Lebanese University
Faculty of Engineering III
Department of Electrical and Electronic

PLANT DISEASE DETECTOR

Submitted in fulfillment of the requirements for
the COMMUNICATION AND MINI PROJECT
grade

Fourth year – Semester VII

Autumn 2021-2022

Prepared by : Katia Hourani 5435

Houssein Shreim 5793

Presented to: Prof. Youssef Harkouss

ACHNOWLEGMENTS:

Before getting further with our explanation, we would like to express our gratitude and appreciation to Dr. Hassan Shreim, the faculty's director for their efforts in keeping our faculty and its students ahead.

To Prof. Youssef Harkouss, chief of the Electrical Engineering department, whose suggestions and instructions have served as the major contributor towards the completion of this project.

To our dear friends, our second family, who never hesitated to help, and always supported us in these difficult times.

Finally, we would like to warmly thank our parents for their endless love, understanding, encouragement, and confidence.

TABLE OF CONTENTS:

ABSTRACT:

The aim of this project is to build a robot using a microcontroller and a microprocessor that can take care of your farm by detecting plant diseases and collecting data to be studied later. The project is divided into two main parts, one related to the disease detection, and the other is related to data collecting and safe navigation. The project started by attaining a clear knowledge about deep learning, choosing a good Deep Learning model, and training it on our custom dataset. RetinaNet model was chosen and trained on GoogleColab. In the second part of the project, our aim was to implement this model on an autonomous navigating robot, based on RaspberryPi and Arduino, that can safely navigate and locate plants. Several sensors were used to measure humidity, temperature, and light intensity in the field. This robot had to be connected to a server constantly to monitor it and receive picture from the field. Every purpose was achieved using Python language and its libraries.

LIST OF FIGURES:

GENERAL INTRODUCTION:

1. Introduction:

As agriculture struggles nowadays to support the rapidly growing population, plants' disease is amongst the most crucial reasons behind the reduction in production, where this loss is sometimes chronic. The study of plant diseases and the search for modern means to detect pre-diseased plant is the core of the plantation studies in modern time. In the last few years, traditional machine-learning algorithms have been widely used to realize disease detection solutions. Deep learning, refers to the use of artificial neural network architectures that contain quite many processing layers, has recently attracted a lot of attention intending to use for plant disease identification such as Retina Net deep learning model.

Inspired by the strength of deep learning, and how far we can go with technologies nowadays, especially when artificial intelligence is integrated with hardware systems.

2. Objective:

The objectives of our project are:

1) Build an autonomous compatible size robot that can detect plant diseases.
2) Implement safe navigation algorithm on the robot avoiding any damage to the plants.
3) Test the strength of the state-of-art Retina Net DL model on a custom dataset to be deployed on the robot.

3. Plan:

The building process is divided into two main parts in which one deals with the deep learning plant disease model and the other one is responsible for robot's navigation and data collection from the field. These data, like humidity, temperature and light intensity, are to be studied later for further analysis about the status of the planting field. This report is divided into two main chapters. The first chapter will explain the concepts behind neural networks and spots light on the Convolution neural network and each layer embedded in it. The second chapter is about the hardware implementation which introduces the used microprocessor and microcontroller, Raspberry-pi and Arduino, and the used sensors and actuators. Then the logic behind each code written on the micro controller and processor is explained.

CHAPTER 1: CONVOLUTIONAL NEURAL NETWORK

1. Introduction:

The term Deep Learning or Deep Neural Network refers to Artificial Neural Networks (ANN) with multi layers . Over the last few decades, it has been considered to be one of the most powerful tools, and has become very popular in the literature as it is able to handle a huge amount of data. The interest in having deeper hidden layers has recently begun to surpass classical methods performance in different fields; especially in pattern recognition. One of the most popular deep neural networks is the Convolutional Neural Network (CNN).

Artificial neural networks are generally a chain of nodes associated with each other via the link from which they start interacting accordingly. Neurons perform operations and carry that result. The optimum goal for a neural network is to find an optimized set of parameters (weights and biases) that best maps the features with their corresponding labels. The process of learning these parameters through a given dataset is called training.

2. Machine learning:

Machine learning is programming computers to optimize a performance criterion using example data or past experience.

The goal of machine learning is to develop methods that can automatically detect patterns in data, and then to use the uncovered patterns to predict future data or other outcomes of interest.

The field of pattern recognition is concerned with the automatic discovery of regularities in data through the use of computer algorithms and with the use of these regularities to take actions, which is the main interest of our project.

3. Deep learning:

Plant diseases are one of the main factors that can cause significant crop damages and yield losses. Deep learning has recently attracted a lot of attention to use for plant disease identification. The recent studies used deep learning techniques to diagnose plant diseases in an attempt to identify plant diseases.

Deep learning is a subfield of machine learning which attempts to learn high-level abstractions in data by utilizing hierarchical architectures.

There are mainly three important reasons for the booming of deep learning today:

i. The dramatically increased chip processing abilities (GPU units).
ii. The significantly lowered cost of computing hardware.
iii. The considerable advances in the machine learning algorithms.

4. Neural network:

4.1 Definition:

A neural network is a series of algorithms that endeavors to recognize underlying relationships in a set of data through a process that mimics the way the human brain operates. In this sense, neural networks refer to systems of neurons, either organic or artificial in nature.

Neural networks can adapt to changing input; so, the network generates the best possible result without needing to redesign the output criteria. The concept of neural networks, which has its roots in artificial intelligence, is swiftly gaining popularity in the development of trading systems.

Neural networks consist of:

- Input
- Output
- Weights and biases
- Activation function(s)

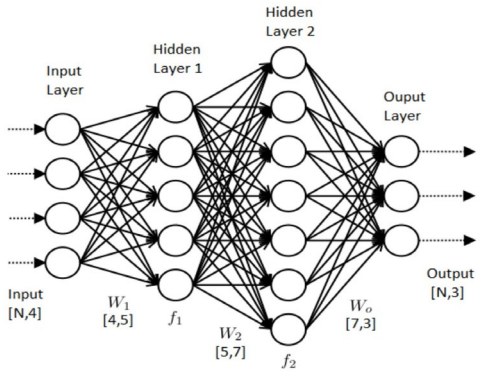

Figure 1: Neural network connections

4.2 Single layer perceptron (SLP) model:

The simplest of the neural network models, SLP, is viewed as the beginning of artificial intelligence and provided inspiration in developing other neural network models and machine learning models. The SLP architecture is such that a single neuron is connected by many synapses, each of which contains a weight.

A single layer perceptron (SLP) is a feed-forward network based on a threshold transfer function. SLP is the simplest type of artificial neural networks and can only classify linearly separable cases with a binary target (1, 0).

The weights affect the output of the neuron. The aggregate values of the weights multiplied by the input are then summed within the neuron and then fed into an activation function, the standard function being the logistic function:

$$\textbf{Input vector } [\boldsymbol{x_0}, \boldsymbol{x_1}, \boldsymbol{x_2}, \dots, \boldsymbol{x_n}]^T$$

$$\textbf{Weights vectors } [\boldsymbol{w_0}, \boldsymbol{w_1}, \boldsymbol{w_2}, \dots, \boldsymbol{w_n}]$$

The output of the function is given by:

$$\boldsymbol{y = f(x, w^T) = f\left(\sum_{i=0}^{i=n} x_i w_i\right)}$$

Generally, $\boldsymbol{x_0} = \boldsymbol{1}$ $\boldsymbol{and}$ $\boldsymbol{w_0} = \boldsymbol{b}$, where b is called the bias.

Then the scheme and equation of one neuron in a layer become

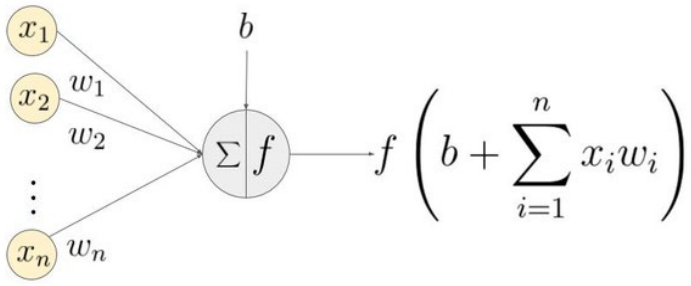

Figure 2: Scheme of one neuron in a layer

When dealing with big datasets, the number of features is large (n will be big), and so it is better to use a vector notation for the features and the weights, as follows:

$$\boldsymbol{x} = \begin{pmatrix} \boldsymbol{x_1} \\ \vdots \\ \boldsymbol{x_n} \end{pmatrix}$$

Where we have indicated the vector with a boldfaced x. For the weights, we use the same notation:

$$\boldsymbol{w} = \begin{pmatrix} \boldsymbol{w_1} \\ \vdots \\ \boldsymbol{w_n} \end{pmatrix}$$

For consistency with formulas that we will use later, to multiply x and w, we will use matrix multiplication notation, and, therefore, we will write:

$$\boldsymbol{w^T x} = (\boldsymbol{w_1} \quad \cdots \quad \boldsymbol{w_n}) \begin{pmatrix} \boldsymbol{x_1} \\ \vdots \\ \boldsymbol{x_n} \end{pmatrix} = \boldsymbol{w_1 x_1 + w_2 x_2 + \cdots + w_n x_n}$$

Where $\boldsymbol{w^T}$ indicates the transpose of w. z can then be written with this vector notation as:

$$\boldsymbol{z = w^T x}$$

And the neuron output $\hat{\boldsymbol{y}}$ as:

$$\boldsymbol{\hat{y} = f(z) = f(w^T x + b)}$$

In brief,

- $\hat{y}$→the neuron output
- *f(z)*→ activation function(or transfer function) applied to z
- *w*→ weights
- *b*→bias

4.3 Multiplayer perceptron model (MLP):

Very similar to SLP, the multilayer perceptron (MLP) model features multiple layers that are interconnected in such a way that they form a feed-forward neural network. Each neuron in one layer has directed connections to the neurons of a separate layer.

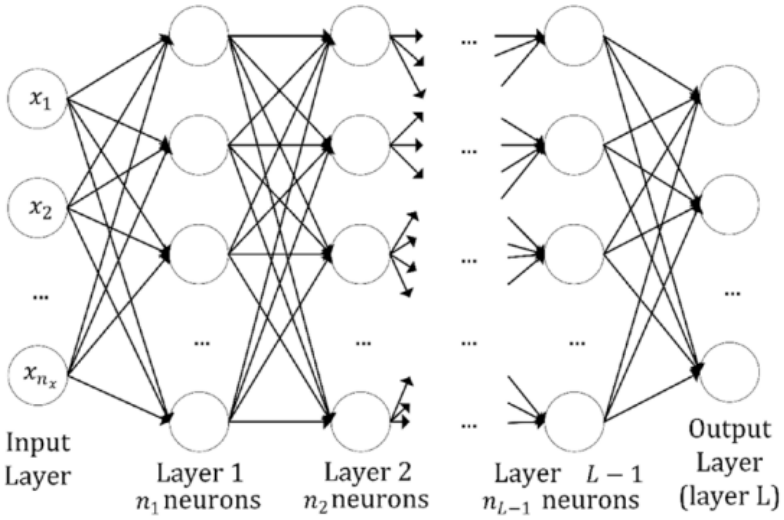

Figure 3: Scheme of MLP

To jump from one neuron to this is quite a big step. To build the model, we will have to work with matrix formalism, and therefore, we must get all the matrix dimensions right. First, here are some new notations.

- L: Number of hidden layers, excluding the input layer but including the output layer.
- $\boldsymbol{nl}$: Number of neurons in layer $\boldsymbol{l}$.

Where, by convention, we defined $\boldsymbol{n_0}=\boldsymbol{n_x}$. Each connection between two neurons will have its own weight. Let's indicate the weight between neuron $\boldsymbol{i}$ in layer $\boldsymbol{l}$ and neuron $\boldsymbol{j}$ in layer $\boldsymbol{l}-\boldsymbol{1}$ with $\boldsymbol{wij}^{[\boldsymbol{l}]}$.

