Plant Doctor

Group Members

Tayyab Shabbir (01-131182-036)
Usama Khalid (01-131182-037)

Supervisor: Dr. Adeel M. Syed

A Final Year Project submitted to the Department of Software Engineering, Faculty of Engineering Sciences, Bahria University, Islamabad in the partial fulfilment for the award of degree in Bachelor of Software Engineering

July 2022

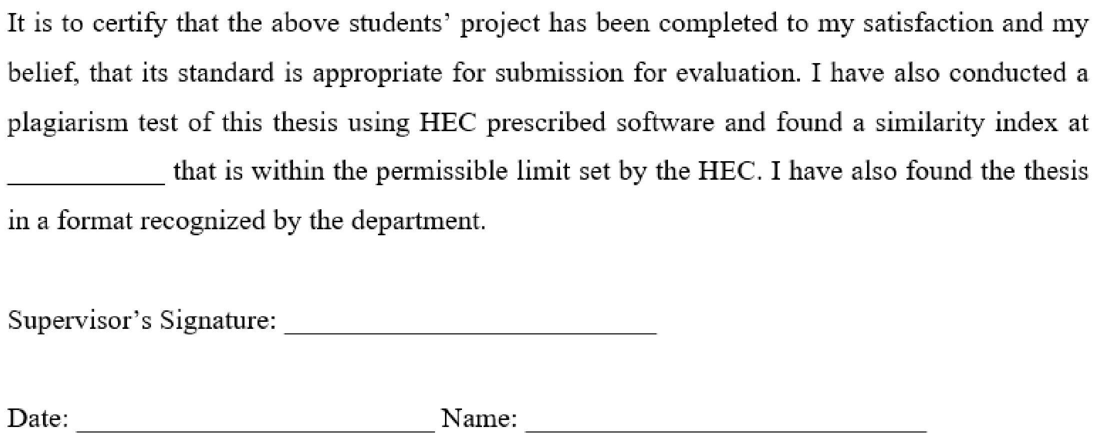
THESIS COMPLETION CERTIFICATE

Student Name: Tayyab Shabbir Enrolment No: 01-131182-036

Student Name: Usama Khalid Enrolment No: 01-131182-037

Programme of Study: Bachelor of Software Engineering

Project Title: PlantDoctor



It is to certify that the above students' project has been completed to my satisfaction and my belief, that its standard is appropriate for submission for evaluation. I have also conducted a plagiarism test of this thesis using HEC prescribed software and found a similarity index at ___________ that is within the permissible limit set by the HEC. I have also found the thesis in a format recognized by the department.

Supervisor's Signature: ___________________________

Date: __________________________ Name: _____________________________

CERTIFICATE OF ORIGINALITY

This is certify that the intellectual contents of the project ____ ____ are the product of my/our own work except, as cited properly and accurately in the acknowledgements and references, the material taken from such sources as research journals, books, internet, etc. solely to support, elaborate, compare, extend and/or implement the earlier work. Further, this work has not been submitted by me/us previously for any degree, nor it shall be submitted by me/us in the future for obtaining any degree from this University, or any other university or institution. The incorrectness of this information, if proved at any stage, shall authorities the University to cancel my/our degree.

Name of the Student: ____________________________

Signature: ________________________ Date: __________________

Name of the Student: ____________________________

Signature: ________________________ Date: __________________

Abstract

Agriculture is the foundation of all civilizations and cultures. According to the Food and Agricultural Organization (FAO), agriculture is essential to the global economy because it provides food for more than half of the world's population (62 percent). But because of crop losses, more than 40% of cultivated plants become unusable annually. In Pakistan, agriculture is the primary source of income for most of the population. Plant diseases are difficult to detect with the naked eye most of the time. PlantDoctor is a cross-platform mobile application that uses deep learning to identify plant diseases by analysing plant leaves.

PlantDoctor is a cross-platform mobile application which consist of 3 main components that a cross-platform application, a model which is trained by deep learning for detecting the disease, and a server-side application that works as the API gateway for the whole application. The report consists of the research part by observing and distinguishing existing similar systems and case studies. Deep learning has been used as an automatic crop disease detection. It was proposed to use Convolutional Neural Network as the deep learning algorithm but the results we got were not good enough. So, we had to move on to Transformers for tetter results.

By using Transformers, the deep learning model has achieved 98.75% accuracy and it was performing well on real-time data as well as compared to CNN other algorithms like using Visual Geometry Group (VGG) and Residual Neural Network (ResNet). It has completed the perspectives used to construct this proposed system, design stage, usage and capacities, and the most critical basic assessment within this application.

Keywords: Crop Disease Detection, Plant Disease Detection, Plant Diseases, Plant Disease Detection using CNN

Dedication

To my parents for their love and support

Acknowledgments

In today's competitive world, there is a race for survival among those who have the determination to succeed. A project serves as a link between the theoretical and practical worlds of work. We began this project with that eagerness. We would like to express our heartfelt gratitude to our loving family members for raising us with love and encouragement during the project's duration. We feel obligated to take this opportunity to express our heartfelt gratitude to our supervisor Dr. Adeel M. Syed for his unwavering support and thoughtfulness throughout the project's duration. Furthermore, we appreciate his assistance, advice, and the constant guidance in ensuring that we stay on track with the project timeline.

We believe that now is the best time to express our heartfelt gratitude to all the lecturers who have helped us grow into the people we are today throughout our academic careers at the university. Finally, we would like to express our gratitude to all our colleagues especially Arslan who assisted us whenever we needed it, as well as all of the Software Engineering department employees for their generous attitudes and friendly support. We do not have the words to express our heartfelt gratitude, but my heart is still overflowing with the kindnesses we have received from everyone.

Table of Contents

List of Figures

List of Tables

Chapter 1
Introduction

This section includes the objectives, problem statement and motivation which describes the starting point to build PlantDoctor from scratch.

1.1. Motivation

All civilizations are built on the foundation of agriculture. Agricultural productivity has already had a significant impact on the industrialized economic development of countries, and its role in developing countries is critical. Plant diseases pose a severe threat to the agriculture industry and have the potential to starve the entire human population if not detected early. Plant disease detection will become easier and cheaper with the deployment of machine learning models in the domain of plant pathology. It will assist many farmers in the prompt diagnosis of plant diseases, preventing plant waste, and preventing disease transmission from diseased to healthy plants. Also nowadays, several people are gardening as a hobby, but they do not have a substantial knowledge of plants themselves. Because of this many of them are unable to timely realize the health condition of their plants and end up losing them. This effort of machine learning models will help people who have minimum knowledge about plant diseases, to stay aware of their plant's health through a quick and easy way. We believe that this will help them be better at gardening and encourage more people to do it, and if a significant number of people adopt this it may lead to a considerable boost in our food production. This can be utilized by more professional farmers as well, like if they are planting a new crop about which they do not have any prior experience or knowledge.

1.2. Problem statement

According to the Food and Agricultural Organization (FAO), agriculture is essential to the global economy since it provides food for more than half of the world's population (62 percent). Grain losses ranging from 10% to 20%, rice losses ranging from 25% to 41%, maize losses of 20%, potato losses ranging from 8% to 21%, soya losses ranging from 11% to 32%, and so on have been discovered internationally [1] due to pests and mainly due to crop and plant diseases, which make a tremendous impact on global economy.

Plant disease detection is so crucial in agriculture. Plant diseases are unavoidable if proper safeguards are not followed in this field, and can have devastating consequences on plants, affecting their quality, abundance, and productivity. Disease management rules are inadequate in developing countries, and annual losses of 30 percent to 50 percent are usual for the country's key crops. The basic way for detecting plant diseases is necked eye observation by professionals. Nonetheless, this necessitates ongoing expert supervision, which can be prohibitively expensive in large estates. By properly resolving the above issue, we will be able to reliably evaluate the influence of pests and plant diseases on crop yield, which will be a huge step forward.

1.3. Objectives

The major goal is to provide a reliable and self-driven plant disease detection system which will run on android/iOS device that can identify and recognise a wide range of plant diseases. The application should be able to diagnose plant disease by looking at the state of the diseased plant leaf. The initial step of the method would be used on a smaller group of plants, such as grapes, potatoes, apples, potatoes, and so on.

To this end the main objectives of thesis are:

1. *To identify crop or plant diseases from their leaf using deep learning.*
2. *Helping hobbyists who are worried about their plants and their health.*

1.4. Main contributions

- User can identify a disease with a single click.
- Identifying a disease using deep learning with 98% accuracy.
- Transformers are used for the first time for plant disease detection.
- Server-side processing let the app to be used on low end devices.

Chapter 2
Background Study/Literature Review

Plant diseases have been a danger to the farmers since the dawn of agriculture. Crops are vulnerable to several different kinds of pathogens like fungi, bacteria, viruses etc. The diseases can affect almost any part of a plant, but the effect is usually most prominent on the leaves. These diseases damage 10% to 16% of plants worldwide every year. By detecting and classifying these diseases contingency measures can be set up for this issue.

Historically speaking primary mode for scrutinizing diseases in plants have been use of ones very own trusty eyes and brains for people since the advent of agriculture. Though with advancement of technology, it has reached a point where it is capable of being deployed to make observations with high accuracy and provide a reliable way of disease detection. This can give the farmers an edge in protecting their crops, interests, and the food supply of the public.

The research in this field has been ongoing for quite some time, using several different methods and technologies. It will be discussed further later. As for with the latest trend of client-server architecture-based applications it would a perfect approach to take a mobile app and do the processing on a server. The use of image classification through technologies like machine learning and deep learning is perfectly supported by the modern smart phone which are equipped with latest high-resolution cameras.

2.1. Approaches

Following are the Machine learning and deep learning approaches that are used in the detection process.

Supervised Learning

Supervised learning is achieved by feeding pre-labelled data to an algorithm. The algorithm then learns to label the any new data provided correctly.

The supervised learning algorithm employs classification algorithms and regression techniques such as linear regression, logistic regression, and neural networks, as well as decision trees, Support Vector Machines (SVM), Naive Bayes, and k-nearest neighbour, to construct prediction models.

Random Forest

Random Forest is one of the most used traditional machine learning models. It consists of several different decision tree which make prediction about a single problem then their output is combined, known as assembling, to make the choice.

Decision tree

These are the building blocks of a random forest. Each node in tree represents a decision. A decision is made based on a property at every node and in the end, it reaches one of several leaf nodes which represent a class.

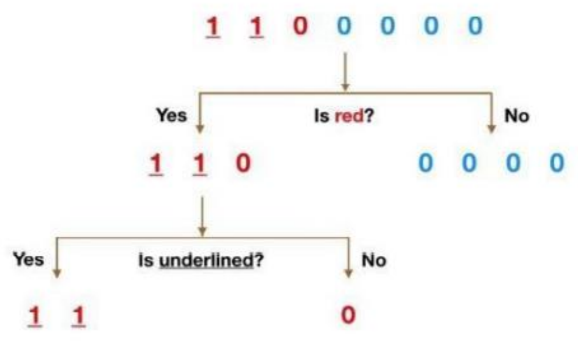


Figure 2.1: Decision Tree

Figure 2.1 first determines the number colour i.e., if a number is red or blue in colour, then determines whether the number is underlined.

Naïve Bayes

Another traditional algorithm, based on Bayes' theory. For our sake, it assumes that every property of data in independent from others.

It's simple to build and works well with large datasets. In addition to its simplicity, Nave Bayes is thought to outperform even the most complex categorization techniques. The fundamental Naive Bayes hypothesis is that each and every highlight generates an independent and break-even commitment to the final outcome. [2] [3]

Advantages	Disadvantages
• Simple to decide dataset classes. • Performs well in classifications with several classes. • Normalized input can obtain the desired level of optimization	• It is difficult to obtain predictors that are completely independent in real-life situations, as suggested by the Nave Bayes algorithm. • Inaccurate estimator – Probability can be deceiving at times. • The Zero Frequency issue.

Table 2.1: Advantages and disadvantages of Naïve Bayes

It's also used widely in other problems like Spam filtering, text classification, sentiment analysis etc. As stated in the similar system comparison section, the research that was conducted using the Nave-Bayes classifier yielded an accuracy of 87 percent.

Artificial Neural Networks (ANN)

Brains of living things are complex network of neurons, consisting of several layers of layers and each layer consisting of several neurons. ANNs try to mimic their behaviour to achieve a higher level of intelligence than traditional machine learning algorithms.

They are considered feed-forward neural networks because the inputs are passed only in one direction i.e., from the input layer to the output layer [4].

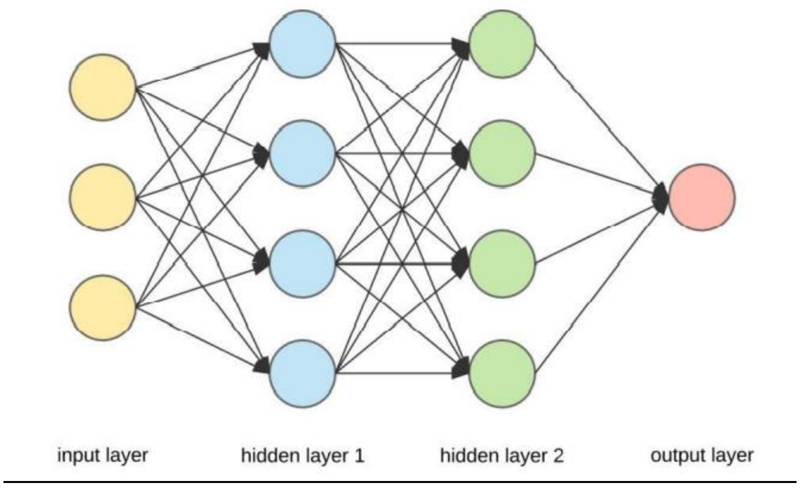


Figure 2.2: Artificial Neural Network (ANN) [8]

There are considered to be 3 fundamental types of layers in ANNs, Input layers, hidden layers, output layer

- Input layer is the one that receives initial information from external sources and propagates it forward for further processing, it can be considered as the interface for incoming information
- Hidden layers are the layers between input and output layers. These layers add bias to incoming input so that output layers correct neurons are activated to generate the results
- Output layer these neurons provide the results and are conditionally activated based on the incoming data

Convolution Neural Networks (CNN/ConvNet)

If ANNs try to mimic an animal brain then CNNs try to mimic the animal vision. The basic principle is based on the understanding of the working of sight. When presented with an image the feature of image is extracted, important information is maximized and unimportant is dropped. Then the image is classified based on the presence of certain features in it [9].

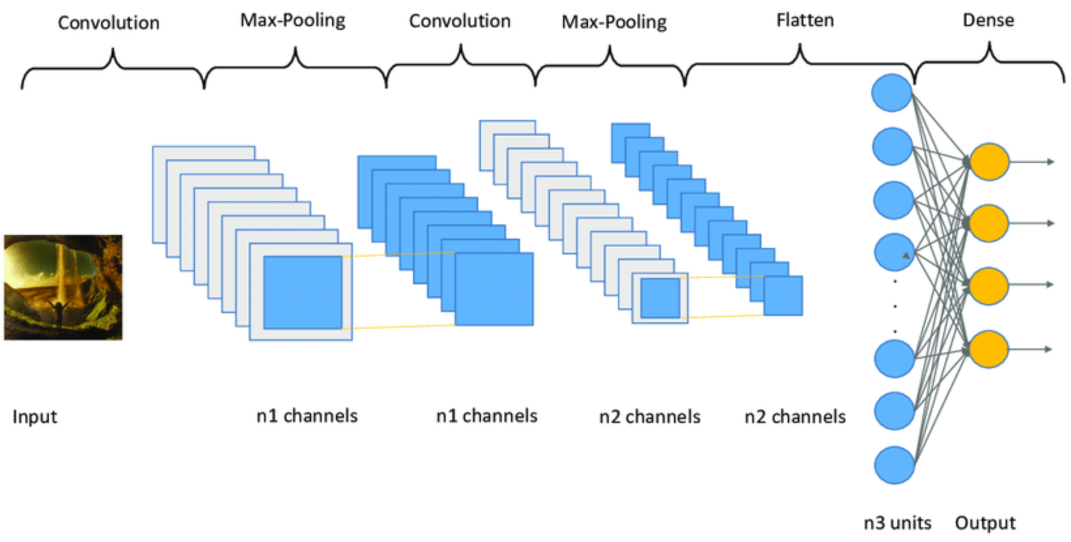


Figure 2.3: Convolution Neural Network [10]

In input layer, each neuron basically receives the value a single pixel. So, the number of neurons and their arrangement has to be exactly the same as the pixels in the image. They can either receive grayscale values for grey images or RBG values for colour images. This has to be decided at the time of developing the CNN along with the shape and size of input layer.

