant nutrient environments and each setup defines the space around the plant which is crucial in image processing. These different setups are presented by Sardar and Admane in their review paper, "A Review on Plant Without Soil – Hydroponics" [3] and Kaushal Kumar and his colleagues in their journal paper "Hydroponics as an advanced technique for vegetable production: An overview" [30]. This research uses the Ebb and Flow system [30] or Deep Flow Technique [3] to produce lettuce in the greenhouse system. Crop production in the greenhouse highly depends on the greenhouse environment. To keep the environment, crop friendly, the environmental parameters such as air temperature, humidity, and carbon dioxide concentration are monitored and controlled continuously. In addition to this, growing conditions are based on humane observation of the plants. This process is cumbersome and labour intensive and mostly the humans cannot access every place of the greenhouse.

## 2.4 Disease identification of greenhouse plants

There are many solutions to mitigate crop loss due to diseases, but identifying the disease beforehand is a crucial step in this disease management. To overcome this, machine vision is applied to monitor plants. For example, Hetzroni, Miles, Engel, and Hammer describe their neural network-based classifier to determine lettuce deficiencies in Advances in Space Research article in 1994 [31]. It can be identified as the very first article about using deep learning on lettuce deficiency detection. In addition to that David and Murat have presented a system to monitor the real-time plant stress using a computer vision-based multi-sensor platform [32]. Their system is consisting of x y movable sensor system to image acquisition, environmental data collection, and distributed processing hierarchy. This can be used in a small area but will not be cost-effective with a large greenhouse. They have applied their system to monitor calcium deficiency of lettuce and was able to identify the deficiency one day before the human vision [6]. Yara, a mineral fertilizer manufacturing company offers details of open field and CEA crops including lettuce to Identify and diagnose nutrient deficiencies and there are images and information to learn more about the symptoms and causes and how to control or correct the deficiency [33].

Current technological enhancements of IoT devices offer a novel approach in identifying plant diseases using deep learning. Mohanty, Hughes, and Salathé discuss this approach in their research paper on “Using Deep Learning for Image-based Plant Detection” [34]. They have used 54306 images of 14 crop species with 26 diseases to train their deep neural network. These images were taken from the project Plant Village, an open-access image repository created by the same group [35]. This repository does not contain lettuce images hence, the mentioned research does not classify lettuce for their deficiencies. Zheng YY and his colleagues present “the CropDeep species classification and detection dataset” where they have collected 1,147 images from 31 different classes with over 49,000 annotated instances. In the same paper, they have presented the accuracy of different classification models. The best performing model in their results is ResNet50 with 99.81% of average accuracy. This dataset includes four species of lettuce but does not provide any information about deficiencies [36]. Greenhouse lettuce is not researched under deficiency identification using deep leaning and images, but there are few other crops under research. Aravind Krishnaswamy, Raja Purushothaman, and Aniirudh Ramesh have presented their approach on “tomato crop disease classification using pre-trained deep learning algorithm” [37]. They have used the tomato images from the PlantVillage dataset [35] and studied the dataset against pre-trained AlexNet and VGG16net deep learning models. Muammer and Davut have also presented their research on “Plant Disease and Pest Detection using Deep Learning-based Features” with images of plant diseases common to the Malatya, Turkey [38].

## 2.5 A summary of latest literature

The following are the 20 recent literature on deep learning-based disease identification of plants. This does not include lettuce but use similar methods in the field.

*Table 2.1: Summary of resent literature*

| Plant | Training dataset | Environment | Classes | Images | Min-Max per class | C/D | Deep CNN architecture | Training strategy | Evaluation | Best accuracy % |
|---|---|---|---|---|---|---|---|---|---|---|
| "A Deep Learning-based Approach for Banana Leaf Diseases Classification", 2017 [39] | | | | | | | | | | |
| Banana | Own | Uncontrolled | 3 | 3700 (1643, 240, 1817) | 240 - 1813 | C | LeNet [Le89] | FS | 80%, 20% | 98.61 |
| "A Robust Deep-Learning-Based Detector for Real-Time Tomato Plant Diseases and Pests Recognition", 2017 [40] | | | | | | | | | | |
| Tomato | Own | Uncontrolled | 10 | 5000 Annotated - 43398 | 40 – 2177 Annotated – 338 - 18899 | D | AlexNet, ZFNet, GoogleNet, VGG16, ResNet50, 101, ResNetXt-101 | TL | 80%, 10%, 10% | 85.98 |
| "Automated Identification of Northern Leaf Blight-Infected Maize Plants from Field Imagery Using Deep Learning", 2017 [41] | | | | | | | | | | |
| Maize | Own | Uncontrolled | 2 | 1796 | 768–1028 | D | Custom three stages architecture with 5 CNNs | FS | 70%, 15%, 15% | 96.70 |
| "Automatic Image-Based Plant Disease Severity Estimation Using Deep Learning", 2017 [42] | | | | | | | | | | |
| Apple | Plant Village Subset | Controlled | 4 | 2086 | 145-1644 | C | VGG16, VGG19, Inception-V3, ResNet50 | TL | 80%, 20% | 90.40 |
| "Can Deep Learning Identify Tomato Leaf Disease?" 2018 [43] | | | | | | | | | | |
| Tomato | Plant Village Subset | Controlled | 9 | 5550 | 405-814 | C | AlexNet, GoogLeNet, ResNet | TL | 80%, 20% | 97.28 |
| "CropDeep: The Crop Vision Dataset for Deep-Learning-Based Classification and Detection in Precision Agriculture", 2019 [36] | | | | | | | | | | |
| Tomato, cucumber, lettuce, cabbage, turnip, endive, rutabaga, | Crop deep dataset | Controlled | 31 | 31147 Annotated 49765 | 575-1294 Annotated 745-1914 | C & D | VGG16, VGG19, SqueezeNet, InceptionV4, DenseNet121, ResNet18, ResNet50 | FS | 80%, 10%, 10% | 100 with Tomato ResNet |

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| celery, spinach, scallion, fingered citron, winter squash, pumpkin, chili pepper, lemon, persimmon, pawpaw, watermelon, muskmelon, wolfberry | | | | | | | Faster R-CNN, SSD, RFB, YOLOv2, YOLOv3, RetNet | | | |
| "Deep convolutional neural networks for mobile capture device-based crop disease classification in the wild", 2018 [44] | | | | | | | | | | |
| Wheat | Dataset from [45] | Uncontrolled | 4 | 8178 | 1116-3338 | C | Custom ResNet50, ResNet50 | TL | 80%, 10%, 10% | 97.0 |
| "Deep Learning for Classification and Severity Estimation of Coffee Leaf Biotic Stress", 2019 [46] | | | | | | | | | | |
| Coffee | Own | Controlled | 5 | 2722 | 256 - 991 | C | AlexNet, GoogLeNet, VGG16 and ResNet50 | TL | 70%, 15% 15% | 95.63 |
| "Deep Learning for Image-Based Cassava Disease Detection", 2017 [47] | | | | | | | | | | |
| Cassava | Own | Uncontrolled | 6 | 2756 | 309-415 | C | Inception V3 | TL | 80%, 10%, 10% | 93.0 |
| "Deep Learning for Plant Diseases: Detection and Saliency Map Visualization", 2018 [48] | | | | | | | | | | |
| Apple Blueberry Cherry Citrus Grape Peach Pepper Potato | Plant Village dataset | Controlled | 39 | 54323 | 152-5507 | C | AlexNet, DenseNet169, Inception v3, ResNet34, SqueezeNet1-1.1, VGG13 | FS - TL | 80%, 20% | 99.76 |

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Raspberry<br>Soy<br>Strawberry<br>Tomato | | | | | | | | | | |
| “Deep Learning for Tomato Diseases: Classification and Symptoms Visualization”, 2017 [49] | | | | | | | | | | |
| Tomato | Plant<br>Village<br>subset | Controlled | 9 | 14828 | 325-4032 | C | AlexNet,<br>GoogleNet | FS – TL | 80%, 20% | 99.18 |
| “Deep learning models for plant disease detection and diagnosis”, 2018 [50] | | | | | | | | | | |
| Apple<br>Banana<br>Blueberry<br>Cabbage<br>Cantaloupe<br>Cassava<br>Celery<br>Cherry<br>Corn<br>Cucumber<br>Eggplant<br>Gourd<br>Grape<br>Onion<br>Orange<br>Peach<br>Pepper<br>Potato<br>Pumpkin<br>Raspberry<br>Soybean<br>Squash<br>Strawberry<br>Tomato<br>Watermelon | Plant<br>Village<br>Dataset | Both | 58 | 87848 | 43-6235 | C | AlexNet,<br>AlexNetOWTBn,<br>GoogleNet,<br>Overfeat, VGG | Unspecified | 80%, 20% | 99.53 |
| “Deep Neural Networks Based Recognition of Plant Diseases by Leaf Image Classification”, 2016 [51] | | | | | | | | | | |

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Apple<br>Pear<br>Cherry<br>Peach<br>Grapevine | Own Dataset (internet) | Both | 15 | 4483<br>Augmented 33469 | 108-1235<br>1347 - 4854 | C | CaffeNet | TL | 30880, 2589 | 96.3 |
| "Deep Residual Learning for Tomato Plant Leaf Disease Identification", 2017 [52] | | | | | | | | | | |
| Tomato | Plant Village Subset | Controlled | 10 | 19742 | 373-5357 | C | VGG16 VGG19, custom architecture | FS-TL | 80%, 20% | 97.53 |
| "High-Performance Deep Neural Network-Based Tomato Plant Diseases and Pests Diagnosis System With Refinement Filter Bank", 2018 [53] | | | | | | | | | | |
| Tomato | Own dataset field | Uncontrolled | 12 | 8927 | 40 - 3927 | D | Custom architecture with Refinement Filter Bank | TL | 80%, 20% | 96.25 |
| "Identification of Apple Leaf Diseases Based on Deep Convolutional Neural Networks", 2017 [54] | | | | | | | | | | |
| Apple | Own dataset | Both | 4 | 1053<br>Augmented 13689 | 182-319<br>2366 - 4147 | C | AlexNet, GoogleNet, ResNet 20, VGG 16 and custom architecture | FS-TL | 10888, 2801 | 97.62 |
| "Impact of dataset size and variety on the effectiveness of deep learning and transfer learning for plant disease classification", 2018 [55] | | | | | | | | | | |
| Common Bean<br>Coffee<br>Cassava<br>Cashew Tree<br>Citrus<br>Grapevines<br>Coconut tree<br>Soybean<br>Corn<br>Cotton<br>Suarcane<br>Wheat | Own dataset | Both | 56 | 1383 | 5-77 | C | GoogleNet | TL | 80%, 20% | 87.0 |
| "Solving Current Limitations of Deep Learning Based Approaches for Plant Disease Detection", 2019 [56] | | | | | | | | | | |

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Apple<br>Bell pepper<br>Cherry<br>Grape<br>Onion<br>Peach<br>Potato<br>Plum<br>Strawberry<br>Sugar beets<br>Tomato<br>Wheat | Plant Disease Dataset | field | 42 | 79265 | 705 - 3084 | D | Faster R-CNN<br>Faster R-CNN with FPN<br>Faster R-CNN with TDM<br>YOLOv3<br>SSD513<br>RetinaNet<br>PlantDiseaseNet | FS | 70%, 20%, 10% | 93.67 |
| "Soybean Plant Disease Identification Using Convolutional Neural Network", 2018 [57] | | | | | | | | | | |
| Soybean | Plant Village Subset | Uncontrolled | 4 | 12673 | 851- 6234 | C | Custom architecture | FS | 70% 10% 20% | 99.32 |
| "Using Deep Learning for Image-Based Plant Disease Detection", 2016 [34] | | | | | | | | | | |
| Apple<br>Blueberry<br>Cherry<br>Citrus<br>Grape<br>Peach<br>Pepper<br>Potato<br>Raspberry<br>Soy<br>Strawberry<br>Tomato | Plant Village Dataset | Controlled | 38 | 54306 | Not mentioned | C | AlexNet, GoogLeNet | FS-TL | 80% 20% | 99.35 |

*Table 2.2: A summary of above literature for each aspect*

| | |
|---|---|
| Plant | 13/20 (65%) did the research for a single plant.  All 20 (100%) were not generalized |
| Dataset | 10/20 (50%) used their own datasets, 8/20 (40%) used Plant Village dataset, 1 with Crop Deep dataset, and 1 with Plant Disease dataset |
| Environment | 9/20 (45%) used images from controlled environment (homogeneous backgrounds), 6/20 (30%) used images from uncontrolled environment (open field), others (25%) used both |
| Classification/ detection | 15/20 (75%) used classification only, 3/20 (15%) used detection method and others (10%) researched with both |
| Deep CNN Architecture | 17/20 (85%) used popular CNNs, 3/20 (15%) used custom architectures alone, 4/20 (20%) used custom along with popular CNNs |
| Training Strategy | 14/20 (70%) used transfer learning, 10/20 (50%) trained from scratch, 5/20 (25%) used both strategies |
| Training, Evaluation | 8/20 (40%) separated the dataset by training, validation and evaluation. 12/20 (60%) separated only for training and testing. In addition, 2/20 (10%) used entirely different dataset for validation. |

## 2.6 Explainable AI in field of research

In addition to the reasoning mentioned in the problem domain chapter, the following factors motivate XAI research. Those are trust, protections against adversarial techniques, detecting bias, and regulatory compliances [58]. Among these, the most important driver of interest is trust in research in XAI.  To trust an AI model prediction, users need to understand and convinced that the predictions are produced for appropriate and valid reasons. Which seeks explanations. Once the AI models are implemented as products, they must act in accordance with a set of regional, national, or international regulations. AI models must be developed with an explainable way to act in accordance with such regulatory requirements. The ability to generalize is one of the key requirements of an effective AI model. Generalizing means the AI model should be able to perform well on samples that were not included in its training phase. In the training phase, models learn the fundamental associative patterns in the training data and rely on spurious associations to yield better performance [59]. Giving explanations to the model predictions gives a method to identify such false relations and gives a better sense of its generalization possibility. The bias of a model can be systematically reinforced with the training period, but XAI can help in identifying the bias of the model [60].

The following figure from DARPA will explain the XAI concept simply.

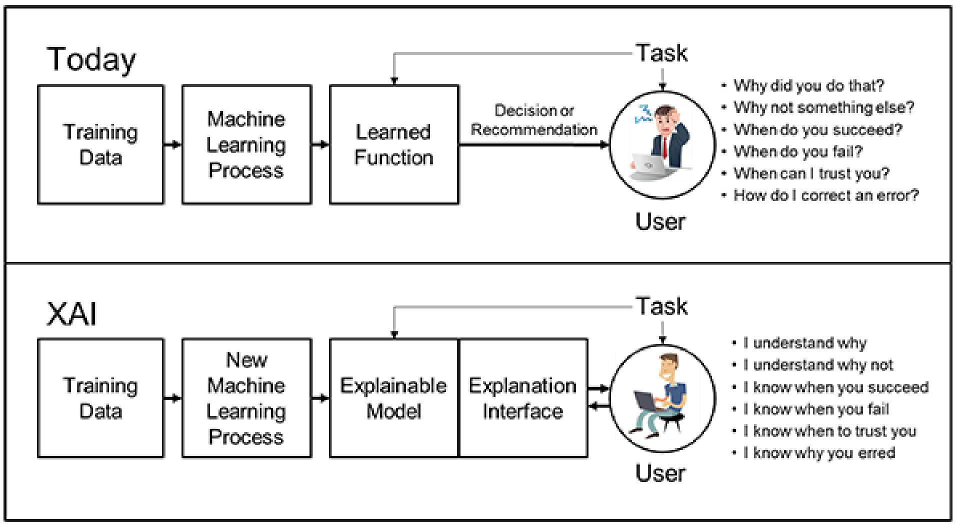


*Figure 2.1: Concept of Explainable Artificial Intelligence [61]*

There are many ways to explain the AI models, but in general, explainable modeling can be applied along with the entire AI development structure. Specifically, it can be applied prior to the model operations (pre-modelling explainability/ante-hoc), during the model operations (explainable modelling), and after the classification/detection model operation (post-modelling explainability/post-hoc) [62]–[65].

There are four main categories of ante-hoc modeling explainability. They are,

1. Exploratory data analysis methods.
2. Dataset description of standardization methods.
3. Explainable feature engineering methods.
4. Dataset summarization methods.

The exploratory data analysis is focused on extracting a summary of the main characteristics of a dataset. This summary includes various statistical properties of the dataset. These statistics include average, mean, range, missing samples, dimensions, etc. Google Facets is one of the powerful toolkits dataset property extraction [66]. But, facts and statistics may not enough for dataset analyzing [67]. Because of that data representation strategies make up a huge amount of the exploratory data analysis methods. Usually, datasets are not released with sufficient documentation. Standardization of these datasets can resolve the issues like the misuse of data or systemic bias in AI models and ensure communication between the users and creators. There are few methods for this standardization including, datasheets for datasets [68], data statements [69], and so on. Explainable feature engineering involves understanding the relationship between and the importance of input features for given model predictions. Domain specific and model based are the two main approaches to achieve this [70]. Domain expert's knowledge and information identified from exploratory data analysis are used in the domain-specific approaches to identify and/or extract features. In contrast, various mathematical models are applied in model-based feature engineering approaches to understand the architecture of the dataset. Pre-modeling explainability or ante-hoc is a set of methodologies to understand the given dataset better before the modeling task.

Explainable modeling can be achieved from the beginning of the model design so that the model can avoid the black box problem and explain itself. This can be achieved by restricting the AI model design to a specific set of models. The traditional way is to adopt from a specific AI model family which is considered as explainable. This family of models often provide one (or more) of the three levels, namely, simulatability, decomposability, and algorithmic transparency, of model transparency [71] proposed by Zack Lipton. Linear models, generalized additive models, rule sets, decision sets, decision trees, and case-based reasoning methods are examples of these families. But simply adopting these families does not guarantee the explainability. And also inheriting these models can reduce the performance of the model itself in some cases [70].

By combining these traditional methods with a complex black-box method, researchers were able to develop hybrid models. The deep k-Nearest Neighbors (DkNN) approach is one of the examples. It proposes to use K-nearest neighbor (kNN) inference on the hidden representation of the training dataset learned through layers of a deep network [72]. The Self-Explaining Neural Network (SENN) is another example introduced by David Alvarez-Melis, Tommi S. Jaakkola [73]. The SENN is an AI model that is trained to provide a prediction along with the corresponding explanation. But there are several limitations to these approaches. Multimodal Explanations: Justifying Decisions and Pointing to the Evidence [74] by Dong Huk Park and his colleagues requires a training dataset augmented with both visual and textual explanations. But the same experiment shows that usage of multi-modal explanations improves predictive performance. TED by [75] Michael Hind and colleagues is somewhat similar to the Multimodal Explanations. One of the limitations of the above methods is that they assume the dataset is containing explainability and the model prediction explanations are more towards human understandability.

The explainability of the model can be also achieved by adjusting the architecture of the AI model. This method is mostly focused on deep network architectures. For example, the explainable convolutional neural network architecture developed by Quanshi Zhang and the team can automatically push representations of higher layer filters to be an object part, as opposed to a mixture of patterns [76]. "This Looks Like That" is another example of developing a deep network explainable [77]. It is developed by adding a prototype layer between convolutional layers. A preset number of image part prototypes for each class are included in the prototype layer. The most relevant parts are captured at each class specific prototype or semantic concepts for identifying images of a given class are captured at each class specific prototype. Another method of developing a self-explainable AI model is to use regularization. Saliency Learning [78] by Reza Ghaeini aims to teach the model to make the right prediction for the right reason. He provides explanation training and ensures that the alignment of the model's explanation is with the ground truth explanation. Similarly, Training Differentiable Models by Constraining their Explanations [79] will match domain knowledge during training so the model predictions will explainable. This approach generalizes much better according to the experimental results.

