Tribhuvan University
Amrit Campus

Final Year Project Report
on
PlantMD: A Plant Disease Prediction System
[Course Code: CSC 412]

Under the supervision of
Yuba Raj Devkota
Lecturer

Submitted by
Anurag Limbu Kandangwa (20124/075)
Ashok Tripathi (20128/075)
Rabin Babu Tiwari (20173/075)

Submitted to
Department of Computer Science and Information Technology
Amrit Campus
Institute of Science and Technology
Tribhuvan University
May 16, 2023

Tribhuvan University
Amrit Campus

Final Year Project Report
on
PlantMD: A Plant Disease Prediction System
[Course Code: CSC 412]

A final year project submitted in partial fulfillment of the requirement for the degree of Bachelor of Science in Computer Science and Information Technology awarded by Tribhuvan University

Submitted by
Anurag Limbu Kandangwa (20124/075)
Ashok Tripathi (20128/075)
Rabin Babu Tiwari (20173/075)

Submitted to
Department of Computer Science and Information Technology
Amrit Campus
Institute of Science and Technology
Tribhuvan University
May 16, 2023

Tribhuvan University
Institute of Science and Technology
Amrit Campus
Department of Computer Science and Information Technology
Email: csitascol@gmail.com

Recommendation Letter of Supervisor

I hereby recommend that the project prepared under my supervision by Mr. Yuba Raj Devkota entitled "PlantMD: A Plant Disease Prediction System" be accepted as fulfilling in partial requirement for the degree of Bachelors of Science in Computer Science and Information Technology. In my best knowledge, this is an original work in Computer Science and Information Technology.

.....

Mr. Yuba Raj Devkota
Supervisor
Lecturer
Amrit Campus

Certificate of Approval

This is to certify that this project prepared by Anurag Limbu Kandangwa, Ashok Tripathi and Rabin Tiwari, entitled "PlantMD: A Plant Disease Prediction System" in partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science and Information Technology has been well studied. In our opinion, it is satisfactory in the scope and quality as a project for the required degree.

.....
Mr. Yuba Raj Devkota
Supervisor
Lecturer
Amrit Campus

.....
External Examiner

.....
Dr. Binod Kumar Adhikari
Head of Department
Lecturer
Amrit Campus

Acknowledgement

It gives us immense pleasure to express our deepest sense of gratitude and sincere thanks to our highly respected and esteemed guide Mr. Yuba Raj Devkota, Department of Computer Science and Information Technology, Amrit Campus, for his valuable guidance, encouragement and help for completing this work. His useful suggestions for this whole work and co-operative behavior are sincerely acknowledged.

We would like to express our sincere thanks to Mr. Yuba Raj Devkota, Department of Computer Science and Information Technology, Amrit Campus, for giving us this opportunity to undertake this project. We would also like to thank Dr. Binod Kumar Adhikari, Asst. Prof, Head of Department of BSc.CSIT, Amrit Campus, for his whole hearted support.

We are also grateful to our teachers for their constant support and guidance.

At the end we would like to express our sincere thanks to all our friends and others who helped us directly or indirectly during this project work.

Anurag Limbu Kandangwa (20124/075)
Ashok Tripathi (20128/075)
Rabin Babu Tiwari (20173/075)

Abstract

Plant diseases pose a significant threat to global agriculture, affecting crop yields and food security. In this final-year project, we address the challenges of limited data availability and real-world applicability in the domain of machine learning and deep learning for plant disease classification, detection, and generation. To tackle these issues, we fine-tuned a deep Convolutional Neural Network (CNN) model on a real-world dataset and employed various object detection models to accurately identify and classify plant diseases. Furthermore, we explored the potential of image generation using different variants of Generative Adversarial Networks (GANs) to overcome the constraints of limited data. Our work demonstrates the feasibility of applying advanced machine-learning techniques to enhance plant disease detection and classification, ultimately contributing to more sustainable and efficient agricultural practices.

In addition to the technical advancements made in plant disease classification, detection, and generation, our approach has successfully enabled the deployment of web-based and mobile-based applications for real-world scenarios. This practical implementation allows users, including farmers and agricultural experts, to easily access and utilize our advanced machine-learning techniques for plant disease identification and management. By bridging the gap between research and real-world application, our project contributes to the development of more accessible and effective tools for addressing plant disease challenges, ultimately promoting sustainable and efficient agricultural practices.

Keywords: machine learning, deep learning, real-world applicability, Convolutional Neural Network (CNN), object detection models, Generative Adversarial Networks (GANs), mobile-based applications.

Contents

List of Figures

List of Abbreviations

List of Tables

1. Introduction

1.1 Introduction

Approximately 40 percent of global crop production is lost to pests. Each year, plant diseases cost the global economy over $220 billion, and invasive insects at least $70 billion [1]. In Nepal, 25-35 percent of yield loss is caused by insect pests and diseases [2]. Plant Diseases are one of the major reasons influencing food production, causing a negative hit on the country's economy and farmers' livelihood as 65 % of the total population of Nepal is involved in agriculture. Furthermore, it may lead to food shortages and become a threat to the rising population. It is important for farmers to be aware of the risks posed by plant diseases and to take steps to protect their crops. Early disease detection is mandatory to take preventive measures and increase the quality & quantity of crop production.

Hence, our project introduces **PlantMD**, an abbreviation for Plant Medic, which is a **a plant disease prediction system**. The project is a Web Application Programming Interface (API) application that takes input as an image of the plant and provides the result as the detected disease, and preventive measures. Furthermore, we have a mobile application utilizing the API to provide an easy-to-use interface for the user.

Essentially, we aim to develop a Machine Learning (ML) model which uses the computer vision technique for plant disease prediction that will be applicable in real-world scenarios through the help of mobile devices. Many computer vision algorithms [3]–[9] have been developed for classifying and recognizing objects with high accuracy. Despite the success of the above algorithms, many challenges [10] are present when applying these algorithms to detect plant diseases. In order to overcome the above challenges, we have used various classification and detection algorithms and compare their performance and inference time. We have evaluated our experiment on the PlantDoc [11] dataset and also use the PlantVillage [12] dataset to further improve the accuracy of our models.

1.2 Problem Statement

Plant Disease Detection has always been a challenging task because of its limited use in real-world scenarios. Most of the work on such systems that detect the disease in real-world scenarios is mostly limited to research works. While some of these systems do exist, they suffer from the problem of either not being deployed or not being easily accessible and most of the time are not easy to use. Also, the dataset used by most of them does not reflect real-world data. Furthermore, the expensive cost of data collection and labeling limits us from experimenting with a large dataset.

1.3 Objectives

- To build an ML model that detects plant disease using a small dataset while achieving acceptable accuracy in real-world scenarios.
- To study the effectiveness of synthetic data generated by variants of Generative Adversarial Network (GAN)s like Deep Convonutional Generative Adversarial Network (DCGAN) and Progressive Growing GAN (ProGAN) in training samples.

1.4 Scope and Limitation

1.4.1 Scope

A plant disease detection system is a tool or system designed to identify and diagnose plant diseases. The scope of such a system depends on the specific goals and objectives of the system, as well as the resources available for its development and operation. The scope of our plant disease detection project includes:

- **Identification of plant diseases:** The system should be able to identify the specific disease affecting a plant, based on visible symptoms and other characteristics.
- **Diagnosis of plant diseases:** The system should be able to accurately diagnose the disease, based on a combination of visible symptoms, laboratory tests, and other diagnostic tools.
- **Prediction of plant disease outbreaks:** The system should be able to predict the likelihood of a disease outbreak based on a variety of factors, such as the presence of disease-causing pathogens, environmental conditions, and other factors.
- **Detection of plant diseases in the early stages:** The system should be able to detect diseases in the early stages, when they are most treatable, to help prevent the spread of the disease and minimize crop losses.
- **Detection of plant diseases in large fields:** The system should be able to detect diseases in large fields, using sensors, drones, or other remote sensing technologies, to help farmers and agronomists identify and treat diseased plants quickly.
- **Integration with other tools and systems:** The system should be able to integrate with other tools and systems, such as weather forecasting systems or irrigation systems, to help optimize disease management strategies.

1.4.2 Limitation

- Larger models lead to an increase in inference time which might not be desirable for some end users.
- The accuracy of the model suffers due to the diversity within the dataset.
- Lack of specialization in the application domain.
- Tedious and expensive cost of data collection and labeling.
- Lack of accurate detection in offline mode due to the computational constraint of the mobile device.

1.5 Development Methodology

Our system involved many parts working in tandem with each other to function as a whole system. For example, the Web API needs the ML prediction model to produce results while the Mobile/Web app requires the Web API to be in place to give input and show results. While these parts are dependent on each other, they didn't require the other to be completely functional and could be built in parallel. Hence, we used the Agile Development Methodology for developing our system. Gantt chart can be referred to in Appendix A, Figure A.1.

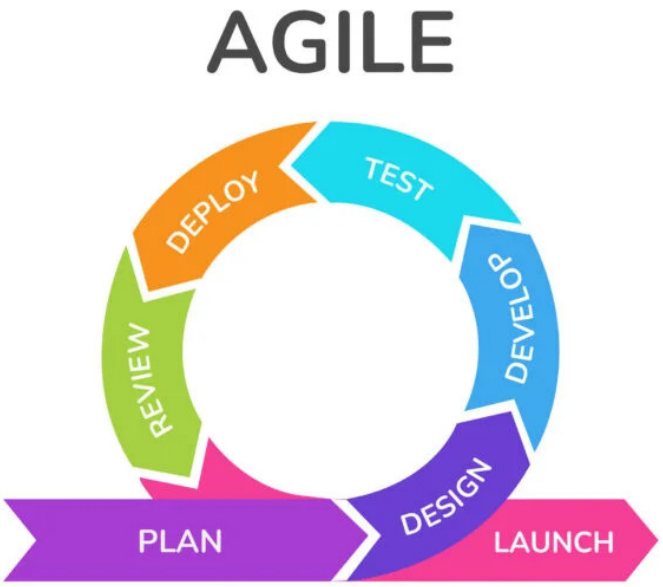


Figure 1.1: Agile Software Development Methodology

How we will implemented the phases of this methodology is described below.

- **Plan:** We planned the next features to be implemented in our system.
- **Design:** We thought of how the solution of the next features will be implemented and how problems will be solved.
- **Develop:** One team member was developing the ML model, another member was developing the Web API and another member was developing the Web/-Mobile app.
- **Test:** At the end of the sprint, all the parts were tested to ensure the features were implemented according to design.
- **Deploy:** The code of all the members at the end of the sprint was pushed into production after all tests had passed.
- **Review:** Finally at the end of the sprint, the system was reviewed to check if the parts were working together properly and if the planning and design goals were met.

1.6 Report Structure

- **Introduction** (Introduction, Problem Statement, Objectives, Scope and Limitation, Development Methodology)
- **Literature Review** (Study of research paper, Comparing system with similar system)
- **System Analysis** (Proposed System, Functional Requirements, Non Functional Requirements, Feasibility Analysis)
- **System Design** (Design of the System)
- **Algorithm Details** (Classification Algorithm, Grad-CAM, Detection Algorithm, Generation Algorithm)
- **Implementaion and Testing** (Tools Used, Implementation details of Modules, Testing)
- **Result Analysis**
- **Conclusion**

2. Literature Review

2.1 Study of Research Papers

Plant disease prediction can be categorized into two subproblems namely Plant Disease Classification and Plant disease detection. Plant disease classification refers to the categorization of plant leaves into one of the predefined classes whereas Plant disease detection is the process of identifying the presence of plant disease in a plant or group of plants. Plant disease prediction can be solved using either ML-based techniques or Deep Learning (DL)-based techniques. Plant disease generation is the technique of generating new synthetic images that can be used for training purposes to make the model more robust and accurate.

Previously, Image processing techniques were widely used to detect plant diseases. It uses hand-crafted features such as colors and texture features to train machine learning models. Fermi energy-based segmentation method was used to isolate the infected region of the image from its background [13]. Based on the segmented image features like color, shape, and position were extracted and passed through the rule-based classification for disease identification of rice plants. Sandika [14] proposed a feature-based approach for the classification of grape disease. They also separated the complex background from disease patches and classified the different diseases based on different texture features like local texture filters, Local Binary Patterns, Gray-Level Co-Occurrence Matrix (GLCM) features, and some statistical features in the Red Green Blue (RGB) plane. They compared the performance of four machine learning algorithms, Probabilistic Neural Network (PNN), Back propagation Neural Network (BPNN), Support Vector Machine (SVM) & Random Forest, and used Random Forest and GLCM features.

Mohanty [15] trained a deep convolutional neural network to identify 14 crop species and 26 diseases. The trained model achieved an accuracy of 99.35% on a held-out test set, demonstrating the feasibility of this approach. Ramcharan [16] proposed a deep Convolutional Neural Network (CNN) to identify three diseases in Cassava. The research showed that the transfer learning approach for image recognition of field images offers a fast, affordable, and easily deployable strategy for digital plant disease detection. Singh [11] used deep learning methods for classification and detection. They also released a real-world data set which was a major challenge for enabling vision-based plant disease detection. In 2017, Fuentes [17] used the object detection approach for plant disease detection. Faster Region-based Convolutional Neural Network (R-CNN), Region-based Fully Convolutional Network (R-FCN), and Single Shot Multibox Detector (SSD) was used to locate tomato disease and pets directly, combined with deep feature extractors such as Visual Geometry Group Network (VGG-Net) and Residual Network (ResNet). The authors proposed a system that can effectively recognize nine different types of diseases and pests, with the ability to deal with complex scenarios from a plant's surrounding area. Sun [18] proposed a technique that incorporates three major steps: the data set preprocessing part, fine-tuning network, and the detection module. Their proposed system achieved the

Mean Average Precision (mAP) of 91.83%. Prakruti [19] proposed a method to locate the health condition of tea leaves in a complex background and with occlusion. They achieved the mAP score of 86% at 50% Intersection Over Union (IOU) using the You Only Look Once version 3 (YOLOv3) model.

Different variants of GAN like Conditional Generative Adversarial Network (CGAN), DCGAN, ProGAN, and StarGAN have been shown to generate synthetic images for various use cases. These methods can be used for augmentation for improved plant disease detection. Hu [20] used Conditional Deep Convonutional Generative Adversarial Network (C-DCGAN) to generate new training samples which are used to train Visual Geometry Group (VGG) 16 deep learning models to identify the tea leaf's disease. The author also stated that models trained with augmented disease spot images can identify tea leaf's diseases accurately and the accuracy of the proposed method reached 90%. Zhang [21] exploited the DCGAN to augment the date fruits dataset. Their study showed the classification using DCGAN with the Mobile Neural Network (MobileNet) model achieved 88% accuracy.

2.2 Comparing accuracy of reviewed papers

Table 2.1: CNN model accuracy table.

Model	Optimizer	Learning Rate	Accuracy
Inception ResNetv2	SGD	0.005	66
MobileNet	SGD	0.0005	70

Table 2.2: Object detection model accuracy table.

Model	mAP
Faster_rcnn_resnet101	45.43
SDD_Mobilenetv2	41.54
YOLOv5	55.67

3. System Analysis

3.1 Proposed System

The system will be a fully-fledged application that can take photos of plant leaves and send them to our Web API that runs the prediction model behind the scenes to send the prediction results back to the user. The user will then be shown the prediction results in the application itself.