- *First hidden layer:*

The first two layers of a generic neural network, with the weights of the connections between the first neuron in the input layers and the others in the second layer. All other neurons and connections are drawn in light gray, to make the diagram clearer. The weights and biases between the input layer and layer 1 can be written as a matrix, as follows:

$$\boldsymbol{W}^{[1]} = \begin{pmatrix} \boldsymbol{w_{11}}^{[1]} & \cdots & \boldsymbol{w_{1n_x}}^{[1]} \\ \vdots & \ddots & \vdots \\ \boldsymbol{w_{n_1 1}}^{[1]} & \cdots & \boldsymbol{w_{n_1 n_x}}^{[1]} \end{pmatrix}$$

$$b^{[1]} = \begin{pmatrix} b_1{}^{[1]} \\ \vdots \\ \vdots \\ b_{n_1}{}^{[1]} \end{pmatrix}$$

This means that our matrix $W^{[1]}$ has dimensions $n_1 x n_x$. Of course, this can be generalized between any two layers l and $l-1$, meaning that the weight matrix between two adjacent layers l and $l-1$, indicated by $W^{[1]}$, will have dimensions $n_l \, x \, n_{l-1}$. By convention, $n_0 = n_x$ is the number of input features (not the number of observations that we indicate with m).

$$W^{[l]} = \begin{pmatrix} w_{11}{}^{[l]} & \cdots & w_{1n_{l-1}}{}^{[l]} \\ \vdots & \ddots & \vdots \\ w_{n_l1}{}^{[l]} & \cdots & w_{n_ln_{l-1}}{}^{[l]} \end{pmatrix}$$

$$b^{[l]} = \begin{pmatrix} b_1{}^{[l]} \\ \vdots \\ \vdots \\ b_{n_l}{}^{[l]} \end{pmatrix}$$

Note the weight matrix between two adjacent layers l and $l-1$, which we indicate with $W^{[l]}$, will have dimensions $n_l x n_{l-1}$, where, by convention, $n_0 = n_x$ is the number of input features.

The bias will be a matrix this time. Remember that each neuron that receives inputs will have its own bias, so when considering our two layers, l and $l-1$, we will require n_l different values of b. We will indicate this matrix with b $[l]$, and it will have dimensions $n_l \times 1$. Note The bias matrix for two adjacent layers l and $l-1$, which we indicate with $b^{[l]}$, will have dimensions $n_l \times 1$.

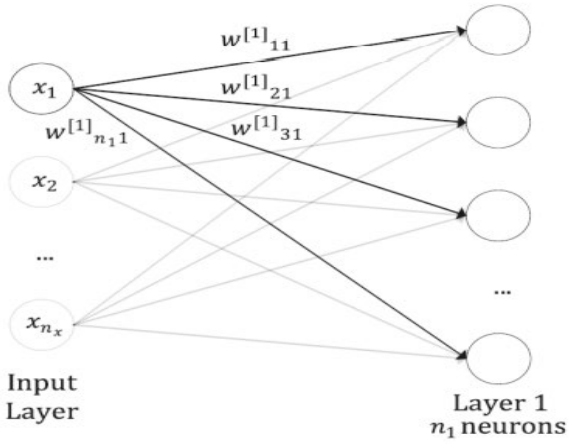

Figure 4: Scheme of the first hidden layer.

4.4 Output of neurons:

Now let's start considering the output of our neurons. To begin, we will consider the neuron of the first layer (remember that our input layer is by definition layer 0). Let's indicate its output with $\boldsymbol{A^{[1]}}$ and assume that all neurons in layer $\boldsymbol{l}$ use the same activation function, which we will indicate by $\boldsymbol{g^{[1]}}$. Then we will have:

$$\boldsymbol{A^{[1]} = g^{[1]}\left(Z^{[1]}\right) = g^{[1]}\left(\sum_{i=0}^{i=n_x} w_{ij}{}^{[1]} x_i + b_j{}^{[1]}\right)}$$

As you can imagine, we want to have a matrix for all the output of layer 1, so we will use the notation

$$\boldsymbol{Z^{[1]} = W^{[1]} X + b^{[1]}}$$

$$\boldsymbol{Z^{[1]} = \begin{pmatrix} z_1{}^{[1]} \\ \vdots \\ \vdots \\ z_{n_1}{}^{[1]} \end{pmatrix}}$$

$$\boldsymbol{A^{[1]} = g^{[1]}\left(Z^{[1]}\right) = \begin{pmatrix} a_1{}^{[1]} \\ \vdots \\ \vdots \\ a_{n_1}{}^{[1]} \end{pmatrix}}$$

Where $\boldsymbol{Z^{[1]}}$ will have dimensions $\boldsymbol{n_1 \times 1}$, and where with $\boldsymbol{X}$, we have indicated our matrix with all our observations (rows for the features, and columns for observations). We assume here that all neurons in layer $\boldsymbol{l}$ will use the same activation function that we will indicate with $\boldsymbol{g^{[l]}}$. We can easily generalize the previous equation for a layer $\boldsymbol{l}$.

$$\boldsymbol{Z^{[l]} = W^{[l]} A^{[l-1]} + b^{[l]}}$$

$$\boldsymbol{Z^{[l]} = \begin{pmatrix} z_1^{[l]} \\ \vdots \\ \vdots \\ z_{n_l}^{[l]} \end{pmatrix}}$$

Because layer $\boldsymbol{l}$ will get its input from layer $\boldsymbol{l-1}$. We just need to substitute $\boldsymbol{X}$ with $\boldsymbol{Z^{[l-1]}}$. Will have dimensions $\boldsymbol{n_l \times 1}$.Our output in matrix form will then be:

$$\boldsymbol{A^{[l]} = g^{[l]}(Z^{[l]})} \qquad \boldsymbol{A^{[l]} = \begin{pmatrix} a_1^{[l]} \\ \vdots \\ \vdots \\ a_{n_l}^{[l]} \end{pmatrix}} \qquad \boldsymbol{\hat{Y} = A^{[L]}}$$

Where the activation function acts, as usual, element by element.

Following is a summary of the dimensions of all the matrices we have described so far:

- $\boldsymbol{W^{[l]}}$ has dimensions $\boldsymbol{n_l x n_{l-1}}$.
- $\boldsymbol{b^{[l]}}$ has dimensions $\boldsymbol{n_l x 1}$.
- $\boldsymbol{Z^{[l-1]}}$ has dimensions $\boldsymbol{n_{l-1}\, x1}$.
- $\boldsymbol{Z^{[l]}}$ has dimensions $\boldsymbol{n_l x 1}$.
- $\boldsymbol{A^{[l]}}$ has dimensions $\boldsymbol{n_l x 1}$.

5. Concept of convolutional neural network:

5.1 Definition:

One of the most popular deep neural networks is the Convolutional Neural Network (CNN). It takes this name from mathematical linear operation between matrixes called convolution. CNN

have multiple layers; including convolutional layer, non-linearity layer, pooling layer and fully-connected layer. The convolutional and fully-connected layers have parameters but pooling and non-linearity layers don't have parameters.

Convolutional Neural Network had ground breaking results over the past decade in a variety of fields related to pattern recognition; from image processing to voice recognition. The most beneficial aspect of CNNs is reducing the number of parameters in ANN.

Convolutional Neural Networks (CNNs) are currently adopted to solve an ever greater number of problems, ranging from speech recognition to image classification and segmentation. They expect and preserve the spatial relationship between pixels by learning internal feature representations using small squares of input data. Feature are learned and used across the whole image, allowing for the objects in the images to be shifted or translated in the scene and still detectable by the network. It is this reason why the network is so useful for object recognition in photographs, picking out digits, faces, objects and so on with varying orientation.

In summary, below are some of the benefits of using convolutional neural networks:

- They use fewer parameters (weights) to learn than a fully connected network.
- They are designed to be invariant to object position and distortion in the scene.
- They automatically learn and generalize features from the input domain.

5.2 CNN layers:

i. Convolutional layer:

Convo layer is sometimes called feature extractor layer because features of the image are get extracted within this layer.

First of all, a part of image is connected to Convo layer to perform convolution operation as we saw earlier and calculating the dot product between receptive field and the filter. Result of the operation is single integer of the output volume. Then we slide the filter over the next receptive field of the same input image by a Stride and do the same operation again.

It contains :

• **Filters**: The filters are essentially the neurons of the layer. They have both weighted inputs and generate an output value like a neuron.

• **Feature Maps:** It is the output of one filter applied to the previous layer. A given filter is drawn across the entire previous layer, moved one pixel at a time. Each position results in an activation of the neuron and the output are collected in the feature map.

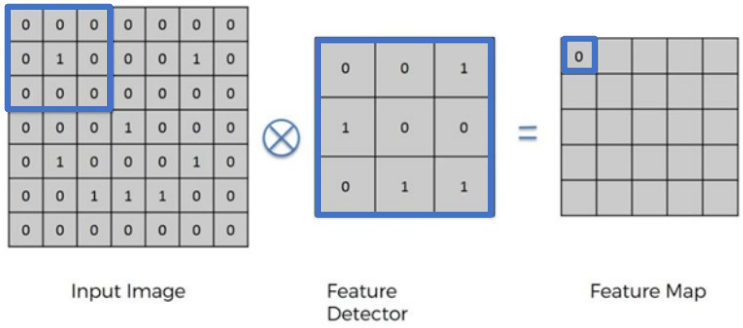

Figure 5: First filter in convolutional layer

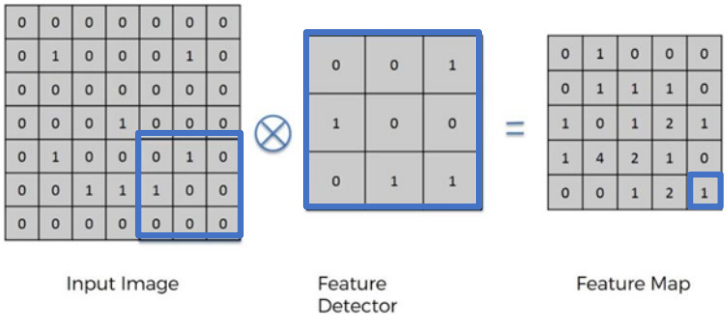

Figure 6: Last filter in convolutional layer.

ii. Pooling layers:

Pooling layers are used to reduce the dimensions of the feature maps. Thus, it reduces the number of parameters to learn and the amount of computation performed in the network.

The pooling layer summarizes the features present in a region of the feature map generated by a convolution layer. So, further operations are performed on summarized features instead of precisely positioned features generated by the convolution layer. This makes the model more robust to variations in the position of the features in the input image.

There are different types of pooling layers:

- *Max pooling:*

Max pooling is a pooling operation that selects the maximum element from the region of the feature map covered by the filter. Thus, the output after max-pooling layer would be a feature map containing the most prominent features of the previous feature map.

- *Min pooling:*

In this type of pooling, the summary of the features in a region is represented by the minimum value in that region. It is mostly used when the image has a light background since min pooling will select darker pixels.

- *Average pooling:*

Average pooling computes the average of the elements present in the region of feature map covered by the filter. Thus, while max pooling gives the most prominent feature in a particular patch of the feature map, average pooling gives the average of features present in a patch.

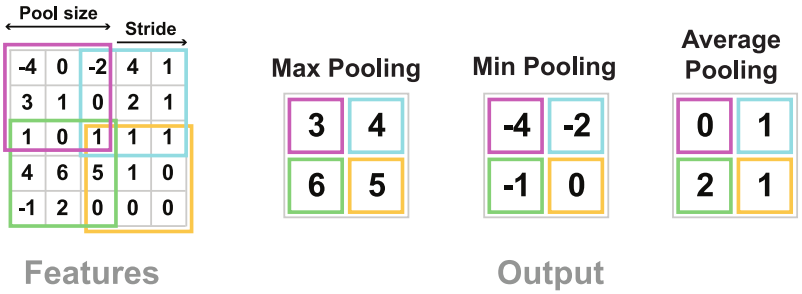

Figure 7 Different types of pooling layers.

iii. Fully-connected layers:

Fully connected layers are the normal flat feedforward neural network layer. These layers may have a nonlinear activation function in order to output probabilities of class predictions.