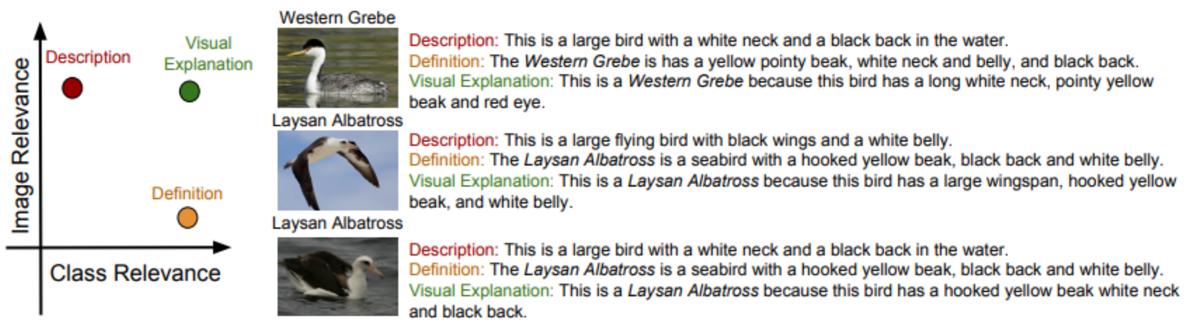
Currently, the AI models are developed focusing on the prediction performance rather than the explainability of the model. Because of that, pre-developed models are much focused on XAI literature. Because of that, many post-hoc methods are developed. The Local Interpretable Model-agnostic Explanations (LIME) [59] approach explains for an instance prediction of a model based on the target, the drivers, the explanation family, and the estimator. The following paragraphs will discuss this breakdown of post-hoc Explainability in detail.

The target specifies the objective of an explainability method. The type, scope, and complexity of the target can be different. The target type can be mechanistic, which is used by the model creators to understand the model predictions to debug or validate the model [80] or the target can be functional, which is used by the outsiders or non-experts to understand the model prediction [59]. The scope of the target can be a local prediction [59] otherwise a global prediction [81]. Explaining prediction for an instance of a class versus all instances of a class.

Input features of an AI model are the most common type of drivers to an explanation. But the raw input features are not the best but aggregated can be. For example, in an image classifier model in this research prediction explanation can be difficult to interpret or expensive to compute if they are based on individual pixels. But if the explanation drives on superpixels (a contiguous patch of similar pixels) [59] it can be more interpretable and less noisy. The explanation drivers include the input features and all other factors with an impact on the AI model development. Those factors include training samples, the choice of model architecture, choice of the optimization algorithm, or hyperparameter settings.

As in explainable model design with explainable families, they can also be used as a post-hoc explanation. The main objective of an explanation family is its information content can be easily interpretable by the use and the explanations should be complete. There are a number of explanation families which can be used in post-hoc explanation modeling. Saliency heatmaps (importance scores) are the most common type of explanation families. The more impactful drivers have a higher score on each explanation. Another common explanation family is decision rules. “if condition then outcome” is the general form of the decision rule. Here, the condition is a simple function defined over input features and the outcome represents a prediction of an AI model. Decision lists and decision sets are two types of decision rules where they are ordered and unordered [82]. Decision trees are quite similar to the decision rules where It can be generalized into a set of decision rules. Decision trees are structured as a graph with internal nodes representing conditional tests on input features and leaf nodes representing the model outcomes. In addition, there can be only one path from the root to leaf in a decision, where in decision rules not the case. Dependency plots are another explanation family aiming to communicate how the prediction depends on the input.

One of the most user-friendly explanation families are the verbal explanations. In this method, explanations are similar to humane explanations and provided in natural language. Template-based approaches are used in the beginning hence it is restricted [83]. Newer methods like Generating Visual Explanations by Lisa Anne and colleagues based on deep learning methods can generate textual augmented visual justifications or explanations [84]. The limitations of this method are lack of understandability in model errors and the explanation is based on the model’s internal logic. The smallest change to explanation drivers required to change a target to a predefined outcome is described as counterfactual explanations. A loss function can be defined to favor the smallest change to make model predictions closer to the desired outcome with few input features to generate counterfactual explanations.



Description: This is a large bird with a white neck and a black back in the water.
Definition: The *Western Grebe* is has a yellow pointy beak, white neck and belly, and black back.
Visual Explanation: This is a *Western Grebe* because this bird has a long white neck, pointy yellow beak and red eye.

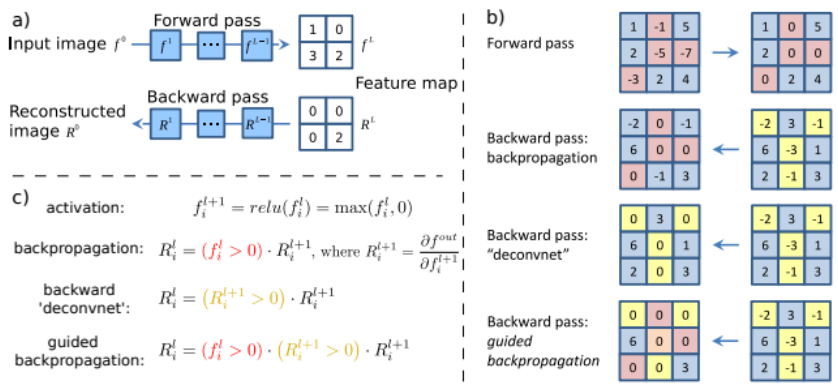
Description: This is a large flying bird with black wings and a white belly.
Definition: The *Laysan Albatross* is a seabird with a hooked yellow beak, black back and white belly.
Visual Explanation: This is a *Laysan Albatross* because this bird has a large wingspan, hooked yellow beak, and white belly.

Description: This is a large bird with a white neck and a black back in the water.
Definition: The *Laysan Albatross* is a seabird with a hooked yellow beak, black back and white belly.
Visual Explanation: This is a *Laysan Albatross* because this bird has a hooked yellow beak white neck and black back.

*Figure 2.2: Template based model generates image relevant and class relevant explanations. The descriptions are image relevant, and definitions are class relevant. [84]*

Explanation estimation can be mainly described based on their model applicability, and the underlying mechanism. Some estimation methods are developed for a specific model architecture, whereas others are developed generally and can be applied to any back-box model. For example, the LIME method can be applied to any model if a set of meaningful perturbations of inputs can be constructed at least in theory. But some of the methods are model-specific. They are both popular and difficult to understand hence targeted towards deep neural networks. The four major underline mechanisms are perturbation, proxy, backward propagation, and activation optimization. The idea of perturbation mechanisms is as follows. It will generate perturbations of desired explanation drivers. Then it will analyze their impact on the given target. Finally, it will summarize it using an importance score family of explanation. There are main two advantages to this method. They are being not limited to specific model architecture and easy to implement. In contrast, the main disadvantage is they are relatively computationally expensive [85] and it is a challenge to construct meaningful perturbations of drivers. Another commonly used mechanism to generate post-hoc explanations for deep network models is the backward propagation. The explanation results are important scores in terms of input features of the model. This method starts with the final layer which produces the given target and estimates the contribution of the previous layer neurons. This process is repeated backward until it reaches the input layer. Some of the most notable backward propagation methods are, Guided Backprop (GB) [86], DeepLIFT [87], and Integrated Gradients (IG) [88]. The main difference between these methods is the method of estimation of the previous layer contribution.

The proxy mechanism is another post-hoc explanation method applied to replace the complex structure of deep neural networks with more simple and explainable methods introduces in later



a)
Forward pass
Input image $f^0$
$f^1$
$f^{L-1}$
$f^L$
Feature map
Backward pass
Reconstructed image $R^0$
$R^1$
$R^{L-1}$
$R^L$
c)
activation: $f_i^{l+1} = relu(f_i^l) = \max(f_i^l, 0)$
backpropagation: $R_i^l = (f_i^l > 0) \cdot R_i^{l+1}$, where $R_i^{l+1} = \frac{\partial f^{out}}{\partial f_i^{l+1}}$
backward 'deconvnet': $R_i^l = (R_i^{l+1} > 0) \cdot R_i^{l+1}$
guided backpropagation: $R_i^l = (f_i^l > 0) \cdot (R_i^{l+1} > 0) \cdot R_i^{l+1}$
b)
Forward pass
Backward pass: backpropagation
Backward pass: "deconvnet"
Backward pass: guided backpropagation

Figure 2.3: *The activations of high layer neurons in Guided Backpropagation. a) Given an input image, the forward pass is performed to an interested layer, then set to zero all activations except one and propagate back to the image to get a reconstruction. b) Different methods of propagating back through a ReLU nonlinearity. c) A Formal definition of different methods for propagating a output activation out back through a ReLU unit in layer 1* [86].

sections such as decision rules and decision trees. Even though the decision trees are simple and explainable, when they are generated based on a deep neural network, they can be large, hence not explainable. The activation optimization mechanism is mostly in deep models' inner functionality to generate explanations. Explanations from this mechanism are obtained by searching for an input pattern that produces maximum or minimum response for an inner component of a model as the target. The downside of this is that the input patterns returned with this mechanism have high frequency noise. This can be overcome to an extent by adding regularization.

Following is a summary of each modeling stage.

Pre-modelling explainability/Ante-hoc [63]:

Goal: Describe or understand the data used to develop AI models

Methods:

- Exploratory data analysis
- Dataset description standardization
- Dataset summarization
- Explainable feature engineering

Explainable modeling [64]:

Goal: Develop AI models which are explainable within themselves

Methods:

- Adopt explainable model family
- Hybrid models
- Joint prediction and explanation
- Architectural adjustments
- Regularization

Post-modelling explainability/Post-hoc [65]

Goal: Understand or describe the output of pre-developed AI models

Methods:

- Perturbation mechanism
- Backward propagation
- Proxy models
- Activation optimization

## 2.7 Summary

This chapter presented a critical review of the research around deficiency identification of greenhouse plants. After looking at the mentioned studies the most appropriate approach to deficiency identification with explanation is the object detection method. Among all the object detection architectures used in the above studies, the most suitable and most promising architecture is You Only Look Once (YOLO). The next chapter clarifies the means followed in recognizing and applying the best strategies to address this study.

# Chapter 3 : Methodology

## 3.1 Introduction

In Chapter 2 an overall picture of the research related literature was given, by giving the data about the early engagement of deficiency identification of lettuce, lettuce in horticulture, disease identification of greenhouse plants, and a summary of the latest literature. The research methodology specifies the procedure used to achieve the research goals with the knowledge gain from the literature review. This chapter provides a comprehensive overview of the methodology behind this research by discussing the dataset creation, algorithm, implementation, etc.

### 3.1.1 Problem Representation

The goal of this study is to identify a solution to detect the nutrient deficiencies of greenhouse lettuce using image processing methods in real time and give explanations. The solution is carried out in three phases. The first phase was creating testbed and the dataset from greenhouse images in collaboration with the greenhouse of AIGrow. The second phase is training the detector to detect nutrient deficiencies using the collected dataset and evaluating the results against the training data. The Final phase is to make the solution to provide explanations to the detector results.

## 3.2 Dataset Creation

The common requirement of neural network related studies is the dataset. The dataset is depending on the area of the study. In this case, the dataset is plant related. There are many accustomed datasets for the plant related studies. As displayed in the literature review section, PlantVillage dataset [89] is vastly used in vision studies related to plants. Table 3.1 displays the most popular plant image datasets.

*Table 3.1: A summary of common and general image datasets for plants [43]*

| Dataset | Classes | Image Number | Annotation Samples Number | Comment |
|---|---|---|---|---|
| Flowers 102 | 102 | 1020 | 0 | Commonly occurring in the United Kingdom are chosen. 40 and 258 images are in each class. |
| PlantVillage | 38 | 19,298 | 0 | Images of previously cropped leaves in the field and captured by a camera in the laboratory |
| CUB 200-2011 | 200 | 5994 | 0 | General-purpose dataset (not only plants) mostly web crawled data |

| | | | | |
|---|---|---|---|---|
| Urban Trees | 18 | 14,572 | 0 | Trees labeled by geo-location and tree species located within Pasadena. The dataset includes dense aerial and street view imagery |
| LeafSnap | 185 | 30,866 | 0 | high-quality images taken of pressed leaves; "typical" images taken by mobile devices (iPhones mostly) in outdoor environments |
| ImageNet | 1000 | 14,197,122 | 1,034,908 | general-purpose dataset (not only plants) mostly web crawled data |
| MS-COCO | 80 | 300,000 | More than 2,000,000 | general-purpose dataset (not only plants) mostly web crawled data |
| AI Challenge | 61 | 47,393 | 0 | general-purpose dataset (not only plants) mostly web crawled data |
| iNat2017 | 5089 | 858,184 | 561,767 | general-purpose dataset (not only plants) mostly web crawled data |
| VegFru | 70 | 160,731 | 0 | vegetables and fruits which has a strong association with the daily life |
| CropDeep | 31 | 31,147 | 49,765 | Contains images collected using various devices including cameras of IoT, autonomous spray robot, autonomous pinking robot and also mobile cameras and smartphones in an intelligent agricultural monitoring and management platform |

Hence the study is focused on greenhouse lettuce, it is difficult to use the above datasets because none of the open access datasets contain lettuce. Hence there is no acceptable dataset (image set) available for lettuce deficiency identification, it is needed to create a suitable dataset (image set) with greenhouse lettuce deficiencies.

### 3.2.1 Experimental Setup and Image Acquisition

The lettuce production system was constructed in the AIGrow [22] research greenhouse located at the Trace Expert City. The research greenhouse dimensions are 16m L x 8m W x 3.4m H. The ridge height is 5.7m. The greenhouse is covered with a double polycarbonate glazing and equipped with a Pad and Fan evaporative cooling system. The climate of the greenhouse

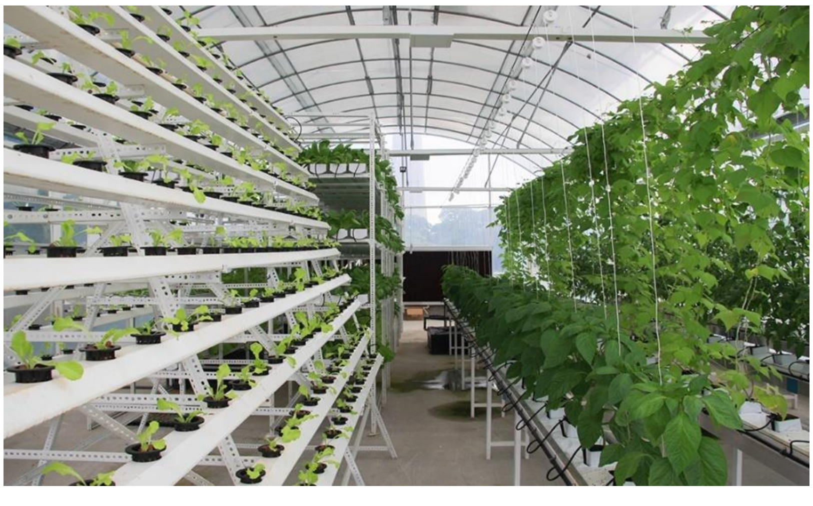


*Figure 3.1: AIGrow Greenhouse at Trace Expert City*

environment is maintained by an automatic climate control system. Figure 3.1 shows an inside image of the greenhouse.

As discussed earlier, the experimental setup was created using the deep flow hydroponic technique. It is one of the "Liquid Hydroponics' method". Lettuce cultivated in a solution culture has its roots immersed in a nutrient solution [3]. The experiment consisted of 28 containers split into 4 groups or flows. Each flow held 7 lettuce plants. Three of the flows were containing treatment nutrient solutions and they are deficient in Calcium, Nitrogen, and Potassium and the final flow is controlled. Figure 3.2 shows the experimental setup.

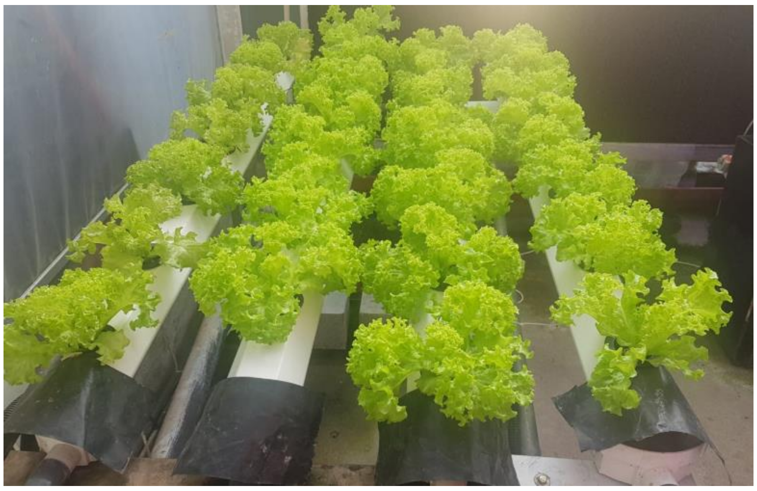


*Figure 3.2: Experimental setup with 3 deficient nutrients and controlled*

Initially, all 28 lettuce plants had the control nutrient solution in the root level for 15 days. Then the deficient solutions were induced through the flows in the treated plants. The experiment is carried out until all the plants show deficiencies. The image acquisition happened once a day at 8.30 am using a handheld Canon EOS 6D Full Frame DSLR Digital Camera under the greenhouse lighting conditions. Figure 3.3 shows an image of calcium deficient lettuce plant.

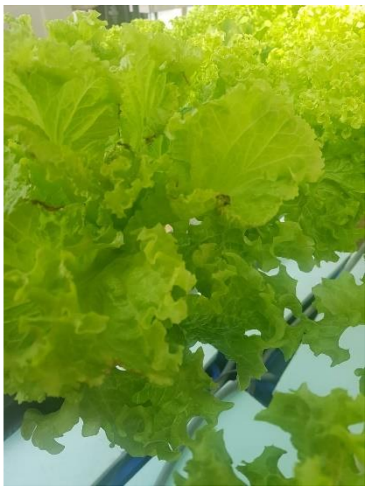


*Figure 3.3: Calcium deficient lettuce plant*

At this point, the tip burn of calcium deficiency is clearly visible. The created image dataset consists with 4 main classes. This images collection will be referred as initial image data set in coming sections. Table 3.2 shows the initial dataset image classes and umber of images in each class. Each image is 3456 x 2304 pixels and had 72 dpi horizontal and vertical resolution.

*Table 3.2: Initial image dataset*

| **Class** | **Number of Images** |
|---|---|
| Controlled (Healthy) | 128 |
| Calcium deficient | 147 |
| Nitrogen deficient | 119 |
| Magnesium deficient | 105 |
| Total Number of Images | 499 |

### 3.2.2 Image Annotation

The next step was the annotation process, which labels location on a deficiency symptom in the image using a bounding box and generate the corresponding class and location information into a text file. In this study, LabelImg [90] was used to annotate and label the images. It is an image annotation tool with a GUI and labels object bounding boxes in images. It is written in Python and uses Qt for its graphical interface and distributed under MIT license. PASCAL VOC format is used to save the annotations as XML files, the same format is used by ImageNet. Other than that, it also supports YOLO format. Following guidelines are used in labeling the image dataset.

1. When an image contains multiple points of deficiencies, each deficiency point should be marked out.
2. When there are overlapped deficiency points in the image, all should be marked and enclosed.
3. Mark deficiencies in an image if it can be identified.

This study follows the LabelImg installation on Windows operating system with Anaconda framework (Figure 3.4). Then the following steps were used to annotate the images.

Step 1: Predefine the image classes as.