Then the convolutional layers do the feature extraction from images, after that the pooling layers make that extracted information easier to process. Then the deep layers add further bias if needed and finally output layer finishes the job.

This approach takes advantage of the raw computing power available with modern machines which wasn't available before.

2.2. Algorithm review

In this section, we will review all the algorithms and find the best for our use.

2.2.1. Traditional Machine Learning Algorithms

As expressed above, different kinds of machine-learning techniques can be used in image recognition. This the comparison between traditional machine learning algorithms available for image classifications.

Algorithm	Problem	Predictors	Power	Implementation	Interpretability	Normalization
K-NN	Multiclass or binary	Numeric	Medium	Easy	Good	Required
Naïve Bayes	Multiclass or binary	Categorical	Medium	Medium	Good	Required
Decision Tree	Multiclass or binary	Numeric or Categorical	High	Difficult	Good	No
Random Forest	Multiclass or binary	Numeric or Categorical	High	Difficult	Good	No

Table 2.2: Machine learning approaches comparison

In image classification, the most extensively used conventional machine learning techniques are the above-mentioned established algorithms, Random Forest and Nave Bayes.

Why deep learning over traditional machine learning?

When a machine-learning algorithm makes a less accurate prediction, the developer/system engineer must intervene and make changes to get the intended outcome. Neural network-based deep learning models can make more accurate predictions while also judging whether the prediction is right. Deep learning accomplishes feature extraction automatically, whereas traditional machine learning requires manual feature extraction.

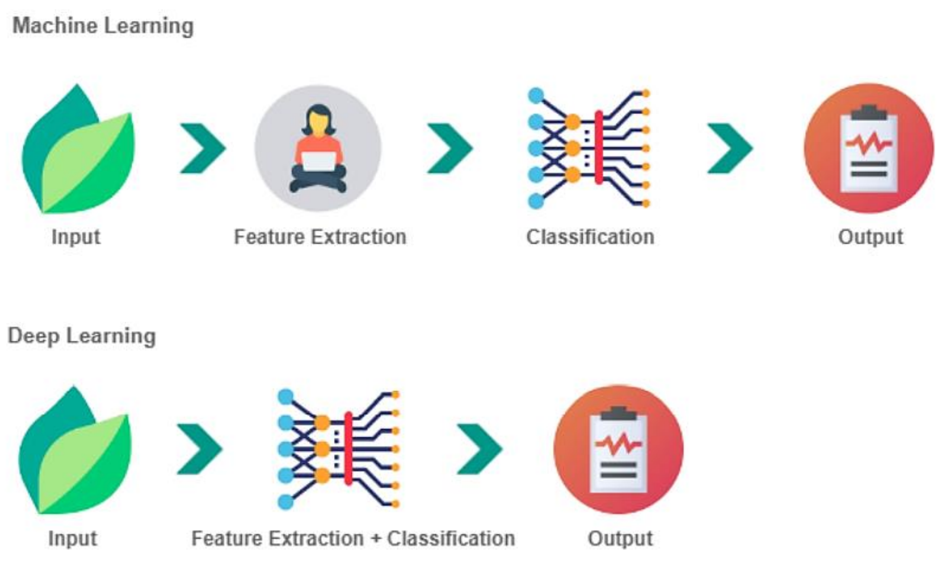


Figure 2.4: Deep learning vs Machine learning [8]

Traditional machine learning and deep learning techniques are compared below in table 2.4.

	Traditional Machine Learning	**Deep Learning**
Process	Input data, develop a mathematical model, use mathematical model to make predictions.	Train the neural network on training data to achieve right bias for internal neurons. Then pass new data through the neurons that make decisions from previous learning.
Data requirements	Less amount of data required compared to deep learning.	To make accurate predictions larger amount of data required.
Hardware requirements	It is necessary to use less computational power.	More computational power is required.
Feature extraction	Manual feature extraction is required.	Feature extraction done automatically (within convolutional layers for CNNs)
Accuracy	When compared to modern deep learning techniques, accuracy is lower.	Possible to achieve high accuracy with enough training.
Execution time	Faster training and faster predictions.	Slow training but less prediction time.
Flexibility and Optimization	Less flexible need more manual optimization	More flexible, can do more self-optimization.

Table 2.3: Differences and similarities between Traditional Machine learning and Deep learning Techniques

Chapter 3

System Requirements

This section enlists and describes the requirements agreed upon during the time of acceptance of this project.

3.1. Use Case Diagram

This is the use-case diagram for manage plant diseases. In this use-case, the user can browse through all the diseases and search any of them through the search bar.

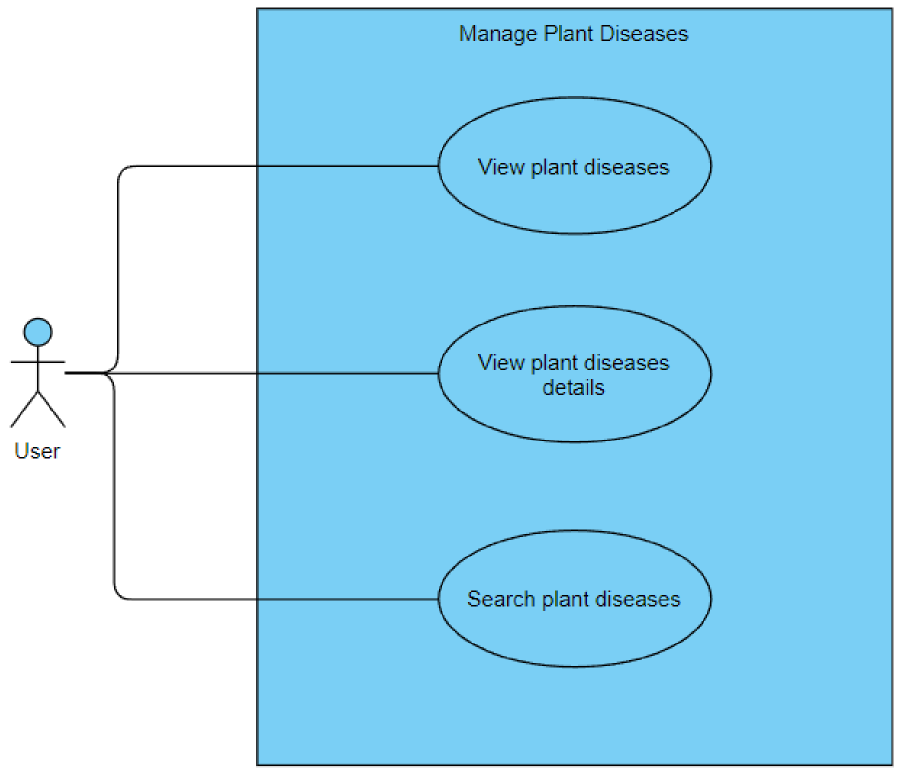


Figure 3.1: Plant diseases management use-case

Management of plant disease detection use-case will allow the user to upload any image from the camera or gallery for the prediction and after prediction a report will be shown to the user.

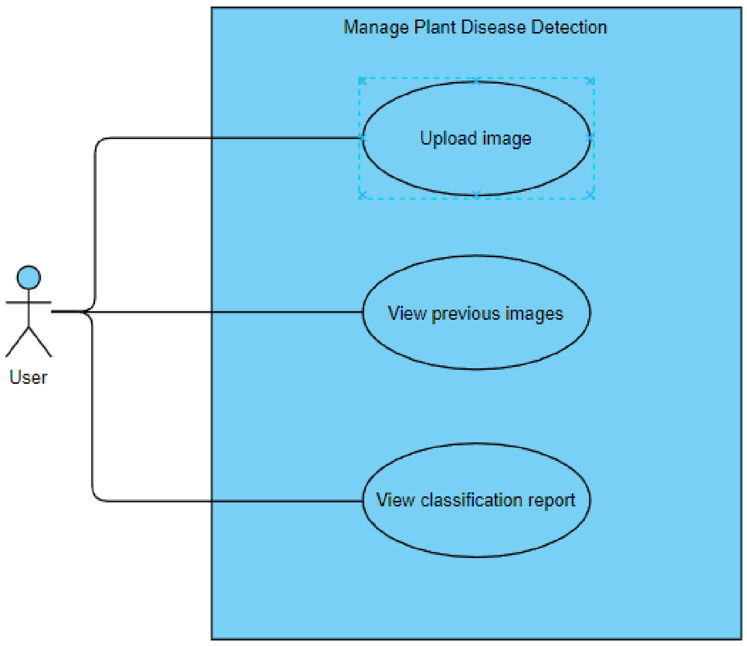


Figure 3.2: Manage plant disease detection use-case

In this manage user account use-case, the admin will be able to manage all the users. He will be able to add, edit or delete any user from the database.

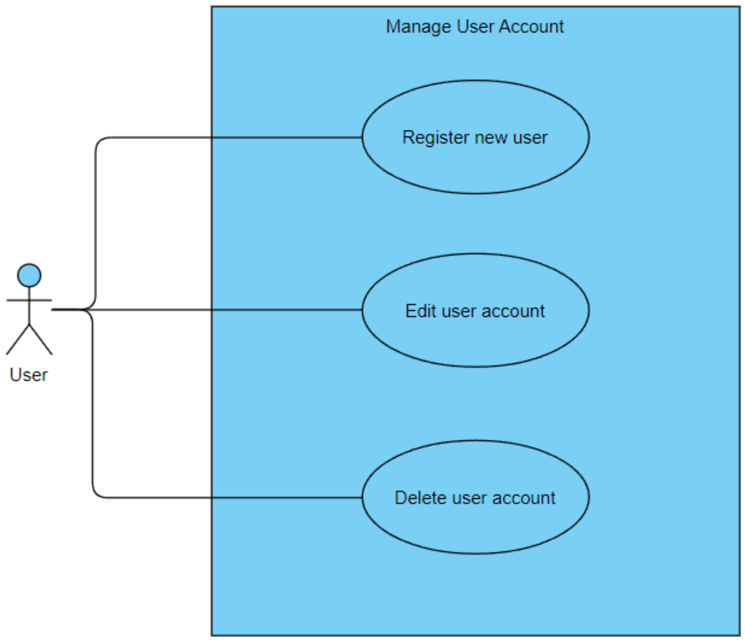


Figure 3.3: Manage user account use-case

PlantDoctor system use-case is the whole view of the system which describes the overall system functionality:

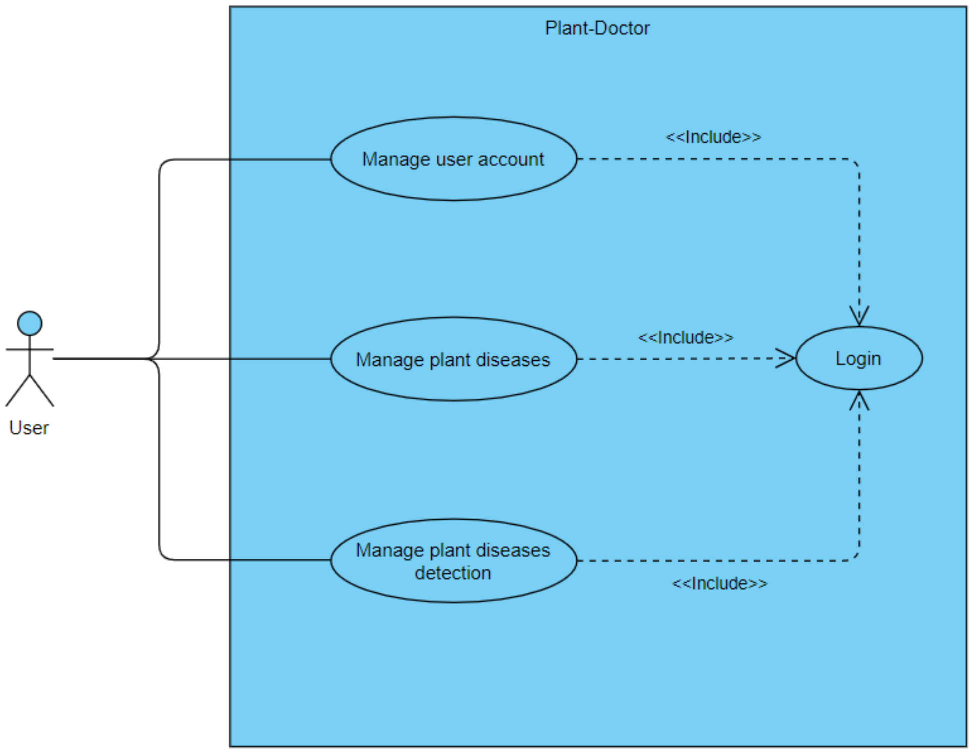


Figure 3.4: PlantDoctor system use-case

3.2. Interface Requirements

User Interfaces

This project offers a decent graphical interface for the user that can be run on the device by any user. The application's front-end is written in React.js and is completely separate from the Flask backend application, which handles plant disease detection and provides relevant plant condition information to the user interface (UI), where the final expected result of the entire process is displayed to the user. The user interface should be able to communicate with the user management module, and a portion of it should be dedicated to the login/logout module.

Hardware Interfaces

1. Processor: Intel Core i7, 10th Gen
2. Memory: 12GB
3. Graphics: GTX 1050, 4GB
4. Storage: Solid State Drive: 512GB

} PC

5. Processor: Hexa-core (2x2.65 GHz Lightning)
6. Memory: 3GB
7. Internal Storage: 64GB
8. Screen (Size & Type): 5.4 inches, IPS LCD

$\}$ Mobile

Software Interfaces

Software requirements

- Operating system (OS):
    - Microsoft Windows 10 or 11
    - iOS 15 or Android 11
- Languages:
    - Python 3.9.7
    - JavaScript
- Libraries:
    - TensorFlow
    - Transformers
    - Torch
    - OpenCV
    - NumPy
    - Flask
    - Pandas
- Frameworks
    - React Native (Expo)
- Database
    - MySQL
- IDEs
    - PyCharm

- Visual Studio Code

3.3. Functional Requirements

Our system has the following functional requirements which the system must provide to the user.

- **User authentication**
    - Only authenticated users will be able to access the system.
- **Image detection**
    - To detect images, the user must be logged in and can take an image from the mobile camera or select an image from the gallery.
- **Plant disease detection**
    - The application should recognize the plant disorder and generate a result for the image that is being sent for the images detection, the API will detect the plant disorder with the help of the model that has been trained by the dataset.
- **Browse or search plant diseases**
    - The system should come up with the feature to the user to browse through all the disease that our application is capable of and find relevant information through the system.
- **Manage user account**
    - The system should allow the user to create an account on the application by providing necessary information of the user.

3.4. User Management

3.4.1. User Login

Use Case ID:	**1**
Use Case Name:	User Login
Actor(s):	User
Pre-Conditions:	User must have an account and connected to the internet.
Priority:	High

Basic Flow:	Authenticated user clicks on Login button and Login page will be displayed so that user can get registered.	
Actor Actions		**System Response**
1	User enters email, and password in the relevant fields. Users click on “Register” button	**2** System check user’s provided credentials. If user’s credentials are valid, user will be Logged in and home screen is displayed.
3	Click “Register” button	**4** Display Register screen
Alternative Course of Action (if any)		
Actor Action		**System Response**
	2.a User press Login button	**2.b** If user’s credentials are invalid an error message is displayed and login screen is shown/redisplayed.
Post condition	• The application database is updated with the relevant details. • User can view the detail and can upload an image to prediction the disease.	

