# CHAPTER ONE: INTRODUCTION

* 1. **BACKGROUND**

Technology is surpassing human’s limitations and intelligence in every dimension exponentially and the Artificial Intelligence has been its core revolutionary scaffolding. This paradigm is being used to diagnosis the plant diseases more easily and at early stage. Artificial Intelligence has also been used in Agriculture field in the recent days to predict crops proneness to disease, in image interpretation and disease identification and generating reminders and alerts.

With the development of computational systems in recent years, and in particular Graphical Processing Units embedded processors, Machine Learning-related Artificial Intelligence applications have achieved exponential growth, leading to the development of novel methodologies and models, which now form a new category, that of Deep Learning. The basic deep learning tool used in our work is Convolutional Neural Networks (CNNs). CNNs constitute one of the most powerful techniques for modeling complex processes and performing pattern recognition in applications with a large amount of data, like the one of pattern recognition in images. Here we presented a CNNs system for the automated recognition of plants disease, based on leaves images. In this work, specific CNN architecture was trained and assessed, to form an automated plant disease detection and diagnosis system, based on simple images of leaves of healthy and diseased plant.

Despite having seen many improvements in the mass production and accessibility of food, food security remains threatened by a variety of factors such as the decline of pollinators and plant diseases.  In the developing world, more than 80 percent of the agricultural production is generated by smallholder farmers, and reports of yield loss of more than 30% due to pests and diseases are common. Fortunately, diseases can be managed by identifying the diseases as soon as it appears on the plant. The presence of disease on the plant is mainly reflected by symptoms on leaves. So, there is a need of an automatic, accurate and inexpensive solution for the detection of the disease from the images and to suggest possible remedies to the farmers.

* 1. **PROBLEM STATEMENT**

Nepali market lacks skilled manpower in the agriculture sector and those who available are mainly concentrated in the city areas. For the farmer it isn’t feasible to monitor crops all the time due to this reason farmer notice the diseases in their crops when the diseases had affected the crops significantly reducing the production of the crop yield and affecting entire economic status.

* 1. **OBJECTIVES**

The basic objective of this project is to design a system that takes an image of the infected plant leaf and predicts that image being diseased or not along with its control measures if diseased.

The core objectives that are fundamentals to the project are:

* To detect diseases in the farm as early as possible.
* To inform farmers about the nature of the diseases and provide possible remedies.

**1.4 APPLICATIONS**

The proposed system periodically monitors the cultivated field so that, crop diseases can be detected in early stage, then informs farmers about the nature of the diseases, and its possible remedy. It is applicable in countries like Nepal, India or West Africa, where majority of farmers are unaware and practice subsistence farming; geographical complexities triggers plenty of inaccessibility and scientific know-how cum expertise in the field of sustainable planning, cultivation, crop infection and disease remedies. This project resolves this typical problem of traditional agriculture and stereotypical cultivation of agricultural countries like Nepal in an efficient manner.

* 1. **PROJECT FEATURES**

The main feature of our project is to make the system which can monitor the cultivated field continuously for checking the plants/crops conditions. Some major features are enlisted below:

1. Disease Detection Unit: captures the image of infected plant leaves and match features with trained model.
2. Robotic Car Unit: control the movement of the cart along with different sensors like humidity and soil moisture sensor for the reading of the soil
3. Web application: To integrate machine learning model, Robotic Car together.

**CHAPTER TWO: LITERATURE REVIEW**

Developing complete solution for the automatic detection of diseased plant is a complex task itself. Selection of good network architecture, high quality hardware, nature of crops, and nature of greenhouses is some of the key fact that are going to be challenging to handle. Therefore, lots of previous work related to autonomous vehicle, disease detections were studied.

Recent advances in hardware technology have allowed the evolution of Deep Convolutional Neural Networks and their large number of applications, including complex tasks such as object recognition and image classification. The facilities of Deep Learning have allowed researchers to design systems that can be trained and tested end-to-end (all included in the same process), unlike when using handcrafted-based methods that use separate processes. Due to the outstanding performance of Convolutional Neural Networks (CNNs) as a feature extractor in image recognition tasks, the idea has been extended to different areas, such as in agriculture, automation, and robotics. Some of the applications for agriculture utilize Computer Vision and CNNs to solve complex tasks, such as plant recognition.

Subsequently, due to the recent advance in Machine Learning, the principle of CNN has been applied to plant diseases recognition in different crops, such as:

Brahimi, Mohammed & Arsenovic and his team [1] have tested multiple state-of-the-art Convolutional Neural Network (CNN) architectures using three learning strategies on a public dataset for plant diseases classification. These new architectures outperform the state-of-the-art results of plant diseases classification with an accuracy reaching 99.76%. Furthermore, they have proposed the use of saliency maps as a visualization method to understand and interpret the CNN classification mechanism. This visualization method increases the transparency of deep learning models and gives more insight into the symptoms of plant diseases.

In [2], Lu et al. proposed a novel identification approach for rice diseases based on deep convolutional neural networks. Using a dataset of 500 natural images of diseased and healthy rice leaves and stems, CNNs were trained to identify 10 common rice diseases. The experimental results showed that the proposed model achieved an average accuracy of 95.48%.

Few researchers proposed the use of CNN for leaf recognition and plant disease classification. [3]Atabay (Atabay2016b) designed a convolutional neural network architecture to identify plants based on leaf images. The proposed architecture consists five layers. After each convolutional layer a Rectified Linear Unit (ReLU) or Exponential Linear Unit (ELU) activation function is used and for each pooling layer, Max Pooling approach is applied. The proposed system [5] is applied on Flavia (Wu et al. 2007) and Swedish (Soderkvist 2001) leaf datasets containing 32 plant species with 1907 samples and 15 species with 1125 samples respectively. The images in the dataset are pictures of a single leaf taken at uniform background. All the input images are160x160 pixel grayscale images. The model achieved a classification accuracy of 97.24% and 99.11% accuracy for each dataset. The results showed that the proposed architecture for CNN-based leaf classification is closely competing with the latest extensive approaches on devising leaf features and classifiers.

Angie K. Reyes et al. (Reyes, Caicedo, and Camargo2015) [4], used a deep learning approach in which the complete system was learned without hand-engineered components. The designed system has 5 Conv layers followed by 2 fully connected layers. The CNN is trained using 1.8 million images from ILSVRC 2012 dataset

[14] and used a fine-tuning strategy to transfer learned recognition capabilities from general domains to the specific challenge of Plant Identification task. The dataset is combination of images of a plant or part of a plant taken both under a controlled environment as well as in the natural environment. They obtained an average precision of 0.486.

Sharada P. Mohanty et al (Mohanty, Hughes, and Salath´e 2016) [16], used the existing deep CNN architectures ,i.e AlexNet (Krizhevsky, Sutskever , and Hinton 2012) and GoogLeNet (Szegedy et al. 2015) to classify plant diseases. Using a public dataset of 54,306 images of diseased and healthy plant leaves collected under controlled conditions, the CNN was trained to identify 14 crop species and 26 diseases (or absence thereof). The models achieved 99.35% accuracy. When tested on a set of images taken at a different environment than the images used for the training, however, the model’s accuracy dropped to 31.4%.

# CHAPTER THREE: HARDWARE

The goal of our project is to detect the diseases plant as early as possible and to notify them about the disease with possible solution. The overall project can be divided into three parts mainly:

**3.1 AgroBot**

AgroBot is a Bluetooth controlled four-wheel drive robocar. It consists of Arduino UNO as a Microcontroller device drives the AgroBot using L298N motor driver and four BO Shaft DC motors. Agrobot is capable of moving in both forward and reverse direction. Its movement is controlled via Android based application using a HC-05 BT Module. AgroBot also consist of a Servo Motor controlled arm which is used for the Pi camera movement in up and down direction.

Alongside Arduino, Agrobot also contains a single board computer i.e. Raspberry Pi 3 Model B+ and a Pi Camera Module [7] which is used to capture still images of leaves for the disease detection. The Camera can also be used a live video feed to control the AgroBot movements precisely.

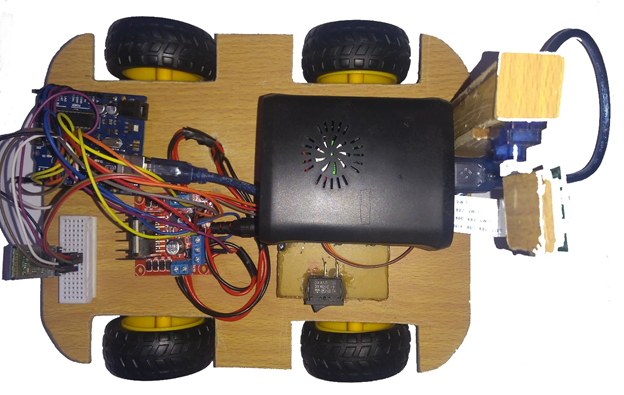
Raspberry Pi uses Motion Eye OS [8] as its Operating System. The purpose of this Pi is to capture high definition still images of plant leaves and directly upload them to cloud storage i.e. Google Drive. The uploaded images are later used to determine the diseases.

Figure 3.1: AgroBot

**3.1.1 Hardware Used**

Our Agrobot consists of following hardware parts:

1. Raspberry Pi 3 Model B+:

Raspberry Pi is a single board computer with wireless WAN and Bluetooth connectivity. The Raspberry Pi 3 Model B+ is the earliest model of the third generation Raspberry Pi. It contains Quad Core 1.2GHz Broadcom BCM2837 64bit CPU, 1GB RAN, 40-pin extended GPIO, CSI camera and DSI display ports.

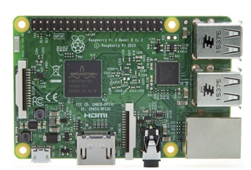
In Agrobot, the Pi captures the images of the leaves via Pi camera and sends it to cloud storage i.e. Google Drive.

Figure 3.2: Raspberry Pi 3

1. Pi Camera Module:

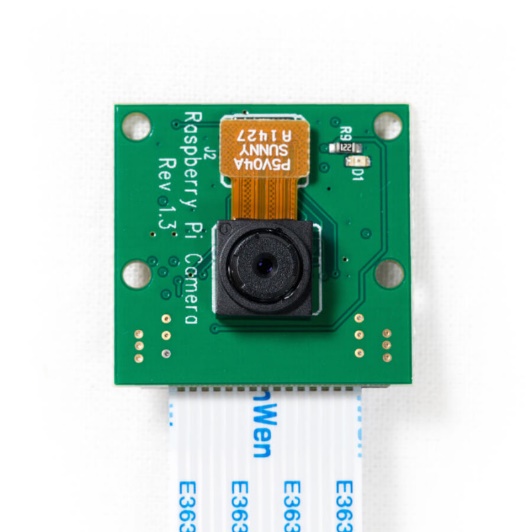
The Raspberry Pi Camera module has a Sony IMX219 8-megapixel sensor. This Camera Module can be used to take high-definition video, as well as stills photographs.

