



A Project Report on
Crop Monitoring and Diagnosis using IoT and CNN

Submitted in partial fulfillment of the requirements for the degree of

BACHELOR OF TECHNOLOGY

in

Electronics and Communication Engineering

by

Ashish Reddy K 1860605

Megha Chandra Reddy N V 1860611

Neeraj Puppala 1860613

Sudeep Reddy N 1860649

Under the Guidance of

Dr. Sujatha S

Department of Electronics and Communication Engineering

School of Engineering and Technology,

CHRIST (Deemed to be University),

Kumbalaguda, Bengaluru - 560 074

May-2022



School of Engineering and Technology

Department of Electronics and Communication

Engineering

CERTIFICATE

This is to certify that **Ashish Reddy K, Megha Chandra Reddy N V, Neeraj Puppala and Sudeep Reddy N** has successfully completed the project work entitled “**Crop Monitoring and Diagnosis using IoT and CNN**” in partial fulfillment for the award of **Bachelor of Technology in Department of Electronics and Communication Engineering** during the year **2021-2022**.

Dr. Sujatha S

Assistant Professor

Dr. Inbanila K

Head of the Department

Dr. Iven Jose

Dean



School of Engineering and Technology

Department of Electronics and Communication

Engineering

BONAFIDE CERTIFICATE

It is to certify that this project titled “**Crop Monitoring and Diagnosis using IoT and CNN**” is the bonafide work of

Name	Register Number
Ashish Reddy K	1860605
Megha Chandra Reddy N V	1860611
Neeraj Puppala	1860613
Sudeep Reddy N	1860649

Examiners [Name and Signature]

Name of the Candidate :

1.

Register Number :

2.

Date of Examination :

Acknowledgement

We would like to thank **Dr. Rev. Fr. Abraham V M**, Vice Chancellor, CHRIST (Deemed to be University), **Dr. Rev. Fr. Joseph CC**, Pro Vice Chancellor, **Dr. Fr. Benny Thomas**, Director, School of Engineering and Technology and **Dr. Iven Jose**, Dean for their kind patronage.

We would also like to express sincere gratitude and appreciation to **Dr. Inbanila K**, Head of the Department, Department of Electronics and Communication Engineering for giving me this opportunity to take up this project.

We also extremely grateful to my guide, **Dr. Sujatha S**, who has supported and helped to carry out the project. Her constant monitoring and encouragement helped me keep up to the project schedule.

We would like to thank all faculty and non-teaching staff, who has helped me to carry out this project.

Declaration

We, hereby declare that the project titled “**Crop Monitoring and Diagnosis using IoT and CNN**” is a record of original project work undertaken for the award of the degree of **Bachelor of Technology in Department of Electronics and Communication Engineering**. We have completed this study under the supervision of **Dr. Sujatha S**, Electronics and Communication Engineering

We also declare that this project report has not been submitted for the award of any degree, diploma, associate ship, fellowship or other title anywhere else. It has not been sent for any publication or presentation purpose.

Place: School of Engineering and Technology,
CHRIST (Deemed to be University),
Bengaluru

Date:

Name	Register Number	Signature
Ashish Reddy K	1860605	
Megha Chandra Reddy N V	1860611	
Neeraj Puppala	1860613	
Sudeep Reddy N	1860649	

Abstract

Agriculture is one of the primary occupations in many countries. Improper cultivation methods and failure to identify the diseases when it is in the nascent stage result in the reduction of crop yield, thus affecting the outcome of cultivation. This project proposes a novel method of crop monitoring and early identification of diseases in plants by using the Internet of Things (IoT), convolutional neural networks (CNN), and image processing.

Crop monitoring was achieved with the help of IoT. With the help of Arduino and sensors, data regarding temperature, humidity, moisture, rain status, and motion detection were gathered. With the help of LORA technology, the data is transmitted up to 10km, which is a significant advantage in remote areas. The sensor data is uploaded to the cloud server, helping the farmer to view the sensor status from anywhere on his mobile. Actuators were also added in order to respond to the sensor data.

Crop diagnosis is achieved with the help of CNN and Image Processing. The disease diagnosis was confined to tomato plants for the prototype but can be extended to other crops. An optimum CNN model was developed by analyzing various architectures of CNN, including the VGG, ResNet, Inception, Xception, MobileNet, and DenseNet. The performance of each of these architectures was compared, and various metrics such as the accuracy, loss, precision, recall, and Area Under the Curve (AUC) were analyzed. The best of these models is selected for crop diagnosis.

Both the crop monitoring and crop diagnosis were integrated into a single application. The farmer can know the status of the field with the help of sensors. If there is any problem with crop health, farmers can use the same application for disease diagnosis and know the type of diseases and remedies/fertilizers he/she needs to use. This project acts as an end-to-end process in crop cultivation.

Keywords: Convolutional Neural Networks (CNN), Image Processing, Internet of Things (IoT), LORA.

Contents

CERTIFICATE	i
BONAFIDE CERTIFICATE	ii
ACKNOWLEDGEMENT	iii
DECLARATION	iv
ABSTRACT	v
LIST OF FIGURES	ix
LIST OF TABLES	xi
GLOSSARY	xii
1 INTRODUCTION	1
1.1 Problem Formulation	1
1.2 Problem Identification	2
1.3 Problem Statement & Objectives	2
1.3.1 Problem Statement	2
1.3.2 Objectives	2
1.4 Limitations	3
2 LITERATURE SURVEY AND REVIEW	4
2.1 Literature Collection & Segregation	6
2.2 Critical Review of Literature	9
3 METHODOLOGY	18
3.1 Crop Monitoring	18
3.1.1 Methodology for the Crop Monitoring	18
3.1.2 Experimental and Analytical Work Completed in the Crop Monitoring	19
3.1.2.1 Components Description	19
3.1.2.2 Tools and Technologies	27

3.1.3	Modeling, Analysis and Design of Crop Monitoring	32
3.1.3.1	Soil Moisture Sensor Configuration	32
3.1.3.2	DHT-11 Sensor Configuration	34
3.1.3.3	Rain Sensor Configuration	34
3.1.3.4	PIR Sensor Configuration	34
3.1.3.5	Speaker and LED Configuration	35
3.1.3.6	LoRa Transmitter Configuration	35
3.1.3.7	Schematic Diagrams	35
3.1.3.8	LoRa Receiver Configuration	38
3.1.3.9	Relay Configuration	38
3.2	Crop Diagnosis	39
3.2.1	Methodology for the Crop Diagnosis	39
3.2.2	Experimental and Analytical Work Completed in the Crop Diagnosis	39
3.2.2.1	Data Acquisition	40
3.2.2.2	Data pre-processing	42
3.2.2.3	Data augmentation	42
3.2.3	Modeling, Analysis and Design of Crop Diagnosis	42
3.2.3.1	VGG19	43
3.2.3.2	ResNet152V2	43
3.2.3.3	InceptionV3	44
3.2.3.4	InceptionResNet152V2	45
3.2.3.5	Xception	45
3.2.3.6	MobileNetV2	46
3.2.3.7	DenseNet201	46
3.2.4	Optimizer and Loss Function	47
3.3	Mobile Application Development	48
3.3.1	Methodology for the Mobile Application Development	48
3.3.2	Experimental and Analytical Work Completed in the Mobile Application Development	48
3.3.2.1	Tools and Technologies	48
3.3.3	Modeling, Analysis and Design of Mobile Application Development	50
4	RESULTS, DISCUSSIONS AND CONCLUSIONS	52
4.1	Results and Analysis	52
4.1.1	Results and Analysis of Crop Monitoring	52
4.1.1.1	Working of Soil Moisture Sensor	52
4.1.1.2	Working of DHT-11 Sensor	53
4.1.1.3	Working of Rain Sensor	53
4.1.1.4	Receiving of Sensor Data at ESP8266	53
4.1.2	Results and Analysis of Crop Diagnosis	55
4.1.2.1	Results with Default Conditions	55
4.1.2.2	Experiments on Epochs	58

4.1.2.3	Experiments on Batch Size	59
4.1.3	Results and Analysis of Mobile Application Development	63
4.2	Comparative Study	68
4.3	Discussions	68
4.4	Conclusions	69
4.5	Scope for Future Work	70

BIBLIOGRAPHY 70

A	Transmitter and Receiver Arduino Sketches	80
A.1	Arduino Sketch of Transmitter Side	80
A.2	Arduino Sketch of Receiver Side	84
B	Android Studio Codes	89
B.1	Back end code of Application login page	89
B.2	Back end code of Registration page	91
B.3	Back end code of Navigating through app	93
B.4	Back end code of Weather page	94
B.5	Back end code of Monitoring page	97
B.6	Back end code of Disease Diagnosis page	101

LIST OF FIGURES

3.1	Soil Moisture Sensor	20
3.2	DHT-11 Sensor	21
3.3	Rain Sensor	21
3.4	PIR Sensor	22
3.5	Submersible Water Pump	22
3.6	Relay	23
3.7	Speaker	23
3.8	Light Emitting Diode	24
3.9	9v Battery	24
3.10	Jumper Wires	25
3.11	Arduino Uno	25
3.12	LoRa Module	26
3.13	ESP8266 Module	26
3.14	Arduino IDE	27
3.15	Board selection	28
3.16	Port selection	29
3.17	Libraries Used	29
3.18	Serial Monitor	29
3.19	LoRa Architecture	30
3.20	MQTT Pub-Sub Model	31
3.21	Crop Monitoring Overall Architecture	33
3.22	Pin Configuration of LoRa SX1278	36
3.24	Receiver Schematic Diagram	36
3.23	Transmitter Schematic Diagram	37
3.25	Flow chart of the Crop Diagnosis system.	40
3.26	Sample tomato leaf images from the dataset.	41
3.27	The architecture of the proposed system.	43
3.28	Architecture of VGG19 model.	43
3.29	Architecture of ResNet model.	44
3.30	Architecture of the InceptionV3 model.	44
3.31	Architecture of the InceptionResNetV2 model.	45
3.32	Architecture of the Xception model.	45
3.33	Architecture of the MobileNetV2 model.	46
3.34	Architecture of the DenseNet201 model.	46
3.35	Android Architecture	49

List of Figures

3.36	Android Studio IDE	50
3.37	Flowchart of Android Application	51
4.1	Serial Monitor of Soil Moisture Sensor	52
4.2	Serial Monitor of DHT-11 Sensor	53
4.3	Serial Monitor of Rain Sensor	54
4.4	Serial Monitor of ESP8266 receiving sensor data	54
4.5	Accuracy of all Models	56
4.6	Loss of all Models	56
4.7	Precision of all Models	56
4.8	Recall of all Models	57
4.9	Area under curve of all Models	57
4.10	Accuracy plot of all CNN models with 50 epochs	58
4.11	Accuracy plot of MobileNetV2 model	61
4.12	Loss plot of MobileNetV2 model	61
4.13	Detection of different diseases in tomato leaf using MobileNetV2 model	62
4.14	Login Page of the Mobile Application	63
4.15	Home Page of the Mobile Application	64
4.16	Weather Page	65
4.17	Crop Monitoring Page	66
4.18	Crop Diagnosis Page	67

LIST OF TABLES

2.1	Major Diseases affecting Tomato leaves and their symptoms.	7
2.2	Summary of the recent works in plant disease detection using CNN. . .	14
3.1	Soil Moisture Sensor Interaction with Arduino Uno	32
3.2	DHT-11 Sensor Interaction with Arduino Uno	34
3.3	Rain Sensor Interaction with Arduino Uno	34
3.4	PIR Sensor Interaction with Arduino Uno	34
3.5	LoRa Transmitter Interaction with Arduino Uno	35
3.6	LoRa Receiver Interaction with ESP8266	38
3.7	Relay Interaction with ESP8266	38
3.8	Description of tomato leaf dataset	41
4.1	Comparison of performance of all models under default conditions. . .	57
4.2	Comparison of performance of models with different number of epochs. .	59
4.3	Comparison of performance of all models with different batch sizes. . .	60

GLOSSARY

Item	Description
Android	Android is an operating system developed by Google Inc for mobile devices.
Bluetooth	It is type of wireless communication technology used primarily in short-range.
CNN	Convolutional Neural Network is a type of machine learning algorithm used to train a model.
DHT	Digital Humidity and Temperature is a sensor used to measure temperature and humidity
Embedded System	It is a computer hardware system with software designed for a particular task
GSM	Global System for Mobile Communication is a standard developed by 2G cellular networks.
IDE	Integrated Development Environment is a software that provides facilities for developers to develop software applications.
Image Processing	It is a method used to do specific operations on an image.
IoT	Internet of Things is a technology used to connect smart devices.
LoRa	Long Range Radio is type of wireless communication technology specifically used for long range operations.
MQTT	MQ Telemetry Transport is a messaging protocol that transports messages through devices.
PIR	Passive Infrared sensor is a motion sensor that detect motion.
Wi-Fi	Wi-Fi is a type of wireless communication standard used to connect two or more devices.
WSS	Wireless Sensor Networks is a type of sensor networks communicated with wireless technologies.
ZigBee	It is an IEEE 802.15.4-based wireless communication standard used to connect devices.

Chapter 1

INTRODUCTION

India is the world's largest and one of the oldest agriculture-dependent countries. More than half of the Indian population is involved in agriculture directly and indirectly. Even with such a heavy labor force, agriculture GDP contribution is low. With the agricultural land shrinking exponentially, it is high time India employed productivity-improving strategies. One of the significant productivity hindrances is improper cultivation methods and poor crop health. Thus, crop monitoring and disease identification are crucial to increasing the crop's yield.

1.1 Problem Formulation

Agriculture is a crucial occupation employing a multitude of people over the years. With such a high percentage at stake, it becomes crucial to properly apply technology and scientific methods in farming to increase the output. With the ever-growing population, there is continuous pressure on the agricultural sector to increase production and ensure efficient delivery to the customers without compromising the quality of the produce. Under such circumstances, applying technology to farming becomes important and assures more returns on less investment. However, the current application of technology is very bleak. Through various sources on the internet, it can be seen that only few countries have adopted technological advancements in cultivation and farming. This may be attributed to farmer's lack of knowledge of technology and their non-confidence in the developed models. This calls for proper research and analysis on the development of technological models that could be used in agriculture to enhance total produce.

1.2 Problem Identification

For many years, the IoT has been an area finding its application in the agricultural sector. IoT and image processing combined with machine learning is a powerful tool to assist farmers in increasing their yields with limited resources. IoT helps in the proper monitoring of the crop, which increases the yield with a decrease in the diseases. In case of damage to crop health, the combination of image processing and machine learning helps identify diseases in plants. Using these technologies will help achieve the objective of increased production of that particular crop and reduced usage of pesticides and other medicines that reduce the disease since it has been identified at an early stage.

1.3 Problem Statement & Objectives

1.3.1 Problem Statement

Proper methods of crop monitoring and early identification of diseases in crops are essential in increasing crop productivity and saving the hard work of the farmers. This project aims to achieve crop monitoring using IoT. Sensors like humidity, temperature, moisture, rain, and actuators like motor pumps and speakers should be used for crop monitoring. Crop diagnosis should be made using Convolutional Neural Networks and Image Processing. A dataset of crop leaves from an open-source should be collected. The images should be pre-processed as per the requirements using image processing techniques. The pre-processed images should be used to train a CNN model to identify disease on the new diseased leaf images. The sensor data and trained CNN model should be integrated into a single mobile application.

1.3.2 Objectives

- To collect and store the sensor data on the cloud.
- To develop a CNN model that can identify the disease when a new diseased leaf image is provided.
- To integrate the sensor data and trained CNN model into a single mobile application.

1.4 Limitations

In this project, there should be two internet active regions, one at the sensor data transmitter and another at the mobile user. This might be a major concern in remote areas. One more limitation of this project is the cost. In case of large coverage area, the number of sensors and data transmitters will be increased which directly increase the cost of the project unless there is a mass production of sensors or any subsidy from the government or NGO.

Chapter 2

LITERATURE SURVEY AND REVIEW

Since we are doing a research project closely associated with agriculture, we started the literature survey in this field. In the first section of the literature review, we have discussed the impact of agriculture on the country's economy and its effect on the lives of people. Later, we surveyed various crops grown widely in different regions of India. Then, we shortlisted the crops that had the most impact on the economy and people's livelihood. Later, we studied the factors that affect the productivity of the crop, like the quality of the seeds, weather conditions, availability of water, nutrients present in the soil, diseases, and pests. Subsequently, we researched the techniques that increase productivity. We studied technological use in implementing these techniques. Later, we found what diseases affect the crop and where the symptoms will reflect, whether on leaves, roots, or stems.

The second section of the literature review involves crop monitoring. We researched what significant factors can be monitored using the technology. We studied the basics and working of the IoT and embedded systems. We gathered data about different IoT devices like Arduino and Raspberry Pi and various versions. Later, we studied the Arduino environment and C++, which is required for programming the Arduino. We learned to design, build, and test an Arduino embedded system. Later, we researched different sensors available in the market based on our requirements. We studied the sensor's pin diagrams and controllers. Subsequently, we studied each sensor's compatibility with the Arduino. We learned how to interface each of these sensors with the Arduino. We gathered information on different actuators that can be employed in our

project. We studied how to stop the animal intrusion onto the crop with the help of sensors and actuators.

Later, we researched different Wireless Sensor Networks (WSS) to transmit the Arduino data. We studied ZigBee, WiFi, and LoRa. We studied these technologies working, implementation, advantages, and limitations. Later, we studied how to establish a connection between Arduino and LoRa transmitter. We studied how to receive the data with the LoRa receiver and send the received data to the cloud. We learned about various protocols for establishing and working on this communication. We researched different WiFi modules available in the market for the above-said purpose. Lastly, we gathered data about different cloud services available and how to use them efficiently.

The third section of our literature survey consists of the details of the crop diagnosis. In this section, we studied numerous techniques for identifying the disease of the crop. We studied the applications and limitations of satellite imagery, Unmanned Aerial Vehicles (UAVs), spectral cameras, and image processing. We researched newer technologies like machine learning, neural networks, and various types. Later, we studied how to create and use Convolutional Neural Networks (CNNs) and various techniques involved in them. In order to create CNNs, we studied Python programming language and gathered data about different platforms available to develop the code, like Google Colaboratory and Jupyter notebook. We studied various python libraries like Tensorflow, Keras, and NumPy. We learned different techniques to train the model and increase the model performance.

The last section comprises mobile application development. We learned how to build an application from scratch. We studied how to use Android Studio and its different libraries. We learned about creating login pages and databases. We studied creating multiple pages in an application and how to use firebase to show the sensor data. We learned about integrating the CNN model into a mobile application. We have referred to various books, research papers, online articles, MOOC courses, and videos. Further details on the same are mentioned in sections 2.1 and 2.2 of the literature survey and review.

2.1 Literature Collection & Segregation

As per the data available through the internet sources, it can be seen that agriculture has an employment percentage of 26.756 of the entire population as surveyed by the World Bank [1] which is an enormous number. Agriculture has a tremendous influence on the economy of any country since a survey by the World Bank in 2018 [2] claims that it accounts for nearly 4 percentage of GDP (Gross Domestic Product) of the world. There has been research [3] carried out on the impact of agriculture in developing countries and it is evident that roughly around 50% to 90% of the total population in small countries depend upon agriculture as their primary source of income. With such a high percentage at stake, it becomes very crucial for the proper application of technology and scientific methods in farming to increase the output.

In India, there are hundreds of different crops cultivated like fruits, vegetables and flowers. Tomatoes are one of the most widely grown crop with about 786,000 hectares of land under cultivation in the year 2017-18 [4]. Moreover tomatoes cultivation and monitoring resembles most of the other major crops. So, for our project work, we considered tomato crop as our reference and crop of application. There are three categories of factors affecting the crop production namely technological, environmental and biological factors. Technological factors involve the cultivation practices, quality of seed bought, and others. These factors entirely depends on the farmer and with the help of agricultural officers these can be done right. Environmental factors include soil properties like nutrients and pH of the soil and climatic properties like temperature and humidity. Lastly, biological factors involve diseases and pests [5]. Except for the technological factors, the other two factors can be minimized with the help of technology.

Tomatoes are one of the most widely grown food crops around the world. It ranks fourth among the most cultivated vegetables in the world. The leaf is the main source for most tomatoes' diseases. The leaves of healthy tomato plants are green and evenly coloured. During the cultivation process, tomato leaves are exposed to many of the diseases. Table 2.1 lists some of the major diseases affecting tomato leaves and their symptoms [6].

The IoT has been a revolutionary technology in recent years. It found its way into a plethora of applications. It is a technology that puts together intelligent devices, sensors, and intelligent systems. Much research has been going on in IoT's application areas. It can be effectively used to gather data from the sensors from anywhere around

TABLE 2.1: Major Diseases affecting Tomato leaves and their symptoms.

Disease name	Leaf symptoms
Bacterial spot	Initially appears as yellow-green circular areas, later turns to brownish-red.
Early blight	Round brown spots are found with concentric rings. Yellow tissues are often found around these rings.
Late blight	A water-soaked area appears and rapidly enlarges to form purple-brown, oily-appearing blotches.
Leaf mold	Irregular yellow or green area appears.
Septoria leaf spot	Round spots, marginal brown, chlorotic yellow, appear.
Two-spotted spider mite	Whitening or yellowing of leaves is found.
Target spot	Begins as small dark lesions which enlarge to form light brown lesions with concentric pattern and a yellow halo around it in the transplants.
Yellow leaf curl disease	Develop strong crumpling, small and curl upward, yellowing, shortened internodes.
Mosaic virus	Develop yellow or green, slightly shrinking.

the world [7]. Embedded systems are widely popular and extensively employed. IoT-based embedded systems can be built using the Arduino and Raspberry Pi platforms. Devices that can control the physical world can be built with the help of the Raspberry Pi platform, and Arduino environment [8]. Raspberry Pi is a compelling device, and it can single-handily replace a computer. However, it is expensive, and such a powerful device can not be used effectively in our project. Arduino Uno is a device that perfectly suits our needs. It is cheap and at the same time serves our purpose. The Arduino Integrated Development Environment is programmed in C++, and all the sensors can be connected with the help of this IDE [9]. The sensors required for our project are the Soil Moisture sensor [10], temperature and humidity sensor [11], and motion sensor

[12]. Each of these sensors should be compatible with the Arduino. Sensors are integrated with the Arduino using the pin diagrams of the sensor, and Arduino [13].

The Arduino needs to transmit the collected sensor data to the cloud or the user. It can be done in two ways, namely: Wired Sensor Networks and Wireless Sensor Networks (WSS) [14]. Though wired sensor networks are relatively fast and more reliable than its counterpart, the infrastructure needed for the networks is expensive and installing the network is difficult in remote areas. There are a plethora of options available in the WSS that can be employed in agriculture applications, namely: Bluetooth [15], ZigBee [16], and LoRa [17, 18]. With the help of any of these technologies, as mentioned earlier, the data can be transmitted to the receiver. A GSM module [19] or a Wi-Fi module [20] can be employed to send the data to the cloud/user. Message Queuing Telemetry Transport (MQTT) protocol is used in communicating between the cloud, and the Wi-Fi module [21].

Disease Identification and monitoring is an arduous task, and if the cultivation is done in large areas, it becomes impossible to control the damage and prone to errors. Notably, many farmers do not know the different kinds of diseases affecting the plants and what remedies to be employed. Many technologies have emerged to help the farmers monitor the field to counter this situation. Geographic Information Systems (GIS) were one of the most used technologies in monitoring the field. With the help of GIS, farmers can assess the condition of the crops, and they also have data regarding the fertility levels of the soil [22]. Unmanned Aerial Vehicles (UAVs) have been the most talked-about technology for crop monitoring in the past decade. Using several high-tech types of equipment like near-infrared cameras, thermal sensors, and other sensors [23]. The data obtained from UAVs is reliable and gives a detailed analysis of the crop obtained from different sensors. To achieve high precision results, Satellite Imaging is used. This technology can be employed to study vast areas of cultivation with the help of high-resolution images obtained from satellites [24]. Apart from GIS, UAVs, and Satellite Imaging, many other technologies are being employed to assist the farmers, such as the IoT and farming software with the help of neural networks. Though all these technologies have been producing excellent results, their usage in farmlands is limited, especially in countries like India, where most farmers have no idea how to use and study these technologies. UAVs and Satellite Imaging are very expensive and can not be afforded by Indian farmers. Also, many of these technologies need an active network connection to the database, which is not feasible in remote areas. So many of these technologies are only suitable for research work but not in practical scenarios.

There is considerable research available on the using machine learning [25], deep learning [26, 27] and neural networks[28] in agriculture for disease detection. CNN have been employed to detect and identify diseases in tomato plants. The CNN models have the advantage of reduced complexity in the identification process through sufficient training in the presence of a large number of images in the dataset, which ensures the removal of complex image pre-processing techniques that include feature extraction, segmentation, and classification [29]. In recent years, research has been carried out on CNN, and several architectures have been developed for classification and object detection. Few such architectures available include ResNet, AlexNet, and GoogleNet, to name a few. In this proposed method, data acquired is processed and augmented before being classified as the proper disease based on the dataset available. The proposed work has also considered CNN architectures VGG19, ResNet152V2, InceptionV3, InceptionResNet152V2, Xception, MobileNetV2, and DenseNet201 and compared them for the same task of early identification of diseases in tomato plants.