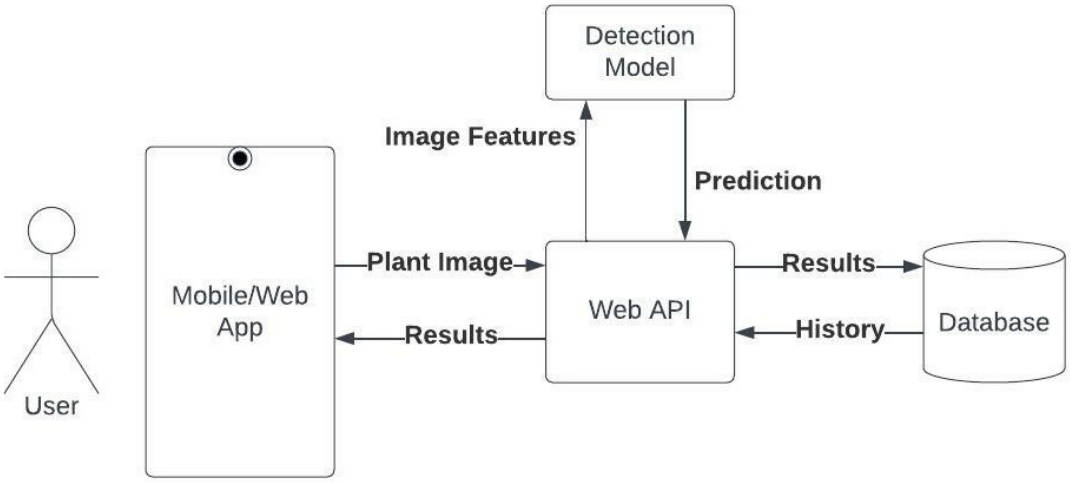


Figure 3.1: Working mechanism of proposed system

Figure 3.1 shows us a diagram that describes the working mechanism of our proposed system. The user will be using the Mobile app or Web app to send images to our Web API which will then be passed to the ML model. The model will return the prediction results which will be stored in the database and then our API will return the result to the app. Finally, the app will display the results to the user.

3.2 Requirement Analysis

3.2.1 Requirement Collection

We used the observation methodology for our requirement collection where we observed the existing systems and studied the research work to come up with the requirements for our project to overcome the existing limitations of systems and models. Hence, keeping our findings as the basis, the following are the requirements of our project.

- A real-world dataset to train the model for robustness and high accuracy in real-world data.
- Virtual Machine (VM) for the training of compute-intensive deep learning models.
- Virtual Private Server (VPS) for faster inference.
- User data collection such as geolocation, time, and date.
- Web-version to increase the reach of our system.

3.2.2 Functional Requirement

The functional requirements for the project are listed as follows:

- The system should contain an AI model that is trained to predict plant diseases based on the images uploaded by the user. The AI model should analyze the uploaded image and provide the user with relevant information about the plant disease.
- The app should allow the user to take a photo of a plant or a group of plants and upload it to the app.
- The app should provide recommendations to the user on how to treat the detected disease.
- The Web API should have an authentication system to ensure that only registered devices can hit the Web API.
- The system should integrate with external systems like weather forecasting services, pest control agencies, and other relevant systems to provide users with up-to-date information about plant diseases and prevention measures.
- The PlantMD app should support multiple platforms like Android, iOS, and the web.
- The app should be able to take high resolution photos.

3.2.3 Non-Functional Requirements

The non-functional requirements for the project are listed as follows:

- **Performance:** The PlantMD app should provide fast and responsive performance, with quick image upload and prediction results within a few seconds.
- **Scalability:** The system should be scalable to handle a large number of users and images without any performance issues or downtime.
- **Availability:** The app should be available 24/7, with minimal downtime for maintenance or updates.
- **Reliability:** The app should be reliable and accurate in predicting diseases and providing recommendations to ensure user satisfaction.
- **Usability:** The PlantMD app should be easy to use, with a simple and intuitive interface that users can navigate easily.
- **Compatibility:** The app should be compatible with different mobile devices, operating systems, and web browsers.
- **Maintainability:** The app should be designed in a way that is easy to maintain and update, with minimal disruptions to user experience.
- **Performance Testing:** The app should undergo regular performance testing to identify and fix any bottlenecks that may affect user experience.
- **Documentation:** The app should come with comprehensive documentation on its features, functionality, and usage guidelines to help users navigate the app with ease.
- **Portability:** The app should be portable, allowing users to access their profiles and disease prediction results across different devices and platforms.

3.3 Feasibility Analysis

3.3.1 Technical

Our system is going to be used mainly on mobile devices while the backend is web-based due to which it has the following technical needs:

- Reliable internet connection
- Mobile device
- High computational need to train and host our model

In recent times, technology to support such needs has progressed a lot and is easily accessible. The available technologies that align with our technical factors are described below:

- Mobile network coverage and high-speed internet connection are widely available which ensures a reliable internet connection.
- VMs are available on demand with high customization which provides good computing power for training our model.
- Lots of VPS service providers exist in the market that allows us to easily configure our servers according to our needs for deploying our model.
- Mobile devices have become powerful enough to provide in-device inference.
- Progressive Web App (PWA) allows us to deliver mobile apps as web apps and can support low-end devices through the browser.
- Web API allows the backend to be accessed and used by any kind of tech stack.

Since the technical needs of our project can be fulfilled by the above technologies on both the software and hardware side, our project is technically feasible.

3.3.2 Operational

Contrary to the trained staff of any tailored system, our system is going to be used by a wide array of people from general-purpose users to users with low technical literacy. Keeping our audience/users in mind, our system has the following features:

- Simple, easy-to-use, and intuitive user interface.
- Users can get the **result in just two taps**.
- Fast inference and response times for the users to get results.
- Fallback mode for offline usage in which the detection is done by the local model inside the app instead of the server.
- Free to use and open to the public.

The system is easy to operate and provides accurate results quickly while being publicly and freely accessible. Hence, the project is operationally feasible.

3.3.3 Economic

Training our deep learning model in VM and a VPS for deploying our model to have fast inference times are part of our project which will be the majority of our cost. While the cost of VMs is expensive but affordable and also the cost of deploying the model on a VPS has become reasonable nowadays. We can get our model up and running at a very affordable and reasonable cost while gaining good performance and reasonable inference time for our deployed system.

The long-term gains from these services outweigh the cost in the following ways:

- We will get to collect more plant images and data to enhance our model which is absolutely necessary.
- We can use the model commercially in large-scale farms.
- Using such services allows us to reach a wider user base.

Since the cost of our system allows us to scale our service as needed and improve our model in the long run, our project is economically feasible.

3.3.4 Schedule

The estimated timespan for completing our project is 4 months. Hence, we will be using cross-platform app development framework like Flutter, React for front end and FastAPI python library for building our Web API. Since, all these frameworks and libraries are widely used and popular, have good documentation and a big community suppot to solve errors, using them will shorten our development time into the estimated time. Therefore, our system's schedule is feasible.

