A Project Report on

Plant Leaf Disease Detection and Classification using ResNet50 and ResNet9

submitted in partial fulfillment for the award of

Bachelor of Technology

in

Computer Science and Engineering

by

V. Neeraj (Y20ACS520) **M. Yeswanth(Y20ACS505)**

M. Subhash (Y20ACS497) **N. Hemadri(Y20ACS521)**

Under the guidance of

Mr. K. Suman

Department of Computer Science and Engineering

Bapatla Engineering College

(Autonomous)

(Affiliated to Acharya Nagarjuna University)

BAPATLA – 522 102, Andhra Pradesh, INDIA

2023-2024

Department of
Computer Science and Engineering

CERTIFICATE

This is to certify that the project report entitled **Plant Leaf Disease Detection and Classification using ReNet50 and ResNet9** that is being submitted by V.Neeraj (Y20ACS520), M.Yeswanth (Y20ACS505), M.Subhash (Y20ACS497), N.Hemadri (Y20ACS521) in partial fulfillment for the award of the Degree of Bachelor of Technology in Computer Science and Engineering to the Acharya Nagarjuna University is a record of bonafide work carried out by them under our guidance and supervision.

Date:

Signature of the Guide	**Signature of the HOD**
Mr. K. Suman	**Dr. M. Rajesh Babu**
Assistant professor	**Associate Professor**

DECLARATION

We declare that this project work is composed by ourselves, that the work contained herein is our own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

Neeraj Velpula (Y20ACS520)
Muppaneni Yeswanth Krishna (Y20ACS505)
Melam Subhash Chandra (Y20ACS497)
Nuthalapati Hemadri (Y20ACS521)

Acknowledgement

We sincerely thank the following distinguished personalities who have given their advice and support for successful completion of the work.

We are deeply indebted to our most respected guide **Mr.K.Suman**, Asst. Prof**,** Department of CSE, for his valuable and inspiring guidance, comments, suggestions and encouragement.

We extend our sincere thanks to **Dr. M. Rajesh Babu** Prof. & Head of the Dept. for extending his cooperation and providing the required resources.

We would like to thank our beloved Principal **Dr. Nazeer Shaik** for providing the online resources and other facilities to carry out this work.

We would like to express our sincere thanks to our project coordinator **Dr. N. Sudhakar,** Prof. Dept. of CSE for his helpful suggestions in presenting this document.

We extend our sincere thanks to all other teaching faculty and non-teaching staff of the department, who helped directly or indirectly for their cooperation and encouragement.

Neeraj Velpula (Y20ACS520)
Muppaneni Yeswanath Krishna (Y20ACS505)
Melam Subhash Chandra (Y20ACS497)
Nuthalapati Hemadri (Y20ACS521)

Table of Contents

List of figures

List of Tables

Abstract

This project uses ResNet50 and ResNet9 to present an advanced plant disease detection system. The major goal is to develop a trustworthy tool that uses deep learning algorithms to precisely detect and classify different plant diseases. It is projected that the system will greatly lower crop loss and increase agricultural productivity by offering a tool for early detection. The developed model is trained with a large dataset of plant images, giving it the ability to recognize complex patterns and features associated with diseases, which are frequently missed by the human eye.

This foundational knowledge for the system design comes from the CNN algorithm's capacity to process and learn from image data. Compared to more time-consuming and less accurate traditional methods, this approach offers significant improvements by giving 97% accuracy for detection and 99% accuracy for classification. Farmers job could be easy as this technology changes everything by making it possible to detect illnesses early, which lowers crop loss and boosts agricultural output. Furthermore, this project represents a step toward more intelligent and sustainable farming methods.

1 Introduction

Plant diseases affect crop yields and food security, which presents serious challenges to agriculture. This breakthrough is revolutionizing plant health by using deep learning and image processing to detect diseases. Farmers can now identify diseases in real time using a basic smartphone image, a significant improvement over earlier technique. With the use of this technology, outbreaks can be predicted, early detection and effective management are all made possible. It's a promising step toward a time when agriculture and technology coexist to improve farming sustainability and ensure our food supply.

Diseases are very detrimental to the health of plants, and that in turn affects them development. India has made significant strides in pesticide, fungicide, and herbicide advancement and research. But each year, owing to unknown factors, plants fall to a variety of recognized illnesses, resulting in the loss of countless tons of yield. The assault of such different forms of crop diseases causes a significant reduction of crop yield both qualitatively and quantitatively. Traditional methods for detecting diseases require manual inspection of plants by experts. This process needs to be continuous, and can be very expensive in large farms, or even completely unavailable to many small farm holders living in rural areas. This is why many attempts to automate disease detection have been made in the last few decades.

Current technological advancements have made the detection and diagnosis of plant diseases conceivable and achievable, hence paving the road for improved plant management in the event that a plant is infected. The suggested approach for the identification of plant leaf diseases concentrates on 14 plant species and 38 varieties or categories. Image Classification, Voice Recognition, and Processing of Natural Language has all shown exceptional performance over recent years due to Deep Learning. Utilizing a CNN to address the issue of identifying plant diseases yields excellent results.

CNN is acknowledged as the most effective Object Recognition technique. It is utilized for the creation of a predictive model that is operated on the input picture and changes the input in order to identify the output labels. In addition to classifying the plants, the model could also provide valuable information such as prevention measures and supplements. This could include recommendations for specific pesticides or fungicides to apply, cultural practices to reduce the risk of disease, or nutritional supplements to improve plant health

Plant disease detection using Convolutional Neural Networks (CNN) is a revolutionary advance in agriculture, providing a transformative solution to the ongoing challenge of accurate and timely detection of crop diseases. As the agricultural landscape increasingly embraces technological innovation, the inclusion of CNNs stands out as a key development that could change traditional practices. Basically, CNNs use deep learning techniques to analyse complex patterns and features in images of plant leaves. This methodology allows these networks to distinguish subtle visual cues that indicate disease, allowing them to classify plants as either healthy or diseased with high accuracy.

The complexity of CNNs lies in their ability to independently learn and adapt to different data, making them adept at dealing with the complexities inherent in the various manifestations of plant diseases. By quickly and accurately detecting diseases, these networks enable farmers and agronomists to take quick and targeted measures to prevent the spread of diseases and minimize crop losses. As the world grapples with the challenge of feeding a growing population in the face of environmental uncertainty, CNNs offer a promising path to increase agricultural productivity, ensure sustainable practices and strengthen the foundation of global food systems.

1.1 Problem Statement

The agricultural sector is vital for global food security, yet it faces challenges like plant diseases, leading to crop losses and economic strain. Prompt disease identification is crucial, but manual inspection is slow and error-prone. Modern technologies, especially in CNN models like ResNet50 and ResNet9, offer automated solutions for accurate disease detection, leveraging visual symptoms captured in images.

i CNNs, known for their prowess in image classification, present a promising avenue for automating plant disease detection.

ii By swiftly identifying visual symptoms, these deep learning techniques can expedite diagnosis, aiding in effective disease management.

iii Automation through CNNs can alleviate the labour-intensive and error-prone nature of manual inspection, enhancing crop protection and agricultural productivity.

iv The suggested solution to crop disease diagnosis is substantially less costly and needs shorter effort for predicting than existing deep learning-based systems.

1.2 Objectives

The primary objective of our project is to assist farmers in identifying diseases affecting their plant leaves. By leveraging modern technologies like deep learning, we strive to enhance farming productivity and cultivate a renewed interest in agricultural practices. Through automated disease detection and classification, we aim to streamline the farming process, reducing the time and financial burden on farmers. Ultimately, our endeavour seeks to empower farmers, mitigate crop losses, and foster sustainable agricultural practices for a thriving farming community.

1.3 Scope

To propose a model that accurately detect diseases present in plant leaves, providing the class of that belongs to. By leveraging advanced algorithms, it aims to reduce the burden on farmers while boosting productivity. By automating disease diagnosis and classification, the software seeks to streamline farming operations and enhance agricultural outcomes.

Ultimately, it aims to empower farmers with efficient tools for disease management, contributing to overall agricultural sustainability and success. In addition to detecting diseased leaves in the plants, it also classifies them. This could include the type of leaf that was gives to model and the class of disease to which it is belong to. Through this there is reduce the risk of disease, or nutritional supplements to improve plant health. The proposed model can accurately detect and classify the disease that is present in the plant leaf which can be very helpful to the farmers.

1.4 Deep Learning

Deep learning, a subset of machine learning, emulates the functioning of the human brain through artificial neural networks composed of interconnected nodes called neurons. These techniques specialize in constructing complex models with multiple hidden layers, enabling the extraction of intricate patterns and features from data. Among the myriad architectures, Convolutional Neural Networks (CNNs) excel in image recognition tasks by efficiently capturing spatial hierarchies, while Recurrent Neural Networks (RNNs) excel in sequential data analysis, making them suitable for tasks like speech recognition and language modelling.

1.4.1 Key Components and Hyperparameters

When constructing models using deep learning, several hyperparameters need to be carefully tuned to ensure optimal performance. Some of the key hyperparameters include:

Learning rate: This hyperparameter controls the step size taken by the optimizer during each iteration of training. Too small a learning rate can result in slow convergence, while too large a learning rate can lead to instability and divergence.

Batch Size: This hyperparameter defines the number of samples we use in one epoch to train a neural network. Choosing an appropriate batch size is crucial in training deep learning models. Larger batch sizes require more memory, both on the GPU/TPU and the CPU. If you have limited memory, you might need to decrease the batch size. Smaller batch sizes can sometimes lead to better generalization, meaning the model performs better on unseen data. This is because smaller batches introduce more randomness into the optimization process.

Epochs: This hyperparameter represents the number of times the entire training dataset is passed through the model during training. Increasing the number of epochs can improve the model's performance but may lead to overfitting if not done carefully.

Number of layers: This hyperparameter determines the depth of the model, which can have a significant impact on its complexity and learning ability.