2.2 Critical Review of Literature

Sachin Kumar et al. [7] suggest that IoT is a revolutionary approach to technology development. The authors described the IoT architecture, which defines the function of intelligent systems. The architecture consists of 5 layers. The bottom layer is called Perception, which contains the devices that interact with the physical world, such as sensors, barcodes, R.F. tags, Etc. The information collected by these physical devices is sent to its upper layer, called the Network layer. This layer is responsible for transmitting the data using wired or wireless communication technologies like optical fibers, ZigBee, Wi-Fi, Etc. The middleware layer is the next layer where the received data is processed. It is responsible for decision-making based on the computation. The next layer uses the results and decisions, called the application layer. The last layer is the business layer, where the data is visualized for future purposes. This entire IoT architecture is dynamic and can be changed as per the clients' requirements.

Y. Kim et al. [15] designed an efficient water management system using a distributed wireless sensor network. The authors chose a Bluetooth module for the wireless data communication between the sensors and the base station. This Bluetooth module uses the Frequency Hopping Spread Spectrum technique in the 2.4GHz band and is suitable in low power, cost, and range scenarios. The power consumed by the Bluetooth module

was 23.8Wh. However, when the Bluetooth module lost power, the communication was offline even after the power was back. T. Kalaivani et al. [16] discussed the role of ZigBee in agriculture for intelligent farming. The Zigbee technology has three devices. The Zigbee Co-ordinator is responsible for initializing, maintaining, and controlling the sensor network. It is the MAC coordinator. The second device is the ZigBee router which does the multi-hop message routing. Both ZigBee Coordinator and ZigBee router should always be plugged into a power supply to monitor the WSS. The last device is the ZigBee End Device which senses or actuates. ZigBee is also used for low-power and data rate applications.

S. Tapashetti and K. Shobha [17] proposed a method to find the nutrients in the soil and other parameters using LoRa technology and Cloud Computing. Lora is a long-range and low-power wireless communication technology. The authors suggest that the LoRa is the most efficient way of communication as it consumes low power yet is secure and long-range. Lora uses the Spread Spectrum modulation technique, which effectively tackles eavesdropping. Spread-spectrum modulation also makes the LoRa insusceptible to noise. D.I. Sacaleanu et al. [18] present the comparison of the wireless sensor network's communication technologies in energy consumption. The authors compared the LoRa with ZigBee and Enhanced ShockBurst. This paper showed that, though LoRa is effective for long-range, there is a slight increase in the time frame for transmitting the data. The increase in time frame directly increased the energy consumption. To counter this, the authors proposed a technique of data compression. This technique has significantly decreased the time frame and also the energy consumed. In LoRa communication, the average current in transmission was 154mA, and the current on a 1190ms time frame was 183mA. This result shows that there is a 31 percent energy consumption improvement.

Sushanth. G and Sujatha. S [19] developed an intelligent field monitoring and irrigation system based on IoT. The authors used a Global System for Mobile Communication (GSM) module. They used this module to send an SMS to the user/farmer about the status of the crop from the data gathered through the sensors. Nayyar. A and Puri. V [20] used the Wi-Fi module instead of the GSM module. The authors developed an agricultural stick for live moisture and temperature reading. Also, instead of a traditional battery power supply, they used solar technology.

Mingyuan Xin and Yong Wang [30] compared Convolution Neural Network, Support Vector Machine (SVM), k-nearest neighbor, naive Bayes, random forest, decision tree,

and gradient elevation decision tree machine learning models on standard databases of CIFAR-10 and MNIST. They concluded that deep CNN has incomparable advantages in image classification applications. Koay K. Leong and Lim L. Tze[31] compared ResNet-50 CNN with Gray-Level Co-Occurrence Matrix (GLCM) for plant leaf disease detection. Using an SVM classifier, CNN, with an accuracy of 96.63%, performed better than GLCM in feature extraction. CNN trained with millions of images, and GLCM can only extract eight features in each image. Thus, it can be concluded that CNN is advantageous over most other machine learning methods available.

Using CNN on plant disease detection with limited data will often result in bad results as CNN demands enormous amounts of data. To counter this issue, Sinno et al. [32] used knowledge transfer which is popularly called transfer learning. They reviewed and classified transfer learning progress on various clustering, classification, and regression problems. Transfer learning is classified into transductive and inductive transfer learning based on source and target domains and tasks. Chuanqi Tan et al. [33] defined categorized deep transfer learning into four types, Instances-based, Mapping-based, Network-based, and Adversarial-based. They reviewed works undertaken in each of these types and gave a standardized elucidation and a sketch map for all four types. Ahmed Afifi et al. [34] tried different approaches like transfer learning, Deep Adversarial Metric learning, and triplet network using ResNet 18, 34, and 50 on the datasets of Plant Village [35] and coffee leaf dataset [36]. They concluded that transfer learning using a good baseline model could achieve high accuracy even with limited data.

Many pre-trained models can be used for various datasets using transfer learning. Choosing a suitable pre-trained model is crucial to achieving good performance. Karen et al. [37] proposed architecture with 3x3 convolution filters and pushed the layer depth to 16-19 weight. They tested the model on the imagenet and called the model VGG. Kaiming et al. [38] analyzed the basic principle of residual building blocks using identity mappings used in ResNet models. They discussed the importance of identity skip connections, and various experiments were conducted on activation functions. They compared the model performance on databases like CIFAR-10/100 and ImageNet. Christian Szegedy et al. [39] discussed Inception architecture, its implementation in computer vision, and its advantage over the VGG model. They showed how InceptionV3 outperformed many other networks with great top-5 and top-1 error rates. C. Szegedy et al. [40] proposed a hybrid architecture using Inception and ResNet and called it as Inception-ResNet. They discussed how this model is created and compared its performance with InceptionV3, InceptionV4, and ResNet151. Francois et al. [41]

proposed a new model called Xception inspired by Inception. On ImageNet, even with the same number of parameters, Xception outperformed InceptionV3. It achieved the best top-1 and top-5 accuracy compared with VGG16, ResNet152, and InceptionV3. Mark Sandler [42] described MobileNetV2, a mobile model, and it is tested on ImageNet and COCO database. It is compared with other mobile networks like SSD300, SSD512, and YOLOv2 and achieved better accuracy. Gao Huang [43] proposed a new network called Dense Convolutional Network (DenseNet). It connects every layer in a feed-forward pattern. This model is evaluated on four datasets, namely CIFAR-10, CIFAR-100, SVHN, and ImageNet. DenseNet achieved high performance with less computation.

CNN has been widely used to detect plant diseases and achieved excellent results. Satwinder Kaur et al. [44] proposed a deep learning 22 layer google net model for plant disease classification of 14 different crops using a database of 54,306 images, and it achieved an accuracy of 97.82%. Sharada et al. [45] using the same database used in Ref. [44], achieved an accuracy of 99.35% using GoogleNet. They gave a detailed comparison between the performance of GoogleNet and AlexNet on the database using different training mechanisms, loss type, train-test set division, and dataset types (color, grayscale and segmented). Konstantinos [46] achieved a success rate of 99.53% using a database of 87,848 images that have 58 different classes belonging to 25 different plants. He used AlexNet, AlexNetOWTBn, GoogleNet, overfeat, and VGG, which later performed accurately. In Ref. [47], Too E.C. et al. evaluated DenseNet121, InceptionV4, VGG16, and three ResNet architectures with a different number of layers. DenseNet constantly improved accuracy with the increase of epochs without overfitting and achieved an accuracy of 99.75%, beating all other models.

Peng Jiang et al. [48] used AlexNet, GoogleNet, VGGNet-16, VGG-INCEP, and four types of ResNet on a dataset of 2029 authentic field images. It detected five common apple diseases, namely Alternaria leaf spot, Brown spot, Mosaic, Grey spot, and Rust. A new deep CNN model is developed using Inception and Rainbow concatenation based on this database. The model performed with 78.80% mAP on the database and with 23.13 FPS.Md. Aashiqui Islam et al. [49] analyzed VGG19, Xception, InceptionResNetV2, and ResNet101 models on a dataset of 984 images of 5 different classes obtained from the Kaggle and UCI machine learning repository. Out of 4, InceptionResNetV2 achieved top accuracy of 92.68% in detecting diseases of paddy leaf. Chand et al. [50] trained a system with an accuracy of 96.7% that can identify northern leaf blight disease in maize crops. They used a computational pipeline of CNN that can counter limited

data and irregularities in images used to train the model. Rahman et al. [51] developed a model that achieved 99.53% accuracy in detecting rice crop diseases and pests using VGG16. They also used InceptionV3, ResNet50, InceptionResNetV2, and Xception models, which achieved high accuracies but less than VGG16. Keke Zhang et al. [52] compared AlexNet, GoogleNet, and ResNet model's performance in identifying tomato leaf diseases. ResNet with stochastic gradient descent (SGD) and a batch size of 16 outperformed the other two models and achieved an accuracy of 96.51%, and with further fine-tuning techniques, accuracy reached 97.28%. In Ref. [53], Anandhakrishnan achieved a 99.45% precision in identifying ten different disease classes of tomato plant using Xception. They compared AlexNet, ResNet, LeNet, and VGG16 with Xception using accuracy, precision, specificity, and sensitivity. Elhassouny [54] used a model adapted from MobileNet that can recognize ten different tomato leaf diseases. It is trained on a dataset of 7716 images and is further implemented as an innovative mobile application. From Table 2.2, it can be observed that the most used models include ResNet, Inception, Xception, DenseNet, and MobileNet. However, no single paper compares all these models on the same dataset.

Mobile applications are being widely used by millions of people around the world. Masi et al. [55] presented a detailed framework for technology decision making for the development of mobile applications. They discussed technologies employed, programming languages used, platforms available, platform-specific software development kits and drivers involved. They even presented various problems faced and solutions to counter them. From entertainment, health, business, productivity and everything else are achieved and improved with the help of mobile applications. Oinas-Kukkonen and Kurkela [56] discussed extensively developing a successful mobile application. The authors talked about different scenarios, applications and key design principles in developing an application for mobile devices. Islam and Mazumder [57] presented different areas of applications of mobile applications. The authors even discussed the effect of mobile applications on society and also talked about the limitations of mobile applications. Sunitha and Elina [58] discussed the impact of mobile applications in the education sector. The authors presented different kinds of apps used in education and subject-specific mobile applications and their advantages. Ventola [59] presented about mobile devices and apps used in the field of healthcare. The author mentioned different types of applications, their need at the point of care and how health care professionals use them. Choe et al. [60] discussed a real-time mobile application that can identify and classify different parrot species using CNN. The authors discussed the implementation

of the mobile application, how the CNN model was built, feature extraction techniques employed and presented different experimental results.

TABLE 2.2: Summary of the recent works in plant disease detection using CNN.

Paper Title	Model/s Used	Dataset	Crop/s	Best Model	Accuracy
Plant Disease Classification using Deep Learning Google Net Model	GoogleNet (Inception)	Plant Village (54,306 images)	38 dis-eases in 14 crops	GoogleNet	97.82%
Using Deep Learning for Image-based Plant Disease Detection	AlexNet, GoogleNet	Plant Village (54,306 images)	38 dis-eases in 14 crops	GoogleNet	99.35%
Deep learning models for plant disease detection and diagnosis	AlexNet, AlexNet, tOWTBn, GoogleNet, Overfeat, VGG	Open Database (87,848 images)	58 dis-eases in 25 crops	VGG	99.48%
Comparative study of fine-tuning deep learning models for plant disease detection	VGG16, InceptionV4, ResNet50, ResNet101, ResNet152, DenseNet121	Plant Village (54,306 images)	38 dis-eases in 14 crops	DenseNet121	99.75%
Continued on next page					

Table 2.2 – *Continued*

Paper Title	Model/s Used	Dataset	Crop/s	Best Model	Accuracy
Real-Time De- tection of Apple Leaf Diseases Using Deep Learning Ap- proach Based on Improved Con- volution Neural Networks	AlexNet, GoogleNet, Incep- tionV3, ResNet50, ResNet101, ResNet34, ResNet18, ResNet16, VGG-Incep	Artificially Apple col- lected, A&F Uni- versity, China (2,029 images)		VGG- Incep	97.14%
An Automated Convolutional Neural Network Based Approach for Paddy Leaf Disease Detec- tion	VGG19, ResNet101, Xception, Inception- ResNetV2	Kaggle, UCI repos- itory (984 images)	Paddy	Inception- ResNetV2	92.68%
Automated Identification of Northern Leaf Blight-Infected Maize Plants from Field Imagery Using Deep Learning	The com- putational pipeline of CNNs	GxE Trail (1,796 images)	Maize	CNN	96.7%
Identification and Recognition of Rice Diseases and Pests Using Convolutional Neural Net- works	VGG16, In- ceptionV3, ResNet50, Inception- ResNetV2, Xception	Collected from paddy fields in Bangladesh (1,426 images)	Paddy	VGG16	99.53%

Continued on next page

Table 2.2 – *Continued*

Paper Title		Model/s Used	Dataset	Crop/s	Best Model	Accuracy
Can Deep Learning Identify Tomato Leaf Disease		AlexNet, GoogleNet, ResNet	Open repository (5,550 images)	Tomato	ResNet	97.28%
Identification of Tomato Leaf Disease Detection Pretrained Deep Convolutional Neural Network Models		AlexNet, Lenet, ResNet, VGG16, Xception	Plant Village (14,528 images)	Tomato	Xception	99.45%
Smart mobile application to recognize tomato leaf diseases using Convolutional Neural Networks		MobileNet adapted model	Plant Village (7,716 images)	Tomato	MobileNet adapted model	88.4%

Mobile applications are also being extensively used in the field of agriculture. Dehnen-Schmutz et al. [61] explored the farmer's usage of mobile phone technology in agriculture. The authors presented many surprising results that say a high percentage of farmers nearly 84% used mobile applications for farm management and many other farmers used it for real-time monitoring, data collection and experimental work. Johannes et al. [62] presented a novel image processing method for disease identification with the help of hot-spot detection and statistical inference methods using mobile devices. Toseef and Khan [63] developed a mobile application using a fuzzy inference system for crop diagnosis and it achieved an accuracy of 99 %.

Esgario et al. [64] presented an application that identifies pests and diseases using deep learning in coffee leaves. The authors discussed how the android application can detect and classify diseases caused by biotic agents in coffee leaves with an accuracy of 97 %. Petrellis [65] presented a mobile application that runs without a server and can diagnose citrus diseases. The application achieved an accuracy of more than 90 % in detecting citrus diseases. Rishiikeshwer et al. [66] built an application that can detect diseases in plants with an accuracy of 98% using CNN and image augmentation. The authors built a web application that requires a server to work. Though many researchers presented a mobile application that can identify diseases, to the best of our knowledge, there is no literature discuss a mobile application that can detect diseases as well as monitor the crop.

Chapter 3

METHODOLOGY

The main aim of this project is to create an end-to-end system for the farmers providing every technological solution to the problems they face. This project aims to achieve the monitoring of the crop using IoT. The significant factors affecting crop productivity such as humidity, temperature, the moisture content in the soil, rain, and animal intrusion should be monitored. Actuators should be used based on the processing of the sensor's data. In case of any disease attack, disease detection should be done effectively and efficiently. All these tasks should be available in a single mobile application for the farmer.

The entire project work can be divided into three sub-categories.

- Crop Monitoring
- Crop Diagnosis
- Mobile Application Development

3.1 Crop Monitoring

3.1.1 Methodology for the Crop Monitoring

Methodologies adopted for the entire study:

1. Literature survey on the recent technological developments in the crop monitoring.
2. Detailed study of the Arduino board.
3. Complete analysis of sensors and actuators.
4. Design and test bench setup.
5. Detailed study and analysis of LoRa technology.
6. Detailed study of Wi-Fi module.
7. Study of sensor data visualization.

3.1.2 Experimental and Analytical Work Completed in the Crop Monitoring

The work completed in the crop monitoring is all the sensors are integrated with the Arduino and the sensor's data is transmitted from the Arduino with the help of LoRa transmitter. The transmitted data is received by the LoRa receiver and the received data is sent to the NodeMCU which again sends the data to the cloud.

3.1.2.1 Components Description

The following hardware components are used for the crop monitoring:

- Soil Moisture Sensor
- DHT-11 Sensor
- Rain Sensor
- Motion Sensor (PIR)
- Submersible water pump
- Relay
- Speaker

- LED
- 9v Battery
- Jumper wires
- Arduino Uno
- LoRa Module
- Wi-Fi Module

1. Soil Moisture Sensor

Soil moisture sensor measures the water content available in the soil. It has two long vertical connecting probes. When these probes are inserted into the soil, the change in resistance depicts the amount of moisture present in the soil. The sensor comes with an LM393 comparator chip. It also consists of a red LED for power indication and a green LED that acts as a digital switching output indicator. The sensor operates at an operating voltage of 3.3V-5V [10]. The soil moisture sensor is shown in the figure 3.1.



FIGURE 3.1: Soil Moisture Sensor

2. DHT-11 Sensor

DHT-11 sensor is the most inexpensive and efficient way of measuring temperature and humidity. The sensor senses temperature with the help of thermistor and measures humidity with the help of capacitor. It has a 20% to 90% RH humidity range and can measure temperatures from 0 to 50 degrees Celcius. The accuracy of temperature and humidity change is 5% RH and 2 degrees, respectively. DHT-11 can measure relative temperature and humidity and also provides excellent long-term stability [11]. The DHT-11 sensor is shown in the figure 3.2.

3. Rain Sensor

The rain sensor is a switching sensor that is activated by the rainfall. It works on the principle of total internal reflection. Less light has reflected in the sensor when there is rain, thus sensing the rain. The sensor provides both digital and analog output. It comes with adjustable sensitivity, and there is an output LED indicator too. The rain sensor is shown in the figure 3.3.

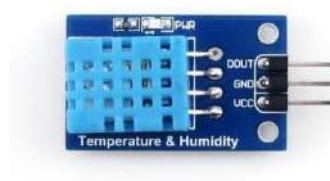


FIGURE 3.2: DHT-11 Sensor



FIGURE 3.3: Rain Sensor

4. Motion Sensor/ PIR Sensor

The motion sensor we used in our project is a Passive Infrared Sensor (PIR). It is an inexpensive and low-power sensor that senses any motion. As the name suggests, it works on the principle of infrared rays radiating from the objects in front of the sensor. The sensor should be connected to a 5V DC power supply [12]. The range of the PIR sensor used in our project is 10m. The figure 3.4 shows an image of PIR sensor.

5. Submersible water pump

A submersible water pump is used as an actuator in our project. When the moisture content detected by the soil moisture content is below a certain threshold, Arduino will automatically start the submersible water pump till the soil moisture sensor raises above the threshold. Submersible water pump used is shown in the figure 3.5.



FIGURE 3.4: PIR Sensor



FIGURE 3.5: Submersible Water Pump

6. Relay

In practical scenarios, a motor can not be directly connected to the power supply that is connected to the other components of the setup. So for safety reasons, a relay is used in between Arduino and submersible water pump. The relay used is shown in the figure 3.6.

7. Speaker

Speaker is used as an actuator in our project. When PIR sensor detects any motion, the speaker is actuated and makes noise in short bursts. If the PIR sensor detects no motion, then the speaker comes to initial state. The speaker used in our project is shown in the figure 3.7.



FIGURE 3.6: Relay

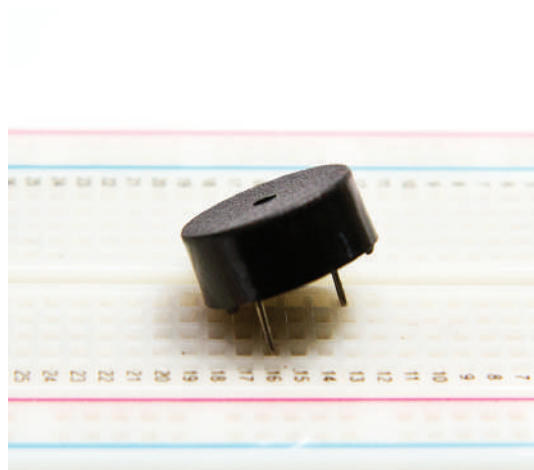


FIGURE 3.7: Speaker

8. Light Emitting Diode (LED)

LED is used as an actuator in our project. When PIR sensor detects any motion, the LED is glown and flashes light continuously. If the PIR sensor detects no motion, then the LED is turned OFF. The LED used in our project is shown in the figure 3.8.



FIGURE 3.8: Light Emitting Diode



FIGURE 3.9: 9v Battery

10. Jumper Wires

Jumper wires are used for connecting the components with the Arduino. Male to male, Female to Female and Male to Female wires were used in the project. The jumper wires used are shown in the figure 3.10.

11. Arduino Uno

Arduino Uno, also called Genuino Uno is a microcontroller board based on the ATmega328P. It has 6 analog inputs, 14 digital input/output pins, a USB connection, a power jack, a 16 MHz quartz crystal, an ICSP header and a reset button [67]. It can be programmed by connecting the board to a computer with an USB. It can be started with an adapter/battery. The Arduino Uno is shown in the figure 3.11.



FIGURE 3.10: Jumper Wires



FIGURE 3.11: Arduino Uno

12. LoRa Module

LoRa module used in this project is the LoRa SX1278 RA02. It works at 3.3V operating voltage and 433MHz operating frequency. It supports all microcontrollers that support SPI communication. The LoRa module supports various modulation techniques [68]. The same module can be used on the transmitter side and on the receiver side. The LoRa module used is shown in the figure 3.12

13. Wi-Fi Module

Wi-Fi module used in this project is the ESP8266 Wi-Fi module, also popularly called as NodeMCU. The NodeMCU is a System on Chip which allows any microcontroller to connect to a Wi-Fi network [69]. In our project, NodeMCU is connected with a LoRa receiver enabling the data received to be sent to the cloud. Also NodeMCU is extremely cost effective. The LoRa module used is shown in the figure 3.13

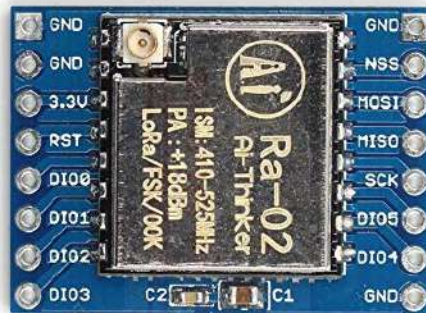


FIGURE 3.12: LoRa Module

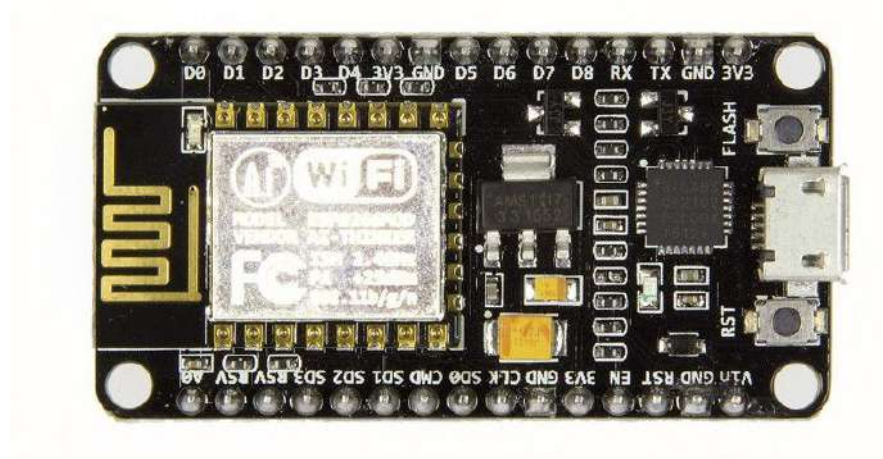


FIGURE 3.13: ESP8266 Module

3.1.2.2 Tools and Technologies

1. Arduino IDE

The Arduino Integrated Development Environment (IDE), simply called Arduino Software is used to program the Arduino board. "The Arduino IDE contains a text editor for writing programs, a message area, a text console, and a toolbar with buttons for implementing functions". The Arduino is connected to the Computer in order to transfer the code from Arduino IDE to the hardware board. Programs written using Arduino Integrated Development Environment are also called sketches. The sketches are saved with a .ino file extension. The text editor works just like any other programming text editor with options of cut/paste and generic edit options. The feedback and errors are shown in the message area when the sketch is saved or executed. "The console displays text output by the Arduino Software (IDE), including complete error messages and other information". The Arduino board configuration and serial port is shown on the bottom right hand corner of the IDE. "The toolbar buttons allow you to verify and upload programs, create, open, and save sketches, and open the serial monitor"[9]. The Arduino Integrated Development Environment is shown in the figure 3.14.