Healthy – Class number 0

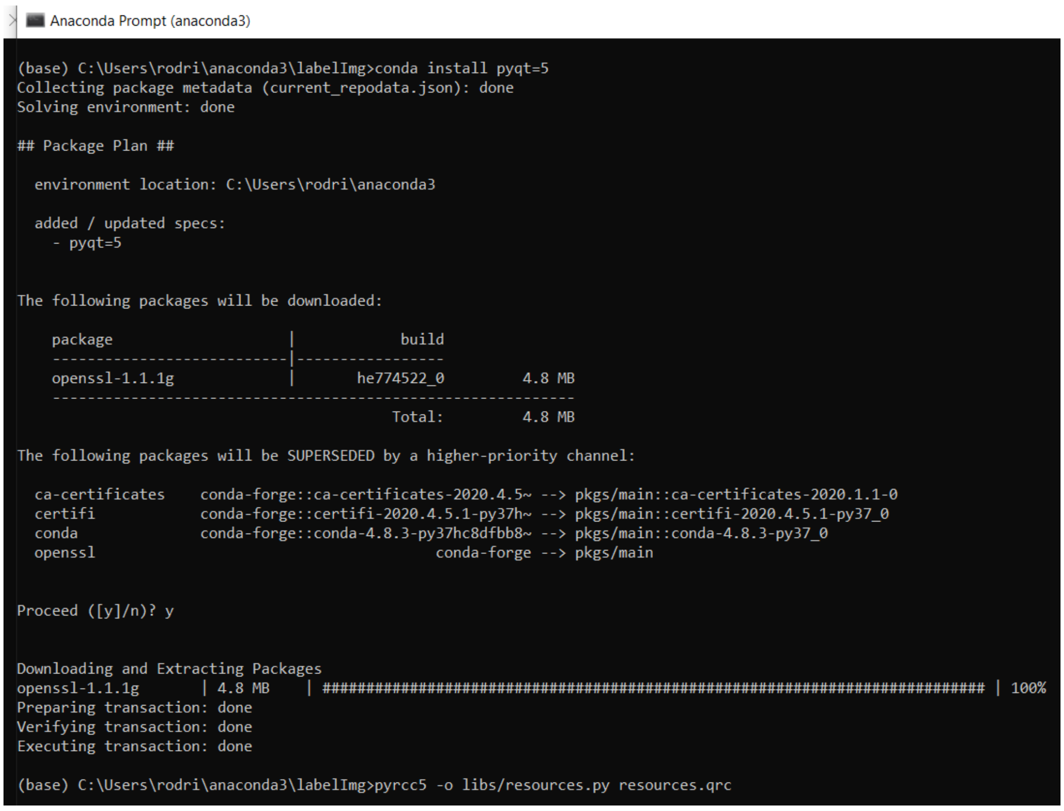
Calcium – Class number 1

Nitrogen – Class number 2

Magnesium – Class number 3

Step 2: Upload images to the running path after resizing them to 72 dpi which will be effectively supported by the YOLOv3 network (Figure 3.5).

Step 3: Build and annotate each point in of the images (Figure 3.6).



```
(base) C:\Users\rodri\anaconda3\labelImg>conda install pyqt=5
Collecting package metadata (current_repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: C:\Users\rodri\anaconda3

  added / updated specs:
    - pyqt=5


The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    openssl-1.1.1g             |       he774522_0         4.8 MB
    ------------------------------------------------------------
                                           Total:         4.8 MB

The following packages will be SUPERSEDED by a higher-priority channel:

  ca-certificates    conda-forge::ca-certificates-2020.4.5~ --> pkgs/main::ca-certificates-2020.1.1-0
  certifi            conda-forge::certifi-2020.4.5.1-py37h~ --> pkgs/main::certifi-2020.4.5.1-py37_0
  conda              conda-forge::conda-4.8.3-py37hc8dfbb8~ --> pkgs/main::conda-4.8.3-py37_0
  openssl                                       conda-forge --> pkgs/main


Proceed ([y]/n)? y


Downloading and Extracting Packages
openssl-1.1.1g       | 4.8 MB    | ############################################################################ | 100%
Preparing transaction: done
Verifying transaction: done
Executing transaction: done

(base) C:\Users\rodri\anaconda3\labelImg>pyrcc5 -o libs/resources.py resources.qrc
```

*Figure 3.4: Installing LabelImg on Windows operating sytem with Anaconda framework*

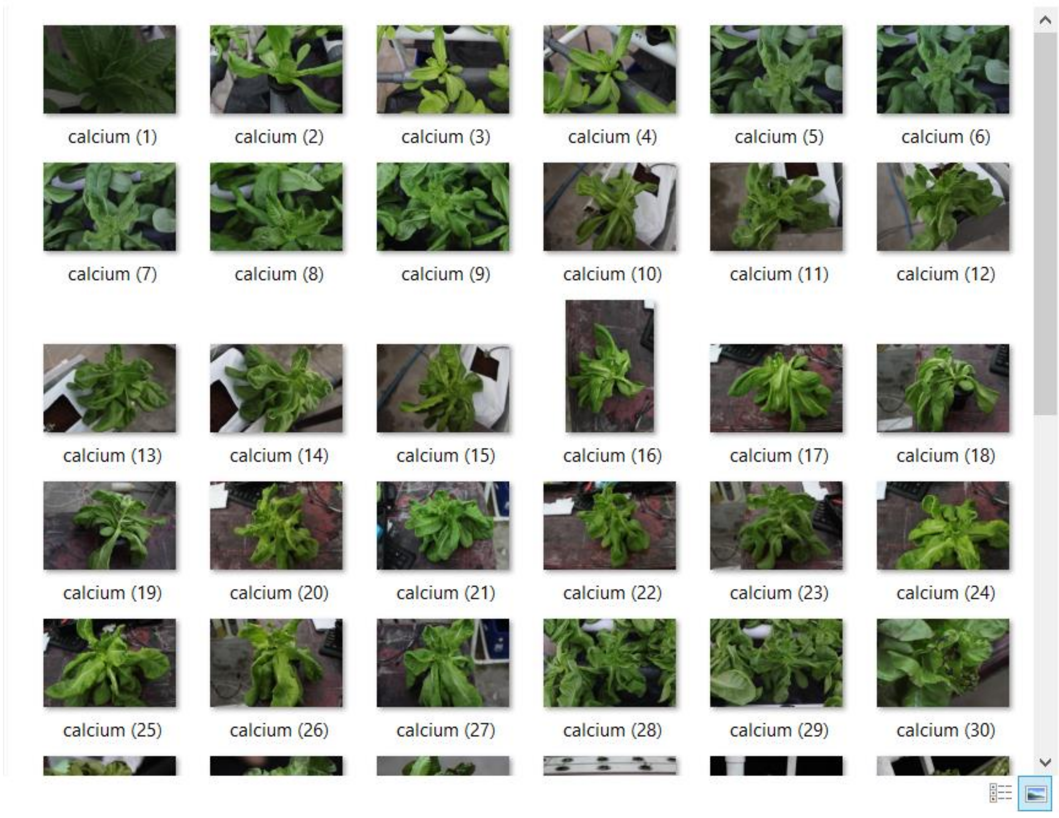


*Figure 3.5: Collection of calcium deficient images prior to annotation process*

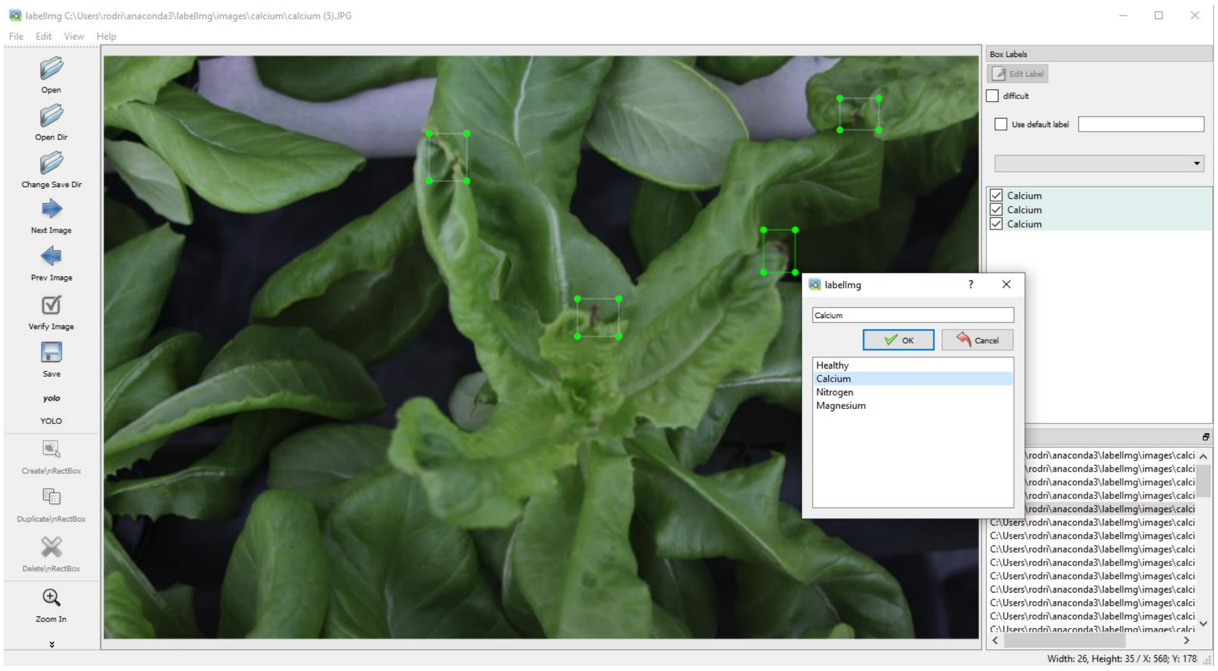
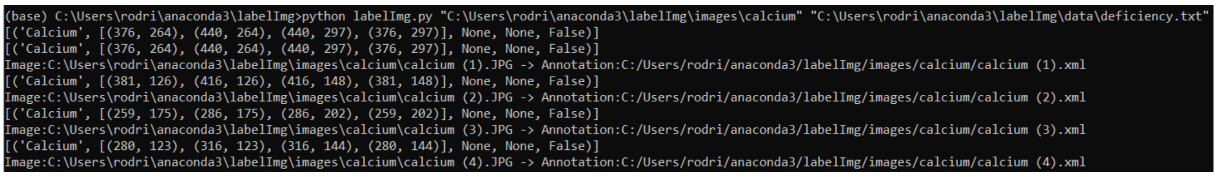


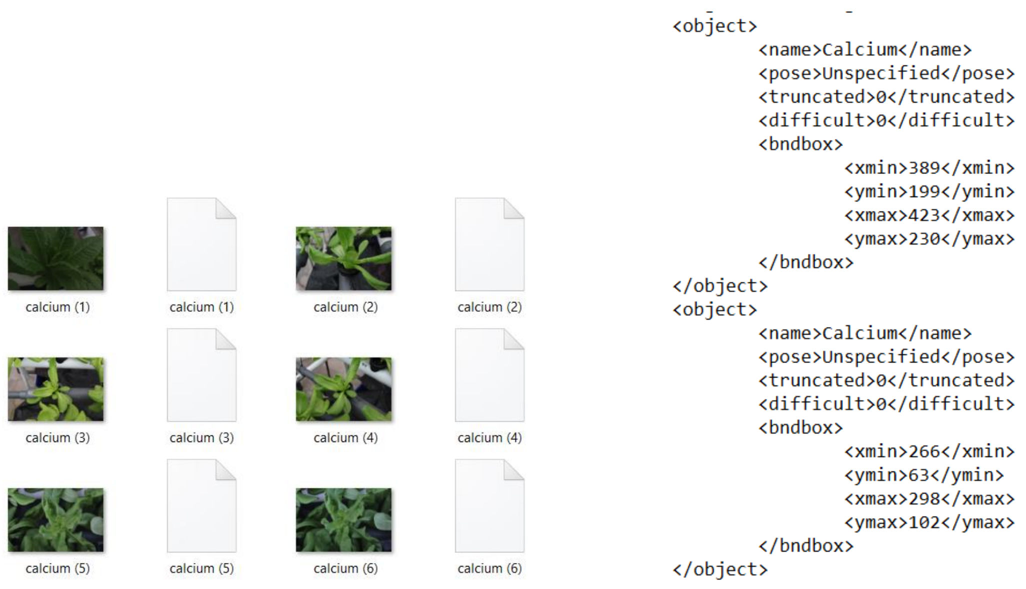
*Figure 3.6: Make bounding boxes on all the spots of deficiency in visible and save the classes using LabelImg tool.*

This creates a list of annotation text files containing the bounding box points (Figure 3.8).



```
(base) C:\Users\rodri\anaconda3\labelImg>python labelImg.py "C:\Users\rodri\anaconda3\labelImg\images\calcium" "C:\Users\rodri\anaconda3\labelImg\data\deficiency.txt"
[('Calcium', [(376, 264), (440, 264), (440, 297), (376, 297)], None, None, False)]
[('Calcium', [(376, 264), (440, 264), (440, 297), (376, 297)], None, None, False)]
Image:C:\Users\rodri\anaconda3\labelImg\images\calcium\calcium (1).JPG -> Annotation:C:/Users/rodri/anaconda3/labelImg/images/calcium/calcium (1).xml
[('Calcium', [(381, 126), (416, 126), (416, 148), (381, 148)], None, None, False)]
Image:C:\Users\rodri\anaconda3\labelImg\images\calcium\calcium (2).JPG -> Annotation:C:/Users/rodri/anaconda3/labelImg/images/calcium/calcium (2).xml
[('Calcium', [(259, 175), (286, 175), (286, 202), (259, 202)], None, None, False)]
Image:C:\Users\rodri\anaconda3\labelImg\images\calcium\calcium (3).JPG -> Annotation:C:/Users/rodri/anaconda3/labelImg/images/calcium/calcium (3).xml
[('Calcium', [(280, 123), (316, 123), (316, 144), (280, 144)], None, None, False)]
Image:C:\Users\rodri\anaconda3\labelImg\images\calcium\calcium (4).JPG -> Annotation:C:/Users/rodri/anaconda3/labelImg/images/calcium/calcium (4).xml
```

*Figure 3.7: Annotation process console output. XML file with the image name is created for each annotated image*



```
<object>
        <name>Calcium</name>
        <pose>Unspecified</pose>
        <truncated>0</truncated>
        <difficult>0</difficult>
        <bndbox>
                <xmin>389</xmin>
                <ymin>199</ymin>
                <xmax>423</xmax>
                <ymax>230</ymax>
        </bndbox>
</object>
<object>
        <name>Calcium</name>
        <pose>Unspecified</pose>
        <truncated>0</truncated>
        <difficult>0</difficult>
        <bndbox>
                <xmin>266</xmin>
                <ymin>63</ymin>
                <xmax>298</xmax>
                <ymax>102</ymax>
        </bndbox>
</object>
```

*Figure 3.8: Left - annotation text files per each image with the same name as image. Right - Annotation file contains the coordinates of each bounding box of the image against the class name*

As shown in the above figure, calcium deficiency is annotated using multiple bounding boxes in the same image. The same goes for the magnesium deficiency, but nitrogen and controlled can only be annotated using a single bounding box. The above annotation files are created as the PascalVOC XML files. Those files will be converted into YOLO text file format in the next steps. The reason for creating the initial annotation files in PascalVOC format is that they can be easily augmented using the existing tools.

*Table 3.3: Number of annotated samples*

| **Class** | **Number of Annotated Samples** |
|---|---|
| Controlled (Healthy) | 128 |
| Calcium deficient | 357 |
| Nitrogen deficient | 119 |
| Magnesium deficient | 173 |
| Total Number of Images | 777 |

### 3.2.3 Dataset Separation

After the annotation and before the augmentation process, the dataset is partitioned into two parts; the train, and test sets, based on the deep learning methods. This is also highlighted in the literature review section. Most of the experiments presented in that section were used only two sets, so this study also uses to data set as training and testing. Hence the number of images in each class is different, the following percentages are used in the partitioning process. 80% of each class as a training set and 20% remaining of each class is for testing. These empirical proportions compensate for the dataset imbalance problem. This step is placed in between the annotation and augmentation steps to avoid placing the same image in both data sets.

*Table 3.4: Separated image dataset into two subsets: training and testing*

| Class | # of Images | |
|---|---|---|
| | Training | Testing |
| Healthy | 102 | 26 |
| Calcium deficient | 117 | 30 |
| Nitrogen deficient | 95 | 24 |
| Magnesium deficient | 84 | 21 |
| Total | 398 | 101 |

### 3.2.4 Image Augmentation

Once the annotation task was completed, the images were subjected to the augmentation process. The augmentation is required because the performance of the deep learning task is highly dependent on the data volume. However, the images are limited in this study and have only around 500 images. Augmentation techniques are used to increase the number of images artificially. In this study, an opensource git hub library, imgaug [91] was used to augment the images with the bounding boxes. The advantage of this library is that it creates the annotation XML for the augmented image. Following python script based on the above library is used for augmentations. Following augmentation techniques are applied against the images with a probability of 0.5 if not mention as all images.

- 50% of all images - Horizontally flip
- 20% of all images - Vertically flip
- Crop by -5% to 10% of image height/width
- Scale to 80 to 120% of image size, individually per axis
- Translate images by -20 to +20 percent

- Rotate images by -45 to +45 degrees
- Shear images by -16 to +16 degrees
- Execute 0 to 5 of the following (less important) augmenters per image
    - Convert images into their superpixel representation
    - Sharpen images
    - Emboss images
    - Add gaussian noise to images

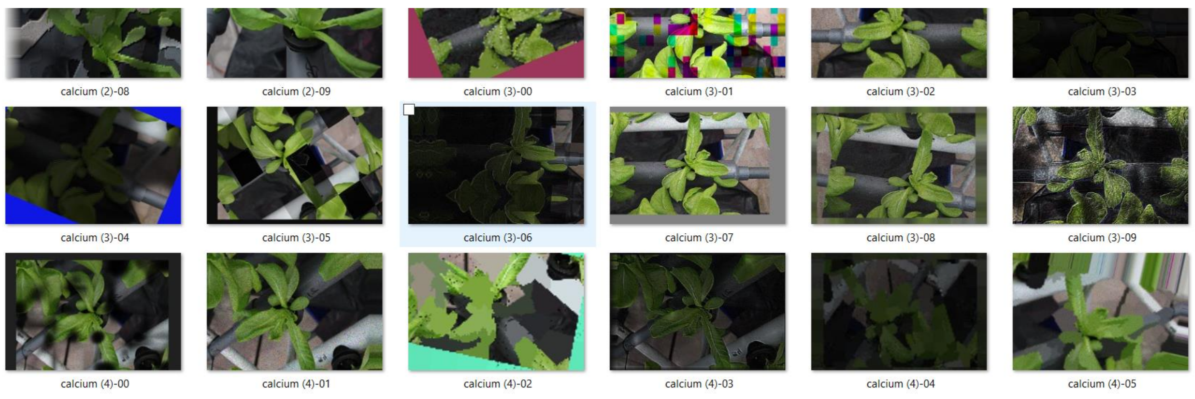


*Figure 3.9: Augmented images using bounding-box-augmentation [92] based on imgaug [91] python library. Each image generates maximum of 10 augmentations according to the above techniques*

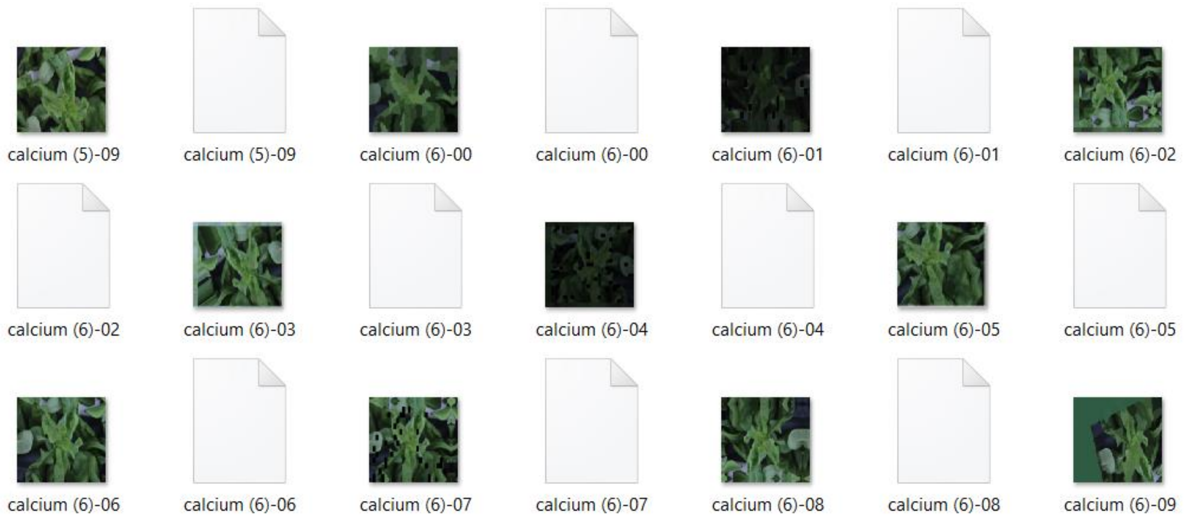


*Figure 3.10: Annotation files are also generated with the augmentation process for each image with the same format*