Number of nodes per layer: This hyperparameter determines the width of the model, influencing its capacity to represent complex relationships in the data.

Architecture: This hyperparameter determines the overall structure of the neural network, including the number of layers, the number of neurons per layer, and the connections between layers. The optimal architecture depends on the complexity of the task and the size of the dataset.

Activation function: This hyperparameter introduces non-linearity into the model, allowing it to learn complex decision boundaries. Common activation functions include sigmoid, tanh, and Rectified Linear Unit (ReLU).

Dropout: Dropout is a regularization technique commonly used in neural networks to prevent overfitting and improve the performance of the model.

Loss Function: This hyperparameter compute error between actual and prediction values and measure models performance. Hyperparameters are fine tuned to minimise the loss function.

Optimizer: a model's optimizer is the algorithm that updates the weights of the model during training, using output from the loss function along with other model parameters.

1.4.2 Popular Activation Functions

Sigmoid Activation Function: The sigmoid activation function, also known as the logistic function, is a commonly used non-linear activation function in neural networks. It maps the input values to a range between 0 and 1, making it suitable for binary classification tasks where the output represents probabilities.

The Equation (1.1) represents the Sigmoid activation function formula.

$$f(x) = 1/1+e^{\wedge}-x \qquad \textbf{1.1}$$

where:

x is the input to the neuron.

e is the base of the natural logarithm (Euler's number).

ReLU Activation Function: The rectified linear unit (ReLU) or rectifier activation function increases the complexity of the neural network by introducing non-linearity, which allows the network to learn more complex representations of the data. The ReLU function sets all negative values to zero.

The Equation (1.2) represents the ReLU activation function formula.

$$f(x) = max(0, x) \qquad \textbf{1.2}$$

where: *x* is the input to the neuron.

Softmax Activation Function: The softmax activation function is commonly used in the output layer of neural networks, particularly in multi-class classification tasks. It converts the raw output scores of a neural network into probabilities, ensuring that the sum of the probabilities across all classes adds up to 1.

The Equation (1.3) represents the Softmax activation function formula.

$$softmax(z_i) = \frac{exp(z_i)}{\sum_j exp(z_j)} \quad \textbf{1.3}$$

where:

Zᵢ is the raw output score (logit) for class *i*.

K is the total number of classes.

e is the base of the natural logarithm (Euler's number).

Tanh Activation Function: The hyperbolic tangent (tanh) activation function is a non-linear function commonly used in neural networks, particularly in the hidden layers. It shares similarities with the sigmoid activation function but produces output values in the range [−1,1] [−1,1], making it centered around zero.

The Equation (1.4) represents the tanh activation function formula.

$$\tanh x = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad \textbf{1.4}$$

where: *x* is the input to the neuron.

e is the base of the natural logarithm (Euler's number).

2 Background and Literature Survey

In this section we provide relevant background on previous work on Plant Disease Detection. Recently, several methods have been proposed for predicting diseases. Many of these methods are based on Convolutional Neural Networks and its pre trained models.

In the paper [1] proposed by Robert et al. a neural network was developed for efficiently identifying diseases in tomato plants. This research created a new method for the effective identification of diseases in tomato plants. The study focused on a specific tomato variety called Diamante Max. The methodology aimed to diagnose Phroma Rot, Leaf Miner, and Target Spot diseases by collecting both damaged and healthy leaves for data collection. Convolutional Neural Networks (CNN) were employed to determine the prevalence of tomato illnesses on the observed plants. The F-RCNN-trained abnormality detection model achieved an 80% confidence score, while the Transfer Learning illness identification model demonstrated a precision of 95.75%.

In the paper [2] X.-P. Fan et al. added a batch standardization layer to the convolutional layer of the Faster R-CNN model, introduced a central cost function to construct a mixed cost function, and used a stochastic gradient descent algorithm to optimize the training model. They used 9 kinds of corn leaf diseases with complex backgrounds in the field as the research object. Under the same experimental environment, the improved method had an average accuracy increase of 8.86%, and a single image detection time was reduced by 0.139s;

compared with the SSD algorithm, the average accuracy was 4.25% higher, and a single image detection time was reduced by 0.018 s.

In the paper [3] proposed by Bin Liu et al. addresses the detection of five apple leaf diseases, namely brown spot, mosaic, aria leaf spot, rust, and grey spot. Deep learning techniques are employed, specifically enhanced Convolutional Neural Networks (CNNs), to improve the detection of these diseases. Image annotation and data augmentation techniques are applied to construct a comprehensive dataset. The model, termed INAR-SSD, is trained and tested on a dataset comprising 26,377 photos of apple leaf diseases. Experimentally, the INAR-SSD model achieves a detection accuracy of 78.80%.

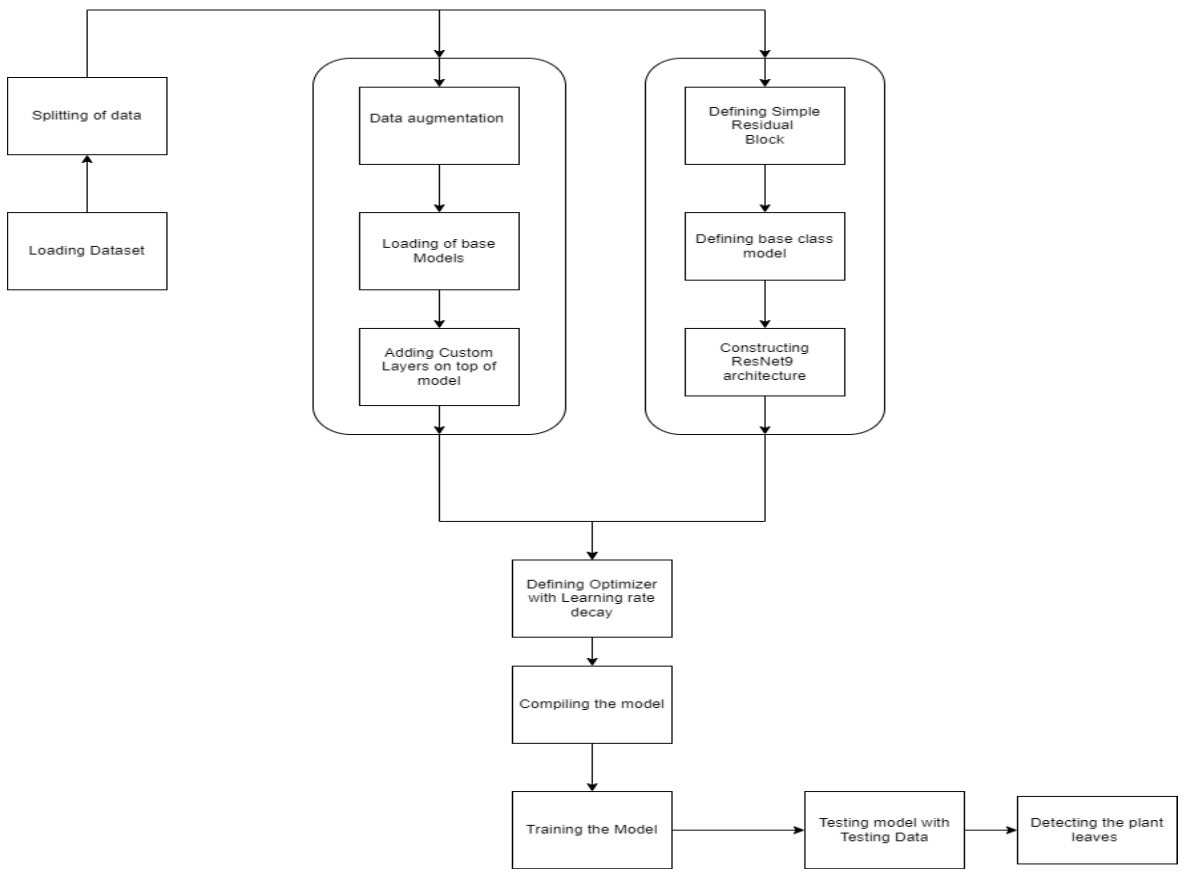
In the paper [4] proposed by Rekha Chahar et al. mainly focuses on the detection of diseases in leaf images of various plants, vegetables, fruits, and flowers, crucial for agriculture. The primary objective is to ascertain the health status and identify infectious diseases based on area identification in leaf images. The research utilizes arbitrarily obtained leaf photos from the internet for various plants, emphasizing the importance of remote disease detection for agricultural management.

In the Paper [5] probe the basic attorney affecting composition and layout of neural network applied in the pathology of plants it inspects in brief style further. They additionally utilized an exchange-taking in procedure from the recently prepared CNN Model. CNN plant infection acknowledgment is influenced by factors' rundown (that is) Image Captured Conditions, Image Background, Covariate Shift, and Limited Annotated Datasets. Disorder with Similar Symptoms, Simultaneous Disorders, Symptom Variation, Indications Segmentation, are seriously associated with the issue.

In paper [6] D. Venkatarama et al. delivered a PC sight depending on way to deal with discover the list of capabilities to order the select therapeutics plant leaves and recover its therapeutic brief. This strategy utilizes an estimating Neural Network classifier to recognize the leaf pattern. The strategy comprises of the accompanying advances: preprocessing, include extraction, characterization, and recovery of restorative qualities. The order includes the component vector computation and comparability coordinating

3 Proposed system

This chapter discusses the proposed system, the working technique, and specifics of the software and hardware components. The Deep learning model was proposed to implement a plant disease detection system using Convolutional Neural Networks (CNNs). The model will analyze images of plant leaves to accurately detect and classify diseases present in the plants. We propose a model that employs Convolutional Neural Networks for plant disease detection. This system can efficiently identify and classify plant diseases based on leaf images. The following block diagram shows the system architecture of this project.