Figure 3.3: Pi Camera Module

In Agrobot, the Pi Camera is used to capture the still images of the leaves.

1. Arduino UNO:

Arduino Uno is a microcontroller board based on the ATmega328P. It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz quartz crystal, a USB connection, a power jack, an ICSP header and a reset button.



Figure 3.4: Arduino UNO

In Agrobot, Arduino acts a controller to control the Agrobot and its accessories.

1. Servo Motor:

A servo motor is a rotary actuator or motor that allows for a precise control in terms of angular position, acceleration and velocity.

Figure 3.5: Servo Motor

In Agrobot, Servo functions as a rotating arm for the Pi Camera Module.

1. L298N Driver:

The L298N is a dual H-Bridge motor driver which allows speed and direction control of two DC motors at the same time. The module can drive DC motors that have voltages between 5 and 35V, with a peak current up to 2A.

In Agrobot, L29N driver controls the movement of i.e. Forward and Reverse movement.

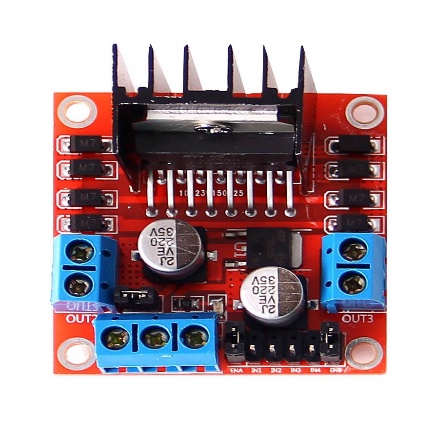


Figure 3.6: L298N Driver

1. BO Dual Shaft DC Motors:

In Agrobot, these DC motors are used for the moving the Agrobot. 4 DC motors are used to move the Agrobot which are controlled by the L298N driver.

Figure 3.7: DC Motor

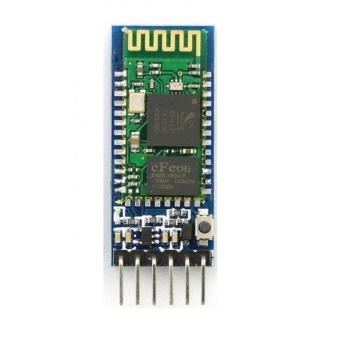
1. HC-05 Bluetooth Module:

Figure 3.8: BT Module

HC-05 module is an easy to use Bluetooth SPP (Serial Port Protocol) module, designed for transparent wireless serial connection setup. It is fully qualified Bluetooth V2.0+EDR (Enhanced Data Rate) 3Mbps Modulation with complete 2.4GHz radio transceiver and baseband.

In Agrobot, this BT Module is used to pair the Arduino and Android app.

**CHAPTER FOUR: SOFTWARE**

**4.1 Software Used**

Our Agrobot consists of following software parts:

1. Arduino IDE:

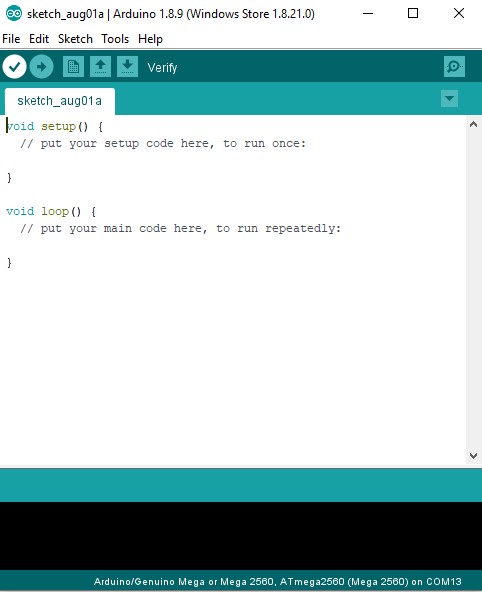
The Arduino integrated development environment (IDE) is a cross-platform application (for Windows, macOS, Linux) that is written in the programming language Java. It is used to write and upload programs to Arduino compatible boards, but also, with the help of 3rd party cores, other vendor development boards.

Figure 4.1: Arduino IDE

The Arduino IDE employs the program avrdude to convert the executable code into a text file in hexadecimal encoding that is loaded into the Arduino board by a loader program in the board's firmware.

1. Motion Eye OS:

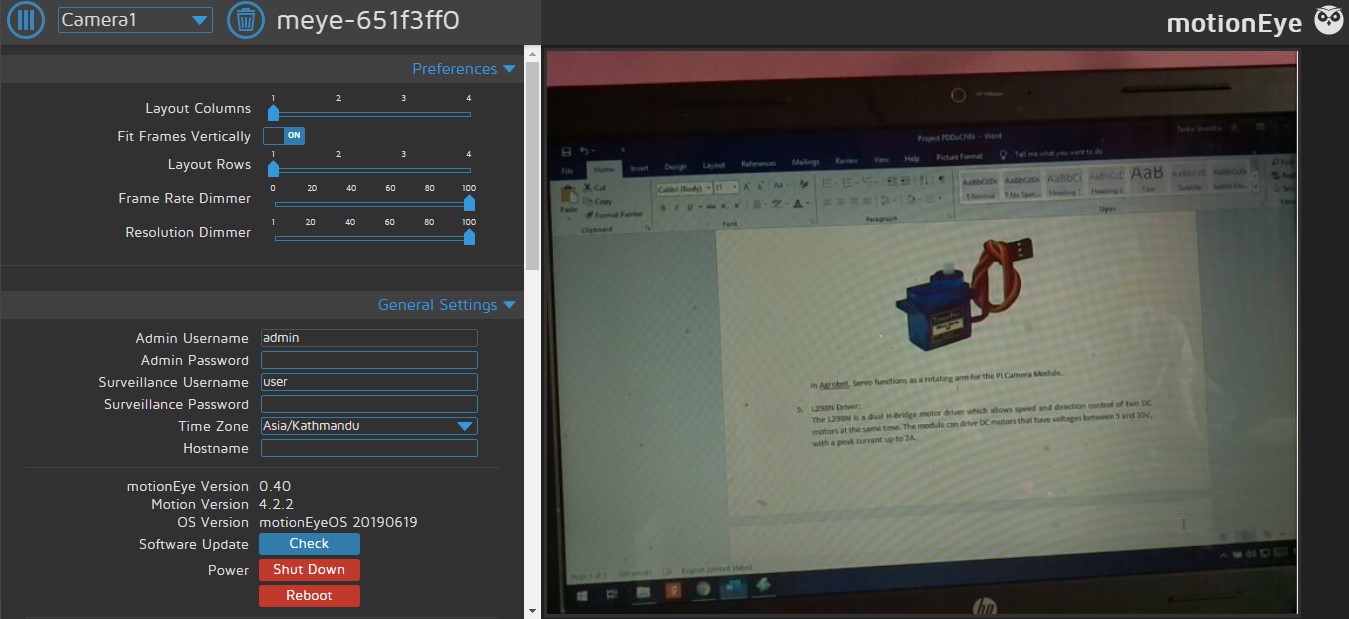
Motion Eye OS is a Linux distribution that turns a single-board computer into a video surveillance system. The OS is based on BuildRoot and uses motion as a backend and motion Eye for the frontend.

Figure 4.2: Motion Eye OS

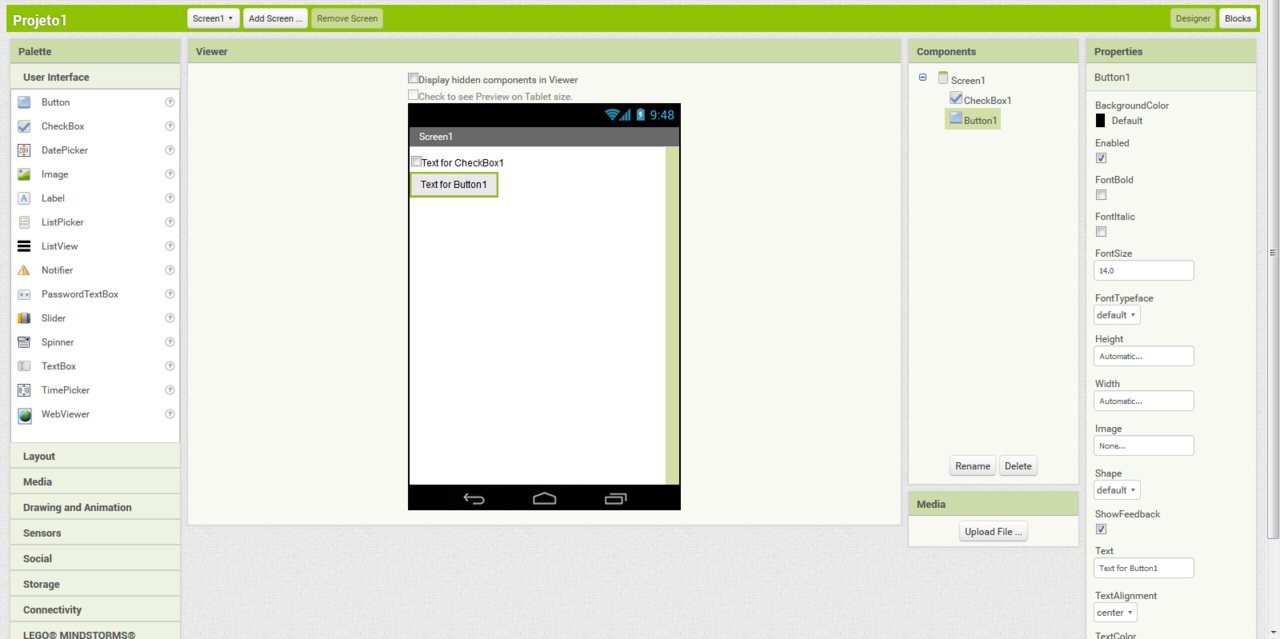
1. MIT App Inventor:

Figure 4.3: MIT App Inventor

MIT App Inventor is a web application integrated development environment originally provided by Google, and now maintained by the Massachusetts Institute of Technology (MIT). It allows users to computer programming to create application software (apps) for two operating systems (OS): Android, and iOS. It uses a graphical user interface (GUI) very similar to the programming languages Scratch and the StarLogo TNG user interface, which allows users to drag and drop visual objects to create an application that can run on mobile.

* 1. **Android Application**

Our Project also consists of an Android App used for controlling the Agrobot. We used a popular platform called MIT App Inventor to build our App. We are using Bluetooth communication as a means for pairing Arduino and our app and for this we are using HC-05 Bluetooth Module.