FIGURE 3.14: Arduino IDE

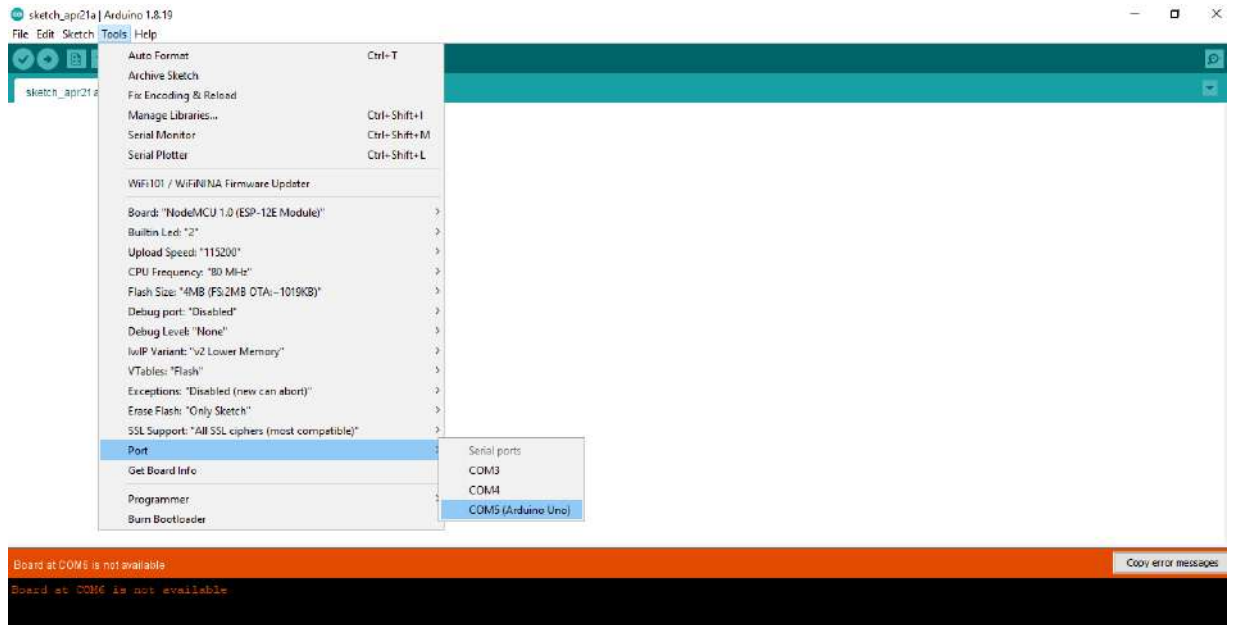


FIGURE 3.16: Port selection



FIGURE 3.17: Libraries Used

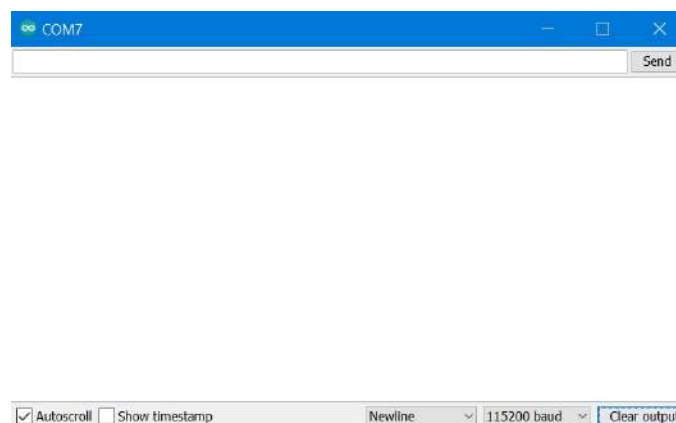


FIGURE 3.18: Serial Monitor

2. LoRa Technology

LoRa is termed as ‘Long Range’ low power remote standard planned for giving a cell style low information rate for the communication network. LoRa is developed by Semtech and it is designed for long-range communication. It uses Spread Spectrum Modulation (SSM) from CSS technology. LoRa is one of the high network coverage, a powerless remote stage that has transformed into development for the Internet of Things (IoT) framework around the world. It enables keen IoT operation that comprehend apparently the most prominent troubles standing up to earth: essentials the officials, common decrease in asset, control of contamination, foundation influence, fiasco nullifying action, and the sky is limited from here. Some of the key features of LoRa technology are it has a battery life of excess of 10 years, has a range of about 10-15 km and millions of nodes can be connected to this. The figure 3.19 represents the LoRa architecture.

LoRa technology utilizes star topology which helps in expanding the battery life for long-range availability. Some of the critical components of this framework are Endhubs, lora portals, and Network servers. The remotely set hub, for example, sensors or application sense and controls the system. The information transmitted by the hubs is sent to the portal and it sends the transmitted flag to the system server. At that point, the server sends the parcel to the particular application.

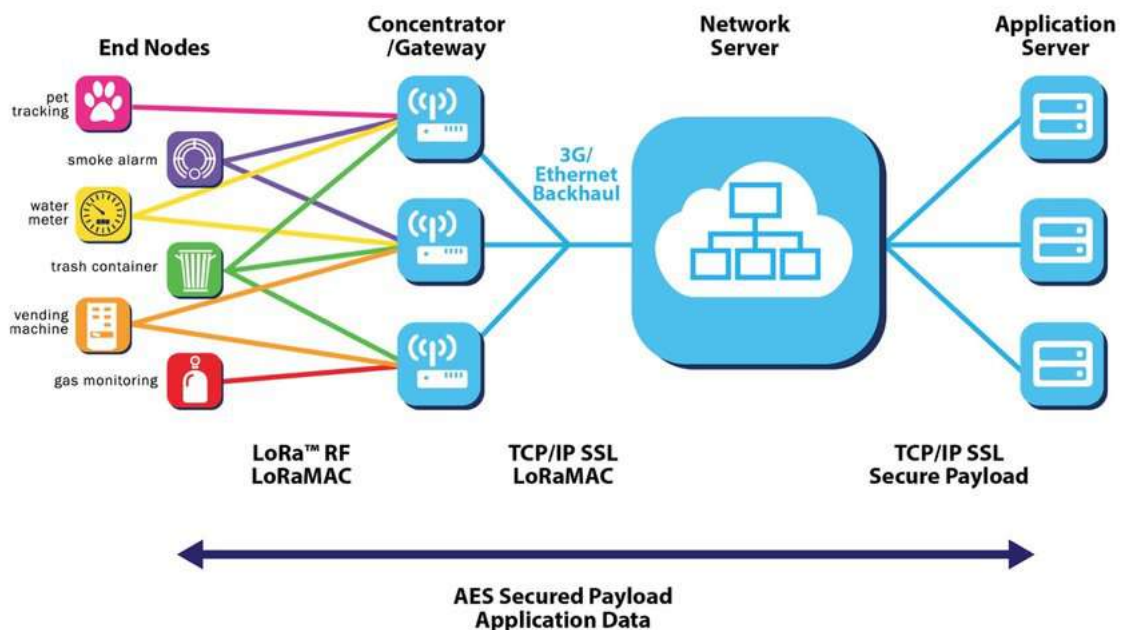


FIGURE 3.19: LoRa Architecture

LoRa Ra-02 uses the wireless standard of 433MHz and has a frequency range of 420-450 MHz. It uses SPI or GPIO ports and has an operating range of 1.8-3.3V. It is mounted with a spring antenna which works on CSS. It has the capability to automatically detect RF signals and has an RSSI wave range of 127dB. It bolsters FSK, MSK, GFSK, and GMSK adjustment mode. It utilizes half-duplex SPI communication. It has a programmable bit rate up to 300kbps [71].

3. MQTT Protocol

An MQTT which works on the topmost layer of the TCP/IP protocol is a pub-sub establishing the messaging protocol. This pub-sub arrangement depends on the messaging broker. It is essential, lightweight, open, and arranged in order to be anything other than difficult to execute. MQTT hubs convey in a one-to-many mapping model, where a message sent by one customer is conveyed to numerous customers through specific topic names. Data is stored out in a chain of command of topics.

An MQTT framework comprises customers communicating with a server regularly called a "Broker". A customer might be either a distributor of data or an endurer. Every customer can interface with the broker. The negligible MQTT control message can be as mere as two bytes of information. A control message can convey about 256 megabytes of information if necessary. The protocol is designed for devices that work on lower bandwidth.

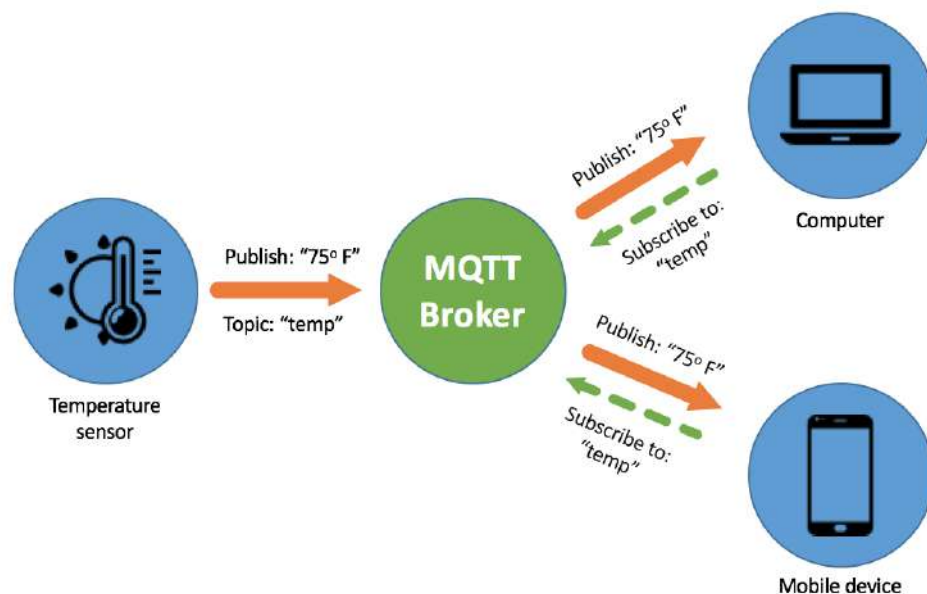


FIGURE 3.20: MQTT Pub-Sub Model

The fundamental elements of the Pub-Sub model represented in Figure 3.20, consist of Publisher/Subscriber, Messages, Topics, and Broker. In a pub-sub framework, a client can distribute the message on a topic, or it can be endorsed to a specific topic to accept the messages. Clients don't have addresses like in email frameworks, and messages are not sent to customers. Messages are distributed to an intermediary on the topic. The task of an MQTT intermediary is to channel messages dependent on the topic and after that convey them to endorsers. A client will get these messages by endorsing that topic on a similar broker. There is no link between the publisher and the subscriber. Hence both of the applicants to the broker can distribute and endorse the topic. MQTT intermediaries usually do not store the messages [21].

3.1.3 Modeling, Analysis and Design of Crop Monitoring

The work completed in the crop monitoring is appropriate sensors were collected for soil moisture, rain, motion, temperature and humidity measurements. Then the sensors were integrated with the Arduino and integrate all these sensors with the Arduino Board. Then the Arduino board transmits the sensor data to the LoRa transmitter. The LoRa transmitter transmits the information to the LoRa receiver which in turn sends the data to the cloud through Wi-Fi module. The overall architecture of crop monitoring is shown in the fig 3.21.

3.1.3.1 Soil Moisture Sensor Configuration

Soil Moisture sensor has four different pins namely, VCC pin, GND pin, Analog Pin and Digital pin. The Table 3.1 shows the pin connections of Soil Moisture sensor with Arduino Uno.

TABLE 3.1: Soil Moisture Sensor Interaction with Arduino Uno

Soil Moisture Sensor	Arduino Uno
Vcc	3.3V/Vin
Gnd	Gnd
Analog (A0)	A0
Digital	-

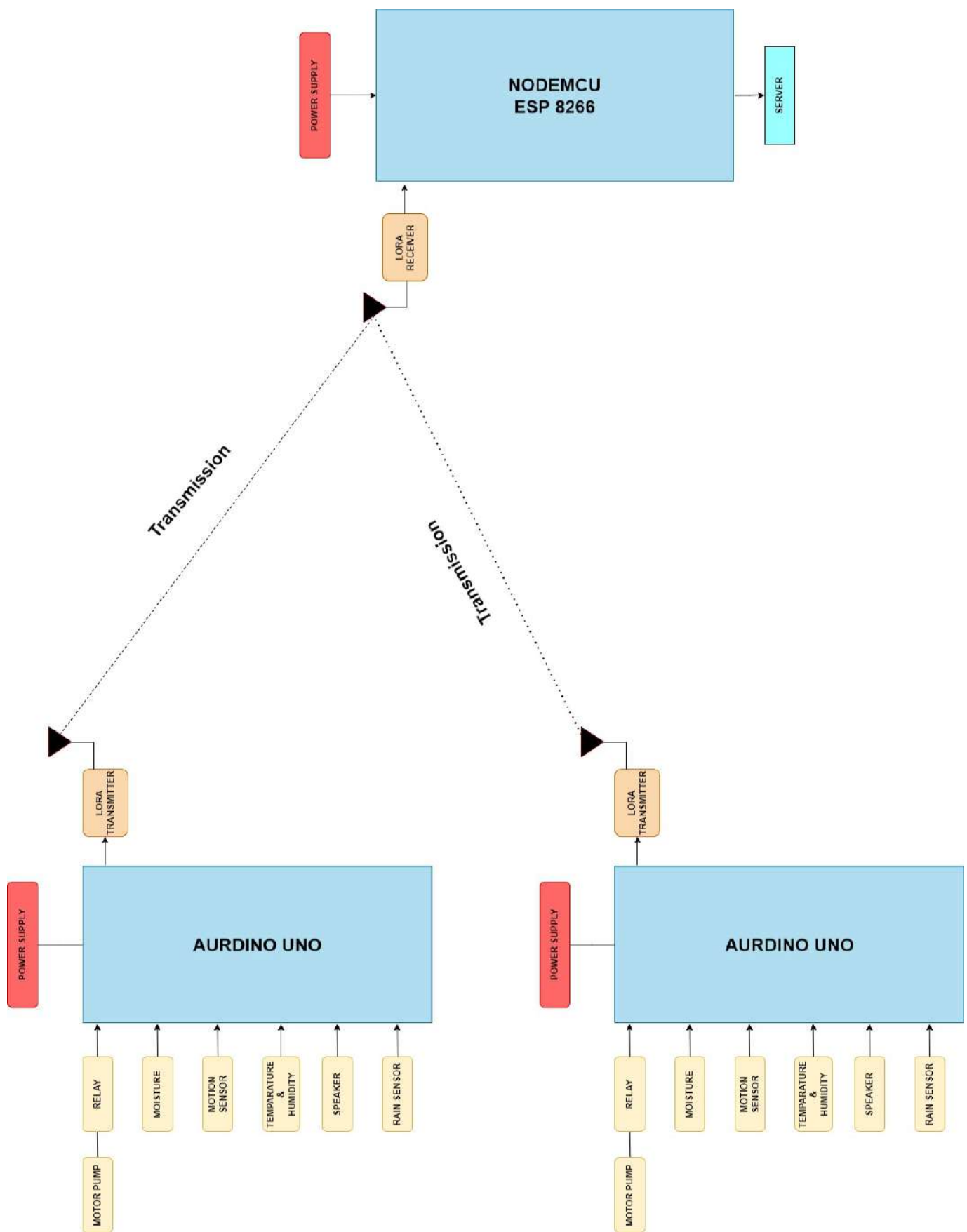


FIGURE 3.21: Crop Monitoring Overall Architecture

3.1.3.2 DHT-11 Sensor Configuration

DHT sensor has three different pins namely, VCC pin, GND pin and DATA pin. The Table 3.2 shows the pin connections of DHT-11 sensor with Arduino Uno.

TABLE 3.2: DHT-11 Sensor Interaction with Arduino Uno

DHT-11 Sensor	Arduino Uno
Vcc	3.3V/Vin
Gnd	Gnd
DATA	D5

3.1.3.3 Rain Sensor Configuration

Rain sensor has four different pins namely, VCC pin, GND pin, Analog Pin and Digital pin. The Table 3.3 shows the pin connections of Rain sensor with Arduino Uno.

TABLE 3.3: Rain Sensor Interaction with Arduino Uno

Rain Sensor	Arduino Uno
Vcc	3.3V/Vin
Gnd	Gnd
Analog (A0)	A1
Digital (D0)	D4

3.1.3.4 PIR Sensor Configuration

PIR sensor has three different pins namely, VCC pin, GND pin, and OUT pin. The Table 3.4 shows the pin connections of PIR sensor with Arduino Uno.

TABLE 3.4: PIR Sensor Interaction with Arduino Uno

PIR Sensor	Arduino Uno
Vcc	3.3V/Vin
Gnd	Gnd
OUT	D3

3.1.3.5 Speaker and LED Configuration

Speaker and LED are connected to pin 6 of the Arduino Uno.

3.1.3.6 LoRa Transmitter Configuration

LoRa SX1278 has in total of 16 pins with 8 pins on one side. DIO0 to DIO5 are used by GPIO pins. Four pins are used by Ground pins. The figure 3.22 shows the pin configuration of LoRa SX1278. The Table 3.5 shows the pin connections of LoRa Transmitter with Arduino Uno.

TABLE 3.5: LoRa Transmitter Interaction with Arduino Uno

LoRa Transmitter	Arduino Uno
3.3V	3.3V/Vin
Gnd	Gnd
NSS	D10
DIO0	D2
SCK	D13
MISO	D12
MOSI	D11
RST	D9

3.1.3.7 Schematic Diagrams

The figures 3.23 and 3.24 shows the schematic diagrams of the Transmitter and Receiver side respectively.

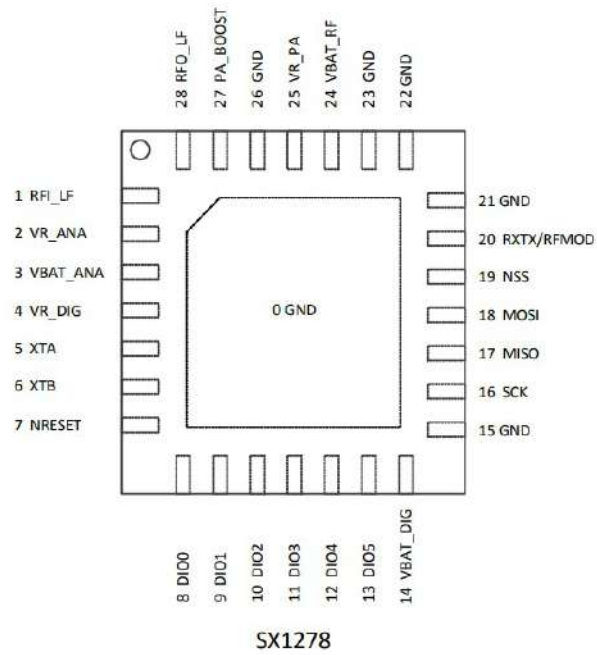


FIGURE 3.22: Pin Configuration of LoRa SX1278

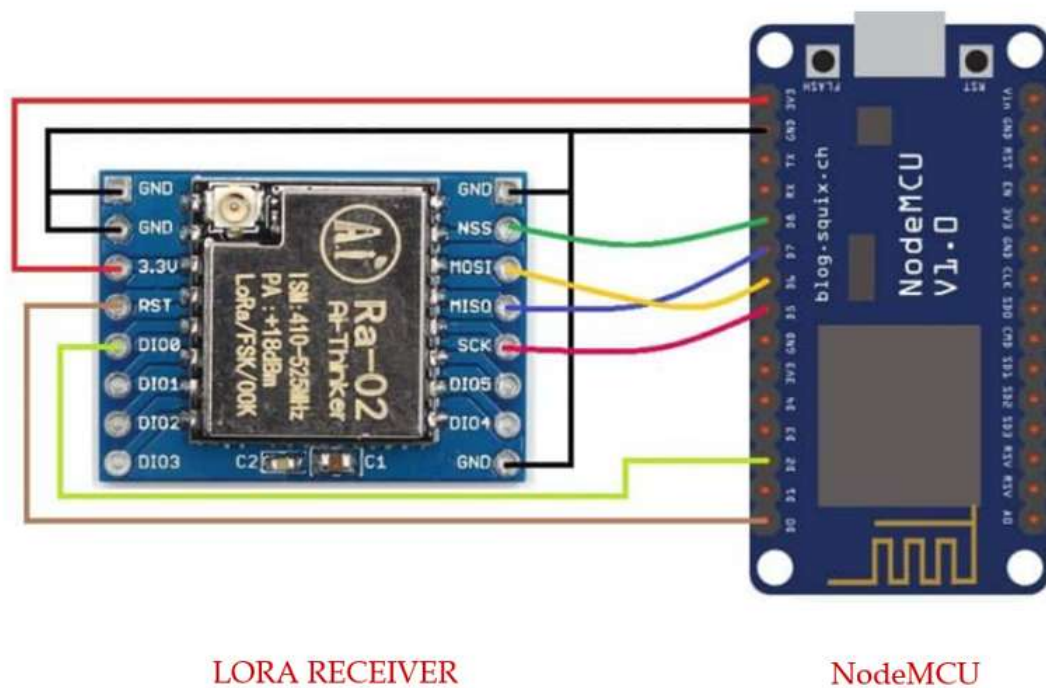


FIGURE 3.24: Receiver Schematic Diagram

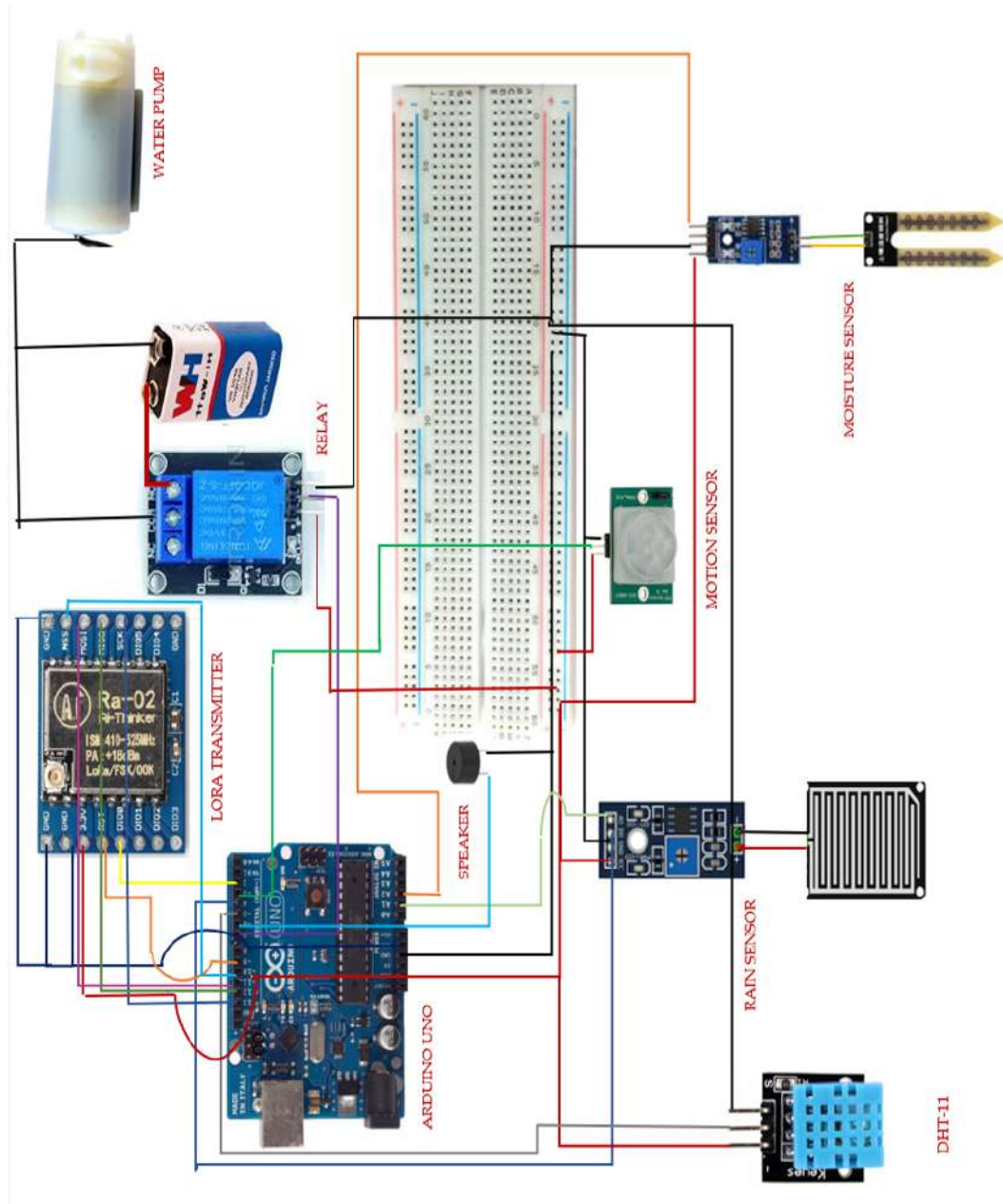


FIGURE 3.23: Transmitter Schematic Diagram

3.1.3.8 LoRa Receiver Configuration

LoRa SX1278 transmitter and receiver are the same boards. On the receiver side, LoRa SX1278 is connected with NodeMCU/ ESP8266. The table 3.7 shows the interaction of LoRa receiver with ESP8266.

TABLE 3.6: LoRa Receiver Interaction with ESP8266

LoRa Receiver	ESP8266
3.3V	3.3V/Vin
Gnd	Gnd
NSS	D8
DIO0	D1
SCK	D5
MISO	D6
MOSI	D7
RST	D0

The Arduino codes of transmitter and receiver side used in this project are available at A.

3.1.3.9 Relay Configuration

Relay is connected with ESP8266 and the motor is connected to the relay. The motor is connected to the receiver side in order to ON/OFF the motor. The table shows the interaction between Relay and ESP8266.