Splitting of data
Loading Dataset
Data augmentation
Loading of base Models
Adding Custom Layers on top of model
Defining Simple Residual Block
Defining base class model
Constructing ResNet9 architecture
Defining Optimizer with Learning rate decay
Compiling the model
Training the Model
Testing model with Testing Data
Detecting the plant leaves

Figure 3.1 System Block Diagram

ADVANTAGES

It aids in effective disease management for farmers, improving productivity and fostering healthier crops.

i Real-time image-based disease analysis allows for immediate identification and response to plant diseases.

ii It enables farmers to build strong preventive strategies against diseases, improving crop yield and quality.

iii Additionally, the system aids in tracking the health conditions of plants, allowing for timely interventions.

APPLICATIONS

i Large-scale farming: Early detection can help prevent disease spread, saving crops.

ii Greenhouses: Perfect for maintaining optimal plant health conditions.

iii Home gardening: Helps hobbyists and home gardeners maintain healthy plants.

iv Agricultural research: Assists in studying plant diseases and their effects.

v Plant nurseries: Ensures the health of young plants before they are sold.

3.1 Working Methodology

The section on working methodology outlines the key steps involved in the process, beginning with data collection. This stage involves gathering relevant datasets that will be used to train and evaluate the model. Following data collection, preprocessing techniques are applied to clean, normalize, and prepare the data for analysis. Subsequently, the dataset is split into training, validation, and test sets to ensure the model's performance can be properly evaluated. Finally, the appropriate model selection process is undertaken, where various algorithms and architectures are considered and evaluated to determine the most suitable approach for the given task.

3.1.1 Overview of the System

We have to import our data set using keras preprocessing image data generator function also we create size, rescale, range, zoom range, horizontal flip. Then we import our image dataset from folder through the data generator function. Here we set train, test, and validation also we set target size, batch size and class mode from this function and we have to train using our own created network by adding layers of CNN.

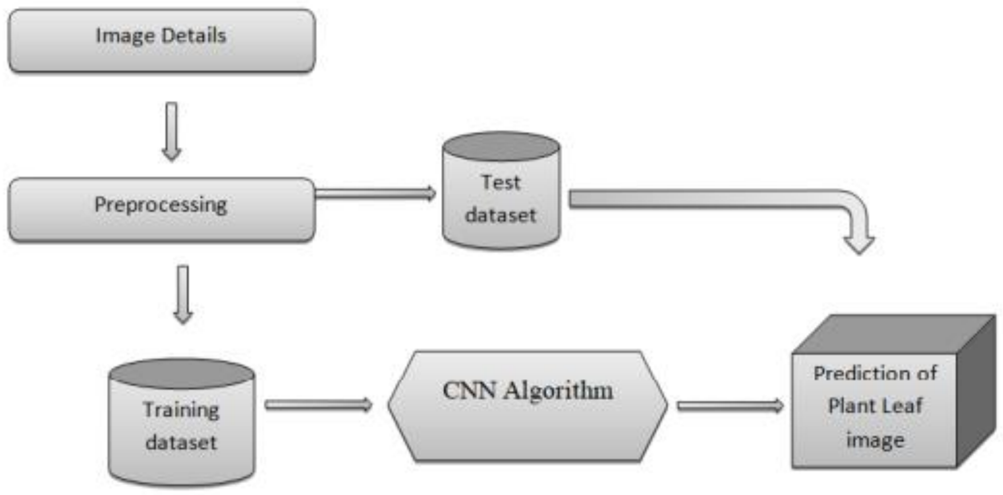


Figure 3.2 Overview of system

3.1.1.1 DFD(Data Flow Diagram):

A data flow diagram (DFD) is a graphical representation of the "flow" of data through an information system, modelling its process aspects. A DFD is often used as a preliminary step to create an overview of the system without going into great detail, which can later be elaborated. DFDs can also be used for the visualization of data processing (structured design). A DFD shows what kind of information will be input to and output from the system, how the data will advance through the system, and where the data will be stored. It does not show information about process timing or whether processes will operate in sequence or in parallel, unlike a traditional structured flowchart which focuses on control flow, or a UML activity workflow diagram, which presents both control and data flows as a unified model.

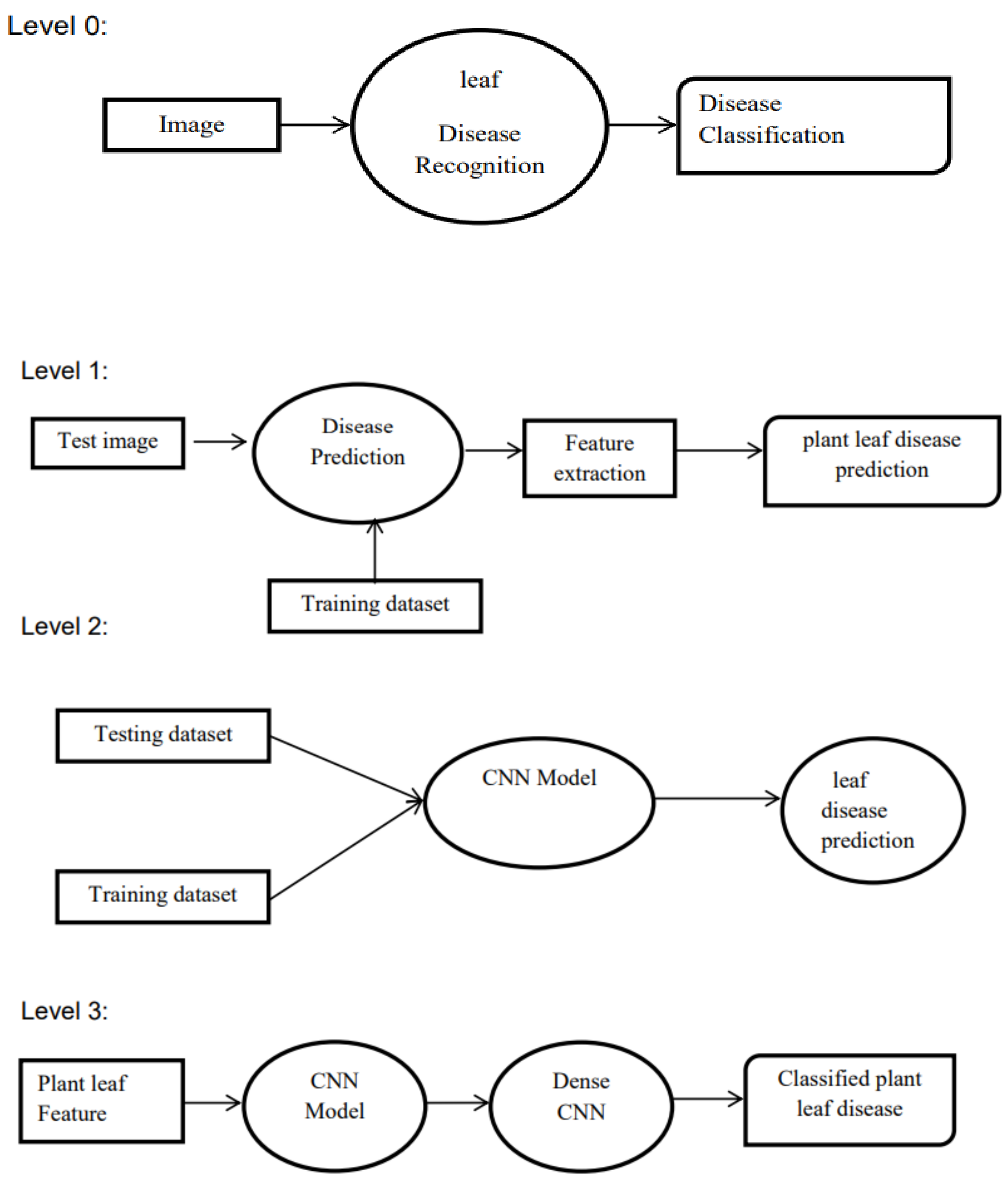


Figure 3.3 Data flow diagram

3.1.2 Model Steps:

Conv2d:

The 2D convolution is a fairly simple operation at heart: you start with a kernel, which is simply a small matrix of weights. This kernel “slides” over the 2D input data, performing an elementwise multiplication with the part of the input it is currently on, and then summing up the results into a single output pixel. The kernel repeats this process for every location it slides over, converting a 2D matrix of features into yet another 2D matrix of features. The output features are essentially, the weighted sums (with the weights being the values of the kernel itself) of the input features located roughly in the same location of the output pixel on the input layer.

MaxPooling2D layer

The MaxPooling2D layer is a type of pooling layer commonly used in convolutional neural networks (CNNs) for image classification tasks. Pooling layers are used to reduce the spatial dimensions (width and height) of the input volume, effectively down sampling the feature maps. MaxPooling2D specifically performs a down sampling operation by taking the maximum value from a set of neighbouring pixels in each region of the input feature map.

Flatten layer

It is used to flatten the dimensions of the image obtained after convolving it. Dense: It is used to make this a fully connected model and is the hidden layer. Dropout: It is used to avoid over fitting on the dataset and dense is the output layer contains only one neuron which decide to which category image belongs. Flatten is used to flatten the input. For example, if flatten is applied to layer having input shape as (batch_size, 2,2), then the output shape of the layer will be (batch_size, 4)

Dense layer

Dense implements the operation: output = activation(dot(input, kernel) + bias) where activation is the element-wise activation function passed as the activation argument, kernel is a weights matrix created by the layer, and bias is a bias vector created by the layer (only applicable if use_bias is True). These are all attributes of Dense.

Dropout layer

The Dropout layer randomly sets input units to 0 with a frequency of rate at each step during training time, which helps prevent overfitting. Inputs not set to 0 are scaled up by 1/(1 - rate) such that the sum over all inputs is unchanged. 17 Note that the Dropout layer only applies when training is set to True such that no values are dropped during inference. When using model.fit, training will be appropriately set to True automatically

Image Data Generator:

It is that rescales the image, applies shear in some range, zooms the image and does horizontal flipping with the image. This Image Data Generator includes all possible orientation of the image.