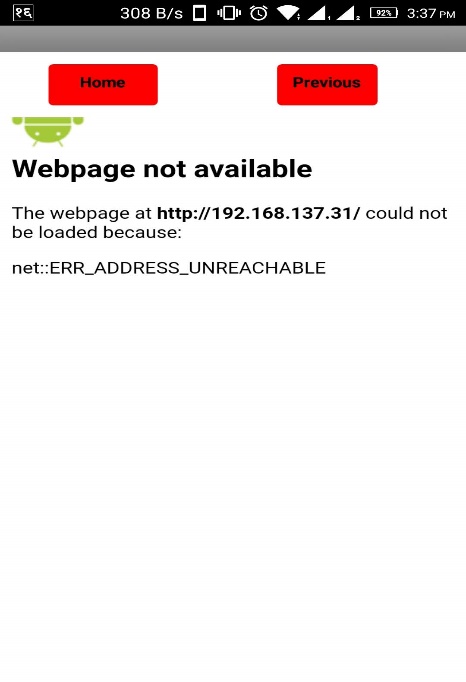
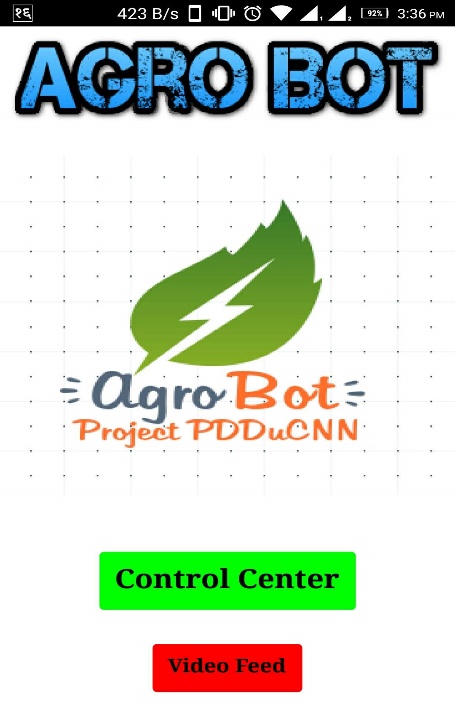
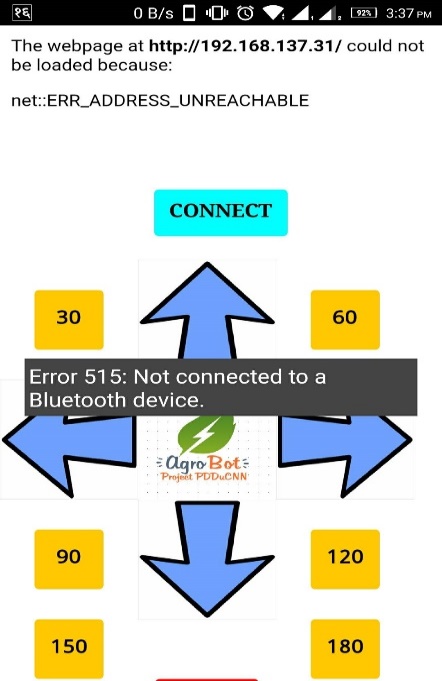
* The Android app is used as a remote controller to move the Agrobot in Forward and Reverse direction.
* The app controls the Servo arm movement in up and down direction.
* The app can also capture the still images of leaves from Pi Camera Module and save it to Google Drive.
* The app also has live video streaming feature for the movement of Agrobot.

Figure 4.4: Android App

**4.3 Web Application**

Our Entire project is connected by the React Web Application which communicates through Django Rest API to load deep learning model and predict the disease of the image. The detected disease is query through the MongoDB collection with the help of Node.js Server which also communicates to the front-end web application via REST API and return the prevention measure, cause and related information. The hardware part is also connected in the React Web Application which loads the image captured by the raspberry pi camera which uploads the captured images to the Google Drive and with the help of Google Drive REST API and node.js server we get the images and pass that image for the prediction to the Django Server.

1. Front End Application:

Technology Used:

a) React: JavaScript library for building user interfaces.

b) Redux: JavaScript library for maintain state in the applications.

c) Axioms: JavaScript Ajax library for performing Network Request.

The front-end application is composed of static components and the dynamic components. Static components are responsible to load some information about our projects.

The Major Dynamic Components include

Predict: This has image upload form which takes the image from the browser and pass it to the Django Server which is responsible to load the deep learning model and generate the predicted outcomes. The predicted outcomes are further processed and top three predictions are displayed with the score. The one disease with the highest prediction score is query against the Node.js Server which return the further information about the disease including prevention measure, cause etc.

Connect Drive: The images clicked by the AgroBot is displayed in the Browser. When clicking Predict button the images is passed to Django Server and follows the same steps as above.

1. Django Server: Django Server loads the machine learning model and predict the disease of the images uploaded by the front-end web application. All the actions in the server is in the RESTful way so that we can communicate efficiently across various platforms.
2. Node.js Server:

Technologies Used:

a) Node.js: JavaScript runtime built on Chrome's V8 JavaScript engine.

b) MongoDB: MongoDB is a cross-platform document-oriented database program which uses JSON-like documents with schema.

**CHAPTER FIVE: METHODOLIGY**

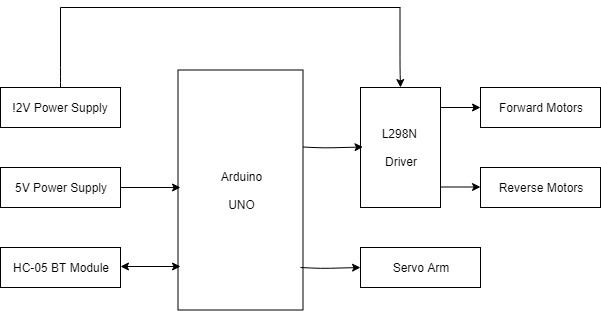
**5.1 Block Diagram**

Figure 5.1: AgroBot Block Diagram

The basic operation of the Agrobot starts by enabling the power to the Agrobot and Raspberry Pi. Now the Agrobot checks for the proper initialization of the system. After checking, the BT module provides pairing signal to the android device. After getting paired with android app, the Agrobot gets commands from the app. The commands are processed by the Arduino board. After processing, Arduino forwards those commands to L298N motor driver and Servo motor for the movement of the Agrobot and camera hand respectively.

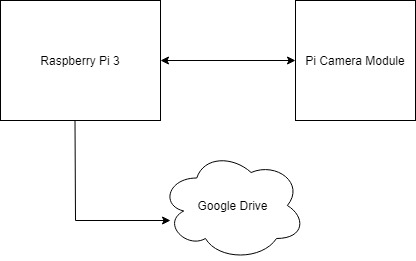


Figure 5.2: Pi and Pi Camera block diagram

After powering the Raspberry Pi, the Motion Eye OS gets loaded. After loading OS, it checks for the available Wi-Fi network and gets connected to the network. After connecting to the network, the Raspberry Pi streams its live video feed to the android app. The user controls the movement of pi camera module by Servo motor via android app. When user captures the image of leaves, its saves one copy of that image to its local storage and sends another copy to the cloud storage i.e. Google Drive.

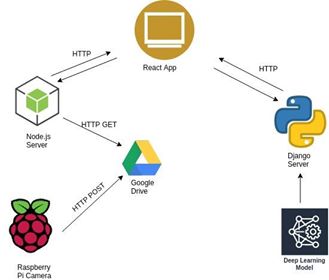


Figure 5.3: Project Flow Diagram

Web application co-ordinates the flow of two sub-ordinates i.e. Agrobot and Deep Learning Model. Image captured by the Agrobot via Raspberry Pi Camera is uploaded to the Google Drive. Using Google Drive API Node.js Server serves the image to the frontend React application. The captured image is then submitted to the Django server which is serving the deep learning model. The deep learning model predict the disease from the image and according to the predicted disease Node.js Server sends information regarding Symptoms, Remedies to frontend.

