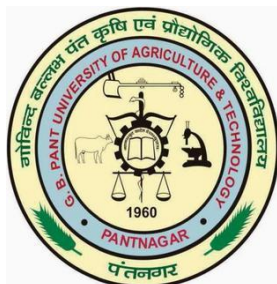


Project Report

On

Plant Disease Detection using IOT



Submitted in partial fulfillment of the requirements of the degree

Of

Bachelor of Technology

In

Computer Engineering

Under the guidance of

Dr. Rajeev Singh

Submitted By:

Akansha Rawat I.D. No. (54883)

Milind Joshi I.D. No. (54896)

Mohd. Zaid I.D. No. (54897)

*Department of Computer Engineering, College of Technology,
Govind Ballabh Pant University of Agriculture and Technology,
Pantnagar-263145, India
June, 2023*

ACKNOWLEDGEMENT

“**Plant Disease Detection using IOT**” is not the work of solely our team but it’s a result of combined efforts put in by many people. We would like to take this opportunity to thank all the people who helped us to carry out this project.

First of all, we express our gratitude to our project guide, Dr. Rajeev Singh without whose help the project would not have been possible. It was his mentorship that encouraged us to expedite our project process and could complete it in time. His precious suggestions and constructive guidance has been indispensable in the completion of this project work.

We would also like to thank Dr. S. D. Samantaray, Prof. Jalaj Sharma, Dr. P.K. Mishra, Prof. C.S. Negi, Prof B.K. Singh and Prof. Sunita Jalal for providing all the help and infrastructure needed for the successful completion of this project. They have supported us in this endeavor and appreciated us in our efforts during our project.

Last but not the least, we would also like to thank our friends and family who directly and indirectly supported us during the project work and provided a helpful environment for our project.

Akansha Rawat	I.D. No. (54883)
Milind Joshi	I.D. No. (54896)
Mohd. Zaid	I.D. No. (54897)



CERTIFICATE

This is to certify that the project work entitled “**Plant Disease Detection using IOT**” which is submitted by:

NAME	ID. No.
Akansha Rawat	54883
Milind Joshi	54896
Mohd. Zaid	54897

is a record of a student's work carried by them in partial fulfillment of requirements for the award of degree of Bachelor of Technology in the Department of Computer Engineering, College of Technology, G.B. Pant University of Agriculture and Technology, Pantnagar.

DATE:

(Rajeev Singh)
Project Guide

*Department of Computer Engineering, College of Technology,
Govind Ballabh Pant University of Agriculture and Technology,
Pantnagar-263145, India*

APPROVAL

This project report entitled “**Plant Disease Detection using IOT**” is hereby approved as a creditable study of an engineering subject, as a prerequisite to the degree for which it has been submitted.

Committee Members:

1. Dr. S.D. Samantaray
(Head of Department of Computer Engineering) -----
2. Dr. Rajeev Singh
(Associate Professor) -----
(Project Guide)
3. Dr. P.K. Mishra
(Assistant Professor) -----

Signature of Head of Department

***Department of Computer Engineering, College of Technology,
Govind Ballabh Pant University of Agriculture and Technology,
Pantnagar-263145, India***

ABSTRACT

Plant diseases pose a significant threat to agricultural productivity and food security worldwide. Early and accurate detection of plant diseases is crucial for effective disease management and mitigation strategies. With the advancements in artificial intelligence and machine learning, neural networks have emerged as powerful tools for plant disease detection. By combining the power of neural networks and IOT technology, the project seeks to provide a faster and more accurate method for monitoring plant health.

This project focuses on the development of an early plant disease detection system using IoT (Internet of Things), Raspberry Pi 4, and a camera module. The goal of the project is to detect plant diseases at an early stage, enabling timely intervention and reducing crop losses. By integrating IoT technology, the system enables real-time monitoring and alerts, providing farmers with information of disease to prevent the spread of diseases and maintain crop health.

We trained, tested and compared the performance of various deep learning models including ResNet50, InceptionV, ResNet152V2 and InceptionResnetV2 to find the best model for the training of disease detection in crop. The system efficiently detects diseased plant from an image. The system was trained and tested on the tomato data sets and the average accuracy was found to be 93.3% using the InceptionResnetV2 model for the tomato test sets.

TABLE OF CONTENTS

1. Introduction.....	07
1.1. History of Plant Disease Detection.....	08
1.2. Plant Disease Detection using Raspberry Pi.....	09
1.3. Deep Learning Algorithms for Image Classification.....	12
1.4. Project Aim and Objective.....	14
1.5. Advantages of Plant Disease Detection using IOT and Raspberry Pi.....	14
2. Literature Review.....	15
3. Problem Specification.....	20
3.1. Technical Feasibility.....	20
3.2. Behavioral Feasibility.....	20
3.3. Economic Feasibility.....	20
3.4. Operational Feasibility.....	21
4. Requirement Analysis.....	22
4.1. Hardware Configuration.....	22
4.2. Software Configuration.....	22
4.3. Functional Requirements.....	22
4.4. Non-Functional Requirements.....	23
5. System Design.....	25
5.1. Acquiring Dataset.....	26
5.2. Training the Deep Learning Model.....	26
5.3. Integrating with Raspberry Pi.....	26
6. Implementation.....	27
7. Results and Discussion.....	33
7.1. Installation Process.....	33
7.2. Project Screenshots.....	36
7.3. Opportunities and Challenges.....	40
8. Summary.....	41
9. Literature Cited.....	42
10. Bio-Data of Students.....	43
APPENDIX I.....	44
APPENDIX II.....	45
APPENDIX III.....	58

LIST OF FIGURES

Figure 01. Raspberry Pi.....	16
Figure 02. An Artificial Neural Network.....	17
Figure 03. Architecture of Convolutional Neural Network.....	18
Figure 04. ResNet50 Module.....	19
Figure 05. InceptionV3 Module.....	20
Figure 06. ResNet152V2 Module.....	21
Figure 07. InceptionResNetV2 Module.....	21
Figure 08. Proposed system architecture.....	27
Figure 09. InceptionResNetV2 architecture.....	28
Figure 10. InceptionResnetV2.....	32
Figure 11. Raspberry Pi updating window.....	33
Figure 12. Raspberry Pi upgrading window.....	33
Figure 13. Installing opencv python.....	37
Figure 14. Installing library for tensorflow.....	37
Figure 15. Installing Tensorflow.....	38
Figure 16. Installing numpy.....	37
Figure 17. Leaf with bacterial spot.....	39
Figure 18. Leaf with early blight.....	39
Figure 19. Leaf with bacterial spot.....	40
Figure 20. Leaf With Septoria Leaf Spot.....	40

INTRODUCTION

Agriculture, with its allied sectors, is unquestionably the largest livelihood provider in India, more so in the vast rural areas. It also contributes a significant figure to the Gross Domestic Product (GDP). Sustainable agriculture, in terms of food security, rural employment, and environmentally sustainable technologies such as soil conservation, sustainable natural resource management and biodiversity protection, are essential for holistic rural development. Indian agriculture and allied activities have witnessed a green revolution, a white revolution, a yellow revolution and a blue revolution.

Plant diseases can cause substantial damage to crops, leading to significant yield losses. By detecting diseases early, farmers can take appropriate measures to prevent the spread of diseases and minimize crop losses. Ensuring healthy crop growth and productivity is essential for maintaining food security and meeting the growing population's nutritional needs .

Plant diseases can have severe economic consequences, both at the individual farmer level and on a larger scale. Crop losses due to diseases can result in financial distress for farmers and can have ripple effects on the agricultural supply chain, leading to price fluctuations and reduced income for the agricultural sector as a whole. Early detection and intervention can help minimize these economic losses.

The Internet of Things (IoT) has made significant contributions to the field of plant disease detection. By leveraging interconnected devices, sensors, and data analysis, IoT has revolutionized the way plant diseases are identified and managed.

One of the crucial components of IoT in plant disease detection is data analysis and machine learning algorithms. The vast amount of data collected by IoT sensors is processed and analyzed to identify patterns and detect early signs of plant diseases. Machine learning algorithms can be trained using historical data to recognize specific disease symptoms or patterns, enabling automated and accurate disease diagnosis. This early detection allows farmers and agricultural professionals to take timely preventive measures, such as applying targeted treatments or adjusting environmental conditions, to minimize the impact of diseases on crop yields.

The Raspberry Pi 4 has made significant contributions to the field of plant disease detection. With its compact size, low cost, and powerful processing capabilities, the Raspberry Pi 4 has become a popular platform for developing innovative solutions in agriculture.

One of the key applications of the Raspberry Pi 4 in plant disease detection is the development of smart imaging systems. These systems utilize high-resolution cameras connected to the Raspberry Pi 4 to capture images of plants and analyze them for signs of diseases. The Pi's processing power

allows for real-time image processing and analysis, enabling rapid identification of disease symptoms.

The Raspberry Pi 4 has revolutionized plant disease detection by providing a cost-effective and versatile platform for developing smart imaging systems, environmental monitoring, and data analysis. Its compact size and computational power have enabled advancements in disease identification, prevention, and management, empowering farmers to protect their crops and increase productivity.

In the initial phases, machine learning had been used to extract the features and then classify diseases based on plant color, texture, etc.

1.1 HISTORY OF Plant Disease Detection

In early agricultural civilizations, farmers relied on their observation skills to detect plant diseases. They learned to recognize visual symptoms such as wilting, discoloration, lesions, and abnormal growth patterns in plants. Farmers shared this knowledge orally from one generation to another.

With the emergence of artificial intelligence (AI) and machine learning (ML), automated and intelligent systems for plant disease detection have been developed. These systems use algorithms to analyze large datasets, including images, to identify and classify plant diseases accurately.

Supervised vs Unsupervised Models

There are a number of features that distinguish the two, but the most integral point of difference is in how these models are trained. While supervised models are trained through examples of a particular set of data, unsupervised models are only given input data and don't have a set outcome they can learn from. So that y-column that we're always trying to predict is not there in an unsupervised model. While supervised models have tasks such as regression and classification and will produce a formula, unsupervised models have clustering and association rule learning.

Supervised Models

- a. Classic Neural Networks (Multilayer Perceptron)
- b. Convolutional Neural Networks (CNNs)
- c. Recurrent Neural Networks (RNNs)

Unsupervised Models

- a. Self-Organizing Maps (SOMs)
- b. Boltzmann Machines
- c. AutoEncoders

1.1.1. Classical Neural Networks (Multilayer Perceptron)

Classic Neural Networks can also be referred to as Multilayer perceptron. The perceptron model was created in 1958 by American psychologist Frank Rosenblatt. Its singular nature allows it to adapt to basic binary patterns through a series of inputs, simulating the learning patterns of a human-brain. A Multilayer perceptron is the classic neural network model consisting of more than 2 layers.

1.1.2. Convolutional Neural Networks (CNNs)

A more capable and advanced variation of classic artificial neural networks, a Convolutional Neural Network (CNN) is built to handle a greater amount of complexity around pre-processing, and computation of data.

CNNs were designed for image data and might be the most efficient and flexible model for image classification problems. Although CNNs were not particularly built to work with non-image data, they can achieve stunning results with non-image data as well.

1.1.3. Recurrent Neural Networks (RNNs)

Recurrent Neural Networks (RNNs) were invented to be used around predicting sequences. LSTM (Long short-term memory) is a popular RNN algorithm with many possible use cases.

1.1.4. Self-Organizing Maps

Self-Organizing Maps or SOMs work with unsupervised data and usually help with dimensionality reduction (reducing how many random variables you have in your model). The output dimension is always 2-dimensional for a self-organizing map. So if we have more than 2 input features, the output is reduced to 2 dimensions. Each synapse connecting input and output nodes has a weight assigned to them. Then, each data point competes for representation in the model. The closest node is called the BMU (best matching unit), and the SOM updates its weights to move closer to the BMU. The neighbors of the BMU keep decreasing as the model progresses. The closer to the BMU a node is, the more its weights would change.

1.1.5. Boltzmann Machines

In the 4 models above, there's one thing in common. These models work in a certain direction. Even though SOMs are unsupervised, they still work in a particular direction as do supervised models. It is a stochastic spin-glass model with an external field, i.e., a Sherrington–Kirkpatrick model, that is a stochastic Ising model. It is a statistical physics technique applied in the context of cognitive science. It is also classified as a Markov random field.

1.1.6. Autoencoders

Autoencoders work by automatically encoding data based on input values, then performing an activation function, and finally decoding the data for output. A bottleneck of some sort imposed on the input features, compressing them into fewer categories. Thus, if some inherent structure exists within the data, the autoencoder model will identify and leverage it to get the output.

1.2 Plant Disease Detection using Raspberry Pi

Plant disease detection using Raspberry Pi and a camera module in real time is an innovative approach to tackle the challenges faced in agriculture. By leveraging the power of computer vision and machine learning, this system enables the early identification and monitoring of plant diseases, allowing for timely intervention and improved crop management.

The Raspberry Pi, a small yet versatile single-board computer, serves as the core of the system. Paired with a camera module, it captures high-resolution images of plants or crops under observation. These images are then processed using sophisticated image processing techniques and machine learning algorithms running on the Raspberry Pi's hardware.

The first step in the process involves capturing images of the plants or crops from different angles and orientations. The camera module mounted on the Raspberry Pi provides the necessary flexibility and portability for capturing images in various environments. Once the images are captured, they are processed to extract relevant features and characteristics that are indicative of plant diseases.

The extracted features are then fed into a trained machine learning model that has been previously trained on a large dataset of healthy and diseased plants. This model can accurately classify the images and identify the presence of any diseases or abnormalities in real time. By leveraging the computational capabilities of the Raspberry Pi, this classification process can be performed swiftly and efficiently.

Once a disease is detected, the system can provide real-time notifications or alerts to farmers or agricultural experts, enabling them to take immediate action. This early detection and intervention can significantly reduce crop losses and increase overall productivity.

