A Major Project Final Report on

Plant Leaf Disease Detection

Submitted in Partial Fulfillment of the Requirements for the

Degree of **Bachelors of Engineering in Information Technology**

Under Pokhara University

Submitted by

Mahanand Mandal, 191519

Sanju kumar Chaudhary, 191539

Date:

3rd July, 2024

Department of Information Technology

NEPAL COLLEGE OF

<u>INFORMATION TECHNOLOGY</u>

Balkumari, Lalitpur, Nepal

Abstract

Plant leaf disease poses significant threats of agriculture productivity and food security. Early detection and diagnosis of these disease are crucial for effective management and crops production. In recent years, computer vision and machine learning techniques have emerged as promising tools for automated plant disease detection. This project present a study on the development and implementation of plant leaf disease detection system using deep learning. The proposed system utilizes convolutional neural networks (CNN) to automatically analyze digital images of plant leaves and classify them into healthy or diseased categories based on visible symptoms. A dataset comprising images various plants species afflicted with common disease was curated and used to train the CNN model. The model was trained using transfer learning on pre-defined network to leverage its feature extraction capabilities and optimize performance with limited data. Experimental results demonstrate the efficacy of the developed system in accurately identifying and classifying plant leaf diseases. The model achieved a high accuracy rate in distinguishing between healthy and diseased leaves across different plant types and disease categories. Furthermore, the systems performance was evaluated in terms of sensitivity, specificity, and overall precision, showcasing its potential for practical application in agricultural settings. This research contributes to the field of precision agriculture by offering a robust and scalable solution for early disease detection in plants, thereby enabling timely interventions and improved crop yield management. Future work will focus on extending the system to broader range of plant diseases, optimizing real-time processing capabilities, and integrating the technology into user-friendly agricultural tools for farmers and growers.

Contents

1. Introduction……………………………………………………………………………..1

1.1. Problem Statement……………………………………………………………….2

1.2. Project Objective…………………………………………………………………2

1.3. Significance of Study…………………………………………………………….2

2. Literature Review…………………………………………………………………3

2.1. Supervised Machine Learning…………………………………………………...5

2.2. Convolution Neural Network…………………………………………………….6

2.2.1. Layers in CNN…………………………………………………………………7

2.2.1.1. Input layer……………………………………………………………….....7

2.2.1.2. Convo Layer……………………………………………………………………7

2.2.1.3. Pooling Layer…………………………………………………………………...7

2.2.1.4. Fully Connected layer…………………………………………………………...8

2.2.1.5. Soft-max/Logistic Layer………………………………………………………...8

2.2.1.6. Output Layer……………………………………………………………………8

2.3. Back-propagation…………………………………………………………………9

2.4. Activation function……………………………………………………………….12

2.4.1. ReLu (Rectified Linear Unit)…………………………………………………12

3. Methodology………………………………………………………………………..14

3.1. Introduction…………………………………………………………………….....14

3.2. Dataset……………………………………………………………………………...14

3.3. Training the model………………………………………………………………….15

3.4. Software Development Lifecycle…………………………………………………..17

3.4.1. Require analysis…………………………………………………………………...17

3.4.2. Design and Development………………………………………………………….17

3.4.3. Testing……………………………………………………………………………..17

3.4.4. Implementation……………………………………………………………………17

3.5. Hardware and Software Required………………………………………………….18

3.5.1. Hardware……………………………………………………………………………..18

3.5.1.1. Mobile Device…………………………………………………………………...18

3.5.1.2. Farm……………………………………………………………………………..18

3.5.2. Software……………………………………………………………………………18

3.5.2.1. Artificial Intelligence……………………………………………………………18

3.5.2.2. Python…………………………………………………………………………...18

4. Block-Diagram……………………………………………………………………………...19

4.1. Pre-processing……………………………………………………………………………..20

4.1.1. Background Removal……………………………………………………………..20

4.1.2. Sharpen Image…………………………………………………………………….20

4.1.2. Remove Noise…………………………………………………………………….20

4.1.4. RGB to HSV………………………………………………………………………20

4.1.5. Segmentation (Fuzzy C-mean Clustering)………………………………………..20

4.1.6. Feature Extraction………………………………………………………………...20

4.1.7. Classification……………………………………………………………………...20

5. Flowchart……………………………………………………………………………………21

6. Use Case Diagram…………………………………………………………………………..22

7. Sequence Diagram…………………………………………………………………………..23

8. Activity Diagram……………………………………………………………………………24

9. Gantt-Chart………………………………………………………………………………….25

10. Project Deliverables………………………………………………………………………..27

11. Project Task and Time Scheduling………………………………………………………..28

12. Conclusion…………………………………………………………………………………29

13. References…………………………………………………………………………………30

.

List of Figures

Figure1.1: A CNN Sequence to classify handwriting digits……………………………………..6

Figure1.2: Layers in CNN…………………………………………………………………...7

Figure1.3: Tomato-Bacterial-spot…………………………………………………………..15

Figure1.4: Tomato-Early-blight……………………………………………………….……15

Figure1.5: Tomato-healthy……………………………………………………………………15

Figure1.6: 20 Epoch Train data…………………………………………………………..15

Figure1.7: 4 Epoch Train data………………………………………………………….....16

Figure1.8: 50 Epoch Train data…………………………………………………………..16

Figure1.9: Incremental Model Design………………………………………………………17

Figure1.10: Block Diagram……………………………………………………………………19

Figure1.11: Flowchart…………………………………………………………………………21

Figure1.12: Use Case Diagram……………………………………………………………...22

Figure1.13: Sequence Diagram……………………………………………………………..23

Figure1.14: Activity Diagram………………………………………………………………24

Figure1.15: Increment-I…………………………………………………………………….25

Figure1.16: Increment-II………………………………………………………………….....26

Figure1.17: Increment-II…………………………………………………………………….27

1. Introduction

In Nepal about 66% of the population relies on agriculture. Identification of the plant disease is important in order to prevent the losses within the yield. It`s terribly troublesome to observe the plant disease manually. It needs tremendous quantity of labor, expertize within the plant disease, and conjointly need the excessive time interval. Hence, image processing and machine learning model can be employed for the detection of plant disease. In this project, we have described the technique for the detection of plant disease with the help of their leaves pictures. Image processing is a branch of signal processing which can extract the image properties or useful information from image. Machine learning is a sub part of artificial intelligence which works automatically or give instructions to do a particular task. The main aim of machine learning is to understand the training data and fit the training data into models that should be useful to the people. So it can assist in good decisions making and predicting the correct output using the large amount of training data. The color of leaves, amount of damage to leaves, area of the leaf, texture parameters are used for classification. In thus project we have analyzed different image parameters or features to identifying different plant leaves disease to achieve the best accuracy. Previously plant disease detection is done by visual inspection of the leaves or some chemical processes by experts. For doing so, a large team of experts as well as continuous observation of plant is needed, which costs high when we do with large farms. In such conditions, the recommended system proves to be helpful in monitoring large fields of crops. Automatic detection of the disease by simply seeing the symptoms on the plant leaves makes it easier as well as cheaper. The proposed solution for plant disease detection is computationally less expensive and requires less time for prediction than other deep learning based approaches since it uses statistical machine learning and image processing algorithm.

1.1. Problem statement

The prevalence of plant disease poses a significant challenge to global agriculture, impacting crop yield, quality, and food security. Timely and accurate detection of plant disease is crucial for implementing targeted disease management strategies and minimizing yields losses. However, manual disease identification methods are often labor-intensive, subjective, and prone to human error. This problem statement outlines the key challenges of plant disease detection project, emphasizing the need for automated, accurate, and accessible disease identification tools in agriculture. The adjustment can be made based on the specific scope, resources, and objective of the project.

1.2. Project Objectives

This project involves both the studying of the existing system and combining the methods of the existing systems in such a way that they give an optimal solution. The objective of the project is to

- Come up with the system for conformation of the disease present in the plant leaves.
- Develop tools for the plant leaf disease detection.
- To develop a user-friendly Web application for Plant leaf disease detection.

1.3. Significance of study

The study helps in the development of the system that detects the disease present in the leaf. It provide accurate information to the users regarding plants leaf. It also helps in making the system dynamic as per the requirement of the user as the data about the plant leaf disease. Finally, the study also looks into other similar kinds of system in use to explore the benefits of the system as well as its drawbacks.

2. Literature Review

Shiroop Madiwalar and Medha Wyawahare analyzed different image processing approaches for plant disease detection in their research [1]. Authors analyzed the color and texture features for the detection of plant disease. They have experimented their algorithms on the dataset of 110 RGB images. The features extracted for classification were mean and standard deviation of RGB and YCbCr channels, grey level occurrence matrix (GLCM) features, the mean and standard deviation of the image convolved with Gabor filter. Support vector machine classifier was used for classification. Authors concluded that GCLM features are effective to detect normal leaves. Whereas color features and Gabor filter features are considered as best for detecting anthracnose affected leaves and leaf spot respectively. They have achieved highest accuracy of 83.34% using all the extracted features.

In 2015, S. Khirade et Al. tackled the problem of plant disease detection using digital image processing techniques and back propagation neural network (BPNN) [2]. Authors have elaborated different techniques for the detection of plant disease using the images of leaves. They have implemented Otsu's thresholding followed by boundary detection and spot detection algorithm to segment the infected part in leaf. After that they have extracted the features such as color, texture, morphology, edges etc. for classification of plant disease. BPNN is used for classification i.e. to detect the plant disease.