4. System Design

4.1 Design of the System

4.1.1 Flowchart

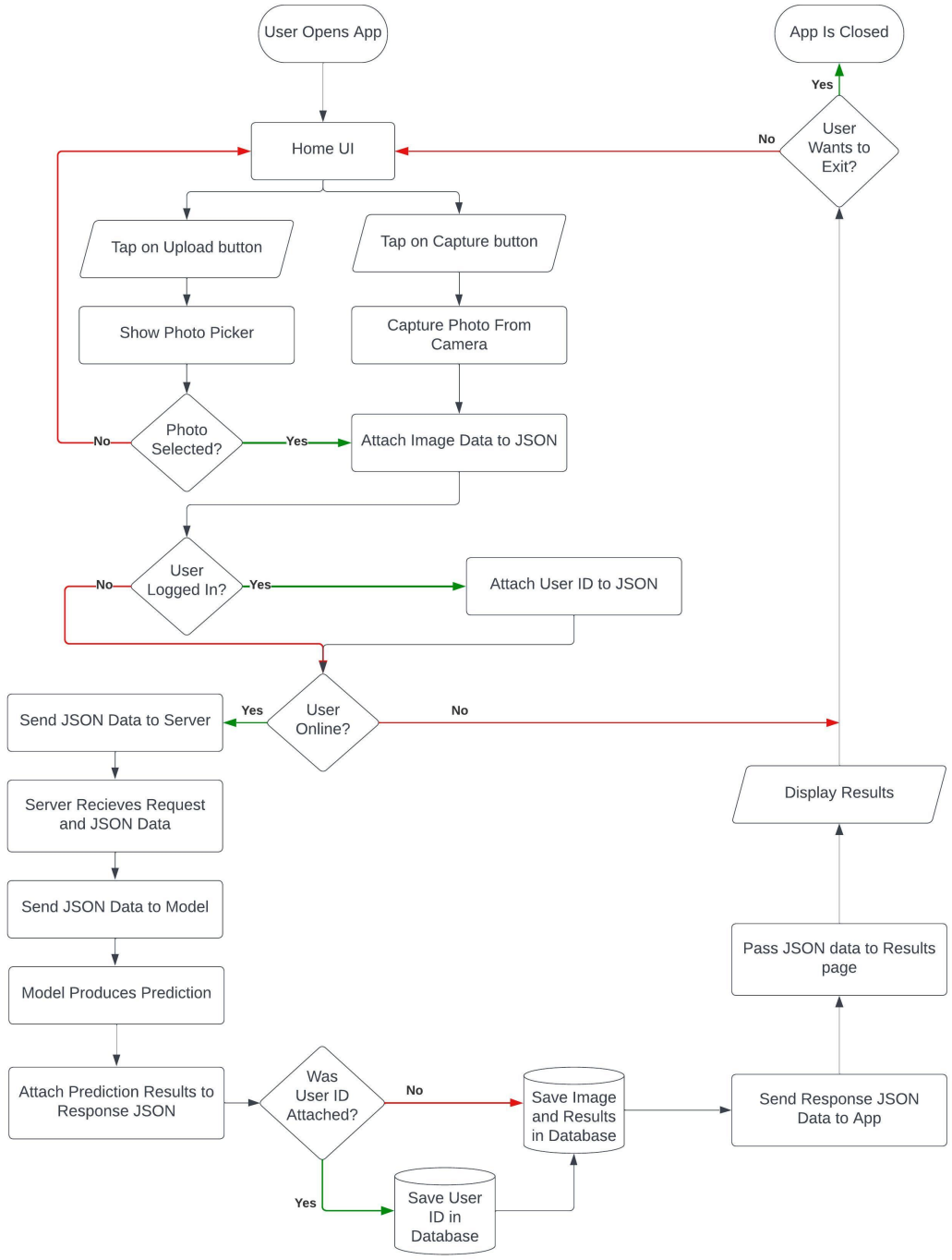


Figure 4.1: High level flowchart of the system

4.1.2 Use Case Diagram

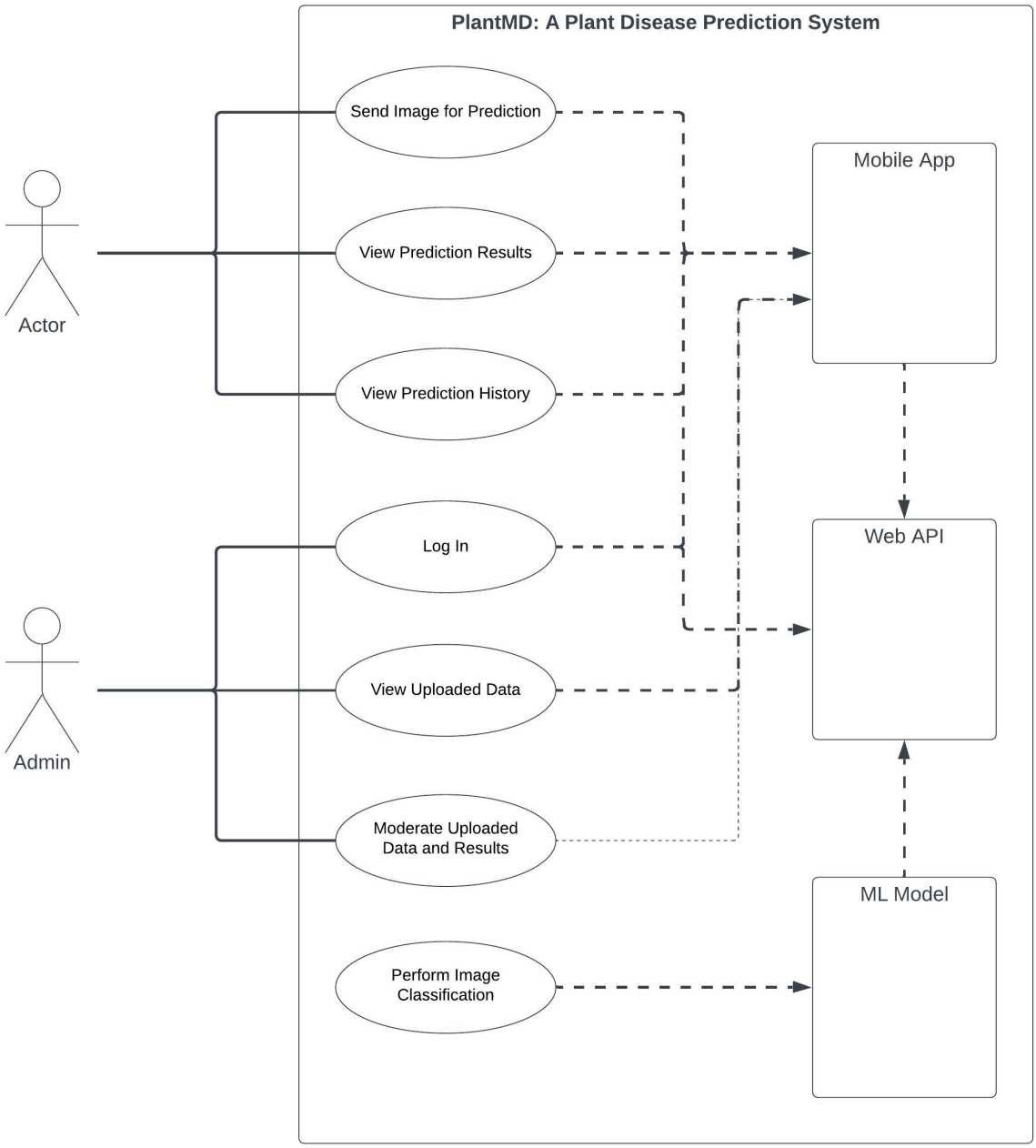


Figure 4.2: Use Case diagram of the system

4.1.3 ER Diagram

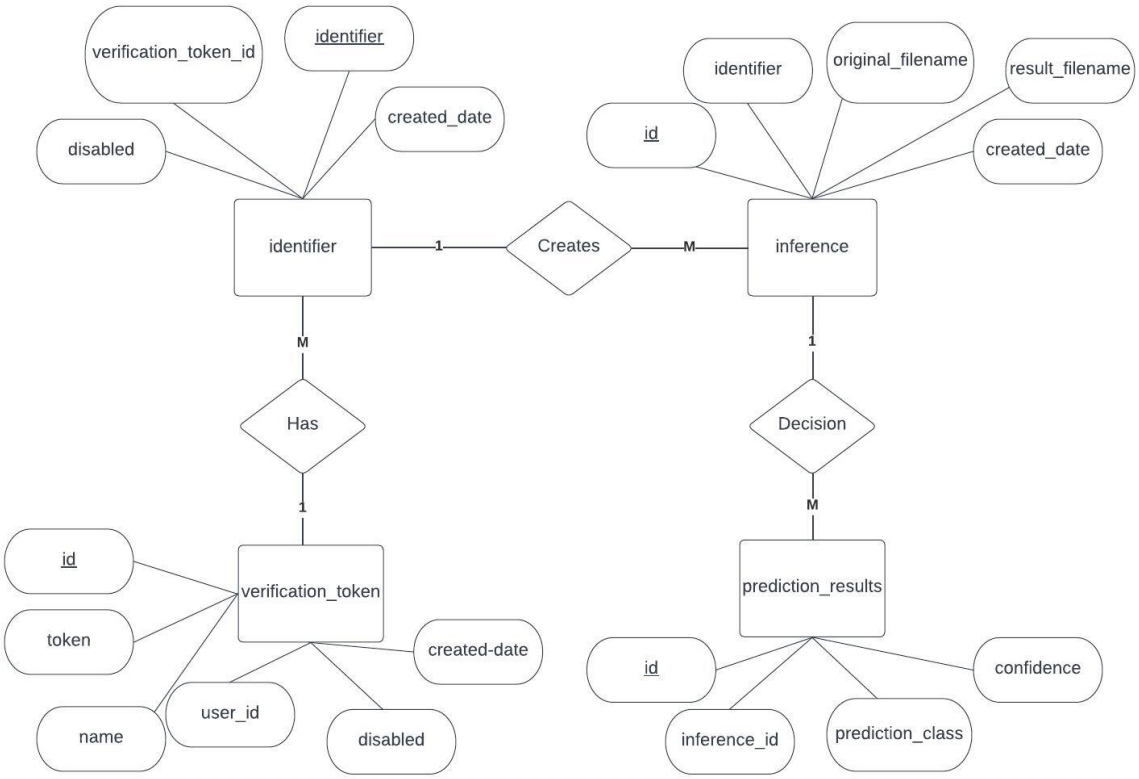


Figure 4.3: ER diagram of the system

4.1.4 Data Flow Diagram: Level 0

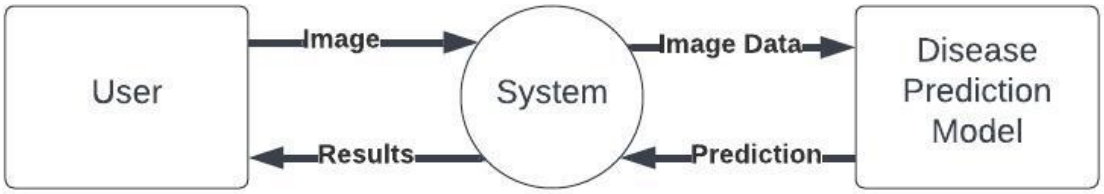


Figure 4.4: Data Flow Diagram: Level 0 of the proposed system

4.1.5 Data Flow Diagram: Level 1

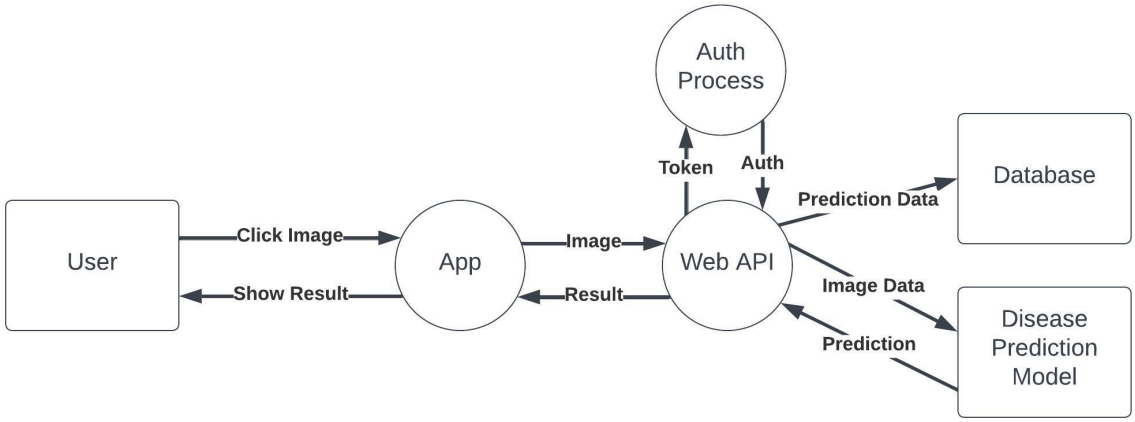


Figure 4.5: Data Flow Diagram: Level 1 of the proposed system

4.1.6 Data Flow Diagram: Level 2

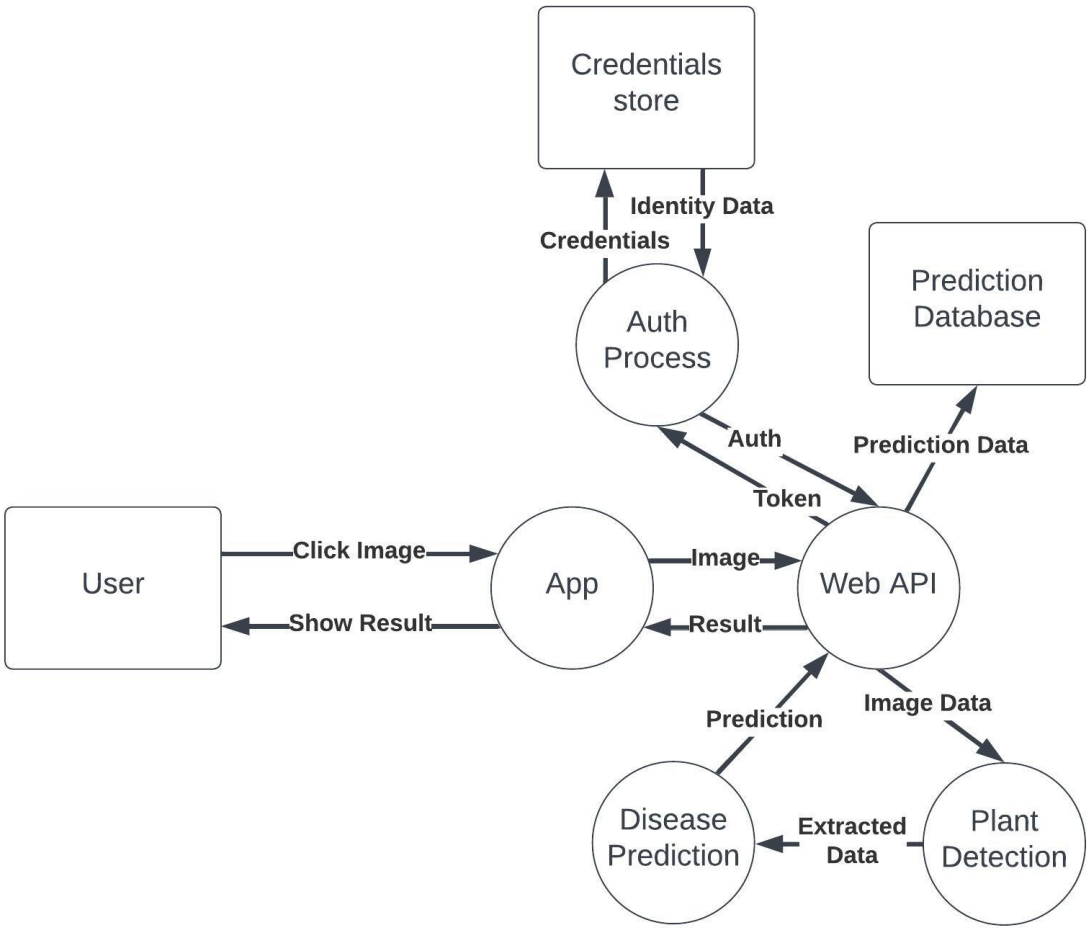


Figure 4.6: Data Flow Diagram: Level 2 of the proposed system

4.1.7 Sequence Diagram

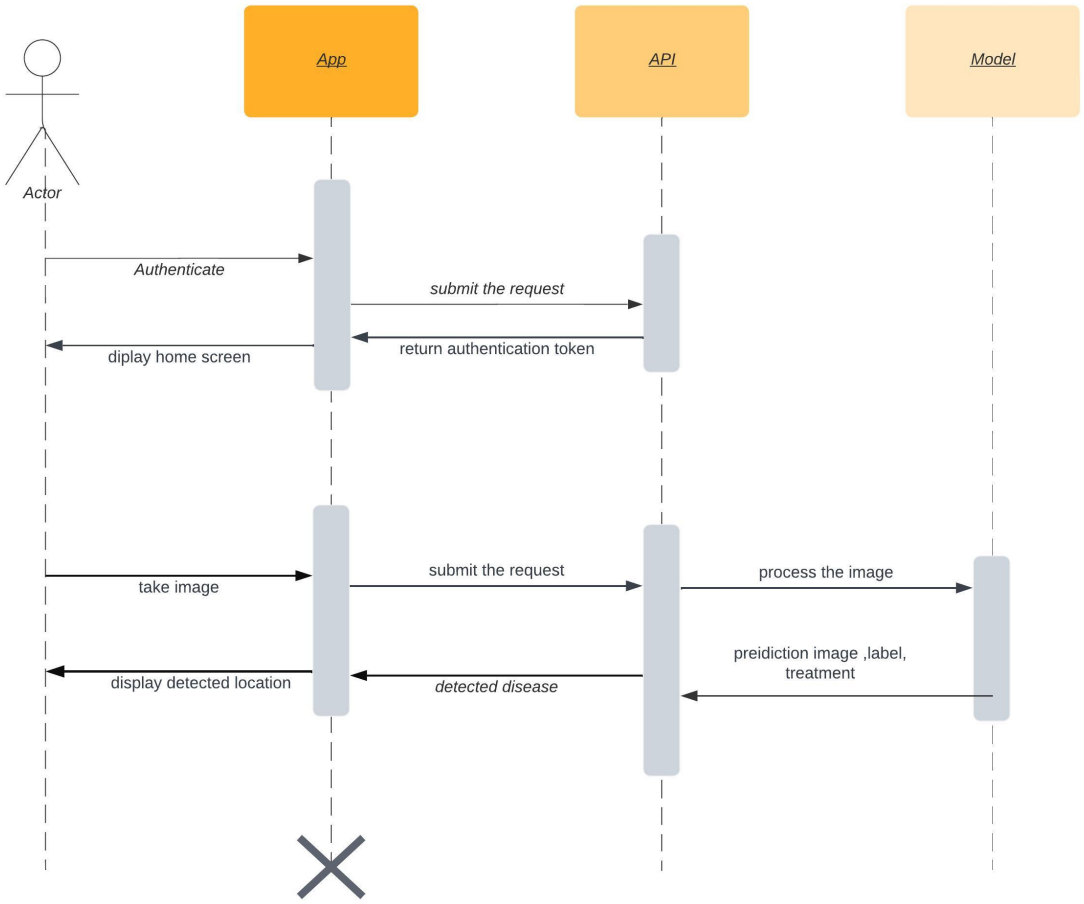


Figure 4.7: Sequence diagram of the proposed system

5. Algorithms Details

Four different types of algorithms are used in this project, which are:

1. Classification Algorithm
2. Grad-CAM
3. Detection Algorithm
4. Generation Algorithm

5.1 Classification Algorithm

The CNN is a type of artificial neural network primarily used for image classification, object detection, and computer vision tasks. It is specifically designed to process and analyze images by mimicking the way the human visual system works.

In a CNN, the input image is passed through multiple layers of interconnected processing nodes or neurons, which form the network. These layers include:

i. **Convolutional Layers:** These layers apply a set of filters or convolutional kernels to the input image, resulting in feature maps that capture the presence of specific features in the original image (like edges, corners, textures, etc.). The convolution operation is a mathematical process that combines the input image with the filters to generate these feature maps.

ii. **Activation Functions:** After convolution, an activation function like the Rectified Linear Unit (ReLU) is applied to introduce non-linearity into the network. This allows the CNN to learn and model more complex patterns in the data.

iii. **Pooling Layers:** These layers are used to reduce the spatial dimensions of the feature maps, making the model less computationally intensive and more robust to small changes in the input image. Common pooling operations include max pooling (taking the maximum value in a pooling window) and average pooling (taking the average value in a pooling window).

iv. **Fully Connected Layers:** After the convolution and pooling layers, the feature maps are flattened into a one-dimensional vector and passed through one or more fully connected layers. These layers perform the final classification task by learning high-level features and relationships between them. The output of the last fully connected layer is passed through a softmax activation function to generate class probabilities.

v. **Loss Function and Optimization:** The difference between the predicted class probabilities and the actual class labels is calculated using a loss function, such as categorical cross-entropy. The neural network's weights are then updated during the training process using optimization algorithms like stochastic gradient descent or Adam optimizer to minimize the loss.

In this project, we have used two CNN model for the fine tuning purpose.

5.1.1 Inception Resnet v2

Inception-ResNet-v2 is a deep convolutional neural network architecture that combines the ideas from the Inception architecture and the Residual Network (ResNet) architecture. It was introduced by Christian Szegedy, Sergey Ioffe, and Vincent Vanhoucke from Google Research [5]. The Inception-ResNet-v2 model achieves state-of-the-art performance in various image classification benchmarks, such as ImageNet.

Inception-ResNet-v2 combines the strengths of the Inception architecture, which focuses on increasing the network's depth and width while maintaining computational efficiency, with the ResNet architecture, which uses residual connections (also known as skip connections) to improve gradient flow and optimize training in deeper networks.

Key components of the Inception-ResNet-v2 architecture include:

i. **Inception Modules:** The model consists of several stacked Inception modules that are responsible for learning a diverse set of features at different scales. These modules employ multiple parallel convolutional branches with different kernel sizes and combinations, which helps the network to capture both local and global information in the input images.

ii. **Residual Connections:** Inception-ResNet-v2 integrates residual connections into the Inception modules, which help to alleviate the vanishing gradient problem in deep networks. These connections create shortcuts between layers, allowing the gradients to flow more easily through the network during the backpropagation process. This results in improved training and better generalization.

iii. **Stem Layers:** The model starts with a "stem" block, which comprises a series of convolutional and pooling layers. The stem block is responsible for reducing the input image's size and increasing the number of feature maps before feeding them into the Inception-ResNet modules.

iv. **Global Average Pooling and Softmax:** After passing through the Inception-ResNet modules, a global average pooling layer is used to reduce each feature map's spatial dimensions to a single value. This results in a feature vector, which is then passed through a fully connected layer and softmax activation function to generate the final class probabilities.

Inception-ResNet-v2 has proven to be a powerful architecture for image classification tasks, offering high accuracy and comparatively lower computational complexity compared to other deep learning models. It has been adopted in various applications, including object detection, image segmentation, and transfer learning for domain-specific tasks.

5.1.2 MobileNet

MobileNet is a family of efficient and lightweight convolutional neural networks designed for mobile and embedded devices with limited computational resources. It was introduced by Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam from Google Research [6]. MobileNet models achieve competitive performance in various computer vision tasks, such as image classification, object detection, and semantic segmentation, while requiring less computation and memory compared to deeper and more complex networks.

The key innovations in the MobileNet architecture are the use of depth wise separable convolutions and the introduction of width and resolution multipliers to create a family of models with different trade-offs between accuracy and computational complexity.

i. **Depthwise Separable Convolution:** Traditional convolutional layers involve both spatial and channel-wise feature mixing, which can be computationally expensive. MobileNet replaces these layers with depth wise separable convolutions, which factorize the convolution process into two steps: depthwise convolution and pointwise (1x1) convolution. Depthwise convolution applies a single filter per input channel, while pointwise convolution combines the outputs from depthwise convolution across channels. This approach significantly reduces the number of parameters and computations in the network, making it more efficient and lightweight.

ii. **Width Multiplier:** The width multiplier is a hyperparameter that allows the creation of a thinner MobileNet model by reducing the number of channels in each layer. By choosing a smaller width multiplier, the number of parameters and computations in the network decreases, resulting in a smaller and faster model at the cost of reduced accuracy.

iii. **Resolution Multiplier:** The resolution multiplier is another hyperparameter that controls the input image's resolution fed into the network. By reducing the input resolution, the computational cost of the network decreases, making it faster but with a potential reduction in accuracy.

MobileNet architecture comprises a series of depthwise separable convolutional layers, followed by a global average pooling layer and a fully connected layer with a softmax activation function for final classification. The overall structure is similar to other convolutional neural networks, but the use of depthwise separable convolutions and the adjustable width and resolution multipliers make MobileNet more efficient and suitable for resource-constrained devices.

MobileNet has been widely used in various applications, including real-time object detection, facial recognition, image segmentation, and transfer learning for domain-specific tasks. Due to its efficiency and lightweight design, MobileNet is particularly well-suited for deployment on mobile devices, IoT devices, and edge computing systems, where computational resources and power consumption are critical factors.

5.2 Grad-CAM

Interpretability is a relation between formal theories that express the possibility of interpreting or translating one into another (from Wikipedia). Interpretability in machine learning and deep learning models has been a major concern for a long time. The traditional machine learning algorithm is more interpretable while its accuracy and robustness are low and on the other hand deep learning is great at accuracy and robustness but lacks interpretability. So there is a tradeoff between accuracy and interpretability. Deep neural networks are considered black boxes because it's really hard to figure out what is happening to make the output. So interpreting these models can help to gain more trust in predictions.