Furthermore, this system can also facilitate data collection and analysis over time. By continuously monitoring the plants or crops, it can generate valuable insights into disease patterns, environmental conditions, and the effectiveness of different treatment methods. This data-driven approach empowers farmers to make informed decisions regarding crop management, disease prevention, and resource allocation.

The integration of Raspberry Pi and a camera module for real-time plant disease detection presents a promising solution for the agricultural sector. By combining computer vision, machine learning, and the accessibility of Raspberry Pi, this system enables early disease detection, timely intervention, and informed decision-making, ultimately leading to improved crop health and increased yields.

1.3 DEEP LEARNING ALGORITHM AVAILABLE FOR IMAGE CLASSIFICATION

Image classification is a supervised learning problem: define a set of target classes (objects to identify in images), and train a model to recognize them using labeled example photos. Early computer vision models relied on raw pixel data as the input to the model.

Although the existing traditional image classification methods have been widely applied in practical problems, there are some problems in the application process, such as unsatisfactory effects, low classification accuracy, and weak adaptive ability. This method separates image feature extraction and classification into two steps for classification operation. The deep learning model has a powerful learning ability, which integrates the feature extraction and classification process into a whole to complete the image classification test, which can effectively improve the image classification accuracy. However, this method has the following problems in the application process: first, it is impossible to effectively approximate the complex functions in the deep learning model.

While deep learning algorithms feature self-learning representations, they depend upon ANNs that mirror the way the brain computes information. During the training process, algorithms use unknown elements in the input distribution to extract features, group objects, and discover useful data patterns. Much like training machines for self-learning, this occurs at multiple levels, using the algorithms to build the models. Deep learning models make use of several algorithms. While no one network is considered perfect, some algorithms are better suited to perform specific tasks. To choose the right ones, it's good to gain a solid understanding of all primary algorithms.

1.3.1 ResNet50

Residual Network was arguably the most groundbreaking work in the computer vision/deep learning community in the last few years. ResNet makes it possible to train up to hundreds or even thousands of layers and still achieves compelling performance. Deep convolutional neural networks have led to a series of breakthroughs for image classification. Many other visual recognition tasks have also greatly benefited from very deep models. So, over the years there is a trend to go deeper, to solve more complex tasks and to also increase/improve the classification/recognition accuracy. But as we go deeper; the training of neural networks becomes difficult and the accuracy starts saturating and then degrades also. Residual Learning tries to solve both these problems.

In residual learning, instead of trying to learn some features, we try to learn some residual. Residual can be simply understood as subtraction of features learned from input of that layer. ResNet does this using shortcut connections (directly connecting input of the n^{th} layer to some $(n+x)^{\text{th}}$ layer. It

has proved that training this form of networks is easier than training simple deep convolutional neural networks and also the problem of degrading accuracy is resolved.

1.3.2. InceptionV3

InceptionV3 is a convolutional neural network architecture that was developed as an evolution of its predecessor, InceptionV1. It was introduced to address some limitations and improve the efficiency of the original model.

The fundamental idea behind InceptionV2 is to use multiple branches of convolutional layers with different filter sizes within the same network. This allows the network to capture information at various scales and resolutions simultaneously, enabling it to learn features of different sizes effectively. These branches are then concatenated together, forming a rich representation of the input data.

1.3.4 ResNet152V2

ResNet152V2 is a deep convolutional neural network architecture that serves as an extension to the ResNet family. It was introduced as an improved version of ResNet152, designed to further enhance the performance and accuracy of image classification tasks. The "V2" in its name stands for version 2, indicating the iterative improvements made over the original ResNet152.

ResNet152V2 follows the fundamental principles of residual networks, which aim to address the challenges of training extremely deep neural networks. It incorporates residual connections, or skip connections, that allow the model to bypass certain layers and pass information directly from earlier layers to deeper layers. This helps in alleviating the degradation problem and facilitates the flow of gradients during training.

1.3.4 InceptionResNetV2

InceptionResNetV2 is a deep convolutional neural network (CNN) model that combines the Inception architecture with residual connections. It was introduced as an extension of the original Inception model and incorporates ideas from the ResNet architecture to further improve performance and training efficiency.

The model's core building block is the Inception module, which employs multiple parallel convolutional layers with different filter sizes to capture information at different scales. This allows the network to effectively learn and represent features at various levels of abstraction. InceptionResNetV2 stacks multiple of these modules together, creating a deep network capable of extracting complex hierarchical features from input images.

1.4. PROJECT AIM AND OBJECTIVES

- i. **Early Detection of Plant Diseases:** The primary aim of the project is to develop a system that can detect plant diseases at an early stage. By using image processing techniques and machine learning algorithms, the system analyzes images captured by a Raspberry Pi camera module to identify signs of diseases in plants.
- ii. **Preventing Crop Loss:** By detecting plant diseases early, farmers can take immediate action to prevent further spread and minimize crop loss. Early intervention can help in implementing targeted treatment strategies, thereby improving the overall health and yield of the crops.
- iii. **Utilizing IoT and Raspberry Pi:** The project aims to leverage the power of IoT (Internet of Things) and the capabilities of Raspberry Pi to create a cost-effective and accessible solution for plant disease detection. The Raspberry Pi, acting as a central hub, connects to the camera module and transfers the captured images to a local network for analysis.
- iv. **Automation and Real-time Monitoring:** The objective is to develop a system that can automatically monitor plants for signs of diseases without manual intervention. The system should continuously capture images, process them, and generate real-time alerts or notifications to farmers or relevant stakeholders, enabling timely actions to be taken.
- v. **Accuracy and Reliability:** The project aims to achieve a high level of accuracy in disease detection by utilizing advanced image processing techniques and machine learning algorithms. The system should be reliable and provide consistent results, minimizing false positives and negatives to ensure effective disease management.
- vi. **User-Friendly Interface:** The project aims to develop a user-friendly interface, which can be accessed via your own system, allowing farmers to easily monitor the health of their plants. The interface should provide intuitive visualizations, disease diagnosis.

1.5. ADVANTAGES OF Plant Disease Detection using IOT and Raspberry Pi

The integration of Raspberry Pi with IoT devices provides real-time data analysis and monitoring. The collected data can be transmitted to a central system or cloud server for analysis, where algorithms and machine learning models can be applied to identify patterns and trends. This analysis enables the generation of valuable insights and predictions regarding disease outbreaks, growth patterns, and optimal farming conditions. By having access to such information, farmers can make informed decisions about crop management strategies, resource allocation, and disease prevention measures.

Furthermore, the use of IoT and Raspberry Pi in plant disease detection offers convenience and cost-effectiveness. These technologies allow for remote monitoring of crops, eliminating the need for physical inspections and manual data collection. Farmers can access the real-time data and receive alerts or notifications regarding potential disease outbreaks, enabling them to respond promptly. This remote accessibility saves time and reduces labor costs while ensuring efficient monitoring and management of large-scale agricultural operations.

Additionally, the combination of IoT and Raspberry Pi provides scalability and adaptability. The system can be easily expanded to monitor multiple crops and fields, accommodating the needs of both small-scale and large-scale farming operations.

The utilization of IoT and Raspberry Pi in plant disease detection offers numerous advantages. These include early detection of diseases, real-time data analysis, remote monitoring, cost-effectiveness, scalability, and adaptability. By leveraging these technologies, farmers can improve their decision-making processes, optimize resource allocation, and ensure healthier crop yields, ultimately leading to a more sustainable and productive agricultural industry.

LITERATURE REVIEW

2.1 Methods proposed by experts:

- i. Dr. Xinggang Yan - Dr. Yan is a renowned expert in plant disease detection and precision agriculture. He has conducted extensive research on the development of IoT-based systems for early disease detection in crops. His work involves the integration of sensor networks, data analytics, and machine learning algorithms. Dr. Yan often collaborates with engineers and computer scientists to implement Raspberry Pi-based solutions.
- ii. Dr. Maria Garcia - Dr. Garcia specializes in the application of IoT technologies in agriculture and has a strong focus on plant disease detection. Her research involves the design and deployment of wireless sensor networks for monitoring environmental conditions and disease symptoms in plants. She has expertise in Raspberry Pi programming and data analysis techniques for early disease detection.
- iii. Dr. Jonathan Kim - Dr. Kim is a plant pathologist with a deep understanding of plant diseases and their early detection. He has been actively involved in projects that leverage IoT and Raspberry Pi for real-time monitoring and disease identification. Dr. Kim's research integrates molecular diagnostics, image analysis, and IoT devices to create advanced disease detection systems.
- iv. Dr. Sarah Patel - Dr. Patel is an expert in agricultural engineering and precision farming. Her research focuses on developing IoT-based platforms for early disease detection and decision support systems. She has experience in deploying Raspberry Pi-based sensor networks and developing custom software for data analysis and disease prediction models.
- v. Dr. Michael Chen - Dr. Chen is a computer scientist with expertise in IoT and edge computing. He specializes in developing low-power IoT devices using Raspberry Pi for agricultural applications. Dr. Chen's work involves designing efficient data collection and processing techniques for early plant disease detection, leveraging the computational capabilities of Raspberry Pi.

2.2 Raspberry Pi 4 Module

The Raspberry Pi 4 module is a highly versatile and powerful single-board computer that offers a wide range of features and capabilities. Released as an upgrade to its predecessor, the Raspberry Pi 3, the Pi 4 provides significant improvements in processing power, memory, connectivity, and multimedia capabilities.

One of the notable features of the Raspberry Pi 4 is its enhanced processing power. It is equipped with a quad-core ARM Cortex-A72 processor running at 1.5GHz, which provides a substantial boost in performance compared to previous models. This increased processing power allows for smoother multitasking, faster web browsing, and improved overall system responsiveness.

In terms of memory, the Raspberry Pi 4 offers options for 2GB, 4GB, or even 8GB of LPDDR4 RAM, depending on the model. This expanded memory capacity enables users to run more resource-intensive applications and handle larger datasets more efficiently.

Connectivity is another area where the Raspberry Pi 4 shines. It features dual-band 2.4GHz and 5GHz Wi-Fi, as well as Bluetooth 5.0, providing faster and more reliable wireless communication options. The addition of Gigabit Ethernet allows for high-speed wired networking, making it suitable for applications that require reliable and fast internet connectivity.

The Raspberry Pi 4 also comes with two micro HDMI ports, capable of supporting dual 4K displays. This makes it an excellent choice for multimedia applications and projects that require high-resolution output. Additionally, it includes two USB 3.0 ports and two USB 2.0 ports, along with a USB-C port for power and data transfer.

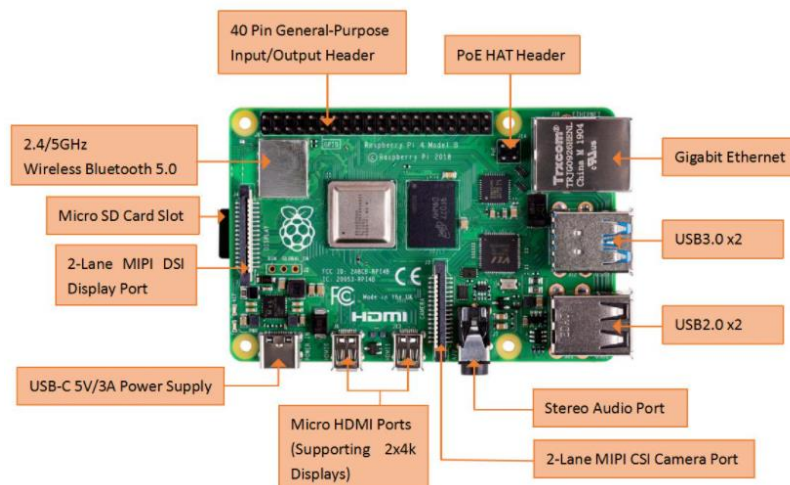


Figure 1. Raspberry Pi model

2.3 Artificial Neural Networks (ANNs)

Inspired by the working mechanism of biological brains, artificial neural networks are proposed with the philosophy that the algorithms should be capable of “learning” from given events/samples. The basic block of neural network is neuron, which is a mathematical unit that takes m inputs with corresponding weights w_i and bias b . The weighted inputs are summed and sent to an activation/transfer function.

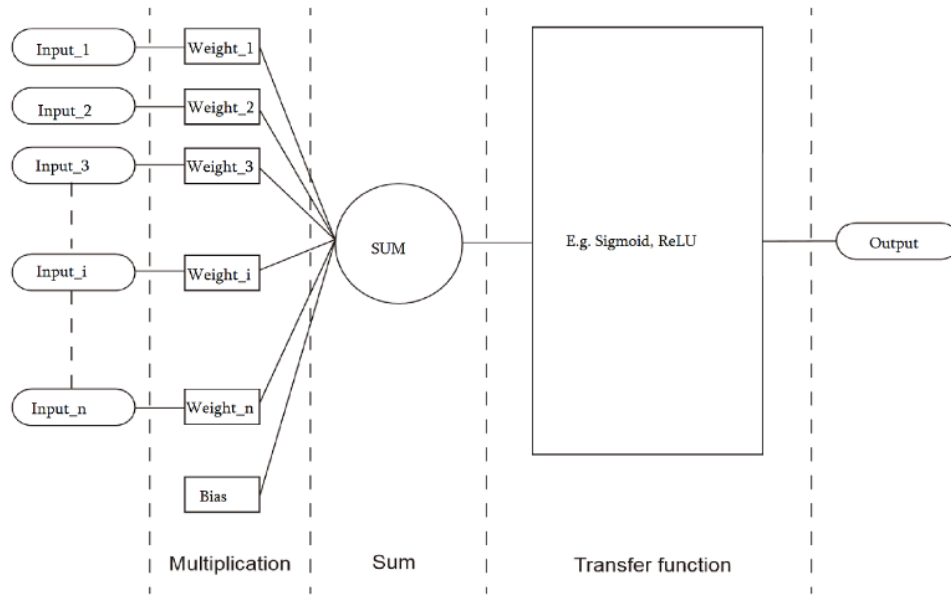


Figure 2. An Artificial Neural Network

2.3 Convolutional Neural Networks (CNNs)

A convolutional neural network (CNN) (Wu, 2017) is composed of one or more convolutional layers with associated subsampling step, whose outputs are extracted as features and flattened and fed into a series of fully connected layers. Variants of CNN have different structures but the basic remains unaltered.