Garima Shrestha et Al. deployed the convolutional neural network to detect the plant disease [3]. Authors have successfully classified 12 plant diseases with 88.80% accuracy. The dataset of 3000 high resolution RGB images were used for experimentation. The network has 3 blocks of convolution and pooling layers. This makes the network computationally expensive. Also the F1 score of the model is 0.12 which is very low because of higher number of false negative predictions.

Tomatoes are the third most widely cultivated crop, following Potato and Sweet Potato. As result of various types of diseases, both the quality and quantity of tomato harvests decline. Here, we discussed the deep learning based approach to dis-ease prevention and productivity improvement. Transfer learning is used to minimizethe amount of training data, the amount of time, and the computational costs associated with building deep learning, according to Gayakwad et. al. [4]

use a deep convolutional neural network to identify Ąve diseases. The PlantVillage dataset contained 9,000 images of tomato leaves, which included speci-mens of Ąve different tomato diseases, including Bacterial Spot, Early Blight, SeptoriaLeaf Spot, Leaf Mold, and Yellow Leaf Curl Virus, as well as healthy leaves. On aheld-out test set, their method achieved 99.84 percent accuracy, proving its viability.

Ashqar et al. [5]

In a separate study [6], authors employed a DCNN model. Using images of 14,903 diseased and healthy plant leaves from the Plant Village dataset, they trained their model to identify the type of leaf. In the performance test, the trained modelachieved an accuracy of 99.25%. Geetharamani et. al. [7]present a a nine-layer con-volutional neural network approach for plant leaf disease identifacation utilizing adeep convolutional neural network (Deep CNN). The proposed model is trained ona comprehensive open dataset consisting of 39 distinct classes of plant leaves and background images. By comparing it with popular transfer learning techniques, theauthors demonstrate that their model outperforms them in terms of performance on the validation data. Through extensive simulations, the proposed model achieves an impressive classiĄcation accuracy of 96.46%, surpassing traditional machine learningmethods. To evaluate the performance of their Deep CNN model, the authors conduct a comparative analysis with well-known transfer learning approaches such as AlexNet,VGG16, Inception-v3, and ResNet.

Another study [8] proposed a CNN model for detecting diseases on tomato leaf. Before tomato leaf detection, the dataset is separated. Using transfer learning, a trained model (ResNet-50) is imported and modified based on the classification problem. Data augmentation has been implemented to improve the quality of the ResNet model and produce a result that corresponds as closely as possible to the actual prevalent disease. All of these factors were considered during the development of a PyTorch and deep-CNN-based disease detection model for tomato leaf. After processing the testing dataset, the ResNet 50 modelŠs learned parameters are utilized to validate the results. To classify the data, six of the most prevalent diseases affecting tomato crops were utilized. This model achieved a 97% accuracy rate when the data set was expanded to four times its original size.
Ozbilge et. al. proposed a compact convolutional neural network (CNN) for disease identiĄcation, utilizing a network architecture consisting of only six layers. The compact nature of the network renders it computationally inexpensive in terms of the employed parameters. The implementation of the compact CNN architecture yielded impressive results in disease identiĄcation.It demonstrates an accuracy of 99.70% for the F1 score, 98.49% for the Matthews correlation coefficient, 98.31% for the truepositive rate, and 98.49% for the true negative rate when tested on the unseen images. . Furthermore, the authors claimed that the proposed model outperforms the pre-trained models due to its compact architecture.Some researchers [9] recommend image processing techniques, including image acquisition and segmentation, for detecting leaf diseases in tomatoes. Two technical models, Resnet50 and MobileNet, have been implemented using the transfer learning technique of deep learning. These models have produced favorable outcomes. The results have improved with each model execution step.
The experimental results of the Resnet-50 Model Ćuctuate between 94 percent and 99.81%, whereas the MobileNet predictions correction Ćuctuates between 95.23% and 99.88%. On the other hand, Wang et. al. [10]presented deep convolutional neural networks and object detection models to develop methods for detecting tomato disease.The proposed architecture uses Faster R-CNN and Mask R-CNN are models, where tomato diseases are identified using faster R-CNN, and the locations and shapes of the diseased areas are detected and segmented using mask R-CNN. The suggested tomato disease detection architectures, however, were unable to identify some tomato disease types or infected areas due to a lack of training images or low image resolutions. A study [11] proposed a deep learning-based method for detecting tomato diseases by employing Conditional Generative Adversarial Network (C-GAN), which can produce synthetic images of tomato plant leaves. The DenseNet121 model was trained on synthetic and real images using transfer learning in order to classify images of tomato leaves into ten categories of disease.

2.1. Supervised Machine Learning

Supervised learning, in the context of artificial intelligence (AI) and machine learning, is a type of system in which both input and desired output data are provided. Input and output data are labelled for classification to provide a learning basis for future data processing. The term supervised learning comes from the idea that an algorithm is learning from a training dataset, which can be thought of as the teacher. Supervised machine learning systems provide the learning algorithms with known quantities to support future judgments. Supervised learning systems are mostly associated with retrieval-based AI but they may also be capable of using a generative learning model. The majority of practical machine learning uses supervised learning. Supervised learning is where you have input variables (x) and output variable (y) and you use an algorithm to learn the mapping function from the input to the output **Y=f (X).**The goal is to approximate the mapping function so well that when you have new data (X) that you can predict the output variable (Y) for that data.

2.2. Convolution Neural Network

A convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and bisase) to various aspects/objects in the image and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics.

The architecture of a ConbNet is analogous to that of the connectivity pattern of Neurons in the Human Brain and was inspired by the organization of the Visual Cortex. Individual neurons respond to stimuli only in a restricted region of the visual field known as the Receptive Field. A collection of such field overlap to cover the entire visual area.

In deep learning, a convolutional neural network (CNN, or ConvNet) is a class of deep neural networks, most commonly applied to analyzing visual imagery. They are also known as shift invariant or space invariant artificial neural networks (SIANN), based on their shared-weights architecture and translation invariance characteristics. They have applications in image and video recognition, recommender systems, image classification, media image analysis, and neural language processing.

CNNs are regularized versions of multilayer perceptrons. Multilayer perceptrons usually mean fully connected networks that is each neuron in one layer is connected to all neurons in the next layer. The “fully-connectedness” of these networks makes them prone to over fitting data. Typical ways of regulation include adding some form of magnitude measurement of weights to the loss function. However, CNNs take a different approach towards regularization: they take advantages of the hierarchical pattern in data and assemble more complex patterns using smaller and simpler patterns. Therefore, on the scale of connectedness and complexity, CNNs are on the lower extreme.

Convolutional networks were inspired by biological process in that the connectivity pattern between neurons resembles the organization of the annual visual cortex. Individual cortical neurons respond to stimuli only in restricted region of the visual field known as receptive field. The respective fields of different neurons partially overlap such that they cover the entire visual field.

CNNs use relatively little pre-processing compared to other image classification algorithms. This means that the network learns the filters that in traditional algorithms were hand-engineered. This independence from prior knowledge and human effort in design is a major advantage. A Convolutional neural network (CNN) is a neural network that has one or more convolutional layers and are mainly used for image processing, classification, segmentation and also for other auto correlated data.

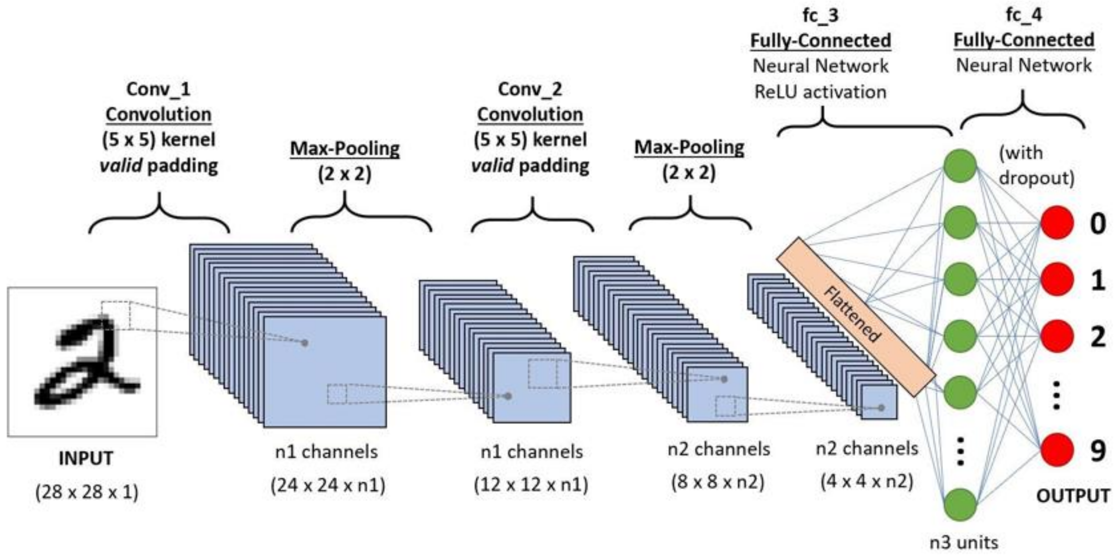
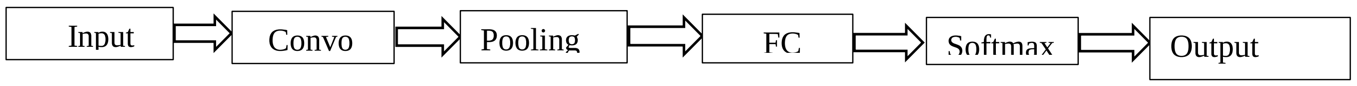


Figure1.1: A CNN sequence to classify handwritten digits

2.2.1. Layers in CNN



Figuire1.2: Layers in CNN

2.1.1.1. Input Layers

Input layer in CNN should contain image data. Image data is represented by three dimensional matrix. We need to reshape it into a single a column.