Table 3.1: User Login

3.4.2. User registration

Use Case ID:	**2**	
Use Case Name:	**User registration**	
Actor(s):		
Pre-Conditions:	User must be connected to the internet.	
Priority:	High	
Basic Flow:	Authenticated user clicks on registration button and registration page will be displayed so that user can get registered.	
Actor Actions		**System Response**
1	User enters full name, email, and password in the relevant fields. Users click on “Register” button	**2** System check user’s provided credentials. If user’s credentials are valid, user get registered and home screen is displayed.
3	Click “Login” button	**4** Display Login screen
Alternative Course of Action (if any)		
Actor Action		**System Response**

	2.a User press register button	**2.b** If user's credentials are invalid an error message is displayed and register screen is shown/redisplayed.
Post condition	• The application database is updated with the relevant details. • User can view the detail and can upload an image to prediction the disease.	

Table 3.2: User registration

3.4.3. Upload plant leaf image

Use Case ID:	**3**		
Use Case Name:	Upload plant leaf image		
Actor(s):	User		
Pre-Conditions:	User should be logged in to the system		
Priority:	High		
Basic Flow:	Authenticated user clicks on camera, then camera will be opened, and user can click the image.		
Actor Actions		**System Response**	
1	The user presses "Take a picture" button.	**2**	The device ask for camera permissions before the camera screen is opened, and instructions are displayed.
3	The user takes a photo of the plant leaf and then clicks the "Select" button. The image is then sent to the server-side application (Flask back-end application).	**4**	The Hugging Face API is used by the Flask backend application to determine whether a plant is infected or healthy, as well as the confidence level, and delivers the response in json format to the frontend so that it may be displayed to the user. The use case ends while showing the user the uploaded image, condition of plant and confidence level in the user interface (UI.

Alternative Course of Action (if any)			
Actor Action		**System Response**	
1.a	Execute another function	**2.a**	Exit from the use case.
3.a	Press “Choose from gallery” button	**1**	Smart phone gallery is opened.
		2	The user chooses an image or images from the gallery.
4.a	Press “Cancel” button	The use case is exited.	
6.a **6.b**	If the image is not a plant leaf image If the image is corrupted	**1**	1. System recognized the uploaded image and “Unknown image” with low confidence is shown to the user. 2. Display waring message saying “Plant leaf image is required”
		2	1. Display Error message
Post Conditions:	• Update the system database. • User can upload more images to prediction.		

Table 3.3: Upload plant leaf image

3.5. Non-Functional Requirements

Accuracy

- PlantDoctor application accuracy must be the most critical feature. The provided results must be accurate with the correct predictions of plant disease, otherwise the users may be misled, resulting in plant damage.

User-Friendly

- Being user-friendly is important because the application users may lack technical expertise, the system must include an easy-to-use graphical user interface (GUI).

Performance

- As the application is primarily concerned with the detection of crop disorders; the process heavily relies on time; and the disease recognition process should be much more efficient and accurate.

Scalable

- As for now, the system only has the functionality for the plant leaf detection but in the future, other needs can be developed for example, the functionality for the full trees identification, birds, or insects' identification. So, system should be flexible for the future changes and upgrades.

3.6. Security Requirements

- Ordinary users can read information and can't adjust it, aside from their own data.
- Every user will have access constraints.

3.7. Software Quality Attributes

- **Useability:**
  The system is designed in such a way that naive user can use it easily. It will not have complex design.
- **Availability:**
  System will be accessible to the users all day, every day. Clients can use in any time.
- **Interoperability:**
  System will be developed in cross platform, so it can run on both Android and IOS.
- **Scalability:**
  The system will accept more image formats in future.

3.8. Analysis Models

This analysis model present with the workflow process of our system with the following activity diagram.

Activity Diagram:

Following diagram is the system activity diagram which is describing step by step activity of the user through the application and how the application will behave accordingly.

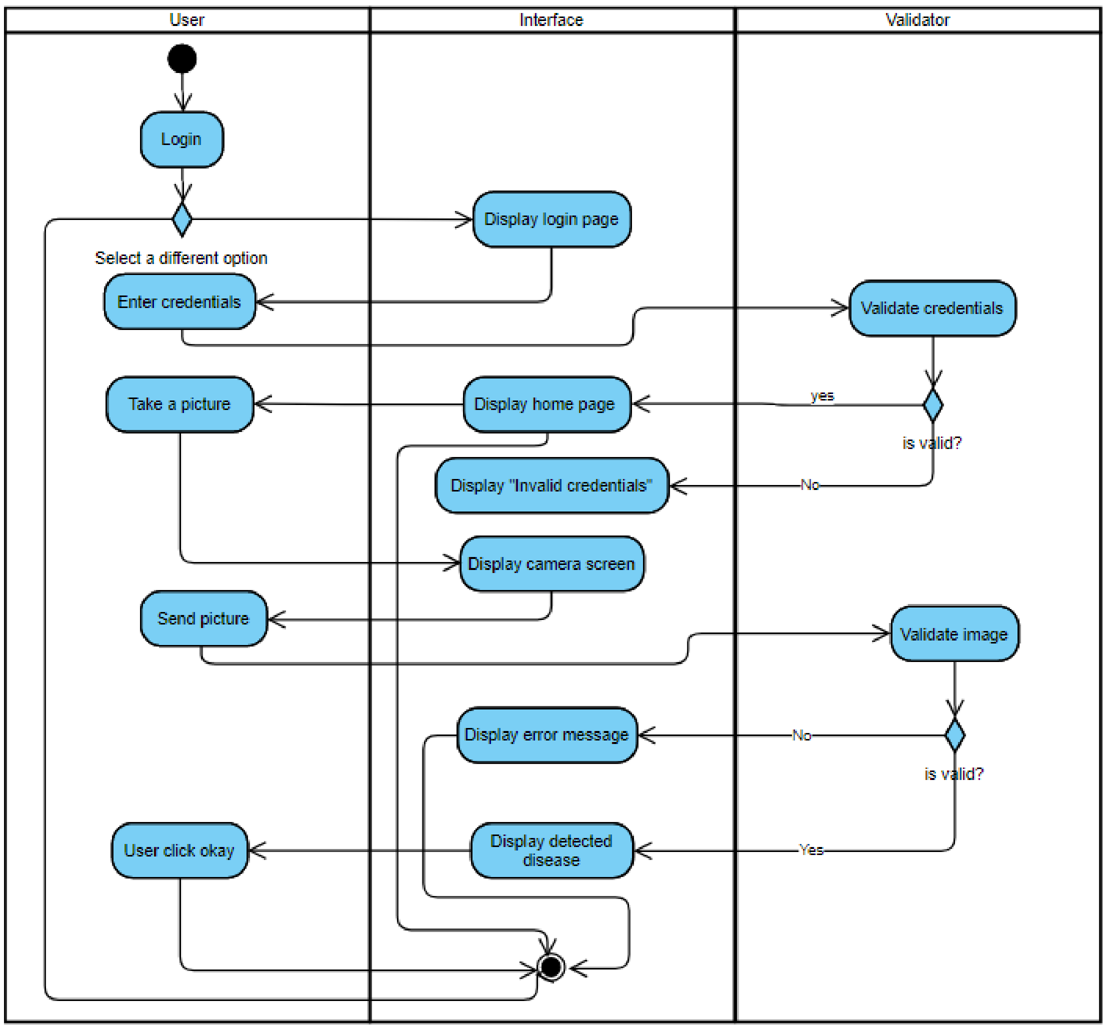


Figure 3.5: Activity diagram

Chapter 4

System Design

Following section contain the design related diagrams.

4.1. System Architecture

React Native is a popular cross-platform framework which helps us to build natively rendered iOS and Android apps. We can use the same codebase for multiples platforms. There are four core sections:

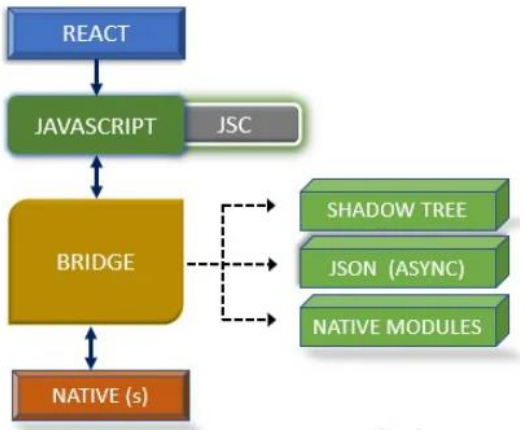


Figure 4.1: Framework Architecture [9]

- The React code written by the developer.
- The JavaScript that is eventually interpreted from the code written by the developer.
- A collection of elements known collectively as The Bridge.
- The indigenous side.

4.2. Logical Design

Following are the class diagrams for overall system and model

4.2.1. Class Diagram:

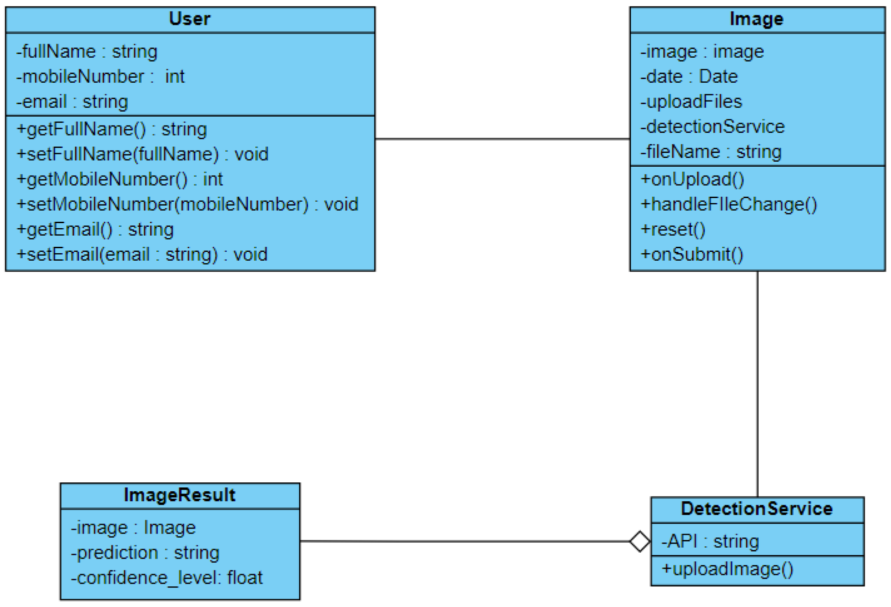


Figure 4.2: Client-Side applications

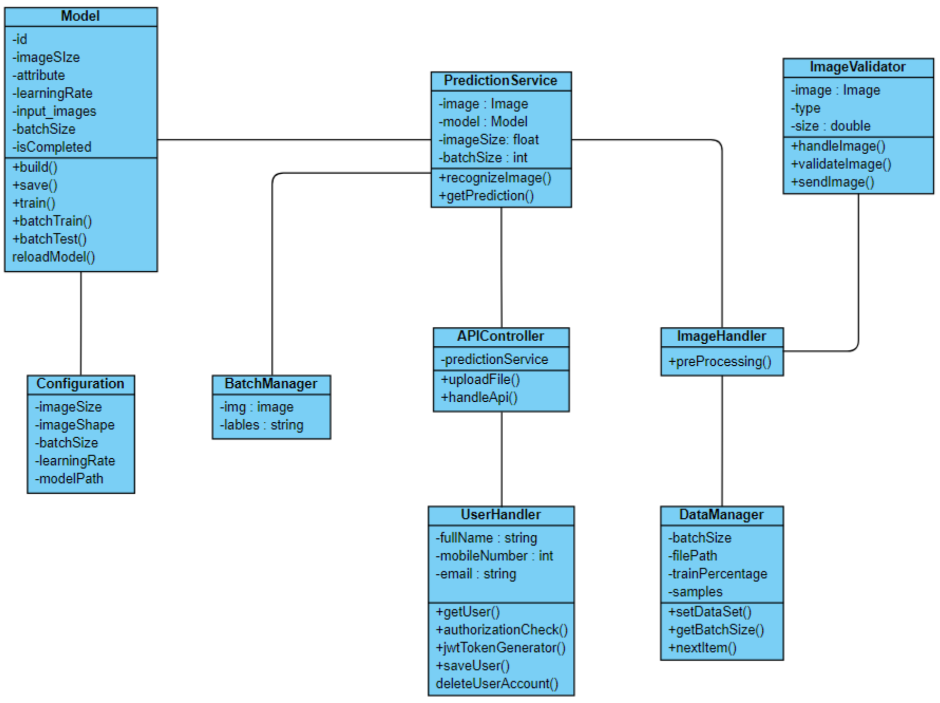


Figure 4.3: Server-side application

4.3. Dynamic View

Overall view of the system:

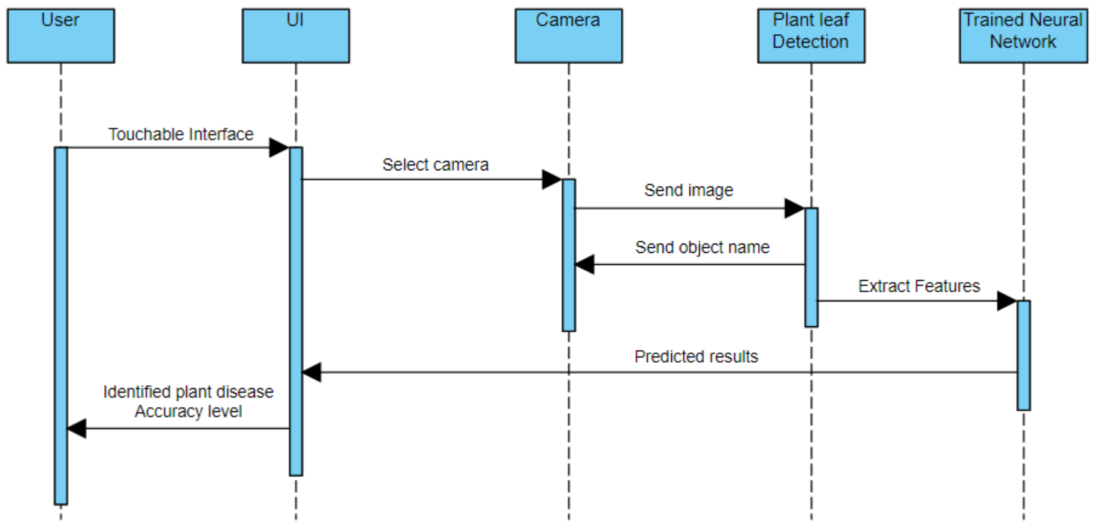


Figure 4.4: System sequence diagram

Create Model View:

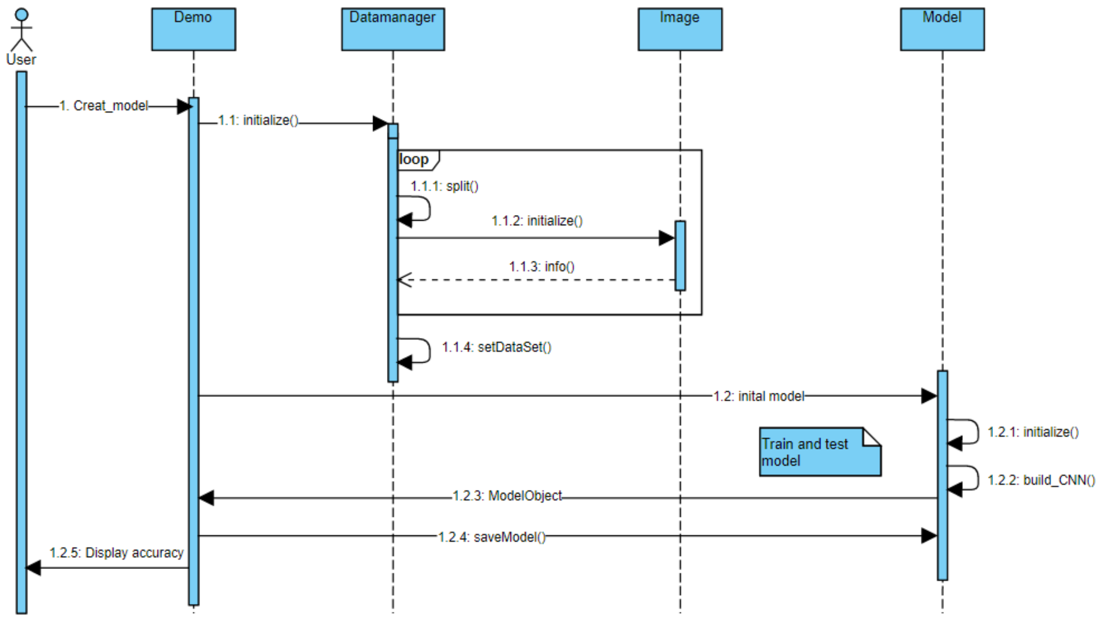


Figure 4.5: Model sequence diagram

4.4. Development View

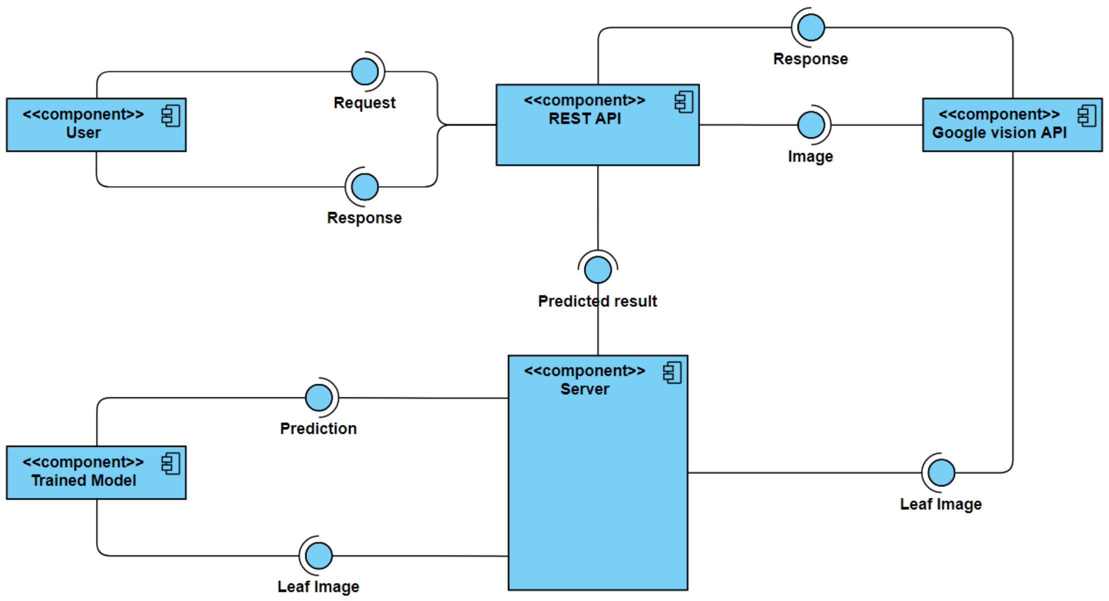


Figure 4.6: Deployment diagram

4.5. Data Models

Client Slide

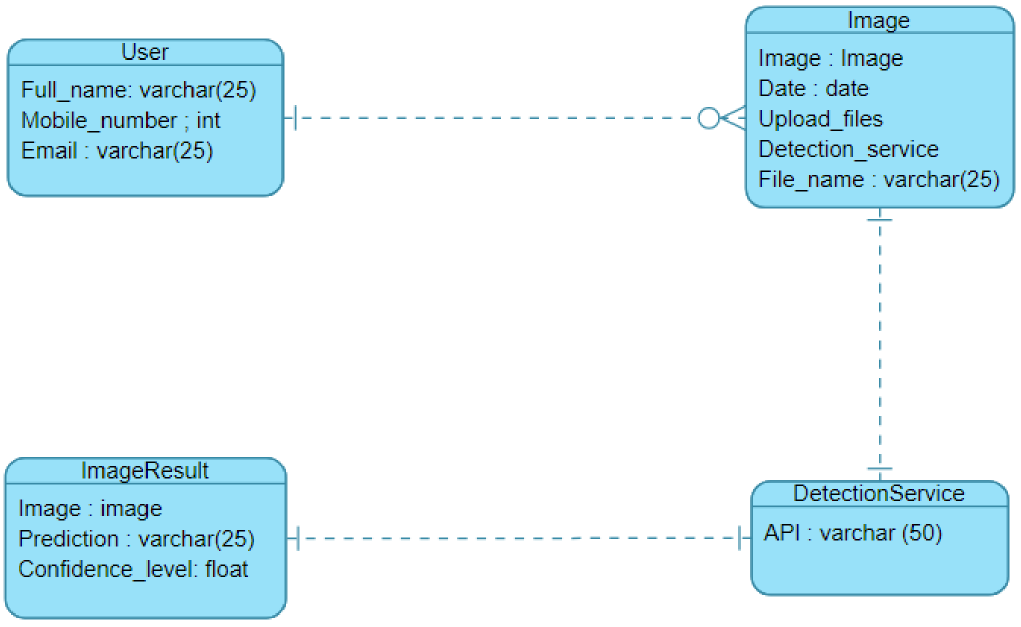


Figure 4.7: Client-side data model

Server Side

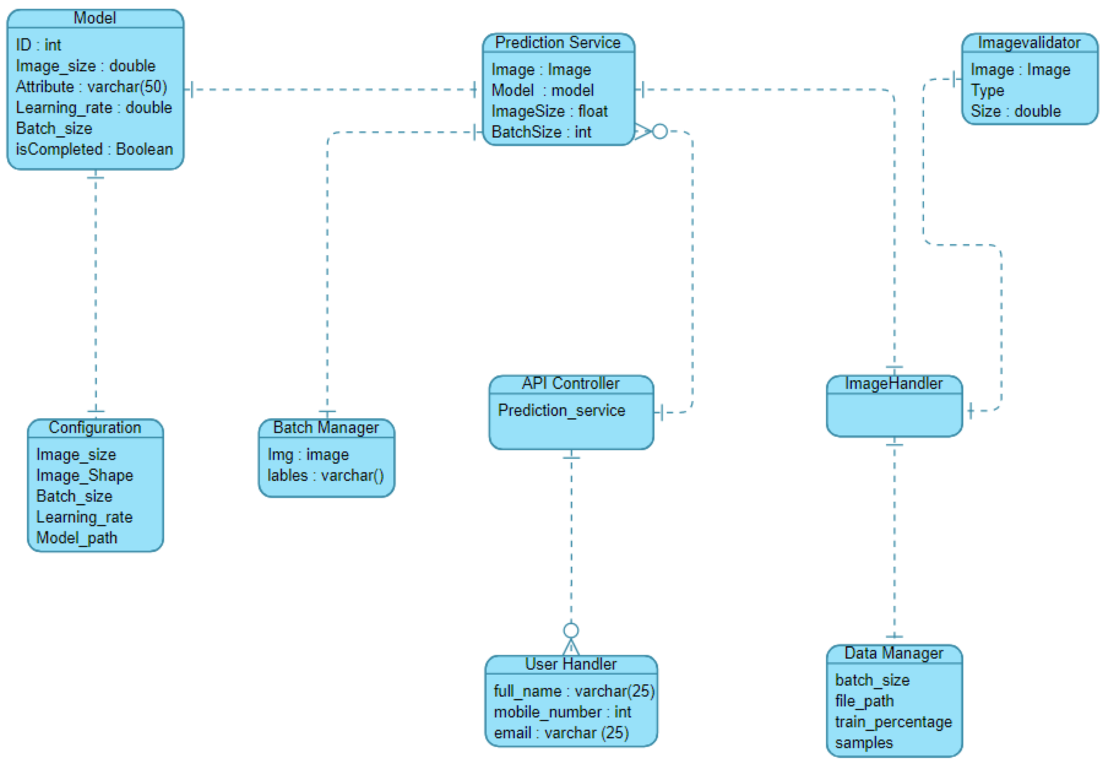


Figure 4.8: Server-side data model

4.6. Component Design

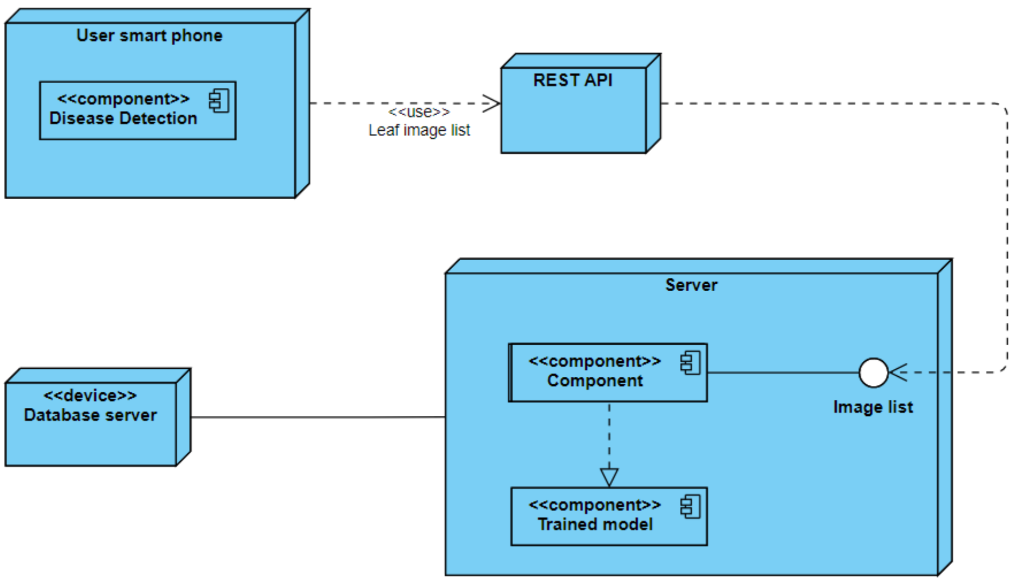


Figure 4.9: Component diagram

4.7. User Interface Design

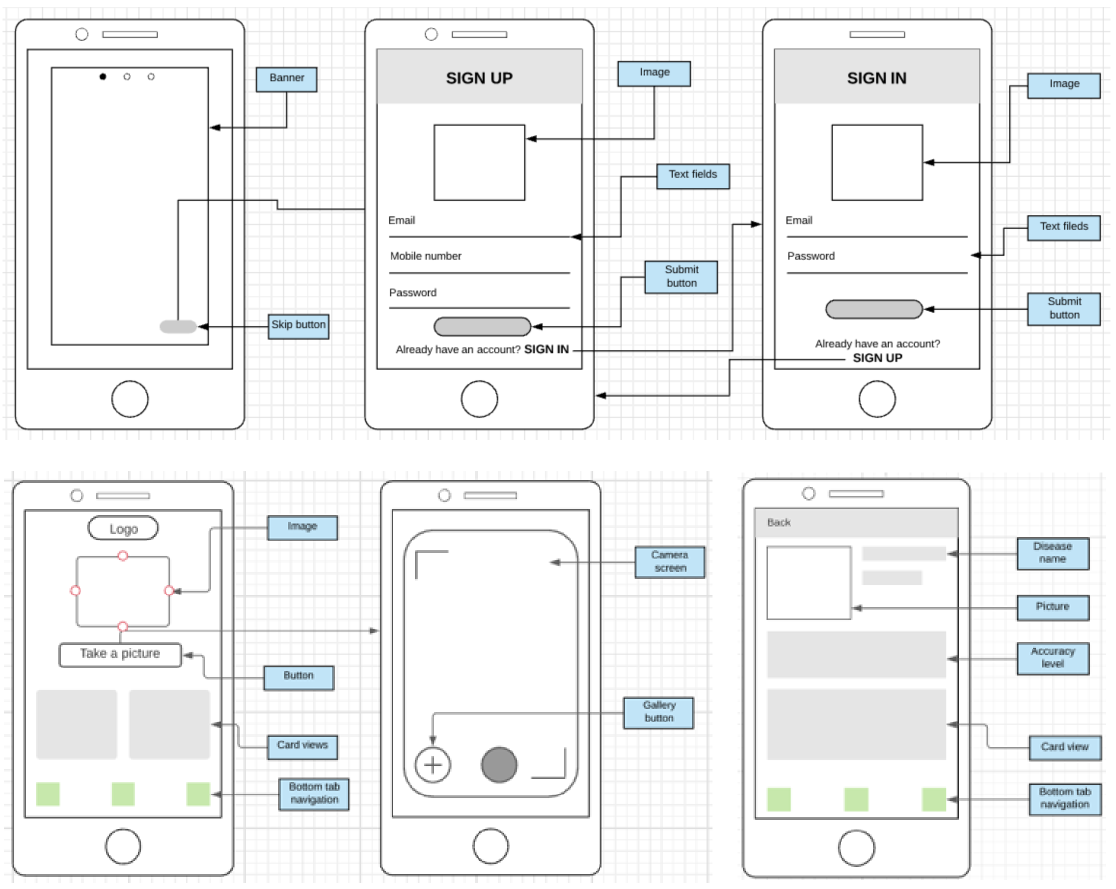


Figure 4.10: PlantDoctor wireframes

Chapter 5

Methodology

5.1. Dataset

The dataset we used for this project is **PlantVillage**, a public dataset curated by Sharada P. Mohanty et al [10]. This dataset contains 87900 RGB images of healthy and diseased plant leaves (after augmentation). It consists of 38 classes, of which we have chosen all of them for training our model. Table 5.1 depicts these classes.