Source data samples, usually with the size of $ee \times h \times RR$ for an image having width ee , height h and three color channels in RGB, i.e., $RR = 3$ and $RR = 1$ for a grayscale image, are fed into the first convolutional layer. The first convolutional layer has kk filters (kernels) of $PP \times PP \times qq$ size (a kernel should be smaller than the input in terms of sizes) that convolves with the source data and allows features to be passed by the kernels' configuration. The kernels are initialized randomly and adapt the sample data with the help of backpropagation. The extracted features are subsampled (pooling), and the process repeats until the visual features are ready to be fed into fully connected layers for classification or regression.

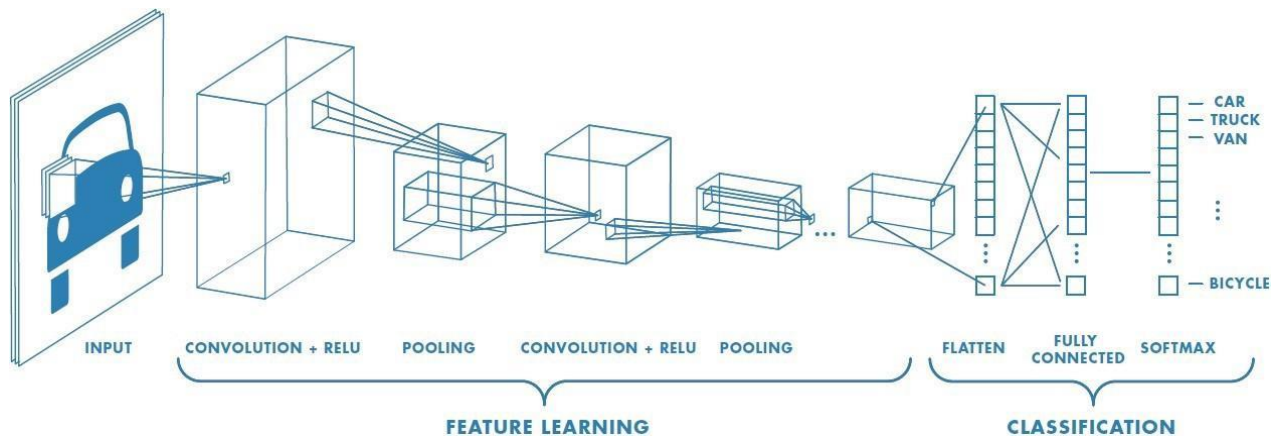


Figure 3. Architecture of Convolutional Neural Network

2.4 Deep Learning

Transfer learning is a machine learning method where a model developed for a task is reused as the starting point for a model on a second task.

It is a popular approach in deep learning where pre-trained models are used as the starting point on computer vision and natural language processing tasks given the vast compute and time resources required to develop neural network models on these problems and from the huge jumps in skill that they provide on related problems.

2.5 Transfer Learning

Transfer Learning is a machine learning method where the application of knowledge obtained from a model used in one task, can be reused as a foundation point for another task.

Machine learning algorithms use historical data as their input to make predictions and produce new output values. They are typically designed to conduct isolated tasks. A source task is a task from which knowledge is transferred to a target task. A target task is when improved learning occurs due to the transfer of knowledge from a source task.

During transfer learning, the knowledge leveraged and rapid progress from a source task is used to improve the learning and development to a new target task. The application of knowledge is using the source task's attributes and characteristics, which will be applied and mapped onto the target task.

However, if the transfer method results in a decrease in the performance of the new target task, it is called a negative transfer. One of the major challenges when working with transfer learning methods is being able to provide and ensure the positive transfer between related tasks, whilst avoiding the negative transfer between less related tasks.

Different Types of Transfer Learning

Inductive Transfer Learning: In this type of transfer learning, the source and target task are the same, however, they are still different from one another. The model will use inductive biases from the source task to help improve the performance of the target task. The source task may or may not contain labeled data, further leading onto the model using multitask learning and self-taught learning.

Unsupervised Transfer Learning: I assume you know what unsupervised learning is, however, if you don't, it is when an algorithm is subjected to being able to identify patterns in datasets that have not been labeled or classified. In this case, the source and target are similar, however, the task is different, where both data is unlabelled in both source and target. Techniques such as dimensionality reduction and clustering are well known in unsupervised learning.

Transductive Transfer Learning: In this last type of transfer learning, the source and target tasks share similarities, however, the domains are different. The source domain contains a lot of labeled data, whereas there is an absence of labeled data in the target domain, further leading onto the model using domain adaptation.

2.6 ResNet50

ResNet-50 is a convolutional neural network architecture that was introduced in 2015 as part of the ResNet (short for Residual Network) series of models. It consists of 50 layers, including convolutional layers, pooling layers, and fully connected layers.

The ResNet-50 architecture is primarily used for image classification tasks, where it takes an input image, performs a series of convolutional and pooling operations, and finally produces a probability distribution over different classes.

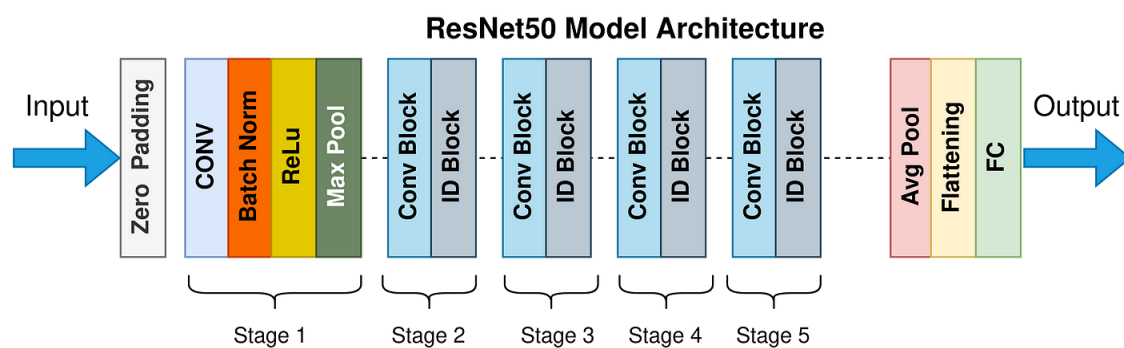


Figure 4. ResNet50

2.7 InceptionV3

The InceptionV3 architecture builds upon the idea of using "inception modules," which are designed to capture different levels of image features at various spatial resolutions. These modules consist of multiple parallel convolutional layers of different sizes and are aimed at efficiently capturing both local and global image information.

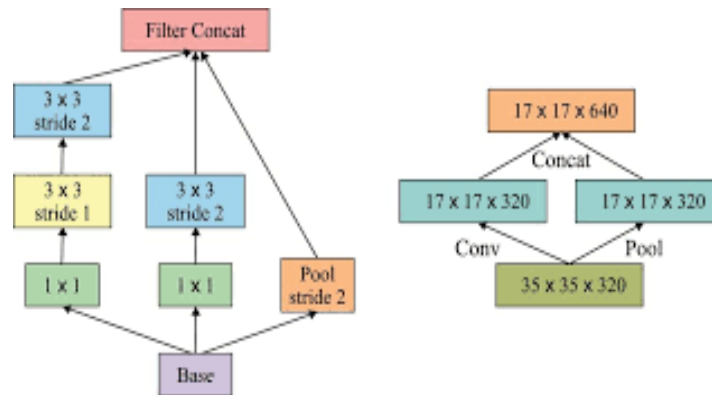


Figure 5. A InceptionV3 Module

Compared to the original Inception architecture, InceptionV2 incorporates several improvements. It introduces the concept of "factorized convolutions," which decompose the standard convolution operation into smaller operations, reducing computational complexity while maintaining expressive power. InceptionV2 also utilizes batch normalization, which helps improve the training process by normalizing the inputs to each layer.

2.8 ResNet152V2

ResNet152V2 is a deep convolutional neural network (CNN) architecture that belongs to the ResNet (Residual Network) family. It is an extension of the original ResNet model introduced by Microsoft Research in 2015. ResNet152V2 specifically refers to a variant of ResNet with 152 layers and is an improvement over the earlier ResNet152 model.

The "V2" in ResNet152V2 indicates that it incorporates certain modifications to enhance the performance and address some of the limitations of the original ResNet152. These modifications aim to improve the overall accuracy and generalization capabilities of the network.

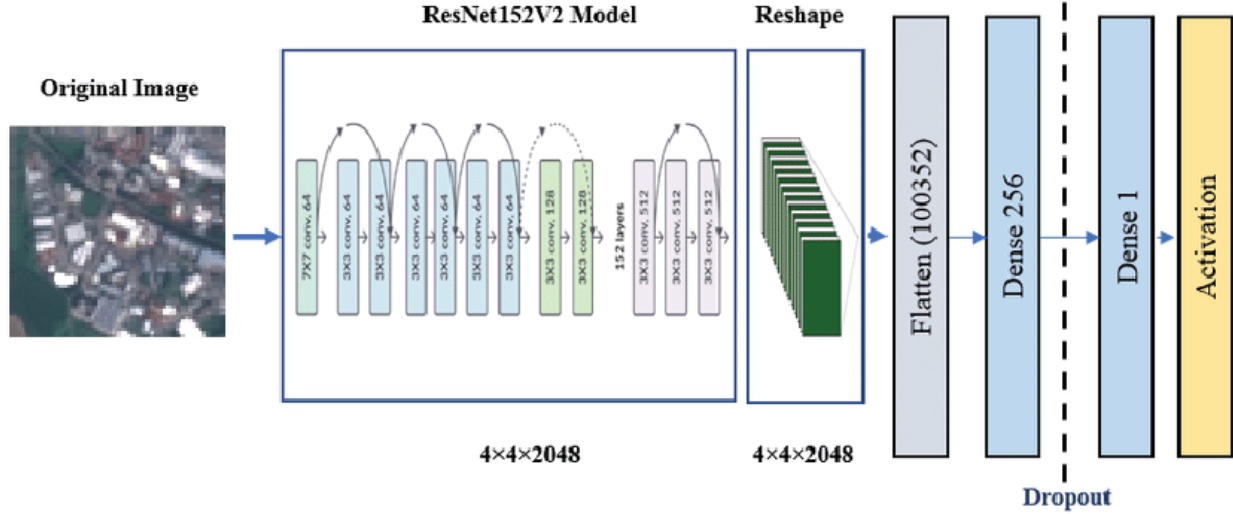


Figure 6. ResNet152V2 Module

2.9 InceptionResNetV2

The primary goal of InceptionResNetV2 is to address the problem of training very deep neural networks effectively while maintaining high accuracy. It achieves this by utilizing residual connections, which are skip connections that allow gradients to flow directly through the network without being hindered by the vanishing gradient problem. This helps in training deeper networks without degradation in performance.

InceptionResNetV2 incorporates the Inception module, which consists of parallel convolutional layers with different kernel sizes, allowing the network to capture information at multiple scales. The Inception module enables the network to efficiently capture both local and global features, leading to improved representational power.

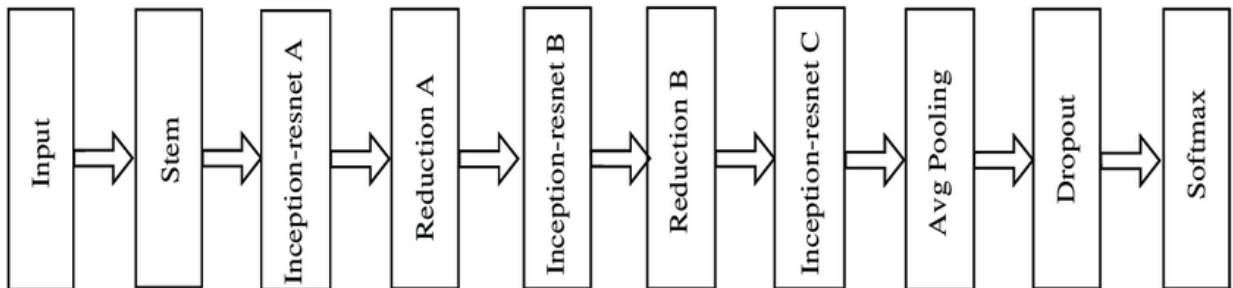


Figure 7. InceptionResNetV2 Module

PROBLEM SPECIFICATION

Plant diseases can have a devastating impact on agricultural productivity, leading to significant crop losses and economic damage. Timely detection and intervention are crucial for preventing the spread of diseases and minimizing the negative impact on crop yields. Traditional methods of disease detection rely on human expertise, visual inspection, and manual sampling, which can be time-consuming, error-prone, and often ineffective in identifying diseases at an early stage. Therefore, there is a need for an automated system that can detect plant diseases accurately and in a timely manner.

3.1 Technical Feasibility

The proposed system is technically feasible as it can be developed easily with the help of available technology. However, significant problems can be caused by gathering the test data. The test data gathered by the team was gathered from kaggle.com. Also significant resources are needed to train the resources for instance, enough RAM and a GPU of 16 GB VRAM. The latter can be taken care of by Google collab which provides these resources but with a time constraint.

3.2 Behavioral Feasibility

The feasibility of plant disease detection through automated methods largely depends on several factors. Firstly, the availability and quality of data play a vital role. Sufficient and diverse datasets comprising images, spectral data, or other relevant information are essential to train machine learning models effectively. Obtaining such datasets can be challenging, especially for rare or newly emerging diseases.