Fully connected layers are used at the end of the network after feature extraction and consolidation have been performed by the convolutional and pooling layers. They are used to create final nonlinear combinations of features and for making predictions by the network.

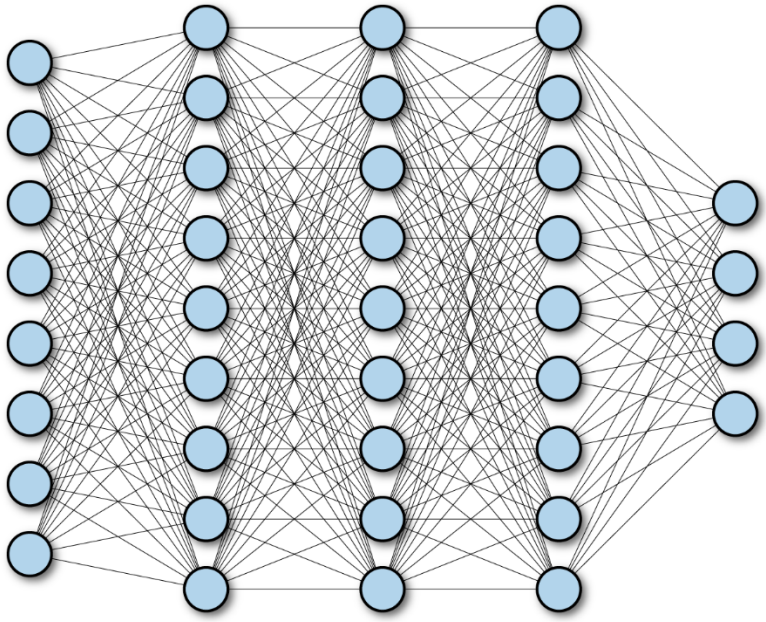

Figure 8 Fully-connected layers.

5.3 Activation function:

The weighted inputs are summed and passed through an activation function, sometimes called a transfer function.

An activation function is a simple mapping of summed weighted input to the output of the neuron. It is called an activation function because it governs the threshold at which the neuron is activated and the strength of the output signal.

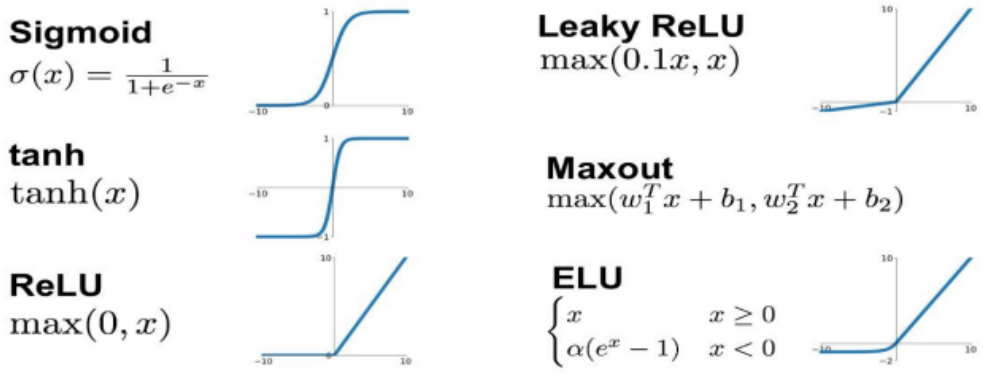

Figure 9: Some activation function.

5.4 Loss functions:

The loss function is one of the important components of Neural Networks. Loss is nothing but a prediction error of Neural Net. And the method to calculate the loss is called loss function.

In simple words, the loss is used to calculate the gradients. And gradients are used to update the weights of the Neural Net. This is how a Neural Net is trained.

There are serval loss functions like:

- Mean Squared Error (MSE).
- Binary Crossentropy (BCE).
- Categorical Crossentropy (CC).
- Sparse Categorical Crossentropy (SCC).

6. Conclusion :

Deep Learning is a sub-field of machine learning which we have previously discuss; which is part of the broader field of Artificial Intelligence. Deep Learning process consists in stacking many “hidden layers” of neurons so they can identify and learn complex patterns by adjusting their weights. We use a cost function to check how wrong the results are and then use backward propagation to readjust the weights. Convolutional Neural Networks are gaining popularity, especially for image recognition. Based on ANNs they add a special convolution layer which uses filters to create a simplified map representing the main characteristics of the image.

CHAPTER 2: MODEL AND IMPLEMENTATION

1. Introduction:

The ability to transform what we have visioned about this robot and its abilities to a working prototype required multiple skills in different sectors like deep learning, microcontrollers, and microprocessors. These three together represent the main base of the project. Working on the chassis, autonomous navigation, and the classification model began and progressed at a good pace. Running into major challenges and delays was totally expected especially in the integration of the microprocessor and the microcontroller together.

2. RetinaNet model:

2.1 The concept of RetinaNet:

Retina-Net is a single, unified network composed of a backbone network and two task-specific subnetworks. The backbone is responsible for computing a conv feature map over an entire input image and is an off-the-self convolution network. The first subnet performs classification on the backbones output; the second subnet performs convolution bounding box regression.

- ***Backbone:***

Feature Pyramid network built on top of ResNet. However, we can use any classifier of your choice; just follow the instructions given in FPN section when designing the network.

- ***Classification subnet:***

It predicts the probability of object presence at each spatial position for each of the A anchors and K object classes. Takes a input feature map with C channels from a pyramid level, the subnet applies four 3x3 conv layers, each with C filters and each followed by ReLU activations. Finally, sigmoid activations are attached to the outputs. Focal loss is applied as the loss function.

- ***Box Regression Subnet:***

Similar to classification net used but the parameters are not shared. Outputs the object location with respect to anchor box if an object exists. smooth_l1_loss with sigma equal to 3 is applied as the loss function to this part of the sub-network.

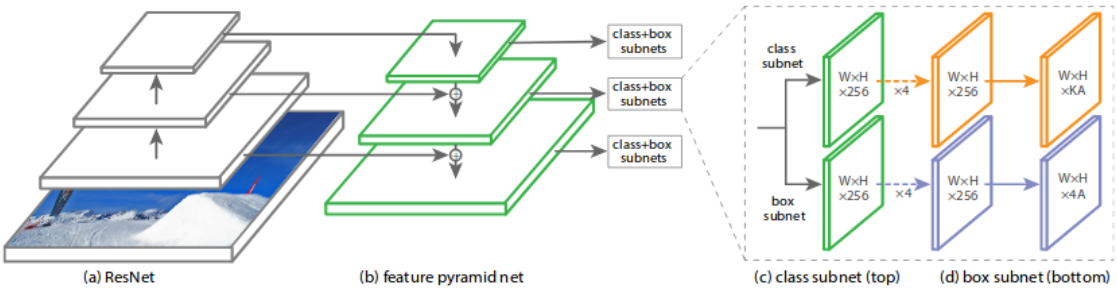

Figure 10: The concept of RetinaNet

2.2 From NN to RetinaNet:

Starting from a fully connected neural network, researchers noticed the bottlenecks of such network regarding the number of parameters and computational power needed. The concept of convolution network and its layers came to solve existing problems and limitations. CNN became the core of every other DL network later. Starting from the state-of-art CNN then, many developments were made regarding the region proposal to lower the computational power and transition from image classification to object detection. These proposals were first made using 'selective search algorithms' then using Region Proposal Network RPN. This development has led to the creation of R-CNN, Fast R-CNN, and Faster R-CNN. The latter has become the state-of-art in the object detection.

Faster R-CNN is known to have the RPN network which is responsible for the region proposal and it is better than the older selective algorithm. But we still have to problems, in which if we were able to tackle, we will reach an extremely fast one stage algorithm, and this is what happened to reach the Retina Net model. The first problem was the loss of semantic information due to the convolution layer present before starting the RPN network. The other problem is that we could not use all the labels due to the significant effect of unbalanced datasets. The first problem, the loss of semantic information was solved using the Feature Pyramid Network. The second problem, the imbalance problem, was solved with using the Focal Loss.

Starting with the loss of semantic information, authors thought of a way of using the hidden layers of the ResNet networks. An important note to state here is that RPN uses a feature map as input, and ResNet is a CNN network that outputs a feature map of the original image. But since we only

used the last feature map produced by the ResNet, small object of the original image was lost. Here is where authors thought of not only using the last layer of the ResNet, but many layers inside it which also output feature maps, but with more semantic information. Then pass these feature maps through the RPN and combine the results using non-maxima suppression. The below image represents the concept is a visual way.

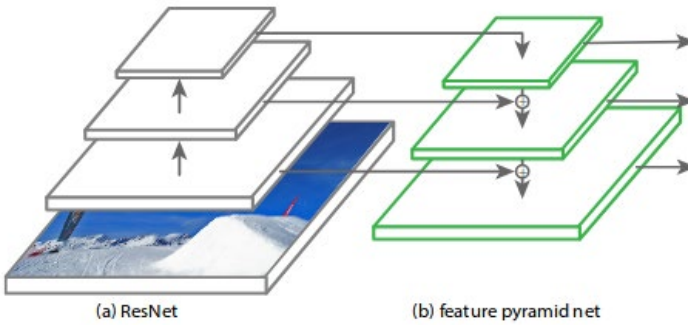

Figure 11: ResNet and feature pyramid net.

Class Imbalance is a common problem in machine learning, especially in classification problems. Imbalance data can hamper our model accuracy.

One-stage detectors that are applied over a regular, dense sampling of possible object locations have the potential to be faster and simpler but have trailed the accuracy of two-stage detectors because of extreme class imbalance encountered during training.

Focal loss is the reshaping of cross entropy loss such that it down-weights the loss assigned to well-classified examples. The novel focal loss focuses training on a sparse set of hard examples and prevents the vast number of easy negatives from overwhelming the detector during training.

$$\mathrm{FL}(p_t) = -\alpha_t(1-p_t)^{\gamma}\log(p_t).$$

Focal loss adds very less weight to well classified examples and large weight to miss-classified or hard classified examples.

This is the basic intuition behind designing Focal loss. The authors have tested different values of alpha and gamma and final settled with the above-mentioned values.

3. Hardware used:

3.1 Arduino Leonardo:

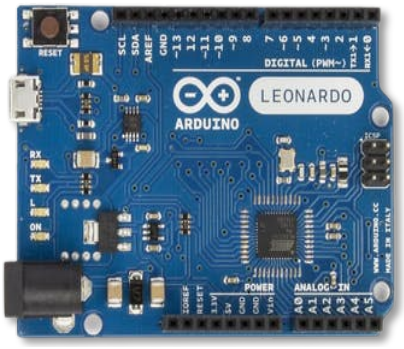

Figure 12: : Arduino Leonardo

We need a brain for the robot that reads values from the sensors and tells the motors what to do. A microcontroller, or µC for short, does this exact job.

The ***Arduino Leonardo*** is a microcontroller board based on the ATmega32u4 .It has 20 digital input/output pins (of which 7 can be used as PWM outputs and 12 as analog inputs), a 16 MHz crystal oscillator, a micro USB connection, a power jack, an ICSP header, and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started.

3.2 RaspberryPi 4 model B:

Figure 13: RaspberryPi 4 model B

Raspberry Pi is a series of small single-board computers (SBCs) developed by the Raspberry Pi Foundation its target market for uses such as robotics. It is widely used in many areas, such as for weather monitoring, because of its low cost, modularity, and open design. It is typically used by computer and electronic hobbyists, due to its adoption of HDMI and USB devices.