No	Plant name	Plant Disease Name	No. of Images
1	Apple	Healthy	2008
		Diseased Scab	2016
		Diseased: Black rot	1987
		Diseased: Cedar apple rust	1760
2	Blueberry	Healthy	1816
3	Cherry	Healthy (Including sour)	1826
		Powdery mildew (Including sour)	1683
4	Corn	Healthy	1859
		Diseased: Cercospora leaf spot	1642
		Diseased: Common rust	1907
		Diseased: Northern Leaf Blight	1908
5	Grapes	Healthy	1692
		Diseased: Black rot	1888
		Diseased: Esca (Black Measles)	1920
		Diseased: Leaf blight (Isariopsis)	1722
6	Orange	Haunglongbing (Citrus_greening)	2010
7	Peach	Healthy	1728
		Diseased: Bacterial spot	1838
8	Pepper	Bell Healthy	1988

		Diseased: Bell (Bacterial spot)	1913
9	Potato	Healthy	1824
		Diseased: Early blight	1939
		Diseased: Late blight	1939
10	Raspberry	Healthy	1781
11	Soybean	Healthy	2022
12	Squash	Powdery mildew	1736
13	Strawberry	Healthy	1824
		Diseased: Leaf scorch	1774
14	Tomato	Healthy	1926
		Diseased: Bacterial spot	1702
		Diseased: Early blight	1920
		Diseased: Late blight	1851
		Diseased: Leaf Mold	1882
		Diseased: Septoria leaf spot	1745
		Diseased: Two-spotted spider mite	1741
		Diseased: Target Spot	1827
		Diseased: Yellow Leaf Curl Virus	1961
		Diseased: Tomato mosaic virus	1790

Table 5.1: Dataset Specification

Some images from the dataset are shown in figure below:

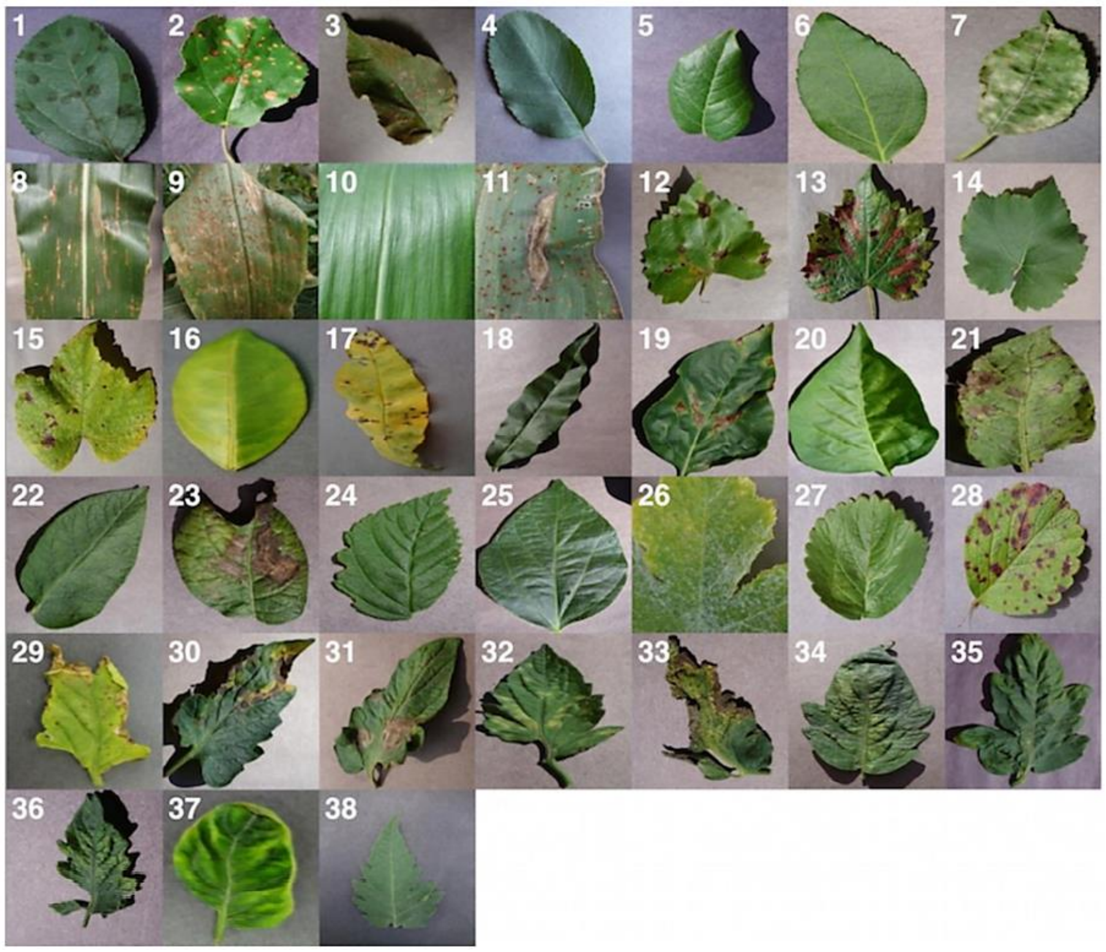


Figure 5.1: Sample images in the dataset [14]

5.2. Splitting dataset

Training data	Validation data
80% (70320 images)	20% (17580 images)

5.3. Model Training:

As we were initially using CNNs for image classification in our project, we used rather basic approaches towards the goal.

1. To build a neural net from scratch based on structures suggested in research papers on plant disease detection.

2. Use a pretrained model with ImageNet weights from Kera's library.

During our interim evaluations, we were suggested to try transformers as well. When we used them, we got the best results of all. We used the Swim Tiny transformer.

Training a CNN from scratch:

Initially, since we had limited resources, we used an image size that was too small and it got overfit. When we increased the image size used the full images the accuracy got low and the model was also underfit. So, we kept on trying different combinations, but we couldn't improve a lot.

The best we could do was reach about 90% percent accuracy and the generalization was also quite bad. The model couldn't correctly classify most images from the dataset it was trained on, if fed by being taken with a camera off the screen of a computer. Also, it gave considerably high confidence on images belonging to classes which were irrelevant to our dataset.

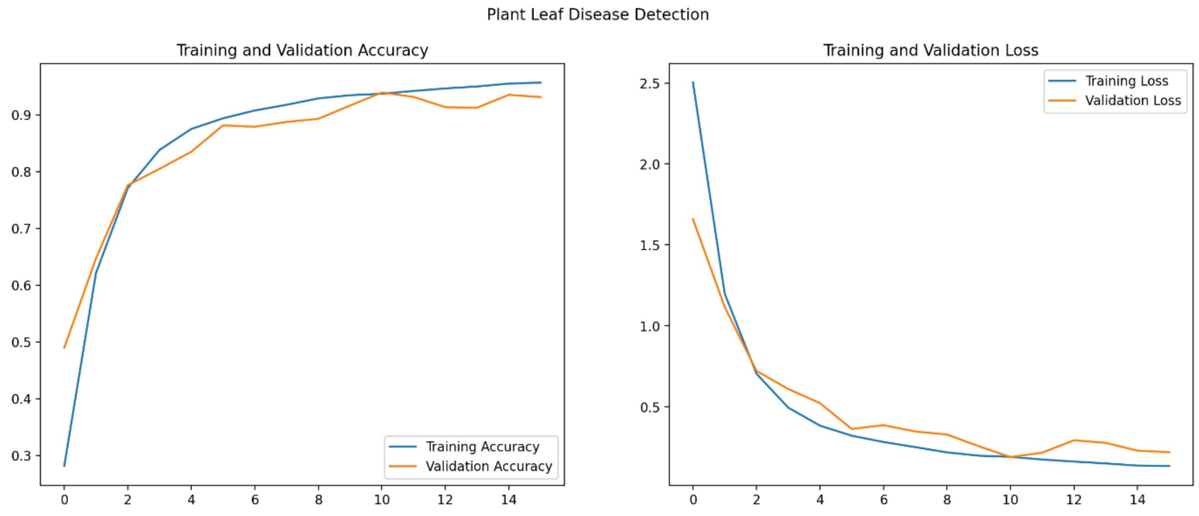


Figure 5.2: CNN training graphs

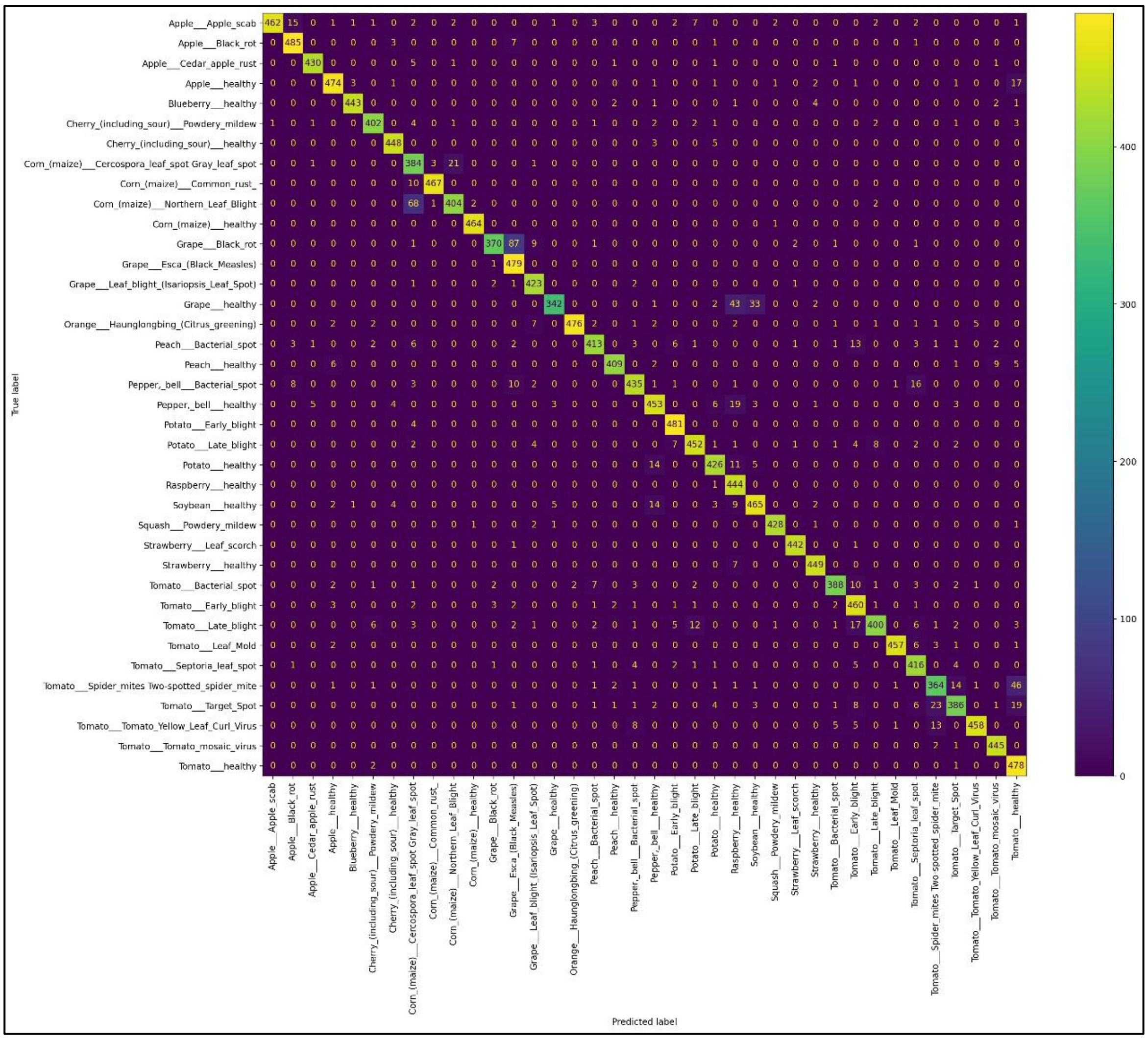


Figure 5.3: Confusion Matrix

Training a CNN with transfer learning:

After failing to achieve any satisfactory result from custom CNNs, we searched for better ways. We realised that we had started off on wrong foot. There was no need to train a CNN from scratch as there were already trained ones out there which could be modified to fit almost any use-case.

We tried 3 different architectures ResNet, Inception and VGG as they were the most suggested ones. All of them provided high training and validation accuracy on dataset but we found out that inception and ResNet architectures were prone to overfitting on our dataset.

Inception was always getting overfit, so we didn't go for it.

The following figures shows the training graphs of CNN model.

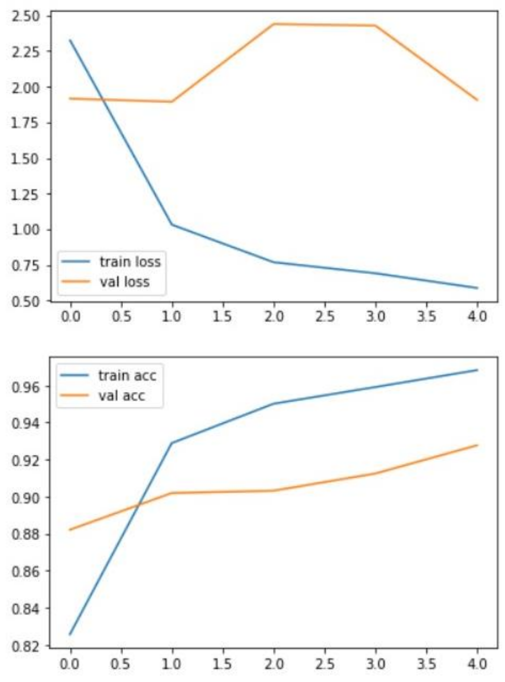


Figure 5.4: CNN (Transfer learning) training graphs

ResNet didn’t do much better either as shown in figure 5.12.

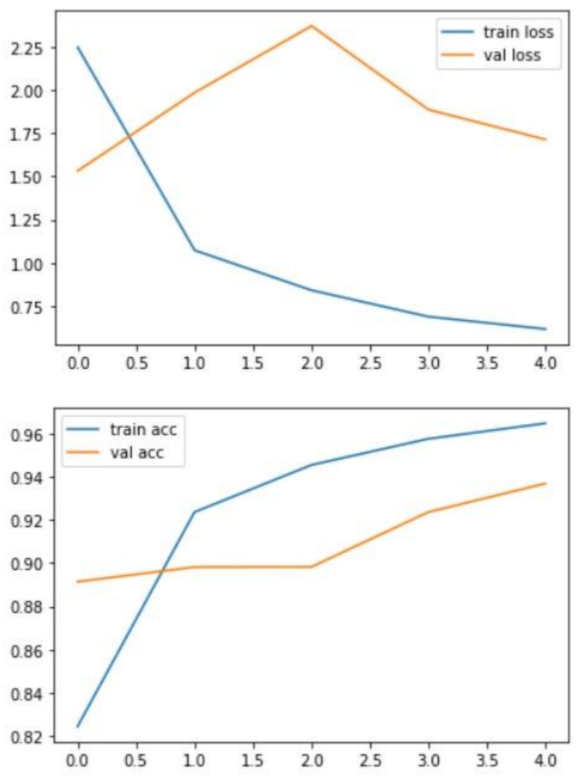


Figure 5.5: CNN (Resnet) training graphs

Only VGG provided some promising initial results.

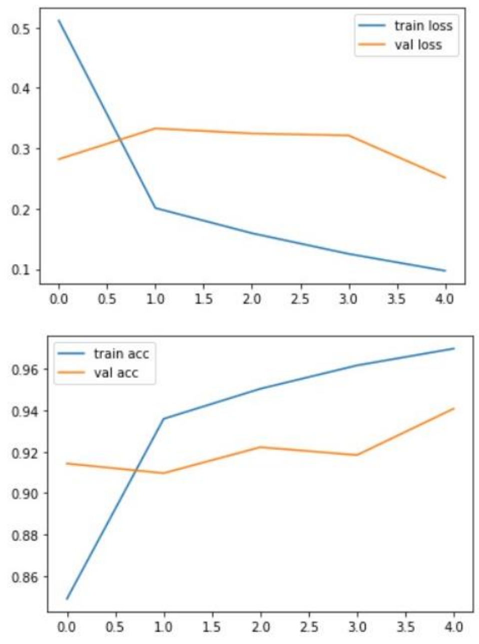


Figure 5.6: CNN (VGG) training graphs

Afterwards we opted for further training and optimizing VGG on our dataset.

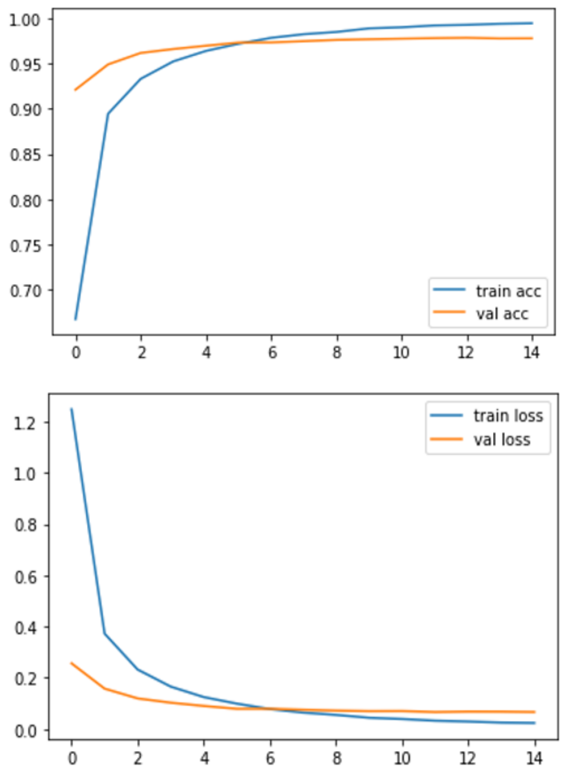


Figure 5.7: CNN (Optimized VGG) training graphs.