3.3 Economic Feasibility

Economic analysis is most frequently used for evaluation of the effectiveness of the system. More commonly known as cost/benefit analysis the procedure is to determine the benefit and savings that are expected from a system and compare them with cost, a decision is made to design and implement the system. This part of the feasibility study gives the economic justification of the system. The system being developed is economic with respect to School or Colleges point of view. It is cost effective in the sense that it has eliminated the paper work completely. The system is also time effective because the calculations are automated which are made at the end of the month or as per the user requirement. The result obtained contains minimum errors and are highly accurate as the data is required.

3.4 Operational Feasibility

Automation makes our life easy. The proposed system is highly user friendly and is much easier to interact with. Only little instruction is required for the user so that he can easily operate the system. It should be user-friendly, requiring minimal training and technical expertise. Ideally, the system should be easily deployable in various agricultural settings, such as small-scale farms or large-scale plantations. Integration with existing agricultural technologies or machinery can also enhance the feasibility of the system by leveraging the infrastructure already in place.

REQUIREMENT ANALYSIS

4.1 Hardware Configuration

- Processor : Intel i5
- Hard disk : 256 GB
- Memory : 8 GB
- Raspberry Pi 4
- Graphic Memory: N/A

4.2 Software Configuration

- Model : InceptionResnetV2
- Frameworks and Libraries
 - Tensorflow
 - cv2
 - numpy
- IDE: Google Colab
- Platform: Windows

4.3 Functional Requirements

Functional Requirements Specification describes what is required to meet the users' business needs. It specifies which actions the design must provide in order to benefit the system's users. These are determined by the needs, user, and task analysis of the current system. Functional requirements needed in this Plant Disease Detection using IOT are as follows:

Image Capture:

The picture of the plants that is tomato plant is to be captured using a webcam which can be connected to the computer/laptop using Raspberry Pi. The image is pre-processed in the sense that the size is reduced to 224x224 pixels. The captured image is then loaded into the memory using the cv2 library. The image can either be selected from the computer or take an image in real time.

Deep Learning Model:

The dataset was trained using the various algorithm and got the best result from InceptionResNetV2. We've used transfer learning to train the model with the number of epochs set to 20.

Showing Results:

After the image has been captured and pre-processed results shall define whether the plant is healthy or diseased and also tells the name of disease. The program works on a loop and therefore an infinite number of pictures can be taken and validated against the model until manually terminated.

4.4 Non-Functional Requirements

Non-functional requirements are requirements that are not directly concerned with the specific functions delivered by the system. They may relate to emergent system properties such as reliability, response time etc. They may specify system performance, security, availability, and other emergent properties. This means that they are often more critical than individual functional requirements. System users can usually find ways to work around a system function that doesn't really meet their needs. However, failing to meet a non-functional requirement can mean that the whole system is unusable. Non-functional requirements needed in Early Plant Disease Detection using IOT are as follows:

Efficiency

The proposed deep learning based system provided an accuracy of about 93.3% and proved to be much more efficient when we used a InceptionResNetV2 training model instead of Resnet50, InceptionV3, Resnet152V2 which had an accuracy of around 80%.

Reliability

The deep learning based model is reliable and provides results accurately 84% of the time. The model was tested with 4 different deep learning based artificial convolutional neural networks, but the InceptionResNetV2 model proved to be more reliable. Thus, we used that model in our testing and implementation.

Maintainability

It is fair to say that a system that undergoes changes with time is better suited and preferred over others. The system that we proposed has the capability to undergo changes and bug fixes in the future.

Portability

It is the usability of the same software in different environments. The pre requirement of portability is the generalized abstraction between the application logic and the system interfaces. The proposed system fulfills the portability requirement. The system that we proposed can be used in different environments. The model can be used on different platforms including linux , windows, mac etc.

Usability

The system is designed for a user friendly environment so that the administrator and user can perform various tasks easily and in an effective way.

Security

We understand that security is one of the most important aspects of system design. The system thus prepared is secure and no exploitations of the software can be done in known ways. In future, further security patches and bug fixes can help strengthen the security and integrity of the system.

SYSTEM DESIGN

Design is the first step into the development phase for any engineered product or system. Design is a creative process. A good design is the key to an effective system. The term “design” is defined as “the process of applying various techniques and principles for the purpose of defining a process or a system in sufficient detail to permit its physical realization”. Software design sits at the technical kernel of the software engineering process and is applied regardless of the development paradigm that is used. The system design develops the architectural detail required to build a system or product. As in the case of any systematic approach, this software too has undergone the best possible design phase fine tuning all efficiency, performance and accuracy levels. The design phase is a transition from a user oriented document to a document to the programmers or database personnel.

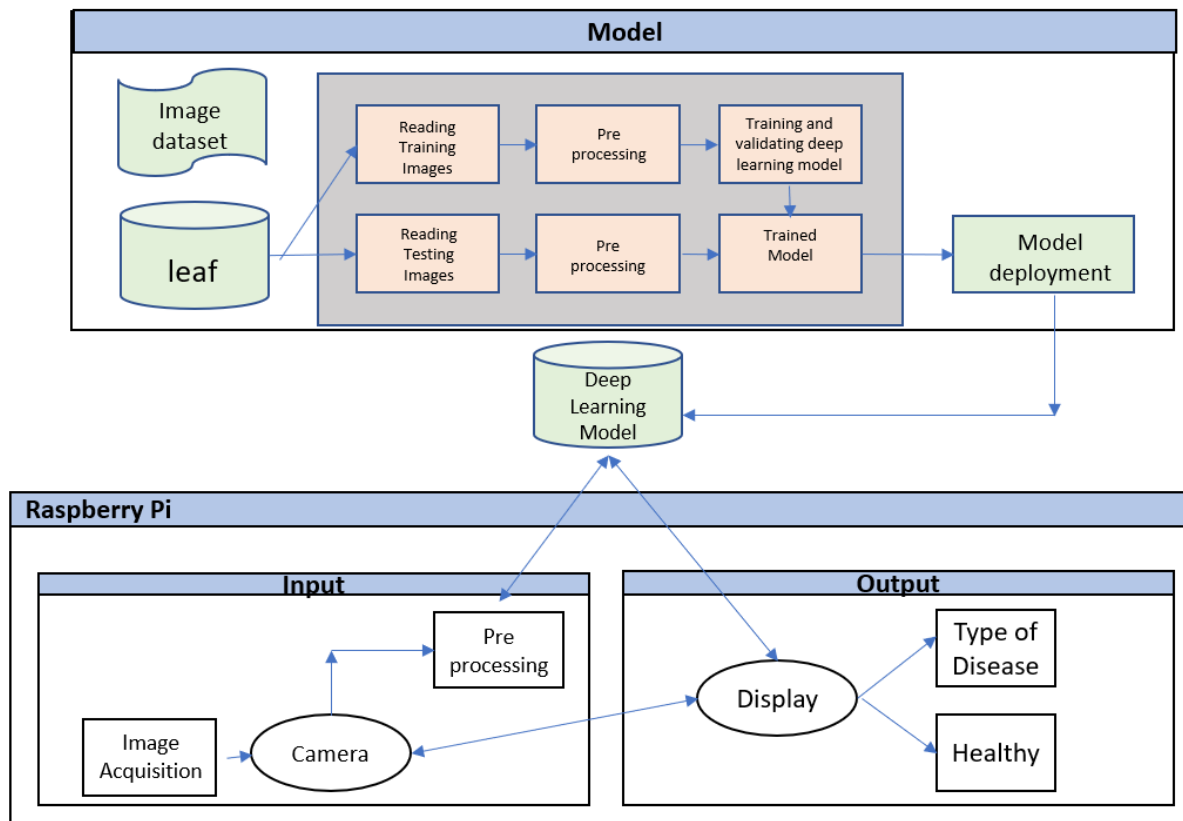


Figure 8. Proposed System Architecture

5.1 Acquiring Dataset

To design a system, the first step is the acquisition of a dataset. Training a deep learning based CNN (Convolved Neural Network) requires data to feed into that network. Based on that data CNN is able to generate results at the output neurons. Training involves the updating of weights based on the data provided. This type of learning is called supervised learning. Before that we need to get the required dataset. The dataset has been acquired from kaggle.com. The total images with which this model has been trained is about 12000 which includes some healthy tomato plants and some plants with different diseases.

5.2 Training the Deep Learning model

After the acquisition of the database, we make use of deep learning methods to develop a model that is able to detect diseases. We feed input data into deep learning algorithms like InceptionResnetV2, ResNet50, etc to generate such a model. Training involves updating of weights so that the accuracy of the model can be improved.

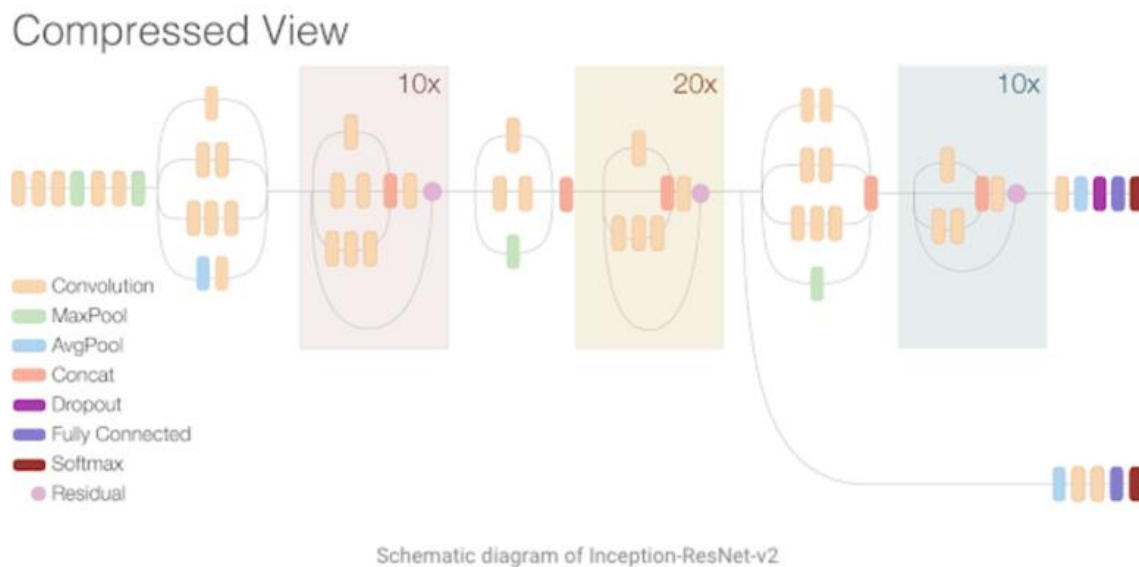


Figure 9. InceptionResNetV2 Architecture

5.3 Developing a User Interface

The next step is to develop a User Interface that helps us to interact with the model and fetch the results. Interaction with the model is done using Raspberry Pi and with the system is done by VNC Viewer. Virtual Network Computing (VNC) is a graphical desktop-sharing system that uses the Remote Frame Buffer protocol (RFB) to remotely control another computer.

IMPLEMENTATION

The implementation phase of the project is the development of the designs produced during the design phase. It is the stage in the project where the theoretical design is turned into a working system and is giving confidence on the new system for the users that it will work efficiently and effectively. It involves careful planning, investigation of the current system and its constraints on implementation, design of methods to achieve the changeover, and evaluation of change over methods. The implementation process begins with preparing a plan for the implementation of the system. According to this plan, the activities are to be carried out, discussions made regarding the equipment and resources and the additional equipment has to be acquired to implement the new system. In a network backup system no additional resources are needed. Implementation is the final and the most important phase. The system can be implemented only after thorough testing is done and if it is found to be working according to the specification. This method also offers the greatest security since the old system can take over if the errors are found or inability to handle certain types of transactions while using the new system.

System Testing

It is the stage of implementation, which is aimed at ensuring that the system works accurately and efficiently before live operation commences. Testing is the process of executing the program with the intent of finding errors and missing operations and also a complete verification to determine whether the objectives are met and the user requirements are satisfied. The ultimate aim is quality assurance. Tests are carried out and the results are compared with the expected document. In the case of erroneous results, debugging is done. Using detailed testing strategies a test plan is carried out on each module. The various tests performed are unit testing, integration testing and user acceptance testing.

Unit Testing

The software units in a system are modules and routines that are assembled and integrated to perform a specific function. Unit testing focuses first on modules, independently of one another, to locate errors. This enables, to detect errors in coding and logic that are contained within each module. This testing includes entering data and ascertaining if the value matches to the type and size supported by the system. The various controls are tested to ensure that each performs its action as required.

Integration Testing

Data can be lost across any interface, one module can have an adverse effect on another, sub functions when combined, may not produce the desired major functions. Integration testing is a systematic testing to discover errors associated within the interface. The objective is to take unit

tested modules and build a program structure. All the modules are combined and tested as a whole. Here the Server module and Client module options are integrated and tested. This testing provides the assurance that the application is a well-integrated functional unit with smooth transition of data.

User Acceptance Testing

User acceptance of a system is the key factor for the success of any system. The system under consideration is tested for user acceptance by constantly keeping in touch with the system users at time of developing and making changes whenever required. Apart from planning, major tasks of preparing the implementation are education and training of users. The most critical stage in achieving a successful new system is giving the users confidence that the new system will work and be effective.

User Training

After the system is implemented successfully, training of the user is one of the most important subtasks of the developer. For this purpose, user manuals are prepared and handed over to the user to operate the developed system. Thus, the users are trained to operate the developed system. Both the hardware and software securities are made to run the developed systems successfully in future. In order to put new application system into use, the following activities were taken care of: -

- a. Preparation of user and system documentation.
- b. Conducting user training with demo and hands on.
- c. Test run for some period to ensure smooth switching over the system

The users are trained to use the newly developed functions. User manuals describing the procedures for using the functions listed on the menu are circulated to all the users. It is confirmed that the system is implemented up to users' needs and expectations.