TABLE 3.7: Relay Interaction with ESP8266

Relay	ESP8266
Vcc	3.3V/Vin
Gnd	Gnd
IN1	D1

3.2 Crop Diagnosis

3.2.1 Methodology for the Crop Diagnosis

Methodologies adopted for the entire study:

1. Literature survey on recent studies on crop diagnosis.
2. Study of Data Acquisition.
3. Complete Analysis of Data pre-processing techniques.
4. Complete study of CNNs design and implementation.
5. Study of Python, TensorFlow, keras and numPy libraries.
6. Training and testing of the model.
7. Visualization of trained model performance.

3.2.2 Experimental and Analytical Work Completed in the Crop Diagnosis

The detailed flow diagram of Crop Diagnosis system is shown in Fig. 3.25. The flow chart can be broadly classified into three parts namely Data Acquisition, Data processing and Classification/ Model Design.

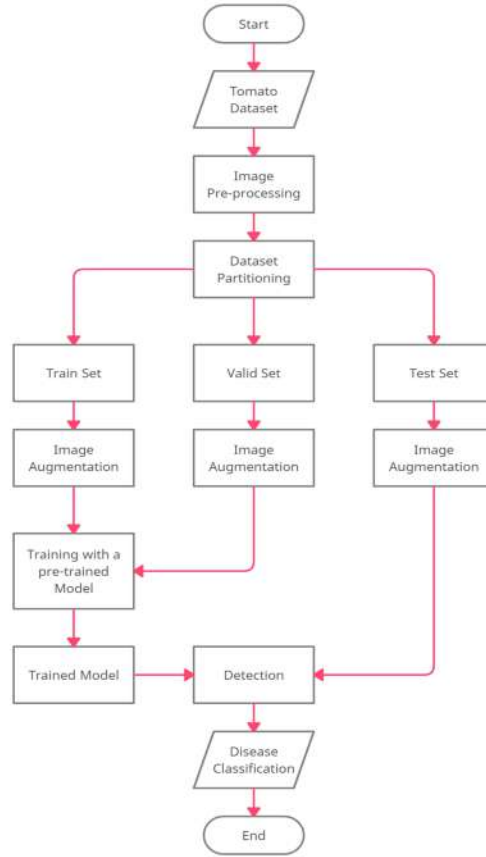


FIGURE 3.25: Flow chart of the Crop Diagnosis system.

3.2.2.1 Data Acquisition

Images of Tomato diseased leaves were taken from an open access repository of images [72]. The dataset includes over 87000 images of 14 crops, such as tomatoes, apples, blueberry, grapes, raspberry, potatoes, strawberry, soybeans and squash. Tomato was selected as the target crop from the above mentioned 14 crops. Healthy leaves and other 9 diseases categories of tomato leaves are shown in Figure. 3.26. The dataset has a total of 18345 images of 10 different classes(see Table 3.8). Training and validation sets are generated from the total dataset in the ratio of 80:20 respectively. The total validation set is 4585 images. Around 50 images in every category from the training set are randomly picked and are withdrawn from training folders for testing purpose. The size of all the images is 256×256 and the format employed is jpeg.



FIGURE 3.26: Sample tomato leaf images from the dataset.

TABLE 3.8: Description of tomato leaf dataset

S. No	Classes	No. of images in the dataset	Images for validation	Illustration
1	Bacterial spot	1702	425	See Figure 2 first row No1, No2
2	Early blight	1920	480	See Figure 2 first row No3, No4
3	Late blight	1851	463	See Figure 2 first row No5, second row No1
4	Leaf mold	1882	470	See Figure 2 second row No2, No3
5	Septoria leaf spot	1745	436	See Figure 2 second row No4, No5
6	Two-spotted spider mite	1741	435	See Figure 2 third row No1, No2
7	Target spot	1827	457	See Figure 2 third row No3, No4
8	Yellow leaf curl disease	1961	490	See Figure 2 third row No5, fourth row No1
9	Mosaic virus	1790	448	See Figure 2 fourth row No2, No3
10	Healthy	1926	481	See Figure 2 fourth row No4, No5
Total		18345	4585	

3.2.2.2 Data pre-processing

For CNN classifier, dataset images should be preprocessed to gain stability for improved feature extraction. The noise in the images of the dataset used is minimal, so noise removal was not used in this data preprocessing. For CNN classifiers used, the image size requirement is 224 x 224 pixels. So, the images were resized to the required pixels. Data standardization is performed by splitting all pixel values. It is also ensured that the several default values involved in initialization and termination are as per requirement.

3.2.2.3 Data augmentation

Typical deep convolutional neural networks contain millions of parameters; therefore, stupendous amounts of data are required. Otherwise, the deep neural network may not be robust or it overfits. The main purpose of using augmentation is to populate the dataset and introduce slight distortion to the images which help in reducing overfitting while training the model [73]. The data augmentation process was conducted to the training database which includes rescaling, setting both sheer and zoom range to 0.2 and doing a horizontal flip [74]. Whereas only rescaling was done for the testing set.

3.2.3 Modeling, Analysis and Design of Crop Diagnosis

Generally, convolutional neural networks (CNN) are used for creating a machine learning model that works on the unlabeled image inputs and converts them to corresponding classification output labels. But, to train such neural networks, a lot of data is needed which is not feasible. But with transfer learning, a solid machine learning model can be built with comparatively minimal training data because the model is already pre-trained but on different data [75]. For tomato leaf disease detection, from popular architectures like VGG, ResNet, Inception, MobileNet and DenseNet, one model per architecture is analyzed. The models used are VGG19, ResNet152V2, InceptionV3, InceptionResNet152V2, Xception, MobileNetV2 and DenseNet201. The architecture of the proposed model (see Fig. 3.27) consists of a pre-trained model followed by a reshape layer, a flatten layer, a dense layer, and finally followed by a Softmax activation function to do classification. Convolution and pooling layers architecture of all models except MobileNetV2 are referred from Ref. [76].

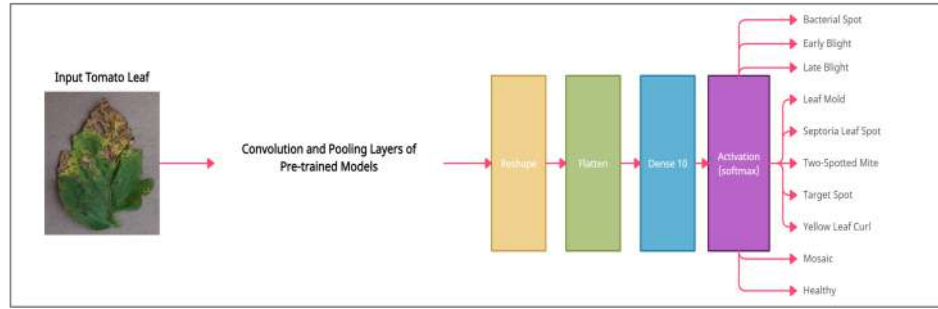


FIGURE 3.27: The architecture of the proposed system.

3.2.3.1 VGG19

VGG19 [37, 77] is the largest model among all the models used for comparison in terms of size and it has 16 convolutional layers, 5 maxpool layers, 3 fully connected layers and one activation layer (Softmax) in a total of 19 layers. VGG19 has a top-1 accuracy of 0.713, a top-5 accuracy of 0.9 and a depth of 26. It has a total of 20,275,274 parameters with 250,890 trainable parameters and 20,024,384 non-trainable parameters. The model architecture is shown in Figure. 3.28.

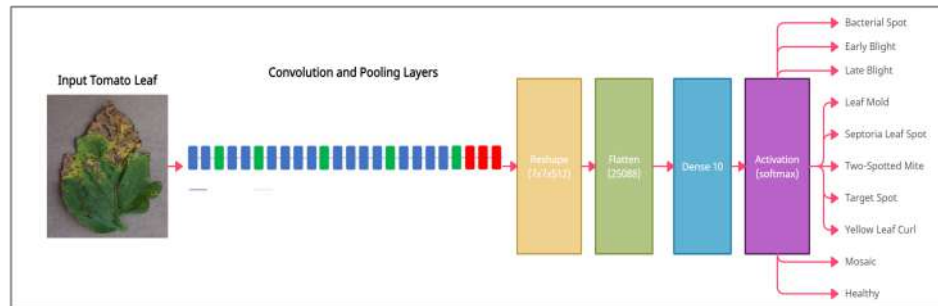


FIGURE 3.28: Architecture of VGG19 model.

3.2.3.2 ResNet152V2

ResNet, short for Residual Network won 1st place in the ILSVRC 2015 classification competition with a top-5 error rate of 3.75% [38, 78]. As the name says, it has 152 layers with top-1 and top-5 accuracies of 0.78 and 0.942 respectively. ResNet152V2 is an improvised version of ResNet152. The total parameters of the ResNet152V2 are 58,733,060 which consist of two types of parameters: the trainable parameters and the

non-trainable parameters, which are 401,412 and 58,331,648 respectively. The architecture of ResNet is shown in Figure. 3.29. ResNet152V2 architecture is the same as ResNet but with increased layers.

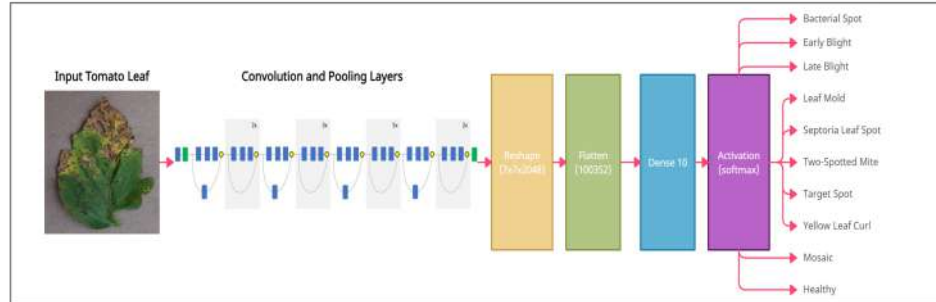


FIGURE 3.29: Architecture of ResNet model.

3.2.3.3 InceptionV3

InceptionV3 [39, 79] is a 42-layer deep learning network with fewer parameters. Less number of parameters makes the model less prone to overfitting thus increasing the accuracy. Similar to ResNet152V2, InceptionV3 is not the first model in its family. The first version, a 22 layered architecture was GoogleNet later called InceptionV1 which won the ILSVRC 2014. InceptionV3 has top-1 and top-5 accuracies of 0.779 and 0.937 respectively and 159 layer depth. The total parameters of the InceptionV3 are 22,007,588 which consist of two types of parameters: the trainable parameters and the non-trainable parameters, which are 401,412 and 23,587,712 respectively. The InceptionV3 model is shown in Figure. 3.30.

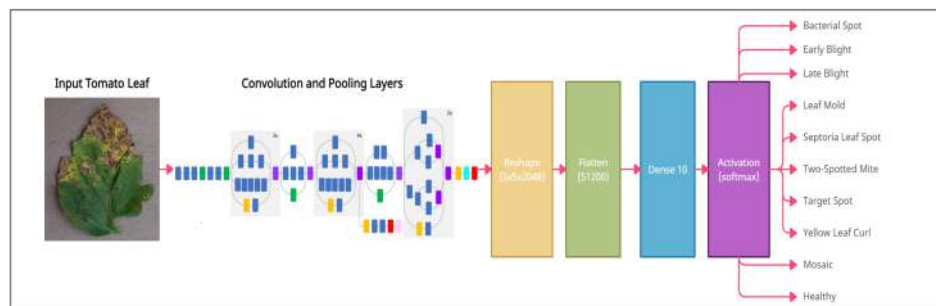


FIGURE 3.30: Architecture of the InceptionV3 model.

3.2.3.4 InceptionResNet152V2

InceptionResNetV2 [40, 80] is a hybrid architecture. Its basic architecture is similar to the Inception family but with connections resembling the ResNet family and size as much as ResNet152V2. With 164 layers, the model has 0.803 and 0.953 as top-1 and top-5 accuracies respectively with total parameters of 54,490,340 and depth of 572. It has trainable parameters of 153,604 and non-trainable parameters of 54,336,736. The model architecture is shown in Figure. 3.31.

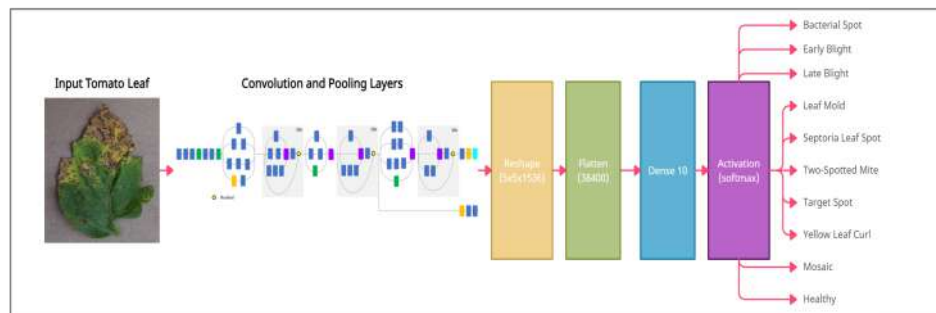


FIGURE 3.31: Architecture of the InceptionResNetV2 model.

3.2.3.5 Xception

Xception [41] is a 36 layer model inspired by Inception architecture. It has a top-1 accuracy of 0.79 and top-5 accuracy of 0.945 with a size similar to InceptionV3 and a depth of 126. It has a total of 21,865,010 parameters with 1,003,530 trainable and 20,861,480 non-trainable parameters. The model architecture is shown in Figure. 3.32.

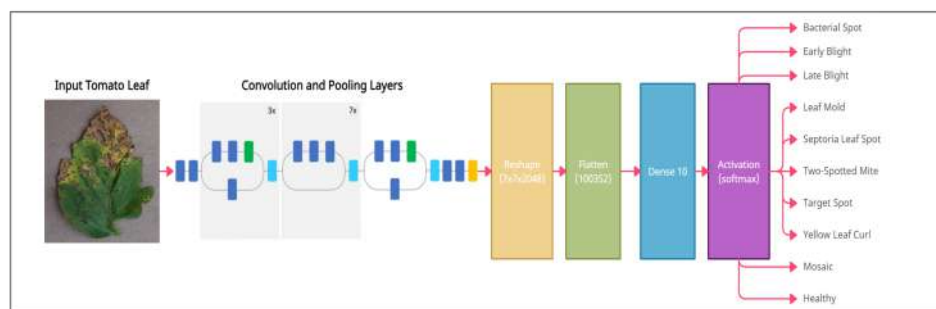


FIGURE 3.32: Architecture of the Xception model.

3.2.3.6 MobileNetV2

MobileNetV2 [42] is a model that is designed for mobile devices and uses an inverted residual structure. As it is used for mobile devices, the size of the model is very small. It has 0.713 and 0.901 top-1 and top-5 accuracies respectively with 88 layer depth. It has a total of 2,885,194 parameters with 627,210 trainable and 2,257,984 non-trainable parameters respectively. The model architecture is shown in Figure. 3.33 [81].

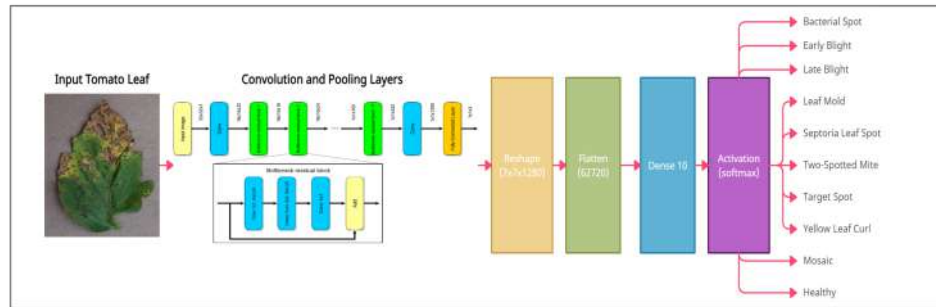


FIGURE 3.33: Architecture of the MobileNetV2 model.

3.2.3.7 DenseNet201

As the name says, DenseNet201 [43] has 201 layers depth with a top-1 accuracy of 0.773 and a top-5 accuracy of 0.936. In DenseNet, each layer has values of every preceding layer. . It has a total of 19,262,794 parameters with trainable parameters of 940,810 and non-trainable parameters of 18,321,984. The model architecture is shown in Figure. 3.34.

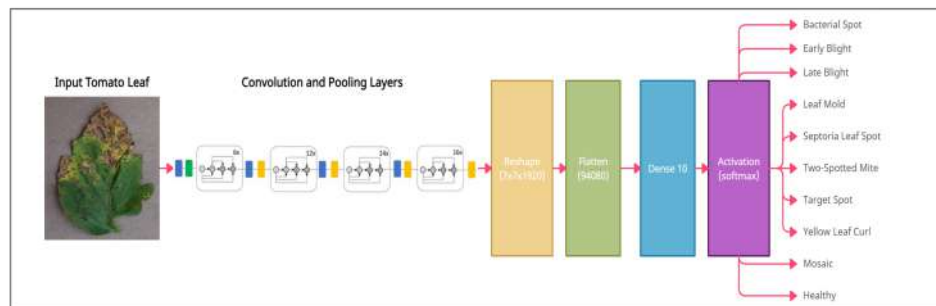


FIGURE 3.34: Architecture of the DenseNet201 model.

3.2.4 Optimizer and Loss Function

In above all 7 models, Adam [82, 83] is the optimizer used. It works based on the formula shown in Eq. (3.1).

$$w_{t+1} = w_t - \left(\frac{m_t}{1 - \beta_1^t} \right) \left(\frac{\alpha}{\sqrt{\frac{v_t}{1 - \beta_2^t}} + \epsilon} \right). \quad (3.1)$$

where w_{t+1} and w_t are the weights at time $t+1$ and t respectively, β_1^t and β_2^t are the decay rates of an average of gradients at time t , α is the learning rate and ϵ is a small positive constant (to avoid zero in the denominator). m_t is derived from the Eq. (3.2) and v_t is derived from equation Eq. (3.3).

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) \left[\frac{\delta L}{\delta w_t} \right]. \quad (3.2)$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) \left[\frac{\delta L}{\delta w_t} \right]^2. \quad (3.3)$$

where β_1 and β_2 are the decay rates of an average of gradients, m_t and m_{t-1} are the aggregate of gradients at time t and $t-1$ respectively, v_t and v_{t-1} are the sum of squares of past gradients at time t and $t-1$ respectively, δL is the derivate of the loss function (shown in Eq. (3.4)) and δw_t is the derivate of weights at time t .

Categorical cross-entropy is the loss function used. The formula for calculating the loss for multiclass classification is represented in the following Eq. (3.4).

$$Loss = - \sum_{c=1}^M y_{o,c} \log(p_{o,c}). \quad (3.4)$$

where M is the number of classes, c is the class label, o is the observation, y is the binary indicator if c is the correct classification of o and p is the predicted probability observation o is of class c .

3.3 Mobile Application Development

3.3.1 Methodology for the Mobile Application Development

Methodologies adopted for the entire study:

1. Detailed study and analysis of mobile application development on Android Studio.
2. Complete study of visualization of sensor data on the mobile application.
3. Detailed study of adding the trained CNN model into the mobile application.
4. Prototyping the entire project setup.

3.3.2 Experimental and Analytical Work Completed in the Mobile Application Development

3.3.2.1 Tools and Technologies

1. Android

It is an open source Operating System that is utilized in smartphones and tablet computers. It is developed by Google Inc. Android application can be written using the languages such as Java, C++ and Kotlin using Software Development Kit (SDK). It offers a unified approach for application development for the devices where we as developers need to develop applications for Android and their execution applications and should be able to run on different devices which are powered by Android.

2. Android Architecture

Android Operating System is considered to be a stack of software component. It is divided into five layers [84]. It is shown in the figure 3.35.

- (a) Linux Kernel

Utilizes Linux 2.6 which is approximately consists of 115 patches. It provides the basic system functionality. The functionality includes device management, process management and memory management. This device

management includes display, keypad, camera etc. The kernel present in the system handles all the things properly.

(b) Android Runtime

It is the third section of the architecture. The section introduces a new key component called Dalvik Virtual Machine that is similar to Java Virtual Machine. It is specifically designed and optimized for Android devices.

(c) Application Framework

The application framework layer gives us with high level services that can be used to create applications in form of Java classes. Application developed will allow this feature utilize these services in their application.

(d) Applications

Android application is at the topmost layer. The application is written and can be installed on this layer only.

3. Android Studio IDE

Android Studio Integrated Development Environment is used to create mobile application [85]. The figure 3.36 shows a snippet from Android Studio IDE. The codes for android studio used in this project are available at B.

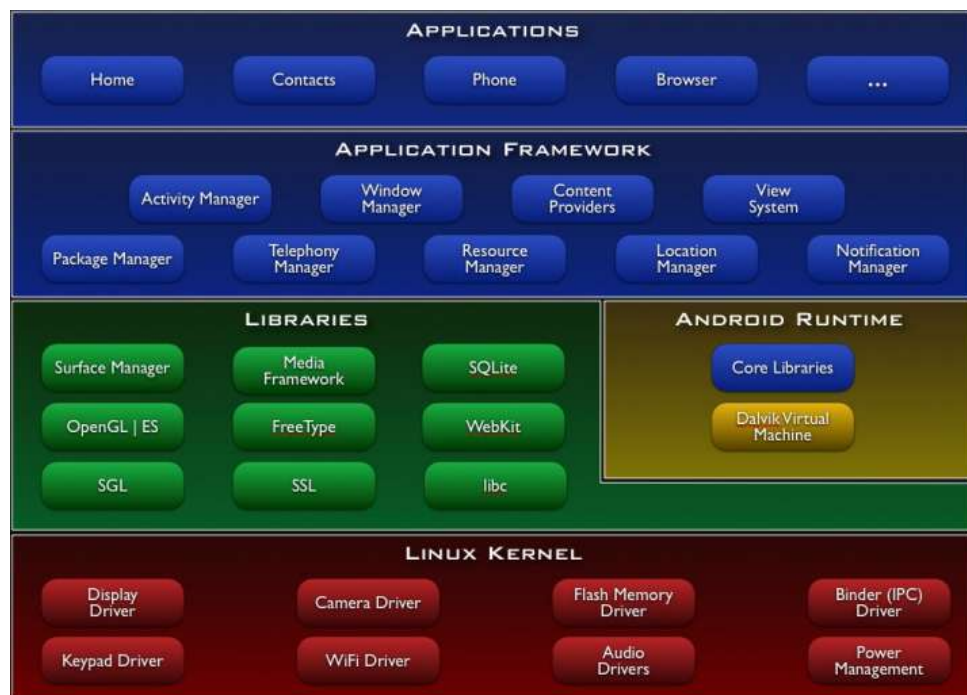


FIGURE 3.35: Android Architecture

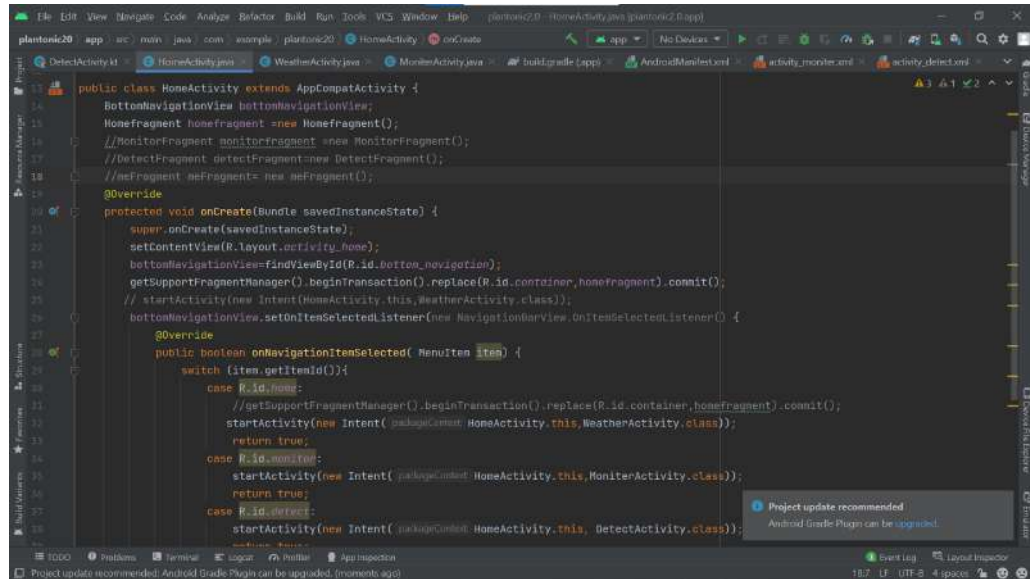


FIGURE 3.36: Android Studio IDE

3.3.3 Modeling, Analysis and Design of Mobile Application Development

The Android Application has the following components which are divided into two main parts; Front End and Back End. Front end consists of XML files, we use linear layout where image view, card view, and test views are used to display images, texts and the card view is the wrapped up small layout where we can add different elements. So, the layout finally makes up an interactive and dynamic UI (User Interface). The back end consists of source code to communicate with the server (MQTT) where the Paho client on the java side (Android Application) tries to communicate with the MQTT broker and then the data from the servers are received and displayed using the front end. The whole Android Application Flow is shown in the figure 3.37.