Transformers:

The final method of image classification we tried was transformers. It achieved the best results of all. Also, it is the first time that transformers have been used for plant disease detections.

A Transformer is a model architecture that does not use recurrence and instead draws global dependencies between input and output using an attention mechanism. The dominant sequence transduction models prior to Transformers were based on complex recurrent or convolutional neural networks that included an encoder and a decoder. The Transformer also has an encoder and decoder, but by foregoing recurrence in favour of attention mechanisms, it can achieve significantly more parallelization than RNNs and CNNs. [12]

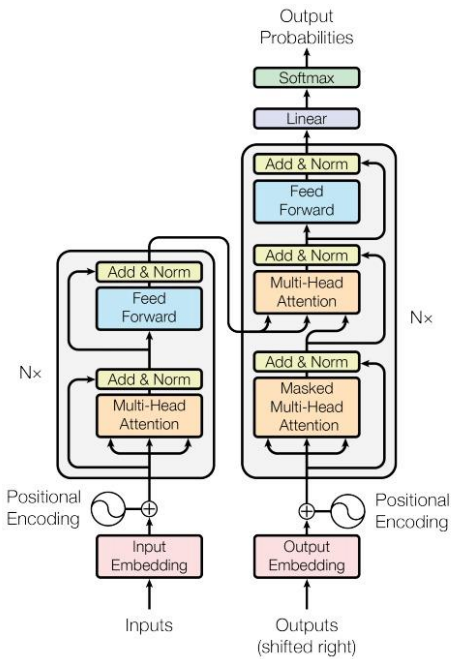


Figure 5.8: Transformer general structure [13]

The Vision Transformer, or ViT, is an image classification model that employs a Transformer-like architecture over patches of an image. An image is divided into fixed-size patches, which are then linearly embedded. Position embeddings are then added, and the resulting vector sequence is fed into a standard Transformer encoder. The standard approach of adding an extra learnable "classification token" to the sequence is used to perform classification.

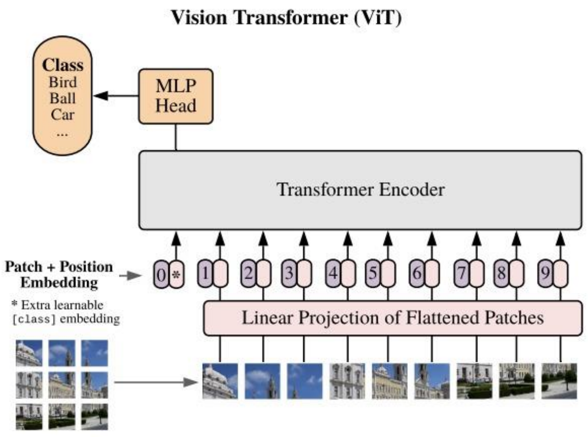
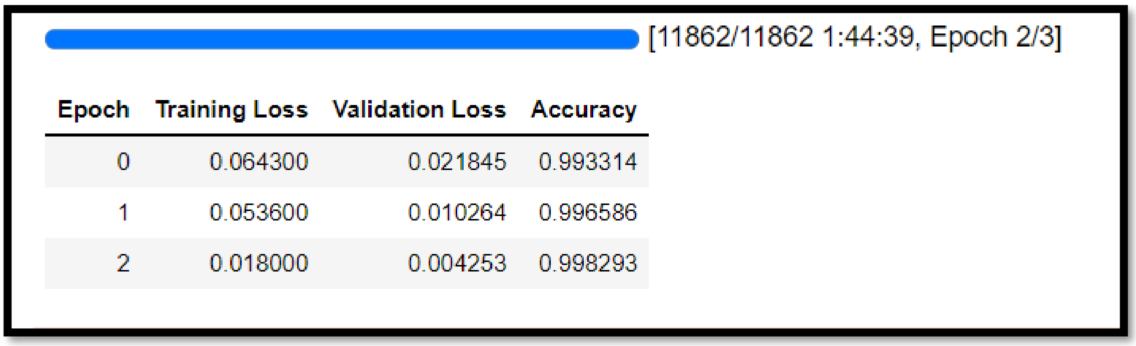

Vision Transformer (ViT)
Class
Bird
Ball
Car
...
MLP Head
Transformer Encoder
Patch + Position Embedding
* Extra learnable [class] embedding
0
*
1
2
3
4
5
6
7
8
9
Linear Projection of Flattened Patches

Figure 5.9: Vision transformer

We used swin-tiny transformer. It's a transformer for image classification based on swin architecture by Microsoft. It uses the same feature extractor under the hood as the ViT transformer. Batch size we used was 4.

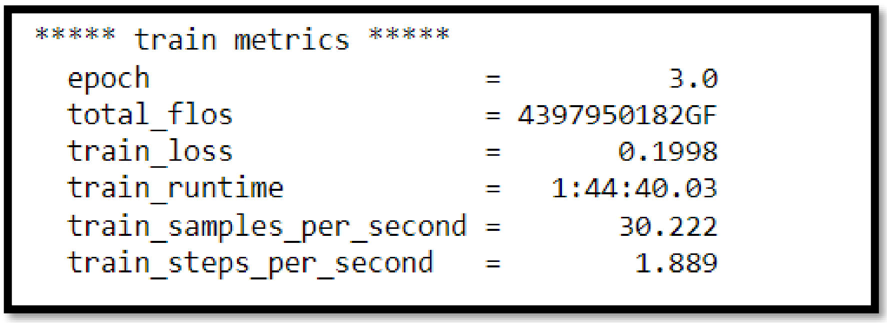
Swin transformer training metrics are shown below,



[11862/11862 1:44:39, Epoch 2/3]

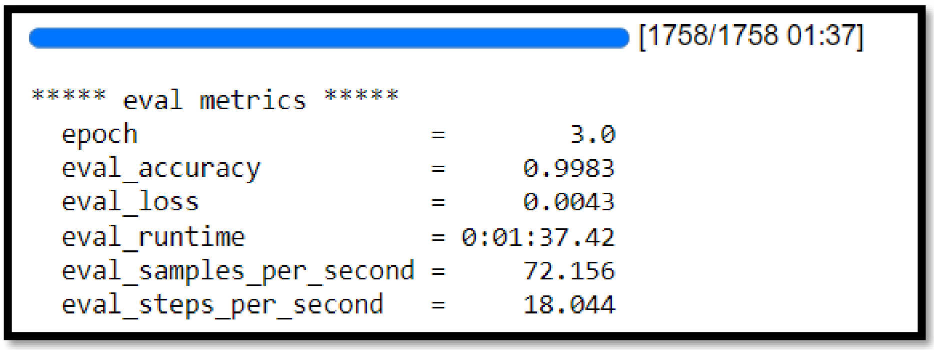
Epoch	Training Loss	Validation Loss	Accuracy
0	0.064300	0.021845	0.993314
1	0.053600	0.010264	0.996586
2	0.018000	0.004253	0.998293

Figure 5.10: Training epochs



```
***** train metrics *****
  epoch                    =          3.0
  total_flos               = 4397950182GF
  train_loss               =       0.1998
  train_runtime            =   1:44:40.03
  train_samples_per_second =       30.222
  train_steps_per_second   =        1.889
```

Figure 5.11: Train metrics



```
[1758/1758 01:37]

***** eval metrics *****
  epoch                   =        3.0
  eval_accuracy           =     0.9983
  eval_loss               =     0.0043
  eval_runtime            = 0:01:37.42
  eval_samples_per_second =     72.156
  eval_steps_per_second   =     18.044
```

Figure 5.12: Eval metrics

Chapter 6

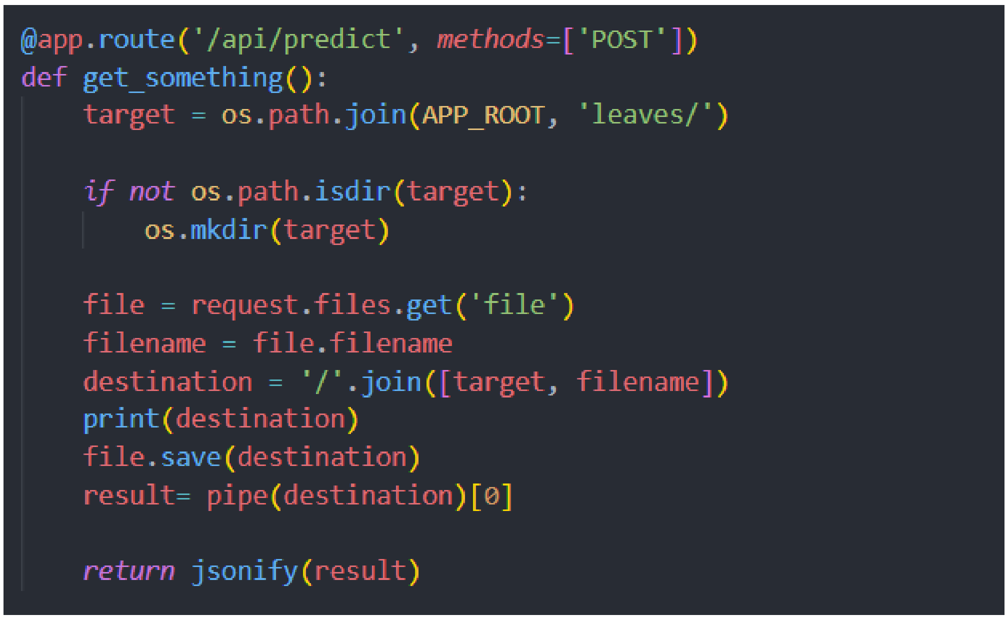
System Implementation

6.1. Flask Backend implementation

We have used the Flask for the backend as it is fit to use for our demands and easy to configure which was discussed in the previous chapters. We used these 3 methods for making an API call to get our disease prediction result.

Method 1 (Using Flask backend with model within the system):

Http Request handling and process of loading deep learning model is done through the backend of the application.



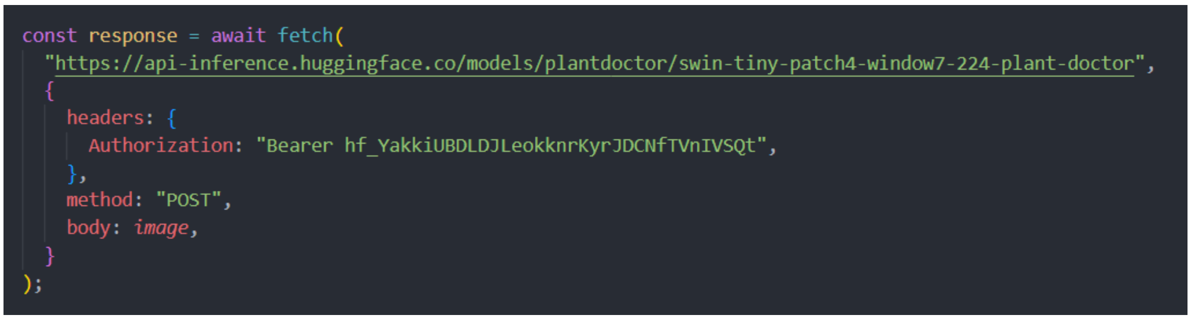
```
@app.route('/api/predict', methods=['POST'])
def get_something():
    target = os.path.join(APP_ROOT, 'leaves/')

    if not os.path.isdir(target):
        os.mkdir(target)

    file = request.files.get('file')
    filename = file.filename
    destination = '/'.join([target, filename])
    print(destination)
    file.save(destination)
    result= pipe(destination)[0]

    return jsonify(result)
```

Figure 6.1: API POST method for disease prediction

When an image is uploaded, the client sends a **POST** request, which is handled by the above method. To upload the image from the client side, the system uses the **/api/predict** route. The image is saved in the specified path and then it is sent to the **output_prediction** method as an argument, which oversees returning the prediction result for the image. Finally, the method returns a json object containing the server response and the request status.

Method 2 (Making a POST request to Hugging Face API):

In this method, we are making a POST request to the Hugging Face API using Fetch from the frontend (Camera.js) file and sending the picture with API request to the Hugging Face where our Transformer is deployed. The API call is returning us an object with Disease Label and Confidence.

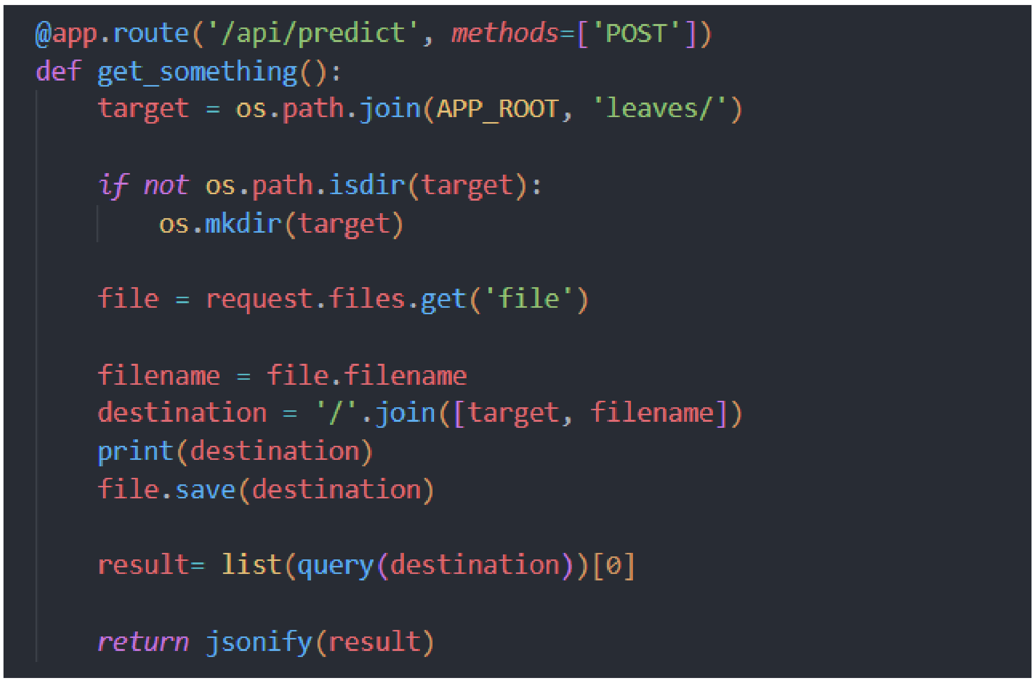


```javascript
const response = await fetch(
  "https://api-inference.huggingface.co/models/plantdoctor/swin-tiny-patch4-window7-224-plant-doctor",
  {
    headers: {
      Authorization: "Bearer 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",
    },
    method: "POST",
    body: image,
  }
);
```

Figure 6.2: Hugging Face API for disease prediction using our transformer

Method 3 (Using Flask backend for Hugging Face API):

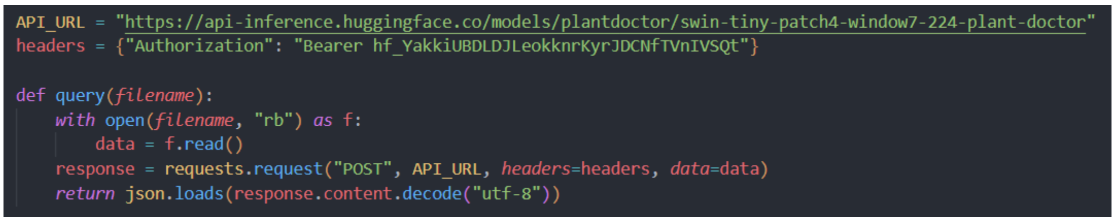
In this third method, we are using flask backend to make an API call to Hugging Face API.



```python
@app.route('/api/predict', methods=['POST'])
def get_something():
    target = os.path.join(APP_ROOT, 'leaves/')

    if not os.path.isdir(target):
        os.mkdir(target)

    file = request.files.get('file')

    filename = file.filename
    destination = '/'.join([target, filename])
    print(destination)
    file.save(destination)

    result= list(query(destination))[0]

    return jsonify(result)
```

Figure 6.3: Disease prediction (get_something) function

When user uploads a photo then frontend makes an API call to Backend and in the backend **get_something** function method calls to **query** function which calls Hugging face API (where our model is deployed) for the prediction result and returns the response in json format as shown in Fig 6.3.



```python
API_URL = "https://api-inference.huggingface.co/models/plantdoctor/swin-tiny-patch4-window7-224-plant-doctor"
headers = {"Authorization": "Bearer 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"}

def query(filename):
    with open(filename, "rb") as f:
        data = f.read()
    response = requests.request("POST", API_URL, headers=headers, data=data)
    return json.loads(response.content.decode("utf-8"))
```

Figure 6.4: Query function for Hugging face API call

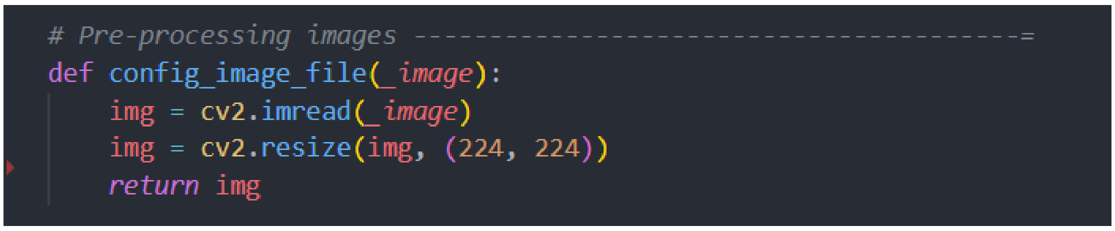
Selected Method

The method we selected for our application is method no 3 because with the first method when we deployed our backend on Heroku, it exceeds the limit of their free storage of 500Mbs only and the size of our backend was 700Mbs, due of the size of our model and libraries.