Security and Maintenance

Maintenance involves the software industry captive, typing up system resources. It means restoring something to its original condition. Maintenance follows conversion to the extent that changes are necessary to maintain satisfactory operations relative to changes in the user's environment. Maintenance often includes minor enhancements or corrections to problems that surface in the system's operation. Maintenance is also done based on fixing the problems reported, changing the interface with other software or hardware, enhancing the software. Any system developed should be secured and protected against possible hazards. Security measures are provided to prevent unauthorized access of the database at various levels. Password protection and simple procedures to prevent unauthorized access are provided to the user.

Step by Step implementation of the app

Deep Learning

- 1) **Collecting Dataset:** This includes getting the dataset through online resources and the surveys based on the requirements of the project. After that the dataset is made to go through the pre-processing phase.
- 2) **Training Models:** 2 models were trained using 2 different algorithms and the best of them is chosen based on their accuracies to make predictions in the app.
- 3) **Integration:** The model is integrated with the front end using the keras (tensorflow) library.

Dataset Description

The dataset contains images of the different types of diseased leaves and healthy leaves. An experiment is performed on the proposed dataset for automated classification and a high accuracy is achieved using Convolutional Neural Network.

Tools and Technology

Google Colaboratory or Colab for short, is a product from Google Research. Colab allows anybody to write and execute arbitrary python code through the browser, and is especially well suited to machine learning, data analysis and education. More technically, Colab is a hosted Jupyter notebook service that requires no setup to use, while providing access free of charge to computing resources including GPUs.

Resources in Colab are prioritized for interactive use cases. We prohibit actions associated with bulk computation, actions that negatively impact others, as well as actions associated with bypassing our policies. The following are disallowed from Colab runtimes:

- a. File hosting, media serving, or other web service offerings not related to interactive compute with Colab
- b. Downloading torrents or engaging in peer-to-peer file-sharing
- c. Using a remote desktop or SSH
- d. Connecting to remote proxies
- e. Mining cryptocurrency
- f. Running denial-of-service attacks
- g. Password cracking
- h. Using multiple accounts to work around access or resource usage restrictions
- i. Creating deep fakes

InceptionResNetV2 The InceptionResNetV2 architecture aims to address some of the limitations of previous CNN models by leveraging both the power of deep and wide network architectures. It achieves this by using a combination of Inception modules, which extract features at different scales, and residual connections, which facilitate the flow of gradients during training.

The main components of the InceptionResNetV2 architecture are:

- i. Stem: The initial layers of the network that perform basic preprocessing, such as convolution and pooling operations, to extract low-level features from the input data.
- ii. Inception modules: These are the building blocks of the network, and they consist of multiple parallel convolutional operations with different filter sizes. These parallel paths capture features at different scales and concatenate their outputs.
- iii. Reduction blocks: These blocks are inserted between Inception modules to reduce the spatial dimensions of the feature maps while increasing the number of channels. This reduction helps control the computational complexity of the network.
- iv. Auxiliary classifiers: InceptionResNetV2 includes auxiliary classifiers, which are additional classifier branches inserted at intermediate layers of the network. They provide regularization during training and help combat the vanishing gradient problem.
- v. Residual connections: Inspired by the ResNet architecture, InceptionResNetV2 incorporates residual connections that enable the network to learn residual mappings. These connections allow gradients to propagate more effectively, making it easier to train deep networks.

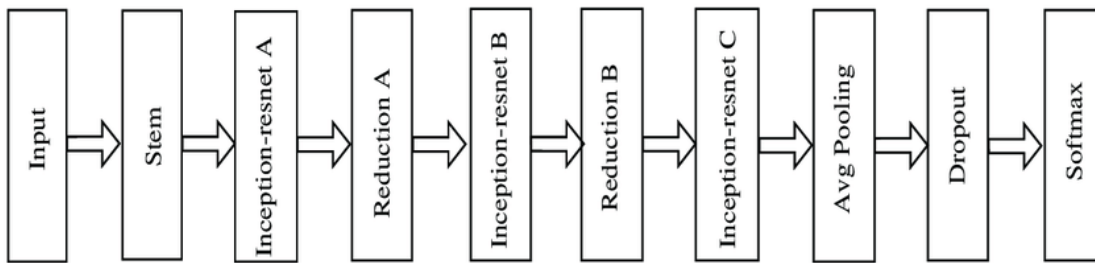


Figure 10. InceptionResNetV2

Python is a dynamic, high level, free open source and interpreted programming language. It supports object-oriented programming as well as procedural oriented programming. In Python, we don't need to declare the type of variable because it is a dynamically typed language. For example, `x = 10` Here, `x` can be anything such as String, int, etc.

- a. **Easy to code:** Python is a high-level programming language. Python is very easy to learn the language as compared to other languages like C, C#, Javascript, Java, etc. It is very easy to code in python language and anybody can learn python basics in a few hours or days. It is also a developer-friendly language.
- b. **Free and Open Source:** Python language is freely available at the official website and you can download it from the given download link below click on the Download Python keyword. Download Python Since it is open-source, this means that source code is also available to the public. So you can download it, use it as well as share it.
- c. **Object-Oriented Language:** One of the key features of python is Object-Oriented programming. Python supports object-oriented language and concepts of classes, object encapsulation, etc.
- d. **GUI Programming Support:** Graphical User interfaces can be made using a module such as PyQt5, PyQt4, wxPython, or Tk in python. PyQt5 is the most popular option for creating graphical apps with Python.
- e. **High-Level Language:** Python is a high-level language. When we write programs in python, we do not need to remember the system architecture, nor do we need to manage the memory.
- f. **Extensible feature:** Python is an Extensible language. We can write some Python code into C or C++ language and also we can compile that code in C/C++ language.
- g. **Python is Portable language:** Python language is also a portable language. For example, if we have python code for windows and if we want to run this code on other platforms such as Linux, Unix, and Mac then we do not need to change it, we can run this code on any platform.

Tensorflow is a Python-friendly open source library for numerical computation that makes machine learning and developing neural networks faster and easier. Machine learning is a complex discipline but implementing machine learning models is far less daunting than it used to be, thanks to machine learning frameworks—such as Google's TensorFlow—that ease the process of acquiring data, training models, serving predictions, and refining future results.

Created by the Google Brain team and initially released to the public in 2015, TensorFlow is an open source library for numerical computation and large-scale machine learning. TensorFlow bundles together a slew of machine learning and deep learning models and algorithms (aka neural networks) and makes them useful by way of common programmatic metaphors. It uses Python or JavaScript to provide a convenient front-end API for building applications, while executing those applications in high-performance C++.

TensorFlow, which competes with frameworks such as PyTorch and Apache MXNet, can train and run deep neural networks for handwritten digit classification, image recognition, word embeddings, recurrent neural networks, sequence-to-sequence models for machine translation, natural language processing, and PDE (partial differential equation)-based simulations. Best of all, TensorFlow supports production prediction at scale, with the same models used for training.

Numpy is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays. It is the fundamental package for scientific computing with Python. It is open-source software. It contains various features including these important ones:

- a. A powerful N-dimensional array object
- b. Sophisticated (broadcasting) functions
- c. Tools for integrating C/C++ and Fortran code
- d. Useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined using Numpy which allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

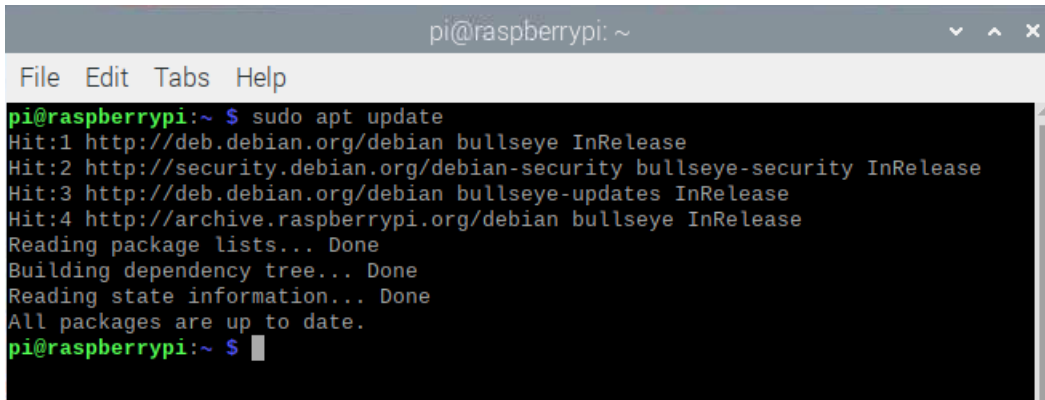
Opencv is an open source library which is very useful for computer vision applications such as video analysis, CCTV footage analysis and image analysis. OpenCV is written by C++ and has more than 2,500 optimized algorithms. When we create applications for computer vision that we don't want to build from scratch we can use this library to start focusing on real world problems. There are many companies using this library today such as Google, Amazon, Microsoft and Toyota. Many researchers and developers contribute. We can easily install it in any OS like Windows, Ubuntu and MacOS.

RESULTS AND DISCUSSION

7.1 Installation process:

Updating Raspberry Pi

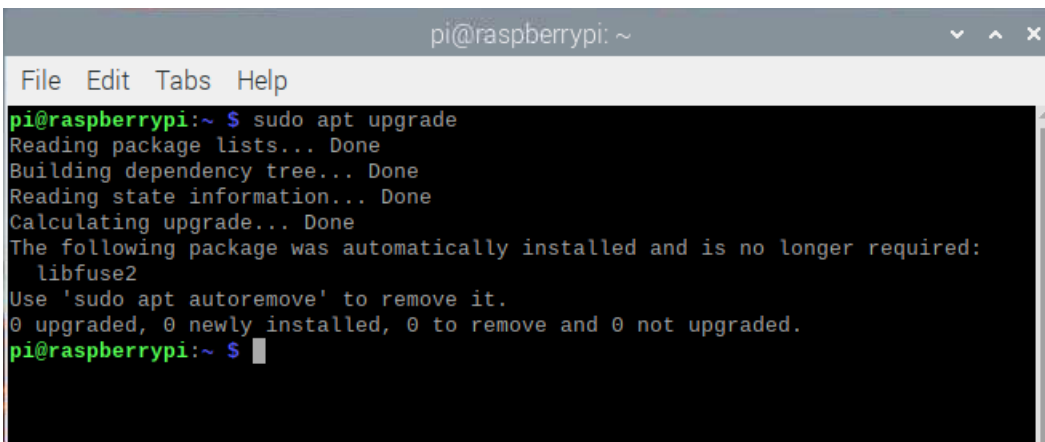
1. Get your Raspberry Updated



```
pi@raspberrypi: ~  
File Edit Tabs Help  
pi@raspberrypi:~ $ sudo apt update  
Hit:1 http://deb.debian.org/debian bullseye InRelease  
Hit:2 http://security.debian.org/debian-security bullseye-security InRelease  
Hit:3 http://deb.debian.org/debian bullseye-updates InRelease  
Hit:4 http://archive.raspberrypi.org/debian bullseye InRelease  
Reading package lists... Done  
Building dependency tree... Done  
Reading state information... Done  
All packages are up to date.  
pi@raspberrypi:~ $
```

Figure 11. Raspberry Pi updating Window

2. Get your Raspberry Upgraded



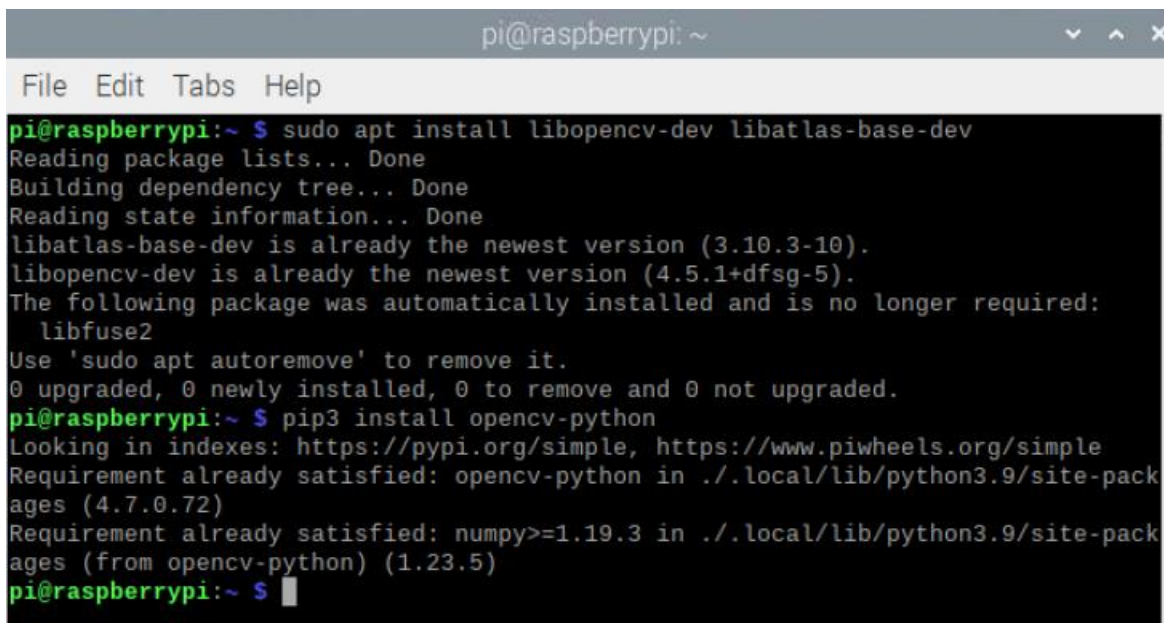
```
pi@raspberrypi: ~  
File Edit Tabs Help  
pi@raspberrypi:~ $ sudo apt upgrade  
Reading package lists... Done  
Building dependency tree... Done  
Reading state information... Done  
Calculating upgrade... Done  
The following package was automatically installed and is no longer required:  
  libfuse2  
Use 'sudo apt autoremove' to remove it.  
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.  
pi@raspberrypi:~ $
```