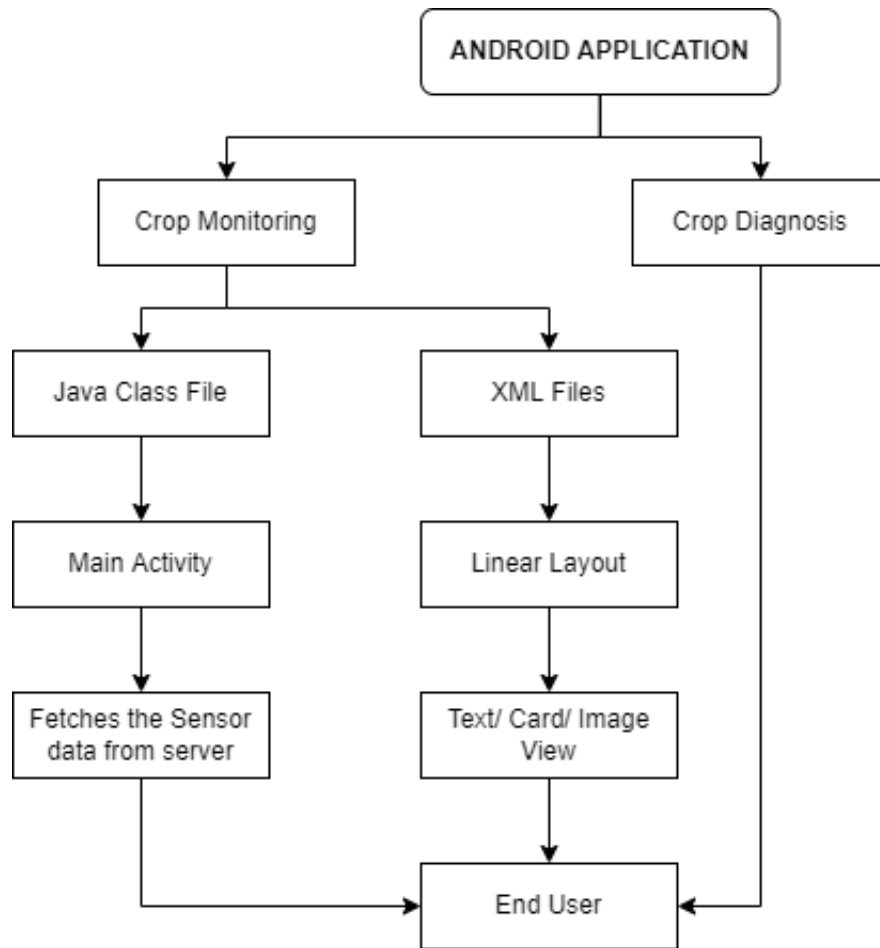


FIGURE 3.37: Flowchart of Android Application

Chapter 4

RESULTS, DISCUSSIONS AND CONCLUSIONS

4.1 Results and Analysis

4.1.1 Results and Analysis of Crop Monitoring

4.1.1.1 Working of Soil Moisture Sensor

The figure 4.1 shows the serial monitor of Soil Moisture Sensor data transmitting from Arduino Uno. The serial monitor shows the packet and value information.



```
COM7
Sending packet: 1
Plants need water..., notification sent
Analog Value :
Soil Moisture Value: 1023
Soil Moisture: -149%
Sending packet: 2
Plants need water..., notification sent
Analog Value :
Soil Moisture Value: 1023
Soil Moisture: -149%
Sending packet: 3
Plants need water..., notification sent
Analog Value :
Soil Moisture Value: 1023
Soil Moisture: -149%
Sending packet: 4
Plants need water..., notification sent
Analog Value :
Soil Moisture Value: 1023
Soil Moisture: -149%
Sending packet: 5
Plants need water..., notification sent
```

FIGURE 4.1: Serial Monitor of Soil Moisture Sensor

4.1.1.2 Working of DHT-11 Sensor

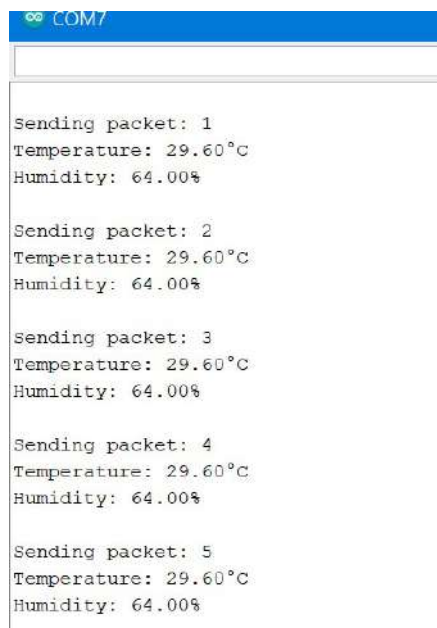
The figure 4.2 shows the serial monitor of DHT-11 Sensor data transmitting from Arduino Uno. The serial monitor shows the packet and value information.

4.1.1.3 Working of Rain Sensor

The figure 4.3 shows the serial monitor of Rain Sensor data transmitting from Arduino Uno. The serial monitor shows the packet and value information.

4.1.1.4 Receiving of Sensor Data at ESP8266

The figure 4.4 shows the serial monitor of all sensor's data received at ESP8266. The serial monitor shows the packet and value information.

A screenshot of a serial monitor window titled 'COM7'. The window displays five identical data packets being sent. Each packet contains the following information: 'Sending packet: 1' (or 2, 3, 4, 5), 'Temperature: 29.60°C', and 'Humidity: 64.00%'.

```
COM7

Sending packet: 1
Temperature: 29.60°C
Humidity: 64.00%

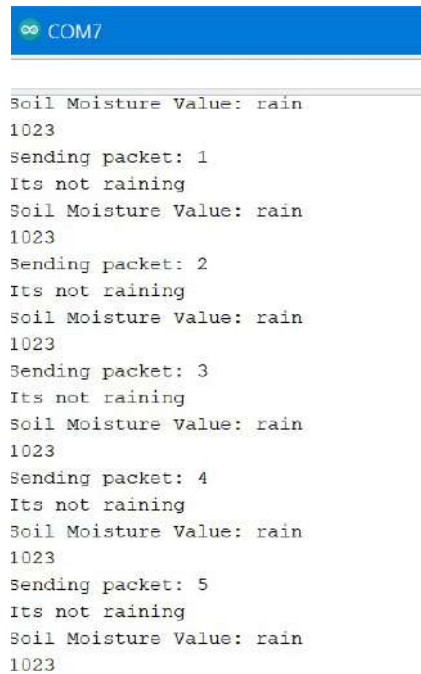
Sending packet: 2
Temperature: 29.60°C
Humidity: 64.00%

Sending packet: 3
Temperature: 29.60°C
Humidity: 64.00%

Sending packet: 4
Temperature: 29.60°C
Humidity: 64.00%

Sending packet: 5
Temperature: 29.60°C
Humidity: 64.00%
```

FIGURE 4.2: Serial Monitor of DHT-11 Sensor



```

COM7
Soil Moisture Value: rain
1023
Sending packet: 1
Its not raining
Soil Moisture Value: rain
1023
Sending packet: 2
Its not raining
Soil Moisture Value: rain
1023
Sending packet: 3
Its not raining
Soil Moisture Value: rain
1023
Sending packet: 4
Its not raining
Soil Moisture Value: rain
1023
Sending packet: 5
Its not raining
Soil Moisture Value: rain
1023

```

FIGURE 4.3: Serial Monitor of Rain Sensor



```

COM6
Received packet: 3/-149&1023@29.50$64.00' with RSSI -113
Packet No = 3
Temperature: 29.50°C
Humidity = 64.00%
Soil Moisture percent: -149
Led Turned ON
1023
its not rainng
Received packet: 6/-149&1023@29.50$63.00' with RSSI -113
Packet No = 6
Temperature: 29.50°C
Humidity = 63.00%
Soil Moisture percent: -149
Led Turned ON
1023
its not rainng
Received packet: 9/-149&1023@29.50$64.00' with RSSI -113
Packet No = 9
Temperature: 29.50°C
Humidity = 64.00%
Soil Moisture percent: -149
Led Turned ON
1023
its not rainng

```

FIGURE 4.4: Serial Monitor of ESP8266 receiving sensor data

4.1.2 Results and Analysis of Crop Diagnosis

In all the models, for tomato leaf disease detection, the weights used are imagenet. Also, the top and bottom layers are removed from the architectures to personalize the input size and to change the output classes respectively. The experiment was conducted in Google Colaboratory, an open-source cloud service in Windows 10 operating system. The GPU allocated for all setups was Tesla T4. For the implementation of the code, the library of NumPy, Pandas, Keras, PyTorch and sklearn is used. Softmax is used as the activation function and no swapping of the dataset used for all models. In compiling these models, categorical cross-entropy and adam are used as the loss function and optimizer respectively. Metrics used are accuracy, precision, recall and AUC (area under the curve). Learning rate = 0.01, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 1e^{-07}$ and amsgrad=False are the parameters set in adam optimizer. Model loss and accuracy plots were used to determine the model performance.

4.1.2.1 Results with Default Conditions

For all models, a batch size of 32 has been used and the model has been trained for 30 epochs. The number of steps per epoch was equal to the length of the training set and the number of validation steps was equal to the length of the test set. As the number of samples taken is 18345 and batch size is 32, the number of iterations per epoch would be 497 that gives a total of 14910 iterations in 30 epochs. Model accuracy, loss, precision, recall and AUC are plotted to evaluate the performance of each model (see Fig. 4.5, 4.6, 4.7, 4.8, 4.9). From Table 4.1, it can be inferred that DenseNet201 performed with the highest accuracy of 98.43% and loss of 0.3873 followed closely by ResNet152V2 and MobileNetV2 with 97.86% and 97.31% accuracies respectively. But DenseNet201 and ResNet152V2 took a large training time too. VGG19 trained exceptionally well with a loss of 0.1524 but can only attain an accuracy of 95.63%.

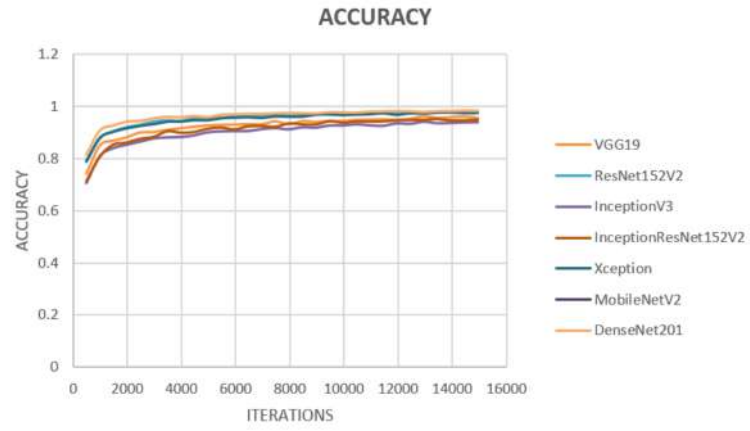


FIGURE 4.5: Accuracy of all Models

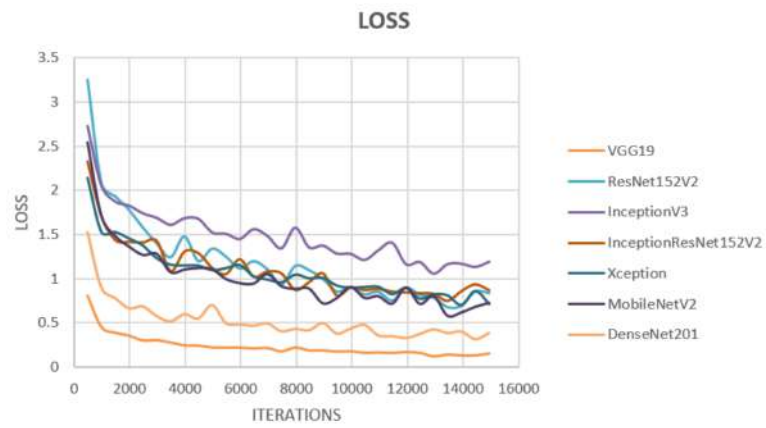


FIGURE 4.6: Loss of all Models

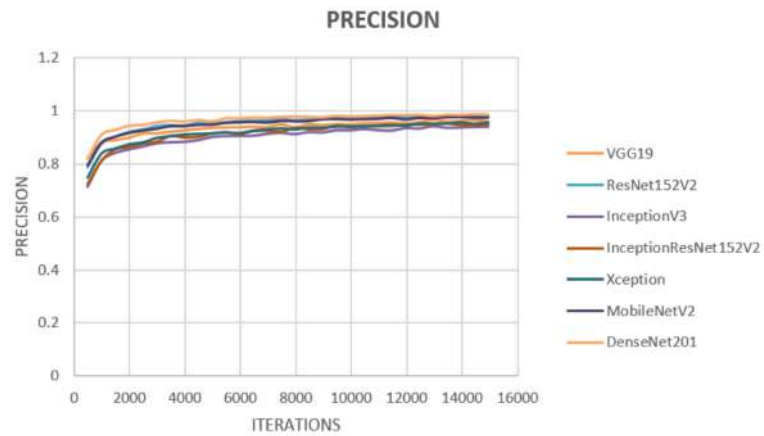


FIGURE 4.7: Precision of all Models

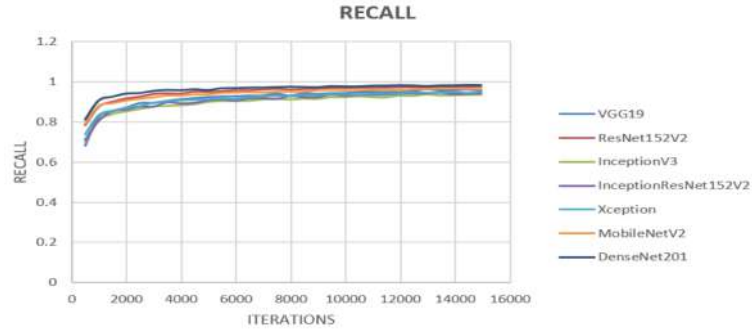


FIGURE 4.8: Recall of all Models

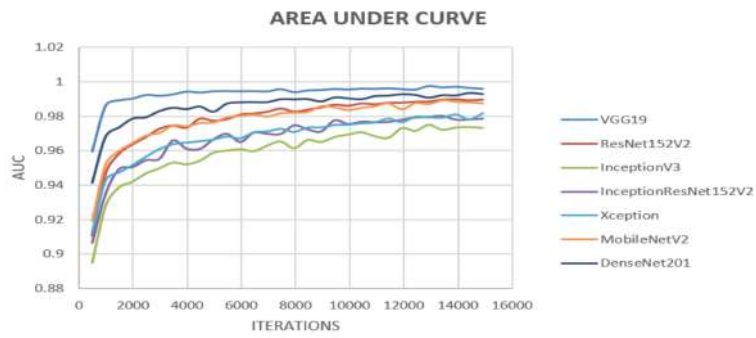


FIGURE 4.9: Area under curve of all Models

TABLE 4.1: Comparison of performance of all models under default conditions.

Model	Accuracy Loss		Precision	Recall	AUC	Parameters	Time
VGG19	0.9563	0.1524	0.9578	0.9547	0.9957	20,275,274	3hr 08min
ResNet152V2	0.9786	0.8403	0.9816	0.9766	0.9898	58,733,060	4hr 35min
InceptionV3	0.9399	1.1951	0.9399	0.9369	0.9733	22,007,588	3hr 38min
Inception- ResNetV2	0.9489	0.8708	0.9498	0.9409	0.9787	54,490,340	4hr 37min
Xception	0.953	0.7187	0.955	0.95	0.9818	21,865,010	3hr 16min
MobileNetV2	0.9731	0.726	0.9741	0.9641	0.9874	02,885,194	2hr 56min
DenseNet201	0.9843	0.3873	0.9845	0.983	0.9929	19,262,794	3hr 53min

4.1.2.2 Experiments on Epochs

To examine the effect of the number of epochs in model performance, all models are trained for 10, 30 and 50 epochs respectively. Except for the number of epochs, remaining all parameters were set the same as in default conditions. The results of the experiments on epochs can be seen in Table 4.2. It can be explicitly seen that an increase in the number of epochs is significantly improving the model performance for most of the models. DenseNet201 achieved an outstanding accuracy of 99.04% at the 50th epoch. But the accuracy didn't keep on increasing with the increase of epochs. The accuracy of the models became almost stable by 50 epoch (see Figure. 4.10). Other performance metrics like loss, precision, recall and AUC also converged and became stable. So, 50 epochs would be the ideal value for attaining the best performance from all the models.

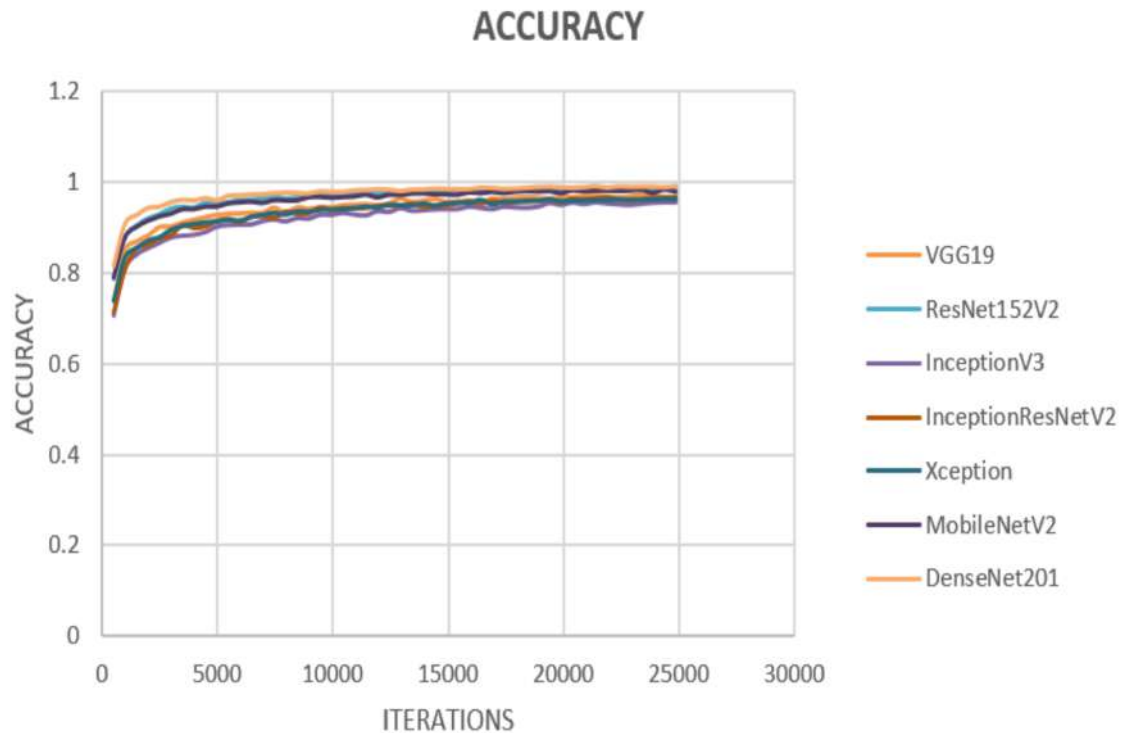


FIGURE 4.10: Accuracy plot of all CNN models with 50 epochs

TABLE 4.2: Comparison of performance of models with different number of epochs.

Model	Epochs	Iterations	Accuracy	Loss	Precision	Recall	AUC	Time
VGG19	10	4,970	0.9281	0.221	0.9342	0.9233	0.9943	2:05
	30	14,910	0.9563	0.1524	0.9578	0.9547	0.9957	3:08
	50	24,850	0.9671	0.1144	0.9683	0.9664	0.9966	4:12
ResNet152V2	10	4,970	0.9521	1.3456	0.9551	0.9501	0.9775	2:41
	30	14,910	0.9786	0.8403	0.9816	0.9766	0.9898	4:35
	50	24,850	0.9869	0.5373	0.9899	0.9849	0.9936	6:18
InceptionV3	10	4,970	0.9013	1.526	0.9016	0.9012	0.9589	2:43
	30	14,910	0.9399	1.1951	0.9399	0.9369	0.9733	3:38
	50	24,850	0.9545	0.8911	0.9546	0.9515	0.9805	4:01
Inception-ResNetV2	10	4,970	0.9145	1.1235	0.9149	0.9065	0.9666	2:53
	30	14,910	0.9489	0.8708	0.9498	0.9409	0.9787	4:37
	50	24,850	0.966	0.587	0.9669	0.958	0.9865	6:15
Xception	10	4,970	0.9136	1.1012	0.9139	0.9134	0.9666	2:10
	30	14,910	0.953	0.7187	0.955	0.95	0.9818	3:16
	50	24,850	0.9624	0.6338	0.9644	0.9594	0.9836	4:17
MobileNetV2	10	4,970	0.9465	1.1051	0.9475	0.9375	0.9763	1:51
	30	14,910	0.9731	0.726	0.9741	0.9641	0.9874	2:56
	50	24,850	0.9787	0.5931	0.9797	0.9697	0.9901	3:57
DenseNet201	10	4,970	0.9594	0.7041	0.9595	0.9581	0.9826	2:46
	30	14,910	0.9843	0.3873	0.9845	0.983	0.9929	3:53
	50	24,850	0.9904	0.2213	0.9905	0.9891	0.9957	4:28

4.1.2.3 Experiments on Batch Size

From previous experiments, it can be observed that DenseNet201, ResNet152V2 and MobileNetV2 performed better than other models. So, for the batch size experiment only these three models are used for experimentation. Batch size is the number of training samples considered in one iteration. In this experiment, batch size was set to 16, 32 and 64 for DenseNet201, ResNet152V2 and MobileNetV2 models. The number of epochs set for all the models is 50. Except for the batch size and epoch, remaining all parameters were set the same as in default conditions. Table 4.3 shows the results of

the used CNN models. As shown, an increase in batch size is not improving the corresponding model's performance. As a higher batch size consumes more memory and needs high computational power, a lower batch size of 16 is desirable.

The best accuracy of 99.06% is attained by DenseNet201 with a batch size of 16. Even ResNet152V2 got a good accuracy of 98.5% but it requires more computation and training time due to a large number of parameters. MobileNetV2 also achieved a high accuracy of 98.19%. Moreover, MobileNetV2 has lesser parameters that consume less memory and less computation which lead to lesser training time. MobileNetV2 model size is very small which is extremely handy for mobile devices. The accuracy and loss plots of MobileNetV2 with a batch size of 16 are shown in Figure. 4.11 and Figure. 4.12. Training time can be made even quicker by choosing an epoch of 30 which gives slightly lower accuracy but in a faster time. Results of identification of disease in tomato leaf using MobileNetV2 are shown in Figure. 4.13.

TABLE 4.3: Comparison of performance of all models with different batch sizes.