Raspberry Pi 4 Model B contains a 1.5 GHz 64-bit quad core ARM Cortex-A72 processor, on-board 802.11ac Wi-Fi, Bluetooth 5, full gigabit Ethernet, two USB 2.0 ports, two USB 3.0 ports, 2-8 GB of RAM, and dual-monitor support via a pair of micro HDMI (HDMI Type D) ports for up to 4K resolution.

3.3 DHT11 temperature and humidity sensor:

The ***DHT11*** is a basic, ultra-low-cost digital temperature and humidity sensor. It uses a capacitive humidity sensor and a thermistor to measure the surrounding air, and spits out a digital signal on the data pin (no analog input pins needed). It's fairly simple to use, but requires careful timing to grab data.

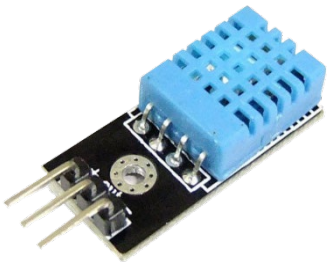

Figure 14: DHT11 temperature and humidity sensor

3.4 Ultrasonic Distance Sensor:

The ***HC-SR04 Ultrasonic Distance Sensor*** is a sensor used for detecting the distance to an object using sonar. The HC-SR04 uses non-contact ultrasound sonar to measure the distance to an object, and consists of two ultrasonic transmitters (basically speakers), a receiver, and a control circuit.

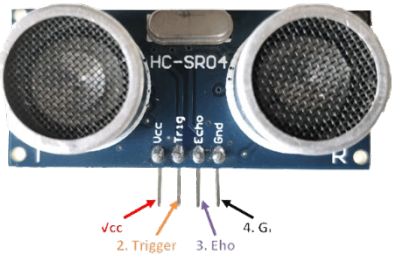

Figure 15: Ultrasonic Distance Sensor

3.5 LDR Sensor:

Photoresistors, also known as ***light dependent resistors (LDR),*** are light sensitive devices most often used to indicate the presence or absence of light, or to measure the light intensity. LDRs have a sensitivity that varies with the wavelength of the light applied and are nonlinear devices.

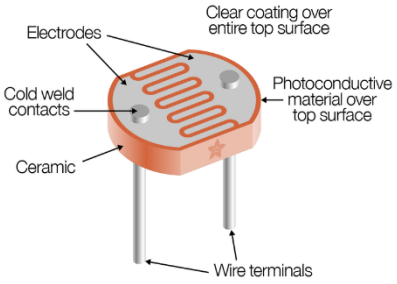

Figure 16: LDR Sensors

3.6 Geared DC Motor:

Geared DC motors can be defined as an extension of DC motor which already had its Insight details demystified here. A geared DC Motor has a gear assembly attached to the motor. This concept where gears reduce the speed of the vehicle but increase its torque is known as gear reduction.

Figure 17: Geared DC Motor

3.7 The schematic of our circuit:

The sensors' and actuators' connections are not necessarily how it was wired in the actual project. But it gives an idea of how to wire each component since they are the same components used. There is no single way of connecting them but make sure to edit the first part of the Arduino code where pins were initialized to match your connections.

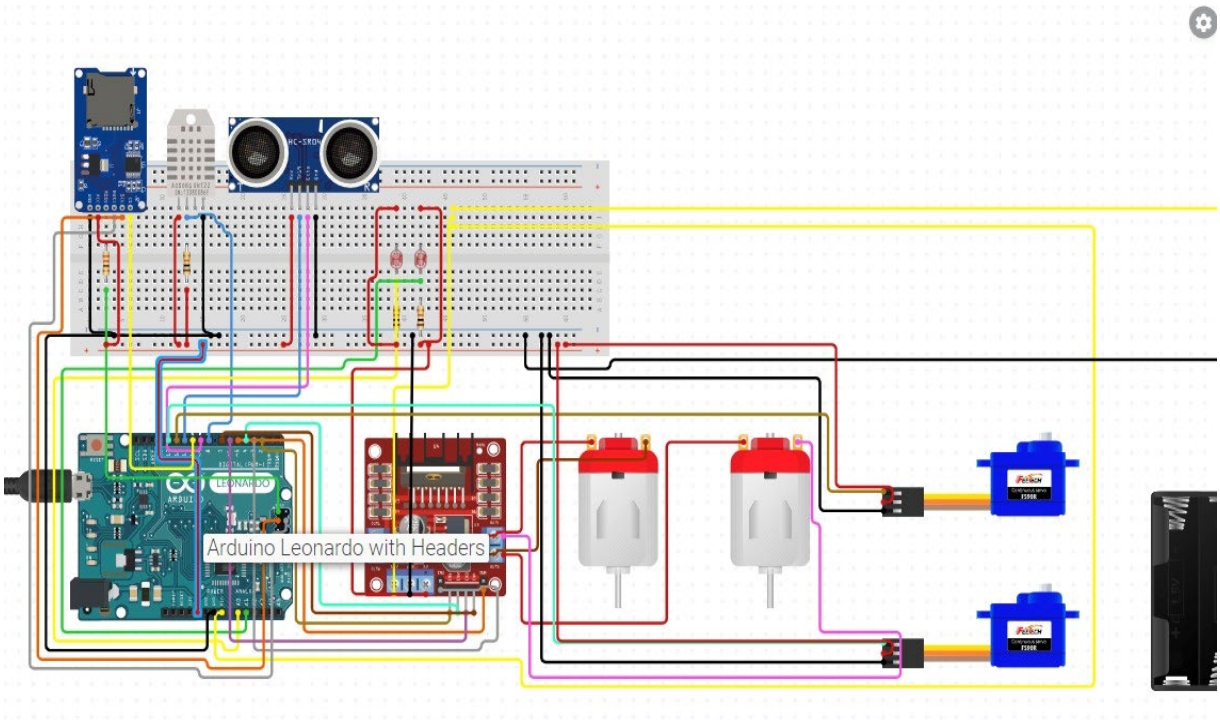

Figure 18: Schematic of the circuit.

4. Setting up the project:

This project can be presumably divided into two parts. The first part is the detection model built using python and with the help of git repositories. The model used will be explained in detail. The other part is what gives the robot the ability to move in the field, collect data to be processed later, and detect whatever diseases are encountered. This part depends on Arduino and raspberry pi, which are boards that depend on a microcontroller and microprocessor respectively. Python was also used with RaspberryPi to achieve the communication with Arduino for sending and receiving data.

4.1 Navigation and data collection:

i. Python libraries:

Multiple libraries were used, some of them to process images for better performance, another to build the detection model we trained in order to test the images from the field. We also used libraries for communication between the RPI and Arduino, and to allow the python script to be able to engage with the operating systems.

These libraries are:

- ***Keras***
- ***Numpy***
- ***Socket***
- ***Time***
- ***Opencv***
- ***Zipfile***
- ***Glob***
- ***Os***
- ***Matplotlib***
- ***Serial***

OPENCV library is the one responsible for capturing image via the webcam and processing them to adjust brightness and any other features. ***MATPLOTLIB*** is also used with OPENCV for better control on images and to show them for later debugging.

Both ***OS*** and ***GLOB*** allow the python script to reach files found in the operating system's storage, create files, and read them. Using these libraries, we locate captured image on the Ubuntu and Windows operating systems of the server and the RPI. ***ZIPFILE*** library does what its name states, zipping and non-zipping files.

KERAS is the library used to build the already trained model and to prepare it for image detection and classification.

SOCKET and ***SERIAL*** ensure the connection and communication between RPI-Arduino and RPI-Server respectively. They allow the transfer of data between different components and systems in the project.

TIME libraries help in adding time delays in the python scripts which is essential in most projects.

NUMPY used to transform images to form readable by the python script, which is numerical arrays. So, it changes colored images to numerical arrays before entering the prediction model.

```
import matplotlib.pyplot as plt
import cv2
import os
import numpy as np
import time
import glob
import tensorflow as tf
import zipfile
import serial
import socket
```

ii. RaspberryPi side:

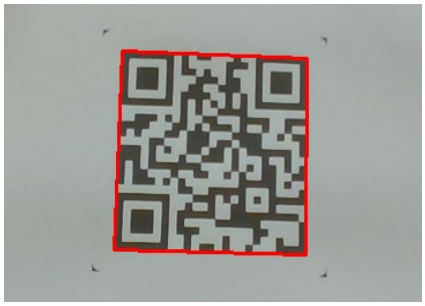

Figure 19: Detection of QR code

This robot can navigate in a field to locate plants and check their health status. To achieve that without any further complexity like using GPS or other complex methods, we had to come up with a way that is rather simple and effective.

Knowing that python offers a great variety of libraries, one of them is the PYZBAR library which can detect QR codes in images taken and decode them. So, we chose our robot to follow an algorithm where it randomly navigates and stops when it detects QR codes put under each plant in which it is encoded with the name of the plant.

Raspberry pi is responsible for running the python code to detect the QR code, and it accordingly send command to Arduino through serial communication so that Arduino can decide when to stop for a while , when to rotate a little bit, or when to navigate randomly.

In the below snippet of code, we are initializing SERIAL communication between Arduino and raspberry-pi, SOCKET connection between RPI and the PC. And using OPENCV library we are initializing the USB camera connected with the RPI to start taking images.

```
#--------------initializing Serial------------
if __name__ == '__main__':
    ser = serial.Serial('/dev/ttyACM0', 9600,
timeout=1)
    ser.reset_input_buffer()


#--------------socket connection----------

s=socket.socket()
host=''
port=8080
s.bind((host,port))
s.listen(1)
print(host)
print("waiting for host to connect.....")
conn , addr =s.accept()
print(addr, " has connected")
ser.write(b"12")
```

```
#--------------initializing camera-------
----------
def capture1():

    global cap

    cap = cv.VideoCapture(0)




    if not cap.isOpened():
        raise IOError("Cannot open
webcam")

    cap.set(cv.CAP_PROP_FRAME_WIDTH,640)
    cap.set(cv.CAP_PROP_FRAME_HEIGHT,480)

capture1()
```

In the below snippet of code, we wrote a function to calculate the area of the barcode for distance calculation purposes, a function that return the position of the barcode, and a function to check if a plant is already tested today.

```
#--------------Functions-------------------------
def PolyArea(x,y):
    return 0.5*np.abs(np.dot(x,np.roll(y,1))-np.dot(y,np.roll(x,1)))
```

```
def center(x1,x2):

    if(  (((x1+x2)/2 -320 ) < 100) & (((x1+x2)/2 -320 ) > -100 )):
        return 0

    if(  (((x1+x2)/2 -320 ) > 100) ):
        return 1

    if( (((x1+x2)/2 -320 ) < -100 ) ):
        return 2

def check(plantnumber):
    for i in checked:
        if i==plantnumber:
            return True
    return False
```

Here we are processing every image with the PYZBAR library to check the presence of QR codes in the vision scope of the robot. And according to the result, a numerical command is sent to the ARDUINO to navigate accordingly.

```
# -------------start detecting barcode------------
ser.write(b"0")
print("once")
while True:
    i=0
    for i in range(12):
        success, img = cap.read()
    success, img = cap.read()

    if finished:

        lapse=time.time()
        ser.write(b"0")
        print("sending :")
        print("0")

    for barcode in decode(img, symbols=[ZBarSymbol.QRCODE]):
        mydata=barcode.data.decode('utf-8')
        #print(barcode)
```

When the RPI detects a QR code and sends a command for Arduino to stop, several images are taken to be processed for any probable disease. But raspberry pi is relatively weak to handle disease detection. Even though it is not impossible, to keep up the performance of the robot, we used a socket connection between RPI and a personal PC, which represents a server, or a computer found near the field. So captured images are sent to the pc which in turn uses the already trained Retina net model to check the images for any disease.