In 2016, a method called Gradient Weighted Class Activation Mapping (Grad-CAM) was developed to make convolution neural network-based model more transparent by visualizing the region of input that is important for prediction. Class discriminative (i.e localize the category in the image) and high-resolution (i.e capture the fine-grained details) make a good visualization. Grad-CAM is a strict generalization of Class Activation Mapping (CAM). The disadvantage of CAM is you need to feed the output of Global Average Pooling to the softmax layer which hurts the accuracy of the model. Grad-CAM generalizes CAM which allows us to visualize the region of input that is important for any kind of CNN architecture. Grad-CAM computes the gradient of the score for class C with respect to feature map activation of a convolution layer. These gradients are global averaged to obtain the neuron importance weight.

$$\alpha_k^c = \overbrace{\frac{1}{Z}\sum_i\sum_j}^{\text{global average pooling}} \underbrace{\frac{\delta y^c}{\delta A_{ij}^k}}_{\text{gradients via backprop}}$$

The neuron importance weight captures the importance of feature map k for a target class C. Then the linear combination of the forward activation map is performed and followed by ReLU to obtain the result.

$$L^{c}_{Grad-CAM} = ReLU \underbrace{\left(\sum_{k} \alpha^{c}_{k} A^{k} \right)}_{\text{linear combination}}$$

This result is the heatmap of the same size of feature map like 14 x 14. The ReLU activation is used because we are only interested in the features that have a positive influence on the class of interest. The produced heatmap is rescaled and superimposed on the original image to visualize the important regions in the image for prediction.

5.3 Detection Algorithm

Deep Learning based object detection can be classified into two different types:

1. Two-stage detector (such as R-CNN, Fast R-CNN, Faster R-CNN)
2. One-stage detector (such as You Only Look Once (YOLO), SSD)

We will briefly look into both types of algorithms and compare their result on plant disease detection tasks.

5.3.1 Two-stage detector

The R-CNN family is a series of two-stage object detectors that have significantly influenced the field of object detection in computer vision. The two-stage approach involves:

i. **Region Proposal:** Proposing potential object bounding boxes (also known as Regions of Interest or RoIs) within an input image.

ii. **Classification and Bounding Box Regression:** Classifying the proposed regions into object categories and refining the bounding box coordinates.

There are several object detectors in the R-CNN family, including R-CNN, Fast R-CNN, and Faster R-CNN. Each iteration improves upon the previous model's efficiency and accuracy.

5.3.1.1 R-CNN

R-CNN is the initial model that introduced the concept of region-based convolutional neural networks for object detection [22]. It uses selective search, a bottom-up region proposal method, to generate around 2000 RoIs for each input image. Then, a pre-trained CNN (e.g., AlexNet or VGG) extracts features from these regions. Finally, a SVM classifies the regions into object classes, and bounding-box regression refines the box coordinates. However, R-CNN is computationally expensive due to the separate CNN feature extraction for each RoI and reliance on external region proposal methods.

5.3.1.2 Fast R-CNN

Fast R-CNN improves upon R-CNN by introducing RoI pooling, which allows the network to extract features from all RoIs in a single forward pass through the CNN [23]. The input image is first passed through a pre-trained CNN, resulting in a feature map. The RoIs generated by selective search are then mapped onto the feature map, and RoI pooling is applied to extract fixed-size features. These features are processed through fully connected layers, followed by two parallel output layers for classification (using softmax) and bounding box regression. Fast R-CNN is significantly faster than R-CNN but still relies on an external region proposal method, which limits its overall speed.

5.3.1.3 Faster R-CNN

Faster R-CNN further accelerates object detection by introducing the Region Proposal Network (RPN), which generates RoIs directly from the feature map produced by the CNN, eliminating the need for an external region proposal method [24]. The RPN is trained to predict both objectness scores (how likely a region contains an object) and bounding box coordinates for anchor boxes. The RoIs generated by the RPN are then fed into the Fast R-CNN pipeline for classification and bounding box regression. Faster R-CNN achieves near real-time performance while maintaining state-of-the-art detection accuracy.

The R-CNN family of detectors has had a significant impact on the field of object detection, providing a foundation for many subsequent architectures and approaches, such as Mask R-CNN for instance segmentation and Cascade R-CNN for higher-quality bounding boxes.

5.3.2 One-stage detector

One-stage detectors are object detection models that perform object localization and classification in a single, unified framework, offering faster detection speeds compared to two-stage detectors like R-CNN family models. Two popular one-stage detectors are SSD and YOLO.

5.3.2.1 Single Shot MultiBox Detector (SSD)

SSD is a fast and accurate one-stage object detector that combines the ideas of anchor boxes and multi-scale feature maps to detect objects of various sizes and aspect ratios [7]. The architecture consists of a base CNN (e.g., VGG or MobileNet) followed by a series of convolutional layers with decreasing spatial dimensions. The model predicts object classes and bounding box offsets for a fixed set of anchor boxes at different feature map levels, capturing objects at different scales and aspect ratios.

During training, SSD uses a matching strategy to assign ground truth boxes to anchor boxes based on IOU scores. The model is then trained using a combined loss function that includes both classification loss (e.g., cross-entropy) and localization loss (e.g., Smooth L1 loss). SSD achieves near real-time detection speeds with competitive accuracy compared to two-stage detectors.

5.3.2.2 You Only Look Once (YOLO)

YOLO is another one-stage object detector that is designed for real-time detection with high accuracy [25]. YOLO divides the input image into a fixed-size grid (e.g., 13x13 or 19x19) and predicts object classes, bounding box coordinates, and objectness scores for each grid cell.

The architecture of YOLO consists of a series of convolutional layers, followed by fully connected layers that predict the final output tensor. Each grid cell is responsible for predicting a fixed number of bounding boxes, and each bounding box has an associated objectness score and class probabilities. During training, YOLO uses a combined loss function that takes into account classification, localization, and objectness prediction errors.

YOLO is known for its speed and ability to detect objects in real-time, making it suitable for applications like video analysis and autonomous driving. However, it may have lower detection accuracy for small objects or objects with similar aspect ratios compared to some two-stage detectors.

In summary, one-stage detectors like SSD and YOLO offer fast object detection by unifying the tasks of object localization and classification into a single framework. While they may not always achieve the same accuracy as some two-stage detectors, they are well-suited for real-time applications and scenarios where speed is a critical factor.

5.4 Generation Algorithm

GANs are a class of deep learning models introduced by Ian Goodfellow et al. in 2014. GANs have gained significant attention for their ability to generate realistic and high-quality images, among other applications. GANs consist of two neural networks, a generator and a discriminator, which are trained together in a competitive setting.

1. **Generator:** The generator network takes a random noise vector as input and produces a generated image. The purpose of the generator is to learn the underlying data distribution of the training images so that it can generate new, realistic images that resemble the training data.
2. **Discriminator:** The discriminator network takes an image as input (either a real image from the training dataset or a generated image from the generator) and classifies whether the input image is real or generated. The goal of the discriminator is to accurately distinguish between real and generated images.

The training process of a GAN involves alternating between training the discriminator and the generator:

1. **Train the Discriminator:** The discriminator is trained by providing it with a batch of real images from the training dataset and a batch of generated images from the generator. The goal is to minimize the classification error, correctly identifying real images as real and generated images as generated.
2. **Train the Generator:** The generator is trained by providing it with a batch of random noise vectors, which it uses to generate images. These generated images are then fed into the discriminator, which classifies them as real or generated. The generator's objective is to "fool" the discriminator by generating images that the discriminator classifies as real. In other words, the generator aims to maximize the classification error of the discriminator.

The training process continues until an equilibrium is reached, where the generator produces realistic images, and the discriminator can no longer accurately distinguish between real and generated images. This adversarial training process allows GANs to generate high-quality and diverse images that mimic the training data distribution.

GANs have been applied to a wide range of image generation tasks, including image synthesis, style transfer, image-to-image translation, inpainting, and super-resolution. Variations and improvements to the original GAN architecture have been proposed, such as DCGANs, CGANs, ProgressiveGANs, and Wasserstein GANs, to address issues like training stability, mode collapse, and image quality.

5.4.1 DCGAN

DCGANs are a variant of GANs that specifically use deep convolutional layers in both the generator and discriminator networks. DCGANs were introduced by Alec Radford, Luke Metz, and Soumith Chintala [26]. DCGANs have demonstrated significant improvements in the stability of GAN training and the quality of generated images compared to the original GAN architecture.

Key architectural changes and guidelines in DCGANs include:

1. **Use of Convolutional and Transposed Convolutional Layers:** In DCGANs, both the generator and the discriminator use convolutional layers instead of fully connected layers. The generator uses transposed convolutional layers (also known as deconvolutional or up-sampling layers) to transform the input noise vector into a generated image, while the discriminator uses convolutional layers to classify real and generated images.

2. **Batch Normalization:** To improve training stability, DCGANs incorporate batch normalization layers in both the generator and discriminator networks (except for the input layer of the generator and the output layer of the discriminator). Batch normalization helps to normalize the activations and gradients, preventing issues like vanishing or exploding gradients.

3. **ReLU and Leaky ReLU Activations:** DCGANs use the ReLU activation function in the generator's hidden layers to promote faster learning, while the Leaky Rectified Linear Unit (Leaky ReLU) activation function is used in the discriminator's hidden layers to prevent sparse gradients and dead neurons.

4. **Elimination of Fully Connected Layers:** DCGANs remove fully connected layers from both the generator and discriminator, relying solely on convolutional layers to learn spatial hierarchies and generate images. This allows for more stable training and better generated images.

5. **Strided Convolutions and Pooling Layers:** DCGANs use strided convolutions instead of pooling layers in the discriminator to down-sample the input images. Similarly, the generator uses strided transposed convolutions to up-sample the feature maps. This approach enables the networks to learn the appropriate spatial transformations, resulting in better image generation.

DCGANs have been used for various image generation tasks, including image synthesis, style transfer, and image-to-image translation. They have also been employed as a basis for more advanced GAN architectures and applications, such as text-to-image synthesis, semi-supervised learning, and unsupervised representation learning. DCGANs have contributed significantly to the development and understanding of stable and efficient GAN training.

5.4.2 ProGAN

ProGANs, or Progressive Growing of GANs, is an advanced approach to training GANs developed by NVIDIA researchers in 2017. GANs consist of two neural networks, a generator and a discriminator, that are trained together in a process of generating synthetic data samples and evaluating their quality. The generator creates fake samples, while the discriminator evaluates the generated samples and distinguishes them from real data.

ProGANs improve the training process of GANs by progressively increasing the resolution and complexity of the generated images in a step-by-step manner. The training begins with low-resolution images, and as the training progresses, the resolution is gradually increased by adding new layers to both the generator and discriminator networks. This allows the networks to learn features at different scales, starting from coarse features and moving towards finer details.

The key advantages of ProGANs include:

1. **Improved Stability:** ProGANs provide a more stable training process compared to traditional GANs, reducing the likelihood of issues such as mode collapse and vanishing gradients.
2. **Faster Convergence:** By starting with low-resolution images and gradually increasing the complexity, ProGANs can achieve faster convergence in training compared to training GANs on high-resolution images from the beginning.
3. **High-Quality Results:** ProGANs are capable of generating high-quality, high-resolution images with impressive detail and realism, outperforming many other GAN architectures.
4. **Scalability:** The progressive training approach makes it possible to generate images at varying resolutions, allowing ProGANs to be more easily adapted to different applications and datasets.

Overall, ProGANs represent a significant advancement in the field of generative modeling and have paved the way for further research and improvements in GAN architectures.

6. Implementation and Testing

6.1 Tools Used

6.1.1 Analysis and Design Tools

Lucidchart.com: Lucidchart is an online design tool that helps us to create a wide range of system design diagrams. We will use it to create our flowchart, use case diagrams, data flow diagrams, etc.

Tensorboard: Tensorboard is an analysis tool that we will use to monitor, track and visualize the metrics such as loss and accuracy of our model.

Jupyter Notebook: Jupyter Notebook is a web-based interactive computing platform that we will use to analyze the data and run different experiments for analysis purposes.

6.1.2 Implementation Tools

6.1.2.1 Backend

Python

Python is widely used for training deep learning models because of its vast number of libraries and frameworks. It also has huge community support, guides, and tutorials which makes it an ideal choice for any machine learning project to quickly train and deploy its model.

Tensorflow and PyTorch

Tensorflow and PyTorch are popular ML libraries because they are industry-standard libraries with good documentation and make it easy to experiment with models. We will be using them to train object detection models.

FastAPI

FastAPI is a python framework that enables developers to quickly and easily make web APIs using python along with other frameworks and libraries. Hence, we will be using it to create our Web API.

PostgreSQL

PostgreSQL is a structured renational database that is easy to use, fast, supports many languages, and has good community support. We will be using it to store the user data and the images uploaded by the users for their history and for training our model. It is an open source database perfectly suitable for our free to use model.

6.1.2.2 Frontend

Flutter

Flutter is an open-source framework by Google which allows faster app development, and native compilation, and can support multiple multiple platforms such as Android and iOS from a single codebase. We will use flutter to build our user side android application and a PWA.

Flutter Web

Flutter Web is a library that allows us to build good-looking progressive webapps from the same codebase of flutter in a short period of time and in a managed way. We will be using flutter web to build the webapp for our system.

6.1.2.3 Other Tools

Android Studio

Android Studio is an Integrated Development Environment (IDE) for developing mobile applications with features essential for app development like code completion, debugging tools, etc. We will be using it to develop our flutter application.

Visual Studio Code

Visual Studio Code (VSCode) is a widely popular code editor with a large array of available extensions which make it easy to write code in any programming language. We will be using VSCode for all of our coding needs.

Postman

Postman is a tool for analyzing and testing web APIs and has features such as editing the Hypertext Transfer Protocol (HTTP) request, viewing the HTTP response, choosing the request method, changing HTTP header properties, etc. We will be using it to test our Web API.

Onnx

Onnx is a tool that allows the conversion of ML models from one format to another which can be run on any platform with a variety of frameworks, tools, runtimes, and compilers without losing accuracy.

6.2 Implementation Details of Modules

6.2.1 Hardware

The training and deployment of the models for the PlantMD project required powerful hardware resources. For this purpose, high-performance GPUs such as NVIDIA Tesla T4 was used to accelerate the training process. Additionally, sufficient RAM and storage were required to handle the large dataset and model files.

6.2.2 Software and frameworks

The implementation of the PlantMD project relied on various software tools and frameworks. The primary programming language used for the project was Python, due to its extensive support for machine learning and image processing libraries. The deep learning models were implemented using popular frameworks like TensorFlow and PyTorch, which provided pre-trained models, GPU support, and various optimization techniques. Image preprocessing and data augmentation were performed using libraries like OpenCV, NumPy, and PIL.

6.2.3 Data Collection and Preprocessing

The first step in the implementation of the PlantMD project was to collect a large dataset of plant images, including healthy plants and plants with various diseases. This dataset was obtained from various sources, such as online plant databases.. The images were then preprocessed by resizing, normalization, and data augmentation to create a diverse and balanced dataset.

6.2.4 Image Classification Models

Two popular image classification models, Inception ResNetV2 and MobileNet, were used to classify the plant images into different categories based on their health and disease conditions. These models were trained on the preprocessed dataset using transfer learning, where the pre-trained weights of the models were fine-tuned on the plant dataset. This allowed the models to achieve high accuracy in classifying the plant images.

6.2.5 Grad-CAM for Model Interpretation

To provide better insights into the decision-making process of the image classification models, the Grad-CAM algorithm was implemented. Grad-CAM is a technique that visualizes the regions in the input image that contribute most significantly to the model's classification decision.

6.2.6 Object Detection Models

In addition to image classification, object detection models like Faster R-CNN, SSD MobileNet, and YOLOv7 were employed to detect and localize the diseased regions in the plant images. These models were trained on a subset of the dataset with annotated bounding boxes around the diseased regions. The object detection models were able to accurately identify and highlight the affected areas in the plant images.

6.2.7 Generative Adversarial Networks (GANs)

To generate realistic images of healthy and diseased plants, various GAN models like DCGAN and ProGAN were implemented. These models were trained on the plant dataset to generate new images, which could be used for further analysis and research. GANs were particularly useful in generating images with simple backgrounds but they weren't effective for images with complex backgrounds.