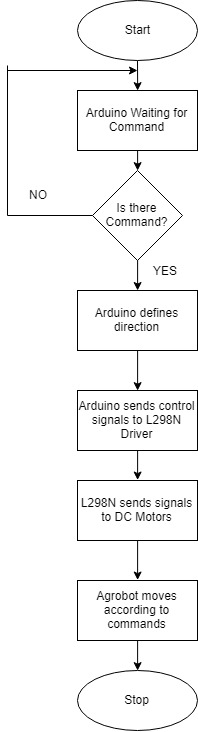
* 1. **Flowchart**

Figure 5.4: AgroBot flowchart

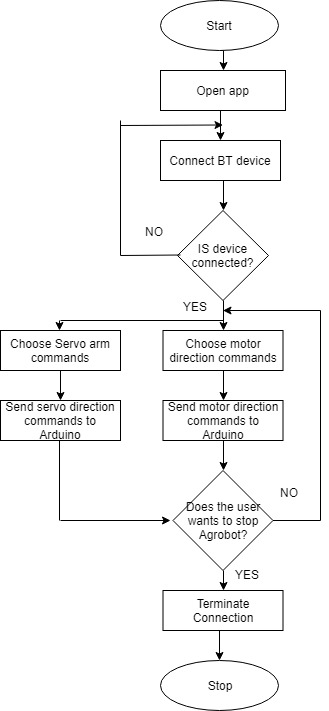
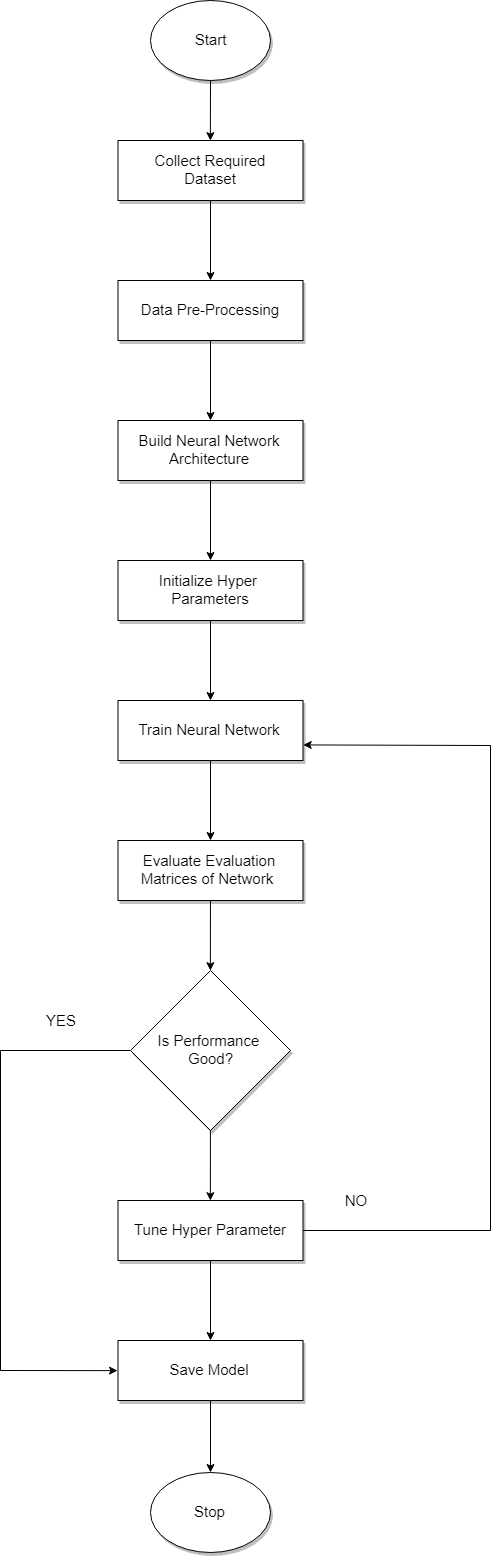
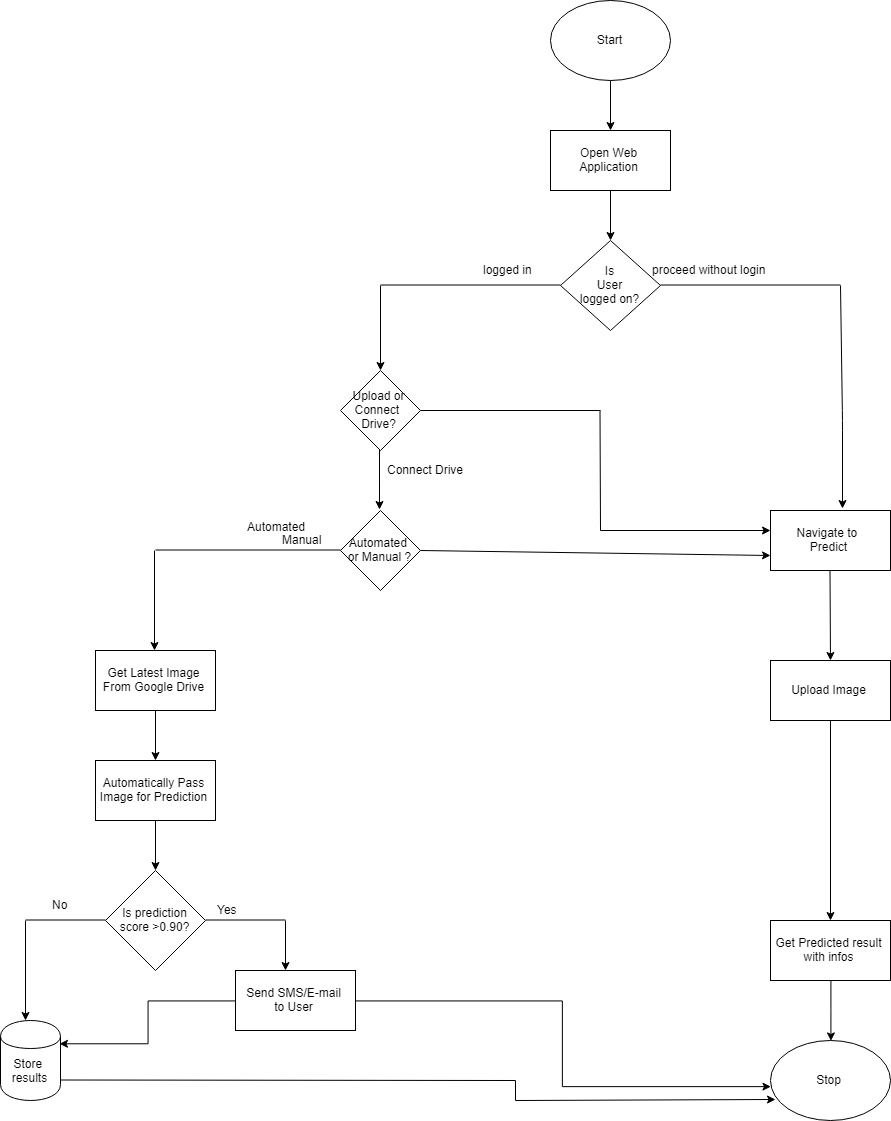


Figure 5.5: Android App flowchart

Figure 5.6: CNN Model flowchart

Figure 5.7: Webapp Flowchart

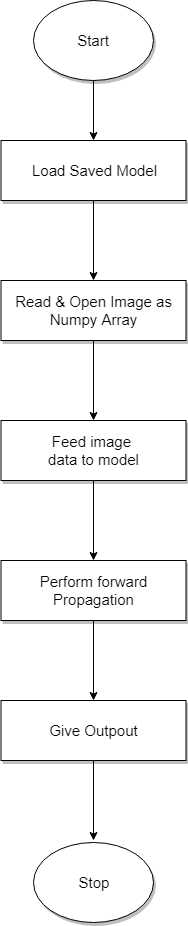
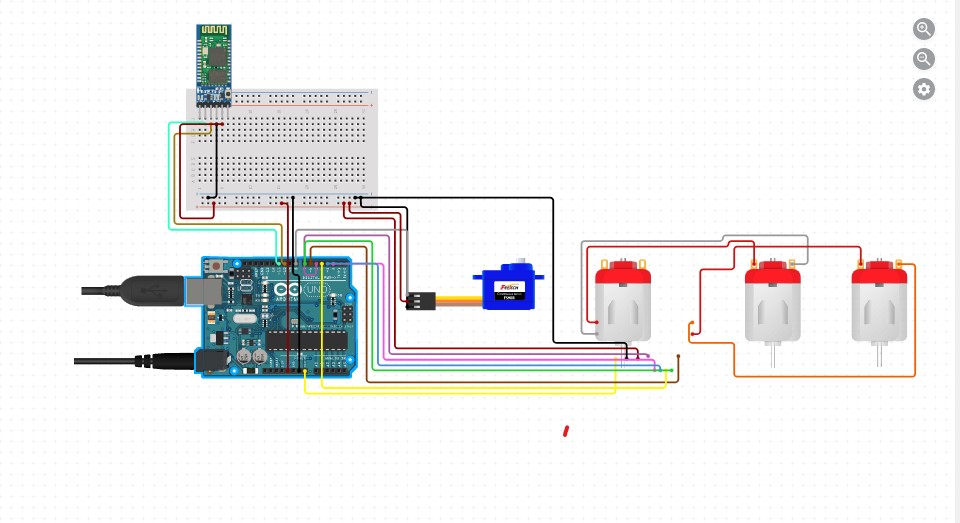


Figure 5.8: User view flowchart

**5.3 Schematic Diagram**

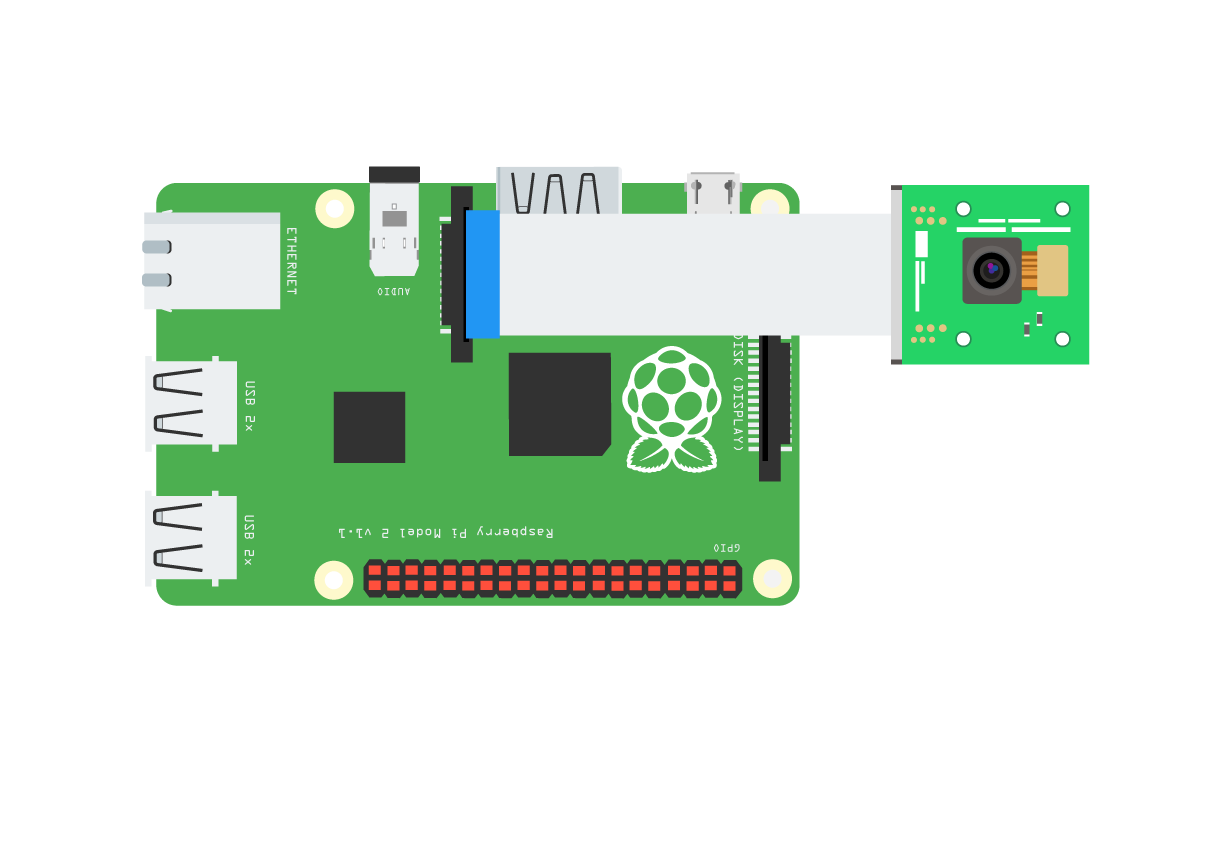
Figure 5.9(a): Agrobot Circuit diagram

Figure 5.9(b): Raspberry Pi circuit diagram

**5.4 Deep learning system for plant diseases classiﬁcation:**

**5.4.1 Datasets**

The prime requirement for the project was the images showing infected parts of leaf caused by various diseases. The courtesy of datasets is Plant Village. PlantVillage is a not-for-profit project by Penn State University in the US and EPFL in Switzerland. They have collected - and continue to collect - tens of thousands of images of diseased and healthy crops. The dataset that we had used offers 38 classes of crop disease pairs with 54305 raw color images. Those data were splitted into 80:20, the ratio of training to testing images by using a data separation generating code. Training images (43755) are used to train the neural network while testing images (10550) are used to test the performance of the network. Multiple approaches can be used for this such as image resizing, cropping part of image, masking and removing the attenuation.