Below in the snippet of code where commands are sent to Arduino according to the Realtime situation:

```
if center(x[0],x[2])==0 and PolyArea(x,y)> 30000 :
```

```
                #ready to scan
                ser.write(b"1")
                print("sending: ")
                print("1")
                #call capAndScan() function
                checked.append(mydata)
                capAndSend()
```

One note to state is that QR codes cannot be detected more than once in the same day, since we keep track of them and flush the history after a specified period.

iii. Arduino side:

As explained above, RPI decides when to navigate, stop, turn or take any necessary move. Arduino them is responsible to control the motors and read the data from sensors.

We started by importing libraries that help control and decode sensors' data like the libraries below. And we defined some variable to be used later.

```
#include <dht.h>
#include <Servo.h>
#include <SPI.h>
#include <SD.h>
const int chipSelect = 10;
File dataFile;
#define echoPin 2 // attach pin D2 Arduino to pin Echo of HC-SR04
#define trigPin 3 //attach pin D3 Arduino to pin Trig of HC-SR04
int I1,I2;
#define dht_apin A4 // Analog Pin sensor is connected to

dht DHT;

Servo updownServo, RLServo;

int check=0;
int incomingByte = 0;
long duration;
int distance;
```

Starting with the setup function, we initialized the serial communication and assured the presence of the SD card which will hold any data collected from the field. Then we initialized the Servo motors, DC motors to initial values.

```
void setup() {
  Serial.begin(9600);
   while (!Serial) {
 ; // wait for serial port to connect. Needed for Leonardo onl\y
 }
  pinMode(SS, OUTPUT);
  if (!SD.begin(chipSelect)) {
 //Serial.println("Card failed, or not present");
 // don't do anything more:
 while (1) ;
 }
 dataFile = SD.open("datalog.txt", FILE_WRITE);


  pinMode(trigPin, OUTPUT); // Sets the trigPin as an OUTPUT
  pinMode(echoPin, INPUT); // Sets the echoPin as an INPUT
```

```
updownServo.attach(5);
RLServo.attach(6);

updownServo.write(100);
RLServo.write(90);

pinMode(12,OUTPUT);
pinMode(9,OUTPUT);
pinMode(7,OUTPUT);
pinMode(11,OUTPUT);


digitalWrite(12,LOW);
digitalWrite(9,LOW);
digitalWrite(7,LOW);
digitalWrite(11,LOW);
```

Now in the loop function, which contains the code to be repeated as long as the system is up, we read the serial data send by the RPI, the distance from the ultrasonic sensor, the decide using a "switch-case" which function to call next.

```
if (Serial.available() > 0) {
  incomingByte = Serial.read();
  check=1;
}
```

```
 digitalWrite(trigPin, LOW);
 delayMicroseconds(2);
 // Sets the trigPin HIGH (ACTIVE) for 10 microseconds
 digitalWrite(trigPin, HIGH);
 delayMicroseconds(10);
 digitalWrite(trigPin, LOW);
 // Reads the echoPin, returns the sound wave travel time in microseconds
 duration = pulseIn(echoPin, HIGH);
 // Calculating the distance
 distance = duration * 0.034 / 2; // Speed of sound wave divided by 2 (go and back)
 // Displays the distance on the Serial Monitor
 Serial.println(distance);

  switch((char)incomingByte){

    case '0' :
    if(distance>30)
    randomNavigate();
    else{
      RR();
    }
    break;

    case '1' :
    if(check==1)
    Stop2scan();
    check=0;
    break;

   case '2' :
   if(distance>30 && check==1){
   goStraight();
   Stop2scan();
   check=0;
   }
   break;
```

Several functions were written to accomplish specific tasks. These functions are shown in the next figure.

And the implementation of some of them is also shown.

```
void randomNavigate();
void goStraight();
void RRstraight();
void RLstraight();
void RR();
void RL();
void Stop2scan();
void rotateAbit();
```

```
void randomNavigate(){
  if(distance>30){
    digitalWrite(12,LOW);
    digitalWrite(9,HIGH);
    digitalWrite(7,LOW);
    digitalWrite(11,HIGH);
}
else{
  RR();
}

}
```

```
void goStraight(){
    digitalWrite(12,LOW);
    digitalWrite(9,HIGH);
    digitalWrite(7,LOW);
    digitalWrite(11,HIGH);
    delay(2000);
}
```

As shown below, the snippet of code where the humidity, light intensity, and temperature are read from the sensors then printed to a file created on the micro SD card connected to the Arduino via a micro SD card module

```
DHT.read11(dht_apin);
String s="Current humidity = ";
dataFile.print(s);
delay(500);
dataFile.print(DHT.humidity);
delay(500);
dataFile.print("%  ");
delay(500);
dataFile.print("temperature = ");
delay(500);
dataFile.print(DHT.temperature);
delay(500);
```

```
dataFile.println("C  ");
delay(500);
dataFile.print("light intensity: ");
delay(500);
I1=(I1+I2)/2;
dataFile.println(I1);
delay(500);
dataFile.flush();
 check =0;
```

iv. Server side:

After capturing images by the RPI, they are sent to the server, a personal computer in our case, through a ***SOCKET*** connection between them. The server receives the images and runs them through the model trained before. Below are snippets of the code written in the server side with brief explanation of each.

The program starts with building the model using KERAS library.

```
# load retinanet model
model = models.load_model('unifinalmodel.h5', backbone_name='resnet50')
```

```
print(model.summary())
```

Then we try to establish a connection with RPI using socket.

```
s=socket.socket()
host=input(str("please enter the host name: "))
port=8080
s.connect((host,port))
print("connected.")
```

After establishing a connection, we start the loop the waits for the images received in a zip file, unzip them, detect all the diseases, them saving them to a file in the server's storage. The below code receives the name of the zip file, its size, and the file itself.

```
    time.sleep(1)
    s.settimeout(None)
    file_name = s.recv(100).decode("utf-8")
    #file_size = int(s.recv(100))
    file_size = s.recv(100).decode("utf-8")
    print(file_name)
    print(file_size)
    s.settimeout(5)
    # Opening and reading file.
    with open(file_name, "wb") as file:
        c = 0
        # Starting the time capture.
        start_time = time.time()
        # Running the loop while file is recieved.
        while c <= int(file_size):
            time.sleep(0.15)
            print("ok")
            try:
                data = s.recv(8192)
                file.write(data)
            except Exception:
```

```
                pass
            print("mm")
            # if not data:
            #     break
            c += len(data)
        file.close()
        # Ending the time capture.
        end_time = time.time()

    print("File transfer Complete.Total time: ", end_time - start_time)
```

After receiving the zip file, it is unzipped using the ZIPFILE library.

```
dir_name = 'C:/Users/user/Desktop/for the project/socket transfer 2'
    extension = ".zip"

    #os.chdir(dir_name) # change directory from working dir to dir with files

    for item in os.listdir(dir_name): # loop through items in dir
        if item.endswith(extension): # check for ".zip" extension
            file_name = os.path.abspath(item) # get full path of files
            zip_ref = zipfile.ZipFile(file_name) # create zipfile object
            zip_ref.extractall(dir_name) # extract file to dir
            zip_ref.close() # close file
```

Now we have the image ready in the server, and all what we want to do is to rum them through the RETINA NET model to detect diseases. And this is achieved with the code in the below image.

```
for image in imgs:
        # copy to draw on
        draw = image.copy()
        draw = cv2.cvtColor(draw, cv2.COLOR_BGR2RGB)
        # preprocess image for network
        image = preprocess_image(image)
```

```
        image, scale = resize_image(image)


        # process image
        start = time.time()
        boxes, scores, labels = model.predict_on_batch(np.expand_dims(image,
axis=0))
        print("processing time: ", time.time() - start)
        print(scores)
        print(labels)
```

Finally, the results are stored in a file in the server's storage using the OS python library.

4.2 Training of RetinaNet model :

i. Dataset :

For the disease detection model we trained, a dataset consisting of plant leaves was collected from the campus of the Lebanese University. The Dataset consists of about 250 high resolution images used for training and about 40 images used for validation. The size of the dataset is surely too small, but one purpose of the project was to highlight the strength of the RetinaNet model, which is the state-of-art in the computer vision field nowadays. In addition, transforming processes were used on the dataset images which ensured a bigger dataset and better results. These images, unlike existing datasets found on the internet, had to labelled by hand using a "labelImg" tool. Each image had an average of 7 to 10 labels in it, as shown in the below images.

Figure 17:Labeling of healthy leaves

Figure 18: labeling of non-healthy leaves

RetinaNet model is made of a complex network that will be briefly explained soon. The implementation of it is already published on the KERAS official site, but even though it is published, using the official documentation needs great programming skills accompanied with clear understanding of the computer vision and deep learning fields. We managed to train a model on the custom dataset that we collected and labelled by the help of a GITHUB repository called FIZYR. It provides some python scripts that help users train at the RetinaNet model based on the official KERAS documentation.

The dataset could have been used in any format, but we used the Pascal VOC 2007 format for our data, which is widely used with different models and networks. The image labeling result was changed to XML format.

For sure, training deep learning model is time consuming and requires powerful resources and high computational power, therefore we used ***google-colab*** which provides the users with a free GPU memory to fasten the work and achieve good results. After training, we got the inferencing model, which was used on the server side, where images captured and sent by the RPI to the server were tested for diseases using this model.

On ***google-colab,*** with the help the FYZIR git repository, the network was built, and the result of each epoch was shown to supervise and stop at the desired results.

Below is a snippet of the result shown when training the model. We can see how the mean average precision is developing and increasing. We were able to achieve almost 0.7 mAP. This result will be discussed later.

```
Epoch 00008: saving model to /content/snapshot/resnet50_pascal_08.h5
/usr/local/lib/python3.7/dist-packages/keras/engine/functional.py:1410: CustomMaskWarning: Custom
  layer_config = serialize_layer_fn(layer)
178/178 [==============================] - 633s 4s/step - loss: 1.5751 - regression_loss: 1.2689 -
Epoch 9/50
Running network: 100% (68 of 68) |########| Elapsed Time: 0:01:02 Time:  0:01:02
Parsing annotations: 100% (68 of 68) |####| Elapsed Time: 0:00:00 Time:  0:00:00
317 instances of class healthy with average precision: 0.4526
320 instances of class nothealthy with average precision: 0.5460
mAP: 0.4993

Epoch 00009: saving model to /content/snapshot/resnet50_pascal_09.h5
/usr/local/lib/python3.7/dist-packages/keras/engine/functional.py:1410: CustomMaskWarning: Custom
  layer_config = serialize_layer_fn(layer)
178/178 [==============================] - 624s 4s/step - loss: 1.5634 - regression_loss: 1.2597 -
Epoch 10/50
Running network: 100% (68 of 68) |########| Elapsed Time: 0:01:03 Time:  0:01:03
Parsing annotations: 100% (68 of 68) |####| Elapsed Time: 0:00:00 Time:  0:00:00
317 instances of class healthy with average precision: 0.5694
320 instances of class nothealthy with average precision: 0.5839
mAP: 0.5767
```

5. Conclusion:

RetinaNet model is the state-of-art deep learning model with its unique architecture, mainly based on convolution layers. It is known for its loss function, Focal Loss, and the feature pyramid network, FPN. Our model was trained on a custom dataset of healthy and diseased collected from the university campus. Arduino and Raspberry are the main hardware used to build the robot, in addition to some sensors and actuators. Python was the backbone of almost everything, the model, raspberry pi, and server side. Its libraries simplified the job. The navigation plan was to navigate searching for plants by their QR code, take photos and send them to the server to be tested for any disease.

RESULTS AND DISCUSSION:

1. Results:

Since this robot is not only able to detect diseases, but also collect field data, and because of the obstacle avoidance and QR detection techniques, results and evaluation must consider every purpose this robot claims to achieve. Upon stating the results found after several testing of the robot in a field like environment, collected data, plant locating responsiveness, obstacle avoidance accuracy, and disease detection precision will be clearly presented to be discussed in the next part.