Training Process:

train_datagen.flow_from_directory is the function that is used to prepare data from the train_dataset directory. Target_size specifies the target size of the image. Test_datagen.flow_from_directory is used to prepare test data for the model and all is similar as above. fit_generator is used to fit the data into the model made above, other factors used are steps_per_epochs tells us about the number of times the model will execute for the training data.

Epochs:

It tells us the number of times model will be trained in forward and backward pass.

Validation process:

Validation_data is used to feed the validation/test data into the model. Validation_steps denotes the number of validation/test samples.

3.1.3 Data Collection:

A dataset [7] is a structured collection of data that is organized and stored for easy access, retrieval, and analysis. In the context of machine learning and data science, a dataset typically refers to a set of observations or examples used to train, validate, or test a model. It consists of multiple instances or samples, each containing one or more features or attributes.

In this project, images were taken from the new plant disease dataset from Kaggle. This dataset is recreated using offline augmentation from the original dataset which is plant village dataset. This dataset consists of about 87K RGB images of healthy and diseased crop leaves which is categorized into 38 different classe**s.** A new directory containing 33 test images is created later for prediction purpose. The plants that are considered in this dataset are Orange, Grape, Raspberry, Tomato, Peach,

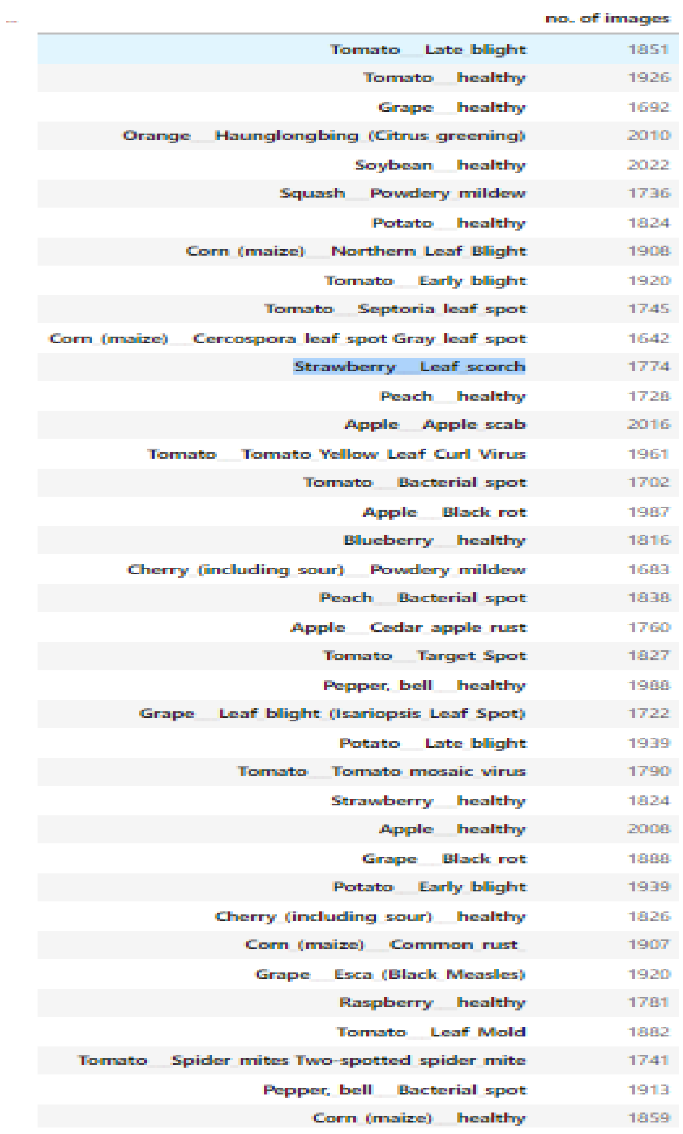
Apple, Cherry, Soybean, Pepper bell, Corn, Potato, Blueberry, Squash and Strawberry.

This dataset consists of following directories:

1) Train (70295 images)
2) Test (33 images)
3) Valid (17572 images)

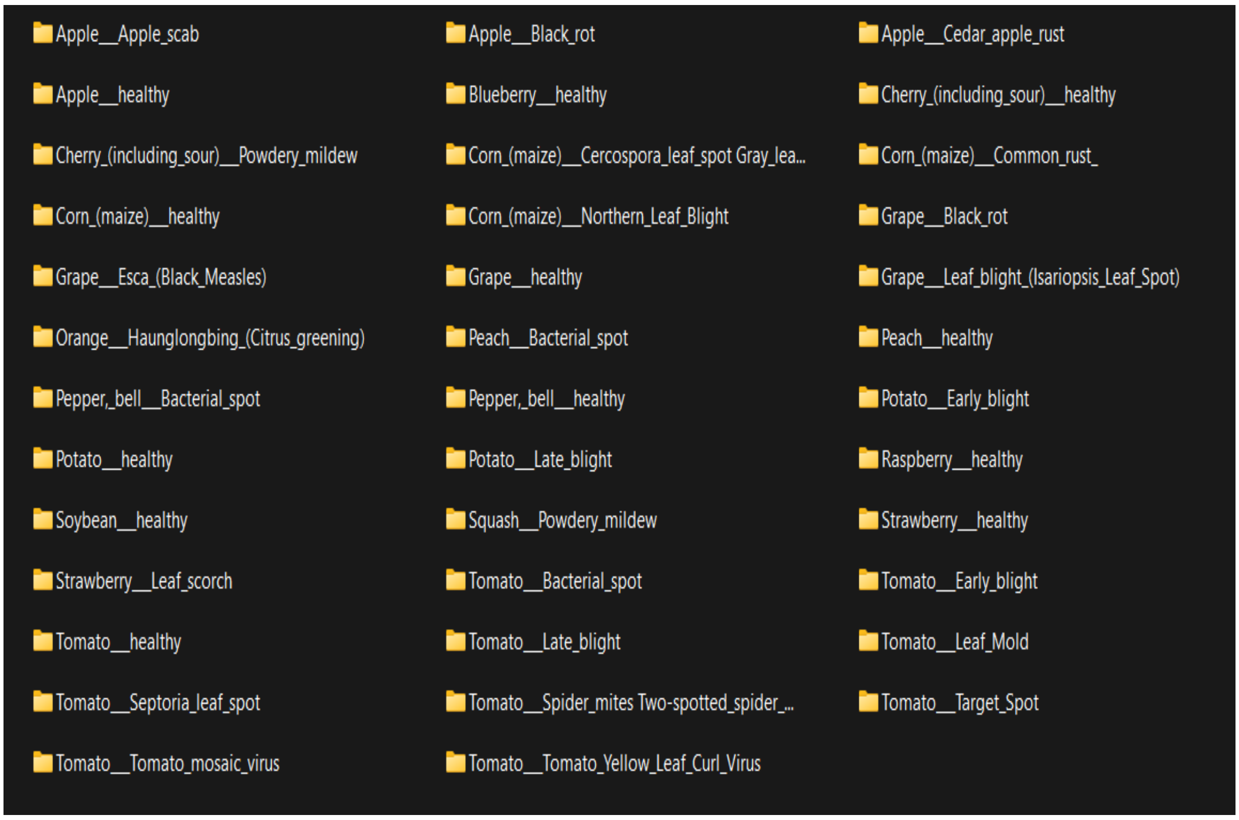
The following Figure 3.1 shows the images present in each class label of the dataset.

Figure 3.4 sample pics from dataset



	no. of images
Tomato___Late blight	1851
Tomato___healthy	1926
Grape___healthy	1692
Orange___Haunglongbing (Citrus greening)	2010
Soybean___healthy	2022
Squash___Powdery mildew	1736
Potato___healthy	1824
Corn (maize)___Northern Leaf Blight	1908
Tomato___Early blight	1920
Tomato___Septoria leaf spot	1745
Corn (maize)___Cercospora leaf spot Gray leaf spot	1642
Strawberry___Leaf scorch	1774
Peach___healthy	1728
Apple___Apple scab	2016
Tomato___Tomato Yellow Leaf Curl Virus	1961
Tomato___Bacterial spot	1702
Apple___Black rot	1987
Blueberry___healthy	1816
Cherry (including sour)___Powdery mildew	1683
Peach___Bacterial spot	1838
Apple___Cedar apple rust	1760
Tomato___Target Spot	1827
Pepper, bell___healthy	1988
Grape___Leaf blight (Isariopsis Leaf Spot)	1722
Potato___Late blight	1939
Tomato___Tomato mosaic virus	1790
Strawberry___healthy	1824
Apple___healthy	2008
Grape___Black rot	1888
Potato___Early blight	1939
Cherry (including sour)___healthy	1826
Corn (maize)___Common rust	1907
Grape___Esca (Black Measles)	1920
Raspberry___healthy	1781
Tomato___Leaf Mold	1882
Tomato___Spider mites Two-spotted spider mite	1741
Pepper, bell___Bacterial spot	1913
Corn (maize)___healthy	1859

Figure 3.5 different classes plants with no. of images



Apple__Apple_scab	Apple__Black_rot	Apple__Cedar_apple_rust
Apple__healthy	Blueberry__healthy	Cherry_(including_sour)__healthy
Cherry_(including_sour)__Powdery_mildew	Corn_(maize)__Cercospora_leaf_spot Gray_lea...	Corn_(maize)__Common_rust_
Corn_(maize)__healthy	Corn_(maize)__Northern_Leaf_Blight	Grape__Black_rot
Grape__Esca_(Black_Measles)	Grape__healthy	Grape__Leaf_blight_(Isariopsis_Leaf_Spot)
Orange__Haunglongbing_(Citrus_greening)	Peach__Bacterial_spot	Peach__healthy
Pepper,_bell__Bacterial_spot	Pepper,_bell__healthy	Potato__Early_blight
Potato__healthy	Potato__Late_blight	Raspberry__healthy
Soybean__healthy	Squash__Powdery_mildew	Strawberry__healthy
Strawberry__Leaf_scorch	Tomato__Bacterial_spot	Tomato__Early_blight
Tomato__healthy	Tomato__Late_blight	Tomato__Leaf_Mold
Tomato__Septoria_leaf_spot	Tomato__Spider_mites Two-spotted_spider_...	Tomato__Target_Spot
Tomato__Tomato_mosaic_virus	Tomato__Tomato_Yellow_Leaf_Curl_Virus	