With the method no 2, the Fetch API was working fine on iOS device but not on android device. We tried to use Axios, but it was not working either so, we had to drop this option too.

The reason why we used method 3 because in this, we removed all the heavy libraries which reduced the size on Heroku because the Model was already deployed on Hugging face and what we needed to do was just to make a REST API call. That is why, we used Flask backend as a middleware for the REST API call to Hugging face and deployed our Flask backend and it is working fine now on both android and iOS device.

Resizing image file for CNN



```python
# Pre-processing images ----------------------------------------=
def config_image_file(_image):
    img = cv2.imread(_image)
    img = cv2.resize(img, (224, 224))
    return img
```

Figure 6.5: Config image file function

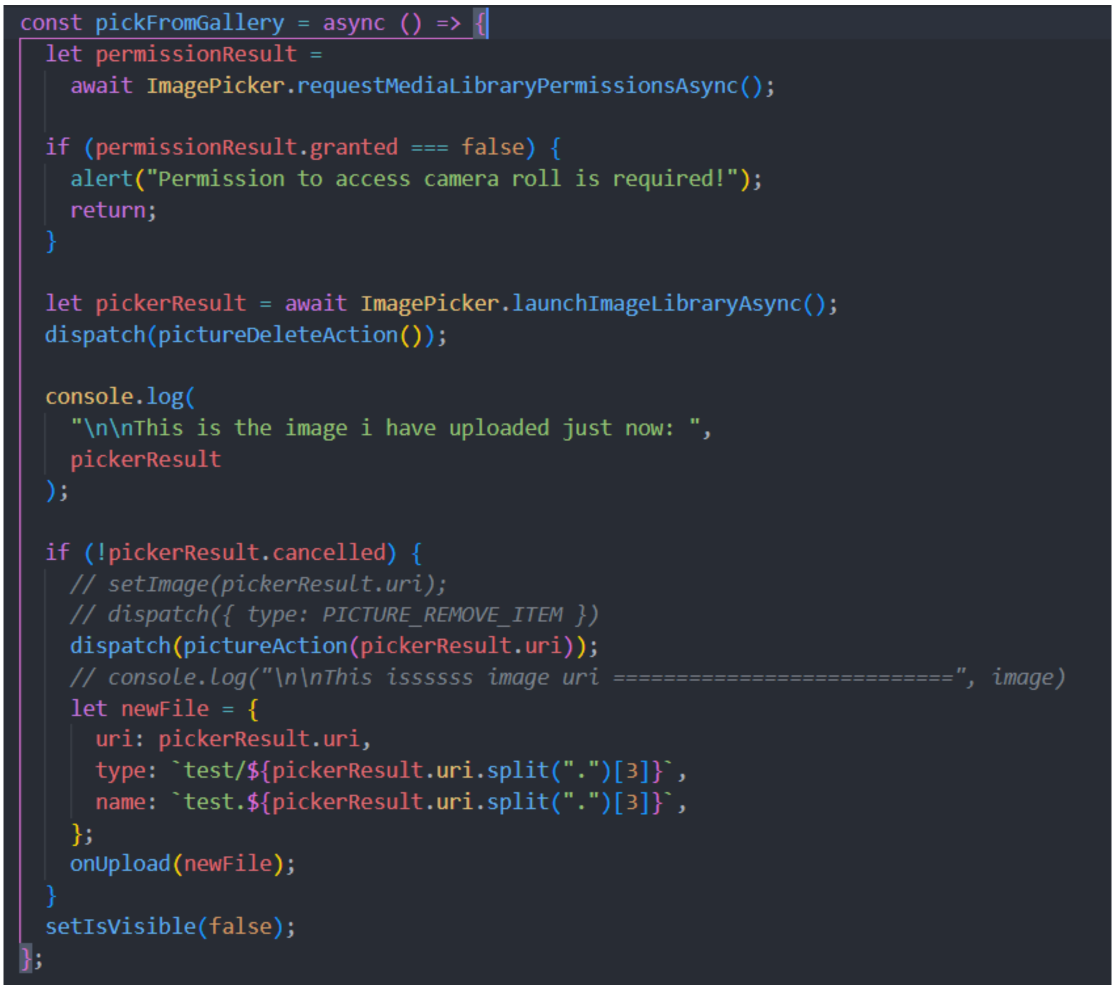
When an image is uploaded to get a prediction, the pre-processing step is handled by the **config_image_file** method. The given image is reshaped, resized, and rescaled using the OpenCV library (discussed earlier) before being fed to the model to obtain the prediction. The model is expecting to process a 224 x 224 shaped RBG image. Finally, the image array is returned by the method.

6.2. Implementation of React Native Mobile application– Visual studio

React Native, a modern cross-platform mobile application development framework, is used to create the client-side mobile application. A user can upload images to the server using one of two methods: selecting from a gallery or taking a picture with the device camera and uploading it to the server.

Selecting an image from gallery function

The approach is used to control how photographs from a gallery are chosen. The component appears and opens the device gallery when the method is called, allowing the user to select and upload an image.



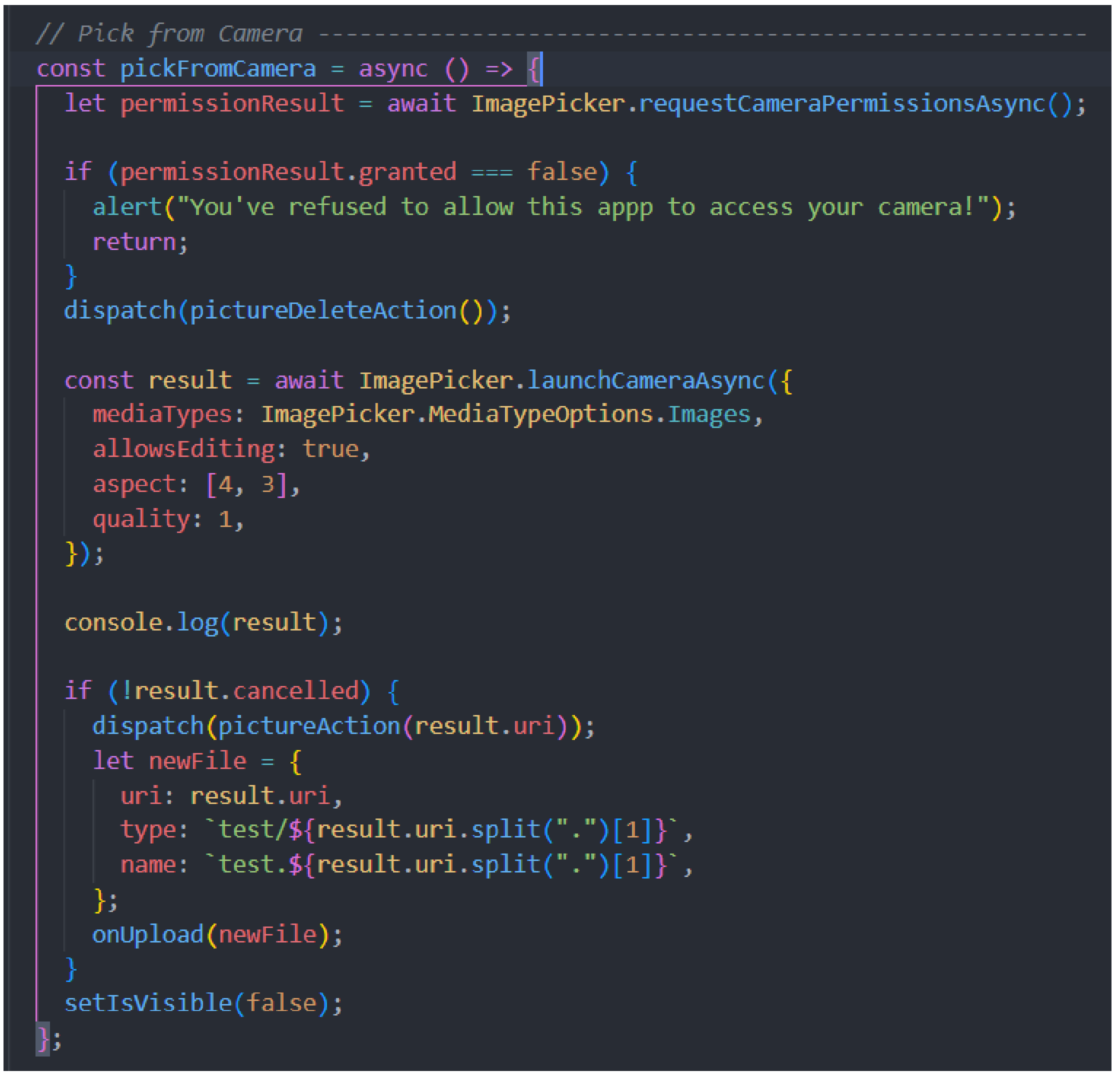
```
const pickFromGallery = async () => {
  let permissionResult =
    await ImagePicker.requestMediaLibraryPermissionsAsync();

  if (permissionResult.granted === false) {
    alert("Permission to access camera roll is required!");
    return;
  }

  let pickerResult = await ImagePicker.launchImageLibraryAsync();
  dispatch(pictureDeleteAction());

  console.log(
    "\n\nThis is the image i have uploaded just now: ",
    pickerResult
  );

  if (!pickerResult.cancelled) {
    // setImage(pickerResult.uri);
    // dispatch({ type: PICTURE_REMOVE_ITEM })
    dispatch(pictureAction(pickerResult.uri));
    // console.log("\n\nThis issssss image uri ===========================", image)
    let newFile = {
      uri: pickerResult.uri,
      type: `test/${pickerResult.uri.split(".")[3]}`,
      name: `test.${pickerResult.uri.split(".")[3]}`,
    };
    onUpload(newFile);
  }
  setIsVisible(false);
};
```

Figure 6.6: pickFromGallery function

The asynchronous **pickFromGallery** method checks the user to see if the application has the necessary permissions to use the device camera and gallery. If the permissions are granted then method proceeds to show the device gallery. **ImagePicker** has several properties that control how the image picker behaves.

Pick From Camera Option

PickFromCamera which is a fat arrow asynchronous method allows the user to capture an image from the camera and upload it to the server in real time. We have used expo camera to fulfil the image capturing functionality.



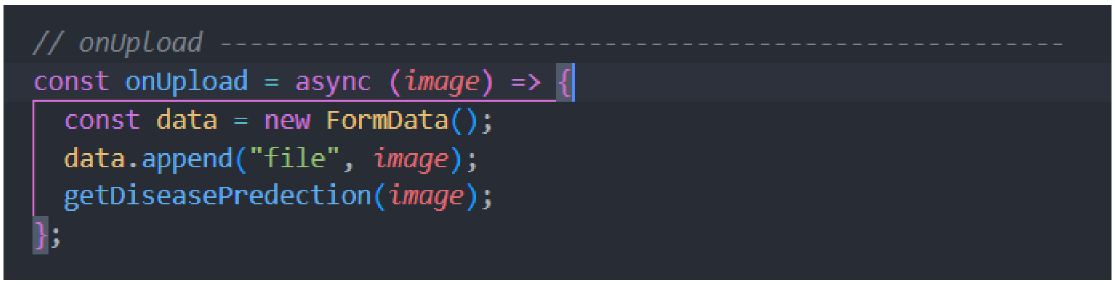
```
// Pick from Camera -----------------------------------------------------
const pickFromCamera = async () => {
  let permissionResult = await ImagePicker.requestCameraPermissionsAsync();

  if (permissionResult.granted === false) {
    alert("You've refused to allow this appp to access your camera!");
    return;
  }
  dispatch(pictureDeleteAction());

  const result = await ImagePicker.launchCameraAsync({
    mediaTypes: ImagePicker.MediaTypeOptions.Images,
    allowsEditing: true,
    aspect: [4, 3],
    quality: 1,
  });

  console.log(result);

  if (!result.cancelled) {
    dispatch(pictureAction(result.uri));
    let newFile = {
      uri: result.uri,
      type: `test/${result.uri.split(".")[1]}`,
      name: `test.${result.uri.split(".")[1]}`,
    };
    onUpload(newFile);
  }
  setIsVisible(false);
};
```

Figure 6.7: pickFromCamera function

The method first checks the user permissions on the device and notifies the user if the permissions are denied before opening the device camera.

The method is an Asynchronous method that asynchronously handles internal functionality. The **launchCameraAsync** method's ImagePicker is in charge of opening the device camera and capturing the image. The taken image is subsequently transmitted to the server via a **POST** request, and the application's image uploading is handled by the **onUpload** method.