Starting with how fast the robot detected the presence of a QR code in a field, it turned out to be a very successful technique that we came up with for the simplicity of this project. It was able to locate the three plants located in the testing field with no problem in relatively short time. It took the robot several minutes to locate and test the plants. The below figure shows the QR codes detected from the robot's view.

Figure 20: The QR codes detected from the robot's view

Another important feature for such robot is not to damage crops while navigating in the field, so an ultrasonic was mounted in the form to detect any obstacle and avoid it. This method also turned out to be quite useful and helped protect the plants in the testing field. As this robot navigates, it collects data about the environment every now and then, and these data are stored on a micro SD card module in a text format.

```
Current humidity = 55.00%  temperature = 19.00C
light intensity: 805
Current humidity = 55.00%  temperature = 19.00C
light intensity: 730
Current humidity = 55.00%  temperature = 19.00C
light intensity: 730
Current humidity = 55.00%  temperature = 19.00C
light intensity: 729
Current humidity = 56.00%  temperature = 19.00C
light intensity: 730
Current humidity = 54.00%  temperature = 19.00C
light intensity: 739
Current humidity = 56.00%  temperature = 19.00C
light intensity: 805
Current humidity = 46.00%  temperature = 21.00C
light intensity: 836
```

Figure 21: The result of humidity, temperature, and light intensity

Later, farmers can take these data to study their environment well which is a great tool to increase the yield, because the more you are familiar with your environment, the better you can achieve. for this robot, the data collected via the Arduino sensors were ***humidity, temperature, and light intensity***. Light intensity was measured through two sensors and an average was taken. A sample of the results are shown in the next couple of pictures.

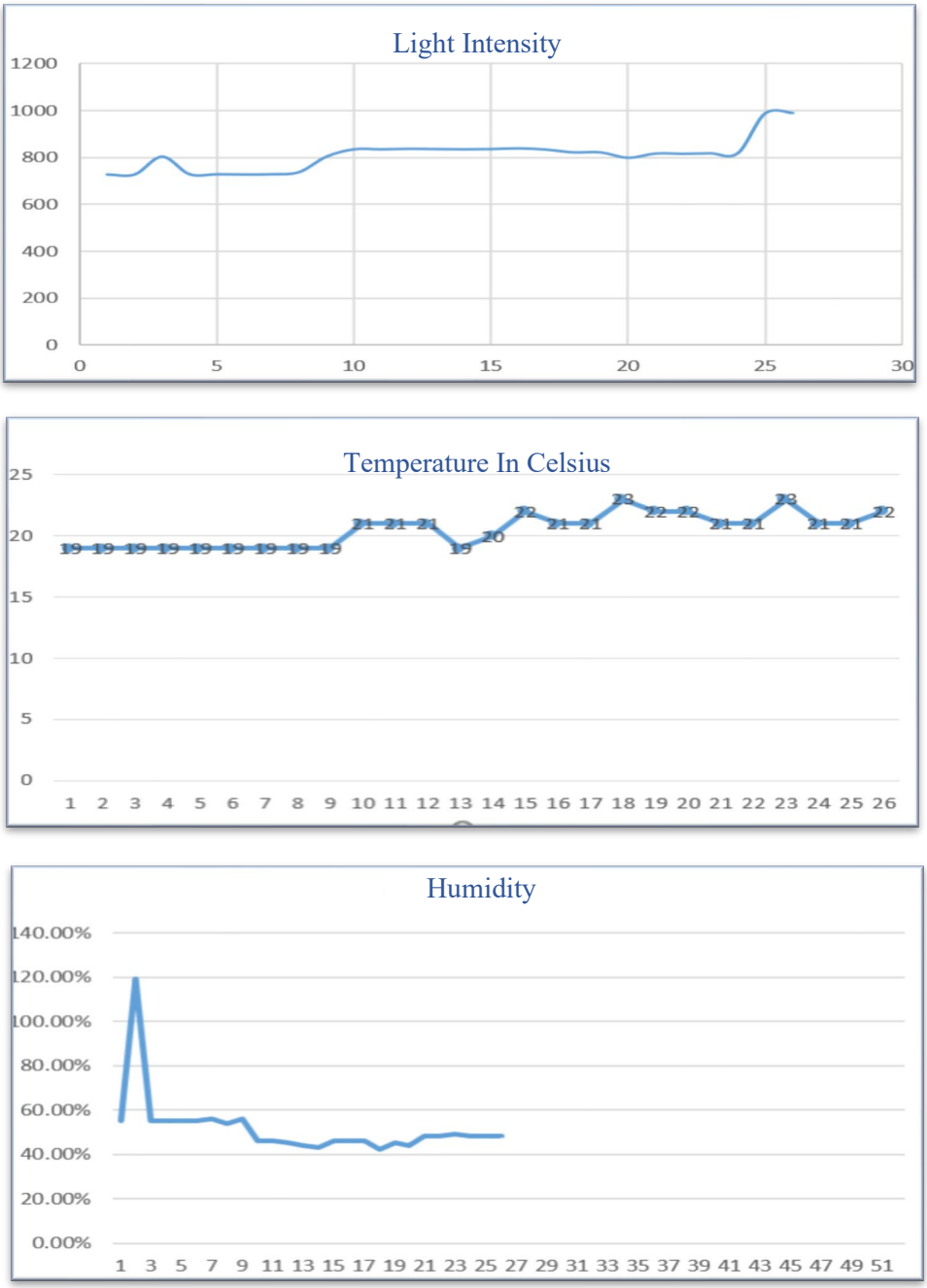

Figure 22: Graphs representing the result of light intensity, temperature, and humidity.

The last part of results left is the detected plant diseases from the pictures taken by the robot and sent to the server. The below picture are samples of the detected plant diseases from several trial in the testing fields.

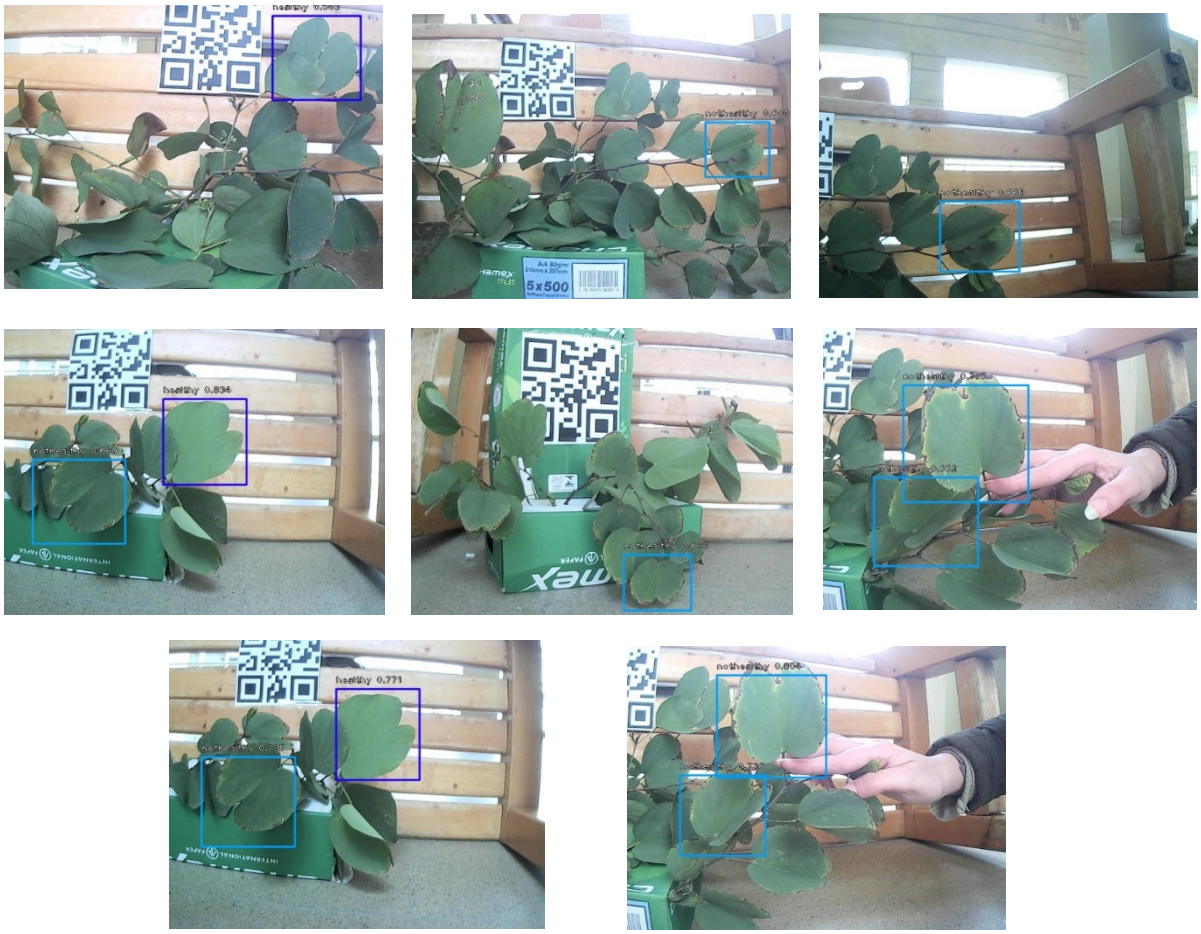

Figure 23: Samples of the detected plant diseases from several trial in the testing fields.

2. Discussion:

We will first discuss the results of data collection from the field, and the plan we made up for the autonomous navigation and plant locating. Then we will elaborate more, in details, about the precision of the model and what are the limitation that we found during testing period.

Starting with how the robot can navigate in a field to locate all the present plants, we came up with a plan where each plant has its own QR code with its information decoded in it. So, the QR code has the plant's name and number in order to be able to identify the plant later from its number. The plan is summed up in few words, navigate until you locate a plant by its QR code, but be careful not to damage a plant or hit any obstacle. The ultrasonic sensor is responsible for the

obstacle avoidance and it is good enough. The plan itself does not ensure that the robot could locate all the plants in a given day, due to the random navigation algorithm, but it surely showed a great hope in the isolated testing field where it located all the three plants present then. The way of navigation is not in its best state, but it surely is a good plan to avoid any later complexities when deploying different ways of navigations. One might think of GPS field navigation or even another DL model which helps the robot navigate smartly in the field to try not to miss any plant, but that is much of a complexity that needs time, effort and new resources. This might be the next step of such project.

Now regarding the data collected, it is more than enough to depend on the Arduino sensors since it is not that of a critical matter. The data's purpose is to give us an insight of how one variable is changing throughout a given period, like a year. This variable could be light intensity, humidity, or temperature.

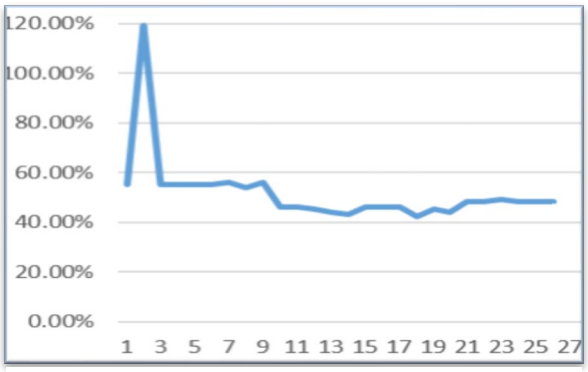

Figure 24: Graph representing the humidity in serval period of times.

The Arduino sensors could give some errors for some readings, but that is not that important given the fact that thousands of readings will be collected from the robot. As you might have noticed in one of the graphs shown before, more specifically the one related to humidity, there is one reading that give a humidity of 119%, which is not logical, and that is a random error. It was obvious before because only 26 reading were shown, which could have been collected in two days only. So, this outlier is not important when dealing with longer periods.