Figure 3.6 Dataset Folder

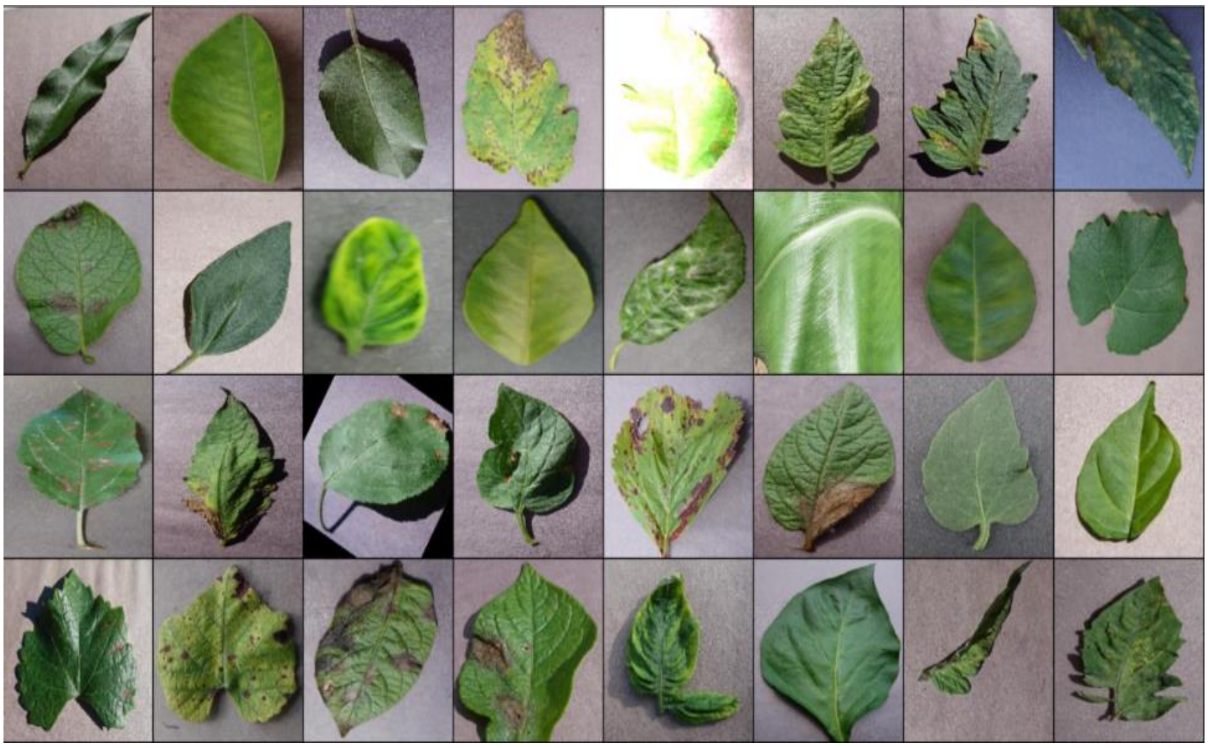


Figure 3.7 sample pictures of Dataset

3.1.4 Preprocessing:

To preprocess and create a dataset for training a model on image data. Firstly, the function resizes all images to a uniform size of 256x256 pixels for ensuring consistency across the dataset. Labels are inferred from the directory structure, where each subdirectory within the "train" and “valid” directories represents a distinct class, and images within those subdirectories are labelled accordingly. These labels are encoded in a categorical format, facilitating multi-class classification tasks. Additionally, the dataset is batched into groups of 32 images per batch, aiding in efficient model training. The data is shuffled randomly after each epoch (shuffle=True) to prevent the model from learning spurious patterns from the order of the data. Images are loaded in the RGB color space (color_mode="rgb"), and bilinear interpolation is employed for resizing (interpolation="bilinear") to ensure smooth image transformations. While basic preprocessing steps are applied, such as resizing, batching, label encoding, and shuffling, we do not explicitly include data augmentation techniques or validation data splitting because the dataset is already augmented.

3.1.5 Splitting the data:

Plant leaf disease detection and classification using deep learning involves leveraging DL techniques to automate identification and categorization of plant diseases. DL enables early detection, precision, efficiency, and contributes to sustainable agriculture. It utilizes CNNs, transfer learning, and object detection. Challenges include dataset quality, real-time implementation, and interpretability, while opportunities lie in remote monitoring, customization, and collaboration. This technology has the potential to revolutionize agricultural practices and enhance global food security. Moreover, it involves splitting the dataset into 80% for training, 20% for validation, and 33 unseen images for testing, facilitating robust model training and validation.

3.1.6 Model Selection:

Convolutional Neural Networks (CNN)

Convolutional Neural Networks (CNNs) are a specialized type of Artificial Neural Network (ANN) designed primarily for processing matrix-like data. They are particularly effective in tasks related to computer vision, such as image recognition and classification. They are especially effective in tasks where the spatial relationships between data points matter, like recognizing patterns in images, text and time-series data as well. The complexity in terms of no. of parameters to be learned and the training time exponentially grows in case of ANNs.

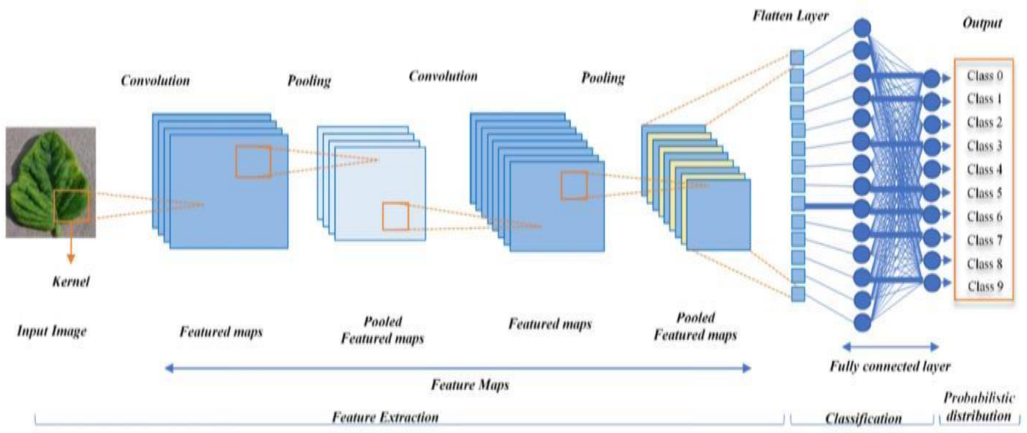


Figure 3.2 CNN Architecture

ResNet50V2:

ResNet50V2 is a sophisticated deep learning model deriving from ResNet, renowned for its integration of residual connections to boost gradient flow and alleviate the vanishing gradient problem commonly encountered in deep networks. As a variant of ResNet50, ResNet50V2 introduces notable enhancements such as transitioning from post-activation to pre-activation within the blocks, which helps in stabilizing and accelerating the training process. Furthermore, a unique feature of ResNet50V2 lies in its utilization of step 2 during the initial 3x3 convolution in each block, as opposed to the traditional use of the first 1x1 convolution. This modification strategically facilitates higher-level learning representations at earlier stages, enhancing the network's overall learning capabilities and convergence speed. By adjusting the sequence of operations and architectural configurations, ResNet50V2 optimizes information propagation across layers, resulting in superior training efficiency and heightened model performance across diverse deep learning tasks. The innovative modifications and structural refinements implemented in ResNet50V2 render it a robust and versatile choice for complex deep learning applications, showcasing its effectiveness in empowering advanced neural network architectures with enhanced learning capacities and performance outcomes.

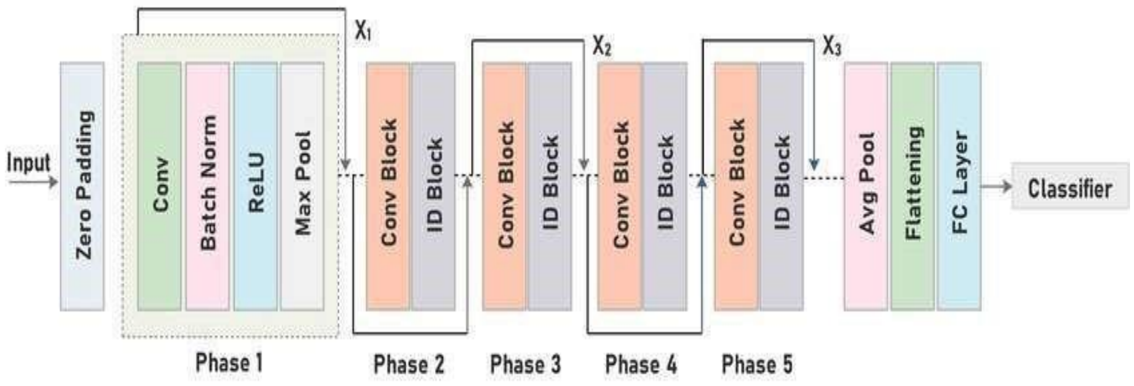


Figure 3.8 ResNet50 architecture Diagram

ResNet9:

ResNet9 is a variant of the popular ResNet (Residual Network) architecture, designed to balance performance and computational efficiency. It consists of nine convolutional layers, including convolutional, batch normalization, and ReLU activation layers. Unlike deeper versions of ResNet, ResNet9 aims to reduce computational complexity while maintaining competitive performance in tasks like image classification. By utilizing skip connections, ResNet9 addresses the vanishing gradient problem, allowing for deeper networks to be trained more effectively. This architecture introduces residual blocks, where the input is added to the output of each block, facilitating the flow of gradients during backpropagation. ResNet9 is particularly well-suited for tasks with limited computational resources, making it an efficient choice for real-world applications. It has been shown to achieve impressive results on various image datasets while requiring fewer parameters and computational resources compared to deeper ResNet variants.