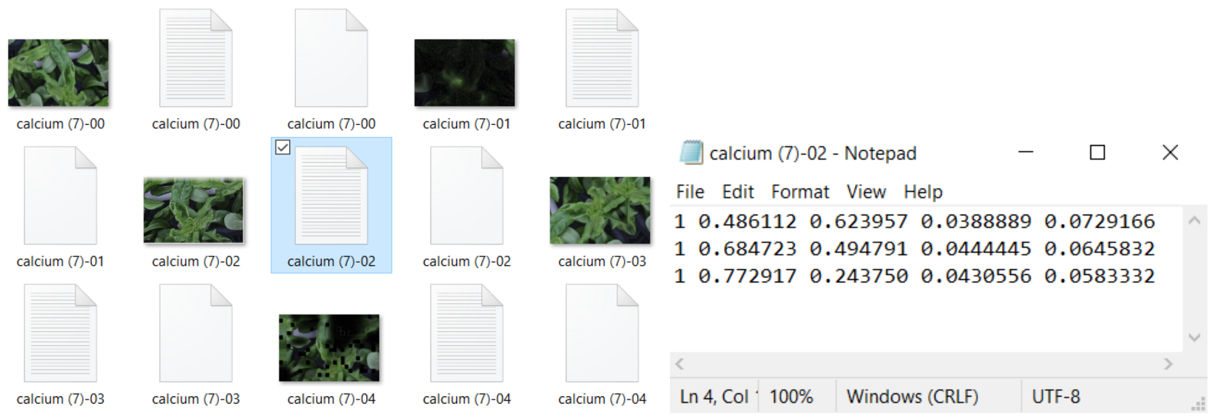
Hence YOLO is not supporting the PascalVOC Annotation XML files, the files are converted into YOLO text file using the following python script. This code is created using "Joseph Redmon's voc_label.py" [92] and Muhammad Younus's "convert_voc_to_yolo.py" [93].

```
import glob
import os
from os import listdir, getcwd
import convert as con

dirs = ['train','test']
classes = ['Healthy','Calcium','Nitrogen','Magnesium']

def getImagesInDir(dir_path):
    image_list = []
    for filename in glob.glob(dir_path + '/*.jpg'):
        image_list.append(filename)
    return image_list

cwd = getcwd()

for dir_path in dirs:
    full_dir_path = cwd + '/' + dir_path
    output_path = full_dir_path +'/'

    if not os.path.exists(output_path):
        os.makedirs(output_path)

    image_paths = getImagesInDir(full_dir_path)
    list_file = open(full_dir_path + '.txt', 'w')

    for image_path in image_paths:
        list_file.write(image_path + '\n')
        con.convert_annotation(full_dir_path, output_path, image_path,
classes)
    list_file.close()

    print("Finished processing: " + dir_path)
```

*Figure 3.11: Main python script, voc_to_yolo.py*

```python
import os
import xml.etree.ElementTree as ET
from os.path import join
from decimal import *

def convert(size, box):
    dw = Decimal(1)/(size[0])
    dh = Decimal(1)/(size[1])
    x = (box[0] + box[1])/Decimal(2) - 1
    y = (box[2] + box[3])/Decimal(2) - 1
    w = box[1] - box[0]
    h = box[3] - box[2]
    x = x*dw
    w = w*dw
    y = y*dh
    h = h*dh
    return (x,y,w,h)

def convert_annotation(dir_path, output_path, image_path, classes):
    basename = os.path.basename(image_path)
    basename_no_ext = os.path.splitext(basename)[0]

    in_file = open(dir_path + '/' + basename_no_ext + '.xml')
    out_file = open(output_path + basename_no_ext + '.txt', 'w')
    tree = ET.parse(in_file)
    root = tree.getroot()
    size = root.find('size')
    getcontext().prec = 6
    w = Decimal(size.find('width').text)
    h = Decimal(size.find('height').text)

    for obj in root.iter('object'):
        cls = obj.find('name').text
        if cls not in classes:
            continue
        cls_id = classes.index(cls)
        xmlbox = obj.find('bndbox')
        b = (Decimal(xmlbox.find('xmin').text),
Decimal(xmlbox.find('xmax').text), Decimal(xmlbox.find('ymin').text),
Decimal(xmlbox.find('ymax').text))
        bb = convert((w,h), b)
        out_file.write(str(cls_id) + " " + " ".join([str(a) for a in bb]) +
'\n')
```

*Figure 3.12: Python script with conversion functions, convert.py*



```
1 0.486112 0.623957 0.0388889 0.0729166
1 0.684723 0.494791 0.0444445 0.0645832
1 0.772917 0.243750 0.0430556 0.0583332
```

*Figure 3.13: Left - newly created annotation text files in the same directory as images and XML files. Right - PascalVOC XML is converted into YOLO text format (<class> <x_center> <y_center> <width> <height>)*

Following table contains the number of images after going through the annotation and augmentation process.

*Table 3.5: Number of images after augmentation*

| Class | Number of Augmented Images | |
|---|---|---|
| | Training | Testing |
| Healthy | 897 | 224 |
| Calcium deficient | 911 | 228 |
| Nitrogen deficient | 878 | 219 |
| Magnesium deficient | 770 | 193 |
| Total | 3456 | 864 |

## 3.3 Detection Algorithm

As described in the literature review chapter, many classification algorithms and detection algorithms are available for plant deficiency identification. These deep learning algorithms vary from each other based on their basics used in the algorithm design and deficiency identification approach. After the critical review of the literature as presented in the previous chapter, version three of the You only look once, or YOLO object detection algorithm is selected for this study. This section will describe the architecture of the algorithm.

### 3.3.1 YOLOv3 Architecture

Most of the detection systems use classifiers underline to perform detection. these systems take a classifier for that object and evaluate it at various locations and scales in a test image to detect an object. More recent algorithms like R-NN use region proposal methods to generate potential bounding boxes on an image then run the classifier on those bounding boxes. Once the classification is done, post processing refines the bounding boxes. This process is somewhat complex. YOLO solves the object detection problem as a simple regression problem where a single convolutional network simultaneously predicts multiple bounding boxes and class probabilities for those boxes. Few advantages of this system are first, it is fast because of the simple pipeline. Second, when making predictions, it reasons globally about the image. It sees the entire image during training and test time in contrast to sliding window and region proposal-based techniques, so it implicitly encodes contextual information about classes as well as their appearance. Finally, it learns to derive the representations of objects so it is less likely to beak on unexpected inputs [15].

This study is based on the third iteration of YOLO over the years and the underline architecture of it is called Darknet. Initially, YOLOv2 used a 30-layer custom architecture of darknet. YOLOv3 architecture is still missing some of the important elements like residual blocks, skip connections, and upsampling, which are now mandatory in most of the highly used AI algorithms. YOLO v3 corrected those missing elements and uses a variant of Darknet, which initially was a 53-layer network trained on ImageNet. 53 more layers are added onto it for the task of detection resulting a 106-layer fully convolutional underlying architecture.

Following diagram created by Ayoosh Kathuria for his blog post on What's new in YOLO v3 posted on towardsdatascience website.

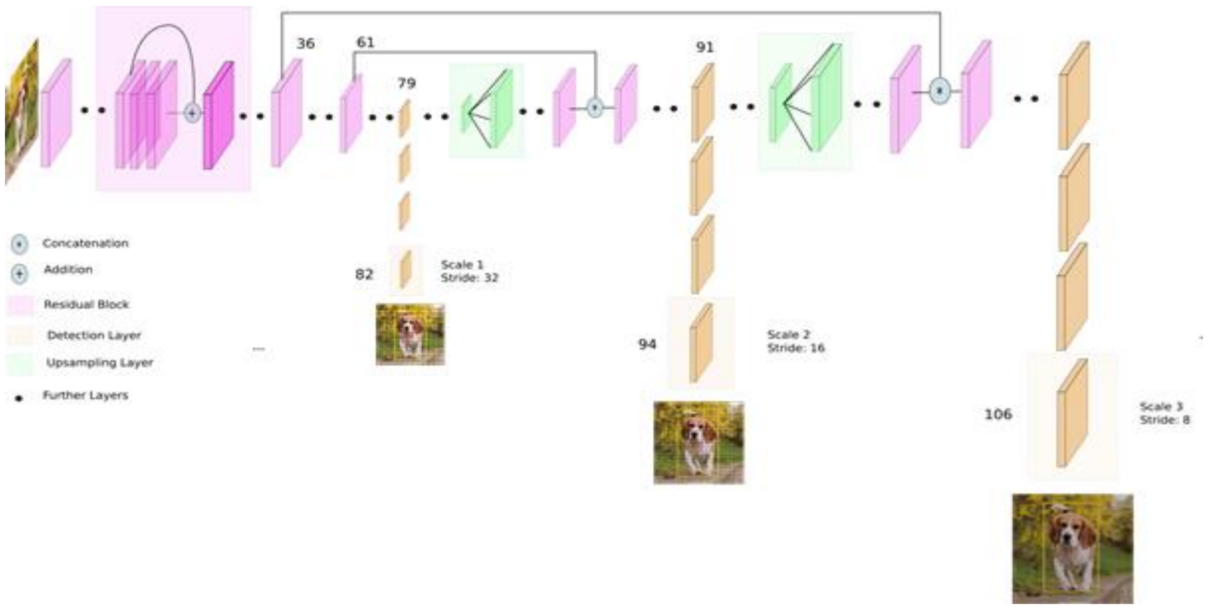


*Figure 3.14: YOLOv3 network architecture*

YOLOv3 architecture is explained in Figure 3.15. (A) model pipeline with an input image size of 416×416 pixels. This returns 3 types of feature maps as outputs. They are 13×13×69, 26×26×69, and 52×52×69 respectively. (B) the basic element of YOLOv3, Darknet conv2D BN Leaky ("DBL" for short), is created with one convolution layer, one batch normalization layer, and one leaky Relu layer. (C) two "DBL" structures following with one "add" layer leads to a residual-like unit ("ResUnit" for short); (D) several "ResUnit" with one zero padding layer and "DBL" structure forward generates a residual-like block, "ResBlock" in short; (E) some detection results with YOLOv3 approach, resize the 732×574 images to 416×416 size as input.

Convolutional layers are the only layers used to make YOLO an FCN (Fully Convolutional Network). Pooling is not used inside the network in any form, and downsampling of the feature maps is also done by using a convolutional layer. Usually pooling leads to a loss of low-level features and the convolutional layer downsampling prevents that. The newer architecture includes upsampling and residual skip connections. The most noticeable or important feature of YOLOv3 is that there are three different scales of detection. YOLOv3 being a fully convolutional network makes it not dependent on the input image size. However, in practice, input images should not be on varying input sizes because various problems that only appear once the algorithm is implemented. As stated earlier, YOLOv3 detects the object by applying a 1×1 detection kernel at three different places in the network on feature maps of three different sizes.

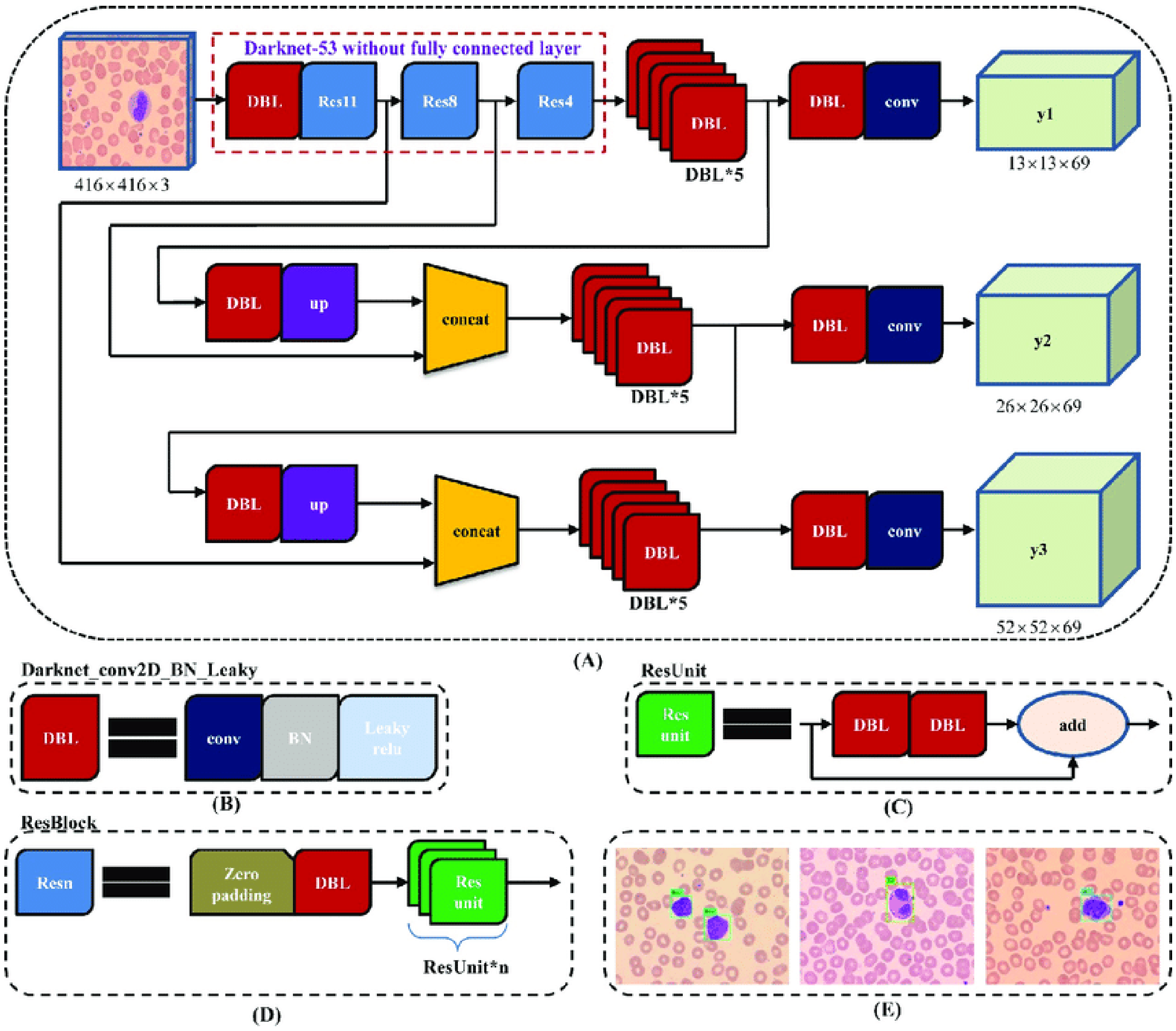


*Figure 3.15: YOLOv3 pipeline architecture*

The composition of the detection kernel is as follows.

$$1 \times 1 \times (B \times (5 + C))$$

In the above algorithm, “B” is the number of bounding boxes can predict on a cell on the feature map, “5” is the summation of the object confidence and 4 bounding box attributes, and “C” indicates the number of classes of the detection problem. The height and width of the feature map produced by this kernel are similar to the previous feature map and detection attributes are placed along with the depth on the newly generated feature map as explained above.

The network downsamples the image using the stride of the network. Stride is the number of pixels shifts over the input matrix. Generally, the outputs of the layers are smaller than the input of the layers in the network and that factor makes stride of any layer in the network. For example, if the stride is 32 in the network, then an input image of size 416×416 will result in a 13×13 output in size. As explained in the previous paragraph YOLO uses a convolutional layer with 1×1 convolution to predict objects. The first output of the layers is a feature map. The size of the final prediction map has the same size as the feature map before the final layer because the 1×1 convolutions are used to predict. In YOLO v3, each cell can predict the same number of bounding boxes using this prediction map. If the center of a detecting object falls in the receptive field of a cell that object is predicted by that cell of the feature map through one of its bounding boxes.

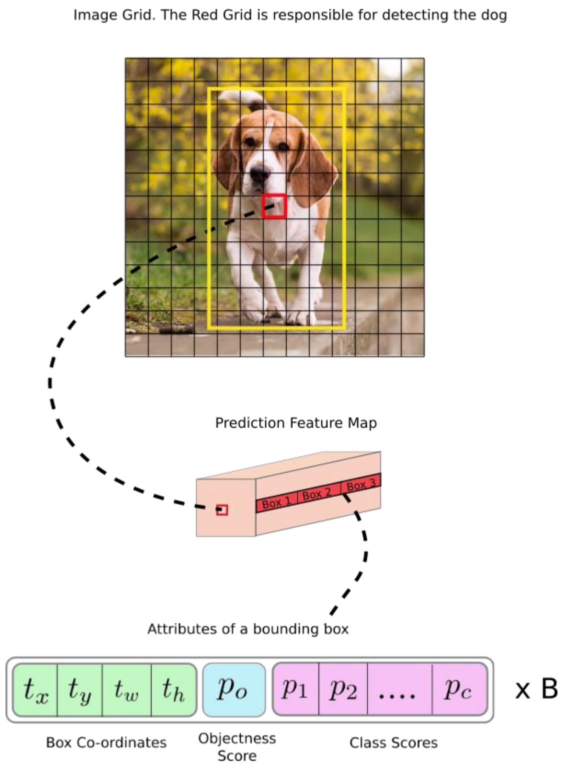


Image Grid. The Red Grid is responsible for detecting the dog
Prediction Feature Map
Box 1
Box 2
Box 3
Attributes of a bounding box
$t_x$ $t_y$ $t_w$ $t_h$ $p_o$ $p_1$ $p_2$ .... $p_c$
x B
Box Co-ordinates
Objectness Score
Class Scores

*Figure 3.16: "The cell (upper image) containing the center of the ground truth box of an object is chosen to be the one responsible for predicting the object. In the image, it is the cell which marked red, which contains the center of the ground truth box (marked yellow)" ~ [94]*

As described earlier the three prediction scales downsample the image to the dimensions of 32, 16, and 8 respectively. Throughout the first 81 layers, network downsamples the image to produce the first detection at 82nd layer by having a stride of 32 at layer 81. As the above example, a 1×1 detection kernel is used to made detection. The resulting feature map has dimensions of 13×13×255. Then, the second detection is made at the 94th layer, yielding a detection feature map with dimensions of 26×26×255. The final of the scales is at the 106th layer, and the resulting feature map has dimensions of 52×52×255.

The result of the detection is a set of bounding boxes around the prediction. Predicting the width and height of the bounding box leads to unstable gradients during training. Because of that, the detectors have a pre-defined default set of bounding boxes called anchors, and the detectors are offset to them. Then the bounding box is predicted by applying the transforms. The following formulae describe how the network output is transformed to obtain bounding box predictions.

$$b_x = \sigma(t_x) + c_x$$

$$b_y = \sigma(t_y) + c_y$$

$$b_w = p_w e^{t_w}$$

$$b_h = p_h e^{t_h}$$

$b_x$, $b_y$, $b_w$, $b_h$ are the x, y center co-ordinates, and the width and height of the prediction. $c_x$ and $c_y$ are the top-left coordinates of the grid. $t_x$, $t_y$, $t_w$, $t_h$ are the network outputs. $p_w$ and $p_h$ are anchors dimensions for the box. The following image shows how to apply a log-space transform to the output to predict the dimensions of the bounding box and then multiplying with an anchor.

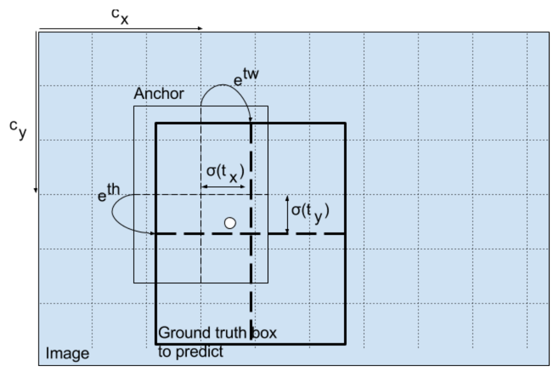


*Figure 3.17: How the final prediction is given by transforming the detector output.*

According to the three scales described in previous paragraph, there can be 10647 bounding box predictions ([13x13] + [26x26] + [52x52] = 10647). YOLO will reduce the boxes using thresholding by object confidence and non-maximum suppression methods [95].