6.2.8 Hyperparameter Tuning

To achieve optimal performance of the models, hyperparameter tuning was performed. This involved experimenting with different learning rates, batch sizes, and optimization algorithms, among other parameters. Techniques like grid search, and random search were employed to find the best hyperparameter settings for each model.

6.2.9 Model Evaluation and Selection

The performance of the different image classification and object detection models was evaluated using various metrics like accuracy, precision, recall, and F1 score. Based on these metrics, the best-performing models were selected for deployment in the PlantMD application.

6.2.10 Model Deployment

Once the models were trained and evaluated, they were deployed on a cloud-based server using platform called Linode.

6.2.11 User Interface and Experience (UI/UX) Design

To make the PlantMD application more user-friendly and accessible, we focused on designing an intuitive and visually appealing user interface. This includes easy-to-use navigation, clear instructions, and informative visualizations of the model's predictions. The user can just open the application, tap on the capture photo button and can get the prediction results.

6.2.12 Application Development and Deployment

The selected models were integrated into a Web API that can be accesed through a user-friendly application, which could be accessed by farmers, researchers, and other stakeholders. This application allows users to upload plant images and receive information about the health and disease conditions of the plants. The application also provided suggestions on how to treat the identified diseases and maintain the overall health of the plants.

6.3 Testing

6.3.1 Test 1: Upload an image

Table 6.1: Upload test.

Test Case	Upload
Expected result	User should be able to upload an image.
Actual result	User uploaded image successfully
Are you satisfied?	Yes, test success

Figure 6.1: Upload Test

6.3.2 Test 2: Disease prediction

Table 6.2: Test case of Disease Prediction

Test Case	Disease Prediction
Expected result	System should be able to predict disease
Actual result	Disease predicted successfully
Are you satisfied?	Yes, test success

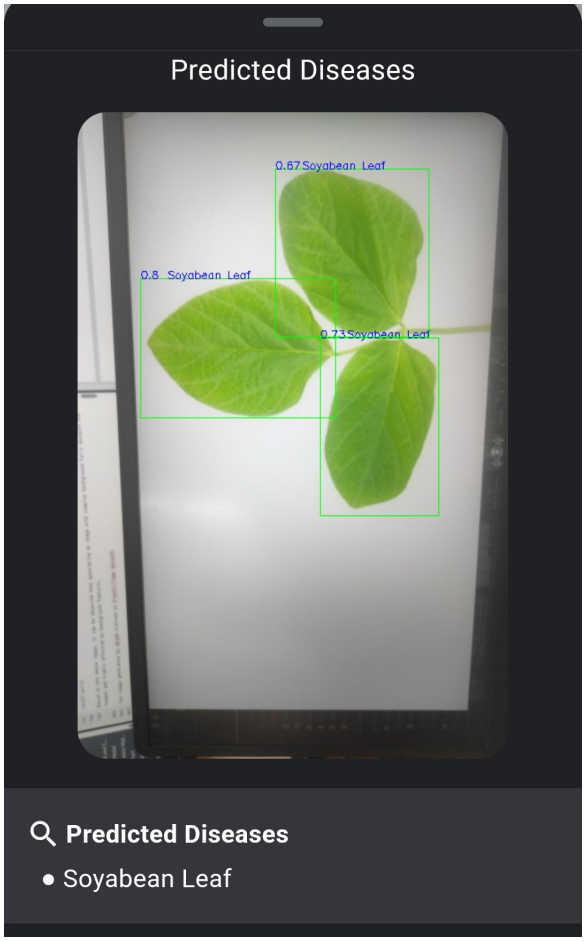


Figure 6.2: Disease Prediction Test

6.3.3 Test 3: Suggest Remedies

Table 6.3: Test case for suggesting remedies

Test Case	Suggest Renedies
Expected result	System should be able to provide remedies
Actual result	Suggested successfully
Are you satisfied?	Yes, test success

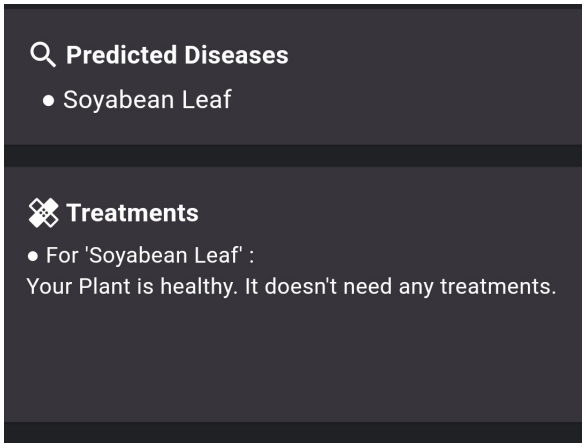


Figure 6.3: Suggest Remedies Test

7. Result Analysis

7.1 Analyzing Heatmap

CNN were used to classify the different diseases that are present in the plant leaves.The CNN model can classify 27 different diseases. The heatmap of confusion matrix is used to evaluate the performance of the given system.

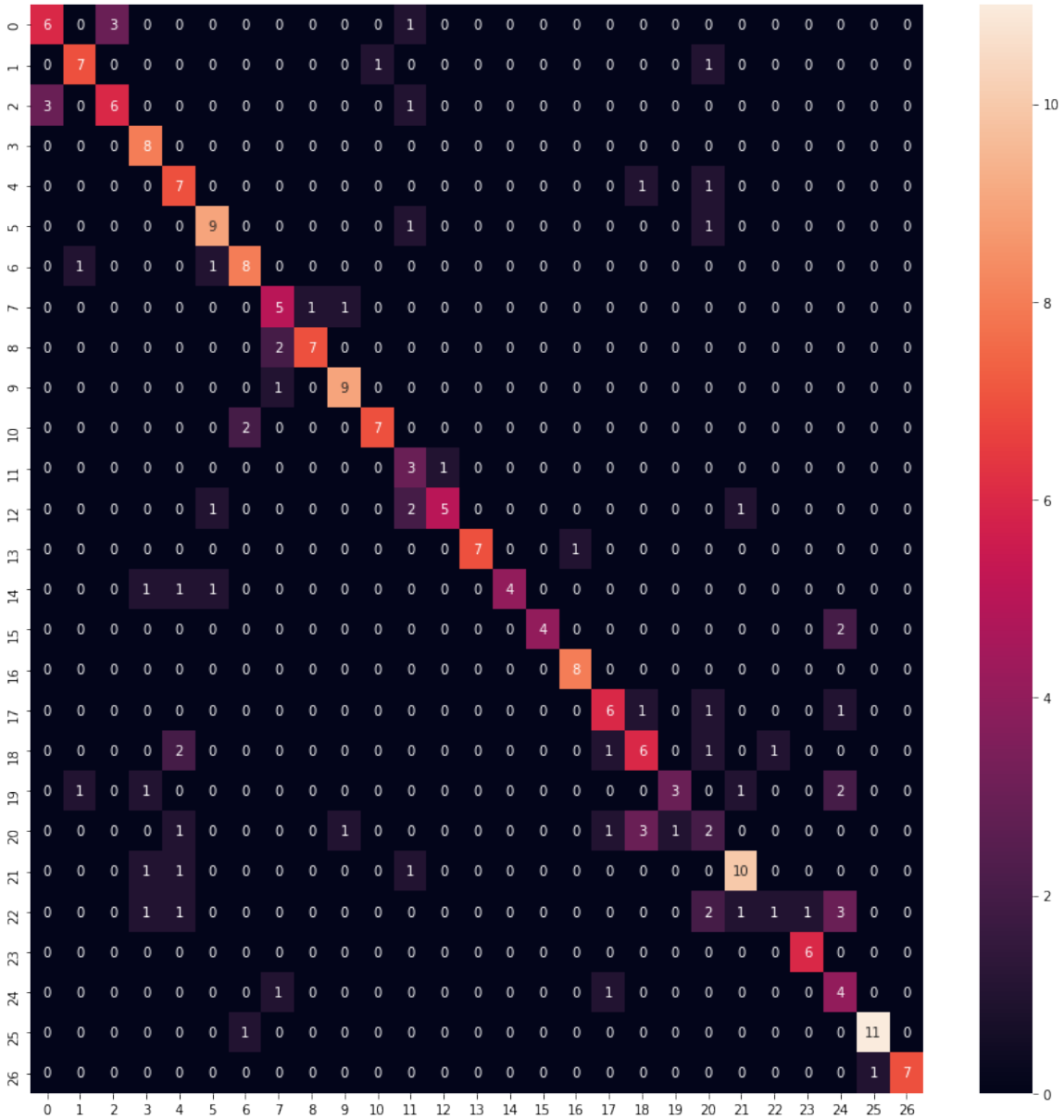


Figure 7.1: Confusion Matrix

Analyzing the heatmap from Figure 7.1, we can see the wrong predictions are from the same category i.e Apple, Corn, Potato and Tomato categories. The images from these classes are very hard to distinguish because they look visually similar. Some of the images are shown in Figure 7.2, 7.3, 7.4 and 7.5.

(a) Tomato Early Blight

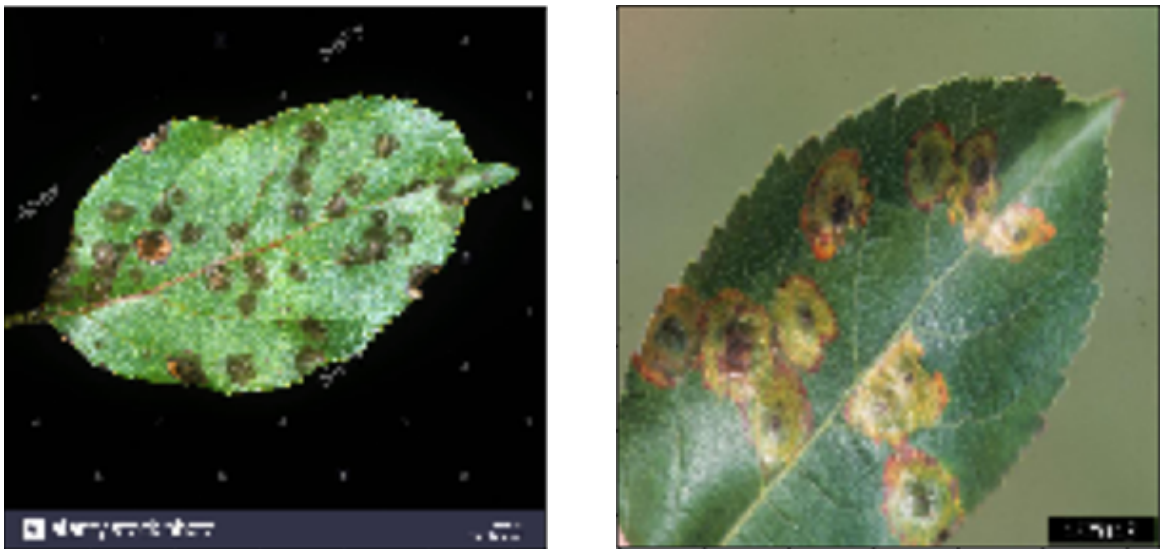
(b) Tomato Late Blight

Figure 7.2: Featured Discriminator for Tomato Leaves

(a) Corn Leaf Blight

(b) Corn Gray Leaf Spot

Figure 7.3: Featured Discriminator for Corn Leaves



(a) Apple Scab Leaf **(b)** Apple Rust Leaf

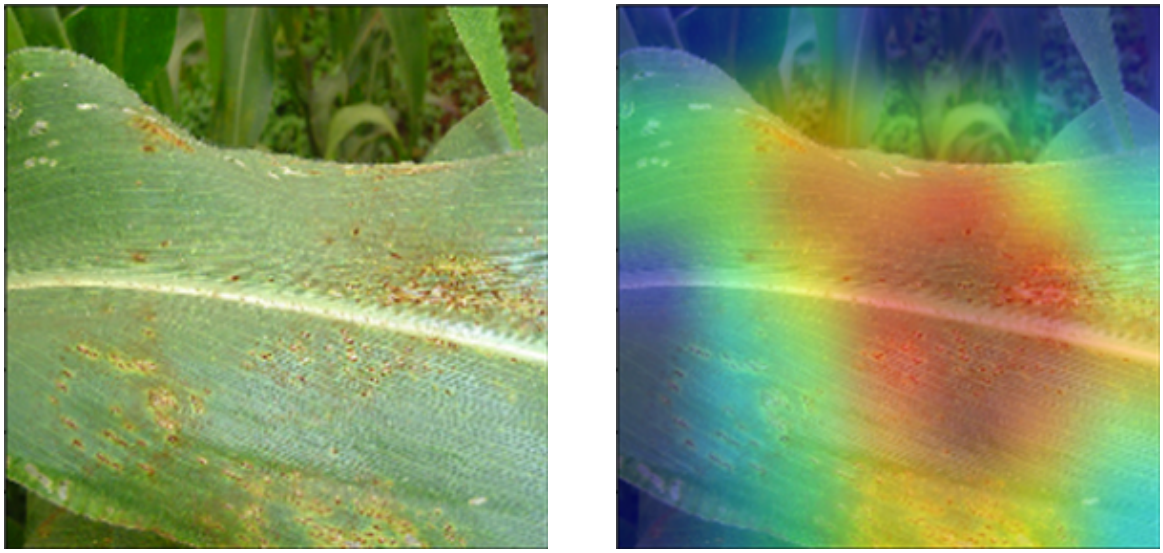
Figure 7.4: Featured Discriminator for Apple leaves

(a) Potato Early Blight **(b)** Potato Late Blight

Figure 7.5: Featured Discriminator for Potato Leaves

7.2 Interpreting the model prediction with Grad-CAM

Figures 7.6, 7.7, 7.8 and 7.9 are some of the results obtained by Grad-CAM.