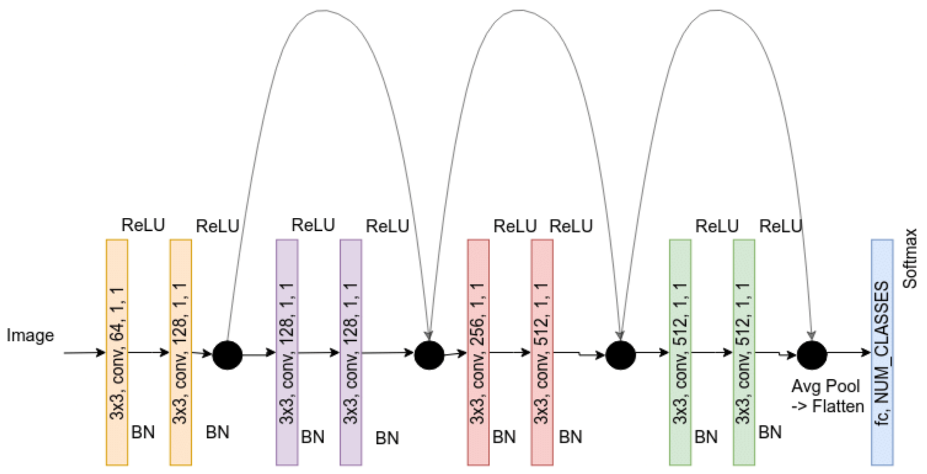


Figure 3.9 ResNet9 architecture diagram

3.1.7 Model training:

The chosen CNN model is trained using the prepared dataset. During training, the model learns to recognize patterns and features indicative of healthy or diseased plants. The training process involves adjusting the model's weights based on the error between predicted and actual labels. This iterative process continues until the model achieves satisfactory accuracy on the validation set.

3.1.8 Testing:

The final step involves testing the trained model on an independent dataset not seen during training or validation. This dataset typically comprises real-world images of plants with unknown disease status. The model's performance on this set provides insights into its ability to generalize to new and unseen data.

4 System Design

System design in software development encompasses creating an architectural plan for system implementation. UML diagrams are essential tools for illustrating various aspects of system design. Key UML diagrams include Use Case, Class, Sequence, Activity, Component, Deployment, State Machine, Package, Communication, and Object Diagrams. These diagrams depict user interactions, static structure, sequence of actions, workflows, physical components, deployment configuration, state transitions, modular structure, object interactions, and system snapshots. By utilizing a combination of these UML diagrams, developers can effectively communicate, plan, and implement software systems with structured documentation. Following are the list of uml diagrams

i Use case diagram

ii Class diagram

iii Activity diagram

iv State Chart Diagram

v Sequence Diagram

4.1 Use Case Diagram

Use-case diagrams describe the high-level functions and scope of a system. These diagrams also identify the interactions between the system and its actors. Actors are the external entities that interact with the system. The use cases are represented by either circles or ellipses. The Figure 4.1 shows the use case representation of the system.

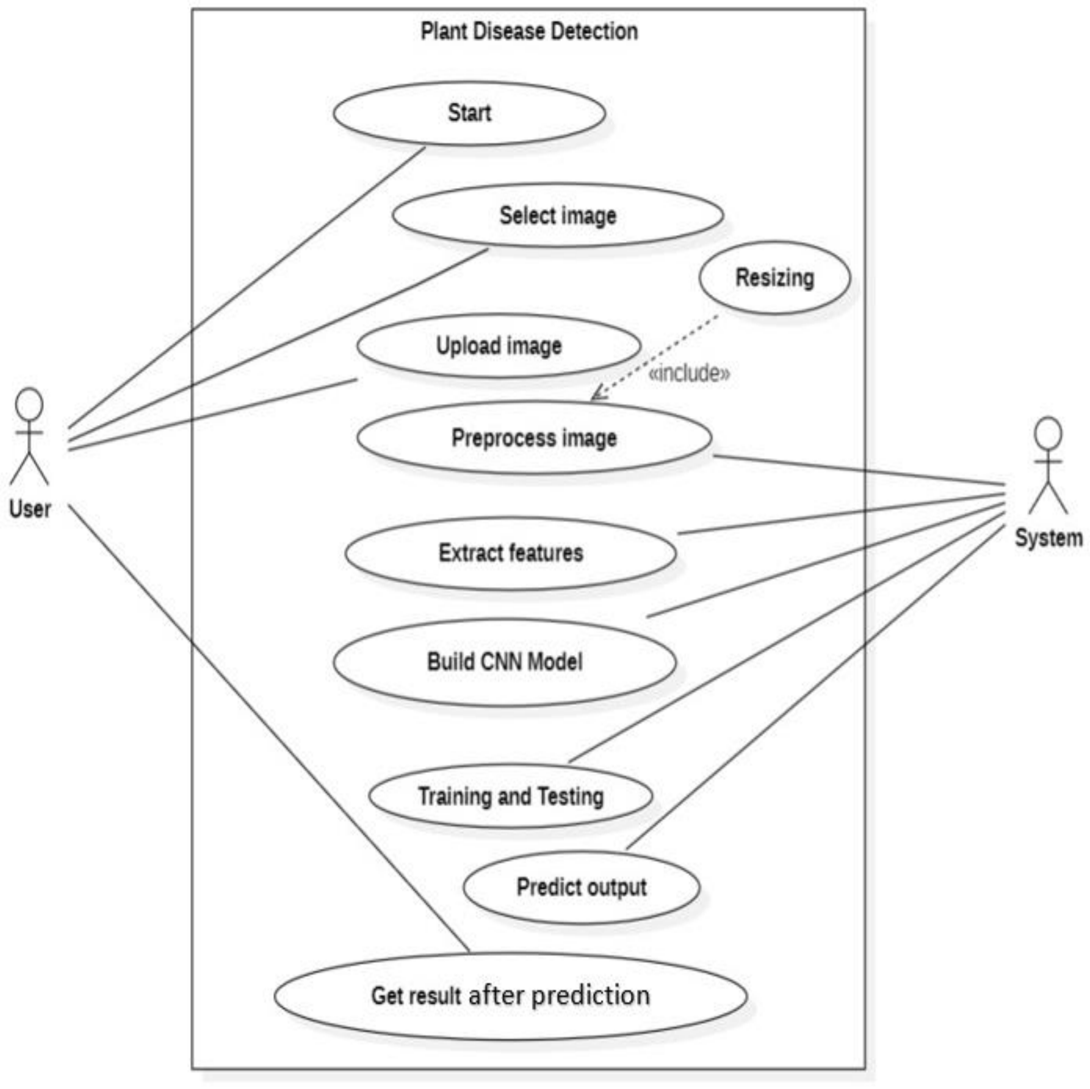


Figure 4.1 Use Case Diagram

4.2 Class Diagram

Class diagrams give an overview of a system by showing its classes and the relationships among them. Class diagrams are static – they display what interacts but not what happens when they do interact. In general a class diagram consists of some set of attributes and operations. Operations will be performed on the data values of attributes. The Figure 4.2 shows the class diagram representation of the system.

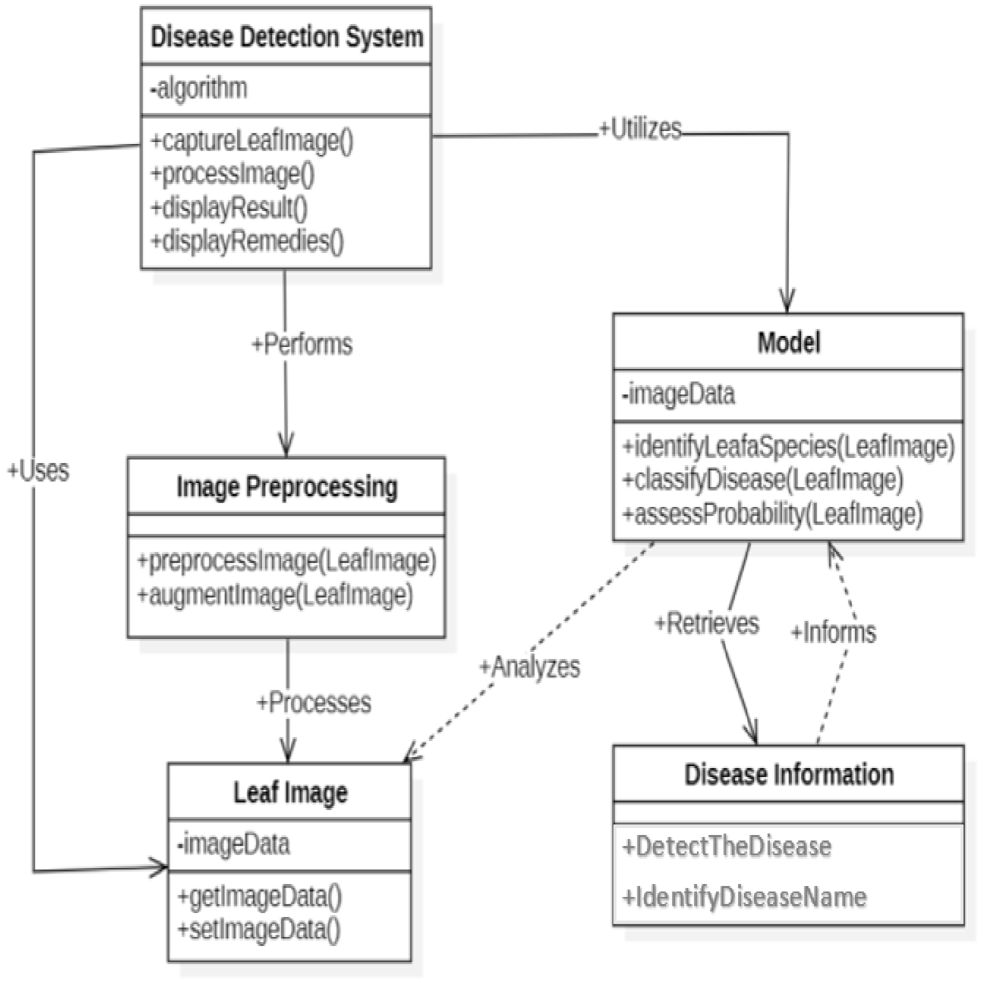


Figure 4.2 Class Diagram

4.3 Activity Diagram

Activity diagram is basically a flowchart to represent the flow from one activity to another activity. The activity can be described as an operation of the system. The control flow is drawn from one operation to another. This flow can be sequential, branched, or concurrent. In UML, an activity diagram provides a view of the behavior of a system by describing the sequence of actions in a process. The Figure 4.3 shows the activity diagram representation of the system.