## 3.4 Implementation

Based on the literature review, the implementation of a detection system requires a hardware setup and a set of software and frameworks to support the development.

### 3.4.1 Workstation Setup

This section explains the hardware selection for the study. It is a known fact that the high performing hardware is required for the calculation tasks in Machine Learning Algorithms, libraries.

Most of the common machine learning algorithms including neural networks cannot be parallelly trained. This makes that they have to train on a single processor. Simple MLPs can run in a viable amount of time using a single processor, it is not feasible to train a large Convolutional Neural Network on a personal laptop or even a personal computer with a decent processing unit. Graphical processing units (GPUs) are used to overcome this challenge of training large neural networks. GPUs are good at performing large amounts of simple operations at the same time with their numerous processor cores and they provide the best memory bandwidth while having almost no drawback due to latency via thread parallelism against the CPU. Most of the neural network operations can be identified as simple matrix operations, and GPUs are able to run these small operations parallelly in their thousands of core to faster the training process of the model.

Therefore, the study used a workstation with the specifications that are summarized in the following table.

*Table 3.6: Workstation specification summary*

| | |
|---|---|
| Machine Type | Desktop Workstation |
| CPU | Intel Core i3-7300 2 Cores 4 Threads 4.00 GHz |
| RAM | 16 GB |
| GPU | GeForce GTX 1060 6 GB 1280 NVIDIA CUDA Cores |
| Operating System | Ubuntu 18.04 |

### 3.4.2 Software and Frameworks

The operating system chosen is an Ubuntu 18.04 Linux distribution, as it is freely available and as it is the primary supported operating system of the Darknet and TensorFlow. To utilize the GPU's capabilities, the libraries CUDA 10.0 and cuDNN 7.4 were installed. Before installing the framework OpenCV 3.4.0 was also installed.

There are various deep learning frameworks capable of creating, training, and using YOLOv3. These frameworks vary in speed, abstraction level, modifiability, and creation of new layers. Table 3.7 lists various frameworks along with their programming language. While preferences may vary, web research concludes that there is no single best network. Darknet framework was selected because of its high abstraction level, which makes network architecture definition, solver, and application relatively easy. It is fast, easy to install, and supports CPU and GPU computation [95].

*Table 3.7: List of deep learning frameworks with programming languages which can be used to implement YOLO*

| Framework | Programming Language |
|---|---|
| Darknet | C, CUDA |
| Tensorflow | Python |
| Pytorch | Python, C++, CUDA |
| OpenCV | C/C++ |

When working with Darknet only the configurations are changed and most of the related work like dataset preparation is done using python scripts.

### 3.4.3 Experimental parameter setting

Once the data set is prepared, YOLOv3 configuration files were created namely .data, . names, and .cfg. .data file includes the following content.

```
classes = 4
train = data/train.txt
valid = data/test.txt
names = data/deficiency.names
backup = backup/
```

classes: the number of class in the data set

train: train file path

test: test file path

names: class names file

backup: path to store generated weight files

The deficiency.names looks like as follows. Every category is defined on a new line, and the line number is the category number in label text files created earlier in the dataset preparation.

```
Healthy
Calcium
Nitrogen
Magnesium
```

The yolov3.cfg file from pjreddie.com was used for training configurations which include three yolo layers. As a traditional method, each object is to be trained for at least 2000 iterations. Hence, the dataset was trained for 8000 iterations as there are 4 classes to train. The values of batch and subdivisions were set to 24 and 8 respectively for optimal training speed. The width and height values were set at 416 each for optimum speed and better accuracy of detection. The number of filters used in the convolution layer was set to 27 as the value is dependent on the total number of classes.

batch=24, indicates that for every training step 24 images are used.

subdivisions=8, the batch is divided by 8 to decrease GPU VRAM requirements.

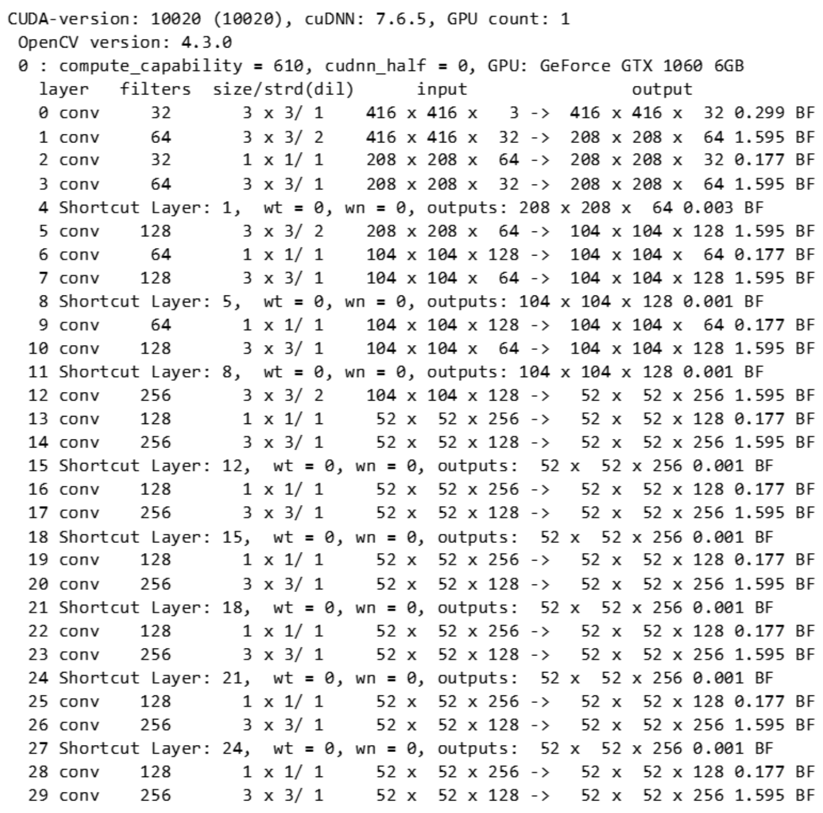
filters=27, this is calculated from $filters = (classes\ +\ 5) \times 3$

classes=4, the number detection categories

max_batches=8000, 2000 iterations per each object over 4 objects

### 3.4.4 Transfer learning

The total amount of time required to train the network with the above configurations was approximately 9-10 hours. The weights thus generated after 8000 iterations were used to carry out detections and analyzing the performance. Once before the training is started, the network architecture was displayed as follows.



```
CUDA-version: 10020 (10020), cuDNN: 7.6.5, GPU count: 1
 OpenCV version: 4.3.0
 0 : compute_capability = 610, cudnn_half = 0, GPU: GeForce GTX 1060 6GB
   layer   filters  size/strd(dil)      input                output
   0 conv     32       3 x 3/ 1    416 x 416 x   3 ->  416 x 416 x  32 0.299 BF
   1 conv     64       3 x 3/ 2    416 x 416 x  32 ->  208 x 208 x  64 1.595 BF
   2 conv     32       1 x 1/ 1    208 x 208 x  64 ->  208 x 208 x  32 0.177 BF
   3 conv     64       3 x 3/ 1    208 x 208 x  32 ->  208 x 208 x  64 1.595 BF
   4 Shortcut Layer: 1,  wt = 0, wn = 0, outputs: 208 x 208 x  64 0.003 BF
   5 conv    128       3 x 3/ 2    208 x 208 x  64 ->  104 x 104 x 128 1.595 BF
   6 conv     64       1 x 1/ 1    104 x 104 x 128 ->  104 x 104 x  64 0.177 BF
   7 conv    128       3 x 3/ 1    104 x 104 x  64 ->  104 x 104 x 128 1.595 BF
   8 Shortcut Layer: 5,  wt = 0, wn = 0, outputs: 104 x 104 x 128 0.001 BF
   9 conv     64       1 x 1/ 1    104 x 104 x 128 ->  104 x 104 x  64 0.177 BF
  10 conv    128       3 x 3/ 1    104 x 104 x  64 ->  104 x 104 x 128 1.595 BF
  11 Shortcut Layer: 8,  wt = 0, wn = 0, outputs: 104 x 104 x 128 0.001 BF
  12 conv    256       3 x 3/ 2    104 x 104 x 128 ->   52 x  52 x 256 1.595 BF
  13 conv    128       1 x 1/ 1     52 x  52 x 256 ->   52 x  52 x 128 0.177 BF
  14 conv    256       3 x 3/ 1     52 x  52 x 128 ->   52 x  52 x 256 1.595 BF
  15 Shortcut Layer: 12,  wt = 0, wn = 0, outputs:  52 x  52 x 256 0.001 BF
  16 conv    128       1 x 1/ 1     52 x  52 x 256 ->   52 x  52 x 128 0.177 BF
  17 conv    256       3 x 3/ 1     52 x  52 x 128 ->   52 x  52 x 256 1.595 BF
  18 Shortcut Layer: 15,  wt = 0, wn = 0, outputs:  52 x  52 x 256 0.001 BF
  19 conv    128       1 x 1/ 1     52 x  52 x 256 ->   52 x  52 x 128 0.177 BF
  20 conv    256       3 x 3/ 1     52 x  52 x 128 ->   52 x  52 x 256 1.595 BF
  21 Shortcut Layer: 18,  wt = 0, wn = 0, outputs:  52 x  52 x 256 0.001 BF
  22 conv    128       1 x 1/ 1     52 x  52 x 256 ->   52 x  52 x 128 0.177 BF
  23 conv    256       3 x 3/ 1     52 x  52 x 128 ->   52 x  52 x 256 1.595 BF
  24 Shortcut Layer: 21,  wt = 0, wn = 0, outputs:  52 x  52 x 256 0.001 BF
  25 conv    128       1 x 1/ 1     52 x  52 x 256 ->   52 x  52 x 128 0.177 BF
  26 conv    256       3 x 3/ 1     52 x  52 x 128 ->   52 x  52 x 256 1.595 BF
  27 Shortcut Layer: 24,  wt = 0, wn = 0, outputs:  52 x  52 x 256 0.001 BF
  28 conv    128       1 x 1/ 1     52 x  52 x 256 ->   52 x  52 x 128 0.177 BF
  29 conv    256       3 x 3/ 1     52 x  52 x 128 ->   52 x  52 x 256 1.595 BF
  30 Shortcut Layer: 27,  wt = 0, wn = 0, outputs:  52 x  52 x 256 0.001 BF
  31 conv    128       1 x 1/ 1     52 x  52 x 256 ->   52 x  52 x 128 0.177 BF
  32 conv    256       3 x 3/ 1     52 x  52 x 128 ->   52 x  52 x 256 1.595 BF
  33 Shortcut Layer: 30,  wt = 0, wn = 0, outputs:  52 x  52 x 256 0.001 BF
  34 conv    128       1 x 1/ 1     52 x  52 x 256 ->   52 x  52 x 128 0.177 BF
  35 conv    256       3 x 3/ 1     52 x  52 x 128 ->   52 x  52 x 256 1.595 BF
  36 Shortcut Layer: 33,  wt = 0, wn = 0, outputs:  52 x  52 x 256 0.001 BF
  37 conv    512       3 x 3/ 2     52 x  52 x 256 ->   26 x  26 x 512 1.595 BF
  38 conv    256       1 x 1/ 1     26 x  26 x 512 ->   26 x  26 x 256 0.177 BF
  39 conv    512       3 x 3/ 1     26 x  26 x 256 ->   26 x  26 x 512 1.595 BF
  40 Shortcut Layer: 37,  wt = 0, wn = 0, outputs:  26 x  26 x 512 0.000 BF
  41 conv    256       1 x 1/ 1     26 x  26 x 512 ->   26 x  26 x 256 0.177 BF
  42 conv    512       3 x 3/ 1     26 x  26 x 256 ->   26 x  26 x 512 1.595 BF
  43 Shortcut Layer: 40,  wt = 0, wn = 0, outputs:  26 x  26 x 512 0.000 BF
  44 conv    256       1 x 1/ 1     26 x  26 x 512 ->   26 x  26 x 256 0.177 BF
  45 conv    512       3 x 3/ 1     26 x  26 x 256 ->   26 x  26 x 512 1.595 BF
  46 Shortcut Layer: 43,  wt = 0, wn = 0, outputs:  26 x  26 x 512 0.000 BF
  47 conv    256       1 x 1/ 1     26 x  26 x 512 ->   26 x  26 x 256 0.177 BF
  48 conv    512       3 x 3/ 1     26 x  26 x 256 ->   26 x  26 x 512 1.595 BF
  49 Shortcut Layer: 46,  wt = 0, wn = 0, outputs:  26 x  26 x 512 0.000 BF
  50 conv    256       1 x 1/ 1     26 x  26 x 512 ->   26 x  26 x 256 0.177 BF
  51 conv    512       3 x 3/ 1     26 x  26 x 256 ->   26 x  26 x 512 1.595 BF
  52 Shortcut Layer: 49,  wt = 0, wn = 0, outputs:  26 x  26 x 512 0.000 BF
  53 conv    256       1 x 1/ 1     26 x  26 x 512 ->   26 x  26 x 256 0.177 BF
  54 conv    512       3 x 3/ 1     26 x  26 x 256 ->   26 x  26 x 512 1.595 BF
  55 Shortcut Layer: 52,  wt = 0, wn = 0, outputs:  26 x  26 x 512 0.000 BF
  56 conv    256       1 x 1/ 1     26 x  26 x 512 ->   26 x  26 x 256 0.177 BF
  57 conv    512       3 x 3/ 1     26 x  26 x 256 ->   26 x  26 x 512 1.595 BF
  58 Shortcut Layer: 55,  wt = 0, wn = 0, outputs:  26 x  26 x 512 0.000 BF
  59 conv    256       1 x 1/ 1     26 x  26 x 512 ->   26 x  26 x 256 0.177 BF
  60 conv    512       3 x 3/ 1     26 x  26 x 256 ->   26 x  26 x 512 1.595 BF
  61 Shortcut Layer: 58,  wt = 0, wn = 0, outputs:  26 x  26 x 512 0.000 BF
  62 conv   1024       3 x 3/ 2     26 x  26 x 512 ->   13 x  13 x1024 1.595 BF
  63 conv    512       1 x 1/ 1     13 x  13 x1024 ->   13 x  13 x 512 0.177 BF
  64 conv   1024       3 x 3/ 1     13 x  13 x 512 ->   13 x  13 x1024 1.595 BF
  65 Shortcut Layer: 62,  wt = 0, wn = 0, outputs:  13 x  13 x1024 0.000 BF
  66 conv    512       1 x 1/ 1     13 x  13 x1024 ->   13 x  13 x 512 0.177 BF
  67 conv   1024       3 x 3/ 1     13 x  13 x 512 ->   13 x  13 x1024 1.595 BF
  68 Shortcut Layer: 65,  wt = 0, wn = 0, outputs:  13 x  13 x1024 0.000 BF
  69 conv    512       1 x 1/ 1     13 x  13 x1024 ->   13 x  13 x 512 0.177 BF
  70 conv   1024       3 x 3/ 1     13 x  13 x 512 ->   13 x  13 x1024 1.595 BF
  71 Shortcut Layer: 68,  wt = 0, wn = 0, outputs:  13 x  13 x1024 0.000 BF
  72 conv    512       1 x 1/ 1     13 x  13 x1024 ->   13 x  13 x 512 0.177 BF
  73 conv   1024       3 x 3/ 1     13 x  13 x 512 ->   13 x  13 x1024 1.595 BF
  74 Shortcut Layer: 71,  wt = 0, wn = 0, outputs:  13 x  13 x1024 0.000 BF
  75 conv    512       1 x 1/ 1     13 x  13 x1024 ->   13 x  13 x 512 0.177 BF
  76 conv   1024       3 x 3/ 1     13 x  13 x 512 ->   13 x  13 x1024 1.595 BF
  77 conv    512       1 x 1/ 1     13 x  13 x1024 ->   13 x  13 x 512 0.177 BF
  78 conv   1024       3 x 3/ 1     13 x  13 x 512 ->   13 x  13 x1024 1.595 BF
  79 conv    512       1 x 1/ 1     13 x  13 x1024 ->   13 x  13 x 512 0.177 BF
  80 conv   1024       3 x 3/ 1     13 x  13 x 512 ->   13 x  13 x1024 1.595 BF
  81 conv     24       1 x 1/ 1     13 x  13 x1024 ->   13 x  13 x  24 0.008 BF
  82 yolo
[yolo] params: iou loss: mse (2), iou_norm: 0.75, cls_norm: 1.00, scale_x_y: 1.00
  83 route  79                                     ->   13 x  13 x 512
  84 conv    256       1 x 1/ 1     13 x  13 x 512 ->   13 x  13 x 256 0.044 BF
  85 upsample                 2x    13 x  13 x 256 ->   26 x  26 x 256
  86 route  85 61                              ->   26 x  26 x 768
  87 conv    256       1 x 1/ 1     26 x  26 x 768 ->   26 x  26 x 256 0.266 BF
  88 conv    512       3 x 3/ 1     26 x  26 x 256 ->   26 x  26 x 512 1.595 BF
  89 conv    256       1 x 1/ 1     26 x  26 x 512 ->   26 x  26 x 256 0.177 BF
  90 conv    512       3 x 3/ 1     26 x  26 x 256 ->   26 x  26 x 512 1.595 BF
  91 conv    256       1 x 1/ 1     26 x  26 x 512 ->   26 x  26 x 256 0.177 BF
  92 conv    512       3 x 3/ 1     26 x  26 x 256 ->   26 x  26 x 512 1.595 BF
  93 conv     24       1 x 1/ 1     26 x  26 x 512 ->   26 x  26 x  24 0.017 BF
  94 yolo
[yolo] params: iou loss: mse (2), iou_norm: 0.75, cls_norm: 1.00, scale_x_y: 1.00
  95 route  91                                     ->   26 x  26 x 256
  96 conv    128       1 x 1/ 1     26 x  26 x 256 ->   26 x  26 x 128 0.044 BF
  97 upsample                 2x    26 x  26 x 128 ->   52 x  52 x 128
  98 route  97 36                              ->   52 x  52 x 384
  99 conv    128       1 x 1/ 1     52 x  52 x 384 ->   52 x  52 x 128 0.266 BF
 100 conv    256       3 x 3/ 1     52 x  52 x 128 ->   52 x  52 x 256 1.595 BF
 101 conv    128       1 x 1/ 1     52 x  52 x 256 ->   52 x  52 x 128 0.177 BF
 102 conv    256       3 x 3/ 1     52 x  52 x 128 ->   52 x  52 x 256 1.595 BF
 103 conv    128       1 x 1/ 1     52 x  52 x 256 ->   52 x  52 x 128 0.177 BF
 104 conv    256       3 x 3/ 1     52 x  52 x 128 ->   52 x  52 x 256 1.595 BF
 105 conv     24       1 x 1/ 1     52 x  52 x 256 ->   52 x  52 x  24 0.033 BF
 106 yolo
```

The parameters used for testing the completeness of the model are mAP, IoU, and f1 score. Mean Average Precision (mAP) is the mean value of average precisions and Intersection Over Union (IoU) is the average intersect over union of objects and detections for a certain threshold and f1 score depends on the precision and recall and can be calculated based on confusion matrix.