Figure: 5.10: Example of leaf images from the PlantVillage dataset, representing every crop-disease pair used. (1) Apple Scab, Venturia inaequalis (2) AppleBlack Rot, Botryosphaeria obtusa (3) Apple Cedar Rust, Gymnosporangium juniperi-virginianae (4) Apple healthy (5) Blueberry healthy (6) Cherry healthy (7) Cherry Powdery Mildew, Podoshaera clandestine (8) Corn Gray Leaf Spot, Cercospora zeae-maydis (9) Corn Common Rust, Puccinia sorghi (10) Corn healthy (11) Corn Northern Leaf Blight, Exserohilum turcicum (12) Grape Black Rot, Guignardia bidwellii, (13) Grape Black Measles (Esca), Phaeomoniella aleophilum, Phaeomoniella chlamydospora (14) Grape Healthy (15) Grape Leaf Blight, Pseudocercospora vitis (16) Orange Huanglongbing (Citrus Greening), Candidatus Liberibacter spp. (17)Peach Bacterial Spot, Xanthomonas campestris (18) Peach healthy (19) Bell Pepper Bacterial Spot, Xanthomonas campestris (20) Bell Pepper healthy (21) Potato Early Blight, Alternaria solani (22) Potato healthy (23) Potato Late Blight, Phytophthora infestans (24) Raspberry healthy (25) Soybean healthy (26) Squash Powdery Mildew, Erysiphe cichoracearum (27) Strawberry Healthy (28) Strawberry Leaf Scorch, Diplocarpon earlianum (29) Tomato Bacterial Spot, Xanthomonas campestrispv. vesicatoria (30) Tomato Early Blight, Alternaria solani (31) Tomato Late Blight, Phytophthora infestans (32) Tomato Leaf Mold, Passalora fulva (33) Tomato Septoria Leaf Spot, Septoria lycopersici (34) Tomato Two Spotted Spider Mite, Tetranychus urticae (35) Tomato Target Spot, Corynespora cassiicola (36) Tomato Mosaic Virus (37) Tomato Yellow Leaf Curl Virus (38) Tomato healthy.

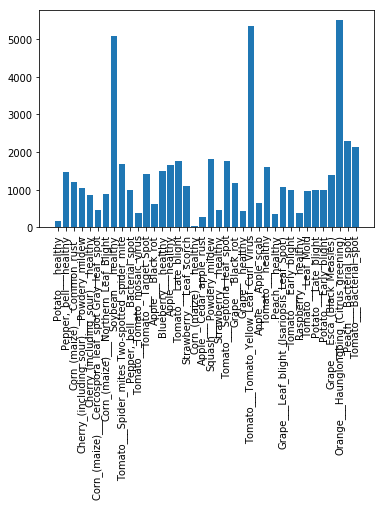
Imagedistribution for each class labels in the datasets can be visualized in bar graph.

Figure 5.11: Image Datasets

**5.4.2 Classification Algorithm**

Our main objective is to construct a highly accurate classifier that generalizes well on data from new individuals. For this purpose, we have tested the performance of different layered convolutional neural networks by switching between different optimization algorithms and tuning hyper-parameters, and assessed why some models performed well while others performed poorly.

**5.4.2.1 Neural Network**

Neural networks are a set of algorithms, modeled loosely after the human brain, that are designed to recognize patterns. They interpret sensory data through a kind of machine perception, labeling or clustering raw input. The patterns they recognize are numerical, contained in vectors, into which all real-world data, be it images, sound and text.

We can model the Plant Disease Detection process by creating a neural network on a computer. A neural network has input and output neurons, which are connected by weighted synapses. The weights affect how much of the forward propagation goes through the neural network. The weights can then be changed during the back propagation — this is the part where the neural network is now learning. This process of forward propagation and backward propagation is conducted iteratively on every piece of data in a training data set. The greater the size of the data set and the greater the variety of data set that there is, the more that the neural network will learn, and the better that the neural network will get at predicting outputs.

**5.4.2.1.1 Neurons**

A neural network is a graph of neurons. A neuron has inputs and outputs. The inputs and outputs of a neural network are represented by input neurons and output neurons. Input neurons have no predecessor neurons but do have an output. Similarly, an output neuron has no successor neuron, but does have inputs. It takes the inputsand multiplies them by their weights**,** then it sums them up, after that it applies the activation function to the sum.

* + - * 1. **Connections and Weights**

A neural network consists of connections, each connection transferring the output of a neuron to the input of another neuron. Each connection is assigned a weight.

**5.4.2.1.3 Learning Rule**

The learning rule is a function that modifies the weights of the connections. This serves to produce a favored output for a given input for the neural network. The learning rule is leveraged during the backward propagation stage of training.

* + - 1. **Convolutional Neural Network**

A **Convolutional Neural Network (ConvNet/CNN)[6]** is a Deep Learning algorithm, which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. The pre-processing (refers to all the transformations on the raw data before it is fed to the machine learning or deep learning algorithm.) required in a CNN is much lower as compared to other classification algorithms. The role of the CNN is to reduce the role of images into a form that which is easier to process, without losing the features, which are critical for getting a good prediction.

In general, the CNN consists of four layers: convolution layers, pooling layers, fully connected layers and output layer. CNN takes an image or a patch of an image as an input and outputs a probability distribution over all classes.

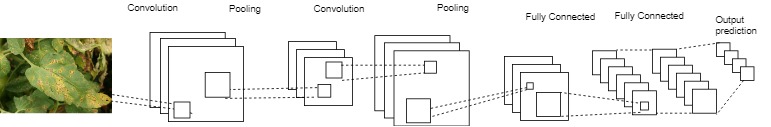


Figure 5.12: CNN layers

* + - * 1. **Convolutional Layer**

A convolutional layer contains a set of filters whose parameters need to be learned. Each filter is convolved with the input volume to compute an [activation](https://www.sciencedirect.com/topics/engineering/activation) map made of [neurons](https://www.sciencedirect.com/topics/engineering/neurons). In other words, the filter is slid across the width and height of the input and the dot products between the input and filter are computed at every [spatial position](https://www.sciencedirect.com/topics/engineering/spatial-position). The output volume of the convolutional layer is obtained by stacking the activation maps of all filters along the depth dimension. Since the width and height of each filter is designed to be smaller than the input, each neuron in the activation map is only connected to a small local region of the input volume. In other words, the receptive field size of each neuron is small, and is equal to the filter size. In addition, as the activation map is obtained by performing convolution between the filter and the input, the [filter parameters](https://www.sciencedirect.com/topics/engineering/filter-parameter) are shared for all local positions. The weight sharing reduces the number of parameters for efficiency of expression, efficiency of learning, and good generalization.

let’s say we have a layer with filter size 5\*5\*3. Also, assume that the input that’s fed to convolutional neuron is an input image of size of 32\*32 with 3 channels. Then, the filter will pick one 5\*5\*3(3 for number of channels in a colored image) sized chunk from image and calculate convolution (dot product) with our filter(w), thus generating 28\*28 sized activation map.[11] After each convolution, the output reduces in size. In a deep neural network with many layers, the output will become very small this way, which doesn’t work very well. So, it’s a standard practice to add zeros on the boundary of the input layer such that the output which is called padding. Let’s say you have an input of size N\*N, filter size is F, you are using S as stride and input is added with 0 pad of size P. Then, the output size will be:

**(N-F+2P)/S +1**

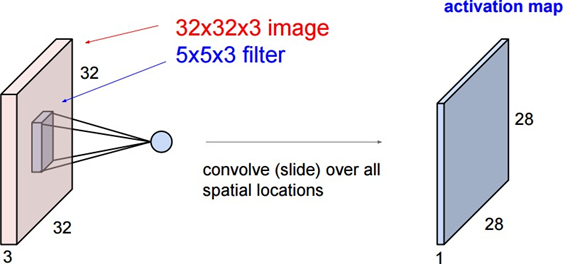
**

Figure 5.13: Working of Convolution Layer

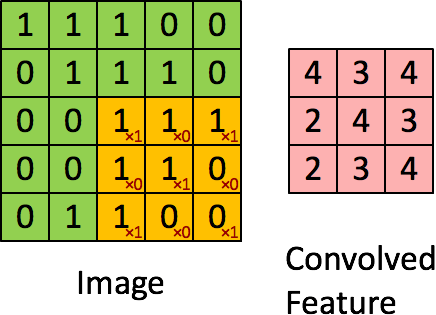


Figure 5.14: Convolution layer output with a filter of 3\*3 and stride 1

**5.4.2.2.2 Pooling Layer**

Pooling layer is mostly used immediately after the convolutional layer to reduce the spatial size (only width and height, not depth). This reduces the number of parameters; hence computation is reduced. Also, less number of parameters avoid overfitting. The most common form of pooling is Max pooling where we take a filter of size F\*F and apply the maximum operation over the F\*F sized part of the image.

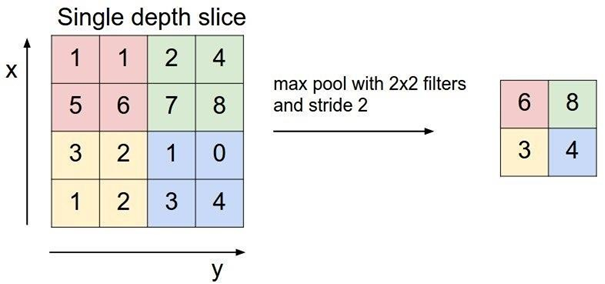


Figure 5.15: Max-pooling with 2\*2 filter

**5.4.2.2.3 Fully Connected Layer**

If each neuron in a layer receives input from all the neurons in the previous layer, then this layer is called fully connected layer. The output of this layer is computed by matrix multiplication followed by bias offset.

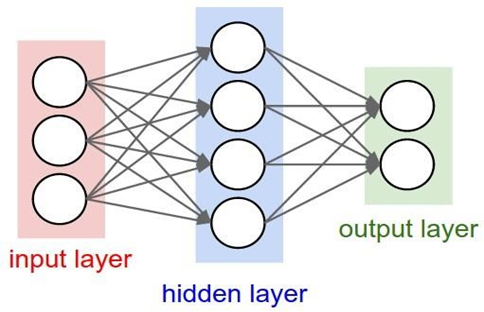


Figure 5.16: Fully Connected Layer

**5.4.2.2.4 Dropout Layer**

Dropout is a technique used to improve over-fit on neural networks. During training half of neurons on a particular layer will be deactivated. This improve generalization because it forces your layer to learn with different neurons the same "concept". During the prediction phase the dropout is deactivated. Normally deep learning models use Dropout on the fully connected layers, but it is also possible to use dropout after the max-pooling layers, creating some kind of image noise augmentation.