Model	Batch Size	Iterations	Accuracy	Loss	Precision	Recall	AUC	Time (hr:min)
DenseNet201	16	49,700	0.9906	0.3985	0.9913	0.9896	0.9951	4hr 11min
	32	24,850	0.9904	0.2213	0.9905	0.9891	0.9957	4hr 28min
	64	12,450	0.9904	0.1781	0.9906	0.9896	0.9961	4hr 37min
ResNet152V2	16	49,700	0.985	0.8391	0.987	0.9848	0.992	4hr 24min
	32	24,850	0.9869	0.5373	0.9899	0.9849	0.9936	6hr 18min
	64	12,450	0.9849	0.453	0.9852	0.9841	0.9931	6hr 46min
MobileNetV2	16	49,700	0.9819	0.7621	0.982	0.98	0.9912	3hr 24min
	32	24,850	0.9787	0.5931	0.9797	0.9697	0.9901	3hr 57min
	64	12,450	0.9816	0.387	0.9823	0.983	0.9922	4hr 13min

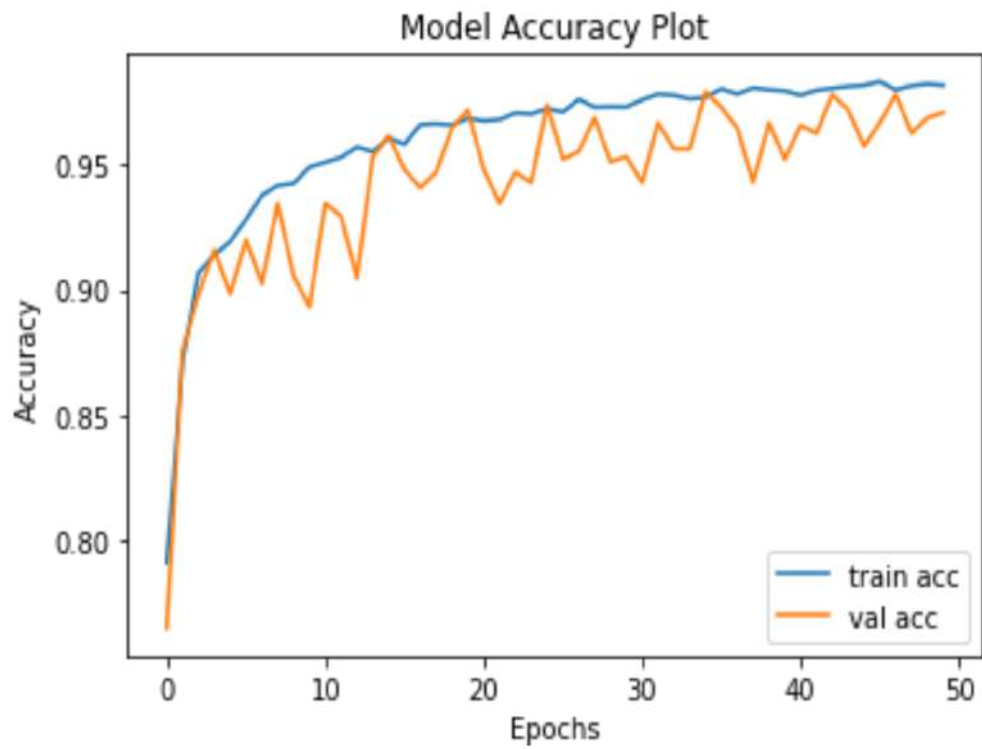


FIGURE 4.11: Accuracy plot of MobileNetV2 model

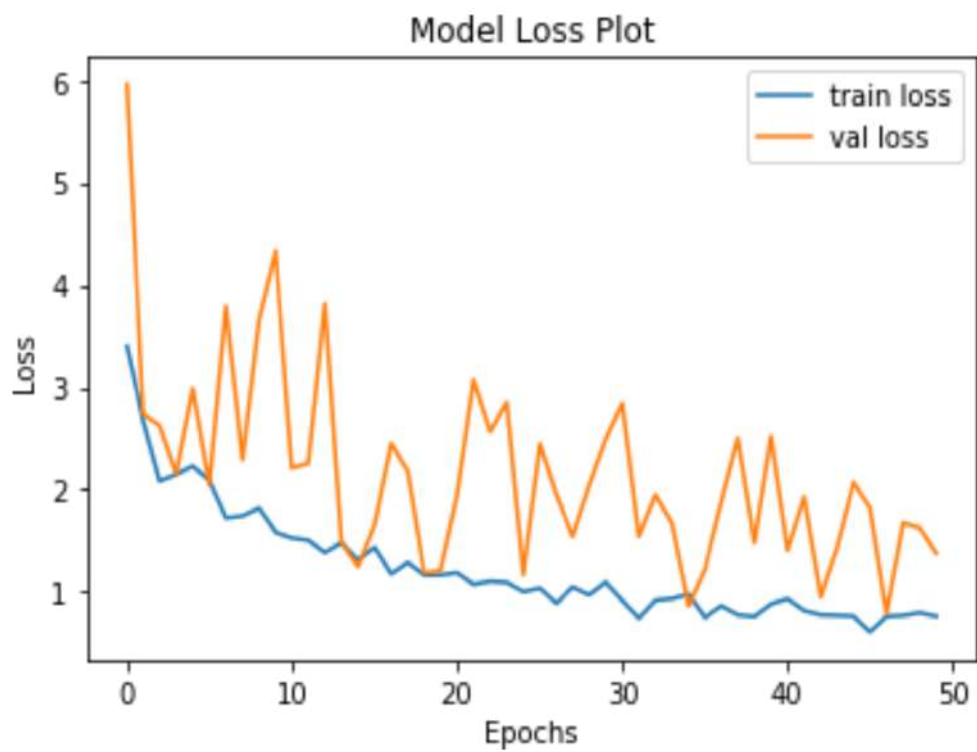


FIGURE 4.12: Loss plot of MobileNetV2 model

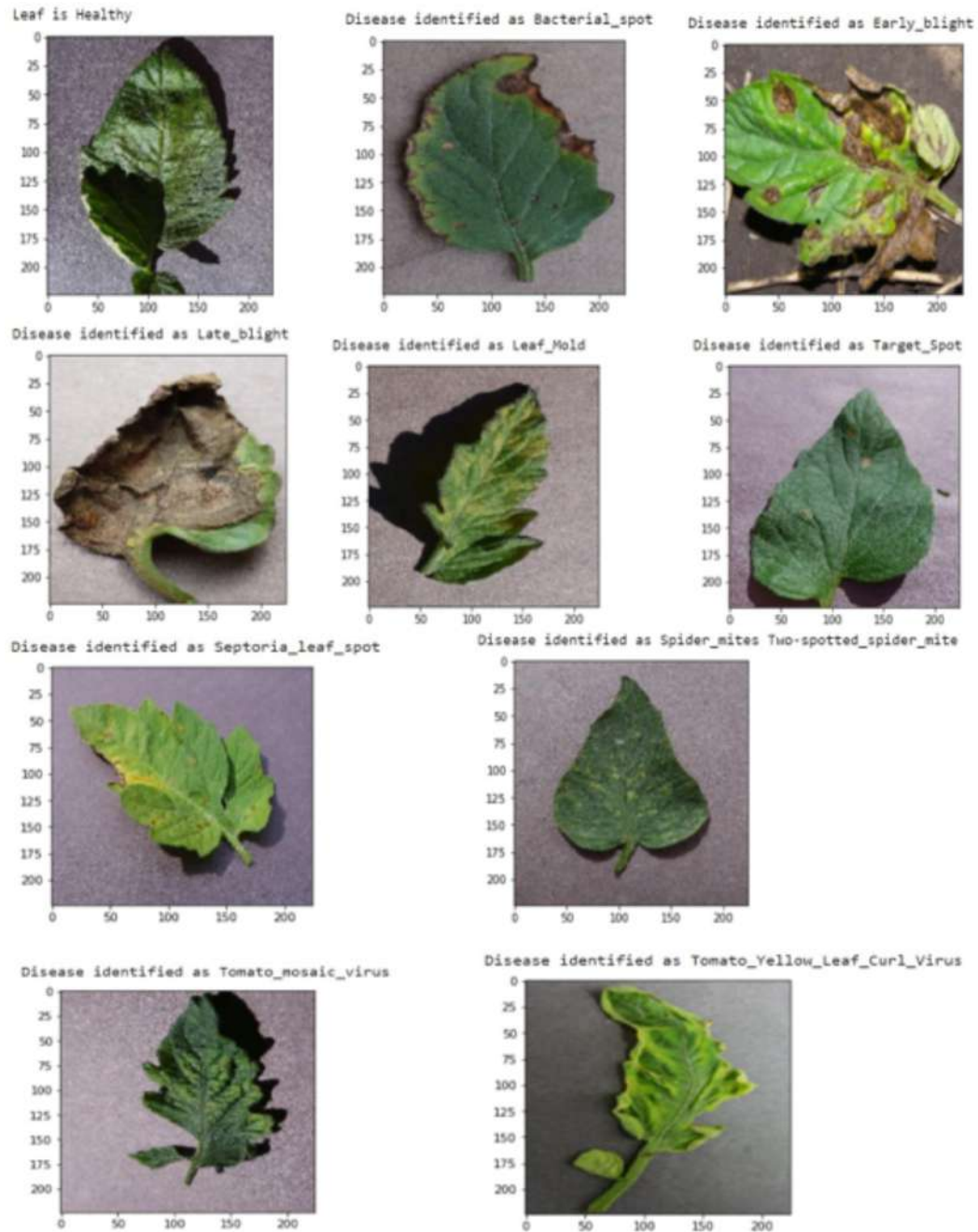


FIGURE 4.13: Detection of different diseases in tomato leaf using MobileNetV2 model

4.1.3 Results and Analysis of Mobile Application Development

The figures 4.14, 4.15, 4.16, 4.17 and 4.18 show the snippets of Mobile Application. The figure 4.14 shows the login page of the mobile application. It is the first page of the application.

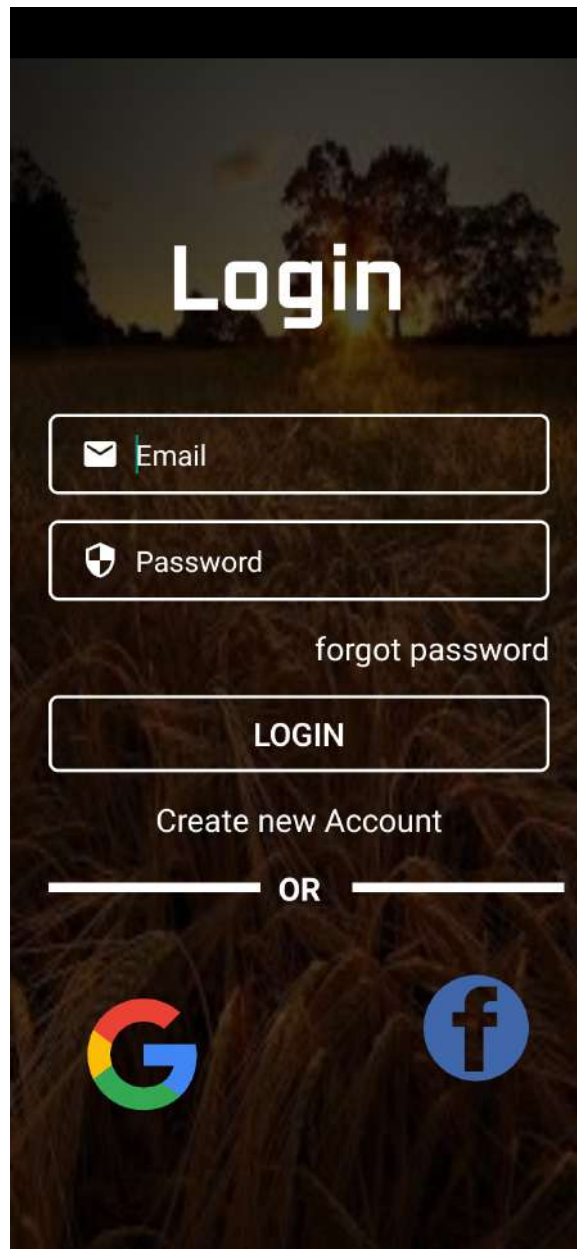


FIGURE 4.14: Login Page of the Mobile Application

The figure 4.15 shows the home page of the mobile application. It shows the name of the application along with the logo.



FIGURE 4.15: Home Page of the Mobile Application

The figure 4.16 shows the weather page of the mobile application. Climatic conditions like temperature, humidity, description, wind speed, cloudiness and pressure can be seen in this page.

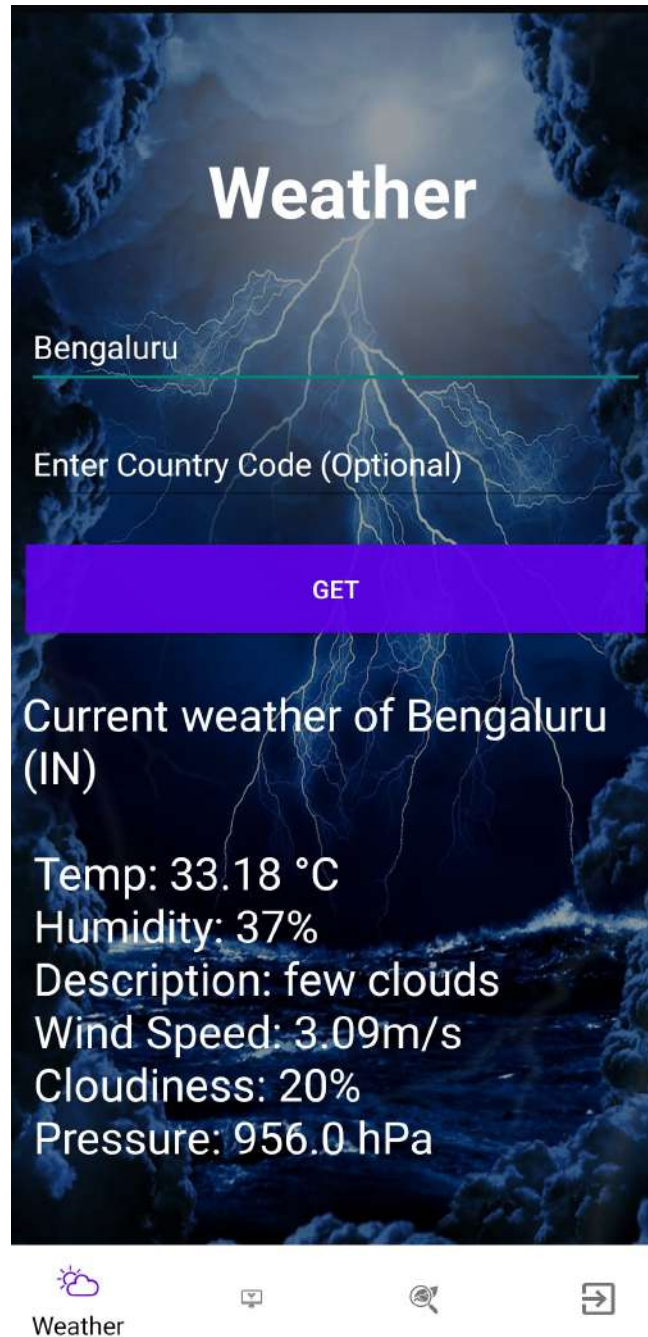


FIGURE 4.16: Weather Page

The figure 4.17 shows the monitoring page of the mobile application. The values of humidity, temperature, soil moisture, rain, motor status and mode of motor operation can be seen in this page.

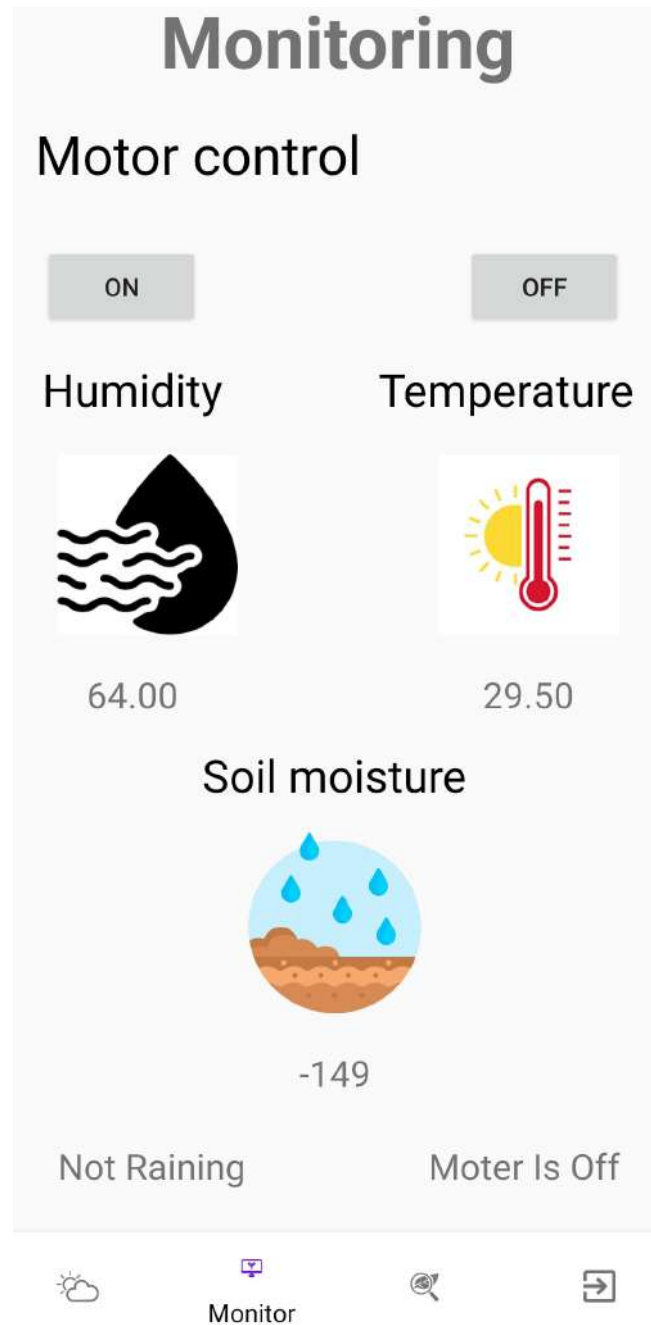


FIGURE 4.17: Crop Monitoring Page

The figure 4.18 shows the disease diagnosis page of the mobile application. This page has two options for inserting the input. One is from mobile storage and other way is real-time using camera. The detect button will show the results obtained along with the remedies.

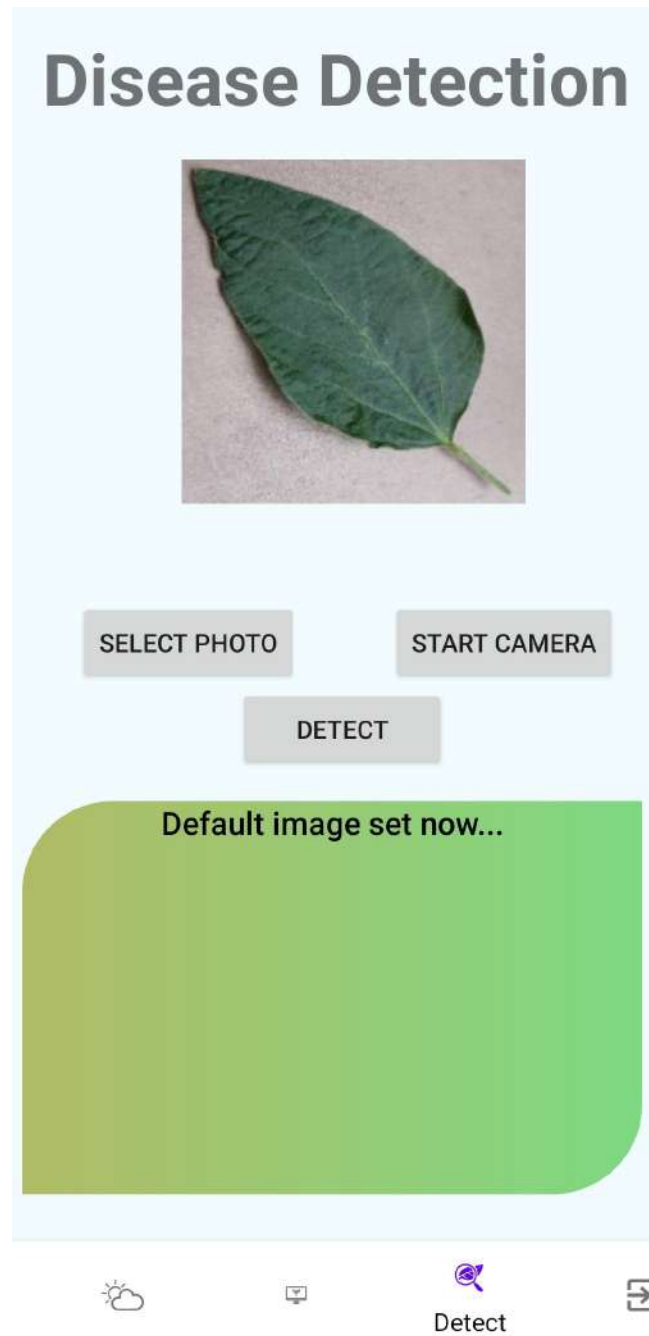


FIGURE 4.18: Crop Diagnosis Page

4.2 Comparative Study

There are a plethora of research papers available on crop monitoring and crop diagnosis. But none of those papers have talked about both Crop Monitoring and Diagnosis. In Crop Monitoring, there are many technologies available to use. Most used are Wi-Fi, ZigBee and LoRa. Out of all three, LoRa is the most effective with its low-power and long-range features. In Crop Diagnosis, plenty of options available in choosing the method of disease detection. Some popular ones include drone surveillance, high-spectral cameras, Image Processing and machine learning techniques. Out of all, machine learning techniques are becoming more popular. Convolutional Neural Networks (CNN) are the best suitable for disease detection.

4.3 Discussions

With the help of newer technologies like LoRa, the farmer can monitor a large area with less equipment and can increase crop yield. To build a most effective CNN model, VGG19, ResNet152V2, InceptionV3, InceptionResNet152V2, Xception, MobileNetV2 and DenseNet201 architectures were used and their performances have been analyzed. First, the performance of all models was evaluated under default conditions which are with a batch size of 32, epochs of 30 using adam optimizer. Later the same models experimented with a different number of epochs. From the results, it was observed that an increase in epochs improved the model's performance significantly but it also increased the training time. Based on the model's performance, DenseNet201, ResNet152V2 and MobileNetV2 were selected for batch size experimentations. Based on the results from these experiments, it was explicitly found that there is no significant improvement in the performance of these models with variations in batch size. As the increase in batch size consumes more memory and need high computation, a lower batch size of 16 is considered the best batch size. DenseNet201 with a batch size of 16 for 50 epochs achieved the best overall accuracy of 99.06%. Whereas ResNet152V2 got an accuracy of 98.5% but with increased computation and training time. Also, MobileNetV2 reached an accuracy of 98.2% moreover it consumed less memory, needed less computation and achieved faster results. The major implementation of the proposed system is as a mobile application that demands a model with less computational power and faster results. So MobileNetV2 is the best suitable model out of DenseNet201 and ResNet152V2. But when considered for best performance, DenseNet201 is unbeatable.

4.4 Conclusions

The aim of the project is to provide farmers with an end-to-end system that can bolster the crop productivity. The end-to-end system includes both crop monitoring system and crop diagnosis system integrated in a single mobile application. Thus, the farmer can rely on this mobile application for everything related to his field. We used various sensors to gather important information on factors that affect productivity. All these sensors are integrated with the Arduino Uno and this Arduino is again connected to LoRa transmitter. The LoRa transmitter sends the sensor data to the LoRa receiver. Single LoRa receiver is capable to receiving the data from multiple transmitter thus saving equipment and in turn decreasing cost of the system. The LoRa receiver sends the data to ESP8266 which sends the data to the server. This server is connected to the mobile application, thus showing the live sensor data. On the crop diagnosis side, Convolutional Neural Networks are used in order to train a model that can detect diseases from the leaf image. The model is trained with help of a huge leaf images database. The model is stored in a .h5 file and this file is uploaded into the mobile application. Thus the single application provides both crop monitoring and crop diagnosis.

4.5 Scope for Future Work

As of now, the system collects most crucial data from the sensors. But much more information can be collected with more advanced sensors like nutrients in the soil can be collected with higher end sensors etc. A detailed analysis can be done on the occurrence of diseases with varying environmental factors. The analysis can be done after collecting enough crop data with the existing system. MobileNetV2 model performance can be improved more by varying different hyper parameters. Also, the model can be trained for different crops. With addition of these features, a single system can be employed for all most every crop.

Bibliography

- [1] "Overview", *World Bank*, 2021. [Online]. Available: <https://www.worldbank.org/en/topic/agriculture/overview>. [Accessed: 10- Apr- 2022].
- [2] Data.worldbank.org. 2021. *Employment in agriculture (% of total employment) (modeled ILO estimate) — Data*. [online] Available at: <https://data.worldbank.org/indicator/SL.AGR.EMPL.ZS?locations=IN> [Accessed 10- Apr- 2022].
- [3] "Agriculture in Developing Countries: Which Way Forward?", *Iatp.org*, 2021. [Online]. Available: <https://www.iatp.org/sites/default/files/AgricultureinDevelopingCountriesWhichWay.htm>. [Accessed: 10- Apr- 2022].
- [4] "Agricultural Statistics at a Glance 2018", *Agricoop.gov.in*, 2022. [Online]. Available: <https://agricoop.gov.in/sites/default/files/agristatglance2018.pdf>. [Accessed: 10- Apr- 2022].
- [5] T. N. Liliane, and M. S. Charles, "Factors Affecting Yield of Crops", *IntechOpen*, 2020 [online]. Available: <https://www.intechopen.com/chapters/70658> doi: 10.5772/intechopen.90672.
- [6] "Tomato Disease Identification Key", *Vegetablemdonline.ppath.cornell.edu*, 2021. [Online]. Available: <http://vegetablemdonline.ppath.cornell.edu/DiagnosticKeys/TomWlt/TomWiltKey.html>. [Accessed: 12- Apr- 2022].
- [7] Kumar, S., Tiwari, P. and Zymbler, M. "Internet of Things is a revolutionary approach for future technology enhancement: a review", *J Big Data* 6, 111 (2019). Available: <https://doi.org/10.1186/s40537-019-0268-2>

- [8] I. Harris, "An Introduction to the Programming of Internet of Things, Specialization", *Coursera*, 2022. [Online]. Available: <https://www.coursera.org/specializations/iot>. [Accessed: 16- Apr- 2022].
- [9] S. Monk, "Programming Arduino: Getting Started with the Sketches", 2nd ed, McGraw Hill Education, 2016.
- [10] D. Hatanaka, A. Ahrary and D. Ludena, "Research on Soil Moisture Measurement Using Moisture Sensor," *IIAI 4th International Congress on Advanced Applied Informatics*, 2015, pp. 663-668, doi: 10.1109/IIAI-AAI.2015.289.
- [11] Gay, Warren, "DHT11 Sensor-Experimenting with Rasp", pp.1-13, 10.1007/978-1-4842-0769-7-1.
- [12] "PIR Sensor Working Principle", 2022. [Online]. Available: <https://robu.in/pir-sensor-working-principle/>. [Accessed: 16- Apr- 2022].
- [13] Massimo Banzi, "Getting Started with Arduino", 2015, (2nd. ed.). Make Books - Imprint of: O'Reilly Media, Sebastopol, CA.
- [14] H. Sharma and S. Sharma, "A review of sensor networks: Technologies and applications," *Recent Advances in Engineering and Computational Sciences (RAECS)*, 2014, pp. 1-4, doi: 10.1109/RAECS.2014.6799579.
- [15] Y. Kim, R. G. Evans, and W. M. Iversen, "Remote Sensing and Control of an Irrigation System Using a Distributed Wireless Sensor Network," *IEEE Trans. Instrum. Meas.*, vol. 57, no. 7, pp. 1379–1387, Jul. 2008, doi: 10.1109/TIM.2008.917198.
- [16] T. Kalaivani, A. Allirani and P. Priya, "A survey on Zigbee based wireless sensor networks in agriculture", *3rd International Conference on Trends in Information Sciences and Computing (TISC2011)*, 2011, pp. 85-89, doi: 10.1109/TISC.2011.6169090.
- [17] Tapashetti, S. and K.R, S., "Precision Agriculture using LoRa", *International Journal of Scientific and Engineering Research*, 2018, 9(5).
- [18] D. I. Săcăleanu, R. Popescu, I. P. Manciu, and L. A. Perișoară, "Data Compression in Wireless Sensor Nodes with LoRa," *10th International Conference on Electronics, Computers and Artificial Intelligence (ECAI)*, Jun. 2018, pp. 1–4, doi: 10.1109/ECAI.2018.8679003.