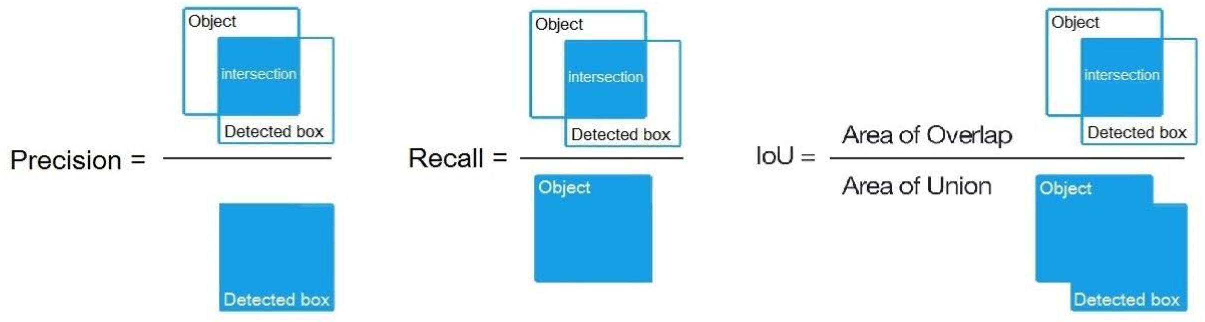


*Figure 3.18: Performance metrics graphical representation [96]*

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

$TP$ = True positive

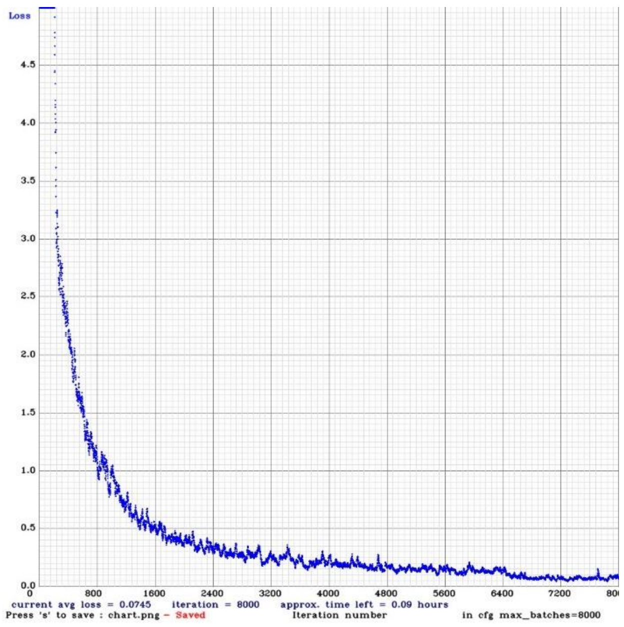
$TN$ = True negative

$FP$ = False positive

$FN$ = False negative

*Figure 3.19: Performance metrics mathematical representation*

Performance charts will be shown in the evaluation and result section. Following chart visualize the loss during training.



```
current avg loss = 0.0745    iteration = 8000    approx. time left = 0.09 hours
Press 's' to save : chart.png - Saved                                in cfg max_batches=8000
```

*Figure 3.20: Loss curve during training process*

Above curve is generated based on the following loss function by Darknet itself.

$$
\begin{aligned}
&\lambda_{\textbf{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[ (x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\
&+ \lambda_{\textbf{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[ \left( \sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left( \sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right] \\
&+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left( C_i - \hat{C}_i \right)^2 \\
&+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left( C_i - \hat{C}_i \right)^2 \\
&+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2
\end{aligned}
$$

*Figure 3.21: YOLO loss function taken from version 1 paper [15]*

The generated model weights are used to combine with the explainable model in the next section and these weights are selected based on the evaluation results which will be discussed in the evaluation and result section.

## 3.5 Explainable Model

As discussed in the literature review section, explainability is added to a model to provide an understanding of the result derived by that model to the end-users. They may have the knowledge of the subject domain of this study but may not have the knowledge of how the convolutional neural networks work and it is not necessary for them to have the knowledge related to convolutional neural networks to understand this study but need a method to return the results in a domain specific manner.

To achieve this goal based on the literature review one of the most user-friendly explanation families, the verbal explanations are used because the explanations are usually provided in natural language form which are like humane explanations.

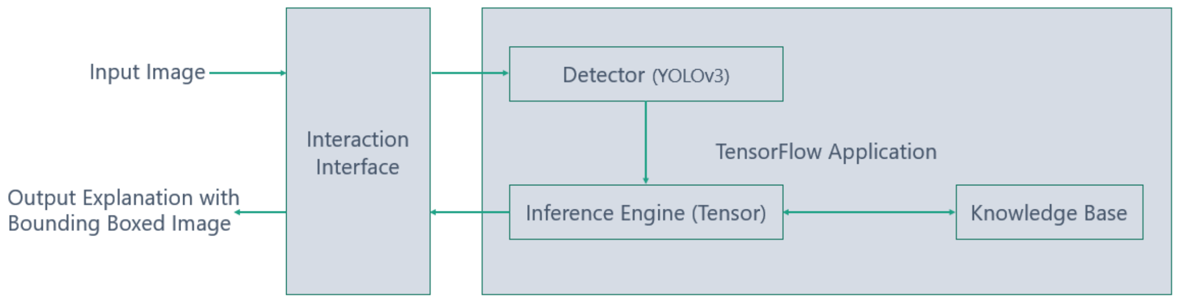


*Figure 3.22: Explainable Architecture*

The interaction interface can be a Mobile application or a server application that takes the input as an image which is evaluated against the YOLOv3 model, which was trained in the previous section. The classes are pre-defined in the TensorFlow application is written on Java TensorFlow Classifier Class.

The knowledge base is created with the help of an Agronomist and online knowledge sources such as YARA [33]. Each of the class has a verbal explanation like humane explanations in the inference engine to provide based on the detector results. As an example,

*Table 3.8: Sample explanations of the deficiencies identified*

| Deficiency Class | Sample Explanation |
|---|---|
| Calcium | Due to Calcium deficiency, the growth of lettuce is observed as reduced and leaves are wavier than normal. Brown or grey lesions has developed starting from leaf margins or tips of young leaves. This symptom is called as 'tip-burn'. When the Calcium -deficiency progresses, the leaves begin to die from tips and margins inwards. Subsequently, the persistent symptoms will spread over the older leaves. |
| Nitrogen | Older leaves have the symptoms of N deficiency at first with light green chlorosis. This moves to the head and will have light green chlorosis. No head is formed with severe Nitrogen deficiency and the growth is restricted. But the leave shape remains normal. |
| Magnesium | Older leaves have the symptoms of Mg deficiency at first with yellowing between veins and leaf margin discoloration of yellowish orange. If this continues, the yellow leaf zones will die but the veins remain green. Growth is incrementally restricted with the severity of the deficiency. |

# Chapter 4 : Evaluation and Results

Evaluating deficiency identification implies, evaluating the deep neural network itself. This research uses the YOLOv3 (You Only Look Once version 3) object detection algorithm based on fully convolutional neural networks. So, the evaluation of the trained algorithm is performed on the testing datasets.

## 4.1 Evaluation Criteria and Datasets

Based on the literature review, the dataset is separated into 80% (training) 20% (testing) subsets as mentioned in Table 4.1 Dataset Separation the dataset is separated into training and testing subsets prior to the augmentation to avoid the same image in both datasets.

*Table 4.1: The separated dataset*

| Class | Number of Images | | Augmented Images | |
|---|---|---|---|---|
| | Training | Testing | Training | Testing |
| Healthy | 102 | 26 | 897 | 224 |
| Calcium deficient | 117 | 30 | 911 | 228 |
| Nitrogen deficient | 95 | 24 | 878 | 219 |
| Magnesium deficient | 84 | 21 | 770 | 193 |

Even though the above dataset is separated, the images are taken in similar conditions and similar backgrounds. To overcome this issue and as the second step of validating the model, a web scraped dataset was used. The images were downloaded using the following web scraper written in NodeJS (JavaScript). Images were scraped from the ecosia.org, a search engine based in Berlin, Germany, that donates 80% or more of its profits to nonprofit organizations that focus on reforestation. Search results are simpler than google, so it is easy to download.

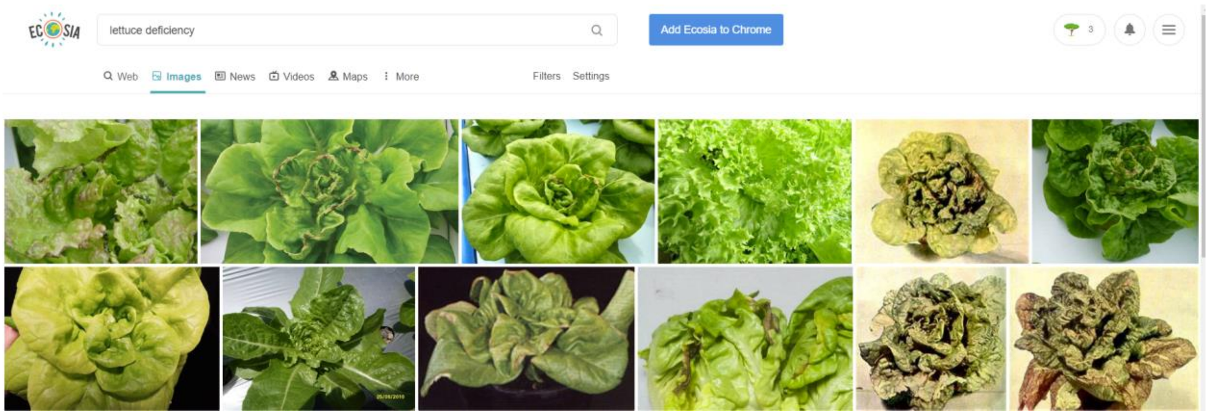


*Figure 4.1: Ecosia is a social business founded in 2009 after a trip around the world forcusing reforestation*

```
const perf = require('execution-time')();
const parser = require('node-html-parser').parse;
const axios = require('axios');
const fs = require('fs');
const download = require('image-downloader');
const del = require('delete');

// start timer
perf.start();
// Get the csvfile path
const csvFile = process.argv[2] ? process.argv[2] : 'veg.csv';
// Config the scrapper
const config = {
    // Number of images to get on each picture, if not set, set default to 4
    Samples: (process.argv[3]) ? process.argv[3] : 4,
    // Where to store the downloaded image, if not set, set default to ./images
    store: "./images/"
};
// Function to prepare the URL
function URL(veg) {
    return `https://ecosia.org/images?q=${veg}`;
}
// Function to download the image
async function download(options) {
    try {
        var {
            file,
            image
        } = await download.image(options);
        console.log("", "", "[Info] Downloaded: " + file); // => /path/to/dest/image.jpg

    } catch (e) {
        console.error("", "", "[Info] Error: Unable to download " + file);
    }
}
// function to scrape the page
async function scrap(url, name) {
    try {
        // Request the page
        var response = await axios.get(url);
        console.log("[INFO]:", "Scrapping " + name);
        // create a folder where to store the images if it does not exits
        if (!fs.existsSync(config.store + name)) {
            fs.mkdirSync(config.store + name);
        } else {
            // if it exists delete and recreate
            del.sync(config.store + name);
            fs.mkdirSync(config.store + name);
        }
        //parse the response
        var dom = parser(response.data);
        var imgs = dom.querySelectorAll("a.image-result");
        for (var x = 0; x < imgs.length; x++) {
            // console.log(imgs[x].attributes.href);
            await download({
                url: imgs[x].attributes.href,
                dest: config.store + name
            });
            if (x >= (config.Samples - 1)) break;
        }

    } catch (e) {
        console.error("", "[Info] Error: Unable to GET " + url);
        console.log(e);
    }
}
//load and read csv file
var contents = fs.readFileSync(csvFile, 'utf8');
var vegetables = contents.split(",");
async function initScrap() {
    for (var i = 0; i < vegetables.length; i++) {
        await scrap(URL(vegetables[i]), vegetables[i]);
        if (i == (vegetables.length - 1)) console.log("[Info]  Done in", ((perf.stop()).time / 1000).toFixed(2)
+ "s");
    }
}
initScrap();
```

These images were not subjected to any augmentation. Among those images, the best 100 images of each deficiency and healthy were selected to validate the model. Figure 4.2 shows a sample set of annotated images.

The Table 4.2 contains and Figure 4.3 shows the bulk downloads of the images and selection.

*Table 4.2: Number of deficiency and healthy images downloaded from ecosia using web crawler*

| Class | Total bulk downloaded images | Selected images |
|---|---|---|
| Controlled | 193 | 100 |
| Calcium deficient | 172 | 100 |
| Nitrogen deficient | 158 | 100 |
| Magnesium deficient | 135 | 100 |

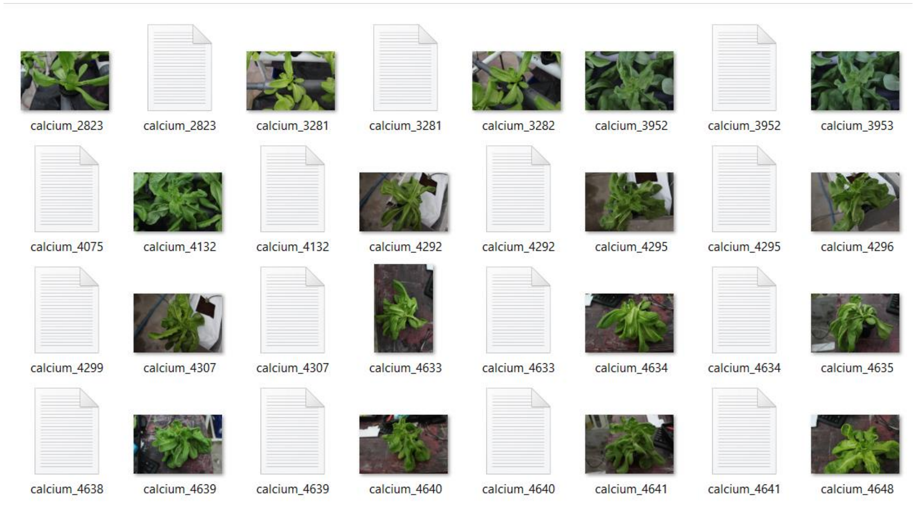


*Figure 4.2: Annotated deficient images in dataset*

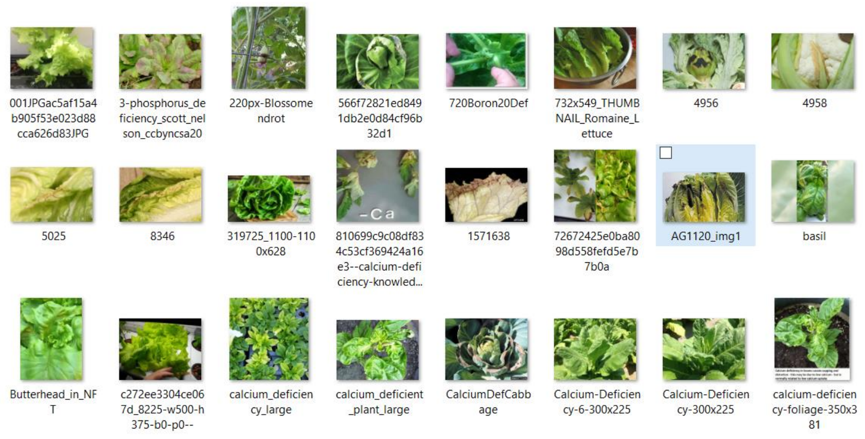

001JPGac5af15a4b905f53e023d88cca626d83JPG
3-phosphorus_deficiency_scott_nelson_ccbyncsa20
220px-Blossomendrot
566f72821ed8491db2e0d84cf96b32d1
720Boron20Def
732x549_THUMBNAIL_Romaine_Lettuce
4956
4958
5025
8346
319725_1100-1100x628
810699c9c08df834c53cf369424a16e3--calcium-deficiency-knowled...
1571638
72672425e0ba8098d558fefd5e7b7b0a
AG1120_img1
basil
Butterhead_in_NFT
c272ee3304ce067d_8225-w500-h375-b0-p0--
calcium_deficiency_large
calcium_deficient_plant_large
CalciumDefCabbage
Calcium-Deficiency-6-300x225
Calcium-Deficiency-300x225
calcium-deficiency-foliage-350x381

*Figure 4.3: Bulk downloaded images from web scrape, prior to selection*

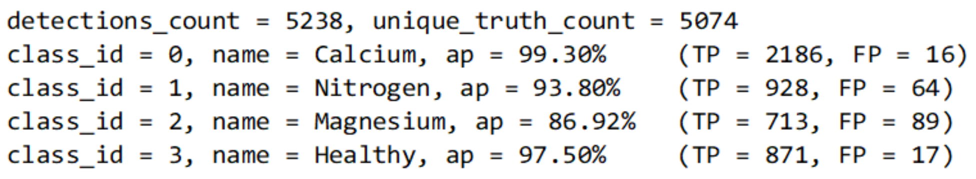
As explained in the 3.2 Dataset Creation of this study, there are several available general fine-grained datasets for image classifications. Table 4.3 shows the most popular and freely available datasets via the internet. From those datasets, PlantVillage and CropDeep datasets include lettuce images, but they have only the lettuce related disease images but neither of them includes nutrient deficiency images that can be used for classification in this study. Hence the model was not validated against a public dataset.

*Table 4.3:Summary of popular general and fine-grained vision datasets with plants [43]*

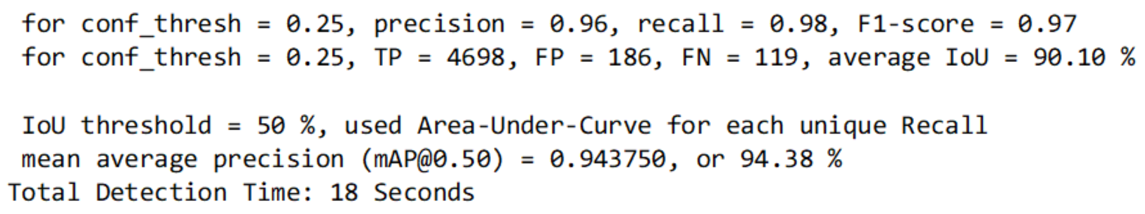
| **Dataset** | **Classes** | **Image Number** | **Annotation Samples Number** | **Comment** |
|---|---|---|---|---|
| Flowers 102 | 102 | 1020 | 0 | The flowers chosen are flowers commonly occurring in the United Kingdom. Each class consists of between 40 and 258 images. |
| PlantVillage | 38 | 19,298 | 0 | contains only images of leaves that are previously cropped in the field and captured by a camera in the laboratory |
| CUB 200-2011 | 200 | 5994 | 0 | general-purpose dataset (not only plants) mostly web crawled data |
| Urban Trees | 18 | 14,572 | 0 | trees labeled by geo-location and tree species located within Pasadena. The dataset includes dense aerial and street view imagery |
| LeafSnap | 185 | 30,866 | 0 | high-quality images taken of pressed leaves; "typical" images taken by mobile devices (iPhones mostly) in outdoor environments |
| ImageNet | 1000 | 14,197,122 | 1,034,908 | general-purpose dataset (not only plants) mostly web crawled data |
| MS-COCO | 80 | 300,000 | More than 2,000,000 | general-purpose dataset (not only plants) mostly web crawled data |
| AI Challenge | 61 | 47,393 | 0 | general-purpose dataset (not only plants) mostly web crawled data |
| iNat2017 | 5089 | 858,184 | 561,767 | general-purpose dataset (not only plants) mostly web crawled data |
| VegFru | 70 | 160,731 | 0 | vegetables and fruits which are closely associated with the daily life of everyone |
| CropDeep | 31 | 31,147 | 49,765 | mages are collected by various equipment including cameras of IoT, autonomous spray robot, autonomous pinking robot, mobile cameras and smartphones in an intelligent agricultural monitoring and management platform |

## 4.2 The Experimental Results

The performance evaluation is carried out while training as well against the testing dataset from the separated images of the original dataset. The following values are the performance metrics obtained on the training dataset.



```
detections_count = 5238, unique_truth_count = 5074
class_id = 0, name = Calcium, ap = 99.30%     (TP = 2186, FP = 16)
class_id = 1, name = Nitrogen, ap = 93.80%    (TP = 928, FP = 64)
class_id = 2, name = Magnesium, ap = 86.92%   (TP = 713, FP = 89)
class_id = 3, name = Healthy, ap = 97.50%     (TP = 871, FP = 17)
```



```
 for conf_thresh = 0.25, precision = 0.96, recall = 0.98, F1-score = 0.97
 for conf_thresh = 0.25, TP = 4698, FP = 186, FN = 119, average IoU = 90.10 %

 IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
 mean average precision (mAP@0.50) = 0.943750, or 94.38 %
Total Detection Time: 18 Seconds
```

### 4.2.1 Confusion Matrix, IoU, mAP, and F1-score

A confusion matrix is a summary that gives the prediction results on a detection problem. The number of correct as well as a number of incorrect predictions is summarized with counted values and broken-down class by class. This is the key to the confusion matrix. The confusion matrix shows the ways in which the defined model is confused when it makes predictions. It gives insight not only into the errors being made by a detector but also more importantly the types of errors that are being made. These counts are made in reference to the number of annotations/bounding boxes in images.