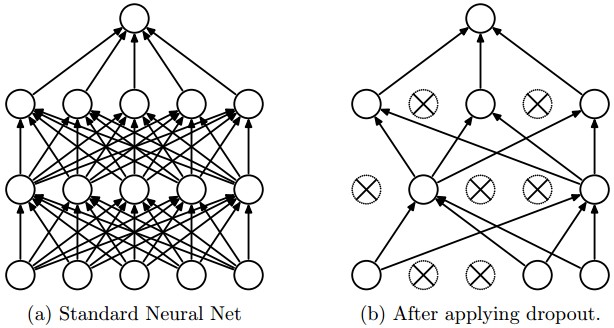


Figure 5.17: Network with and without dropout

* + - * 1. **Working of Convolutional Neural Network**

Neural Networks are essentially mathematical models to solve an optimization problem. They are made of neurons, the basic computation unit of neural networks. A neuron takes an input (say x), do some computation on it (say: multiply it with a variable w and adds another variable b) to produce a value (say; z= wx+b).[11] This value is passed to a non- linear function called activation function(f) to produce the final output(activation) of a neuron. There are many kinds of activation functions. One of the popular activation function is ReLu , which is:

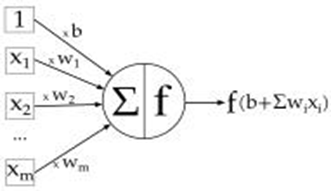


Figure 5.18: A simple neuron

Convolutional Neural Network doesn’t perceive image as like humans do. Convolutional networks perceive images as volumes; i.e. three-dimensional objects, rather than flat canvases to be measured only by width and height. That’s because digital color images have a red-blue-green (RGB) encoding, mixing those three colors to produce the color spectrum humans perceive. So a convolutional network receives a normal color image as a rectangular box whose width and height are measured by the number of pixels along those dimensions, and whose depth is three layers deep, one for each letter in RGB. Those depth layers are referred to as channels. Now, for each pixel of an image, the intensity of R, G and B will be expressed by a number, and that number will be an element in one of the three, stacked two-dimensional matrices, which together form the image volume.

Those numbers are the initial, raw, sensory features being fed into the convolutional network, and the ConvNet purpose is to find which of those numbers are significant signals that actually help it classify images more accurately.

* + 1. **Back Propagation Algorithm**

The weights of each node are assigned randomly at the start. Then an input image along with the label is input in the convolutional neural network. The network calculates the (Wx+b) in each layer and finally outputs a value. Since, it’s the first training example, the network calculates large error and loss. Thus, calculated error is back-propagated to each node, such that each node adjusts their weight according to the error propagated. This process continues for each training example and the node weights will become more able to predict the image label accurately. This is how back propagation algorithm works.

* + 1. **Hyper-parameters of CNN**

Hyper-parameters are fixed before training to tune the training model itself [12]. Hyper- parameter can be separated into 3 categories.

* + - 1. **Neural Nets Structure**
* Number of hidden layers:

A hidden layer is an intermediate set of neurons made by mapping weights and non- linear transformation (activation function) onto neurons in the previous layer. If data is simple like it can be separated them by drawing a straight line, you won't need any hidden layer. Beyond that, it is said one hidden layer is sufficient for the large majority of conventional problem. However, we need more of them when we apply it to more complex problems that we are going to explore.

* Number of nodes in each hidden layer:

In a layer, each node holds different intermediate computational values since each is mapped with different weights configuration. The optimal size of the hidden layer is usually between the size of the input and size of the output layers.

* + - 1. **Numeric Operation**
* Learning Rate:

Step size of adjustments of weights/parameters in each iteration. Though how it impacts is different between machine learning architectures, it should be optimized to find its best value to achieve faster training process and more accurate result. Higher value gives faster learning but it may cause the model to fail to converge. Lower value gives higher chance to converge but may cause the model to learn too slow and even stuck in a local minima/maximum.

* Activation function:

Different activation function changes how we convert weighted values through synapses for a better training. ReLU is a primary choice (avoid gradient vanishing, efficient without pre-training in deep neural nets.

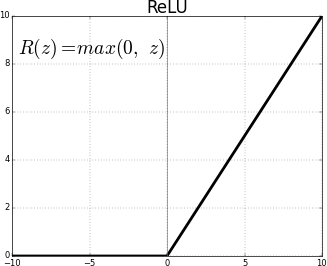
**

Figure 5.19: ReLU Activation Function

**5.4.4.3 Dataset Preconditioning**

* Ratio of training to test: 20%-80%
* Batch size:

Batch size defines how many samples that going to be propagate through network. The smaller the batch size is, the less memory is required. But if it is set too small, the estimate of gradients gets less accurate.

**5.4.5 Optimization Algorithm**

Optimization algorithms[13] helps us to minimize (or maximize) an Objective function (another name for Error function) E(x) which is simply a mathematical function dependent on the Model’s internal learnable parameters which are used in computing the target values(Y) from the set of predictors(X) used in the model. For example — we call the Weights(W) and the Bias(b) values of the neural network as its internal learnable parameters which are used in computing the output values and are learned and updated in the direction of optimal solution i.e minimizing the Loss by the network’s training process and also play a major role in the training process of the Neural Network Model.

**5.4.5.1 Gradient Descent and SGD**

Gradient Descent is the most important technique and the foundation of how we train and optimize Intelligent Systems.

The formula for the parameter updates in forward propagation:

θ=θ−η⋅∇J(θ)

where,

‘η’ is the learning rate,

’∇J(θ)’ is the Gradient of Loss function,

J(θ) w.r.t parameters ‘θ’.

It is the most popular Optimization algorithms used in optimizing a Neural Network. Weights updates in a Neural Network Model, i.e update and tune the Model’s parameters

in a direction so that we can minimize the Loss function. In backpropagation, we first propagate forward calculating the dot product of Inputs signals and their corresponding Weights and then apply a activation function to those sum of products, which transforms the input signal to an output signal.After this we propagate backwards in the Network carrying Error terms and updating Weights values using Gradient Descent, in which we calculate the gradient of Error(E) function with respect to the Weights (W) or the parameters, and update the parameters (here Weights) in the opposite direction of the Gradient of the Loss function w.r.t to the Model’s parameters.

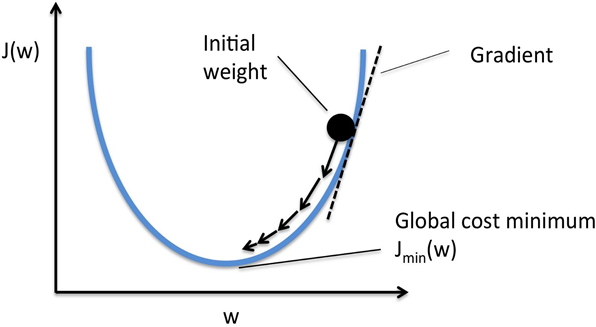


Figure 5.20: Working of Gradient Descent

Stochastic Gradient Descent (SGD) on the other hand performs a parameter update for each training example It is usually much faster technique. It performs one update at a time.

θ=θ−η⋅∇J(θ;x(i);y(i)), where {x(i) ,y(i)} are the training examples.

Now due to these frequent updates, parameters updates have high variance and causes the Loss function to fluctuate to different intensities. This is actually a good thing because it helps us discover new and possibly better local minima, whereas Standard Gradient Descent will only converge to the minimum of the basin as mentioned above.

**5.4.6 Deep Neural Network**

A Deep Neural Network simply has more layers than smaller Neural Networks. A smaller Neural Network might have 1–3 layers of neurons. However, a Deep Neural Network (DNN) [6] has more than a few layers of neurons. A DNN might have 20 or 1,000 layers of neurons.

When designing features or algorithms for learning features, our goal is to separate the factors of variation that explain the observed data. These factors indicate separate influencing sources & are not combined by multiplication. Either they are unobserved

objects/forces in the physical world that affect observable quantities or constructs in human mind providing simplified explanations or inferred causes of the observed data. They are concepts or abstractions that help us make sense of the rich variability in the data.

Deep learning solves this central problem in representation learning by introducing representations that are expressed in terms of other, simpler representations as it enables the computer to build complex concepts out of simpler concepts. Quintessential example of a deep learning model is the feed forward deep network, or multilayer perceptron (MLP). A multilayer perceptron is just a mathematical function formed by combining many simpler functions to map some set of input values to output values. Each application of a different mathematical function provides a new representation of the input.

Apart from learning right representation from data, another aspect is depth that enables computer to learn a multi-step program. Each layer of a representation is a state of the computer’s memory after simultaneously executing another set of instructions & that empowers networks with greater depth to execute more instructions in sequence. Later instructions can refer back to the results of prior instructions, so all the information in a layer’s activation don’t necessarily encode factors of variation that explain the input. Representation also stores state information that helps to execute a program that can make sense of the input and keep model processing organized.[15]

Depth of a model can be viewed either based on number of sequential instructions (depth of Computational graph) OR based on correlation of concepts with each other (depth of Probabilistic modeling graph). Neither there is a single correct value for the depth of an architecture, nor is there a consensus about how much depth a model requires to qualify as ‘deep’. However, Deep Learning can be safely regarded as the study of models that involve a greater amount of composition of either learned functions or learned concepts than traditional machine learning does.[15]

* + 1. **Model Evaluation**

Predictive Modeling works on constructive feedback principle. Once a model is built, its feedbacks are collected from metrics, improvements are made and the process is continued until desired accuracy or desired value for certain metric is achieved. Computing just the accuracy score for a classification model gives an incomplete picture of model’s performance. Especially in this type of classification problem where number of benign images in sample are greater than malignant images, evaluation metrics other than accuracy should be calculated. These evaluation metrics explain the performance of a model. An important aspects of evaluation metrics is their capability to discriminate among model results. The evaluation metrics used in this project are described below:

* + - 1. **Classification Accuracy**

Classification Accuracy is what we usually mean, when we use the term accuracy. It is the ratio of number of correct predictions to the total number of input samples.

𝑁𝑢𝑚𝑏𝑒𝑟 𝑜𝑓 𝐶𝑜𝑟𝑟𝑒𝑐𝑡 𝑃𝑟𝑒𝑑𝑖𝑐𝑡𝑖𝑜𝑛𝑠

𝐴𝑐𝑐𝑢𝑟𝑎𝑐𝑦 =  ——————————————————

𝑇𝑜𝑡𝑎𝑙 𝑁𝑢𝑚𝑏𝑒𝑟 𝑜𝑓 𝑃𝑟𝑒𝑑𝑖𝑐𝑡𝑖𝑜𝑛𝑠 𝑚𝑎de

It works well only if there are equal number of samples belonging to each class. The real problem arises, when the cost of misclassification of the minor class samples are very high.

**5.4.8 Confusion Matrix**

A confusion matrix is a technique used for summarizing the performance of a classification algorithm i.e. it has binary outputs. Example for a classification algorithm: Predicting if the patient has cancer. Here, there can only be two outputs i.e. Yes or No. A confusion matrix gives us a better idea of what our classification model is predicting right and what types of errors it is making. The number of correct and incorrect predicted values is summarized with count values and stored in the table against the actual values. The major 4 terms associated with confusion matrix are:

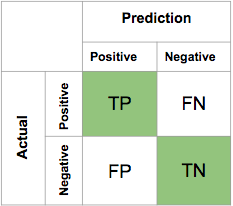


Figure 5.21: Confusion Matrix

* **True Positives (TP):**

True positives are the cases when the actual class of the data point was 1(True) and the predicted is also 1(True). Ex: The case where a plant is actually having Apple scab (1) and the model classifying this case as Apple scab (1) comes under True positive.

* **True Negatives (TN):**

True negatives are the cases when the actual class of the data point was 0(False) and the predicted is also 0(False). Ex: The case where a plant not having Apple scab and the model classifying this case as not Apple scab comes under True Negatives.