- [19] G. Sushanth and S. Sujatha, "IoT Based Smart Agriculture System," *International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET)*, Mar. 2018, pp. 1–4, doi: 10.1109/WiSPNET.2018.8538702.
- [20] Nayyar, Anand and Puri, Vikram, "Smart farming: IoT based smart sensors agriculture stick for live temperature and moisture monitoring using Arduino, cloud computing and solar technology", *The International Conference on Communication and Computing Systems (ICCCS)*, 2016, 673-680. 10.1201/9781315364094-121.
- [21] B. Mishra and A. Kertesz, "The Use of MQTT in M2M and IoT Systems: A Survey," *IEEE Access*, vol. 8, pp. 201071-201086, 2020, doi: 10.1109/ACCESS.2020.3035849.
- [22] Dunaieva, I., Mirschel, W., Popovych, V., Pashtetsky, V., Golovastova, E., Vecherkov, V., Melnichuk, A., Terleev, V., Nikonorov, A., Ginevsky, R., Lazarev, V. and Topaj, A., "GIS Services for Agriculture Monitoring and Forecasting: Development Concept", 2019, *Advances in Intelligent Systems and Computing*, pp.236-246.
- [23] Ballesteros, R.; Ortega, J.F.; Hernández, D.; Moreno, M.A, "Applications of geo-referenced high-resolution images obtained with unmanned aerial vehicles. Part I: Description of image acquisition and processing", 2014, *Precis. Agric.*, 15, 579–592.
- [24] Mihai Valentin, Herbei and Popescu, Cosmin and Bertici, Radu and Smuleac, Adrian and Popescu, George, "Processing and Use of Satellite Images in Order to Extract Useful Information in Precision Agriculture", 2016, *Bulletin of University of Agricultural Sciences and Veterinary Medicine Cluj-Napoca. Agriculture*, 2016, 73. 238. 10.15835/buasvmcnagr:12442.
- [25] R. Ben Ayed and M. Hanana, "Artificial Intelligence to Improve the Food and Agriculture Sector", *Journal of Food Quality*, vol. 2021, pp. 1-7, 2021. Available: 10.1155/2021/5584754.
- [26] Ildar Rakhmatulin, "Deep learning, machine vision in agriculture in 2021", *arXiv*, 2021. Available: arxiv.org/abs/2103.04893
- [27] S. Mohana Saranya, R. Rajalaxmi, R. Prabavathi, T. Suganya, S. Mohanapriya and T. Tamilselvi, "Deep Learning Techniques in Tomato Plant – A Review", *Journal of Physics: Conference Series*, vol. 1767, no. 1, p. 012010, 2021. Available: 10.1088/1742-6596/1767/1/012010.

- [28] S. Kujawa and G. Niedbała, "Artificial Neural Networks in Agriculture", *Agriculture*, vol. 11, no. 6, p. 497, 2021. Available: 10.3390/agriculture11060497.
- [29] J. Liu and X. Wang, "Early recognition of tomato gray leaf spot disease based on MobileNetv2-YOLOv3 model", *Plant Methods*, vol. 16, no. 1, 2020. Available: 10.1186/s13007-020-00624-2.
- [30] M. Xin and Y. Wang, "Research on image classification model based on deep convolution neural network", *EURASIP Journal on Image and Video Processing*, vol. 2019, no. 1, 2019. Available: 10.1186/s13640-019-0417-8.
- [31] K. K. Leong and L. L. Tze, "Plant Leaf Diseases Identification using Convolutional Neural Network with Treatment Handling System," 2020 *IEEE International Conference on Automatic Control and Intelligent Systems (I2CACIS)*, 2020, pp. 39-44, doi: 10.1109/I2CACIS49202.2020.9140103.
- [32] S. Pan and Q. Yang, "A Survey on Transfer Learning", *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 10, pp. 1345-1359, 2010. Available: 10.1109/tkde.2009.191.
- [33] C. Tan, F. Sun, T. Kong, W. Zhang, C. Yang and C. Liu, "A Survey on Deep Transfer Learning", *arXiv*, 2018. Available: arxiv.org/abs/1808.01974.
- [34] A. Afifi, A. Alhumam and A. Abdelwahab, "Convolutional Neural Network for Automatic Identification of Plant Diseases with Limited Data", *Plants*, vol. 10, no. 1, p. 28, 2020. Available: 10.3390/plants10010028.
- [35] "PlantVillage", *Plantvillage.psu.edu*, 2021. [Online]. Available: <https://plantvillage.psu.edu/>. [Accessed: 16- Apr- 2022].
- [36] Hughes, David and Salathe, Marcel, "An open access repository of images on plant health to enable the development of mobile disease diagnostics through machine learning and crowdsourcing." *arXiv*, 2015. [Online]. Available: <https://arxiv.org/abs/1511.08060>.
- [37] K. Simonyan and A. Zisserman, "Very deep Convolutional Networks for Large-Scale Image Recognition." *arXiv*, 2014. [Online] Available: <https://arxiv.org/abs/1409.1556>.
- [38] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770-778, doi: 10.1109/CVPR.2016.90.

- [39] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens and Z. Wojna, "Rethinking the Inception Architecture for Computer Vision", *arXiv*, 2015, pp. 2818-2826, doi: 10.1109/CVPR.2016.308.
- [40] C. Szegedy, S. Ioffe, V. Vanhoucke and A. A. Alemi, "Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning", *AAAI'17: Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, no. 4278-4284, pp. 4278-4284, 2016.
- [41] Chollet, Francois, "Xception: Deep Learning with Depthwise Separable Convolutions. 1800-1807. 10.1109/CVPR.2017.195." *arXiv*, 2017. [Online] Available: <https://arxiv.org/abs/1610.02357>.
- [42] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov and L. Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," 2018 *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 4510-4520, doi: 10.1109/CVPR.2018.00474.
- [43] Huang, Gao and Liu, Zhuang and van der Maaten, Laurens and Weinberger, Kilian, "Densely Connected Convolutional Networks." 10.1109/CVPR.2017.243, *arXiv*, 2017. [Online] Available: <https://arxiv.org/abs/1608.06993>.
- [44] S. Kaur, G. Joshi and R. Vig, "Plant Disease Classification using Deep Learning Google Net Model", *International Journal of Innovative Technology and Exploring Engineering*, vol. 8, no. 9, pp. 319-322, 2019. Available: 10.35940/ijitee.i1051.0789s19.
- [45] S. Mohanty, D. Hughes and M. Salathé, "Using Deep Learning for Image-Based Plant Disease Detection", *Frontiers in Plant Science*, vol. 7, 2016. Available: 10.3389/fpls.2016.01419.
- [46] K. Ferentinos, "Deep learning models for plant disease detection and diagnosis", *Computers and Electronics in Agriculture*, vol. 145, pp. 311-318, 2018. Available: 10.1016/j.compag.2018.01.009.
- [47] E. Too, L. Yujian, S. Njuki and L. Yingchun, "A comparative study of fine-tuning deep learning models for plant disease identification", *Computers and Electronics in Agriculture*, vol. 161, pp. 272-279, 2019. Available: 10.1016/j.compag.2018.03.032.

- [48] P. Jiang, Y. Chen, B. Liu, D. He and C. Liang, "Real-Time Detection of Apple Leaf Diseases Using Deep Learning Approach Based on Improved Convolutional Neural Networks", *IEEE Access*, vol. 7, pp. 59069-59080, 2019. Available: 10.1109/access.2019.2914929.
- [49] M. Islam, M. Nymur, M. Shamsoddin, S. Hasan, M. Shahadat and T. Khatun, "An Automated Convolutional Neural Network Based Approach for Paddy Leaf Disease Detection", *International Journal of Advanced Computer Science and Applications*, vol. 12, no. 1, 2021. Available: 10.14569/ijacsa.2021.0120134.
- [50] DeChant C, Wiesner-Hanks T, Chen S, Stewart EL, Yosinski J, Gore MA, Nelson RJ, Lipson H. "Automated Identification of Northern Leaf Blight-Infected Maize Plants from Field Imagery Using Deep Learning. *Phytopathology*." 2017 Nov;107(11):1426-1432. doi: 10.1094/PHYTO-11-16-0417-R. Epub 2017 Aug 24. PMID: 28653579.
- [51] C. Rahman et al., "Identification and recognition of rice diseases and pests using convolutional neural networks", *Biosystems Engineering*, vol. 194, pp. 112-120, 2020. Available: 10.1016/j.biosystemseng.2020.03.020.
- [52] K. Zhang, Q. Wu, A. Liu and X. Meng, "Can Deep Learning Identify Tomato Leaf Disease?", *Advances in Multimedia*, vol. 2018, pp. 1-10, 2018. Available: 10.1155/2018/6710865.
- [53] A. T and J. Murugaiyan, "Identification of Tomato Leaf Disease Detection using Pretrained Deep Convolutional Neural Network Models", *Scalable Computing: Practice and Experience*, vol. 21, no. 4, pp. 625-635, 2020. Available: 10.12694/scpe.v21i4.1780.
- [54] A. Elhassouny and F. Smarandache, "Smart mobile application to recognize tomato leaf diseases using Convolutional Neural Networks," *International Conference of Computer Science and Renewable Energies (ICCSRE)*, 2019, pp. 1-4, doi: 10.1109/ICCSRE.2019.8807737.
- [55] Masi, Emiliano Cantone, Giovanni Calavaro, Giuseppe Mastrofini, Manuel Subiaco, Paolo, "Mobile Apps Development: A Framework for Technology Decision Making", 2012, 10.1007/978-3-642-36632-1-4.
- [56] Oinas-Kukkonen, Harri Kurkela, Virpi, "Developing Successful Mobile Applications", *Journal of Computer Science and Technology - JCST*, 2003.

- [57] Islam, Dr. MD Rashedul and Mazumder Tridib, "Mobile application and its global impact", *International Journal of Engineering and Technology*, 2010, 10. 72-78.
- [58] Sunitha, R. and Elina, S., "A Study on Mobile Applications in Education" *IITM Journal of Management and IT*, 2020, 11(1), pp.91-97.
- [59] Ventola CL, "Mobile devices and apps for health care professionals: uses and benefits", 2014, P T;39(5):356-64. PMID: 24883008; PMCID: PMC4029126.
- [60] Choe, D., Choi, E. and Kim, D., "The Real-Time Mobile Application for Classifying of Endangered Parrot Species Using the CNN Models Based on Transfer Learning", *Mobile Information Systems*, 2020, pp.1-13.
- [61] Katharina Dehnen-Schmutz, Gemma L. Foster, Luke Owen, Séverine Persello, "Exploring the role of smartphone technology for citizen science in agriculture", *Agronomy for Sustainable Development*, 2016, Springer Verlag/EDP Sciences/INRA, 36 (2), pp.25. 10.1007/s13593-016-0359-9.
- [62] A. Johannes, A. Picon, A. Alvarez-Gila, J. Echazarra, S. Rodriguez-Vaamonde, A. D. Navajas, and A. Ortiz-Barredo, "Automatic plant disease diagnosis using mobile capture devices, applied on a wheat use case", *Computers and Electronics in Agriculture*, 2018, vol. 138, pp. 200–209.
- [63] Toseef, M. and Khan, M., "An intelligent mobile application for diagnosis of crop diseases in Pakistan using fuzzy inference system", *Computers and Electronics in Agriculture*, 2018, 153, pp.1-11.
- [64] Esgario, J., de Castro, P., Tassis, L. and Krohling, R., "An app to assist farmers in the identification of diseases and pests of coffee leaves using deep learning", *Information Processing in Agriculture*, 2021.
- [65] Petrellis, N., "Plant Disease Diagnosis for Smart Phone Applications with Extensible Set of Diseases. *Applied Sciences*, 2019, 9(9), p.1952.
- [66] S., R., Shriram, T., Raju, J., Hari, M., Santhi, B. and Brindha, G., "Farmer-Friendly Mobile Application for Automated Leaf Disease Detection of Real-Time Augmented Data Set using Convolution Neural Networks", *Journal of Computer Science*, 2020, 16(2), pp.158-166.
- [67] "Arduino - ArduinoBoardUno", *Arduino.cc*, 2022. [Online]. Available:<https://www.arduino.cc/en/main/arduinoBoardUno>. [Accessed: 17- Apr- 2022].

- [68] "Semtech SX1278", *semtech.com*, 2022. [Online]. Available: <https://www.semtech.com/products/wireless-rf/lora-core/sx1278>. [Accessed: 17- Apr- 2022].
- [69] "WiFi Module - ESP8266 (4MB Flash) - WRL-17146 - Spark-Fun Electronics", *Sparkfun.com*, 2022. [Online]. Available: <https://www.sparkfun.com/products/17146>. [Accessed: 17- Apr- 2022].
- [70] "Arduino Integrated Development Environment", *Docs.arduino.cc*, 2022. [Online]. Available: <https://docs.arduino.cc/software/ide-v1/tutorials/arduino-ide-v1-basics> [Accessed: 17- Apr- 2022].
- [71] "LoRa® and LoRaWAN®: A Technical Overview", *Lora-developers.semtech.com*, 2022. [Online]. Available: <https://lora-developers.semtech.com/uploads/documents/files/LoRa and LoRaWAN-A Tech Overview-Downloadable.pdf>. [Accessed: 17- Apr- 2022].
- [72] "New Plant Diseases Dataset", *Kaggle.com*, 2021. [Online]. Available: <https://www.kaggle.com/vipooooool/new-plant-diseases-dataset>. [Accessed: 18- Apr- 2022].
- [73] Shorten, C., Khoshgoftaar, T.M. A survey on Image Data Augmentation for Deep Learning. *J Big Data* 6, 60 (2019). <https://doi.org/10.1186/s40537-019-0197-0>.
- [74] S. Sladojevic, M. Arsenovic, A. Anderla, D. Culibrk and D. Stefanovic, "Deep Neural Networks Based Recognition of Plant Diseases by Leaf Image Classification", *Computational Intelligence and Neuroscience*, vol. 2016, pp. 1-11, 2016. Available: 10.1155/2016/3289801.
- [75] N. Elshennawy and D. Ibrahim, "Deep-Pneumonia Framework Using Deep Learning Models Based on Chest X-Ray Images", *Diagnostics*, vol. 10, no. 9, p. 649, 2020. Available: 10.3390/diagnostics10090649.
- [76] Mahdianpari, Masoud and Salehi, Bahram and Rezaee, Mohammad and Mohammadimanesh, Fariba and Zhang, Yun, "Very Deep Convolutional Neural Networks for Complex Land Cover Mapping Using Multispectral Remote Sensing Imagery. Remote Sensing", 2018. 10. 1119. 10.3390/rs10071119.

- [77] S. U. Habiba and M. K. Islam, "Tomato Plant Diseases Classification Using Deep Learning Based Classifier From Leaves Images," *International Conference on Information and Communication Technology for Sustainable Development (ICICT4SD)*, 2021, pp. 82-86, doi: 10.1109/ICICT4SD50815.2021.9396883.
- [78] He K., Zhang X., Ren S., Sun J. (2016) Identity Mappings in Deep Residual Networks, *Leibe B., Matas J., Sebe N., Welling M. (eds) Computer Vision – ECCV 2016. ECCV 2016. Lecture Notes in Computer Science*, vol 9908. Springer, Cham, doi: 10.1007/978-3-319-46493-0-38.
- [79] C. Wang et al., "Pulmonary Image Classification Based on Inception-v3 Transfer Learning Model", *IEEE Access*, vol. 7, pp. 146533-146541, 2019. Available: 10.1109/access.2019.2946000.
- [80] Mari, Kamarasan and Senthilkumar, C, "A novel citrus disease detection and classification using deep learning based inception resnet V2 model", 2020. 9. 1008-1034.
- [81] Seidaliyeva, Ulzhalgas and Akhmetov, Daryn and Ilipbayeva, Lyazzat and Matson, Eric, "Real-Time and Accurate Drone Detection in a Video with a Static Background", *Sensors*, 2020. 20. 3856. 10.3390/s20143856.
- [82] Kingma, Diederik and Ba, Jimmy, "Adam: A Method for Stochastic Optimization", *International Conference on Learning Representations*, 2014. Available: arxiv.org/abs/1412.6980.
- [83] Ruder, Sebastian, "An overview of gradient descent optimization algorithms", *arXiv*, 2016. Available: arxiv.org/abs/1609.04747.
- [84] "Android (operating system) - Wikipedia", En.wikipedia.org, 2022. [Online]. Available: [https://en.wikipedia.org/wiki/Android\(operating system\)](https://en.wikipedia.org/wiki/Android(operating_system)). [Accessed: 22- Apr- 2022].
- [85] "Android Studio 4.0", Android Developers Blog, 2022. [Online]. Available: <https://android-developers.googleblog.com/2020/05/android-studio-4.html>. [Accessed: 22- Apr- 2022].

Appendix A

Transmitter and Receiver Arduino Sketches

A.1 Arduino Sketch of Transmitter Side

```
#include <Wire.h>
#include <SPI.h>
#include <LoRa.h>
#include <DHT.h>

#define DHTPIN 5           //pin where the dht22 is connected
DHT dht(DHTPIN, DHT11);
//MOIST VARIABLES
//int sensorPin = A2;
//int sensorValue;
//int limit = 300;

int SensorPin = A0;
int soilMoistureValue = 0;
int soilmoisturepercent = 0;

//MOTION VARIABLES
#define SPEAKER 6           // the pin that the LED is attached to
int sensor = 3;             // the pin that the sensor is attached to
int state = LOW;            // by default, no motion detected
int val = 0;               // variable to store the sensor status (value)
String motion;
//rain
const int capteur_D = 4; //digital input pin
const int capteur_A = A1; // analog input
String rain;
int rainstatus=0;
```

```

#define relay 7
#define ss 10
#define rst 9
#define dio0 2
// #define ONE_WIRE_BUS 6
// OneWire oneWire(ONE_WIRE_BUS);
// DallasTemperature sensors(&oneWire);

String LoRaMessage = "";
int counter = 0;

const int AirValue = 590; //you need to replace this value with Value_1
const int WaterValue = 300; //you need to replace this value with Value_2

void setup()
{
  Serial.begin(115200);
  pinMode(relay, OUTPUT);

  pinMode(SPEAKER, OUTPUT); // initialize SPEAKER as an output
  pinMode(sensor, INPUT); // initialize sensor as an input
  pinMode(capteur_D, INPUT);
  pinMode(capteur_A, INPUT);
  dht.begin();
  while (!Serial);
  Serial.println("LoRa Sender");
  LoRa.setPins(ss, rst, dio0);
  if (!LoRa.begin(433E6)) {
    Serial.println("Starting LoRa failed!");
    delay(100);
    while (1);
  }
}

void loop()
{
  // MOIST
  soilMoistureValue = analogRead(SensorPin); //put Sensor insert into soil
  soilmoisturepercent = map(soilMoistureValue, AirValue, WaterValue, 0, 100);
  rainstatus = analogRead(capteur_A);
  Serial.print("Soil Moisture Value: ");
  Serial.println(soilMoistureValue);
  Serial.print("Soil Moisture: ");
  Serial.print(soilmoisturepercent);
  Serial.println("%");
  Serial.println("rain");
  Serial.println(rainstatus);
  float h = dht.readHumidity();
  float t = dht.readTemperature();
  // sensors.requestTemperatures();
  // float temp = sensors.getTempCByIndex(0);

```

```

    if (isnan(h) || isnan(t))
    {
        Serial.println("Failed to read from DHT sensor!");
        return;
    }
    Serial.print("Temperature: ");
    Serial.print(t);
    Serial.println(" C ");

    Serial.print("Humidity: ");
    Serial.print(h);
    Serial.println("%");
    Serial.println("");
    Serial.print("Sending packet: ");
    Serial.println(counter);
    LoRaMessage = String(counter) + "/" + String(soilmoisturepercent) + "&" + String(←
        rainstatus)
        /* + "#" + String(temp)*/ + "@" + String(t) + "$" + String(h);
    /*
    //   if ( soilMoistureValue < 400)
    //   {
    //       Serial.println("Soil Moisture level looks good...");
    //       // digitalWrite(relay, HIGH);
    //       Serial.println("Motor turned off");
    //       //WidgetLED PumpLed(V5);
    //       //PumpLed.on();
    //   }
    //   else
    //   {
    //       Serial.println("Plants need water..., notification sent");
    //       // digitalWrite(relay, LOW);
    //       Serial.println("Motor turned on");
    //       //WidgetLED PumpLed(V5);
    //       //PumpLed.off();
    //   }
    //
    */

    // send packet
    LoRa.beginPacket();
    LoRa.print(LoRaMessage);
    LoRa.endPacket();

    counter++;
    if ( soilMoistureValue < 400)
    {
        Serial.println("Soil Moisture level looks good...");
        digitalWrite(relay, HIGH);
        Serial.println("Motor turned off");

    }
    else
    {
        Serial.println("Plants need water..., notification sent");
        digitalWrite(relay, LOW);
    }

```

```

    }
    Serial.println("Analog Value : ");
    //// Serial.println(sensorValue);
    if(digitalRead(capteur_D) == LOW)
    {
        // Serial.println("Digital value : wet");
        Serial.println("Its raining");
    }
else
    {
        //Serial.println("Digital value : dry");
        Serial.println("Its not raining");
    }

    //MOTION
    val = digitalRead(sensor); // read sensor value
    if (val == HIGH) {
        tone(SPEAKER, 100 , 500);
        Serial.println("Motion detected!"); // check if the sensor is HIGH
        digitalWrite(SPEAKER, HIGH); // turn LED ON
        // delay 100 milliseconds

        //if (state == LOW) {
        //    tone(SPEAKER, 100 , 50);
        //    Serial.println("Motion detected!");
        //    state = HIGH; // update variable state to HIGH
        //}
    }

    delay(1500);
}

////

////
// }
// Serial.print("Sending packet: ");
// Serial.println(counter);
//
// if (sensorValue<limit) {
//    digitalWrite(moist, HIGH);
//    Serial.println(" Moter turned off");
// }
// else {
//    digitalWrite(moist, LOW);
//
//    Serial.println(" Moter turned on");
// }

//
// LoRaMessage = String(counter) + "/" + String(sensorValue); // + "&" + String(↵
//    soilmoisturepercent)

```



```

//          // + "#" + String(temp) + "@" + String(t) + "$" + String(h);
//
//  // send packet
//  LoRa.beginPacket();
//  LoRa.print(LoRaMessage);
//  LoRa.endPacket();
//
//  counter++;
//
//  delay(1500);
//}

```

A.2 Arduino Sketch of Receiver Side

```

#include <ESP8266WiFi.h>
#include <FirebaseArduino.h>

#define FIREBASE_HOST "login-c923d-default-rtdb.firebaseio.com" // the ↵
    project name address from firebase id
#define FIREBASE_AUTH "bi2HZKctchTiqc5QZnfpRqTdbMKB0bDhfwEwqkKH" // the secret ↵
    key generated from firebase
#define WIFI_SSID "admin"
#define WIFI_PASSWORD "123456789"

String fireStatus = ""; // led ↵
    status received from firebase
String fireStatusa = "";
int led = D1;

#include <SPI.h>
#include <LoRa.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

#define ss 15
#define rst 16
#define dio0 4

String counter;
String soilmoisturepercent;
String motor_status;
String soiltemp;
String temperature;
String humidity;
String rain_status;
String v;
String m;

```

```

void setup()
{
  Serial.begin(115200);
  delay(1000);
  pinMode(led, OUTPUT);
  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
  Serial.print("Connecting to ");
  Serial.print(WIFI_SSID);
  while (WiFi.status() != WL_CONNECTED)
  {
    Serial.print(".");
    delay(500);
  }
  Serial.println();
  Serial.print("Connected to ");
  Serial.println(WIFI_SSID);
  Firebase.begin(FIREBASE_HOST); // connect to firebase
  Firebase.setString("led_status", "OFF"); //send initial ↵
    string of led status
  Firebase.setString("automatic", "OFF");
  while (!Serial);

  Serial.println("LoRa Receiver");

  LoRa.setPins(ss, rst, dio0);

  if (!LoRa.begin(433E6)) {
    Serial.println("Starting LoRa failed!");
    while (1);
  }
}

void loop()
{
  // try to parse packet
  int pos1, pos2, pos3, pos4, pos5;
  //Serial.print("Receiving packet: ");
  int packetSize = LoRa.parsePacket();
  //Serial.print("ok ");
  if (packetSize)
  {
    // received a packet
    Serial.print("Received packet: ");
    String LoRaData = LoRa.readString();
    Serial.print(LoRaData);
    // read packet
    while (LoRa.available()) {
      Serial.print((char)LoRa.read());
    }
    // print RSSI of packet
    Serial.print(" ' with RSSI ");
    Serial.println(LoRa.packetRssi());

    pos1 = LoRaData.indexOf('/');
  }
}