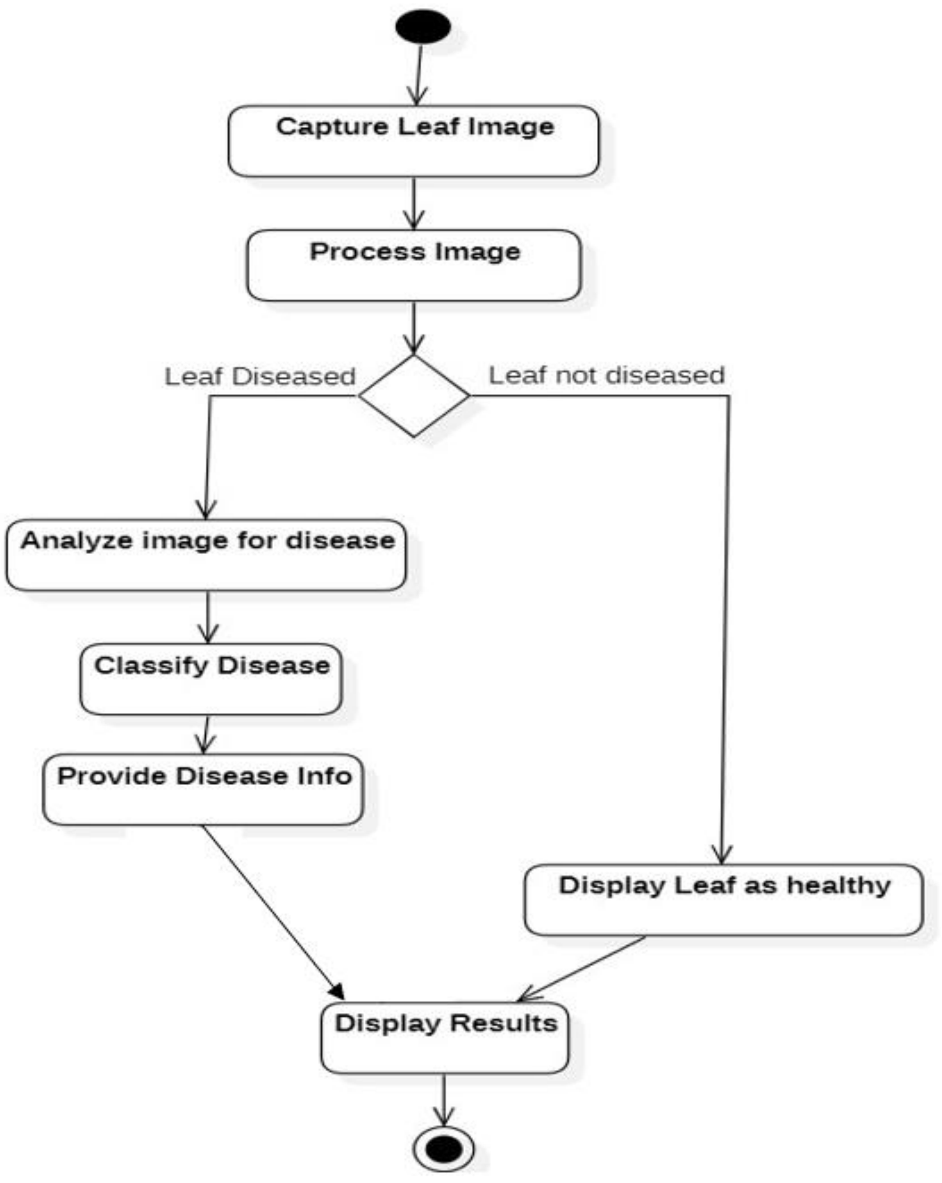


Figure 4.3 Activity Diagram

4.4 Sate Chart Diagram

A state diagram, also known as a state machine diagram or state chart diagram, is an illustration of the states an object can attain as well as the transitions between those states in the Unified Modeling Language (UML). The Figure 4.4 shows the state chart diagram representation of the system.

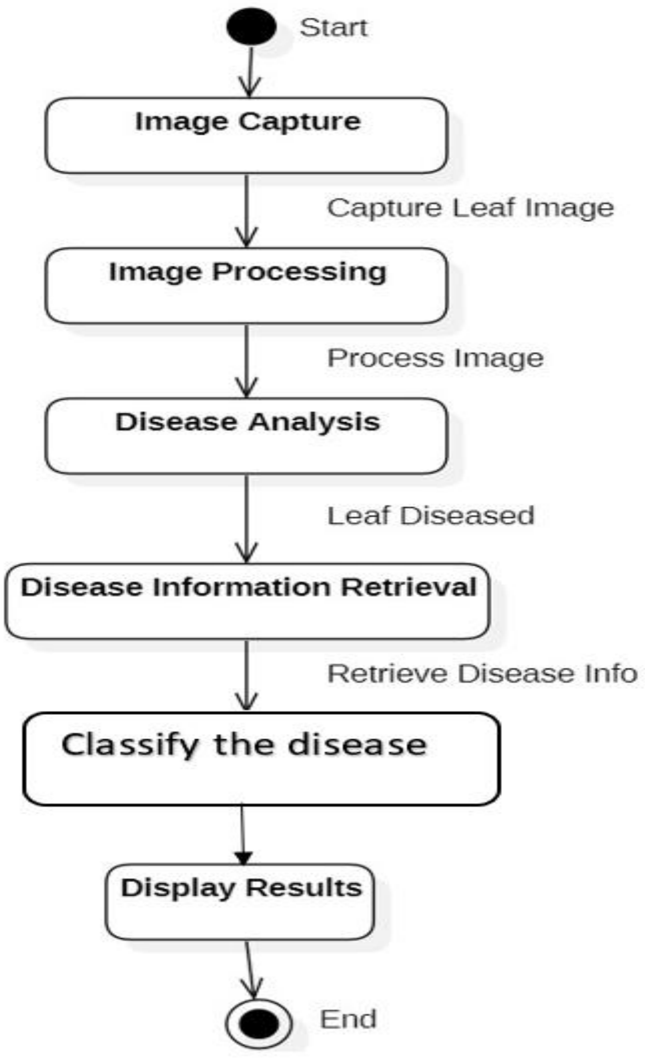


Figure 4.4 State Chart Diagram

4.5 Sequence Diagram

A sequence diagram shows object interactions arranged in time sequence. It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario. These diagrams are used by software developers and business professionals to understand requirements for a new system or to document an existing process. The Figure 4.5 shows the sequence diagram representation of the system.

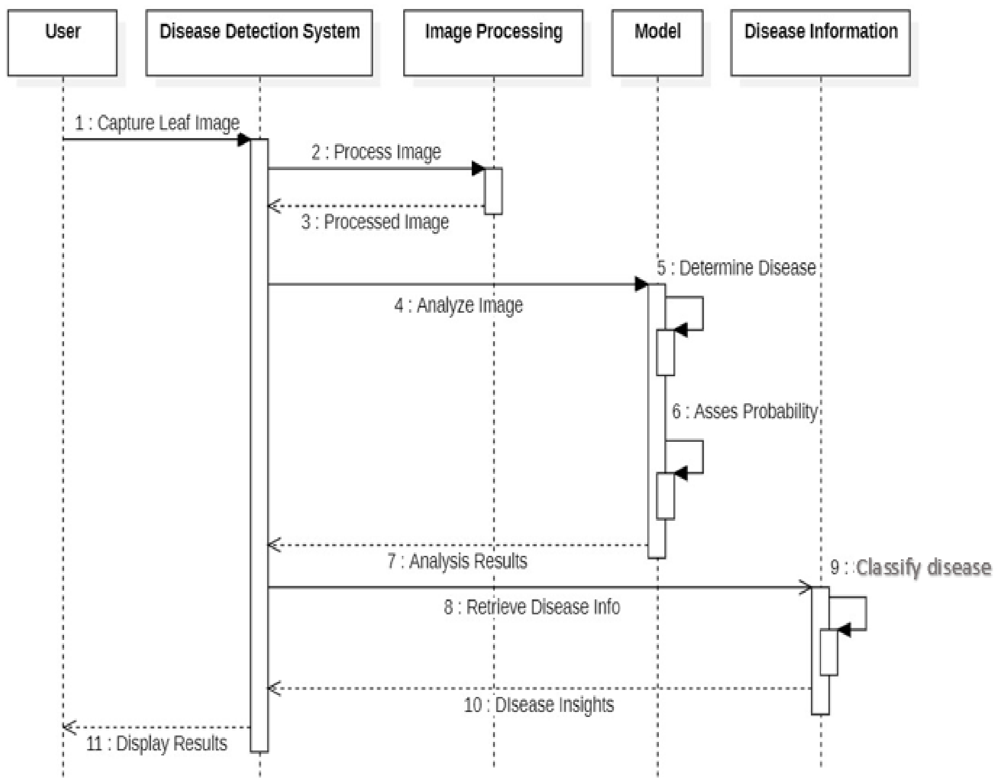


Figure 4.5 Sequence Diagram

5 Implementation

5.1 Exploratory Analysis

Prior to embarking on model development, it is essential to thoroughly explore the dataset. This investigation provides a thorough grasp of the dataset's traits and assists in pinpointing the essential features required for accurate disease detection.