* **False Positives (FP):**

False positives are the cases when the actual class of the data point was 0(False) and the predicted is 1(True). False is because the model has predicted incorrectly and positive because the class predicted was a positive one. Ex: A plant not having Apple scab and the model classifying this case as Apple scab comes under False Positives.

* **False Negatives (FN):**

False negatives are the cases when the actual class of the data point was 1(True) and the predicted is 0(False). False is because the model has predicted incorrectly and negative because the class predicted was a negative one. Ex: A plant having Apple scab and the model classifying this case as not Apple scab comes under False Negative.

The ideal scenario that we all want is that the model should give 0 False Positives and 0 False Negatives. But that’s not the case in real life as any model will NOT be 100% accurate most of the times.

**5.4.8.1 Precision**

It is the number of correct positive results divided by the number of positive results predicted by the classifier.

𝑇𝑟𝑢𝑒 𝑃𝑜𝑠𝑖𝑡𝑖𝑣𝑒

𝑃𝑟𝑒𝑐𝑖𝑠𝑖𝑜𝑛 = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

𝑇𝑟𝑢𝑒 𝑃𝑜𝑠𝑖𝑡𝑖𝑣𝑒 + 𝐹𝑎𝑙𝑠𝑒 𝑃𝑜𝑠𝑖𝑡𝑖𝑣e

Precision is a measure that tells us what proportion of plants that diagnosed as having disease, actually diseased.

**5.4.8.2 Recall**

It is the number of correct positive results divided by the number of all relevant samples (all samples that should have been identified as positive).

𝑇𝑟𝑢𝑒 𝑃𝑜𝑠𝑖𝑡𝑖𝑣𝑒

𝑅𝑒𝑐𝑎𝑙𝑙 = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

𝑇𝑟𝑢𝑒 𝑃𝑜𝑠𝑖𝑡𝑖𝑣𝑒 + 𝐹𝑎𝑙𝑠𝑒 𝑁𝑒𝑔𝑎𝑡𝑖𝑣𝑒

Recall is a measure that tells us what proportion of plants that actually had was diagnosed by the algorithm as diseased plant.

**5.4.8.3 Score**

F1 Score is the weighted average of Precision and Recall. Therefore, this score takes both false positives and false negatives into account.

2.𝑝𝑟𝑒𝑐𝑖𝑠𝑖𝑜𝑛 ∙ 𝑟𝑒𝑐𝑎𝑙l

𝑓1 − 𝑠𝑐𝑜𝑟𝑒 = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

𝑝𝑟𝑒𝑐𝑖𝑠𝑖𝑜𝑛 + 𝑟𝑒𝑐𝑎𝑙𝑙

It is useful when we need to take both precision and recall into account. If we try to only optimize recall, our algorithm will predict most examples to belong to the positive class, but that will result in many false positives and, hence, low precision. On the other hand, if we try to optimize precision, our model will predict very few examples as positive results (the ones which highest probability), but recall will be very low. Finding the good trade-off between precision and recall becomes tough rescue which enables one to evaluate a model with single metric.

**5.4.9** **Neural Network Architecture**

Different neural network architecture is sketched and implemented using the machine learning library and images are fed into the network. The network is trained with the training images and tested with unseen datasets to evaluate the ability of model to generalize across the problem domain. The major model that we use is described below:

**5.4.9.1 Model**

After having required number of datasets for this stage of project and formulated basic algorithm, neural network is used and data is fitted into the network to see how the network fares. It is mainly for viewing of the output result and carrying out the optimization as suggested by the output result. After analyzing the output result, we kept on calibrating parameters, adding and removing optimizers until the next model with higher performance than the previous model is found. Deeper and denser network is used by using convolutional layers and fully connected layer, thus making it similar to VGGnet. This showed little improvement in the model. Since, the deep network increases the chance of model being overfit, dropout is increased as a regularization parameter

* + - 1. **Architecture**
* Convolutional Neural Network is chosen having 13 convolution layers and 3 fully connected layer.
* The convolution layers were followed by Max-Pool operation and Batch Normalization was done after Max-Pool.
* For activation function, Rectified Linear Units (ReLU) are used for introducing nonlinearity.
* For optimization, optimizer ‘SGD’ is used with its default settings as the optimizer.
* The total data is splitted into 80:20 (training set:testing set ratio). Images are scaled into 224\*224 pixels and batch size of 64 is set.

The architecture of thus formed neural network can be viewed as follows:



Figure 5.22: Architecture of Model

**5.4.10 Algorithms**

* **Model Building Using CNN**

Step 1: Start

Step 2: Collect required Dataset

Step 3: Data Pre-Processing

Step 4: Build Neural Network Architecture

Step 5: Initialize Hyper-parameter

Step 6: Train Neural Network

Step 7: Evaluate evaluation metrics of the network

Step 8: Is performance good?

If Yes, Go to Step 10 If No, Go to Step 9

Step 9: Tune Hyper-parameters Go to Step 6

Step 10: Save model

Step 11: Exit

* **Prediction of an image using saved model**

Step 1: Start

Step 2: Load saved model

Step 3: Read Image path and open image as numpy array

Step 4: Feed image data to the model

Step 5: Perform forward propagation

Step 6: Give output

Step 7: End

* **User’s view**

Step 1: Start

Step 2: Enter the homepage of the app

Step 3: Click on Start Now Button

Step 4: Upload image in the Drop or Upload Image section.

Step 5: Get prediction result

Step 6: End

* **Working of the webapp**

Step 1: Start

Step 2: Start the server

Step 3: Wait for the POST request containing image details

Step 4: Get the image and send command to Python module to predict the file

Step 5: Receive predicted data from Python module

Step 6: Display data to User

Step 7: End

**Example:** For the disease Apple Scab, the collected information includes

"CAUSAL AGENT": "Apple scab is caused by the fungus Venturia inaequalis. It is common on susceptible apples and crabapples and causes leaves to yellow and drop prematurely in midsummer. In years with moist and cool spring weather, infection can cause widespread defoliation",

"EFFECT/SYMPTOMS": "The disease usually noticed on leaves and fruits. Affected leaves become twisted or puckered and have black, circular spots on their upper surface. On the under surface of leaves, the spots are velvety and may coalesce to cover the whole leaf surface. Severely affected leaves may turn yellow and drop. Scab can also infect flower stems and cause flowers to drop. The lesions later become sunken and brown and may have spores around their margins Infected fruit become distorted and may crack, allowing entry of secondary organisms.",

"CONTROL MEASURES":

"Cultural Control: In affected orchards, new infections can be reduced by removing leaf litter and trimmings containing infected tissue from the orchard and incinerating them. This will reduce the amount of new ascospores released in the spring. Additionally, scab lesions on woody tissue can be excised from the tree if possible and similarly destroyed.

Chemical Control: Chemical controls can include a variety of compounds. Benzimidazole fungicides, e.g., Benlate (now banned in many countries due to it containing the harmful chemical benzene) work well but resistance can arise quickly. A number of other chemical classes including sterol inhibitors such as Nova 40, and strobilurins such as Sovran are used extensively; however, some of these are slowly being phased out because of resistance problems.",

We have added the similar kind of information for the all the diseases among 38 different classed and stored the data in MongoDB hosted at the MLAB (cloud database service that hosts MongoDB databases.)

Once the model predicts the disease from the image the Web Application request data from the server and the server return the requested data in the similar form as shown above.

A) Update the Database Collection: We have used MongoDB No-SQL Database to store information about the disease in the datasets. We have manually searched in the Search engine to collect information about the corresponding diseases. The information collected includes

Disease, Causal agent, effects/symptoms, control measures.

B) Connect to the AgroBot via Google Drive: The remote AgroBot running in the farm land sends the image captured by it via HTTP protocols to the Google Drive. We have used Google Drive API to get that images and pass the images to the front-end web application.

C) Mail/SMS Client: In the Web Application User have two option after connecting to their Google Drive. Either they can click predict in the selected images or the application can do automatically prediction to the all the clicked image. If the predicted score is above certain threshold than the Mail/Telephone Client is activated which sends the alert message to the User.

**CHAPTER SIX: RESULTS AND ANALYSIS**

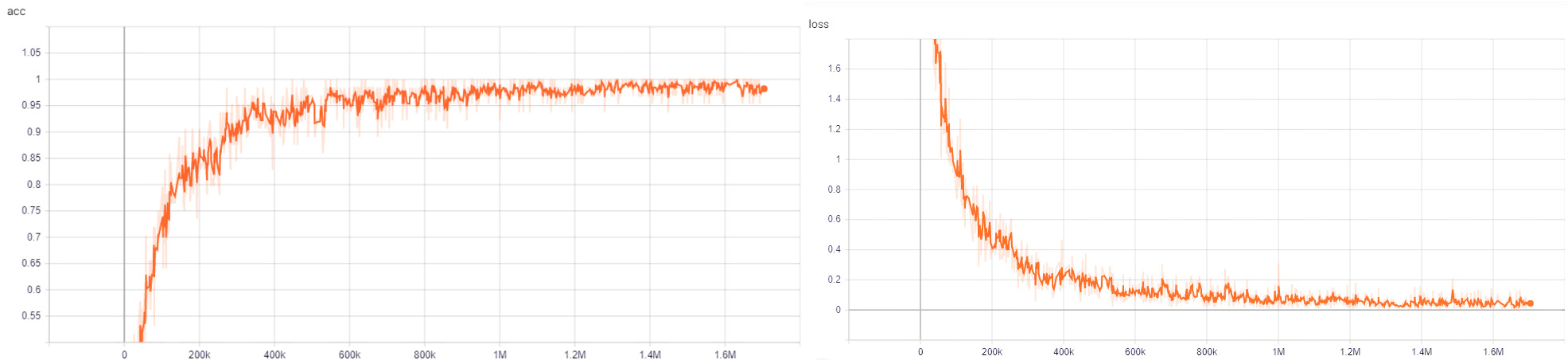
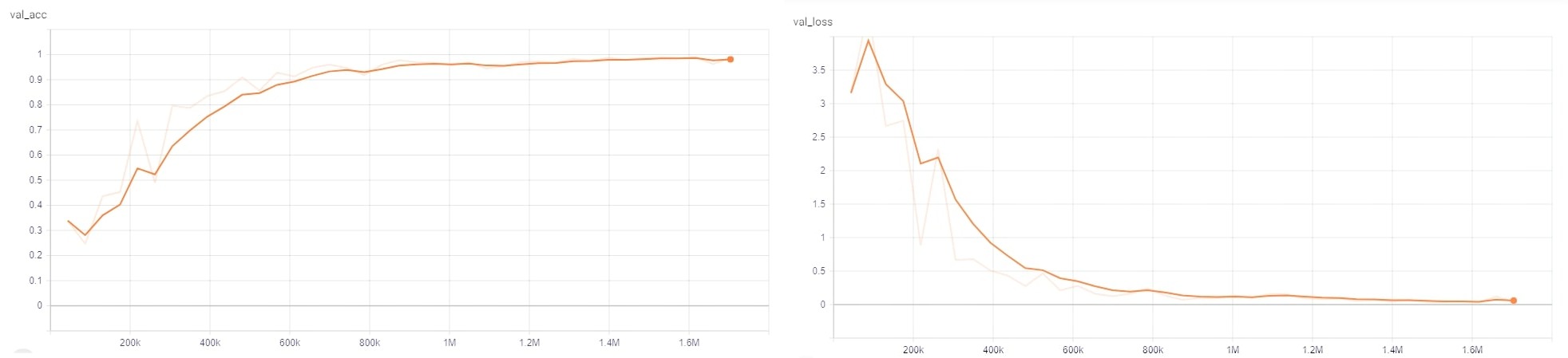
The output from the model that was built and trained upon are described and analyzed. The architecture based on the VGG Model was built, the data is trained using the training set and simultaneously the performance of network is tested using the testing set. The best accuracy reported by this model is around 98.78% achieved after 50 epochs and validation accuracy was 98.76%. After 35 epochs the accuracy remains same for the all other epochs hence the training process terminated after 50 epochs.