For Calcium (ClassId = 0), TP = 2186, FP = 16

For Nitrogen (ClassId = 1), TP = 928, FP = 64

For Magnesium (ClassId = 2), TP = 713, FP = 89

For Healthy (ClassId = 3) TP = 871, FP = 17

For confidence threshold 0.25, the average Intersection over Union is 90.10%

For IoU threshold of 0.5, that is 50%, the mean average precision 94.38%

For confidence threshold 0.25, the F1-score (the harmonic mean(average) of the precision and recall) is 0.97

Figure 4.4 visualize the mean average precision of the model throughout the training process over the training image iterations.

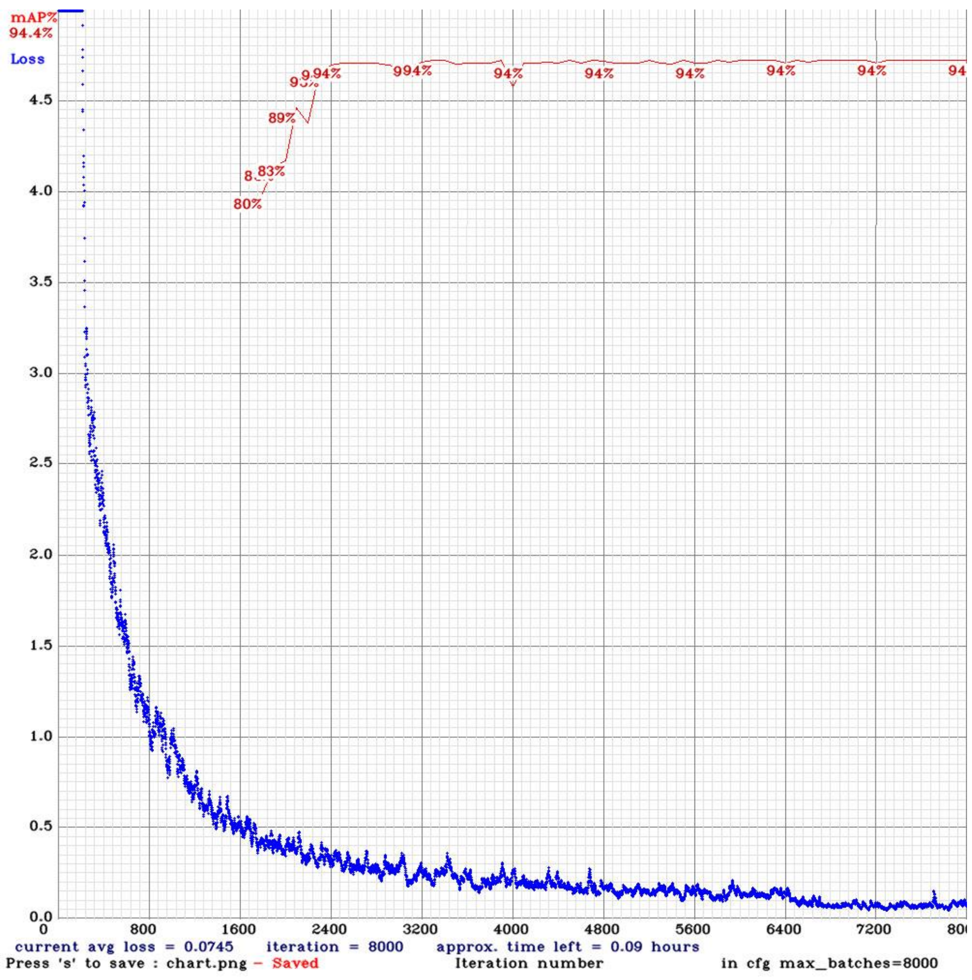


*Figure 4.4: Mean Average Precision against the iterations over training dataset along with the loss curve*

Once the training is completed, the weights were chosen to test the model based on the above-mentioned parameters and by the following graph, the best accuracy model achieved is 94.38% and the loss is minimum at the 8000.

The following values are the performance metrics obtained on the testing dataset.

```
detections_count = 912, unique_truth_count = 838
class_id = 0, name = Calcium, ap = 91.20%     (TP = 259, FP = 19)
class_id = 1, name = Nitrogen, ap = 87.60%    (TP = 221, FP = 28)
class_id = 2, name = Magnesium, ap = 78.13%   (TP = 156, FP = 34)
class_id = 3, name = Healthy, ap = 89.91%     (TP = 202, FP = 22)

 for conf_thresh = 0.25, precision = 0.89, recall = 0.92, F1-score = 0.90
 for conf_thresh = 0.25, TP = 838, FP = 103, FN = 73, average IoU = 82.22 %

 IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
 mean average precision (mAP@0.50) = 0.854613, or 85.46 %
Total Detection Time: 8 Seconds
```

Following figure shows the results of the testing. (a) is classified as Calcium deficient, (b) is classified as Nitrogen deficient, (c) is classified as Magnesium deficient, and (d) is healthy.

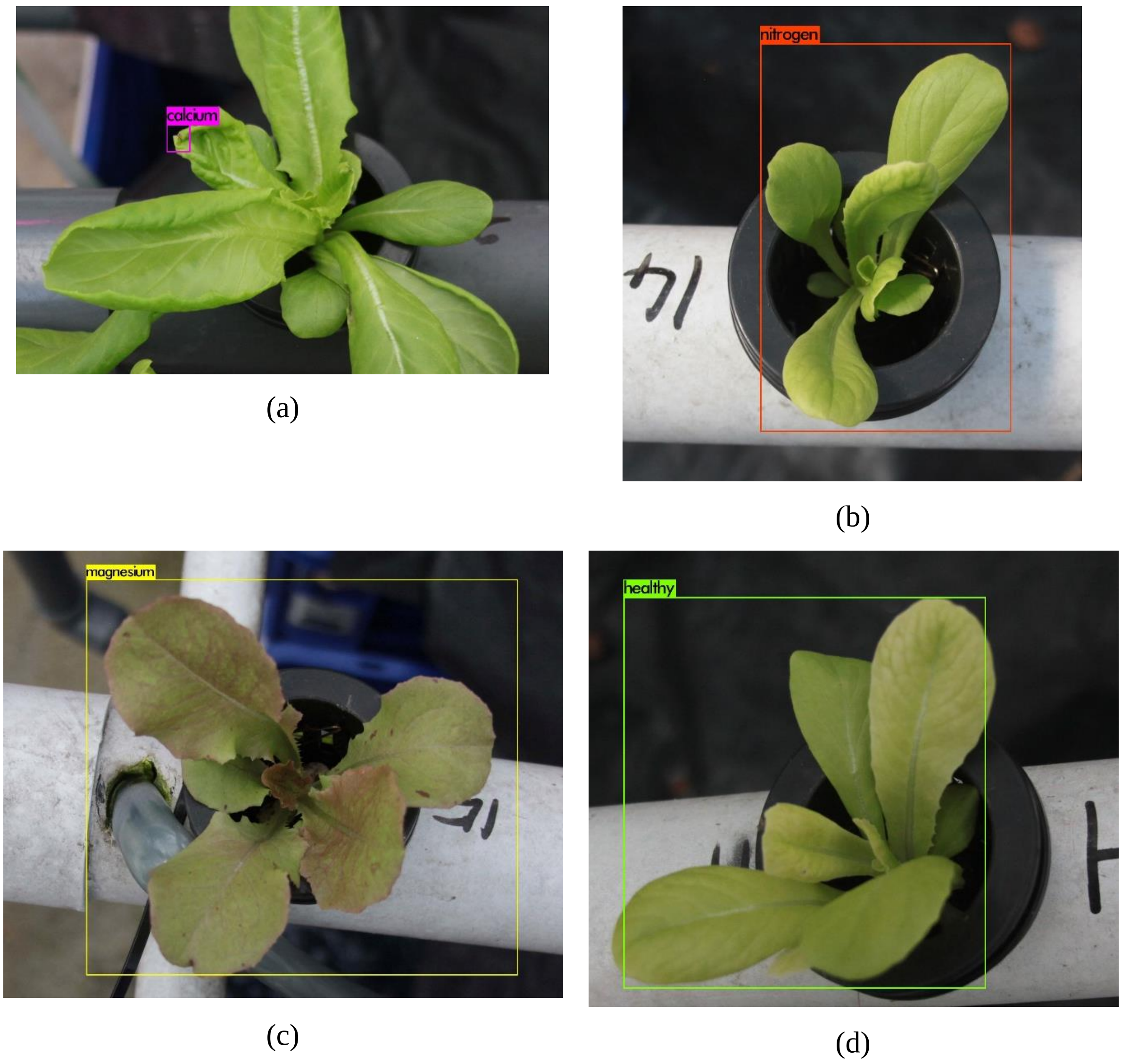


*Figure 4.5: Detection output. (a) Calcium deficient, (b) Nitrogen deficient, (c) Magnesium deficient, and (d) healthy*

As the next step of evaluation, the web crawled dataset is tested against the model and the following are the results.

```
detections_count = 511, unique_truth_count = 309
class_id = 0, name = Calcium, ap = 81.17%     (TP = 83, FP = 14)
class_id = 1, name = Nitrogen, ap = 73.40%    (TP = 77, FP = 21)
class_id = 2, name = Magnesium, ap = 69.33%   (TP = 69, FP = 28)
class_id = 3, name = Healthy, ap = 79.01%     (TP = 80, FP = 15)

 for conf_thresh = 0.25, precision = 0.80, recall = 0.72, F1-score = 0.86
 for conf_thresh = 0.25, TP = 309, FP = 78, FN = 63, average IoU = 76.48 %

 IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
 mean average precision (mAP@0.50) = 0.757275, or 75.72 %
Total Detection Time: 8 Seconds
```

## 4.2.2 Summary

The performance results from the training, testing on own data, and testing on web crawled data can be summarized into following table.

*Table 4.4: Summary of performance test results*

| | Training | | | Testing - Own | | | Testing - Web | | |
|---|---|---|---|---|---|---|---|---|---|
| | **AP** | **TP** | **FP** | **AP** | **TP** | **FP** | **AP** | **TP** | **FP** |
| **Calcium** | 99.30 | 2186 | 16 | 91.20 | 259 | 19 | 81.17 | 83 | 14 |
| **Nitrogen** | 93.80 | 928 | 64 | 82.60 | 221 | 28 | 73.40 | 77 | 21 |
| **Magnesium** | 86.92 | 713 | 89 | 78.13 | 156 | 34 | 69.33 | 69 | 28 |
| **Healthy** | 97.50 | 871 | 17 | 89.91 | 202 | 22 | 79.01 | 80 | 15 |
| | | | | | | | | | |
| **mAP** | 94.38 | | | 85.46 | | | 75.73 | | |
| **TP** | 4698 | | | 838 | | | 309 | | |
| **FP** | 186 | | | 103 | | | 78 | | |
| **FN** | 119 | | | 73 | | | 63 | | |
| **Precision** | 0.96 | | | 0.89 | | | 0.80 | | |
| **Recall** | 0.98 | | | 0.92 | | | 0.83 | | |
| **F1-score** | 0.97 | | | 0.90 | | | 0.81 | | |

As seen from the above table the performance of the model with the same dataset as the training data is having the best mean average precision and the least value is the web crawled dataset. The reason behind this is that the model is already trained with the training data and the features are already extracted based on them and the study's own testing dataset is a separation on the training dataset hence have similar backgrounds and similarly featured as in the training dataset. But as for the web crawled dataset, it has a collection of images with different color gradients, contrasts, and sizes. This may cause this decrement in performance.

When comparing the classes against the same dataset, the difference in performance values can be seen among them. Calcium has the overall best performance values. The reason is that the maximum number of images in the dataset is related to calcium and the maximum number of annotations is also related to calcium. Tip burn symptom of the calcium makes that there as multiple bounding boxes in the same image and there are multiple positions of symptoms in the same image. In contrast, Nitrogen and Magnesium symptoms are spread throughout the plant, so the bounding box covers the entire plant. This makes a smaller number of bounding boxes for these two classes. Finally, the healthy class also have better performance than Nitrogen and Magnesium and lesser performance than Calcium.

# Chapter 5 : Conclusion

With the improvements of horticultural lettuce production, greenhouses, one of the major controlled environmental architectures, are vastly used with the aim of providing an optimal nutrient composition to the corps, to protect the crops from pests without applying pesticides, and to provide the optimal environmental conditions to the corps as in temperature, humidity, etc. Even though the greenhouse is a controlled environment and lettuce is produced in a nutrient solution, there can be a production loss due to nutrient deficiencies. Some of the deficiencies generate manifestation in the visible spectrum. There are many solutions to mitigate the crop loss due to nutrient deficiencies, but identifying the deficiency beforehand is a crucial step in this deficiency management. To overcome this, machine vision is applied to monitor plants. There are multiple approaches for this task as discussed in the literature review, but the current limitation is the lack of explanation of results in a humane understandable method.

## 5.1 Study Conclusion

In this study, YOLOv3 is used to detect the deficiencies of greenhouse lettuce because it has good detection speed and good detection results according to the literature review. In addition to that, the detector takes the first step of the explanation model by providing the bounding boxes of the deficiency. This helps to understand the result in a way. Hence there is a need of a dataset for detector model training, the literature review was carried out for finding a dataset but as explained the publicly available fine-grained datasets do not include lettuce deficiency images. Because of that, an experimental setup was created in AiGROW greenhouse with the help of an Agronomist to acquire images with deficiency symptoms. The deficiency images were acquired for lettuce calcium, nitrogen and magnesium deficiencies along with a control setup for healthy plants. The acquired images were annotated, and the annotated images were subjected to the augmentation process to grow the dataset. Before the augmentation, the dataset was separated to training and testing subsets as for the testing dataset to be used in the first step of evaluation of the model. This process is explained in 3.2 Dataset Creation Section of Chapter 3.

The YOLOv3 model was trained on top of the Darknet framework, which is an open source neural network framework written in C and CUDA. It is fast, easy to install, and supports CPU and GPU computation. As explained in 3.4 Implementation section of Chapter 3, Darknet was compiled on specific hardware setup and YOLOv3 was trained using transfer learning methods where the weights were pre-trained weights for the convolutional layers by Darknet. The training mAP achieved 94.38% and the loss was stabilized in 8000 iterations. The testing was carried out on the own dataset and another dataset was created using a web crawler as described in 4.1 Evaluation Criteria and Datasets section of Chapter 4. The separated testing dataset achieved an mAP of 85.15% while the web crawled test dataset settled on 75.73%.

As the explainable architecture, the weights generated from the model training was combined with the TensorFlow Inference engine for Java and generated the final output with explanation based on the detector output.

## 5.2 Study Generalization

This research is focused to overcome the challenge of explaining the nutrient deficiencies identified using Convolutional Neural Networks. The study combines the results obtained from the object detector with the TensorFlow inference engine to provide humane understandable results for deficiency identification of crops. This research studied this concept using greenhouse lettuce as the plant.

The solution can be generalized to use in any kind of crop which has nutrient deficiencies with the symptoms available in the visible spectrum. The weights from the trained neural network can only be used to identify the similar images which were trained. In this case, the neural network is trained to identify the Calcium, Nitrogen, and Magnesium nutrient deficiencies of lettuce but the same wights can be used to identify other crops with visually similar nutrient deficiencies. The accuracy may lower than the accuracy with lettuce.

To gain the best results out of the same network, it should be trained with other images related to desired deficiencies and the weight must be obtained to combine with the inference engine. The knowledge base should be updated to support the additional classes added to the neural network so that the inference engine can return the explanations for those classes as well.

The next subsection will explain how the increase the accuracy and escape the bound of the visible spectrum. But the generalization is valid with all the extensions discussed in the final section.

## 5.3 Future Work

The model was providing quite good results for calcium deficiency and healthy lettuce, but nitrogen and magnesium were not quite good. The reason was identified as a lack of enough training material. Few extensions can be carried out regarding this study to expand and make it more efficient and accurate.

One of the most constructive extensions is to expand the dataset with new images not focusing on a single lettuce variety but multiple varieties of the same deficiency setups. This can provide the features to identify deficiencies of multiple varieties of lettuce.

The next extension is to capture images using an IR camera. Even though the colored images visualize the symptoms of tip burn clearly, the color changes and color patches are not much visible in the RGB spectrum of the images, because of that the annotation task was quite difficult to point the deficiency symptom.

Finally, other pre-trained weights of YOLOv3 on darknet can be used to train the model.

# Reference

[1] "Global Greenhouse Horticulture Market 2018-2022," *Technavio*. https://www.technavio.com/report/global-greenhouse-horticulture-market-analysis-share-2018 (accessed Sep. 19, 2019).

[2] M. Hasan, A. K. M. M. Tahsin, M. N. Islam, M. Ali, and J. Uddain, "Growth and Yield of Lettuce (Lactuca Sativa L.) Influenced As Nitrogen Fertilizer and Plant Spacing," *IOSR J. Agric. Vet. Sci.*, vol. 10, pp. 62–71, Jun. 2017, doi: 10.9790/2380-1006016271.

[3] M. Sardare, "A REVIEW ON PLANT WITHOUT SOIL - HYDROPONICS," *Int. J. Res. Eng. Technol.*, vol. 02, pp. 299–304, Mar. 2013, doi: 10.15623/ijret.2013.0203013.

[4] J. G. Arnal Barbedo, "Digital image processing techniques for detecting, quantifying and classifying plant diseases," *SpringerPlus*, vol. 2, no. 1, Dec. 2013, doi: 10.1186/2193-1801-2-660.

[5] N. Mattson and T. Merrill, "Nutrient Deficiencies in Hydroponic Lettuce," p. 10.

[6] D. Story, M. Kacira, C. Kubota, A. Akoglu, and L. An, "Lettuce calcium deficiency detection with machine vision computed plant features in controlled environments," *Comput. Electron. Agric.*, vol. 74, no. 2, pp. 238–243, Nov. 2010, doi: 10.1016/j.compag.2010.08.010.

[7] D. Story and M. Kacira, "Design and implementation of a computer vision-guided greenhouse crop diagnostics system," *Mach. Vis. Appl.*, vol. 26, no. 4, pp. 495–506, May 2015, doi: 10.1007/s00138-015-0670-5.

[8] C. Bock, G. Poole, P. E. Parker, and T. Gottwald, "Plant Disease Severity Estimated Visually, by Digital Photography and Image Analysis, and by Hyperspectral Imaging," *Crit. Rev. Plant Sci.*, vol. 29, Mar. 2010, doi: 10.1080/07352681003617285.

[9] K. O'Shea and R. Nash, "An Introduction to Convolutional Neural Networks," *ArXiv E-Prints*, Nov. 2015.

[10] S. Albawi, T. Abed Mohammed, and S. ALZAWI, "Understanding of a Convolutional Neural Network," Aug. 2017, doi: 10.1109/ICEngTechnol.2017.8308186.

[11] A. Krizhevsky, I. Sutskever, and G. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," *Neural Inf. Process. Syst.*, vol. 25, Jan. 2012, doi: 10.1145/3065386.

[12] Z. Yang and R. Nevatia, "A Multi-Scale Cascade Fully Convolutional Network Face Detector," *ArXiv160903536 Cs*, Sep. 2016, Accessed: Jan. 19, 2020. [Online]. Available: http://arxiv.org/abs/1609.03536.

[13] D. G. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints," *Int. J. Comput. Vis.*, vol. 60, no. 2, pp. 91–110, Nov. 2004, doi: 10.1023/B:VISI.0000029664.99615.94.

[14] R. Girshick, "Fast R-CNN," *ArXiv150408083 Cs*, Sep. 2015, Accessed: Jan. 19, 2020. [Online]. Available: http://arxiv.org/abs/1504.08083.