2.2.1.2. Convo Layer

Convo layer is sometimes called feature extractor layer because features of the image are extracted within this layer. First of all, a part of image is connected to convo layer to perform convolution operation and calculating the dot product between receptive field (it is a local region of the input image that has the same size as that of filter) and the filter. Result of the operation is single integer of the output volume. Then we slide the filter over the next receptive field of the same input image by a Stride and do the same operation again. We have to repeat the same process again and again until we go through the whole image. The output will be the input for the next layer. Convo layer also contains ReLu activation to make all negative value zero.

2.2.1.3. Pooling Layer

Pooling layer is used to reduce the spatial volume of input image after convolution. It is used between two convolution Layer. If FC layer is applied after Convo layer without applying pooling or max pooling, then it will be computationally expensive. So, the max pooling is only way to reduce the spatial volume of input image. Max pooling is applied in single depth slice with Stride of 2. We can observe the 4 x 4 dimension input is reduce to 2 x 2 dimension. There is no parameter in pooling layer but it has two hyper parameters----Filters (F) and Stride (S). In general, if input dimension is W1 x H1 x D1, then

$W2 = (W1-F)/S+1$

$H2=(H1-F)/S+1$

$D2 = D1$

Where W2, H2 and D2 are the width, height and depth of output.

2.2.1.4. Fully Connected Layer (FC)

Fully connected layer involves weights, biases, and neurons. It connects in one layer to neurons in another layer. It is used to classify images between different category by training.

2.2.1.5. Soft-max/Logistic Layer

Soft-max or logistic layer is the last layer of CNN. It resides at the end of FC layer. Logistic is used for classification and soft-max is for multi-classification.

2.2.1.6. Output Layer

Output layer contains the label which is in the form of one-hot encoded.

2.3. Back-propagation

Back-propagation is the essence of neural net training. It is the method of fine-tuning the weight of a neural net based on the error rate obtained in the previous epoch (i.e., iteration). Proper turning of the weights allows you to reduce error rates and to make the model reliable by increasing its generalization.

Back-propagation is a short form for "backward propagation of errors." It is a standard method of training artificial neural networks. This method helps to calculate the gradient of a loss function with respects to all the weights in the network.

Back-propagation repeatedly adjusts the weights of the connections in the network so as to minimize a measure of the difference between the actual output vector of the net and the desired output vector. In other words, back-propagation aims to minimize the cost function by adjusting networks weights and biases. The level of adjustment is determined by the gradients of the cost function with respect to those parameters.

Computing Gradients:

- Gradient of a function C (x_1, x_2, …, x_m) in point x is a vector of the partial derivatives of C in x.

$$\frac{\partial C}{\partial x} = \left[\frac{\partial C}{\partial x_1}, \frac{\partial C}{\partial x_2}, \cdots, \frac{\partial C}{\partial x_m}\right]$$

Equation for derivatives of C in x

- The derivative of a function C measures the sensitivity to change of the function value (output value) with respect to change in its agreement x (input value). In other words, the derivative tells us the direction C is going.

- The gradient shows how much the parameter x needs to change (in positive or negative direction) to minimize c.

Computation of those gradients happens using a technique called chain rule.

For a single weight (w_ jk)^{l} the gradient is:

$$\frac{\partial C}{\partial w_{jk}^{l}} = \frac{\partial C}{\partial z_j^{l}} \frac{\partial z_j^{l}}{\partial w_{jk}^{l}} \qquad \textit{chain rule}$$

$$z_j^l = \sum_{k=1}^{m} w_{jk}^l a_k^{l-1} + b_j^l \qquad by\ defination$$

$$where, m = number\ of\ neurons\ in\ l-1\ layer$$

$$\frac{\partial z_j^l}{\partial w_{jk}^l} = a_k^{l-1} \qquad by\ differentiation\ (calculating\ derivative)$$

$$\frac{\partial C}{\partial w_{jk}^l} = \frac{\partial C}{\partial z_j^l}\ a_k^{l-1} \qquad final\ value$$

Equations for derivative of C in a single weight (w _jk)^l :

Similar set of equation can be applied to (b _J)^l

$$\frac{\partial C}{\partial b_j^l} = \frac{\partial C}{\partial z_j^l}\frac{\partial z_j^l}{\partial b_j^l} \qquad chain\ rule$$

$$\frac{\partial z_j^l}{\partial b_j^l} = 1 \qquad by\ differentiation\ (calculating\ derivative)$$

$$\frac{\partial C}{\partial b_j^l} = \frac{\partial C}{\partial z_j^l}1 \qquad final\ value$$

Equations for derivative of C in a single bias b_j^l :

The common part in both equations is often called “local gradient” and is expressed as follows:

$$\delta_j^l = \frac{\partial C}{\partial z_j^l} \qquad local\ gradient$$

Equation for local gradient

The “local gradient” can easily be determined using the chain rule. I won`t go over the process now but if you have any questions, please comment below.

The gradients allow us to optimize the models parameters:

While (termination condition not met)

$$w := w - \in \frac{\partial C}{\partial w}$$

$$b := b - \in \frac{\partial C}{\partial b}$$

End.

Algorithm for optimizing weights and biases (also called “Gradient descent”)

- Initial values of w and b randomly chosen.
- Epsilon (e) is the learning rate. It determines the gradients influence.
- b are matrix representations of the weights and biases. Derivative of C in w or b can be calculated using partial derivatives of C in the individual weights or biases.
- Termination condition is met once the cost function is minimized.

I would like to dedicate the final part of this section to a simple example in which we will calculate the gradient of C with respect to a single weight w_{22}^2.
Weight w_{22}^2 connects a 2/2 and z_2^2, so computing the gradient requires applying the chain rule through $z_2^3$and $a^3{}_2$:

$$\frac{\partial C}{\partial w_{22}^{(2)}} = \frac{\partial C}{\partial z_2^{(3)}} \cdot \frac{\partial z_2^{(3)}}{\partial w_{22}^{(2)}} = \frac{\partial C}{\partial a_2^{(3)}} \cdot \frac{\partial a_2^{(3)}}{\partial z_2^{(3)}} \cdot a_2^{(2)} = \frac{\partial C}{\partial a_2^{(3)}} \cdot f'\left(z_2^{(3)}\right) . a_2^{(2)}$$

2.4. Activation Function

Activation functions are an extremely important feature of the artificial neural networks. They basically decide whether a neuron should be activated or not. Whether the information that the neuron is receiving is relevant for the given information or information or should it be ignored.

Y = Activation (∑(weight * input) + bias)

The activation () function is the one –linear transformation that we do over the input signal. This transform output is then send to the next layer of neurons as input.

When we do not have the activation function the weights and bias would simply do a linear transformation. A linear equation is simple to solve but is limited in its capacity to solve complex problems. A neutral networks without an activation function is essentially just a linear regressions model. The activation function does the non-linear transformation to the input making it capable to learn and perform more complex tasks like language translations and image classifications. Linear transformations would never be able to perform such tasks. Activation function make the back-propagation possible since the gradients are supplied along with the error to update the weights and biases.

2.4.1. ReLU (Rectified Linear Unit)

Advantages

- Computationally efficient – allows the network to coverage very quickly.
- Non-linear – although it looks like a linear function, ReLU has a derivative function and allows for back-propagation

Disadvantages

- The Dying ReLU problem – when inputs approach zero, or negative, the gradient of the function becomes zero, the network cannot perform back-propagation and cannot learn.

Mathematically it is defined by:

$$\boxed{F(x) = \max(0,x) \left\{ \begin{matrix} \boxed{x_i \text{ if } x_i > 0} \\ \boxed{0 \text{ if } x_i < 0} \end{matrix} \right\}}$$

3. Methodology

3.1. Introduction

There are two main parts of the system. The first part is client which is built from …… Its interface is used to upload the image and see the predicted result. The last part is API which built from flask which runs independently. Initially, the client inputs the plant image into the client part which identify the data from the dataset and passes the data to the flask API. Then the API will analyze the image and predict the output. In the API, the trained plant disease classifier model is loaded with its target size, the default target size for the model is 128*128. This string encoded image is decoded into jpeg format. The decoded image is passed to pre-process image where the image is resized and formatted. Finally, image is passed to production algorithm. The prediction returns the output of the plant disease present in the plant leaf.

We use activation function ReLU for training. During training, we run the model to 50 epochs. After training the model to 20 epochs we got, loss: 0.0703, accuracy: o.9743 -val loss: 0.0493 – val acc: 1.0000.

3.2. Dataset

The dataset is taken from Kaggle. The data set is organized into three folders (train, test, val) and contains subfolders for each image category.

Put here the data set link from kaggle.