Figure 12. Raspberry Pi upgrading Window

Installing python libraries and modules

1. Installing opencv-python

Command: > pip install opencv-python

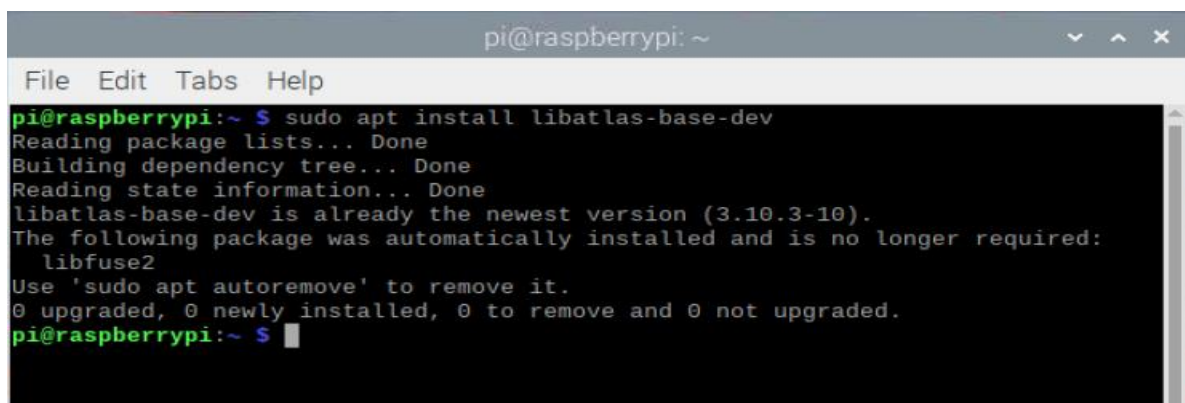


```
pi@raspberrypi:~ $ sudo apt install libopencv-dev libatlas-base-dev
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
libatlas-base-dev is already the newest version (3.10.3-10).
libopencv-dev is already the newest version (4.5.1+dfsg-5).
The following package was automatically installed and is no longer required:
  libfuse2
Use 'sudo apt autoremove' to remove it.
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
pi@raspberrypi:~ $ pip3 install opencv-python
Looking in indexes: https://pypi.org/simple, https://www.piwheels.org/simple
Requirement already satisfied: opencv-python in ./local/lib/python3.9/site-pack
ages (4.7.0.72)
Requirement already satisfied: numpy>=1.19.3 in ./local/lib/python3.9/site-pack
ages (from opencv-python) (1.23.5)
pi@raspberrypi:~ $
```

Figure 13. Installing opencv-python

2. Installing TensorFlow

Command: > pip install tensorflow



```
pi@raspberrypi:~ $ sudo apt install libatlas-base-dev
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
libatlas-base-dev is already the newest version (3.10.3-10).
The following package was automatically installed and is no longer required:
  libfuse2
Use 'sudo apt autoremove' to remove it.
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
pi@raspberrypi:~ $
```

Figure 14. Installing library for tensorflow

```

pi@raspberrypi:~ $ pip3 install tensorflow
Looking in indexes: https://pypi.org/simple, https://www.piwheels.org/simple
Requirement already satisfied: tensorflow in ./local/lib/python3.9/site-packages (2.12.0)
Requirement already satisfied: tensorflow-cpu-aws==2.12.0 in ./local/lib/python3.9/site-packages (from tensorflow) (2.12.0)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in ./local/lib/python3.9/site-packages (from tensorflow-cpu-aws==2.12.0->tensorflow) (0.32.0)
Requirement already satisfied: tensorboard<2.13,>=2.12 in ./local/lib/python3.9/site-packages (from tensorflow-cpu-aws==2.12.0->tensorflow) (2.12.3)
Requirement already satisfied: setuptools in /usr/lib/python3/dist-packages (from tensorflow-cpu-aws==2.12.0->tensorflow) (52.0.0)
Requirement already satisfied: six>=1.12.0 in /usr/lib/python3/dist-packages (from tensorflow-cpu-aws==2.12.0->tensorflow) (1.16.0)
Requirement already satisfied: packaging in ./local/lib/python3.9/site-packages (from tensorflow-cpu-aws==2.12.0->tensorflow) (23.1)
Requirement already satisfied: termcolor>=1.1.0 in ./local/lib/python3.9/site-packages (from tensorflow-cpu-aws==2.12.0->tensorflow) (2.3.0)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in ./local/lib/python3.9/site-packages (from tensorflow-cpu-aws==2.12.0->tensorflow) (1.54.2)
Requirement already satisfied: jax>=0.3.15 in ./local/lib/python3.9/site-packages (from tensorflow-cpu-aws==2.12.0->tensorflow) (0.4.12)
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/lib/python3/dist-packages (from tensorflow-cpu-aws==2.12.0->tensorflow) (3.7.4.3)
Requirement already satisfied: h5py>=2.9.0 in ./local/lib/python3.9/site-packages (from tensorflow-cpu-aws==2.12.0->tensorflow) (3.8.0)
Requirement already satisfied: protobuf!=4.21.0,!<4.21.1,!<4.21.2,!<4.21.3,!<4.21.4,!<4.21.5,<5.0.0dev,>=3.20.3 in ./local/lib/python3.9/site-packages (from tensorflow-cpu-aws==2.12.0->tensorflow) (4.23.3)
Requirement already satisfied: flatbuffers>=2.0 in ./local/lib/python3.9/site-packages (from tensorflow-cpu-aws==2.12.0->tensorflow) (20181003210633)
Requirement already satisfied: astunparse>=1.6.0 in ./local/lib/python3.9/site-packages (from tensorflow-cpu-aws==2.12.0->tensorflow) (1.6.3)
Requirement already satisfied: wrapt<1.15,>=1.11.0 in /usr/lib/python3/dist-packages (from tensorflow-cpu-aws==2.12.0->tensorflow) (1.12.1)
Requirement already satisfied: google-pasta>=0.1.1 in ./local/lib/python3.9/site-packages (from tensorflow-cpu-aws==2.12.0->tensorflow) (0.2.0)
Requirement already satisfied: keras<2.13,>=2.12.0 in ./local/lib/python3.9/site-packages (from tensorflow-cpu-aws==2.12.0->tensorflow) (2.12.0)
Requirement already satisfied: gast<=0.4.0,>=0.2.1 in ./local/lib/python3.9/site-packages (from tensorflow-cpu-aws==2.12.0->tensorflow) (0.4.0)
Requirement already satisfied: opt-einsum>=2.3.2 in ./local/lib/python3.9/site-packages (from tensorflow-cpu-aws==2.12.0->tensorflow) (3.3.0)
Requirement already satisfied: tensorflow-estimator<2.13,>=2.12.0 in ./local/lib/python3.9/site-packages (from tensorflow-cpu-aws==2.12.0->tensorflow) (2.12.0)

```

Figure 15. Installing tensorflow

3. Installing numpy

Command: > apt install numpy

```

pi@raspberrypi:~ $ sudo apt install python3-dev python3-pip python3-numpy
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
python3-numpy is already the newest version (1:1.19.5-1).
python3-dev is already the newest version (3.9.2-3).
python3-pip is already the newest version (20.3.4-4+rpt1+deb11u1).
The following package was automatically installed and is no longer required:
  libfuse2
Use 'sudo apt autoremove' to remove it.
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
pi@raspberrypi:~ $

```

Figure 16. Installing numpy

4. Running the project

Command: %Run project.py

7.2 Project Screenshots



Figure 17. leaf with Bacterial Spot

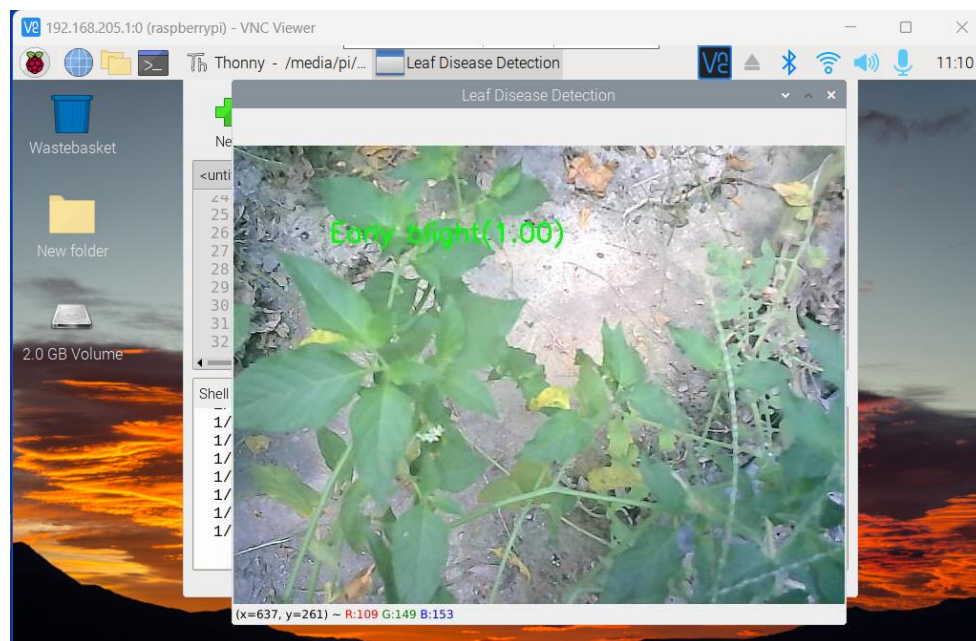


Figure 18. Leaf With Early Blight



Figure 19. Leaf With Late Blight

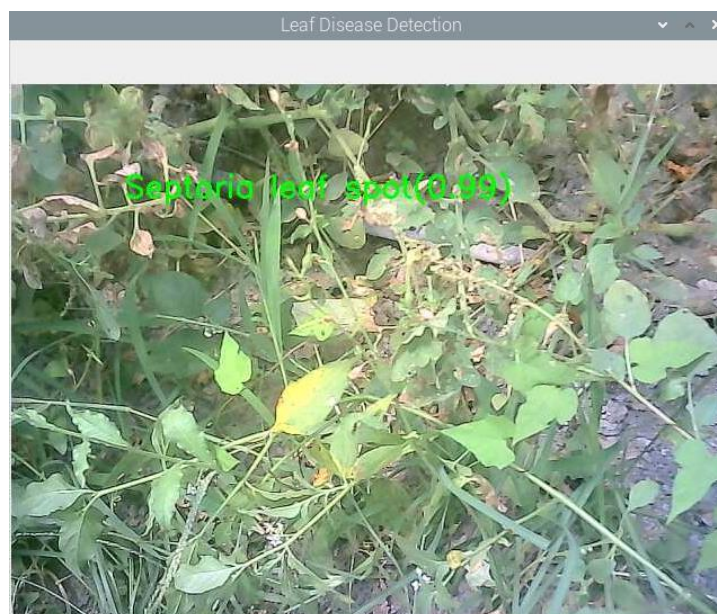


Figure 20. Leaf With Septoria Leaf Spot

7.3 Opportunities and Challenges of Application of Deep Learning for Plant Disease detection

In the modern time with growing population across the world, it is important for us to maximize our resources. Agriculture is a field which is the basic requirement of all the living beings to survive. The early detection of disease in plants helps us to take steps in initial stages which increase our chances of a favorable outcome i.e, minimum damage done to the crop. This leads to increase in productivity and is also an important factor for the economy of the country.

Opportunities

1. **To address the needs of disease detection in large areas:** The application of current state-of-the-art deep learning models for developing an automated visual inspection system for crops. This approach is cost effective and swifter than the manual approach.
2. **To promote automation:** The world is moving towards automation. This will save human resources and time required in the processing and prevention of crops from diseases.

Challenges

1. **Unavailability of dataset:** The datasets are often not available to train and hence will result in inaccurate results.
2. **Accuracy Issues:** The results often do not comply with the input dataset. This could be attributed to a lot of things. For instance, choosing an inefficient algorithm, setting less number of epochs, or due to other environmental constraints.
3. **Improper selection of dataset.**
4. **Poor Quality of Images:** If either the quality of images in the dataset is poor or the captured image is of the poor quality, the results shall also be inaccurate.
5. **Computational Power of Raspberry Pi:** Since Raspberry Pi has much lesser computational power as compared to laptops and desktops, therefore there is a lag when the application is used. This is because the system requires high processing power. The lag causes problems in accurate detection of the disease.
6. **Camera:** The camera used in raspberry pi has limited resolution. This results in a poor quality of input images which leads to poor accuracy of the model. Also, the camera is unable to auto focus on the leaves and hence the efficiency of the device. The device is able to make accurate predictions on locally saved images but is unable to do so with the live images that are provided by the camera.

SUMMARY

India heavily relies on agriculture for its food security. Early detection of plant diseases helps prevent widespread crop failures and ensures a stable food supply. By minimizing crop losses due to diseases, early detection contributes to maintaining food security and reducing the vulnerability of farmers to economic and food-related crises.

This project aims to develop an efficient and automated system for early plant disease detection using the Internet of Things (IoT) technology and Raspberry Pi. Plant diseases can have detrimental effects on crop yields, leading to significant economic losses in agriculture. Detecting diseases at an early stage is crucial for implementing timely interventions and preventing further spread. The proposed system combines image processing techniques, IoT connectivity, and machine learning algorithms to detect and classify plant diseases accurately.

The system comprises a Raspberry Pi, which acts as a central processing unit, and a camera module that captures images of plants. By integrating the camera with the Raspberry Pi, the system can capture high-resolution images of the plants in real-time. These images are then processed using advanced image processing algorithms to identify any signs of diseases or anomalies present in the plants.




To enable remote monitoring and analysis, the system utilizes IoT technology, allowing the captured images and disease-related data to be transmitted to a cloud-based server. This server hosts a comprehensive database and employs machine learning algorithms to classify the detected diseases accurately.

The project report emphasizes the significance of early detection in disease management and highlights the advantages of employing IoT and Raspberry Pi technology for this purpose. It showcases the potential of this system to significantly reduce crop losses and increase agricultural productivity by facilitating proactive disease control measures. Furthermore, the report discusses the implementation details, challenges encountered, and potential improvements that can be made to enhance the system's efficiency and accuracy.