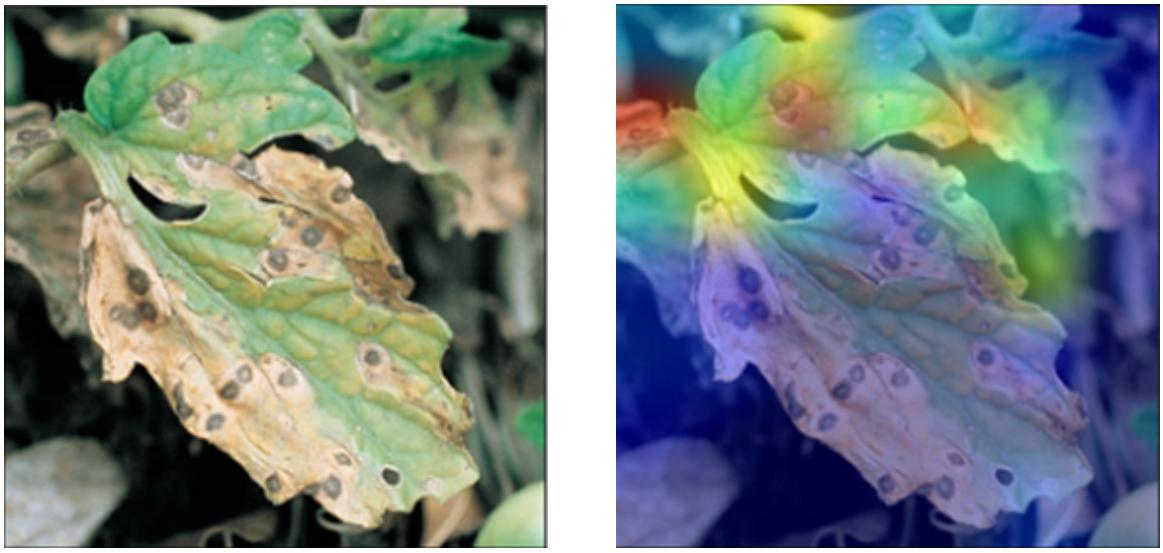


(a) Corn Rust Leaf **(b) Prediction:** Corn Rust Leaf

Figure 7.6: Grad-CAM Visualization for Corn Rust Leaf

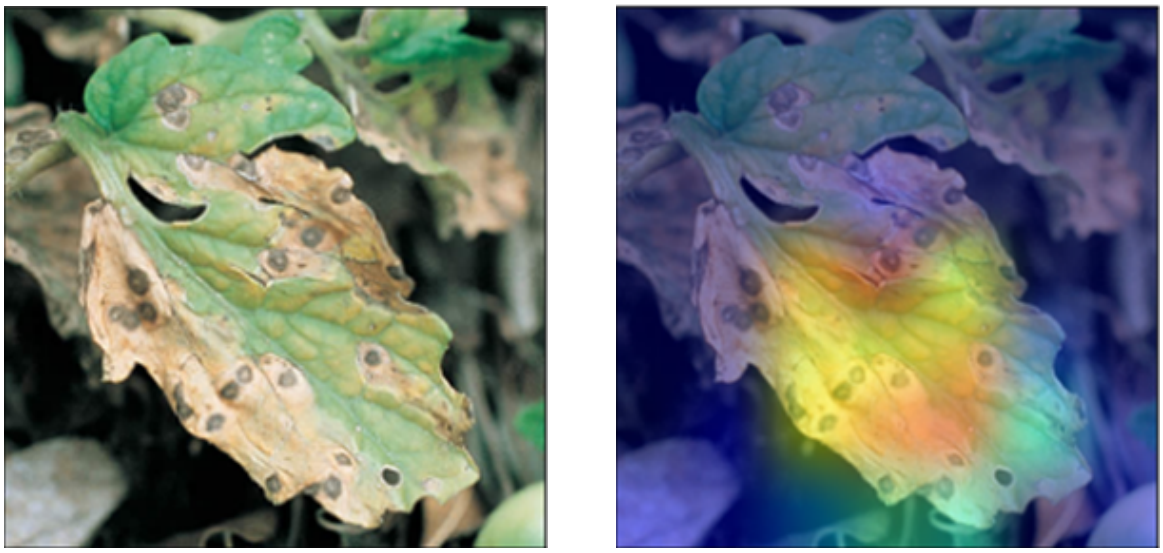
(a) Potato Early Blight **(b) Prediction:** Potato Early Blight

Figure 7.7: Grad-CAM Visualization for Potato Early Blight



(a) Tomato Early Blight **(b) Prediction:** Tomato Early Blight

Figure 7.8: Grad-CAM Visualization for Tomato Early Blight

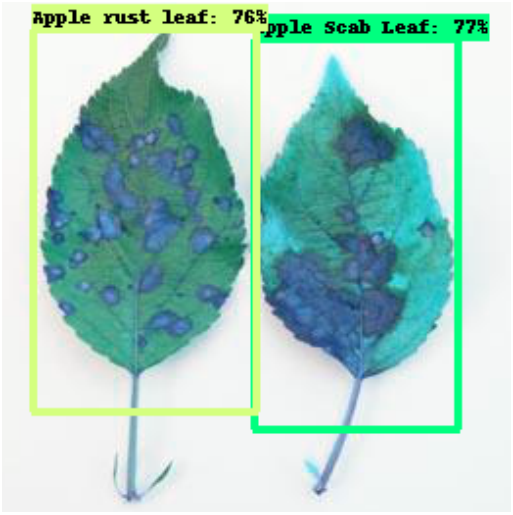


(a) Tomato Early Blight **(b) Prediction:** Tomato Septoria Blight

Figure 7.9: Grad-CAM Visualization for Tomato Early Blight

From figures 7.6, 7.7, 7.8 and 7.9, we can see the model is learning to focus on the infected parts of the leaves to make predictions. The predictions in Figures 7.8 and 7.9 are from the same class i.e Tomato Early Blight. In Figure 7.8b and Figure 7.9b we made our model predict them as Tomato Early Blight and Tomato Septoria Gray Spot respectively. The visualizations are quite interesting. It focuses on the infected upper part to make its prediction as Tomato Early Blight and it focuses on the infected lower part to make its prediction as Tomato Septoria Gray Spot.

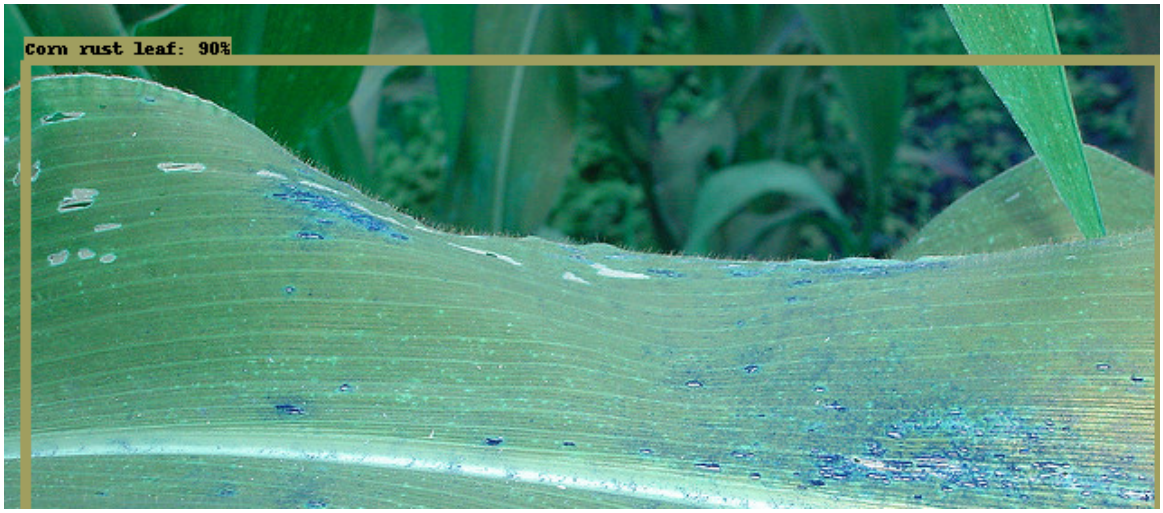
7.3 Plant Disease Detection model results



(a) Apple Rust (76%) and Scab Leaf (77%)

(b) Tomato Mold Leaf (64%)

(c) Tomato Early Blight Leaf (72%)



(d) Corn Rust Leaf (90%)

Figure 7.10: Results produced by the Plant Disease Detection model

7.4 Image generation using GANs

7.4.1 Images generated by DCGAN trained on PlantDoc dataset

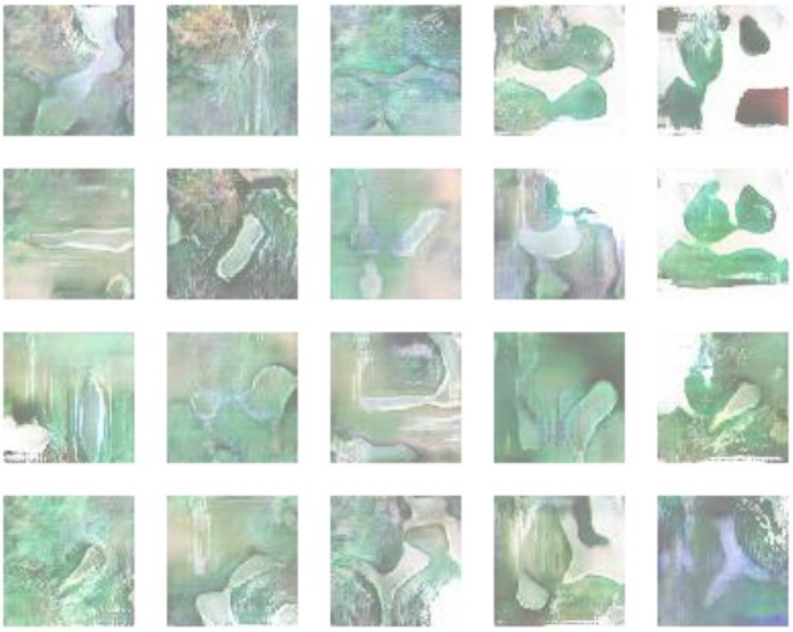


Figure 7.11: Synthetic Image generated by DCGAN trained on PlantDoc dataset

Based on the Figure 7.11, it can be observed that generating an image with complex background fails because the images get highly affected by background features.

7.4.2 Images generated by DCGAN trained on PlantVillage dataset

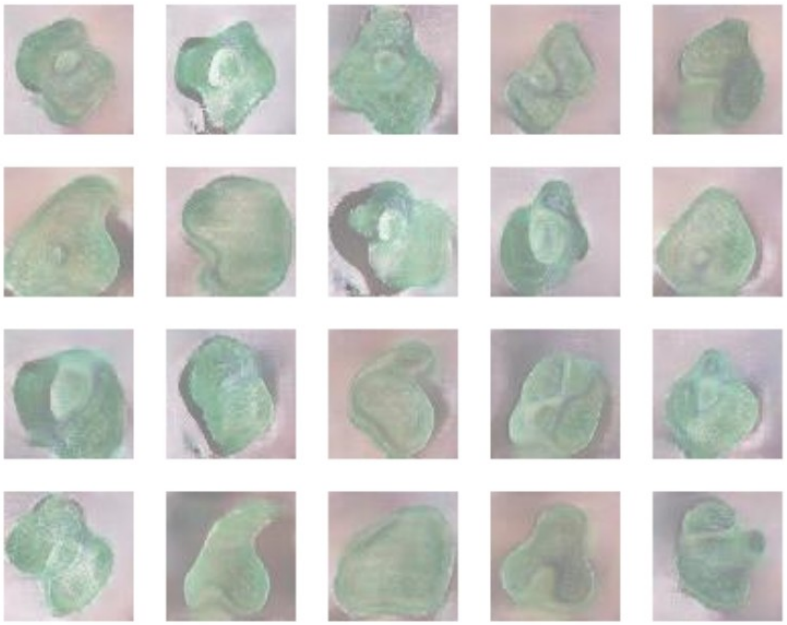


Figure 7.12: Synthetic image generated by DCGAN trained on PlantVillage dataset

Based on Figure 7.12, it can be observed that the generated images are much more appealing than the images from the PlantDoc dataset. The images are trying to capture the shape of the leaf. The DCGAN model is only trained for 2500 epochs because training the DCGAN WGAN_GP model takes a lot of resources and time. Training on PlantVillage Dataset for a longer duration can produce images that look similar to original images. Also these image are trained at resolution of 64 x 64 pixels and training DCGAN to produce a higher resolution image can be hard because of instability.

7.4.3 Images generated by ProGANs

ProGAN were trained on both the PlantVillage and PlantDoc dataset.

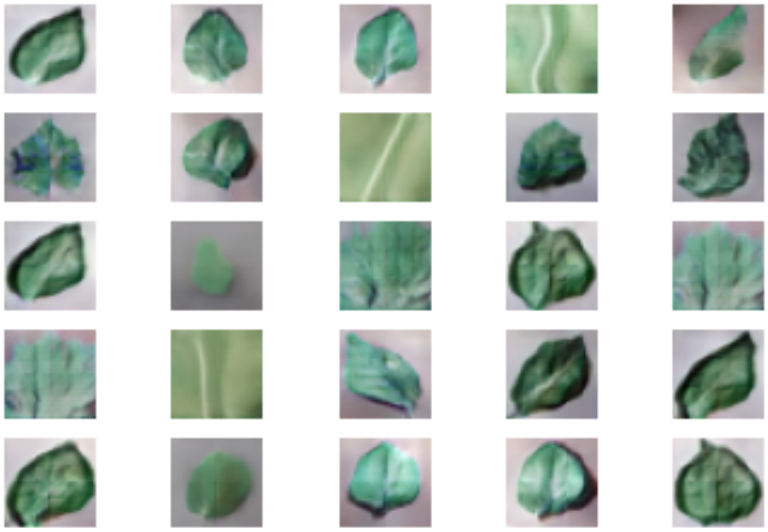


Figure 7.13: Synthetic image generated by ProGAN of resolution 32x32

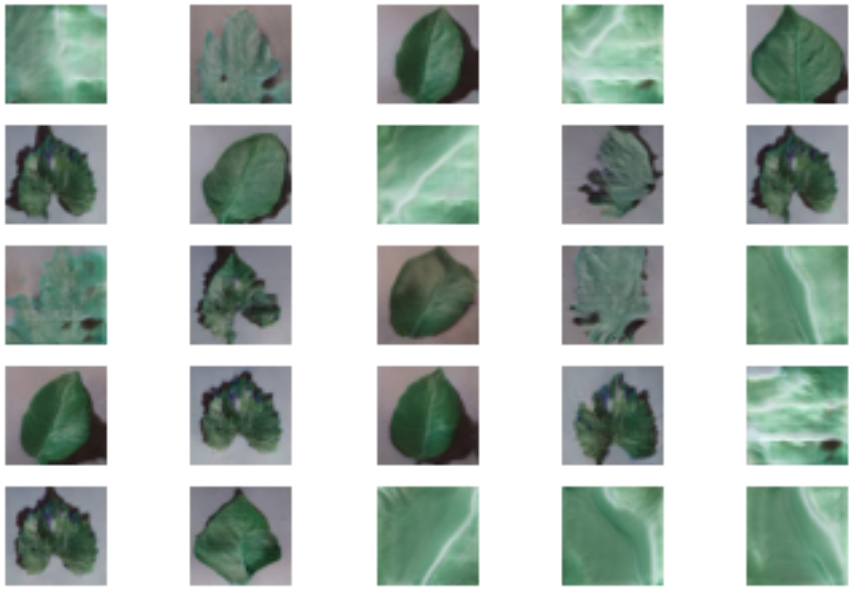


Figure 7.14: Synthetic image generated by ProGAN of resolution 64x64

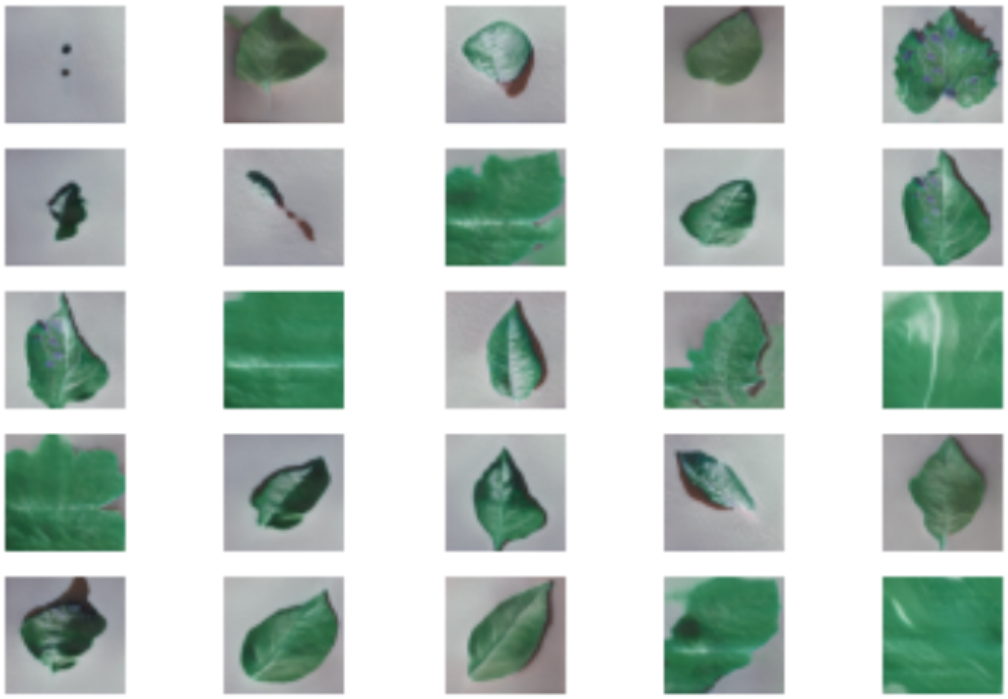


Figure 7.15: Synthetic image generated by ProGAN of resolution 128x128

The generated images shown in Figure 7.13, 7.14 and 7.15 are of resolution 32 x 32, 64 x 64 and 128 x 128 respectively. Based on the above image, it can be observed that the generated image poses quality like shape, color, and texture as of real images. But, training the same ProGAN architecture on the PlantDoc dataset produces images that are very much indistinguishable because of complex background features. So it is difficult to generate images having complex backgrounds or images which are taken from real-life conditions. The images generated by ProGAN on PlantVillage look similar to original images which can potentially be used as training samples. Due to resource constraints, we couldn't generate an image of size 256X256. So we only generated images up to size 128X128.