- Tomato___Bacterial_spot
- Tomato___Early_blight
- Tomato___healthy
- Tomato___Late_blight
- Tomato___Leaf_Mold
- Tomato___Septoria_leaf_spot
- Tomato___Spider_mites Two-spotted_spider_mite
- Tomato___Target_Spot,
- Tomato___Tomato_mosaic_virus
- Tomato___Tomato_Yellow_Leaf_Curl_Virus

https://www.kaggle.com/datasets/kaustubhb999/tomatoleaf

Dataset photo

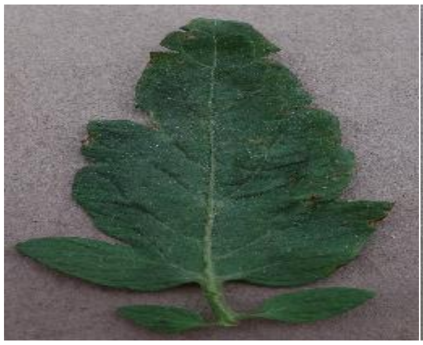
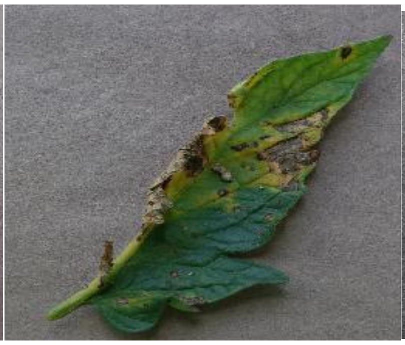
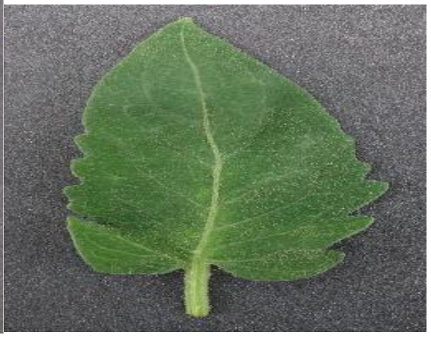




Figure1.3: Tomato-Bacterial-spot   Figure1.4: Tomato-Early-blight   Figure1.5: Tomato-healthy

3.3. Training the model

There are various ways to train the model. We use CNN to train the model which we customize the output classes for the prediction of custom dataset. Dataset are divided into 80% of data for trainingand 20% of data in testing where model will be trained only in training dataset untouched and evaluated in test dataset. Model will be run on testing dataset for better result.

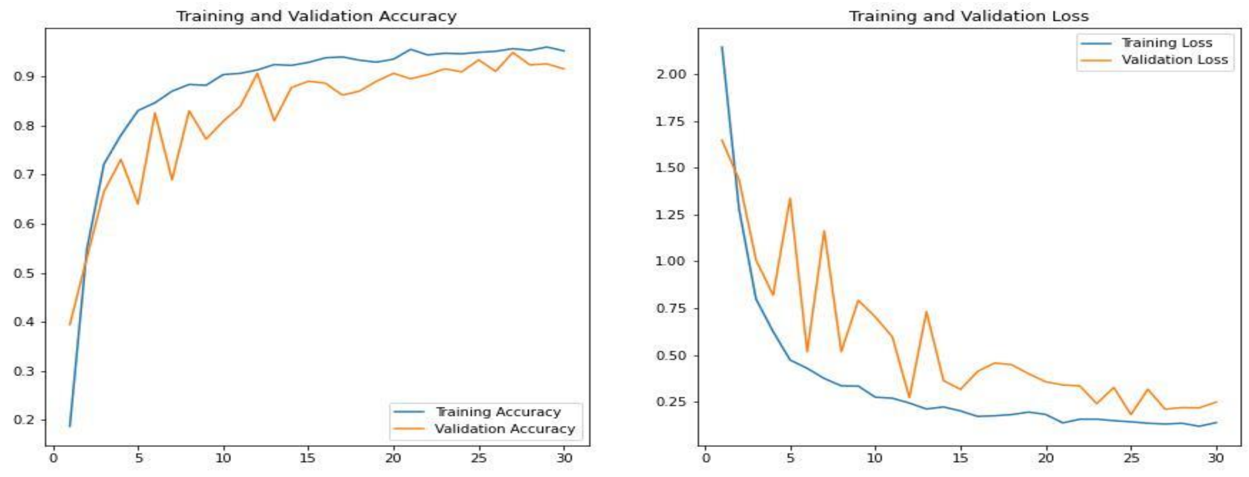


Figure1.6: 20 epoch

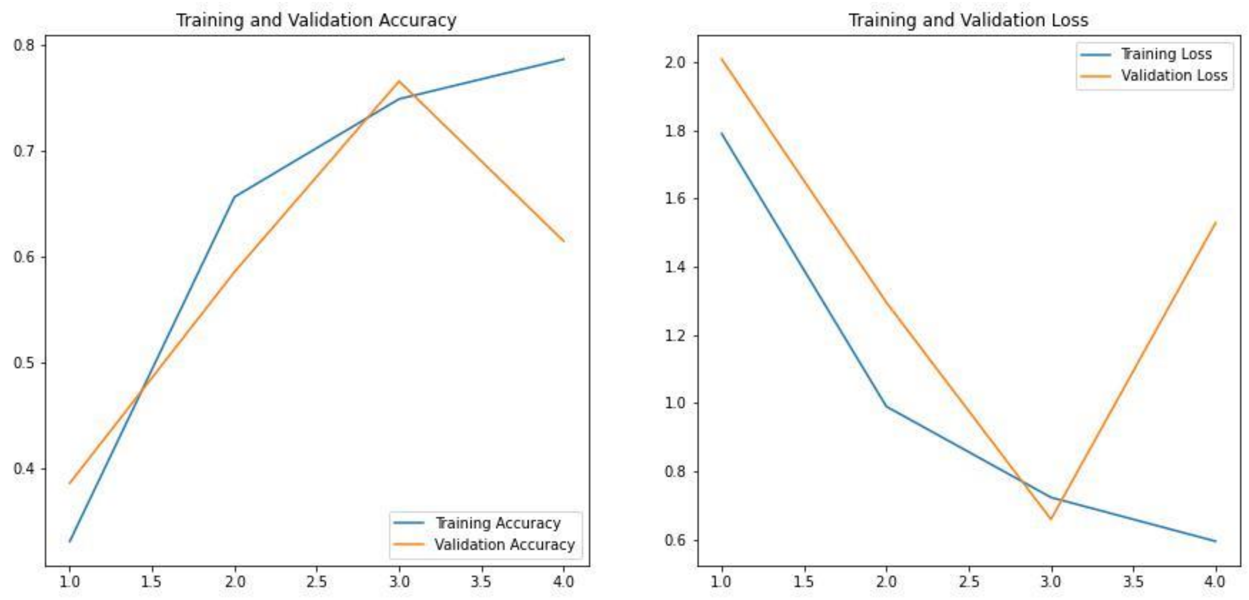


Figure1.7: 4 epoch

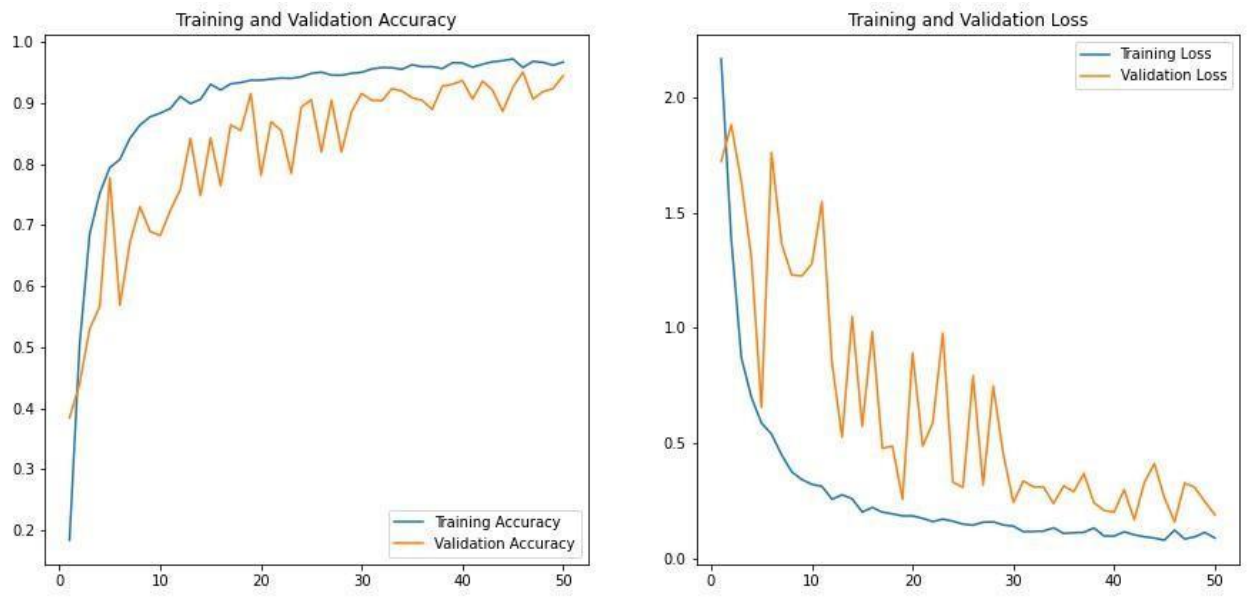


Figure1.8: 50 epoch

3.4. Software Development Lifecycle

The framework that was used for developing this project is the incremental model. This model combines the linear sequential model with the iterative prototype model. New functionalities were added as each increment developed. The phases of the linear sequential model were: analysis, design, coding, and testing. The software repeatedly passed through these phases in iteration and increment was delivered with progressive changes.