[15] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," *ArXiv150602640 Cs*, May 2016, Accessed: Jan. 19, 2020. [Online]. Available: http://arxiv.org/abs/1506.02640.

[16] Z.-Q. Zhao, P. Zheng, S. Xu, and X. Wu, "Object Detection with Deep Learning: A Review," *ArXiv180705511 Cs*, Apr. 2019, Accessed: Jan. 19, 2020. [Online]. Available: http://arxiv.org/abs/1807.05511.

[17] S.-H. Tsang, "Review: YOLOv3 — You Only Look Once (Object Detection)," *Medium*, Mar. 20, 2019. https://towardsdatascience.com/review-yolov3-you-only-look-once-object-detection-eab75d7a1ba6 (accessed Jan. 19, 2020).

[18] B. Goodman and S. Flaxman, "European Union regulations on algorithmic decision-making and a 'right to explanation,'" *AI Mag.*, vol. 38, no. 3, pp. 50–57, Oct. 2017, doi: 10.1609/aimag.v38i3.2741.

[19] D. Castelvecchi, "Can we open the black box of AI?," *Nature*, vol. 538, pp. 20–23, Oct. 2016, doi: 10.1038/538020a.

[20] A. B. Arrieta *et al.*, "Explainable Artificial Intelligence (XAI): Concepts, Taxonomies, Opportunities and Challenges toward Responsible AI," *ArXiv191010045 Cs*, Dec. 2019, Accessed: Jan. 19, 2020. [Online]. Available: http://arxiv.org/abs/1910.10045.

[21] D. Gunning, “DARPA’s explainable artificial intelligence (XAI) program,” Mar. 2019, pp. ii–ii, doi: 10.1145/3301275.3308446.

[22] “AiGrow - CodeGen,” *AiGrow - CodeGen International*. https://aigrow.lk/ (accessed Jan. 20, 2020).

[23] J. A. K. Noumedem, D. E. Djeussi, L. Hritcu, M. Mihasan, and V. Kuete, “Chapter 20 - Lactuca sativa,” in *Medicinal Spices and Vegetables from Africa*, V. Kuete, Ed. Academic Press, 2017, pp. 437–449.

[24] M. Koudela and K. Petříková, “Nutrients content and yield in selected cultivars of leaf lettuce (Lactuca sativa L. var. crispa),” *Hortic. Sci.*, vol. 35 (2008), no. No. 3, pp. 99–106, Aug. 2008, doi: 10.17221/3/2008-HORTSCI.

[25] D. W. Goodall, A. G. Lipp, and W. G. Slater, “Nutrient Interactions and Deficiency Diagnosis in the Lettuce,” *Aust. J. Biol. Sci.*, vol. 8, no. 3, pp. 301–329, 1955, doi: 10.1071/bi9550301.

[26] J. P. N. L. Roorda van Eysinga and K. W. Smilde, *Nutritional disorders in glasshouse lettuce*. 1971.

[27] *Hydroponic Lettuce Handbook*. .

[28] R. Shamshiri *et al.*, “Advances in greenhouse automation and controlled environment agriculture: A transition to plant factories and urban agriculture,” *Int. J. Agric. Biol. Eng.*, vol. 11, Jan. 2018, doi: 10.25165/j.ijabe.20181101.3210.

[29] G. L. Barbosa *et al.*, “Comparison of Land, Water, and Energy Requirements of Lettuce Grown Using Hydroponic vs. Conventional Agricultural Methods,” in *International journal of environmental research and public health*, 2015, doi: 10.3390/ijerph120606879.

[30] N. Sharma, S. Acharya, K. Kumar, N. Singh, and O. Chaurasia, “Hydroponics as an advanced technique for vegetable production: An overview,” *J. Soil Water Conserv.*, vol. 17, pp. 364–371, Jan. 2019, doi: 10.5958/2455-7145.2018.00056.5.

[31] A. Hetzroni, G. E Miles, B. A. Engel, P. Hammer, and R. X Latin, “Machine vision monitoring of plant health,” *Adv. Space Res. Off. J. Comm. Space Res. COSPAR*, vol. 14, pp. 203–12, Dec. 1994, doi: 10.1016/0273-1177(94)90298-4.

[32] D. Story and M. Kacira, “Design and implementation of a computer vision-guided greenhouse crop diagnostics system,” *Mach. Vis. Appl.*, vol. 26, no. 4, pp. 495–506, May 2015, doi: 10.1007/s00138-015-0670-5.

[33] “Yara UK.” https://www.yara.co.uk/ (accessed Sep. 20, 2019).

[34] S. P. Mohanty, D. P. Hughes, and M. Salathé, “Using Deep Learning for Image-Based Plant Disease Detection,” *Front. Plant Sci.*, vol. 7, 2016, doi: 10.3389/fpls.2016.01419.

[35] D. P. Hughes and M. Salathe, “An open access repository of images on plant health to enable the development of mobile disease diagnostics,” *ArXiv151108060 Cs*, Nov. 2015, Accessed: Sep. 19, 2019. [Online]. Available: http://arxiv.org/abs/1511.08060.

[36] Y.-Y. Zheng, J.-L. Kong, X.-B. Jin, X.-Y. Wang, T.-L. Su, and M. Zuo, “CropDeep: The Crop Vision Dataset for Deep-Learning-Based Classification and Detection in Precision Agriculture,” *Sensors*, vol. 19, no. 5, Mar. 2019, doi: 10.3390/s19051058.

[37] A. K. Rangarajan, R. Purushothaman, and A. Ramesh, “Tomato crop disease classification using pre-trained deep learning algorithm,” *Procedia Comput. Sci.*, vol. 133, pp. 1040–1047, Jan. 2018, doi: 10.1016/j.procs.2018.07.070.

[38] M. Türkoğlu and D. Hanbay, “Plant disease and pest detection using deep learning-based features,” *Turk. J. Electr. Eng. Comput. Sci.*, vol. 27, no. 3, pp. 1636–1651, May 2019.

[39] J. Amara, B. Bouaziz, and A. Algergawy, “A Deep Learning-based Approach for Banana Leaf Diseases Classification,” in *BTW*, 2017.

[40] A. Fuentes, S. Yoon, S. C. Kim, and D. S. Park, “A Robust Deep-Learning-Based Detector for Real-Time Tomato Plant Diseases and Pests Recognition,” *Sensors*, vol. 17, no. 9, Sep. 2017, doi: 10.3390/s17092022.

[41] C. DeChant *et al.*, “Automated Identification of Northern Leaf Blight-Infected Maize Plants from Field Imagery Using Deep Learning,” *Phytopathology*, vol. 107, no. 11, pp. 1426–1432, 2017, doi: 10.1094/PHYTO-11-16-0417-R.

[42] G. Wang, Y. Sun, and J. Wang, “Automatic Image-Based Plant Disease Severity Estimation Using Deep Learning,” *Computational Intelligence and Neuroscience*, 2017. https://www.hindawi.com/journals/cin/2017/2917536/ (accessed Oct. 12, 2019).

[43] K. Zhang, Q. Wu, A. Liu, and X. Meng, “Can Deep Learning Identify Tomato Leaf Disease?,” *Advances in Multimedia*, 2018. https://www.hindawi.com/journals/am/2018/6710865/ (accessed Oct. 12, 2019).

[44] A. Picon, A. Alvarez-Gila, M. Seitz, A. Ortiz Barredo, J. Echazarra, and A. Johannes, “Deep convolutional neural networks for mobile capture device-based crop disease classification in the wild,” *Comput. Electron. Agric.*, Apr. 2018, doi: 10.1016/j.compag.2018.04.002.

[45] A. Johannes *et al.*, “Automatic plant disease diagnosis using mobile capture devices, applied on a wheat use case,” *Comput. Electron. Agric.*, vol. 138, pp. 200–209, Jun. 2017, doi: 10.1016/j.compag.2017.04.013.

[46] J. Esgario, R. Krohling, and J. Ventura, *Deep Learning for Classification and Severity Estimation of Coffee Leaf Biotic Stress*. 2019.

[47] A. Ramcharan, K. Baranowski, P. McCloskey, B. Ahmed, J. Legg, and D. P. Hughes, “Deep Learning for Image-Based Cassava Disease Detection,” *Front. Plant Sci.*, vol. 8, Oct. 2017, doi: 10.3389/fpls.2017.01852.

[48] “Deep Learning for Plant Diseases: Detection and Saliency Map Visualisation,” *springerprofessional.de*. https://www.springerprofessional.de/en/deep-learning-for-plant-diseases-detection-and-saliency-map-visu/15829084 (accessed Oct. 12, 2019).

[49] M. Brahimi, B. Kamel, and A. Moussaoui, “Deep Learning for Tomato Diseases: Classification and Symptoms Visualization,” *Appl. Artif. Intell.*, pp. 1–17, May 2017, doi: 10.1080/08839514.2017.1315516.

[50] K. Ferentinos, “Deep learning models for plant disease detection and diagnosis,” *Comput. Electron. Agric.*, vol. 145, pp. 311–318, Feb. 2018, doi: 10.1016/j.compag.2018.01.009.

[51] S. Sladojevic, M. Arsenovic, A. Anderla, D. Culibrk, and D. Stefanovic, “Deep Neural Networks Based Recognition of Plant Diseases by Leaf Image Classification,” *Computational Intelligence and Neuroscience*, 2016. https://www.hindawi.com/journals/cin/2016/3289801/ (accessed Oct. 06, 2019).

[52] H. Agh Atabay, “Deep Residual Learning for Tomato Plant Leaf Disease Identification,” *J. Theor. Appl. Inf. Technol.*, vol. 95, pp. 6800–6808, Dec. 2017.

[53] A. F. Fuentes, S. Yoon, J. Lee, and D. S. Park, “High-Performance Deep Neural Network-Based Tomato Plant Diseases and Pests Diagnosis System With Refinement Filter Bank,” *Front. Plant Sci.*, vol. 9, Aug. 2018, doi: 10.3389/fpls.2018.01162.

[54] L. Bin, Y. Zhang, D. He, and Y. Li, “Identification of Apple Leaf Diseases Based on Deep Convolutional Neural Networks,” *Symmetry*, vol. 10, p. 11, Dec. 2017, doi: 10.3390/sym10010011.

[55] J. Barbedo, “Impact of dataset size and variety on the effectiveness of deep learning and transfer learning for plant disease classification,” *Comput. Electron. Agric.*, vol. 153, pp. 46–53, Oct. 2018, doi: 10.1016/j.compag.2018.08.013.

[56] M. Arsenovic, M. Karanovic, S. Sladojevic, A. Anderla, and D. Stefanovic, “Solving Current Limitations of Deep Learning Based Approaches for Plant Disease Detection,” *Symmetry*, vol. 11, no. 7, p. 939, Jul. 2019, doi: 10.3390/sym11070939.

[57] S. Wallelign, M. Polceanu, and C. Buche, “Soybean Plant Disease Identification Using Convolutional Neural Network,” in *FLAIRS-31*, Melbourne, United States, May 2018, pp. 146–151, Accessed: Oct. 15, 2019. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01807760.

[58] “The Why of Explainable AI,” *Element AI*. https://www.elementai.com/news/2019/the-why-of-explainable-ai (accessed Feb. 10, 2020).

[59] M. T. Ribeiro, S. Singh, and C. Guestrin, “‘Why Should I Trust You?’: Explaining the Predictions of Any Classifier,” *ArXiv160204938 Cs Stat*, Aug. 2016, Accessed: Mar. 10, 2020. [Online]. Available: http://arxiv.org/abs/1602.04938.

[60] A. Datta, S. Sen, and Y. Zick, “Algorithmic Transparency via Quantitative Input Influence:,” p. 20.

[61] "Explainable Artificial Intelligence." https://www.darpa.mil/program/explainable-artificial-intelligence (accessed Feb. 10, 2020).
[62] K. Sokol and P. Flach, "Explainability Fact Sheets: A Framework for Systematic Assessment of Explainable Approaches," *Proc. 2020 Conf. Fairness Account. Transpar.*, pp. 56–67, Jan. 2020, doi: 10.1145/3351095.3372870.
[63] B. Khaleghi, "The How of Explainable AI: Pre-modelling Explainability," *Medium*, Aug. 15, 2019. https://towardsdatascience.com/the-how-of-explainable-ai-pre-modelling-explainability-699150495fe4 (accessed Feb. 10, 2020).
[64] B. Khaleghi, "The How of Explainable AI: Explainable Modelling," *Medium*, Aug. 03, 2019. https://towardsdatascience.com/the-how-of-explainable-ai-explainable-modelling-55c8c43d7bed (accessed Feb. 10, 2020).
[65] B. Khaleghi, "The How of Explainable AI: Post-modelling Explainability," *Medium*, Aug. 03, 2019. https://towardsdatascience.com/the-how-of-explainable-ai-post-modelling-explainability-8b4cbc7adf5f (accessed Feb. 10, 2020).
[66] "Facets - Visualizations for ML datasets." https://pair-code.github.io/facets/ (accessed Feb. 11, 2020).
[67] "Same Stats, Different Graphs | Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems." https://dl.acm.org/doi/abs/10.1145/3025453.3025912 (accessed Mar. 02, 2020).
[68] T. Gebru *et al.*, "Datasheets for Datasets," *ArXiv180309010 Cs*, Mar. 2020, Accessed: Mar. 02, 2020. [Online]. Available: http://arxiv.org/abs/1803.09010.
[69] E. M. Bender and B. Friedman, "Data Statements for Natural Language Processing: Toward Mitigating System Bias and Enabling Better Science," *Trans. Assoc. Comput. Linguist.*, vol. 6, pp. 587–604, 2018, doi: 10.1162/tacl_a_00041.
[70] W. J. Murdoch, C. Singh, K. Kumbier, R. Abbasi-Asl, and B. Yu, "Interpretable machine learning: definitions, methods, and applications," *Proc. Natl. Acad. Sci.*, vol. 116, no. 44, pp. 22071–22080, Oct. 2019, doi: 10.1073/pnas.1900654116.
[71] Z. C. Lipton, "The Mythos of Model Interpretability," *ArXiv160603490 Cs Stat*, Mar. 2017, Accessed: Mar. 02, 2020. [Online]. Available: http://arxiv.org/abs/1606.03490.
[72] N. Papernot and P. McDaniel, "Deep k-Nearest Neighbors: Towards Confident, Interpretable and Robust Deep Learning," *ArXiv180304765 Cs Stat*, Mar. 2018, Accessed: Mar. 03, 2020. [Online]. Available: http://arxiv.org/abs/1803.04765.
[73] D. Alvarez-Melis and T. S. Jaakkola, "Towards Robust Interpretability with Self-Explaining Neural Networks," *ArXiv180607538 Cs Stat*, Dec. 2018, Accessed: Mar. 03, 2020. [Online]. Available: http://arxiv.org/abs/1806.07538.
[74] D. H. Park *et al.*, "Multimodal Explanations: Justifying Decisions and Pointing to the Evidence," *ArXiv180208129 Cs*, Feb. 2018, Accessed: Mar. 03, 2020. [Online]. Available: http://arxiv.org/abs/1802.08129.
[75] M. Hind *et al.*, "TED: Teaching AI to Explain its Decisions," *ArXiv181104896 Cs*, Jun. 2019, Accessed: Mar. 03, 2020. [Online]. Available: http://arxiv.org/abs/1811.04896.
[76] Q. Zhang, Y. N. Wu, and S.-C. Zhu, "Interpretable Convolutional Neural Networks," *ArXiv171000935 Cs*, Feb. 2018, Accessed: Mar. 04, 2020. [Online]. Available: http://arxiv.org/abs/1710.00935.
[77] C. Chen, O. Li, C. Tao, A. J. Barnett, J. Su, and C. Rudin, "This Looks Like That: Deep Learning for Interpretable Image Recognition," *ArXiv180610574 Cs Stat*, Dec. 2019, Accessed: Mar. 04, 2020. [Online]. Available: http://arxiv.org/abs/1806.10574.
[78] R. Ghaeini, X. Z. Fern, H. Shahbazi, and P. Tadepalli, "Saliency Learning: Teaching the Model Where to Pay Attention," *ArXiv190208649 Cs*, Apr. 2019, Accessed: Mar. 04, 2020. [Online]. Available: http://arxiv.org/abs/1902.08649.
[79] A. S. Ross, M. C. Hughes, and F. Doshi-Velez, "Right for the Right Reasons: Training Differentiable Models by Constraining their Explanations," *ArXiv170303717 Cs Stat*, May 2017, Accessed: Mar. 04, 2020. [Online]. Available: http://arxiv.org/abs/1703.03717.

[80] G. Alain and Y. Bengio, “Understanding intermediate layers using linear classifier probes,” *ArXiv161001644 Cs Stat*, Nov. 2018, Accessed: Mar. 04, 2020. [Online]. Available: http://arxiv.org/abs/1610.01644.

[81] B. Kim *et al.*, “Interpretability Beyond Feature Attribution: Quantitative Testing with Concept Activation Vectors (TCAV),” *ArXiv171111279 Stat*, Jun. 2018, Accessed: Mar. 04, 2020. [Online]. Available: http://arxiv.org/abs/1711.11279.

[82] R. Guidotti, A. Monreale, S. Ruggieri, F. Turini, D. Pedreschi, and F. Giannotti, “A Survey Of Methods For Explaining Black Box Models,” *ArXiv180201933 Cs*, Jun. 2018, Accessed: Mar. 04, 2020. [Online]. Available: http://arxiv.org/abs/1802.01933.

[83] M. Core, H. Lane, M. Lent, D. Gomboc, S. Solomon, and M. Rosenberg, “Building Explainable Artificial Intelligence Systems.,” Jan. 2006.

[84] L. A. Hendricks, Z. Akata, M. Rohrbach, J. Donahue, B. Schiele, and T. Darrell, “Generating Visual Explanations,” *ArXiv160308507 Cs*, Mar. 2016, Accessed: Mar. 04, 2020. [Online]. Available: http://arxiv.org/abs/1603.08507.

[85] M. Ancona, E. Ceolini, C. Öztireli, and M. Gross, “Towards better understanding of gradient-based attribution methods for Deep Neural Networks,” *ArXiv171106104 Cs Stat*, Mar. 2018, Accessed: May 03, 2020. [Online]. Available: http://arxiv.org/abs/1711.06104.

[86] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller, “Striving for Simplicity: The All Convolutional Net,” *ArXiv14126806 Cs*, Apr. 2015, Accessed: Mar 03, 2020. [Online]. Available: http://arxiv.org/abs/1412.6806.

[87] A. Shrikumar, P. Greenside, and A. Kundaje, “Learning Important Features Through Propagating Activation Differences,” *ArXiv170402685 Cs*, Oct. 2019, Accessed: Mar 03, 2020. [Online]. Available: http://arxiv.org/abs/1704.02685.

[88] M. Sundararajan, A. Taly, and Q. Yan, “Gradients of Counterfactuals,” *ArXiv161102639 Cs*, Nov. 2016, Accessed: May 03, 2020. [Online]. Available: http://arxiv.org/abs/1611.02639.

[89] “PlantVillage Dataset.” https://kaggle.com/emmarex/plantdisease (accessed Sep. 29, 2019).

[90] darrenl, *tzutalin/labelImg*. 2020.

[91] “imgaug — imgaug 0.4.0 documentation.” https://imgaug.readthedocs.io/en/latest/ (accessed Jan. 21, 2020).

[92] “pjreddie/darknet,” *GitHub*. https://github.com/pjreddie/darknet (accessed Jan. 21, 2020).

[93] 262588213843476, “convert pascal voc dataset to yolo format,” *Gist*. https://gist.github.com/M-Younus/ceaf66e11a9c0f555b66a75d5b557465 (accessed Jan. 21, 2020).

[94] A. Kathuria, “What’s new in YOLO v3?,” *Medium*, Apr. 29, 2018. https://towardsdatascience.com/yolo-v3-object-detection-53fb7d3bfe6b (accessed Dec. 10, 2019).

[95] J. Redmon and A. Farhadi, “YOLOv3: An Incremental Improvement,” p. 6.

[96] “AlexeyAB/darknet,” *GitHub*. https://github.com/AlexeyAB/darknet (accessed Nov. 12, 2019).