On image upload



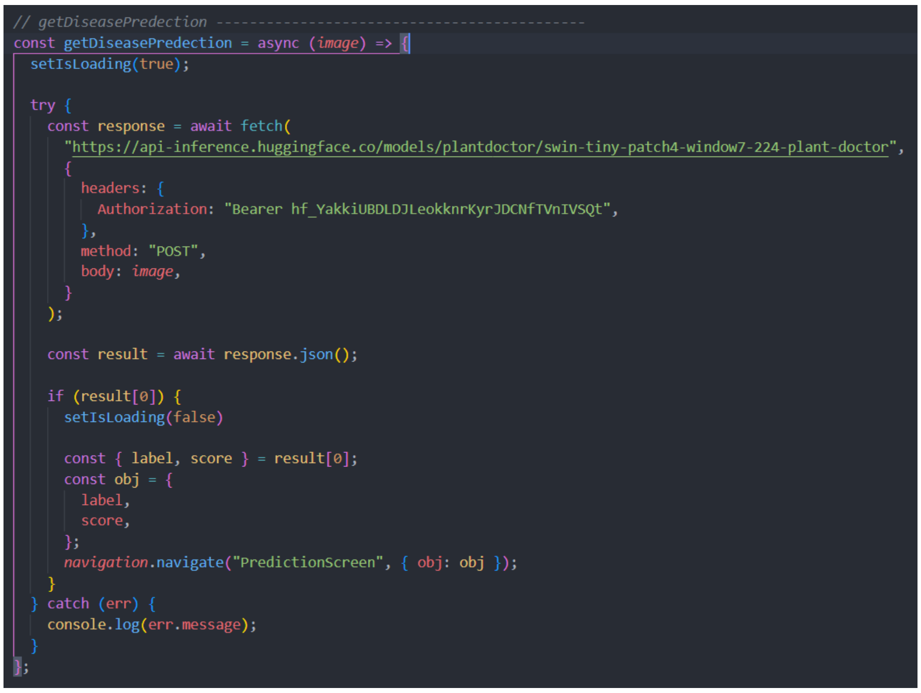
```javascript
// onUpload --------------------------------------------------------------
const onUpload = async (image) => {
  const data = new FormData();
  data.append("file", image);
  getDiseasePredection(image);
};
```

Figure 6.8: onUpload method

This method receives an image and then initialises the FormData() object so that the selected image is converted and assigned to a formData typed object for transmission to the Flask server.

Get Prediction

- #### Using Hugging Face API

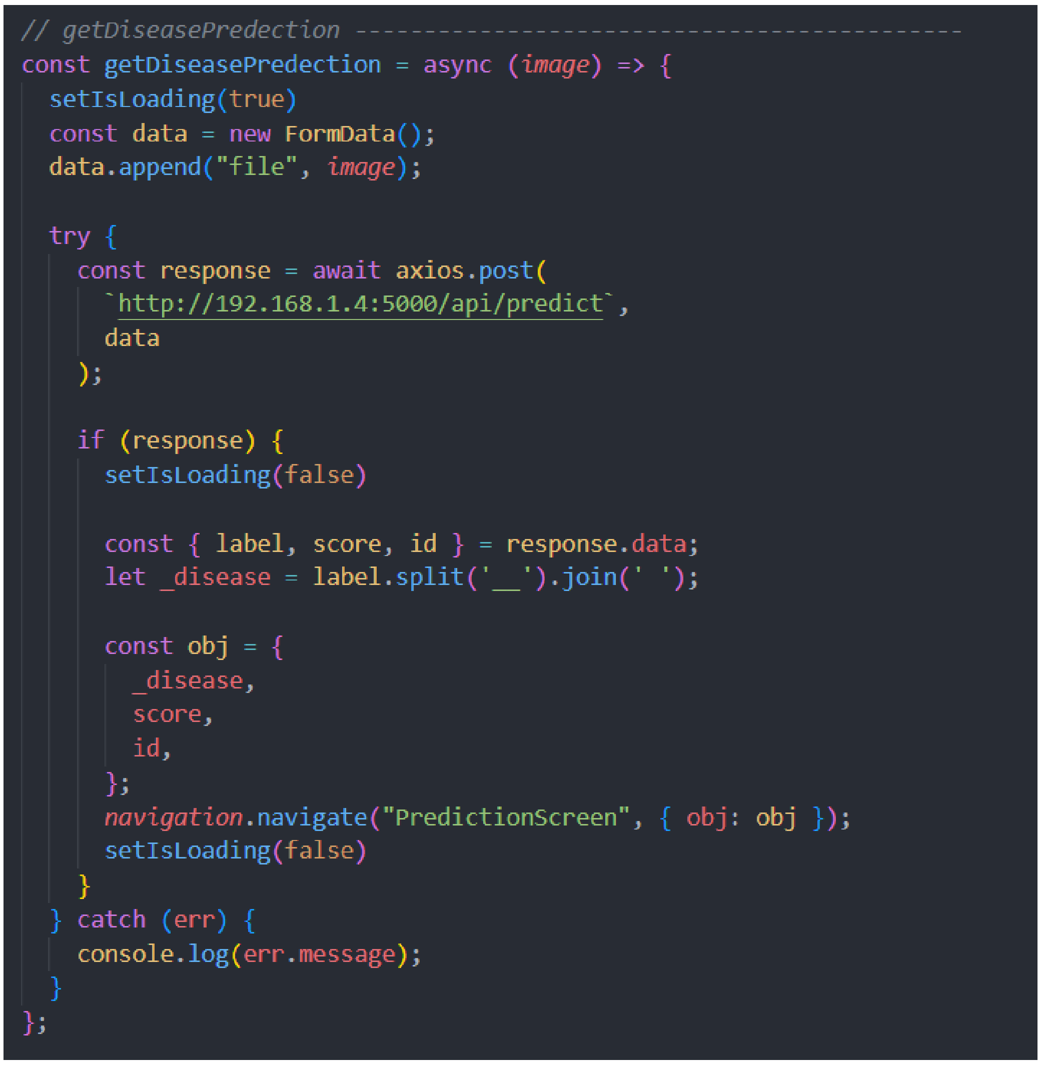


```javascript
// getDiseasePredection ------------------------------------------
const getDiseasePredection = async (image) => {
  setIsLoading(true);

  try {
    const response = await fetch(
      "https://api-inference.huggingface.co/models/plantdoctor/swin-tiny-patch4-window7-224-plant-doctor",
      {
        headers: {
          Authorization: "Bearer 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",
        },
        method: "POST",
        body: image,
      }
    );

    const result = await response.json();

    if (result[0]) {
      setIsLoading(false)

      const { label, score } = result[0];
      const obj = {
        label,
        score,
      };
      navigation.navigate("PredictionScreen", { obj: obj });
    }
  } catch (err) {
    console.log(err.message);
  }
};
```

Figure 6.9: Hugging Face API

In this method, a request will be sent to Hugging Face API with authorization Token, an Image and the API returns an object with label and score that will be sent to **PredictionScreen** for showing the results to the user. If a user encounters a network or server error, the user will be notified via an alert.

- **Using Flask Server**



```
// getDiseasePredection -------------------------------------------
const getDiseasePredection = async (image) => {
  setIsLoading(true)
  const data = new FormData();
  data.append("file", image);

  try {
    const response = await axios.post(
      `http://192.168.1.4:5000/api/predict`,
      data
    );

    if (response) {
      setIsLoading(false)

      const { label, score, id } = response.data;
      let _disease = label.split('__').join(' ');

      const obj = {
        _disease,
        score,
        id,
      };
      navigation.navigate("PredictionScreen", { obj: obj });
      setIsLoading(false)
    }
  } catch (err) {
    console.log(err.message);
  }
};
```

Figure 6.10: Method for API call to Flask

The function **getDiseasePrediction** accepts the selected image as an input and sends it via api/predict, to the server-side application, which returns three responses: _disease, confidence, and the id, which is then delivered to the **predictionScreen** for the user to see. A user will be notified via an alert if there is a network or server fault.

Chapter 7

System Testing & Evaluation

Each test method is intended to detect a specific sort of product failure. All test types, however, are designed to achieve the same goal: "early detection of all problems before the product is issued to the consumer."

7.1. Test Plan

Prior to the application's publication, the following types of testing should be carried out.

- Unit test – Unit testing is essential as the system will be built in division of the functionality. So, to test each part of the application is mandatory.
- API test – Test the API which is built at the backend before integrating with the frontend.
- Integration testing – Testing of all the individual components as a single component.
- System test – Performed on a fully integrated platform to ensure that the system meets its criteria.

7.2. Test Environment

By breaking down plant leaves, the application hopes to identify many types of plant leaf diseases. Because the software is designed to run on a phone, the test environment will be a phone with a camera and a working internet connection.

7.3. Test Data

Diseased plant leaf images are the test data for the testing.
This could be images of recently collected diseased plant leaf images or images captured by the testing system.

7.4. Testing Techniques

Following are the testing techniques that will be used for the testing of this system.

- Functional testing

- Performance testing
- Validation testing
- Accuracy testing

The application user interface (UI) is used for functional testing, while the machine learning (ML) component of the system is used for validation, accuracy, and performance testing. Validation testing for the system looks at how the validation dataset helps train the model, as well as comparing validation loss to training loss and validation accuracy to training accuracy. The purpose of accuracy testing was to evaluate the parameters and choose the optimal training model for the predicted system needs. The system uses a confusion matrix to check the accuracy of the testing data. The training model's performance was checked, as well as how it predicts the correct output when using the model.

7.5. Test Cases

Following are the test cases given below:

Test cases = TC

7.5.1. Test Case no.1

Test case description	Convert an image into an array.
Pre-condition	It is necessary to upload an image of a diseased plant leaf. It is necessary to resize the image.

TC no	Action	Test inputs	Expected outcome	Actual outcome	Pass/ Fail
TC-1	1. Image of a leaf to be uploaded. 2. Changing the received image to an array.	• Image	An array should be received representing the input image.	Image is converted into an array format.	Pass

Table 7.1: Converting an input image to array use-case

7.5.2. Test Case no.2

Test case description	For the prediction, choose the most accurate possibility value from each class.
Pre-condition	Image should be uploaded after converted into an array.

TC no	Action	Test inputs	Expected outcome	Actual outcome	Pass/Fail
TC-2	For each class, choose the highest possible value.	Set of possible values for every class.	Highest possible value.	The highest possible value was viewed.	Pass

Table 7.2: Choosing the most accurate possible value

7.5.3. Test Case no.3

Test case description	Checking the final prediction result for a given image
Pre-condition	From the array of alternatives for each class, find the most possible value for a given image.

TC no	Action	Test inputs	Expected outcome	Actual outcome	Pass/ Fail
TC-3	1. Choose the highest value for the array index. 2. From the json object array, map the label. 3. Return the result.	Set of possible values for every class.	View the final result for a specific plant leaf image.	The confidence value is displayed alongside the predicted class.	Pass

Table 7.3: Checking the final prediction result for the given image

7.6. Functional Test Evaluation

Following are the test cases for the functionality evaluation.

7.6.1. Test Case no.1

Test case description	Capturing image functionality
Pre-condition	• Android/IOS device. • User must be logged in to the system.

TC no	Action	Inputs	Expected outcome	Actual outcome	Pass/ Fail
TC-1	1. Select the camera option. 2. Take a picture. 3. Click 'select' button.	• Select or touch the camera icon. • Image	The captured image should appear on the screen, ready to be edited or submitted.	Image from the camera should display on the screen.	Pass

Table 7.4: Testing capturing image functionality

7.6.2. Test Case no.2

Test case description	Test for selecting image from the gallery.
Pre-condition	• Android/IOS mobile phone. • User must be logged in to the system.

TC no	Action	Inputs	Expected outcome	Actual outcome	Pass/ Fail
TC-2	Select the 'gallery' icon.	• Select an image from the gallery by clicking on gallery icon. • Image	On the screen, the chosen image should show, ready to be altered or uploaded.	Image from the gallery is displaying on the screen.	Pass

Table 7.5: Testing selecting image from the gallery

7.6.3. Test Case no.3

Test case description	Testing image upload functionality
Pre-condition	• The application is installed on an Android/IOS phone. • The user must be logged in to the system.

TC no	Action	Test inputs	Expected outcome	Actual outcome	Pass/ Fail
TC-3	1. Choose an image from the gallery or take a photo with your camera. 2. Click the confirm button.	• Click the confirm button • Image	Until the upload is complete, a spinner should be visible. The reaction should be obvious.	There was no spinner visible, and the result was displayed directly.	Fail

Table 7.6: Testing image upload functionality

7.6.4. Test Case no.4

Test case description	Testing result view functionality
Pre-condition	• The application is installed on an Android/IOS phone. • User must be logged in to the system. • There should be a chosen image.

TC no	Action	Test inputs	Expected outcome	Actual outcome	Pass/ Fail
TC-004	1. After selecting an image, click the 'Confirm' button.	• Click the confirm button • Image	An image of the predicted plant disease should appear, along with the disease's name, level of confidence, and a brief description.	An image of the predicted plant disease should appear, along with the disease's name, level of confidence, and a brief description.	Pass

Table 7.7: Testing result view functionality

Chapter 8

Conclusion

The report emphasised the plan and implementation of a deep learning-based cross-platform mobile application that analyses plant leaf images to detect plant diseases. Image processing techniques, artificial intelligence, traditional machine learning techniques, deep learning-based approaches, and case studies of similar systems were considered to ensure satisfactory performance. Deep learning, which can handle automatic feature extraction, is used to process feature extraction from the leaves of diseased plants. Each technology chosen for each task in this project has been justified, along with supporting evidence.

We achieved 98.36% accuracy for the deep learning model as we transitioned from CNNs to Transformers over the course of development, for which the results were satisfactory for validation data and real time input. Many farmers and gardeners can benefit from a server-based cross-platform smartphone application that can recognise plant diseases by analysing plant leaves. This will assist people who do not have any knowledge of crop diseases in gaining an understanding of the plant disease spectrums.

8.1. Future work

PlantDoctor is scalable for the functionality or features we would add in future. A project always has a space for the extra functionality when it evolves. Following are the features we would be adding in future:

- Improving multi-lingual feature (Especially Urdu)
- User friendly interface for non-technical users
- More accurate model
- Add more disease classes and improve the accuracy of existing classes.

Chapter 9

REFERENCES

[1] F. a. A. Organization, “Trends and challenges,” [Online]. Available: https://www.fao.org/3/i6583e/i6583e.pdf. [Accessed 15 10 2021].

[2] L. N. B. Algorithm, “Naive Bayes Classifier,” [Online]. Available: https://www.analyticsvidhya.com/blog/2017/09/naive-bayes-explained/. [Accessed 18 2 2022].

[3] GeeksforGeeks, “Naive Bayes Classifiers,” [Online]. Available: https://www.geeksforgeeks.org/naive-bayes-classifiers/. [Accessed 8 2 2022].

[4] A. Pai. [Online]. Available: https://www.analyticsvidhya.com/blog/author/aravindpai/. [Accessed 17 1 2022].

[5] A. N. Network, “ANN,” [Online]. Available: https://medium.com/@dhea.larasati326/artificial-neural-network-55797915f14a.

[6] KamleshGolhani, “Convolution Neural Networks,” [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2214317317301774.

[7] CNN, “Feature is extracted with the help of CNN,” [Online]. Available: https://cs231n.github.io/convolutional-networks/.

[8] D. l. v. m. learning. [Online]. [Accessed 16 1 2022].

[9] R. n. f. architecture. [Online]. Available: https://www.educba.com/react-native-architecture/. [Accessed 12 2 2022].

[10] S. P. M. e. Al., PlantVillage, [Online]. Available: https://www.kaggle.com/datasets/abdallahalidev/plantvillage-dataset. [Accessed 24 12 2021].

[11] P. Dataset. [Online]. Available: https://www.kaggle.com/datasets/abdallahalidev/plantvillage-dataset.

[12] V. transformer. [Online]. Available: https://paperswithcode.com/method/vision-transformer. [Accessed 2 4 2022].

[13] T. g. structure. [Online]. Available: https://paperswithcode.com/method/transformer. [Accessed 2 4 2022].

FYP

ORIGINALITY REPORT

15%	5%	4%	12%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to Asia Pacific Instutute of Information Technology Student Paper	7%
2	Akibur Rahman Prodeep, A S M Morshedul Hoque, Md. Mohsin Kabir, Md. Saifur Rahman, M.F. Mridha. "Plant Disease Identification from Leaf Images using Deep CNN's EfficientNet", 2022 International Conference on Decision Aid Sciences and Applications (DASA), 2022 Publication	1%
3	arxiv.org Internet Source	<1%
4	Submitted to Swinburne University of Technology Student Paper	<1%
5	paperswithcode.com Internet Source	<1%
6	Abir Rahali, Moulay A. Akhloufi. " MalBERT: Malware Detection using Bidirectional Encoder Representations from Transformers	<1%

[illegible]

Engr. Dr. Adeel M. Syed
PhD Software Engineering
Senior Assistant Professor
Bahria University, Islamabad