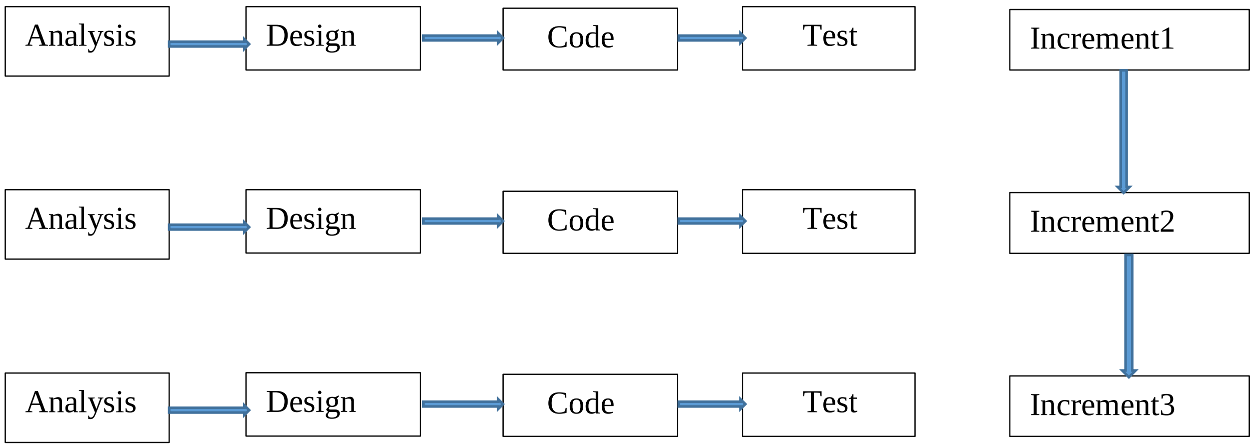


Figure1.9: Incremental Model-Design

The Incremental model include following steps:

3.4.1. Requirement analysis: In the first phase of the incremental model, the product analysis expertise identifies the requirement analysis team. To develop the software under the incremental model, this phase performs a crucial role.

3.4.2. Design and Development: In this phase of the incremental model of SDLC, the design of the system functionality and the development method are finished with success. When software develops new practicality, the incremental model uses style and development phase.

3.4.3. Testing: In this incremental model, the testing phase checks the performance of each existing function as well as additional functionality. In the testing phase, the various methods are used to test the behavior of each task.

3.4.4. Implementation: Implementation phase enable the coding phase of the development system. It involves the final coding that design in designing and development phase and tests the functionality in the testing phase. After completion of this phase, the number of the product working is enhanced and upgraded up to the final system product.

3.5. Hardware and Software Required

3.5.1. Hardware

3.5.1.1. Mobile device:

In our project, the device is used to take image and to see our result.

3.5.1.2. Farm: From the farm we collect different tomato leaf to train the system for the achievement of output.

3.5.2. Software

3.5.2.1. AI: Artificial intelligence is the stimulation of human intelligence processes by machines, especially machine systems. Specific application of AI includes expert systems, natural language processing, and speech recognition and vision.

3.5.2.2. Python: Python is an interpreted high-level general-purpose programming language. Its design philosophy emphasizes code readability with its use of significant indentation. Its language constructs as well as its object-oriented approach aim to help programmer write clear, logical code for small and large scale project.

4. Block Diagram

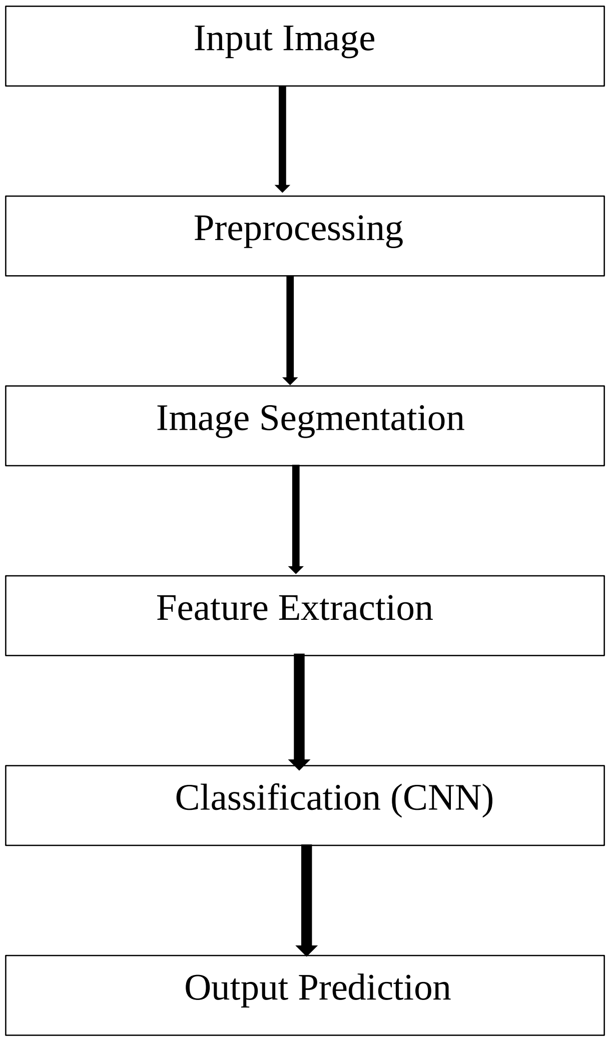


Figure1.10: Block-Diagram

4.1. Preprocessing

4.1.1. Background removal: The removal of the unrelated background will occur and only leaf remain same.

4.1.2. Sharpen image: It improves the appearance of detail by increasing a slightly complex mechanism.

4.1.3. Remove noise: The effect of noise in a picture causes it to look mottled, grainy, texture or snowy.

4.1.4. RGB to HSV: The HSV model provides the best starting point for creating optimal color combination.

4.1.5. Segmentation (Fuzzy C-mean Clustering): The proposed algorithm is it can find the optimal number of clusters by using cluster validity functions. A segmentation technique based on fuzzy clustering is performed on the enhanced picture.

4.1.6. Feature Extraction: The dataset need a more complex neural network to learn, so weights (accuracy) begin to saturate and then degrade. The ResNet-50 architecture modify the entire connected layer and final layer parameters. For ResNet-50 architecture, image input is set to 256*256.

4.1.7. Classification: The desired feature vectors such as color, texture, morphology and form are extracted in the feature. The fully connected layer is the higher level representation of the input signal, the output resulting from the convolution activation and pooling layers previously added. The dense layer is used at this stage to identify the input picture according to the training set by looking at the features

5. Flowchart

.