Another thing we need to discuss is the responsiveness of the QR detection. Experimentally, while the robot is navigating and taking pictures constantly, it will be shaking a little a bit due to the noise from motor and shape of the ground. This usually leads to non-clear pictures, but in our case that show any problem, unlike we expected to see. This is because that size of the QR codes is not very small, and the python library used, PYZBAR, is strong in detecting QR codes, which helped a lot.

To sum up, navigation, plant locating, and data collecting all turned out to be very good and gave good results that we can depend on.

The last part we need to discuss is the model we trained, and how it did when deployed and tested on the robot. We will compare the results of the diseases detected from the images taken by the robot, with images taken by better camera for similar plant, and this comparison is to highlight one limitation when dealing with DL models.

As we stated before in our objectives, testing the RETINA-NET model was one of our purposes. A dataset of about 400 images is surely a very small dataset, and no one would expect good results from models less efficient than RetinaNet. But surprisingly, this is not the case here. Even with a very small custom dataset, RetinaNet gave astonishing results when tested with high resolution results. Observe the following two sets of images:

1. *Set 1:*

Figure 25: Tested high resolution images.

2. *Set 2:*

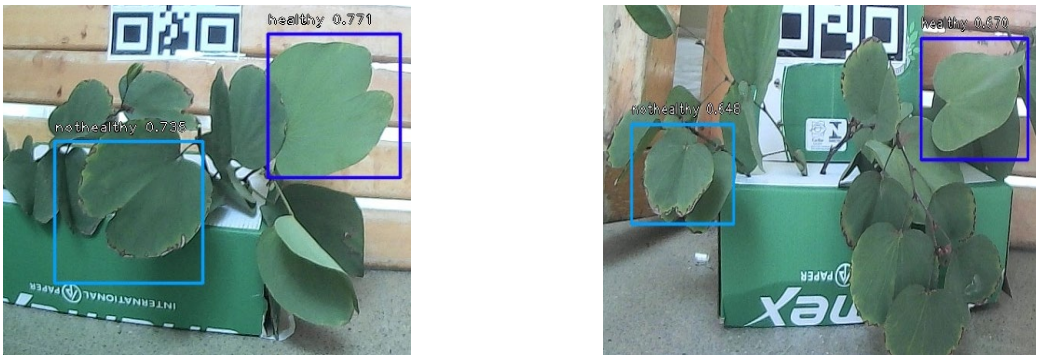

Figure 26: Tested low resolution images.

The first set is for images taken by a high-resolution camera, the same camera that was used when taking images to train the model. The second set is for images taken by the webcam mounted on the robot. We will analyze the results of both sets then state the two limitations found here. Just by looking at the first set of images, it is obligatory to appreciate the astonishing results of the RetinaNet model based on a relatively small dataset. In this set, many leaves were able to be detected and classified well. Now by looking at the second set of images, some leaves were able to be detected and classified, but not as good as images from set 1. The two limitation here are the small dataset and the difference in resolution of pictures between datasets.

The small dataset problem was not a bottleneck here, since RetinaNet is really that powerful, and that is obvious from the set1 images, but a bigger data set is surely the best to done.

The resolution difference between the training and testing images left a gap somewhere. This gap was obvious in the precision of the model when tested with live images from the robot. The model in our case was trained on high resolution images. And based on the fact that images pass through a CNN network, ResNet in our case, small details are actually lost. Even though RetinaNet uses FPN, feature pyramid network, the small details in the low-resolution images will still be lost and cannot be easily recognized by a model trained on high resolution images.

But still, this robot was successfully able to navigate, locate, and detect plant diseases as it was supposed to do. And I successfully showed the strength of the state-of-art DL model, Retina-Net.

GENERAL CONCLUSION:

We aimed in this project to deal with plant disease problems in a modern smart way and build a robot that helps take care of a planting field. An autonomous navigating robot was our solution. Obviously, Python was the most powerful tool we used, especially with its strong DL libraries. And such projects show the strength of the Deep Learning domain and its role in the future. Convolutional networks are the backbone of our model, RetinaNet, since it is highly efficient with image processing and detection. Our model was trained on a custom dataset of healthy and diseased collected from the university campus. Arduino and Raspberry are the main hardware used to build the robot, in addition to some sensors and actuators to measure temperature, humidity and light intensity.

Regarding the RetinaNet model, we are now more convinced about how excellent it is, even with very small dataset. Had it been for more time, we could have collected a richer dataset, but would not use any other than the RetinaNet model. The robot itself was successful and showed great potential for later development. It was able to locate every plant present in the field, take photos of it, send them to the server. And these images where tested for any disease where some of them turned out to be diseased. The full time taken was very good for the three plants. The data was successfully and accurately collected on a micro SD card mounted on the robot.

This project, like any other project, was once an idea, and it developed due to hard work and determination to become at its current state. Although it was able to achieve all its objectives, we still envision additional objectives and abilities to be added for it. The main thing we look forward to making better, is the trained model where we want to increase the accuracy by increasing the dataset and even adding more types of plants so that this robot can be deployed in any field. Another thing we are working on is a model that replaces the current navigation algorithm to a smarter one. The last thing is that to train a model that can study the collected data and suggest time periods to plant specific plants according to the history of the field. This project's life has not ended yet, but with what envision to develop in it, we can safely say it is just the beginning of the journey.

REFERENCES:

1) T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollar, "Focal Loss for Dense Object Detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 42, no. 2, Feb. 2020.
2) K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," 2015.
3) T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature Pyramid Networks for Object Detection," Apr. 2017.
4) Umberto Michelucci, Applied deep learning : a case-based approach to understanding deep neural networks. Berkeley, California: Apress, New York, Ny, 2018.
5) V Kůrková, Yannis Manolopoulos, B. Hammer, L. S. Iliadis, and I. G. Maglogiannis, Artificial neural networks and machine learning -- ICANN 2018 : 27th International Conference on Artificial Neural Networks, Rhodes, Greece, October 4-7, 2018, Proceedings. Part I. Cham, Switzerland: Springer, 2018.
6) P. Jay, "The intuition behind RetinaNet," *Medium*, Apr. 08, 2018: https://medium.com/@14prakash/the-intuition-behind-retinanet-eb636755607d.
7) Dshahid380, "Convolutional Neural Network," *Medium*, Feb. 24, 2019: https://towardsdatascience.com/covolutional-neural-network-cb0883dd6529.
8) J. Brown, "Retinanet-Tutorial," *GitHub*, Jan. 19, 2022: https://github.com/jaspereb/Retinanet-Tutorial.
9) Fizyr, "fizyr/keras-retinanet," *GitHub*, Sep. 09, 2019: https://github.com/fizyr/keras-retinanet.
10) Tzutalin, "tzutalin/labelImg," *GitHub*, Aug. 02, 2019: https://github.com/tzutalin/labelImg.
11) J. Ramos, "A Quick Summary of AI. Part 3: Deep Learning," Medium, Nov. 30, 2018: https://javier-ramos.medium.com/a-quick-summary-of-ai-part-3-deep-learning-c88157368f00

APPENDIX:

1. Arduino's code:

```
#include "Arduino.h"
#include <dht.h>
#include <Servo.h>
#include <SPI.h>
#include <SD.h>
const int chipSelect = 10;
File dataFile;
#define echoPin 2 // attach pin D2 Arduino to pin Echo of HC-SR04
#define trigPin 3 //attach pin D3 Arduino to pin Trig of HC-SR04
int I1,I2;
#define dht_apin A4 // Analog Pin sensor is connected to

dht DHT;

Servo updownServo, RLServo;

int check=0;
int incomingByte = 0;
long duration;
int distance;


void randomNavigate();
void goStraight();
void RRstraight();
void RLstraight();
void RR();
void RL();
void Stop2scan();
void rotateAbit();
void back();

#define echoPin 2 // attach pin D2 Arduino to pin Echo of HC-SR04
#define trigPin 3 //attach pin D3 Arduino to pin Trig of HC-SR04



void setup() {
  Serial.begin(9600);
   while (!Serial) {
 ; // wait for serial port to connect. Needed for Leonardo onl\y
 }
  pinMode(SS, OUTPUT);
  if (!SD.begin(chipSelect)) {
 //Serial.println("Card failed, or not present");
 //Serial.println("Card failed, or not present");
 // don't do anything more:
 while (1) ;
 }
 dataFile = SD.open("datalog.txt", FILE_WRITE);


  pinMode(trigPin, OUTPUT); // Sets the trigPin as an OUTPUT
  pinMode(echoPin, INPUT); // Sets the echoPin as an INPUT

  updownServo.attach(5);
  RLServo.attach(6);

  updownServo.write(100);
  RLServo.write(120);

  pinMode(12,OUTPUT);
  pinMode(9,OUTPUT);
  pinMode(7,OUTPUT);
  pinMode(11,OUTPUT);


  digitalWrite(12,LOW);
  digitalWrite(9,LOW);
  digitalWrite(7,LOW);
  digitalWrite(11,LOW);

  while(Serial. read() >= 0) ;


}

void loop() {


  if (Serial.available() > 0) {
    incomingByte = Serial.read();
    check=1;
  //}

  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  // Sets the trigPin HIGH (ACTIVE) for 10 microseconds
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
```

```
  digitalWrite(trigPin, LOW);
  // Reads the echoPin, returns the sound wave travel time in microseconds
  duration = pulseIn(echoPin, HIGH);
  // Calculating the distance
  distance = duration * 0.034 / 2; // Speed of sound wave divided by 2 (go and back)
  // Displays the distance on the Serial Monitor
  //Serial.println(distance);

  switch((char)incomingByte){

    case '0' :
    if(distance>=50){
    randomNavigate();}
    else{
      back();
      RR();    // new function
    }
    break;

    case '1' :
    if(check==1 ){
    Stop2scan();
    check=0;}
    break;

//    case '2' :
//    if(distance>60 && check==1){
//    goStraight();
//    Stop2scan();
//    check=0;
//    }
//    break;

    case '2' :
    if(check==1 ){
    Stop2scan();
    check=0;}
    break;

    case '3' :
    if( check==1){
    RL();
    //goStraight();
    Stop2scan();
    check=0;
    }
    break;

    case '4' :
    if( check==1){
    RL();
    Stop2scan();
    check=0;
    }
    break;

    case '5' :
    if( check==1){
    RR();
   // goStraight();
    Stop2scan();
    check=0;
    }
    break;

    case '6' :
    if( check==1){
    RR();
    Stop2scan();
    check=0;
    }
    break;

  }
  }
}
////////////////////////////////////
////////////////////////////////////
////////////////////////////////////

void randomNavigate(){

    digitalWrite(12,LOW);
    digitalWrite(9,HIGH);
    digitalWrite(7,LOW);
    digitalWrite(11,HIGH);

}
//void goStraight(){
//    digitalWrite(12,LOW);
```

```
void RR(){
    digitalWrite(12,LOW);
    digitalWrite(9,LOW);
    digitalWrite(7,LOW);
    digitalWrite(11,LOW);
    delay(100);
    digitalWrite(12,LOW);
    digitalWrite(9,LOW);
    digitalWrite(7,HIGH);
    digitalWrite(11,HIGH);
    delay(1000);
    digitalWrite(12,LOW);
    digitalWrite(9,LOW);
    digitalWrite(7,LOW);
    digitalWrite(11,LOW);



}

void RL(){
    digitalWrite(12,LOW);
    digitalWrite(9,LOW);
    digitalWrite(7,LOW);
    digitalWrite(11,LOW);
    delay(100);
    digitalWrite(12,HIGH);
    digitalWrite(9,HIGH);
    digitalWrite(7,LOW);
    digitalWrite(11,LOW);
    delay(1000);
    digitalWrite(12,LOW);
    digitalWrite(9,LOW);
    digitalWrite(7,LOW);
    digitalWrite(11,LOW);



}

void Stop2scan(){
    //Serial.flush();

    digitalWrite(12,LOW);
    digitalWrite(9,LOW);
    digitalWrite(7,LOW);
```

```
    digitalWrite(7,LOW);
    digitalWrite(11,LOW);
    delay(1000);
    updownServo.write(60);
    delay(1000);
    updownServo.write(100);
    delay(1000);
    updownServo.write(140);
    delay(1000);
    updownServo.write(100);
    delay(1000);



    I1=analogRead(A0);
    I2=analogRead(A1);

    DHT.readl1(dht_apin);
    String s="Current humidity = ";
    dataFile.print(s);
    delay(500);
    dataFile.print(DHT.humidity);
    delay(500);
    dataFile.print("%  ");
    delay(500);
    dataFile.print("temperature = ");
    delay(500);
    dataFile.print(DHT.temperature);
    delay(500);
    dataFile.println("C  ");
    delay(500);
    dataFile.print("light intensity: ");
    delay(500);
    I1=(I1+I2)/2;
    dataFile.println(I1);
    delay(500);
    dataFile.flush();
     check =0;


}
void back(){
    digitalWrite(12,LOW);
    digitalWrite(9,LOW);
    digitalWrite(7,LOW);
```

```
void back(){
    digitalWrite(12,LOW);
    digitalWrite(9,LOW);
    digitalWrite(7,LOW);
    digitalWrite(11,LOW);
    delay(100);
    digitalWrite(12,HIGH);
    digitalWrite(9,LOW);
    digitalWrite(7,HIGH);
    digitalWrite(11,LOW);
    delay(1000);
    digitalWrite(12,LOW);
    digitalWrite(9,LOW);
    digitalWrite(7,LOW);
    digitalWrite(11,LOW);



}
```