8. Conclusion

In conclusion, the PlantMD project aimed to create an effective plant disease prediction system by utilizing various machine learning techniques. Although our initial attempt with image classification did not yield the desired performance, the subsequent implementation of an object detection model proved to be highly successful in predicting diseases in leaves accurately.

Moreover, we explored the potential of GANs in generating synthetic images to supplement our training data. While the GAN model demonstrated promising results in generating images with simple backgrounds, it faced limitations in handling complex backgrounds. This suggests that further research and development in GANs could potentially improve the model's performance and capabilities.

The final phase of our project involved deploying the trained model via a web API, allowing seamless interaction with a mobile application to serve users efficiently. Overall, the PlantMD project represents a significant step forward in plant disease prediction and management, with the potential for further improvements and applications in the field of agriculture and plant health.

References

[1] *FAO - News Article: Climate change fans spread of pests and threatens plants and crops, new FAO study*, [Online; accessed 18. Dec. 2022], Dec. 2022. [Online]. Available: https://www.fao.org/news/story/en/item/1402920/icode.

[2] *Plant Protection Directorate 2013*, [Online; accessed 18. Dec. 2022], Dec. 2022. [Online]. Available: http://www.npponepal.gov.np/.

[3] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[4] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[5] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *Thirty-first AAAI conference on artificial intelligence*, 2017.

[6] A. G. Howard, M. Zhu, B. Chen, *et al.*, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[7] W. Liu, D. Anguelov, D. Erhan, *et al.*, "Ssd: Single shot multibox detector," in *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14*, Springer, 2016, pp. 21–37.

[8] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao, "Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors," *arXiv preprint arXiv:2207.02696*, 2022.

[9] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *Advances in neural information processing systems*, vol. 28, 2015.

[10] R. I. Hasan, S. M. Yusuf, and L. Alzubaidi, "Review of the state of the art of deep learning for plant diseases: A broad analysis and discussion," *Plants*, vol. 9, no. 10, p. 1302, 2020.

[11] D. Singh, N. Jain, P. Jain, P. Kayal, S. Kumawat, and N. Batra, "Plantdoc: A dataset for visual plant disease detection," in *Proceedings of the 7th ACM IKDD CoDS and 25th COMAD*, ser. CoDS COMAD 2020, Hyderabad, India: Association for Computing Machinery, 2020, pp. 249–253, ISBN: 9781450377386. DOI: 10.1145/3371158.3371196. [Online]. Available: https://doi.org/10.1145/3371158.3371196.

[12] *PlantVillage Dataset*, [Online; accessed 19. Dec. 2022], Dec. 2022. [Online]. Available: https://www.kaggle.com/datasets/emmarex/plantdisease.

[13] S. Phadikar, J. Sil, and A. K. Das, "Rice diseases classification using feature selection and rule generation techniques," *Computers and electronics in agriculture*, vol. 90, pp. 76–85, 2013.

[14] B. Sandika, S. Avil, S. Sanat, and P. Srinivasu, "Random forest based classification of diseases in grapes from images captured in uncontrolled environments," in *2016 IEEE 13th international conference on signal processing (ICSP)*, IEEE, 2016, pp. 1775–1780.

[15] S. P. Mohanty, D. P. Hughes, and M. Salathé, "Using deep learning for image-based plant disease detection," *Frontiers in plant science*, vol. 7, p. 1419, 2016.

[16] A. Ramcharan, K. Baranowski, P. McCloskey, B. Ahmed, J. Legg, and D. P. Hughes, "Deep learning for image-based cassava disease detection," *Frontiers in plant science*, vol. 8, p. 1852, 2017.

[17] A. Fuentes, S. Yoon, S. C. Kim, and D. S. Park, "A robust deep-learning-based detector for real-time tomato plant diseases and pests recognition," *Sensors*, vol. 17, no. 9, p. 2022, 2017.

[18] J. Sun, Y. Yang, X. He, and X. Wu, "Northern maize leaf blight detection under complex field environment based on deep learning," *IEEE Access*, vol. 8, pp. 33 679–33 688, 2020.

[19] P. V. Bhatt, S. Sarangi, and S. Pappula, "Detection of diseases and pests on images captured in uncontrolled conditions from tea plantations," in *Autonomous Air and Ground Sensing Systems for Agricultural Optimization and Phenotyping IV*, SPIE, vol. 11008, 2019, pp. 73–82.

[20] Z. Chen, Y. Qian, Y. Wang, and Y. Fang, "Deep convolutional generative adversarial network-based emg data enhancement for hand motion classification," *Frontiers in Bioengineering and Biotechnology*, vol. 10, 2022.

[21] N. Alajlan, M. Alyahya, N. Alghasham, and D. M. Ibrahim, "Data enhancement for date fruit classification using dcgan," 2021.

[22] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 580–587.

[23] R. Girshick, "Fast r-cnn," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1440–1448.

[24] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *Advances in neural information processing systems*, vol. 28, 2015.

[25] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[26] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," *arXiv preprint arXiv:1511.06434*, 2015.

Appendix A

Gantt Chart

1. Gantt Chart of the project

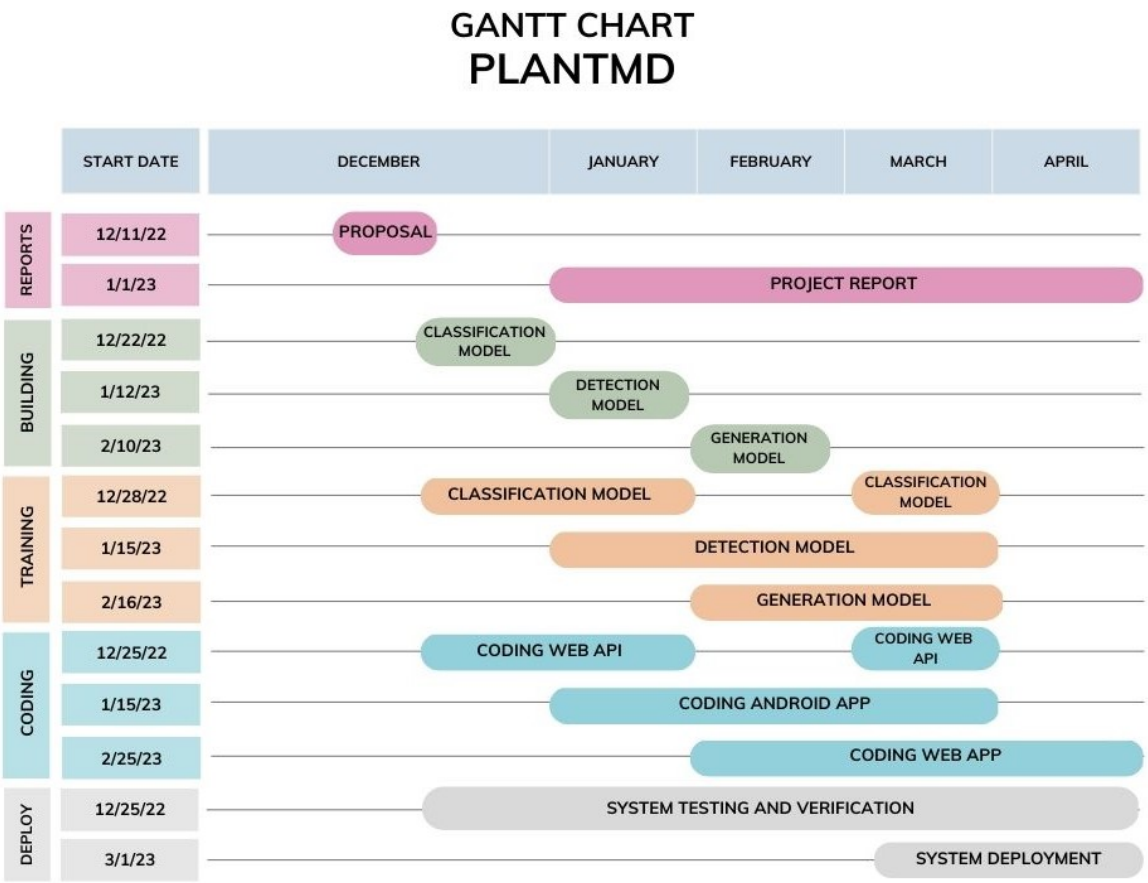


Figure A.1: Gantt Chart of the project.

Appendix B

Wireframes and Screenshots of Mobile App

1. Home Screen

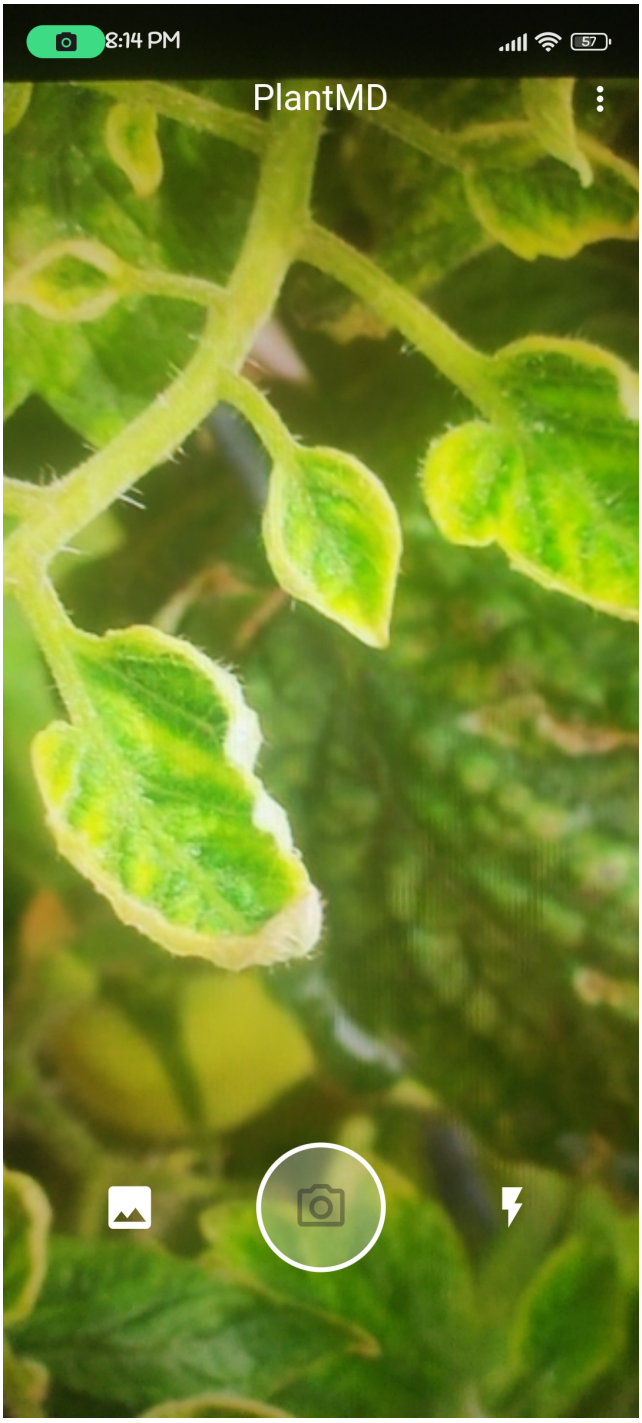


Figure B.1: Home screen of the mobile app

2. Select Image From Storage

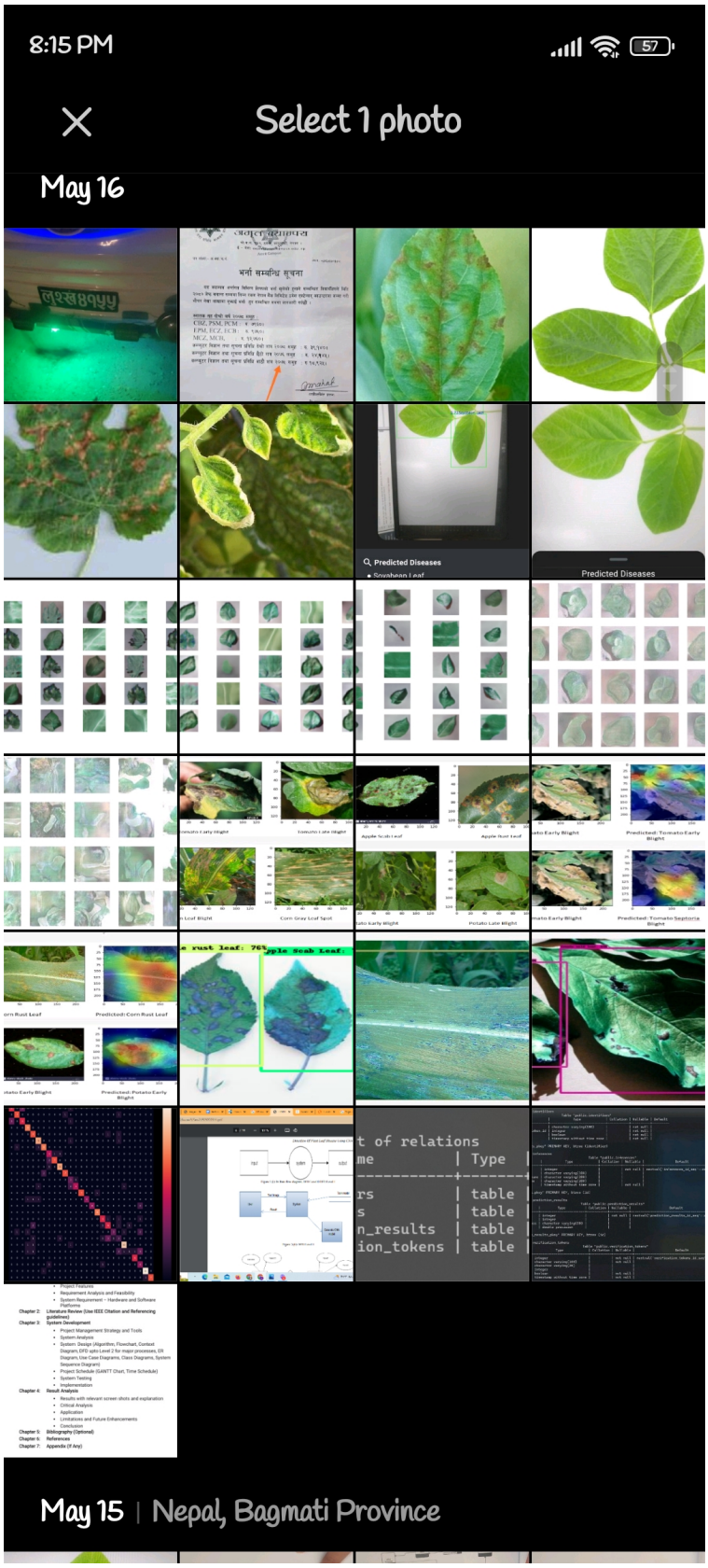


Figure B.2: Select image from storage options.

3. Scanning Plant UI

Figure B.3: Loading screen when photo is captured or selected.

4. Result Popup UI

Figure B.4: Small fragment of results that pops up after scanning is complete.