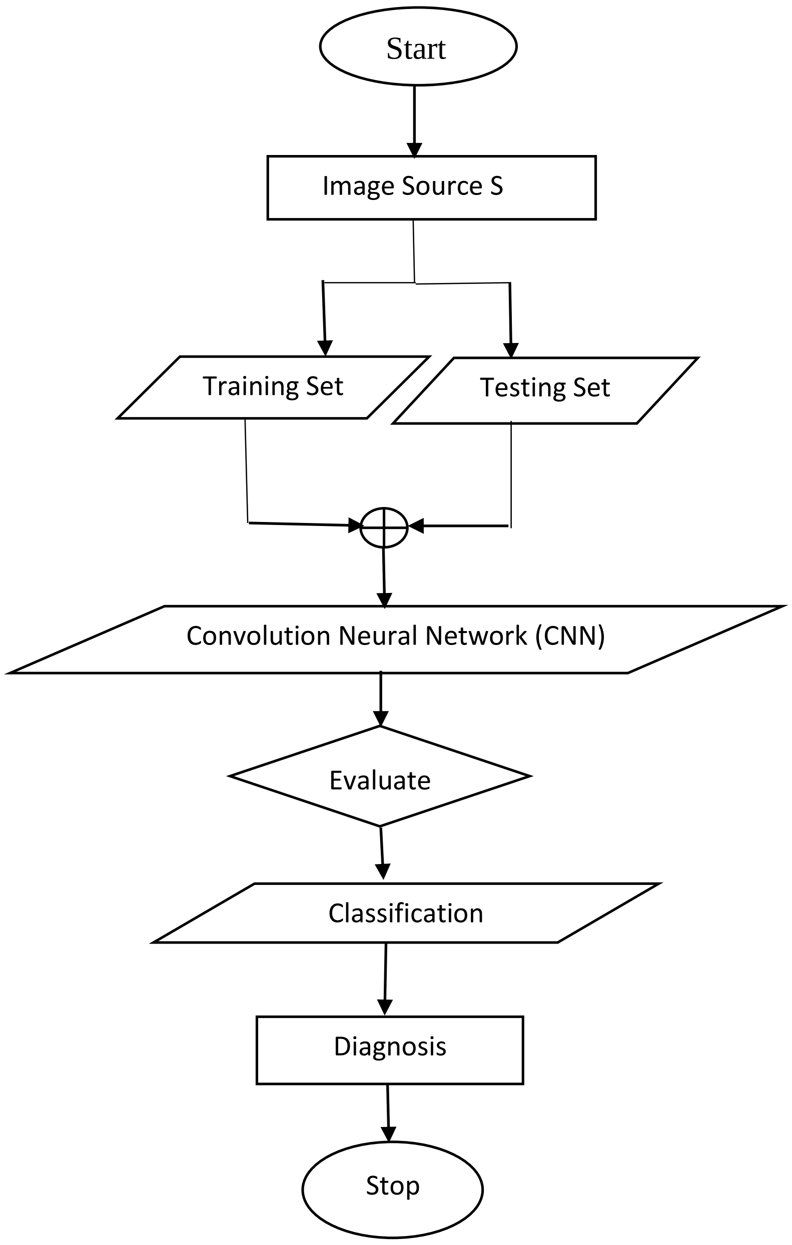


Figure1.11: Flowchart

6. Use Case Diagram

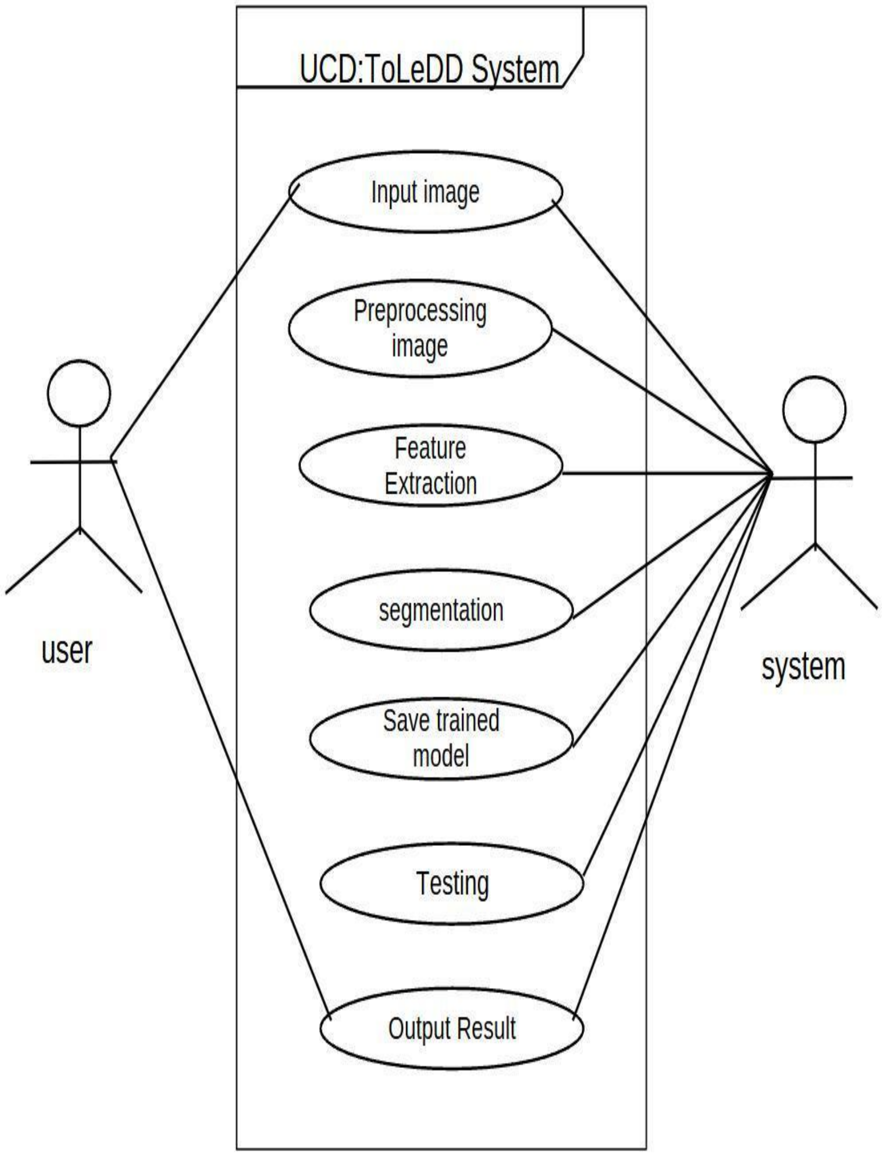


Figure1.12: Use Case Diagram

7. Sequence Diagram

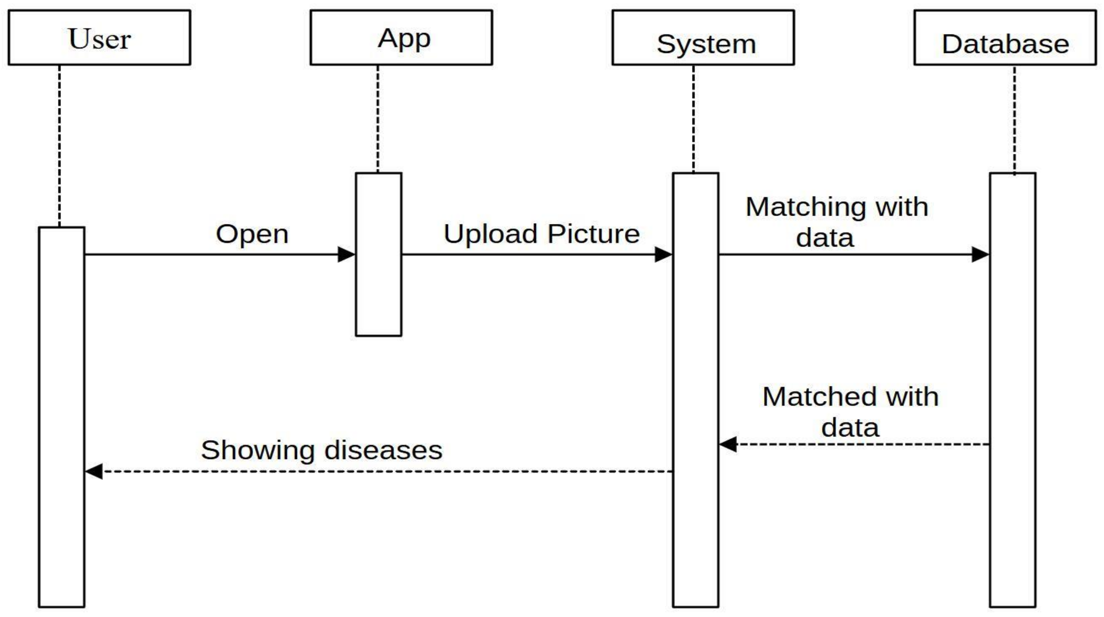


Figure1.13: Sequence Diagram

8. Activity Diagram

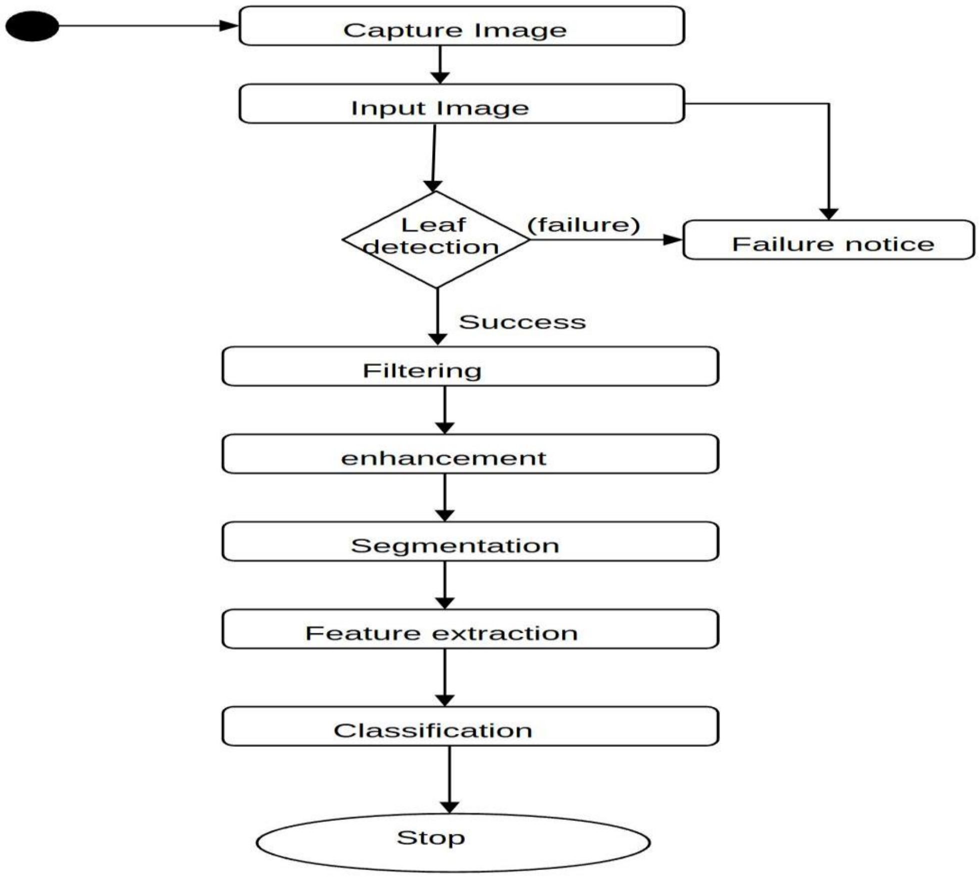


Figure1.14: Activity Diagram

9. Gantt-Chart

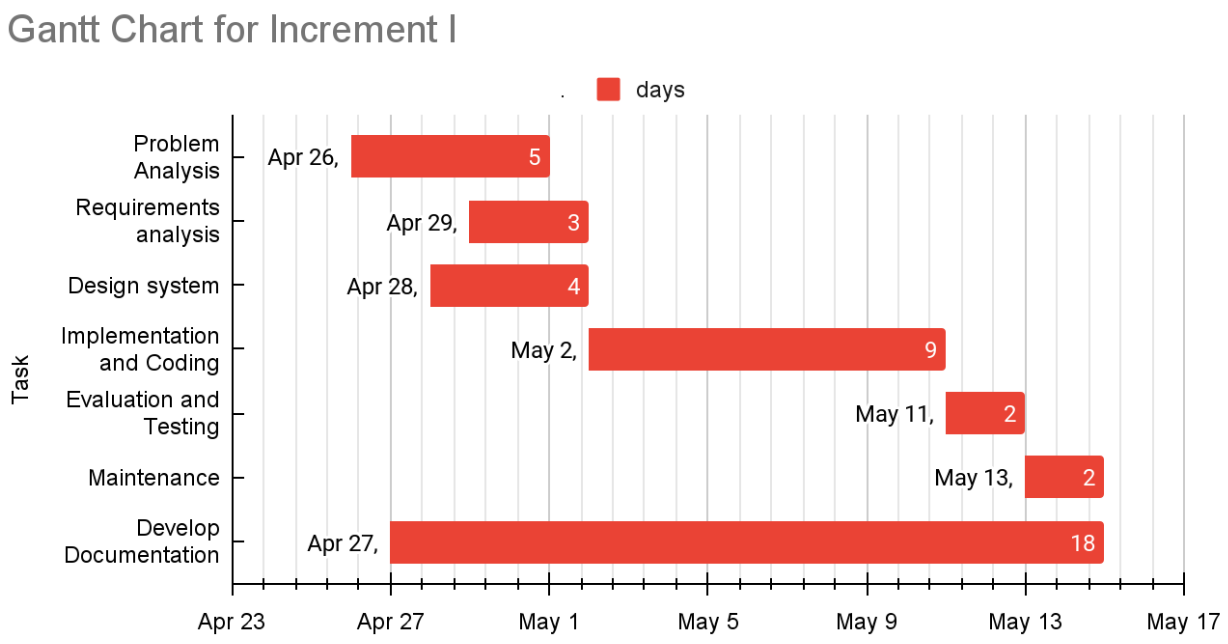


Figure1.15: Increment-I

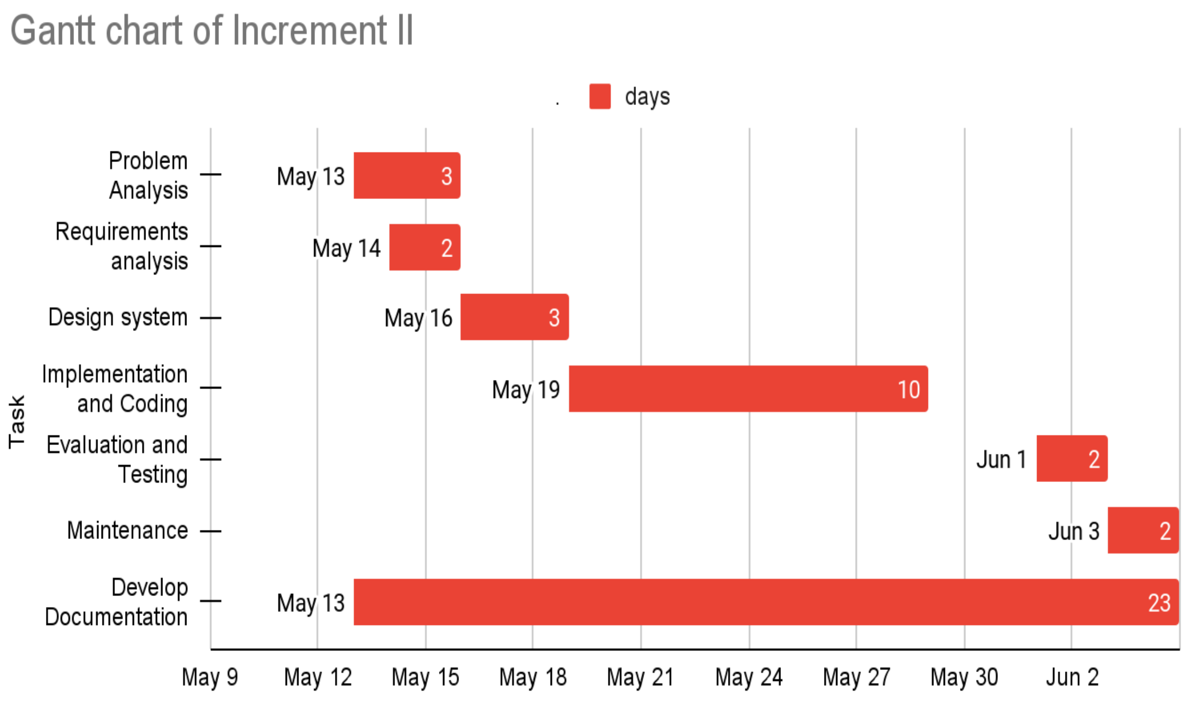


Figure1.16: Increment-II

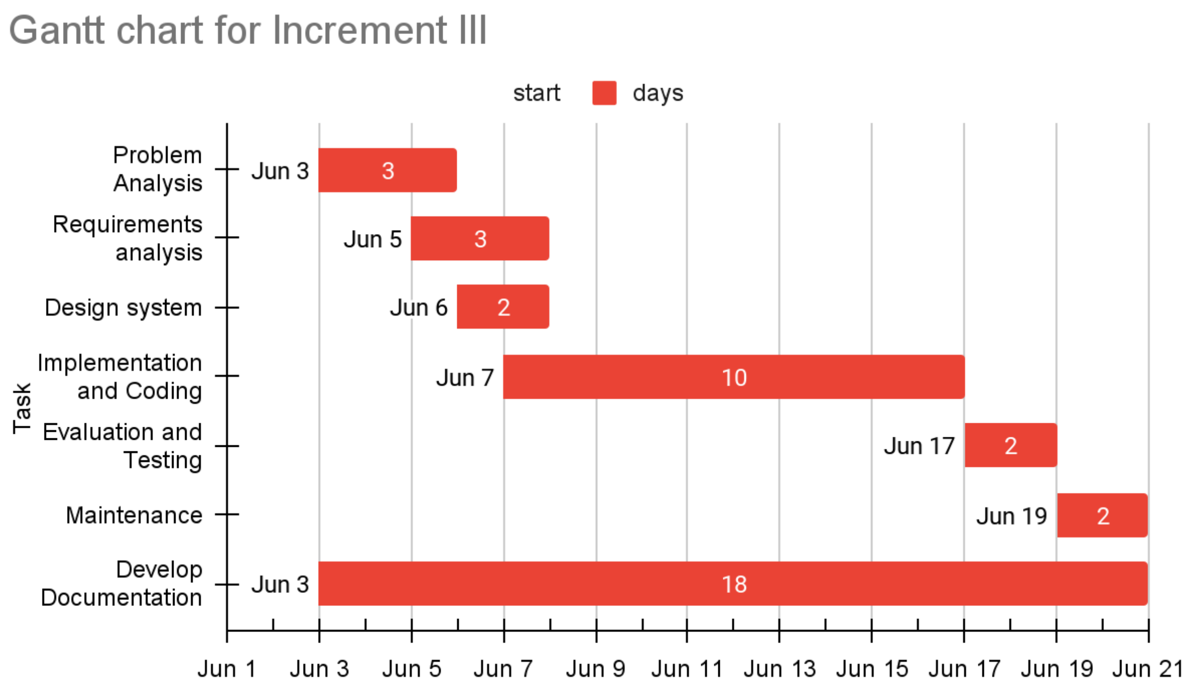


Figure1.17: Increment-III

10. Project Deliverables

Protecting plant leaf in organic farming is not an easy task. To prevent losses, small holder farmers are dependent on a timely and accurate crop disease diagnosis. This proposes a CNN based method for plant disease classification using the leaves of diseased plants. Building such a neural network with high efficiency is a complex task. In our system, a special deep learning model will be developed based on a special architectural convolution network to detect plant leaf diseases through images of healthy or diseased leaves. In this study a pre-trained convolution neural network will be fine-tuned and model will be deployed online. And the final result will be plant leaf diseases detection.

11. Project Task and Time Scheduling

The project schedule was designed as per requirements and constraints involved. This project was scheduled to be completed in about 3 months. Requirements analysis had been given emphasis. Research and database management was done first and well documented. Debugging and Testing was done prior to the completion of the project.

Task	**Increment I**	**Increment II**	**Increment III**
Problem analysis	5 Days	3 Days	3 Days
Requirements analysis and specification	3 Days	2 Days	3 Days
Design system	4 Days	3 Days	2 Days
Implementation and Coding	9 Days	10 Days	8 Days
Evaluation and Testing	2 Days	2 Days	2 Days
Maintenance	2 Days	2 Days	2 Days
Develop Documentation	1 Days	1 Days	1 Days
Total approx. duration in days	26 Days	23 Days	21 Days

The project schedule were performed as per the requirements and time constraints involved. Numerous informal conversations with the user which had assisted a lot in software development is not included in the chart.