2. RaspberryPi's code:

```
from cmath import polar
from math import fabs
import cv2 as cv
import time
import numpy as np
from pyzbar.pyzbar import decode
from pyzbar.pyzbar import ZBarSymbol
import serial
from serial import Serial
import socket
import os
import sys
import zipfile

zip_name='main.zip'
k=20
#---------------initializing Serial-------------
if __name__ == '__main__':
    ser = serial.Serial('/dev/ttyACM0', 9600, timeout=1)
    ser.reset_input_buffer()


#---------------socket connection----------

s=socket.socket()
host=''
port=8080
s.bind((host,port))
s.listen(1)
print(host)
print("waiting for host to connect.....")
conn , addr =s.accept()
print(addr, " has connected")
ser.write(b"12") #turn on lamp using arduino green color


#-------------- variable--------------------
# "0" random search
# "1" approach the target
checked=[]
finished=True



#--------------initializing camera-----------------
def capture1():

    global cap

    cap = cv.VideoCapture(0)




    if not cap.isOpened():
        raise IOError("Cannot open webcam")

    cap.set(cv.CAP_PROP_FRAME_WIDTH,640)
    cap.set(cv.CAP_PROP_FRAME_HEIGHT,480)

capture1()
```

```
#-------------Functions-------------------------
def PolyArea(x,y):
    return 0.5*np.abs(np.dot(x,np.roll(y,1))-np.dot(y,np.roll(x,1)))


def center(x1,x2):

    if(  (((x1+x2)/2 -320 ) < 100) & (((x1+x2)/2 -320 ) > -100 )):
        return 0

    if(  (((x1+x2)/2 -320 ) > 100) ):
        return 1

    if( (((x1+x2)/2 -320 ) < -100 ) ):
        return 2



def check(plantnumber):
    for i in checked:
        if i==plantnumber:
            return True
    return False

def capAndSend():


    p=1
    while p !=21:
        time.sleep(2)

        for t in range(30):
            success, img = cap.read()

        filename=str(p)+".jpg"
        success, img = cap.read()
        cv.imwrite(filename, img)
        p=p+1


    with zipfile.ZipFile(zip_name, 'w') as file:
        for j in range(1, (k+1)):
            file.write('{}.jpg'.format(j))
            print('[+] {}.jpg is sent'.format(j))

    #file_name = input("File Name:")
    file_name= 'main.zip'
    file_size = os.path.getsize(file_name)
    file_size=str(file_size)
    print(file_size)

    conn.send(file_name.encode("utf-8"))
    time.sleep(1)
    conn.send(file_size.encode("utf-8"))

    with open(file_name, "rb") as file:
        c = 0
    # Starting the time capture.
        start_time = time.time()

    # Running loop while c != file_size.
```

```
    # Running loop while c != file_size.
        while c <= int(file_size):
            data = file.read(8192)
            time.sleep(0.1)


            if not (data):
                break
            conn.send(data)
            c += len(data)
    #s.close()
    finished=True

# -------------start detecting barcode------------

ser.write(b"0")
print("once")


while True:
    i=0
    for i in range(12):
        success, img = cap.read()
    success, img = cap.read()

    if finished:

        lapse=time.time()
        ser.write(b"0")
        print("sending :")
        print("0")


    for barcode in decode(img, symbols=[ZBarSymbol.QRCODE]):
        mydata=barcode.data.decode('utf-8')
        #print(barcode)




        #preparing points arrays and drawing polygon
        pts= np.array([barcode.polygon],np.int32)
        pts=pts.reshape((-1,1,2))
        cv.polylines(img,[pts],True,(255,0,255),5)
        x=[]
        y=[]
        for i in barcode.polygon:
            x.append(i[0])
        for i in barcode.polygon:
            y.append(i[1])
        x=np.array(x)
        y=np.array(y)
        print(PolyArea(x,y))
        print(x)
        print(y)

        #send serial

        if not check(mydata):
            print("fetet")
```

```
    lapse=time.time()

    if center(x[0],x[2])==0 and PolyArea(x,y)> 30000 :

        #ready to scan
        ser.write(b"1")
        print("sending: ")
        print("1")
        #call capAndScan() function
        checked.append(mydata)
        capAndSend()



    elif  center(x[0],x[2])==0 and PolyArea(x,y)< 30000 :
        #go straight
        ser.write(b"2")
        print("sending: ")
        print("2")
        time.sleep(2)
        checked.append(mydata)
        capAndSend()


    elif ( center(x[0],x[2])==1) and PolyArea(x,y)< 30000 :
        #rotate left then go straight
        ser.write(b"3")
        print("sending: ")
        print("3")
        time.sleep(2)
        checked.append(mydata)
        capAndSend()

    elif ( center(x[0],x[2])==1) and PolyArea(x,y)> 30000 :
        #only rotate left
        ser.write(b"4")
        print("sending: ")
        print("4")
        time.sleep(2)
        checked.append(mydata)
        capAndSend()

    elif ( center(x[0],x[2])==2) and PolyArea(x,y)< 30000 :
        #rotate right then go straight
        ser.write(b"5")
        print("sending: ")
        print("5")
        time.sleep(2)
        checked.append(mydata)
        capAndSend()


    elif ( center(x[0],x[2])==2) and PolyArea(x,y)> 30000 :
        #only rotate right
        ser.write(b"6")
        print("sending: ")
        print("6")
        time.sleep(2)
        checked.append(mydata)
        capAndSend()
```

```
        elif ( center(x[0],x[2])==2) and PolyArea(x,y)> 30000 :
            #only rotate right
            ser.write(b"6")
            print("sending: ")
            print("6")
            time.sleep(2)
            checked.append(mydata)
            capAndSend()


        ser_bytes = ser.readline()
        decoded_bytes = str(ser_bytes[0:len(ser_bytes)].decode("utf-8"))
        print(decoded_bytes)



#period=time.time()-lapse
#print(period)
#if(period>3):
  #  finished=True




#cv.imshow("h",img)

#c=cv.waitKey(1)
#if(c==27):
   # break
```

3. Server's code:

```
import socket
import time
# import keras_retinanet
from keras_retinanet import models
from keras_retinanet.utils.image import read_image_bgr, preprocess_image, resize_image
from keras_retinanet.utils.visualization import draw_box, draw_caption
from keras_retinanet.utils.colors import label_color
from keras_retinanet.utils.gpu import setup_gpu
import matplotlib.pyplot as plt
import cv2
import os
import numpy as np
import time
import glob
import tensorflow as tf
import zipfile

# load retinanet model
model = models.load_model('unifinalmodel.h5', backbone_name='resnet50')
print(model.summary())


s=socket.socket()
host=input(str("please enter the host name: "))
port=8080
s.connect((host,port))
print("connected.")

while True:

    time.sleep(1)
    s.settimeout(None)
    file_name = s.recv(100).decode("utf-8")
    #file_size = int(s.recv(100))

    file_size = s.recv(100).decode("utf-8")
    print(file_name)
    print(file_size)
    s.settimeout(5)

    # Opening and reading file.
    with open(file_name, "wb") as file:
        c = 0
        # Starting the time capture.
        start_time = time.time()

        # Running the loop while file is recieved.
        while c <= int(file_size):
            time.sleep(0.15)
            print("ok")
            try:
                data = s.recv(8192)
                file.write(data)
            except Exception:
                pass
            print("mm")


            # if not data:
            #     break

            c += len(data)

        file.close()
        # Ending the time capture.
        end_time = time.time()

    print("File transfer Complete.Total time: ", end_time - start_time)

    ##############################################################################
```

```
    ##############################################################################

    dir_name = 'C:/Users/user/Desktop/for the project/socket transfer 2'
    extension = ".zip"

    #os.chdir(dir_name) # change directory from working dir to dir with files

    for item in os.listdir(dir_name): # loop through items in dir
        if item.endswith(extension): # check for ".zip" extension
            file_name = os.path.abspath(item) # get full path of files
            zip_ref = zipfile.ZipFile(file_name) # create zipfile object
            zip_ref.extractall(dir_name) # extract file to dir
            zip_ref.close() # close file
            #os.remove(file_name) # delete zipped file
    ##############################################################################



    # load label to names mapping for visualization purposes
    labels_to_names = {0: 'healthy', 1: 'nothealthy'}


    # load image
    imgs=[]

    imgs = [cv2.imread(file) for file in glob.glob("*.jpg")]

    # image = read_image_bgr('a4.jpg')
    # imgs.append(image)
    # image = read_image_bgr('a3.jpg')
    # imgs.append(image)

    i=0

    for image in imgs:


        # copy to draw on
        draw = image.copy()
        draw = cv2.cvtColor(draw, cv2.COLOR_BGR2RGB)



        # preprocess image for network
        image = preprocess_image(image)
        image, scale = resize_image(image)

        # process image
        start = time.time()
        boxes, scores, labels = model.predict_on_batch(np.expand_dims(image, axis=0))
        print("processing time: ", time.time() - start)

        print(scores)
        print(labels)


        # correct for image scale
        boxes /= scale
        healthy_count=0
        nonhealthy_count=0


        # visualize detections
        for box, score, label in zip(boxes[0], scores[0], labels[0]):
            # scores are sorted so we can break
            if score < 0.6:
                break
```

```
                break

            color = label_color(label)

            b = box.astype(int)
            draw_box(draw, b, color=color)

            caption = "{} {:.3f}".format(labels_to_names[label], score)
            draw_caption(draw, b, caption)



        cv2.imwrite(os.path.join('C:/Users/user/Desktop/for the project/socket transfer 2/detected','Frame'+str(i)+'.jpg'),cv2.cvtColor(draw,cv2.COLOR_RGB2BGR))
        i+=1


        # plt.figure(figsize=(15, 15))
        # plt.axis('off')
        # plt.imshow(draw)
        # plt.show()

        time.sleep(0.1)

        ##########################delete images###################

        # dir_name = 'C:/Users/user/Desktop/for the project/socket transfer 2'
        # extension = ".jpg"

        # #os.chdir(dir_name) # change directory from working dir to dir with files

        # for item in os.listdir(dir_name): # loop through items in dir
        #     if item.endswith(extension): # check for ".zip" extension
        #         file_jpg = os.path.abspath(item) # get full path of files
        #         os.remove(file_jpg) # delete zipped file
```