```

```

pos2 = LoRaData.indexOf('&');
pos3 = LoRaData.indexOf('@');
pos4 = LoRaData.indexOf('$');

counter = LoRaData.substring(0, pos1);
soilmoisturepercent = LoRaData.substring(pos1 + 1, pos2);
rain_status = LoRaData.substring(pos2 + 1, pos3);
//soiltemp = LoRaData.substring(pos3 + 1, pos4);
temperature = LoRaData.substring(pos3 + 1, pos4);
humidity = LoRaData.substring(pos4 + 1, LoRaData.length());

//send data to blynk
//   Blynk.virtualWrite(V1, soilmoisturepercent); //Soil Moisture
//   Blynk.virtualWrite(V2, rain_status); //rain status
//   Blynk.virtualWrite(V3, temperature); // for Temperature
//   Blynk.virtualWrite(V4, humidity); //for Humidity

Serial.print(F("Packet No = "));
Serial.println(counter);
Serial.print(F("Temperature: "));
Serial.print(temperature);
Serial.println(F(" C "));

Serial.print(F("Humidity = "));
Serial.print(humidity);
Serial.println(F("%"));

Serial.print("Soil Moisture percent: ");
Serial.print(soilmoisturepercent);
Serial.println();

fireStatus = Firebase.getString("led_status");
// get ld status input from firebase
fireStatusa = Firebase.getString("automatic");
if(fireStatusa == "OFF"){

if (fireStatus == "OFF")
{
// compare the input of ↵
    led status received from firebase
    Serial.println("Led Turned ON");
    digitalWrite(led, HIGH);
    m="Moter Is Off";
    // make external led ON
}
else if (fireStatus == "ON")
{
// compare the input of led ↵
    status received from firebase
    Serial.println("Led Turned OFF");
    digitalWrite(led, LOW);
    m="Moter Is On";
    // make external led OFF
}
else

```

```

{
  Serial.println("Command Error! Please send ON/OFF");
}}
else if(fireStatusa=="ON"){
  if (soilmoisturepercent.toInt()<60)
  {
    Serial.println("needs water, send notification");
    //send notification
    // Blynk.logEvent("water_your_plants", "Please Water your plants they are about ↵
to die..." ) ;
    Serial.println("Motor is ON");
    // digitalWrite(led, HIGH);
    // m="Moter Is On";
    delay(1000);
  }
  else if (soilmoisturepercent.toInt()>60)
  {
    Serial.println("Soil Moisture level looks good...");
    Serial.println("Motor is OFF");
    //digitalWrite(led, LOW);
    // m="Moter Is Off";
    delay(1000);
  }
}

// rain
Serial.println(rain_status);
if (rain_status.toInt()<200)
{
  Serial.println("its raining");
  //send notification
  // Blynk.logEvent("water_your_plants", "Please Water your plants they are about ↵
to die..." ) ;

  v = "Raining";
  delay(1000);
}
else
{
  Serial.println("its not rainng");
  v = "Not Raining";
  delay(1000);
}
Firebase.setString("Humidity", humidity); //setup path to send ↵
Humidity readings
Firebase.setString("Temperature", temperature); //setup path to send ↵
Temperature readings
Firebase.setString("soilmoisturepercent",soilmoisturepercent);
Firebase.setString("moter_status",m);
Firebase.setString("rain_status",v);
if (Firebase.failed())
{

  Serial.print("pushing /logs failed:");
  Serial.println(Firebase.error());
}

```

```
        return ;  
    }  
}
```

Appendix B

Android Studio Codes

B.1 Back end code of Application login page

```
package com.example.plantonic20;

import androidx.annotation.NonNull;
import androidx.appcompat.app.AppCompatActivity;

import android.app.ProgressDialog;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.view.Window;
import android.view.WindowManager;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;

import com.google.android.gms.tasks.OnCompleteListener;
import com.google.android.gms.tasks.Task;
import com.google.firebase.auth.AuthResult;
import com.google.firebase.auth.FirebaseAuth;
import com.google.firebase.auth.FirebaseUser;

public class MainActivity extends AppCompatActivity {
    TextView createnewAccount;
    EditText inputEmail, inputPassword;
    Button btnlogin;
    String emailPattern = "[a-zA-z0-9._-]+@[a-z]+\\.[a-z]+";
    ProgressDialog progressDialog;

    FirebaseAuth mAuth;
    FirebaseUser mUser;
```

```

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    createNewAccount = findViewById(R.id.createNewAccount);
    getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN, ←
WindowManager.LayoutParams.FLAG_FULLSCREEN);
    inputEmail = findViewById(R.id.inputEmail);
    inputPassword = findViewById(R.id.inputPassword);
    btnLogin = findViewById(R.id.btnLogin);
    progressDialog = new ProgressDialog(this);
    mAuth = FirebaseAuth.getInstance();
    currentUser = mAuth.getCurrentUser();
    createNewAccount.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            startActivity(new Intent(MainActivity.this, RegisterActivity.class));
        }
    });
    btnLogin.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            performLogin();
        }
    });
}

private void performLogin() {
    String email = inputEmail.getText().toString();
    String password = inputPassword.getText().toString();

    if (!email.matches(emailPattern)) {
        inputEmail.setError("Enter correct Email");
    } else if (password.isEmpty() || password.length() < 6) {
        inputPassword.setError("Enter correct Password");
    } else {
        progressDialog.setMessage("Please wait while Login....");
        progressDialog.setTitle("Login");
        progressDialog.setCancelable(false);
        progressDialog.show();
        mAuth.signInWithEmailAndPassword(email, password).addOnCompleteListener(←
new OnCompleteListener<AuthResult>() {
            @Override
            public void onComplete(@NonNull Task<AuthResult> task) {
                if (task.isSuccessful()) {
                    progressDialog.dismiss();
                    sendUserToNextActivity();
                    Toast.makeText(MainActivity.this, "Login Successful", Toast.←
LENGTH_SHORT).show();
                }
                else

```

```

        {
            progressDialog.dismiss();

            Toast.makeText(MainActivity.this, ""+task.getException(), ←
            Toast.LENGTH_SHORT).show();
        }
    }
});
}

private void sendUserToNextActivity() {
    Intent intent=new Intent(MainActivity.this,HomeActivity.class);
    intent.setFlags(Intent.FLAG_ACTIVITY_CLEAR_TASK|Intent.FLAG_ACTIVITY_NEW_TASK←
);
    startActivity(intent);
}
}
}

```

B.2 Back end code of Registration page

```

package com.example.plantonic20;

import androidx.annotation.NonNull;
import androidx.appcompat.app.AppCompatActivity;

import android.app.ProgressDialog;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;

import com.google.android.gms.tasks.OnCompleteListener;
import com.google.android.gms.tasks.Task;
import com.google.firebase.auth.AuthResult;
import com.google.firebase.auth.FirebaseAuth;
import com.google.firebase.auth.FirebaseUser;

public class RegisterActivity extends AppCompatActivity {
    TextView Alreadyhaveaccount;
    EditText inputEmail,inputPassword,inputConfirmPassword;
    Button btnRegister;
    String emailPattern="[a-zA-z0-9._-]+@[a-z]+\.\.[a-z]+";
    ProgressDialog progressDialog;
}

```



```

FirebaseAuth mAuth;
FirebaseUser mUser;
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_register);
    Alreadyhaveaccount=findViewById(R.id.Alreadyhaveaccount);
    inputEmail=findViewById(R.id.inputEmail);
    inputPassword=findViewById(R.id.inputPassword);
    inputConfirmPassword=findViewById(R.id.inputCofirmPassword);
    btnRegister=findViewById(R.id.btnRegister);
    progressDialog=new ProgressDialog(this);
    mAuth=FirebaseAuth.getInstance();
    mUser=mAuth.getCurrentUser();
    Alreadyhaveaccount.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            startActivity(new Intent(RegisterActivity.this,RegisterActivity.class↵
    ));
        }
    });

    btnRegister.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            PerformAthu();
        }
    });
}

private void PerformAthu(){
    String email=inputEmail.getText().toString();
    String password=inputPassword.getText().toString();
    String conpass=inputConfirmPassword.getText().toString();

    if (!email.matches(emailPattern))
    {
        inputEmail.setError("Enter connext Email");
    }
    else if(password.isEmpty() || password.length() < 6)
    {
        inputPassword.setError("Enter correct Password");
    }
    else if (!password.equals(conpass)){
        inputConfirmPassword.setError("Password not match");
    }
    else
    {
        progressDialog.setMessage("Please wait while Registration....");
        progressDialog.setTitle("Registration");
        progressDialog.setCanceledOnTouchOutside(false);
        progressDialog.show();

        mAuth.createUserWithEmailAndPassword(email,password).↵
        addOnCompleteListener(new OnCompleteListener<AuthResult>() {

```

```

        @Override
        public void onComplete(@NonNull Task<AuthResult> task) {
            if (task.isSuccessful()){
                progressDialog.dismiss();
                sendUserToNextActivity();
                Toast.makeText(RegisterActivity.this,"Registration Successful",Toast.LENGTH_SHORT).show();
            }
            else

            {
                progressDialog.dismiss();
                Toast.makeText(RegisterActivity.this, ""+task.getException(),Toast.LENGTH_SHORT).show();
            }
        }
    });
}

private void sendUserToNextActivity(){
    Intent intent=new Intent(RegisterActivity.this,HomeActivity.class);
    intent.setFlags(Intent.FLAG_ACTIVITY_CLEAR_TASK|Intent.FLAG_ACTIVITY_NEW_TASK);
    startActivity(intent);
}
}

```

B.3 Back end code of Navigating through app

```

package com.example.plantonic20;

import androidx.annotation.NonNull;
import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;
import android.os.Bundle;
import android.view.MenuItem;

import com.google.android.material.bottomnavigation.BottomNavigationView;
import com.google.android.material.navigation.NavigationBarView;

public class HomeActivity extends AppCompatActivity {
    BottomNavigationView bottomNavigationView;
    Homefragment homefragment =new Homefragment();
    //MonitorFragment monitorfragment =new MonitorFragment();
    //DetectFragment detectFragment=new DetectFragment();
    //meFragment meFragment= new meFragment();
    @Override
    protected void onCreate(Bundle savedInstanceState) {

```

```

        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_home);
        bottomNavigationView=findViewById(R.id.bottom_navigation);
        getSupportFragmentManager().beginTransaction().replace(R.id.container,↵
homefragment).commit();
        // startActivity(new Intent(HomeActivity.this,WeatherActivity.class));
        bottomNavigationView.setOnItemSelectedListener(new NavigationBarView.↵
OnItemSelectedListener() {
            @Override
            public boolean onNavigationItemSelected( MenuItem item) {
                switch (item.getItemId()){
                    case R.id.home:
                        //getSupportFragmentManager().beginTransaction().replace(R.id↵
.container,homefragment).commit();
                        startActivity(new Intent(HomeActivity.this,WeatherActivity.↵
class));
                        return true;
                    case R.id.monitor:
                        startActivity(new Intent(HomeActivity.this,MoniterActivity.↵
class));
                        return true;
                    case R.id.detect:
                        startActivity(new Intent(HomeActivity.this, DetectActivity.↵
class));
                        return true;
                    case R.id.me:
                        startActivity(new Intent(HomeActivity.this, MainActivity.↵
class));
                        return true;
                }
                return false;
            }
        });
    }
}

```

B.4 Back end code of Weather page

```

package com.example.plantonic20;

import android.content.Intent;
import android.graphics.Color;
import android.os.Bundle;
import android.view.MenuItem;
import android.view.View;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;

```

```

import androidx.appcompat.app.AppCompatActivity;

import com.android.volley.Request;
import com.android.volley.RequestQueue;
import com.android.volley.Response;
import com.android.volley.VolleyError;
import com.android.volley.toolbox.StringRequest;
import com.android.volley.toolbox.Volley;
import com.google.android.material.bottomnavigation.BottomNavigationView;
import com.google.android.material.navigation.NavigationBarView;

import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;

import java.text.DecimalFormat;

public class WeatherActivity extends AppCompatActivity {
    BottomNavigationView bottomNavigationView;

    EditText etCity, etCountry;
    TextView tvResult;
    private final String url = "https://api.openweathermap.org/data/2.5/weather";
    private final String appid = "e53301e27efa0b66d05045d91b2742d3";
    DecimalFormat df = new DecimalFormat("#.##");

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_weather);
        bottomNavigationView = findViewById(R.id.bottom_navigation);
        etCity = findViewById(R.id.etCity);
        etCountry = findViewById(R.id.etCountry);
        tvResult = findViewById(R.id.tvResult);

        bottomNavigationView.setOnItemSelectedListener(new NavigationBarView.OnItemSelectedListener() {
            @Override
            public boolean onNavigationItemSelected(MenuItem item) {
                switch (item.getItemId()) {
                    case R.id.home:
                        //getSupportFragmentManager().beginTransaction().replace(R.id←
                        .container,homefragment).commit();
                        // startActivity(new Intent(WeatherActivity.this, HomeActivity←
                        .class));

                        return true;
                    case R.id.monitor:
                        //getSupportFragmentManager().beginTransaction().replace(R.id←
                        .container, monitorfragment).commit();
                        startActivity(new Intent(WeatherActivity.this, ←
                        MoniterActivity.class));

                        return true;
                    case R.id.detect:

```

```

        startActivity(new Intent(WeatherActivity.this, DetectActivity.↵
        .class));

        return true;
    case R.id.me:
        startActivity(new Intent(WeatherActivity.this, MainActivity.↵
        class));

        return true;
    }
    return false;
}

});
}

public void getWeatherDetails(View view) {
    String tempUrl = "";
    String city = etCity.getText().toString().trim();
    String country = etCountry.getText().toString().trim();
    if (city.equals("")) {
        tvResult.setText("City field can not be empty!");
    } else {
        if (!country.equals("")) {
            tempUrl = url + "?q=" + city + "," + country + "&appid=" + appid;
        } else {
            tempUrl = url + "?q=" + city + "&appid=" + appid;
        }
        StringRequest stringRequest = new StringRequest(Request.Method.POST, ↵
        tempUrl, new Response.Listener<String>() {
            @Override
            public void onResponse(String response) {
                String output = "";
                try {
                    JSONObject jsonResponse = new JSONObject(response);
                    JSONArray jsonArray = jsonResponse.getJSONArray("weather");
                    JSONObject jsonObjectWeather = jsonArray.getJSONObject(0);
                    String description = jsonObjectWeather.getString("description↵
                ");

                    JSONObject jsonObjectMain = jsonResponse.getJSONObject("main↵
                ");

                    double temp = jsonObjectMain.getDouble("temp") - 273.15;
                    double feelsLike = jsonObjectMain.getDouble("feels_like") - ↵
                273.15;

                    float pressure = jsonObjectMain.getInt("pressure");
                    int humidity = jsonObjectMain.getInt("humidity");
                    JSONObject jsonObjectWind = jsonResponse.getJSONObject("wind↵
                ");

                    String wind = jsonObjectWind.getString("speed");
                    JSONObject jsonObjectClouds = jsonResponse.getJSONObject("↵
                clouds");

                    String clouds = jsonObjectClouds.getString("all");
                    JSONObject jsonObjectSys = jsonResponse.getJSONObject("sys");
                    String countryName = jsonObjectSys.getString("country");
                    String cityName = jsonResponse.getString("name");
                    tvResult.setTextColor(Color.rgb(255, 255, 255));
                    tvResult.setTextSize(25);

```

```

        output += "Current weather of " + cityName + " (" + ↵
countryName + ")"

        + "\n"
        + "\n Temp: " + df.format(temp) + " C "
        //+ "\n Feels Like: " + df.format(feelsLike) + " C "
        + "\n Humidity: " + humidity + "%"
        + "\n Description: " + description
        + "\n Wind Speed: " + wind + "m/s"
        + "\n Cloudiness: " + clouds + "%"
        + "\n Pressure: " + pressure + " hPa";

        tvResult.setText(output);
    } catch (JSONException e) {
        e.printStackTrace();
    }
}

}, new Response.ErrorListener() {

    @Override
    public void onErrorResponse(VolleyError error) {
        Toast.makeText(getApplicationContext(), error.toString().trim(), ↵
Toast.LENGTH_SHORT).show();
    }
});

RequestQueue requestQueue = Volley.newRequestQueue(getApplicationContext()↵
());

requestQueue.add(stringRequest);
}
}
}

```

B.5 Back end code of Monitoring page

```

package com.example.plantonic20;

import androidx.annotation.NonNull;
import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;
import android.os.Bundle;
import android.view.MenuItem;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
//import com.pusher.pushnotifications.PushNotifications;
import com.google.android.gms.common.api.Status;
import com.google.android.material.bottomnavigation.BottomNavigationView;
import com.google.android.material.navigation.NavigationBarView;
import com.google.firebase.database.DataSnapshot;
import com.google.firebase.database.DatabaseError;

```

```

import com.google.firebase.database.DatabaseReference;
import com.google.firebase.database.FirebaseDatabase;
import com.google.firebase.database.ValueEventListener;

public class MonitorActivity extends AppCompatActivity {

    Button on;
    Button off;
    Button auto;
    Button manual;
    TextView humidity;
    TextView temp;
    TextView motor;
    TextView rain;
    TextView soil;
    BottomNavigationView bottomNavigationView;
    DatabaseReference dref;
    String status;
    String tstatus;
    String mstatus;
    String rstatus;
    String sstatus;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_monitor);
        // PushNotifications.start(getApplicationContext(), "07e42fab-1aa0-4c60-b076-aaf149da7627");
        // PushNotifications.addDeviceInterest("Its Raining");
        bottomNavigationView = findViewById(R.id.bottom_navigation);
        on = (Button) findViewById(R.id.bon);
        off = (Button) findViewById(R.id.boff);
        auto = (Button) findViewById(R.id.auto);
        manual = (Button) findViewById(R.id.manual);
        off = (Button) findViewById(R.id.boff);
        humidity = (TextView) findViewById(R.id.humidity);
        temp = (TextView) findViewById(R.id.temp);
        motor = (TextView) findViewById(R.id.ms);
        rain = (TextView) findViewById(R.id.rs);
        soil = (TextView) findViewById(R.id.soil);
        bottomNavigationView.setOnItemSelectedListener(new NavigationBarView.OnItemSelectedListener() {
            @Override
            public boolean onNavigationItemSelected(MenuItem item) {
                switch (item.getItemId()) {
                    case R.id.home:
                        //getSupportFragmentManager().beginTransaction().replace(R.id.container, homefragment).commit();
                        startActivity(new Intent(MonitorActivity.this, WeatherActivity.class));
                        return true;
                    case R.id.monitor:
                        //getSupportFragmentManager().beginTransaction().replace(R.id.container, monitorfragment).commit();

```

```

        //startActivity(new Intent(MoniterActivity.this, ↵
MoniterActivity.class));
        return true;
        case R.id.detect:
            startActivity(new Intent(MoniterActivity.this, DetectActivity↵
.class));
            return true;
        case R.id.me:
            //getSupportFragmentManager().beginTransaction().replace(R.id↵
.container, ).commit();
            startActivity(new Intent(MoniterActivity.this, MainActivity.↵
class));
            return true;
    }
    return false;
}

});

dref= FirebaseDatabase.getInstance().getReference();
dref.addValueEventListener(new ValueEventListener() {
    @Override
    public void onDataChange(@NonNull DataSnapshot datasnapshot) {
        status=datasnapshot.child("Humidity").getValue().toString();
        humidity.setText(status);
        tstatus=datasnapshot.child("Temperature").getValue().toString();
        temp.setText(tstatus);
        mstatus=datasnapshot.child("moter_status").getValue().toString();
        motor.setText(mstatus);
        rstatus=datasnapshot.child("rain_status").getValue().toString();
        rain.setText(rstatus);
        sstatus=datasnapshot.child("soilmoisturepercent").getValue().toString↵
());
        soil.setText(sstatus);
    }

    @Override
    public void onCancelled(@NonNull DatabaseError error) {

    }
});
on.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        FirebaseDatabase database = FirebaseDatabase.getInstance();
        DatabaseReference myRef = database.getReference("led_status");

        myRef.setValue("ON");
    }
});
auto.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {

```



```

    });

    off.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            FirebaseDatabase database = FirebaseDatabase.getInstance();
            DatabaseReference myRef = database.getReference("led_status");

            myRef.setValue("OFF");
        }
    });
}
}

```

B.6 Back end code of Disease Diagnosis page

```

package com.example.plantonic20

import android.os.Bundle
import android.widget.Toast
import android.app.Activity
import android.content.Intent
import android.content.pm.ActivityInfo
import android.graphics.Bitmap
import android.graphics.BitmapFactory
import android.graphics.Camera
import android.graphics.Matrix
import android.os.Build
import android.provider.MediaStore
import androidx.annotation.RequiresApi
import androidx.appcompat.app.AppCompatActivity
import android.view.Gravity
import android.view.MenuItem
import com.google.android.material.bottomnavigation.BottomNavigationView
import kotlinx.android.synthetic.main.activity_detect.*
import java.io.IOException

class DetectActivity : AppCompatActivity() {
    // var bottomNavigationView: BottomNavigationView? = null
    // var bottomNavigationView: BottomNavigationView? = null
    var homefragment = Homefragment()
    // var monitorfragment = MonitorFragment()
    // var detectFragment = DetectFragment()
    private lateinit var mClassifier: Classifier
    private lateinit var mBitmap: Bitmap

    private val mCameraRequestCode = 0
    private val mGalleryRequestCode = 2

```

```

private val mInputSize = 224
private val mModelPath = "plant_disease_model.tflite"
private val mLabelPath = "plant_labels.txt"
private val mSamplePath = "soybean.JPG"

@RequiresApi(Build.VERSION_CODES.O)
override fun onCreate(savedInstanceState: Bundle?)
{
    super.onCreate(savedInstanceState)
    requestedOrientation = ActivityInfo.SCREEN_ORIENTATION_PORTRAIT
    setContentView(R.layout.activity_detect)
    // val intent= Intent(this@DetectActivity,WeatherActivity::class.java)
    //startActivity(intent)
    // val i= Intent(this@DetectActivity,HomeActivity::class.java)

    bottom_navigation.setOnItemSelectedListener {
        when (it.itemId) {
            R.id.home ->
                startActivity( Intent(this@DetectActivity,WeatherActivity::class.↵
java))
            R.id.monitor ->
                startActivity( Intent(this@DetectActivity,MoniterActivity::class.↵
.java))
            R.id.me->startActivity( Intent(this@DetectActivity,MainActivity::↵
class.java))
        }
        true
    }

    mClassifier = Classifier(assets, mModelPath, mLabelPath, mInputSize)

    resources.assets.open(mSamplePath).use {
        mBitmap = BitmapFactory.decodeStream(it)
        mBitmap = Bitmap.createScaledBitmap(mBitmap, mInputSize, mInputSize, true↵
)

        mPhotoImageView.setImageBitmap(mBitmap)

    }

    mCameraButton.setOnClickListener {
        val callCameraIntent = Intent(MediaStore.ACTION_IMAGE_CAPTURE)
        startActivityForResult(callCameraIntent, mCameraRequestCode)
    }

    mGalleryButton.setOnClickListener {
        val callGalleryIntent = Intent(Intent.ACTION_PICK)
        callGalleryIntent.type = "image/*"
        startActivityForResult(callGalleryIntent, mGalleryRequestCode)
    }

    mDetectButton.setOnClickListener {
        val results = mClassifier.recognizeImage(mBitmap).firstOrNull()
        mResultTextView.text = "\n Confidence:" + results?.confidence+"\n"+↵
results?.title

    }
}

```

```

override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
    super.onActivityResult(requestCode, resultCode, data)
    if (requestCode == mCameraRequestCode) {
        //Consid rons le cas de la cam ra annul e
        if (resultCode == Activity.RESULT_OK && data != null) {
            mBitmap = data.extras!!.get("data") as Bitmap
            mBitmap = scaleImage(mBitmap)
            val toast = Toast.makeText(
                this,
                ("Image crop to: w= ${mBitmap.width} h= ${mBitmap.height}"),
                Toast.LENGTH_LONG
            )
            toast.setGravity(Gravity.BOTTOM, 0, 20)
            toast.show()
            mPhotoImageView.setImageBitmap(mBitmap)
            mResultTextView.text = "Your photo image set now."
        } else {
            Toast.makeText(this, "Camera cancel..", Toast.LENGTH_LONG).show()
        }
    } else if (requestCode == mGalleryRequestCode) {
        if (data != null) {
            val uri = data.data

            try {
                mBitmap = MediaStore.Images.Media.getBitmap(this.↵
contentResolver, uri)
            } catch (e: IOException) {
                e.printStackTrace()
            }

            println("Success!!!")
            mBitmap = scaleImage(mBitmap)
            mPhotoImageView.setImageBitmap(mBitmap)

        }
    } else {
        Toast.makeText(this, "Unrecognized request code", Toast.LENGTH_LONG).↵
show()
    }
}

fun scaleImage(bitmap: Bitmap?): Bitmap {
    val originalWidth = bitmap!!.width
    val originalHeight = bitmap.height
    val scaleWidth = mInputSize.toFloat() / originalWidth
    val scaleHeight = mInputSize.toFloat() / originalHeight
    val matrix = Matrix()
    matrix.postScale(scaleWidth, scaleHeight)
    return Bitmap.createBitmap(bitmap, 0, 0, originalWidth, originalHeight, ↵
matrix, true)
}

```

}