12. Conclusion

In conclusion, plant leaf disease detection offers efficient solution for individuals and the farmers to detect the disease present in the plant leaf and make efficient and effective treatment of the disease. The development of plant leaf disease detection system holds great promise for revolutionizing agriculture machine learning algorithms, highly image processing techniques, and utilization of cutting edge technologies like drones and remote sensing, we have potential to detect and manage plant leaf disease with unprecedented accuracy and speed. This project not only aids in minimizing crops losses but also reduces the chemical treatments of plant leaf disease and promotes sustainable and environmentally friendly farming practices. As we continues to refine and expand these technologies, the future of plant leaf disease detection is bright, offering new hope for increased crops yields, improved food production, and a healthier planet.

13. References

[1] S. C. Madiwalar and M. V. Wyawahare, "Plant disease identification: A comparative study," 2017 International Conference on Data Management, Analytics and Innovation (ICDMAI), 2017, pp. 13-18, doi: 10.1109/ICDMAI.2017.8073478.

[2] S.D. Khirade, "Plant Disease Detection Using Image Processing,"2015 International Conference Communication Control and Automation, 2015,pp. 768-771, doi:10.1109/ICCUBEA.2015.153.

[3] G. Shrestha, Deepikha, M. Das and N. Dey,"Plant Disease Detection Using CNN,"2020 IEEE Applied Signal Processing Conference (ASPCON), 2020, pp. 109-113, doi:10.1109/ASPCON49795.2020.9276722.

[4] Ekansh Gayakwad, J Prabhu, R Vijay Anand, and M Sandeep Kumar. Trainingtime reduction in transfer learning for a similar dataset using deep learning. InIntelligent Data Engineering and Analytics: Frontiers in Intelligent Computing:Theory and Applications (FICTA 2020), Volume 2, pages 359Ű367. Springer,2020

[5] Belal AM Ashqar and Samy S Abu-Naser. Image-based tomato leaves diseases detection using deep learning. 2018.

[6] Prajwala Tm, Alla Pranathi, Kandiraju SaiAshritha, Nagaratna B Chittaragi,and Shashidhar G Koolagudi. Tomato leaf disease detection using convolutionalneural networks. In 2018 eleventh international conference on contemporary computing (IC3), pages 1Ű5. IEEE, 2018.

[7] G Geetharamani and Arun Pandian. IdentiĄcation of plant leaf diseases usinga nine-layer deep convolutional neural network. Computers & Electrical Engineering, 76:323Ű338, 2019.

[8] Mohit Agarwal, Abhishek Singh, Siddhartha Arjaria, Amit Sinha, and SuneetGupta. Toled: Tomato leaf disease detection using convolution neural network.Procedia Computer Science, 167:293Ű301, 2020.

[9] Emre ¨Ozbȷlge, Mehtap K¨ose Uluk¨ok, ¨Onsen Toygar, and Ebru Ozbȷlge. Tomato disease recognition using a compact convolutional neural network. IEEE Access,10:77213Ű77224, 2022.

[10] Akshay Kumar and M Vani. Image based tomato leaf disease detection. In 2019 10th International Conference on Computing, Communication and NetworkingTechnologies (ICCCNT), pages 1Ű6. IEEE, 2019

Appendix

Training.py

```
from tensorflow.compat.v1 import ConfigProto

from tensorflow.compat.v1 import InteractiveSession


config = ConfigProto()

config.gpu_options.allow_growth = True

session = InteractiveSession(config=config)


from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

from tensorflow.keras.preprocessing.image import ImageDataGenerator

import tensorflow as tf

tf.compat.v1.disable_eager_execution()

import matplotlib.pyplot as plt

import numpy as np

import os

from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay


# Initialising the CNN

classifier = Sequential()


# Step 1 - Convolution

classifier.add(Conv2D(32, (3, 3), input_shape=(128, 128, 3), activation='relu'))


# Step 2 - Pooling

classifier.add(MaxPooling2D(pool_size=(2, 2)))


# Adding a second convolutional layer

classifier.add(Conv2D(32, (3, 3), activation='relu'))

classifier.add(MaxPooling2D(pool_size=(2, 2)))


# Step 3 - Flattening

classifier.add(Flatten())
```

```
# Step 4 - Full connection
classifier.add(Dense(units=128, activation='relu'))
classifier.add(Dense(units=10, activation='sigmoid'))

# Compiling the CNN
classifier.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Print the model architecture
classifier.summary()

# Data augmentation and preprocessing
train_datagen = ImageDataGenerator(rescale=1./255, shear_range=0.2, zoom_range=0.2, horizontal_flip=True)
test_datagen = ImageDataGenerator(rescale=1./255)

training_set = train_datagen.flow_from_directory(
    'C://Users//Asus//OneDrive//Desktop//Plant-Leaf-Disease-Prediction-main//Dataset//train',
    target_size=(128, 128),
    batch_size=6,
    class_mode='categorical'
)

valid_set = test_datagen.flow_from_directory(
    'C://Users//Asus//OneDrive//Desktop//Plant-Leaf-Disease-Prediction-main//Dataset//val',
    target_size=(128, 128),
    batch_size=3,
    class_mode='categorical'
)

labels = training_set.class_indices
print(labels)

# Train the model and capture the history
history = classifier.fit(
    training_set,
    steps_per_epoch=50,
    epochs=20,
    validation_data=valid_set
)

# Save the model
classifier_json = classifier.to_json()
with open("model1.json", "w") as json_file:
    json_file.write(classifier_json)

classifier.save_weights("sanju.h5")
classifier.save("mandal.h5")
print("Saved model to disk")

# Plotting the training and validation accuracy
plt.figure(figsize=(12, 6))

# Plot training & validation accuracy values
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(['Train', 'Validation'], loc='upper left')

# Plot training & validation loss values
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(['Train', 'Validation'], loc='upper left')

plt.tight_layout()
plt.show()


# Confusion Matrix

# Get true labels and predictions from the validation set

valid_set.reset()

predictions = classifier.predict(valid_set)

predicted_classes = np.argmax(predictions, axis=1)

true_classes = valid_set.classes

class_labels = list(valid_set.class_indices.keys())


# Compute confusion matrix

cm = confusion_matrix(true_classes, predicted_classes)

cm_display = ConfusionMatrixDisplay(cm, display_labels=class_labels)


# Plot confusion matrix

plt.figure(figsize=(10, 7))

cm_display.plot(cmap=plt.cm.Blues)

plt.title('Confusion Matrix')

plt.show()
```

leaf.py

```
#Import necessary libraries

from flask import Flask, render_template, request


import numpy as np

import os


from tensorflow.keras.preprocessing.image import load_img

from tensorflow.keras.preprocessing.image import img_to_array

from tensorflow.keras.models import load_model


filepath = 'C://Users//Asus//OneDrive//Desktop//Plant-Leaf-Disease-Prediction-main//model.h5'

model = load_model(filepath)

print(model)


print("Model Loaded Successfully")
```

```
def pred_tomato_dieas(tomato_plant):
  test_image = load_img(tomato_plant, target_size = (128, 128)) # load image
  print("@@ Got Image for prediction")

  test_image = img_to_array(test_image)/255 # convert image to np array and normalize
  test_image = np.expand_dims(test_image, axis = 0) # change dimention 3D to 4D

  result = model.predict(test_image) # predict diseased palnt or not
  print('@@ Raw result = ', result)

  pred = np.argmax(result, axis=1)
  print(pred)
  if pred==0:
      return "Tomato - Bacteria Spot Disease", 'Tomato-Bacteria Spot.html'

  elif pred==1:
      return "Tomato - Early Blight Disease", 'Tomato-Early_Blight.html'

  elif pred==2:
      return "Tomato - Healthy and Fresh", 'Tomato-Healthy.html'

  elif pred==3:
      return "Tomato - Late Blight Disease", 'Tomato - Late_blight.html'

  elif pred==4:
      return "Tomato - Leaf Mold Disease", 'Tomato - Leaf_Mold.html'

  elif pred==5:
      return "Tomato - Septoria Leaf Spot Disease", 'Tomato - Septoria_leaf_spot.html'

  elif pred==6:
      return "Tomato - Target Spot Disease", 'Tomato - Target_Spot.html'

  elif pred==7:
      return "Tomato - Tomato Mosaic Virus Disease", 'Tomato - Tomato_mosaic_virus.html'

   elif pred==8:
      return "Tomato - Tomoato Yellow Leaf Curl Virus Disease", 'Tomato - Tomato_Yellow_Leaf_Curl_Virus.html'

   elif pred==9:
      return "Tomato - Two Spotted Spider Mite Disease", 'Tomato - Two-spotted_spider_mite.html'




# Create flask instance
app = Flask(__name__)

# render index.html page
@app.route("/", methods=['GET', 'POST'])
def home():
        return render_template('index.html')


# get input image from client then predict class and render respective .html page for solution
@app.route("/predict", methods = ['GET','POST'])
def predict():
     if request.method == 'POST':
        file = request.files['image'] # fet input
        filename = file.filename
        print("@@ Input posted = ", filename)

        file_path = os.path.join('C://Users//Asus//OneDrive//Desktop//Plant-Leaf-Disease-Prediction-main//static//upload', filename)
        file.save(file_path)

        print("@@ Predicting class......")
        pred, output_page = pred_tomato_dieas(tomato_plant=file_path)

        return render_template(output_page, pred_output = pred, user_image = file_path)
```

```
# For local system & cloud

if __name__ == "__main__":

    app.run(threaded=False,port=8080)
```