Figure 6.1: Training accuracy and loss of the model

Figure 6.2: Testing accuracy and loss of the model

Other hyper-parameters that were set for this model are:

* Learning rate: 0.001
* Learning rate decay factor:0.0001
* Activation function: ReLU
* Ratio of training to test: 80:20
* Batch Size: 64
* Optimization Algorithm: SGD with default settings
* Image Size: 224\*224
* Dropout: 0.4

**6.1 Confusion Matrix:**

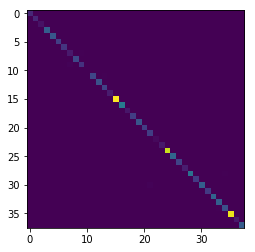


Figure 6.3: Confusion Matrix

**6.2 Result**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | precision | recall | fscore | support |
| 0 | 1.0 | 0.9603174603174603 | 0.9797570850202428 | 126 |
| 1 | 0.9612403100775194 | 1.0 | 0.9802371541501977 | 124 |
| 2 | 1.0 | 1.0 | 1.0 | 55 |
| 3 | 0.9969604863221885 | 0.9969604863221885 | 0.9969604863221885 | 329 |
| 4 | 1.0 | 0.9766666666666667 | 0.988195615514334 | 300 |
| 5 | 1.0 | 0.9809523809523809 | 0.9903846153846153 | 210 |
| 6 | 0.9714285714285714 | 1.0 | 0.9855072463768115 | 170 |
| 7 | 0.8804347826086957 | 0.9101123595505618 | 0.8950276243093923 | 89 |
| 8 | 1.0 | 1.0 | 1.0 | 238 |
| 9 | 0.9532163742690059 | 0.9367816091954023 | 0.9449275362318841 | 174 |
| 10 | 1.0 | 1.0 | 1.0 | 4 |
| 11 | 0.9956896551724138 | 0.9788135593220338 | 0.9871794871794871 | 236 |
| 12 | 0.9963636363636363 | 0.9927536231884058 | 0.9945553539019963 | 276 |
| 13 | 0.9513274336283186 | 1.0 | 0.9750566893424036 | 215 |
| 14 | 0.9767441860465116 | 1.0 | 0.988235294117647 | 84 |
| 15 | 0.9981851179673321 | 0.9990917347865577 | 0.9986382206082615 | 1101 |
| 16 | 0.9870689655172413 | 0.9978213507625272 | 0.9924160346695556 | 459 |
| 17 | 1.0 | 0.9305555555555556 | 0.9640287769784173 | 72 |
| 18 | 0.99 | 0.9949748743718593 | 0.9924812030075189 | 199 |
| 19 | 0.9965753424657534 | 0.9864406779661017 | 0.991482112436116 | 295 |
| 20 | 0.9802955665024631 | 0.995 | 0.9875930521091811 | 200 |
| 21 | 0.9342723004694836 | 0.995 | 0.963680387409201 | 200 |
| 22 | 1.0 | 1.0 | 1.0 | 30 |
| 23 | 0.9487179487179487 | 1.0 | 0.9736842105263158 | 74 |
| 24 | 0.9970472440944882 | 0.9950884086444007 | 0.9960668633235005 | 1018 |
| 25 | 1.0 | 1.0 | 1.0 | 362 |
| 26 | 1.0 | 1.0 | 1.0 | 221 |
| 27 | 0.967391304347826 | 0.978021978021978 | 0.9726775956284153 | 91 |
| 28 | 0.988009592326139 | 0.9694117647058823 | 0.9786223277909738 | 425 |
| 29 | 0.984375 | 0.945 | 0.9642857142857143 | 200 |
| 30 | 0.9823008849557522 | 0.9487179487179487 | 0.9652173913043478 | 351 |
| 31 | 0.9946524064171123 | 0.9789473684210527 | 0.9867374005305041 | 190 |
| 32 | 0.9777777777777777 | 0.9943502824858758 | 0.9859943977591036 | 354 |
| 33 | 0.9734513274336283 | 0.9850746268656716 | 0.9792284866468842 | 335 |
| 34 | 0.9571428571428572 | 0.9571428571428572 | 0.9571428571428572 | 280 |
| 35 | 0.9962616822429906 | 0.9953314659197012 | 0.9957963568425968 | 1071 |
| 36 | 0.9866666666666667 | 1.0 | 0.9932885906040269 | 74 |
| 37 | 0.9937304075235109 | 0.9968553459119497 | 0.9952904238618524 | 318 |

Table 6.1: Model Evaluation Scores

* 1. **Future Enhancement**
     1. **Collecting more data**

Collecting the images of various disease is very time consuming and classification of these images to respective class of disease is not possible without the help of expert. So, we have downloaded around 54K images for this project but if we are planning to take this project forward, we need to collect more data samples of diseased plant leaf images. The dataset we use in this project is combination of images of a plant or part of a plant taken both under a controlled environment. When tested on a set of images taken at a different environment than the images used for the training, however, the model’s accuracy dropped to low value. So, we need to collect more data samples of diseased plant leaf images in controlled environment as well as in the natural environment. Since technique of disease detection based on a set of masks generated by analysis of the color, lightness and saturation components of different parts of the images in several color spaces. During Validating, We are facing the accuracy problem when we fed the other image which does not lie on same image space as the training image.

The second limitation is that we are currently constrained to the classification of single leaves, facing up, on a homogeneous background. While these are straightforward conditions, a real-world application should be able to classify images of a disease as it presents itself directly on the plant. Indeed, many diseases don’t present themselves on the upper side of leaves only (or at all), but on many different parts of the plant. Thus, new image collection efforts should try to obtain images from many different perspectives, and ideally from settings that are as realistic as possible. Then only this project resolves the typical problem of traditional agriculture and stereotypical cultivation of agricultural countries like Nepal in an efficient manner.

**6.3.2 Autonomous Agrobot**

We can build an autonomous robot that can move in a farm environment without damaging existing plants or soil and use object detection to find and mark diseased crops with and environmentally safe color.  A uniform robotic platform going around the farm will solve problems like manually inspecting large farms using phones to mark the crops and make the marking much faster. The speed can also make it easier to share the platform between multiple farms.

**6.3.3 Real-Time Plant Diseases and Pests Recognition**

By the use of recent development in object detection and recognition system, we can further develop new model with the ability to deal with complex scenarios from a plant’s surrounding area in real time.

**CHAPTER SEVEN: CONCLUSION**

As the conclusion, the report includes the final status of the project. We have implemented the basics of AI, deep machine learning, interconnected hardware and fully functional Web application to predict the disease of the leaves. We have trained model with different parameters, optimization algorithms and successfully achieved over 98% validation accuracy. The system was integrated with the web application from where user can upload an image or can use AgroBot to capture the images of leaves from the remote location and predict the disease of the leaves with the possible solution and related information.

**CHAPTER EIGHT: REFERENCE**

[1] Brahimi, Mohammed & Arsenovic, Marko & Laraba, Sohaib & Sladojevic, Srdjan & Kamel, Boukhalfa & Moussaoui, Abdelouhab. Deep Learning for Plant Diseases: Detection and Saliency Map Visualization. 2018.

[2] Lu, Yang & Yi, Shujuan & Zeng, Nianyin & Liu, Yurong & Zhang, Yong. Identification of Rice Diseases using Deep Convolutional Neural Networks. Neurocomputing. 2017.

[3] S. Atabay, H. A. 2016b. A convolutional neural network with a new architecture applied on leaf classification. IIOAB J 7(5):226–331.

[4] Reyes, A. K.; Caicedo, J. C.; and Camargo, J. E. 2015. Fine-tuning deep convolutional networks for plant recognition. In CLEF (Working Notes)

[5] Laga, Hamid & Kurtek, Sebastian & Srivastava, Anuj & Miklavcic, Stanley. (2013). Statistical shape models of plant leaves. 3503-3507. 10.1109/ICIP.2013.6738723.

[6] M. Nielsen, "Neural Networks and Deep Learning", *Neuralnetworksanddeeplearning.com*, 2019. [Online]. Available:<http://neuralnetworksanddeeplearning.com/chap2.html>.

[7] A. RAFFIN, "Autonomous Racing Robot with an Arduino, a Raspberry Pi and a Pi Camera", Becoming Human: Artificial Intelligence Magazine, 2017.

[8]GitHub.(2019). *ccrisan/motioneyeos*.[online] Available at: https://github.com/ccrisan/motioneyeos

[9] Chenghai Yang, "A high-resolution airborne four-camera imaging system for agricultural remote sensing", Computers and Electronics in Agriculture, pp. 13-24, 2012.

[10] S. Arif, A. Polonsky, J. Yu-Co, D. W and F. Robot, "Farmaid: Plant Disease Detection Robot", Hackster.io, 2018

[11] Shaid. *Convolutional Neural Network*. Medium. (2019). [online] Available at: https://towardsdatascience.com/covolutional-neural-network-cb0883dd6529

[12]TakaI.“GrowNeuralNetsPart1:IntrotoHyperparameters”.https://medium.com /@tk2bit/grow-neural-nets-part-1-intro-to-hyperparameters- 39e59b212118

[13]Anish S. “Types of Optimization Algorithms used in Neural Networks and Ways to OptimizeGradientDescent”.https://towardsdatascience.com/types-of- optimization-algorithms-used-in-neural-networks-and-ways-to-optimize-gradient- 95ae5d39529f

[14] Reyes, A., Caicedo, J. and Camargo, J. (2014). *Fine-tuning Deep Convolutional Networks for Plant Recognition*. [ebook] Colombia. Available at: http://ceur-ws.org

[15]Sunindu D. “Introduction to Deep Learning”. https://medium.com/machine- learning-bootcamp/introduction-to-deep-learning-part-1-cbfda681c5ff

[16] Mohanty S. P, Hughes D. P and Salathe, M. 2016. Using deep learning for image-based plant disease detection. Frontiers in plant science