LITERATURE CITED

1. <https://www.kaggle.com/datasets/kaustubhb999/tomatoleaf>
2. https://www.researchgate.net/figure/ResNet152V2-architecture_fig5_343978097
3. <https://blog.devgenius.io/resnet50-6b42934db431>
4. <https://medium.com/@zahraelhamraoui1997/inceptionresnetv2-simple-introduction-9a2000edcdb6>
5. <https://blog.paperspace.com/popular-deep-learning-architectures-resnet-inceptionv3-squeezenet/>
6. <https://link.springer.com/article/10.1007/s42979-022-01152-7>
7. <https://www.sciencedirect.com/science/article/pii/S266615432100020X>

BIO-DATA OF STUDENTS

	<p>Name : Akansha Rawat ID No. : 54883 Place : Haldwani</p>
	<p>Name : Milind Joshi ID No. : 54896 Place : Haldwani</p>
	<p>Name : Mohd. Zaid ID No. : 54897 Place : Roorkee</p>

Scanned with CamScanner

APPENDIX I

Installation of the Project

Early detection allows for timely intervention, which can prevent the spread of diseases and minimize crop losses. By identifying diseases at their initial stages, farmers can take appropriate measures such as applying targeted treatments or removing infected plants, thereby safeguarding the health of the remaining crops. This leads to higher crop yields and improved productivity.

Python is one of the most widely adopted programming languages in the world. It is especially used for web development, backend development, training of machine learning, artificial neural networks and deep learning materials.

The model is based on transfer learning and trained using the tomato dataset using the InceptionResnetV2 model. The dataset comprised 10 classes, namely healthy and other 9 diseases. The weights were set using the imagenet model and the activation function were categorical respectively.

The batch set used to train the model were set to 314 with the number of epochs set to 20.

In order to install the project, the model needs to be first trained and saved (Appendix II). The user then needs to install the following libraries, namely tensorflow, cv2 using the pip python package manager.

The user can then either supply a saved photo or take a fresh photo through a webcam or a camera and then can see results. The model will show if plant is healthy or not, if they are not healthy then will show the disease name.

APPENDIX II

Source Code

1. # Training the Model with InceptionResNetV2

```
from tensorflow.compat.v1 import ConfigProto
from tensorflow.compat.v1 import InteractiveSession

config = ConfigProto()
config.gpu_options.per_process_gpu_memory_fraction = 0.5
config.gpu_options.allow_growth = True
session = InteractiveSession(config=config)

from google.colab import drive
IMAGE_SIZE = [224, 224]

train_path = '/content/drive/MyDrive/tomato/train'
valid_path = '/content/drive/MyDrive/tomato/val'

# Import the Vgg 16 library as shown below and add preprocessing layer to
the front of VGG
# Here we will be using imagenet weights
import tensorflow
inceptionResNetV2
=tensorflow.keras.applications.InceptionResNetV2(input_shape=IMAGE_SIZE +
[3], weights='imagenet', include_top=False)

# don't train existing weights
for layer in inceptionResNetV2.layers:
    layer.trainable = False

# useful for getting number of output classes
folders = glob('/content/drive/MyDrive/tomato/train/*')

# our layers - you can add more if you want
x = Flatten()(inceptionResNetV2.output)

prediction = Dense(len(folders), activation='softmax')(x)

# create a model object
model = Model(inputs=inceptionResNetV2.input, outputs=prediction)
```

```

# view the structure of the model
model.summary()

# tell the model what cost and optimization method to use
model.compile(
    loss='categorical_crossentropy',
    optimizer='adam',
    metrics=['accuracy']
)

# Use the Image Data Generator to import the images from the dataset
from tensorflow.keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(rescale = 1./255,
                                    shear_range = 0.2,
                                    zoom_range = 0.2,
                                    horizontal_flip = True)

test_datagen = ImageDataGenerator(rescale = 1./255)

# Make sure you provide the same target size as initialied for the image
size
training_set =
train_datagen.flow_from_directory('/content/drive/MyDrive/tomato/train',
                                  target_size = (224, 224),
                                  batch_size = 32,
                                  class_mode =
'categorical')

test_set =
test_datagen.flow_from_directory('/content/drive/MyDrive/tomato/val',
                                  target_size = (224, 224),
                                  batch_size = 32,
                                  class_mode = 'categorical')

# fit the model
# Run the cell. It will take some time to execute
r = model.fit_generator(
    training_set,
    validation_data=test_set,
    epochs=20,
    steps_per_epoch=len(training_set),
    validation_steps=len(test_set)
)

```

```

import matplotlib.pyplot as plt

# plot the loss
plt.plot(r.history['loss'], label='train loss')
plt.plot(r.history['val_loss'], label='val loss')
plt.legend()
plt.show()
plt.savefig('LossVal_loss')

# plot the accuracy
plt.plot(r.history['accuracy'], label='train acc')
plt.plot(r.history['val_accuracy'], label='val acc')
plt.legend()
plt.show()
plt.savefig('AccVal_acc')

from tensorflow.keras.models import load_model

model.save('/content/drive/MyDrive/tomato/inceptionResNet.h5')

```

	Code	Text	
314/314	[=====]	- 183s 583ms/step - loss: 1.6612 - accuracy: 0.8096 - val_loss: 2.3701 - val_accuracy: 0.8040	
Epoch 4/20			
314/314	[=====]	- 185s 589ms/step - loss: 1.3503 - accuracy: 0.8484 - val_loss: 2.7026 - val_accuracy: 0.7980	
Epoch 5/20			
314/314	[=====]	- 190s 606ms/step - loss: 1.3780 - accuracy: 0.8543 - val_loss: 2.0760 - val_accuracy: 0.8360	
Epoch 6/20			
314/314	[=====]	- 189s 602ms/step - loss: 1.1718 - accuracy: 0.8729 - val_loss: 2.4680 - val_accuracy: 0.8260	
Epoch 7/20			
314/314	[=====]	- 185s 590ms/step - loss: 1.1375 - accuracy: 0.8829 - val_loss: 2.7506 - val_accuracy: 0.7840	
Epoch 8/20			
314/314	[=====]	- 183s 582ms/step - loss: 1.2273 - accuracy: 0.8806 - val_loss: 2.7105 - val_accuracy: 0.8270	
Epoch 9/20			
314/314	[=====]	- 183s 581ms/step - loss: 1.1669 - accuracy: 0.8876 - val_loss: 2.7227 - val_accuracy: 0.8440	
Epoch 10/20			
314/314	[=====]	- 180s 572ms/step - loss: 1.0632 - accuracy: 0.8992 - val_loss: 2.4872 - val_accuracy: 0.8550	
Epoch 11/20			
314/314	[=====]	- 181s 575ms/step - loss: 1.0618 - accuracy: 0.9007 - val_loss: 2.8224 - val_accuracy: 0.8250	
Epoch 12/20			
314/314	[=====]	- 187s 595ms/step - loss: 1.0148 - accuracy: 0.9073 - val_loss: 4.2367 - val_accuracy: 0.7950	
Epoch 13/20			
314/314	[=====]	- 187s 597ms/step - loss: 0.9304 - accuracy: 0.9139 - val_loss: 3.1482 - val_accuracy: 0.8390	
Epoch 14/20			
314/314	[=====]	- 183s 582ms/step - loss: 0.9281 - accuracy: 0.9146 - val_loss: 3.2325 - val_accuracy: 0.8300	
Epoch 15/20			
314/314	[=====]	- 184s 587ms/step - loss: 0.9830 - accuracy: 0.9127 - val_loss: 2.5535 - val_accuracy: 0.8580	
Epoch 16/20			
314/314	[=====]	- 182s 581ms/step - loss: 0.9271 - accuracy: 0.9213 - val_loss: 3.8761 - val_accuracy: 0.8050	
Epoch 17/20			
314/314	[=====]	- 186s 593ms/step - loss: 0.9531 - accuracy: 0.9238 - val_loss: 2.9948 - val_accuracy: 0.8570	
Epoch 18/20			
314/314	[=====]	- 182s 578ms/step - loss: 1.0117 - accuracy: 0.9204 - val_loss: 3.5738 - val_accuracy: 0.8190	
Epoch 19/20			
314/314	[=====]	- 181s 578ms/step - loss: 0.8725 - accuracy: 0.9287 - val_loss: 3.6485 - val_accuracy: 0.8510	
Epoch 20/20			
314/314	[=====]	- 181s 578ms/step - loss: 0.7777 - accuracy: 0.9339 - val_loss: 2.9506 - val_accuracy: 0.8430	

2. # Training the Model with ResNet152V2

```

from tensorflow.compat.v1 import ConfigProto
from tensorflow.compat.v1 import InteractiveSession

config = ConfigProto()
config.gpu_options.per_process_gpu_memory_fraction = 0.5

```



```

config.gpu_options.allow_growth = True
session = InteractiveSession(config=config)

from google.colab import drive
IMAGE_SIZE = [224, 224]

train_path = '/content/drive/MyDrive/tomato/train'
valid_path = '/content/drive/MyDrive/tomato/val'

# Import the Vgg 16 library as shown below and add preprocessing layer to
the front of VGG
# Here we will be using imagenet weights
import tensorflow
inceptionResNetV2
=tensorflow.keras.applications.ResNet152V2(input_shape=IMAGE_SIZE + [3],
weights='imagenet', include_top=False)

# don't train existing weights
for layer in inceptionResNetV2.layers:
    layer.trainable = False

# useful for getting number of output classes
folders = glob('/content/drive/MyDrive/tomato/train/*')

# our layers - you can add more if you want
x = Flatten()(ResNet152V2.output)

prediction = Dense(len(folders), activation='softmax')(x)

# create a model object
model = Model(inputs=ResNet152V2.input, outputs=prediction)

# view the structure of the model
model.summary()

# tell the model what cost and optimization method to use
model.compile(
    loss='categorical_crossentropy',
    optimizer='adam',
    metrics=['accuracy']
)

# Use the Image Data Generator to import the images from the dataset
from tensorflow.keras.preprocessing.image import ImageDataGenerator

```

```

train_datagen = ImageDataGenerator(rescale = 1./255,
                                   shear_range = 0.2,
                                   zoom_range = 0.2,
                                   horizontal_flip = True)

test_datagen = ImageDataGenerator(rescale = 1./255)

# Make sure you provide the same target size as initialied for the image
size
training_set =
train_datagen.flow_from_directory('/content/drive/MyDrive/tomato/train',
                                  target_size = (224, 224),
                                  batch_size = 32,
                                  class_mode =
'categorical')

test_set =
test_datagen.flow_from_directory('/content/drive/MyDrive/tomato/val',
                                  target_size = (224, 224),
                                  batch_size = 32,
                                  class_mode = 'categorical')

# fit the model
# Run the cell. It will take some time to execute
r = model.fit_generator(
    training_set,
    validation_data=test_set,
    epochs=20,
    steps_per_epoch=len(training_set),
    validation_steps=len(test_set)
)

import matplotlib.pyplot as plt

# plot the loss
plt.plot(r.history['loss'], label='train loss')
plt.plot(r.history['val_loss'], label='val loss')
plt.legend()
plt.show()
plt.savefig('LossVal_loss')

# plot the accuracy
plt.plot(r.history['accuracy'], label='train acc')
plt.plot(r.history['val_accuracy'], label='val acc')
plt.legend()

```

```
plt.show()
plt.savefig('AccVal_acc')

from tensorflow.keras.models import load_model

model.save('/content/drive/MyDrive/tomato/ResNet152V2.h5')
```

+ Code + Text Connect ▾ ^

```
313/313 [=====] - 173s 552ms/step - loss: 1.9987 - accuracy: 0.8879 - val_loss: 4.5727 - val_accuracy: 0.8330
Epoch 4/20
313/313 [=====] - 173s 552ms/step - loss: 1.7216 - accuracy: 0.9009 - val_loss: 4.9812 - val_accuracy: 0.8390
Epoch 5/20
313/313 [=====] - 171s 545ms/step - loss: 1.4434 - accuracy: 0.9228 - val_loss: 4.7592 - val_accuracy: 0.8560
Epoch 6/20
313/313 [=====] - 173s 552ms/step - loss: 1.3782 - accuracy: 0.9261 - val_loss: 5.0916 - val_accuracy: 0.8460
Epoch 7/20
313/313 [=====] - 172s 549ms/step - loss: 1.2384 - accuracy: 0.9328 - val_loss: 4.1981 - val_accuracy: 0.8700
Epoch 8/20
313/313 [=====] - 170s 542ms/step - loss: 1.2929 - accuracy: 0.9392 - val_loss: 6.1920 - val_accuracy: 0.8570
Epoch 9/20
313/313 [=====] - 172s 549ms/step - loss: 1.4071 - accuracy: 0.9392 - val_loss: 5.4516 - val_accuracy: 0.8530
Epoch 10/20
313/313 [=====] - 172s 549ms/step - loss: 1.3787 - accuracy: 0.9440 - val_loss: 5.4262 - val_accuracy: 0.8610
Epoch 11/20
313/313 [=====] - 173s 552ms/step - loss: 1.1550 - accuracy: 0.9478 - val_loss: 5.3259 - val_accuracy: 0.8680
Epoch 12/20
313/313 [=====] - 173s 551ms/step - loss: 1.0752 - accuracy: 0.9553 - val_loss: 6.8245 - val_accuracy: 0.8710
Epoch 13/20
313/313 [=====] - 168s 536ms/step - loss: 0.9018 - accuracy: 0.9587 - val_loss: 7.5217 - val_accuracy: 0.8590
Epoch 14/20
313/313 [=====] - 172s 549ms/step - loss: 1.0900 - accuracy: 0.9556 - val_loss: 6.9104 - val_accuracy: 0.8690
Epoch 15/20
313/313 [=====] - 172s 548ms/step - loss: 0.9053 - accuracy: 0.9611 - val_loss: 7.5369 - val_accuracy: 0.8560
Epoch 16/20
313/313 [=====] - 169s 539ms/step - loss: 0.9493 - accuracy: 0.9609 - val_loss: 5.7347 - val_accuracy: 0.8790
Epoch 17/20
313/313 [=====] - 173s 551ms/step - loss: 1.1412 - accuracy: 0.9589 - val_loss: 6.5633 - val_accuracy: 0.8780
Epoch 18/20
313/313 [=====] - 173s 551ms/step - loss: 0.9292 - accuracy: 0.9653 - val_loss: 9.5018 - val_accuracy: 0.8400
Epoch 19/20
313/313 [=====] - 171s 547ms/step - loss: 0.9289 - accuracy: 0.9615 - val_loss: 7.3312 - val_accuracy: 0.8720
Epoch 20/20
313/313 [=====] - 167s 534ms/step - loss: 0.8588 - accuracy: 0.9667 - val_loss: 7.1662 - val_accuracy: 0.8800
```