5. Result Screen UI

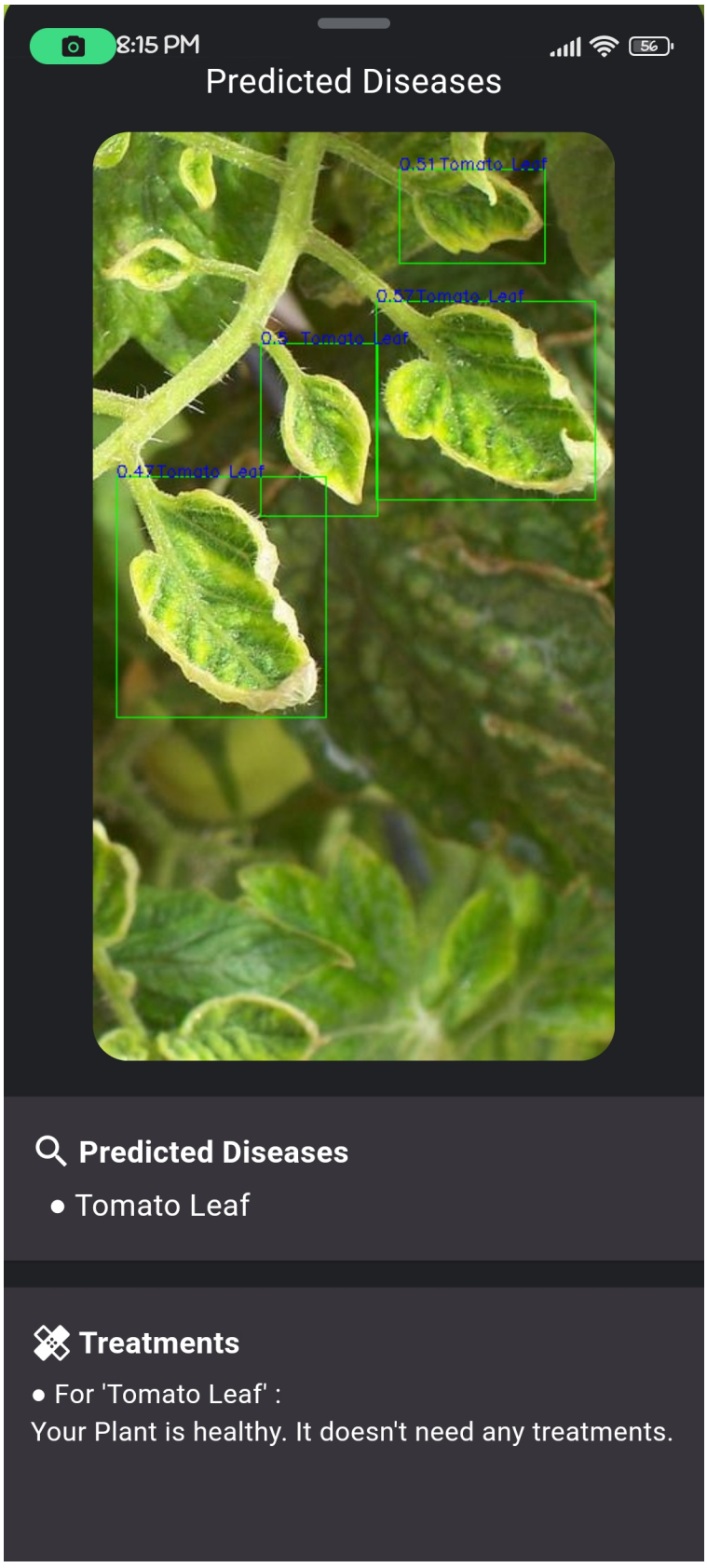


Figure B.5: Result screen after the fragment is pulled up.

6. Result Screen of Diseases Plant

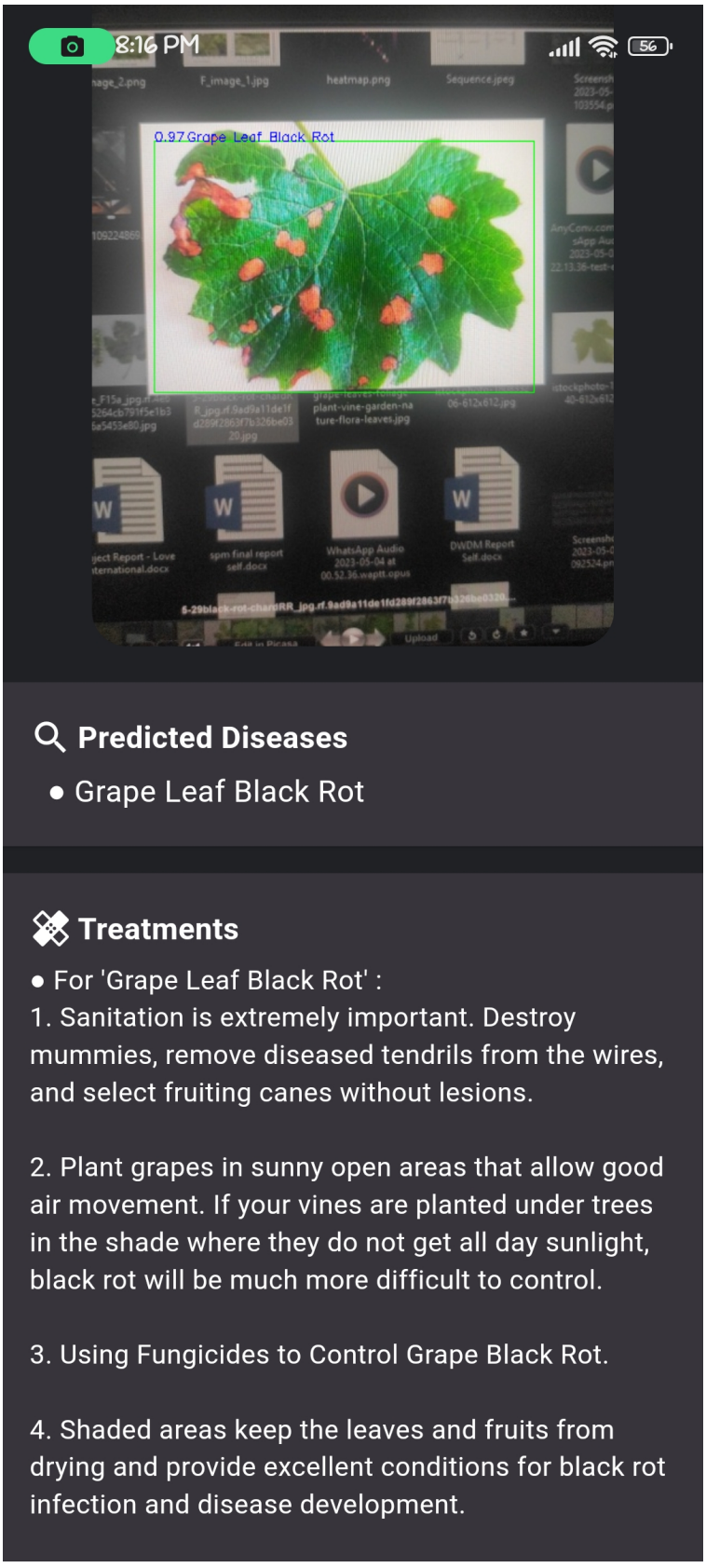


Figure B.6: Result Screen with detected disease and remedies.

Appendix C

Using Postman to Interact with Web API

1. Hitting the \ping endpoint of the Web API without auth token

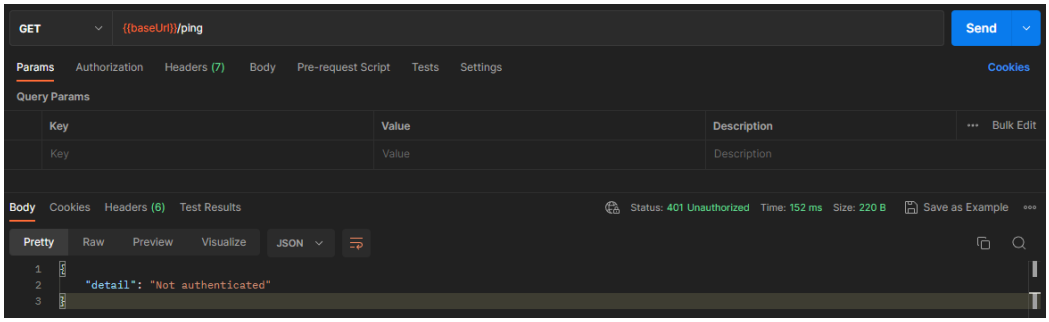


Figure C.1: Hitting \ping endpoint without auth token.

2. Posting credentials to the \token endpoint of the Web API

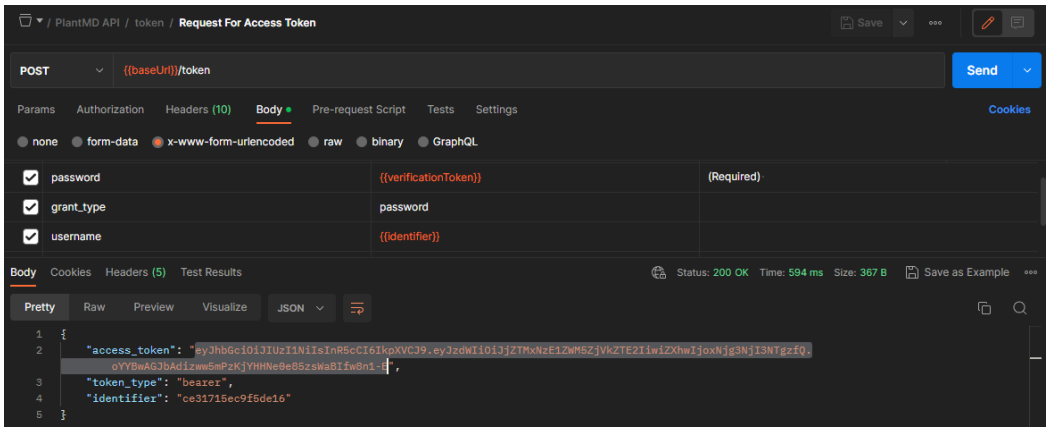


Figure C.2: Posting \token endpoint with credentials to obtain an auth token.

3. Hitting the \ping endpoint of the Web API with auth token

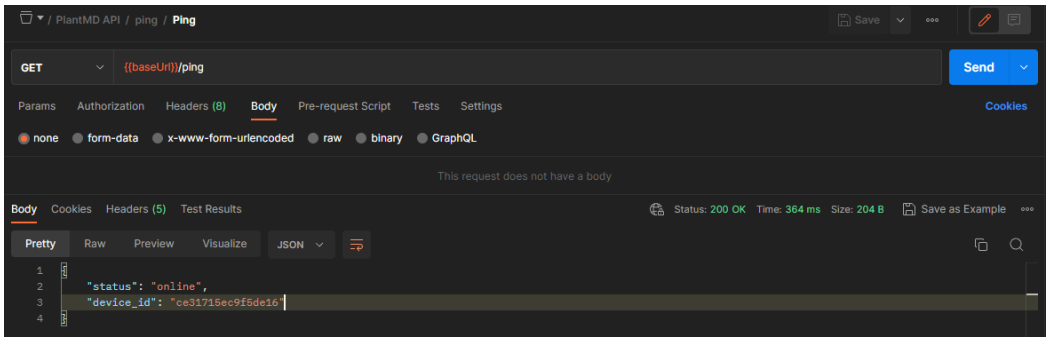


Figure C.3: Hitting \ping endpoint with auth token.

4. Posting image to the \inferences endpoint of the Web API

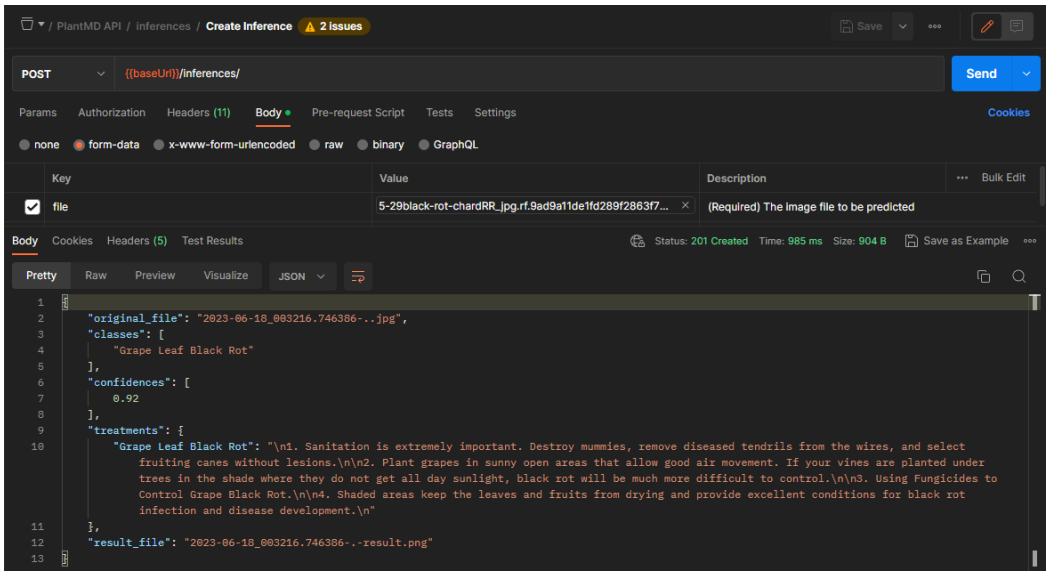


Figure C.4: Posting \inferences endpoint with image to predict.

5. Hitting the \image endpoint of the Web API with image name

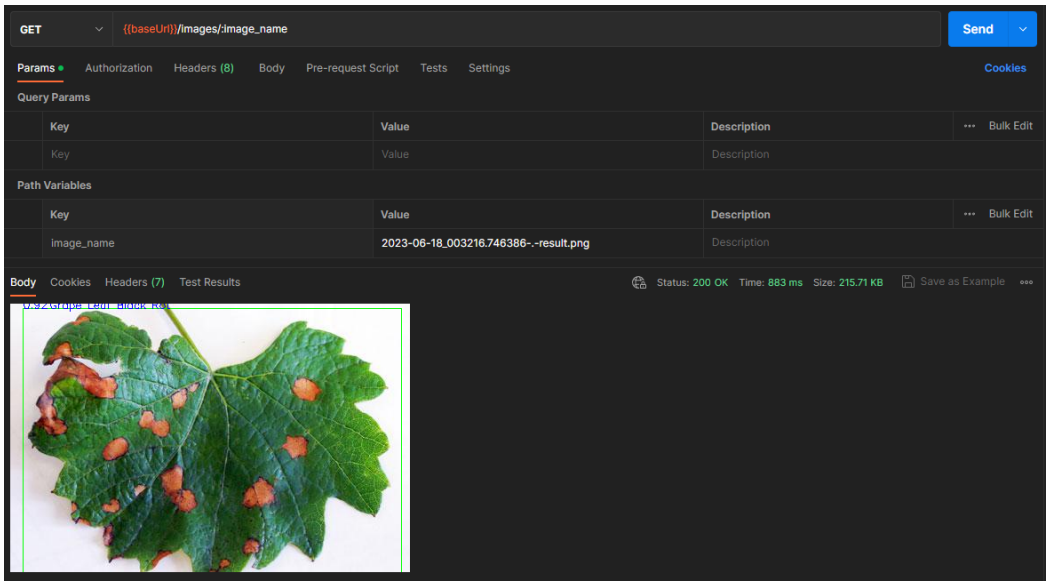
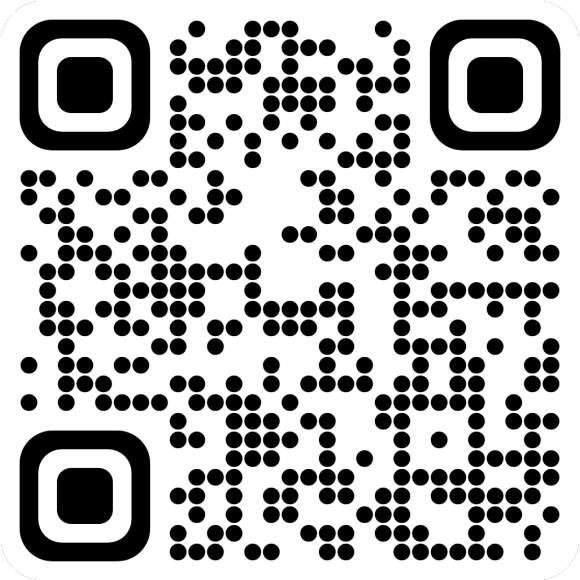


Figure C.5: Hitting \image endpoint with image name.

Appendix D

Source Code and Repository

The soft copy of this report and the source code to all the artifacts of this project can be found in the anuraglimbu/PlantMD github repository.



Repo Link: https://github.com/anuraglimbu/PlantMD