3. # Training the Model with InceptionV3

```
from tensorflow.compat.v1 import ConfigProto
from tensorflow.compat.v1 import InteractiveSession

config = ConfigProto()
config.gpu_options.per_process_gpu_memory_fraction = 0.5
config.gpu_options.allow_growth = True
session = InteractiveSession(config=config)

from google.colab import drive
IMAGE_SIZE = [224, 224]

train_path = '/content/drive/MyDrive/tomato/train'
valid_path = '/content/drive/MyDrive/tomato/val'
```

```

# Import the Vgg 16 library as shown below and add preprocessing layer to
the front of VGG
# Here we will be using imagenet weights
import tensorflow
ResNet50 =tensorflow.keras.applications.InceptionV3(input_shape=IMAGE_SIZE
+ [3], weights='imagenet', include_top=False)

# don't train existing weights
for layer in InceptionV3.layers:
    layer.trainable = False

# useful for getting number of output classes
folders = glob('/content/drive/MyDrive/tomato/train/*')

# our layers - you can add more if you want
x = Flatten()(InceptionV3.output)

prediction = Dense(len(folders), activation='softmax')(x)

# create a model object
model = Model(inputs=InceptionV3.input, outputs=prediction)

# view the structure of the model
model.summary()

# tell the model what cost and optimization method to use
model.compile(
    loss='categorical_crossentropy',
    optimizer='adam',
    metrics=['accuracy']
)

# Use the Image Data Generator to import the images from the dataset
from tensorflow.keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(rescale = 1./255,
                                   shear_range = 0.2,
                                   zoom_range = 0.2,
                                   horizontal_flip = True)

test_datagen = ImageDataGenerator(rescale = 1./255)

# Make sure you provide the same target size as initialied for the image
size

```

```

training_set =
train_datagen.flow_from_directory('/content/drive/MyDrive/tomato/train',
                                target_size = (224, 224),
                                batch_size = 32,
                                class_mode =
'categorical')

test_set =
test_datagen.flow_from_directory('/content/drive/MyDrive/tomato/val',
                                target_size = (224, 224),
                                batch_size = 32,
                                class_mode = 'categorical')

# fit the model
# Run the cell. It will take some time to execute
r = model.fit_generator(
    training_set,
    validation_data=test_set,
    epochs=20,
    steps_per_epoch=len(training_set),
    validation_steps=len(test_set)
)

import matplotlib.pyplot as plt

# plot the loss
plt.plot(r.history['loss'], label='train loss')
plt.plot(r.history['val_loss'], label='val loss')
plt.legend()
plt.show()
plt.savefig('LossVal_loss')

# plot the accuracy
plt.plot(r.history['accuracy'], label='train acc')
plt.plot(r.history['val_accuracy'], label='val acc')
plt.legend()
plt.show()
plt.savefig('AccVal_acc')

from tensorflow.keras.models import load_model

model.save('/content/drive/MyDrive/tomato/InceptionV3.h5')

```

```

Epoch 1/10
316/316 [=====] - 2515s 8s/step - loss: 3.3360 - accuracy: 0.6420 - val_loss: 1.6680 - val_accuracy: 0.7703
Epoch 2/10
316/316 [=====] - 1471s 5s/step - loss: 1.9448 - accuracy: 0.7700 - val_loss: 3.3021 - val_accuracy: 0.6980
Epoch 3/10
316/316 [=====] - 1499s 5s/step - loss: 2.0389 - accuracy: 0.7960 - val_loss: 3.2009 - val_accuracy: 0.7468
Epoch 4/10
316/316 [=====] - 1490s 5s/step - loss: 1.7043 - accuracy: 0.8272 - val_loss: 3.6602 - val_accuracy: 0.7360
Epoch 5/10
316/316 [=====] - 1517s 5s/step - loss: 1.7057 - accuracy: 0.8438 - val_loss: 4.2725 - val_accuracy: 0.7414
Epoch 6/10
316/316 [=====] - 1517s 5s/step - loss: 1.6682 - accuracy: 0.8550 - val_loss: 3.1859 - val_accuracy: 0.7884
Epoch 7/10
316/316 [=====] - 1509s 5s/step - loss: 1.5209 - accuracy: 0.8664 - val_loss: 4.3891 - val_accuracy: 0.7631
Epoch 8/10
316/316 [=====] - 1525s 5s/step - loss: 1.6416 - accuracy: 0.8612 - val_loss: 3.2082 - val_accuracy: 0.8246
Epoch 9/10
316/316 [=====] - 1520s 5s/step - loss: 1.2888 - accuracy: 0.8909 - val_loss: 4.2443 - val_accuracy: 0.7812
Epoch 10/10
316/316 [=====] - 1528s 5s/step - loss: 1.3159 - accuracy: 0.8871 - val_loss: 4.3514 - val_accuracy: 0.7794

```

4. # Training the Model with ResNet50

```

from tensorflow.compat.v1 import ConfigProto
from tensorflow.compat.v1 import InteractiveSession

config = ConfigProto()
config.gpu_options.per_process_gpu_memory_fraction = 0.5
config.gpu_options.allow_growth = True
session = InteractiveSession(config=config)

from google.colab import drive
IMAGE_SIZE = [224, 224]

train_path = '/content/drive/MyDrive/tomato/train'
valid_path = '/content/drive/MyDrive/tomato/val'

# Import the Vgg 16 library as shown below and add preprocessing layer to
the front of VGG
# Here we will be using imagenet weights
import tensorflow
ResNet50 = tensorflow.keras.applications.ResNet50(input_shape=IMAGE_SIZE +
[3], weights='imagenet', include_top=False)

# don't train existing weights
for layer in ResNet50.layers:
    layer.trainable = False

# useful for getting number of output classes
folders = glob('/content/drive/MyDrive/tomato/train/*')

```

```

# our layers - you can add more if you want
x = Flatten()(InceptionV3.output)

prediction = Dense(len(folders), activation='softmax')(x)

# create a model object
model = Model(inputs=InceptionV3.input, outputs=prediction)

# view the structure of the model
model.summary()

# tell the model what cost and optimization method to use
model.compile(
    loss='categorical_crossentropy',
    optimizer='adam',
    metrics=['accuracy']
)

# Use the Image Data Generator to import the images from the dataset
from tensorflow.keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(rescale = 1./255,
                                    shear_range = 0.2,
                                    zoom_range = 0.2,
                                    horizontal_flip = True)

test_datagen = ImageDataGenerator(rescale = 1./255)

# Make sure you provide the same target size as initialied for the image
size
training_set =
train_datagen.flow_from_directory('/content/drive/MyDrive/tomato/train',
                                target_size = (224, 224),
                                batch_size = 32,
                                class_mode =
'categorical')

test_set =
test_datagen.flow_from_directory('/content/drive/MyDrive/tomato/val',
                                target_size = (224, 224),
                                batch_size = 32,
                                class_mode = 'categorical')

# fit the model
# Run the cell. It will take some time to execute

```

```

r = model.fit_generator(
    training_set,
    validation_data=test_set,
    epochs=20,
    steps_per_epoch=len(training_set),
    validation_steps=len(test_set)
)

import matplotlib.pyplot as plt

# plot the loss
plt.plot(r.history['loss'], label='train loss')
plt.plot(r.history['val_loss'], label='val loss')
plt.legend()
plt.show()
plt.savefig('LossVal_loss')

# plot the accuracy
plt.plot(r.history['accuracy'], label='train acc')
plt.plot(r.history['val_accuracy'], label='val acc')
plt.legend()
plt.show()
plt.savefig('AccVal_acc')

from tensorflow.keras.models import load_model
model.save('/content/drive/MyDrive/tomato/ResNet50.h5')

```

+ Code + Text Connect ^

```

Epoch 4/20
314/314 [=====] - 145s 463ms/step - loss: 1.8329 - accuracy: 0.4642 - val_loss: 2.1796 - val_accuracy: 0.4460
Epoch 5/20
314/314 [=====] - 146s 464ms/step - loss: 1.7521 - accuracy: 0.4908 - val_loss: 2.5152 - val_accuracy: 0.4240
Epoch 6/20
314/314 [=====] - 144s 457ms/step - loss: 1.6522 - accuracy: 0.5283 - val_loss: 1.8755 - val_accuracy: 0.5110
Epoch 7/20
314/314 [=====] - 144s 459ms/step - loss: 1.6030 - accuracy: 0.5362 - val_loss: 1.4632 - val_accuracy: 0.5520
Epoch 8/20
314/314 [=====] - 143s 454ms/step - loss: 1.5548 - accuracy: 0.5436 - val_loss: 2.3122 - val_accuracy: 0.4340
Epoch 9/20
314/314 [=====] - 148s 470ms/step - loss: 1.6061 - accuracy: 0.5473 - val_loss: 1.5996 - val_accuracy: 0.5360
Epoch 10/20
314/314 [=====] - 144s 457ms/step - loss: 1.5507 - accuracy: 0.5616 - val_loss: 2.4007 - val_accuracy: 0.4580
Epoch 11/20
314/314 [=====] - 144s 458ms/step - loss: 1.7014 - accuracy: 0.5451 - val_loss: 1.9556 - val_accuracy: 0.5310
Epoch 12/20
314/314 [=====] - 144s 458ms/step - loss: 1.4521 - accuracy: 0.5867 - val_loss: 2.3030 - val_accuracy: 0.4370
Epoch 13/20
314/314 [=====] - 145s 461ms/step - loss: 1.5513 - accuracy: 0.5719 - val_loss: 1.7673 - val_accuracy: 0.5600
Epoch 14/20
314/314 [=====] - 144s 459ms/step - loss: 1.4191 - accuracy: 0.5947 - val_loss: 2.3201 - val_accuracy: 0.5490
Epoch 15/20
314/314 [=====] - 144s 459ms/step - loss: 1.5518 - accuracy: 0.5769 - val_loss: 2.0747 - val_accuracy: 0.5470
Epoch 16/20
314/314 [=====] - 145s 462ms/step - loss: 1.5095 - accuracy: 0.5916 - val_loss: 1.7704 - val_accuracy: 0.5640
Epoch 17/20
314/314 [=====] - 145s 462ms/step - loss: 1.4514 - accuracy: 0.5955 - val_loss: 2.1103 - val_accuracy: 0.5370
Epoch 18/20
314/314 [=====] - 144s 458ms/step - loss: 1.3935 - accuracy: 0.6065 - val_loss: 2.1384 - val_accuracy: 0.5650
Epoch 19/20
314/314 [=====] - 145s 462ms/step - loss: 1.4937 - accuracy: 0.5990 - val_loss: 1.7984 - val_accuracy: 0.5850
Epoch 20/20
314/314 [=====] - 146s 464ms/step - loss: 1.3901 - accuracy: 0.6061 - val_loss: 1.5070 - val_accuracy: 0.6210

```


5. # Program to be executed on Raspberry Pi

```
import cv2

import numpy as np

import tensorflow as tf

from tensorflow.keras.models import load_model


# Load the pre-trained model

model = load_model('/home/pi/inceptionResNetV2.h5')


# Set up the camera

camera = cv2.VideoCapture(0) # 0 for the default camera


# Define the class labels for disease and healthy

class_labels = ['Tomato__healthy', 'Tomato__Tomato_mosaic_virus',
'Tomato__Tomato_Yellow_Leaf_Curl_Virus',
'Tomato__Target_Spot', 'Tomato__Spider_mites', 'Two-
spotted_spider_mite', 'Tomato__Septoria_leaf_spot',
'Tomato__Leaf_Mold', 'Tomato__Late_blight',
'Tomato__Early_blight', 'Tomato__Bacterial_spot']


# Capture and process frames

while True:

    # Capture frame from the camera

    ret, frame = camera.read()


    # Flip the frame horizontally

    frame = cv2.flip(frame, 1)
```

```

# Resize the frame to a smaller size for faster processing
resized_frame = cv2.resize(frame, (224, 224))

# Preprocess the image
preprocessed_frame = resized_frame.astype('float32') / 255.0
preprocessed_frame = np.expand_dims(preprocessed_frame, axis=0)

# Perform inference using the pre-trained model
predictions = model.predict(preprocessed_frame)
class_index = np.argmax(predictions[0])
label = class_labels[class_index]
confidence = predictions[0][class_index]

# Draw the label and confidence on the frame
label_text = f'{label} ({confidence:.2f})'
cv2.putText(frame, label_text, (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 255, 0), 2)

# Display the frame
cv2.imshow('Leaf Disease Detection', frame)

# Break the loop if 'q' is pressed
if cv2.waitKey(1) & 0xFF == ord('q'):
    break

```

```
# Release the camera and close all windows
```

```
camera.release()
```

```
cv2.destroyAllWindows()
```

```
%Run project.py
```

APPENDIX III

User Manual & Tutorial

PRE-INSTALLATION REQUIREMENTS

- a. Python
- b. Raspberry Pi 4
- c. Webcam
- d. The following libraries
 - i. Tensorflow
 - ii. cv2
 - iii. numpy
- e. Laptop