The dataset will be thoroughly examined to evaluate its current condition, identify any sources of interference, and develop a plan to merge different data sources into a cohesive and top-notch dataset. Below, you will find the steps involved in detecting plant diseases.

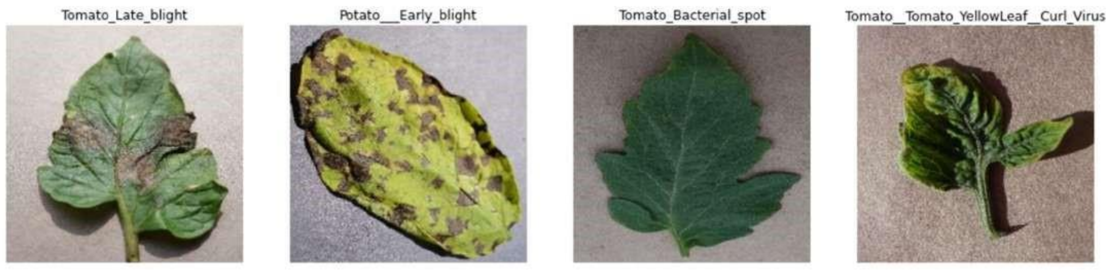
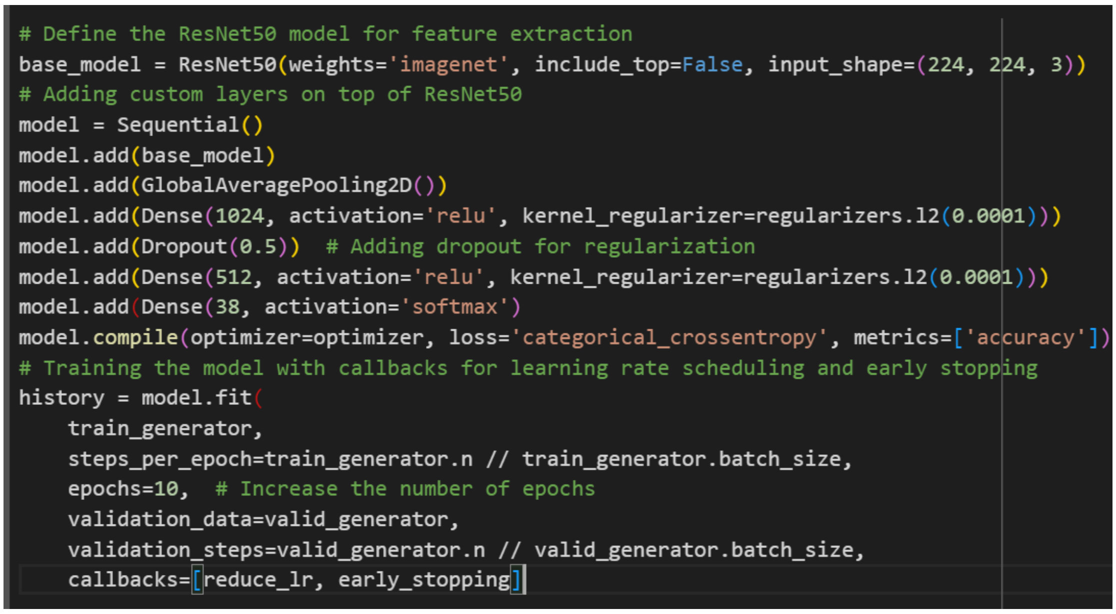


Figure 5.1 Images from the Dataset

5.1.1 ResNet50 Model:

The provided code snippet implements a transfer learning model using ResNet50 pretrained on ImageNet. It initializes the base_model without the classification layer to extract features, freezing the ResNet50 layers to preserve the pre-trained weights. A GlobalAveragePooling2D layer transforms the output into a one-dimensional feature array. Custom fully connected layers are then added for additional processing and classification. The model is compiled with categorical cross-entropy loss and accuracy metrics to optimize training. Callbacks for learning rate scheduling and early stopping enhance training efficiency and prevent overfitting. By leveraging the robust ResNet50 architecture for feature extraction and classification, this model demonstrates a powerful framework for handling complex deep learning tasks efficiently.

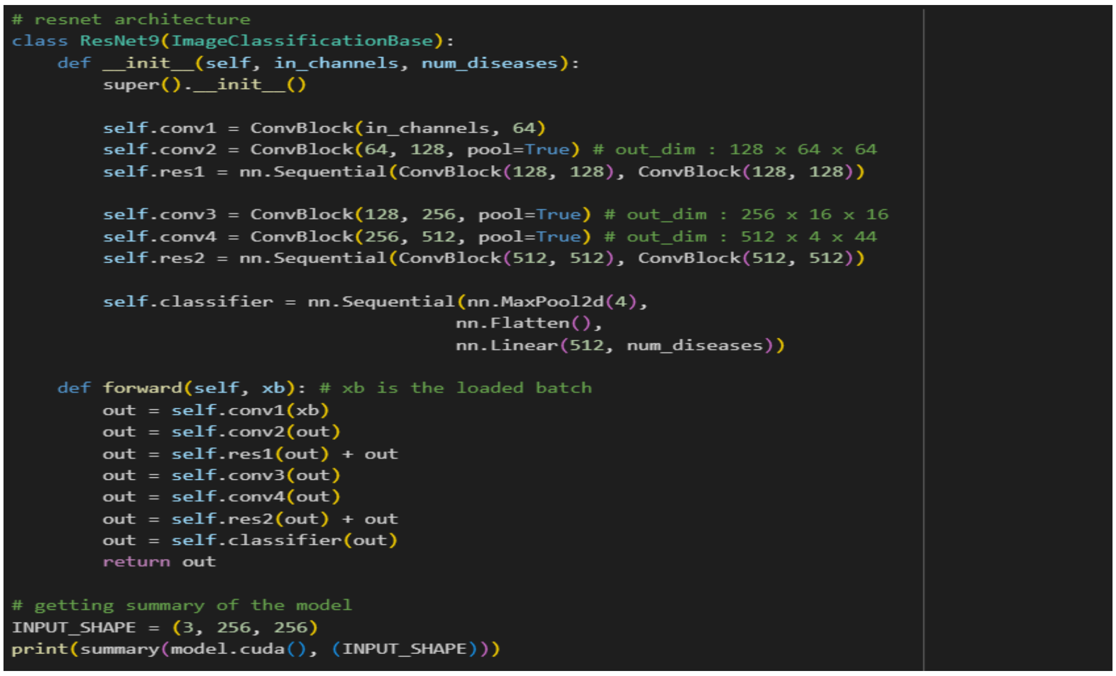


```
# Define the ResNet50 model for feature extraction
base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
# Adding custom layers on top of ResNet50
model = Sequential()
model.add(base_model)
model.add(GlobalAveragePooling2D())
model.add(Dense(1024, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dropout(0.5))  # Adding dropout for regularization
model.add(Dense(512, activation='relu', kernel_regularizer=regularizers.l2(0.0001)))
model.add(Dense(38, activation='softmax'))
model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['accuracy'])
# Training the model with callbacks for learning rate scheduling and early stopping
history = model.fit(
    train_generator,
    steps_per_epoch=train_generator.n // train_generator.batch_size,
    epochs=10,  # Increase the number of epochs
    validation_data=valid_generator,
    validation_steps=valid_generator.n // valid_generator.batch_size,
    callbacks=[reduce_lr, early_stopping]
```

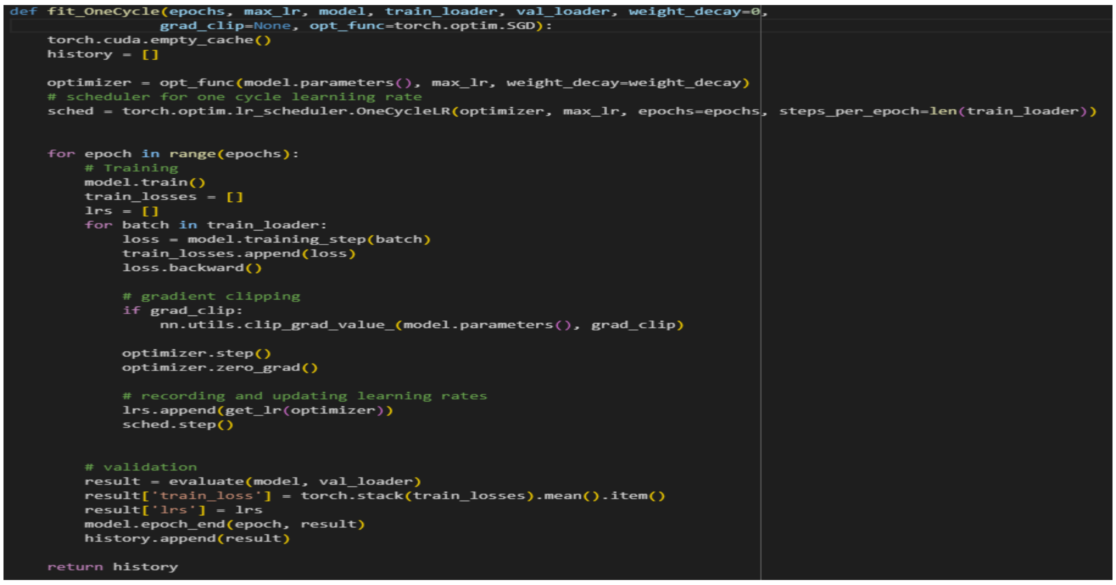
Figure 5.2 ResNet50 code

5.1.2 ResNet9 Model:

The ResNet9 model is structured as an ImageClassificationBase tailored for disease classification tasks, integrating specific convolutional blocks like res1 and res2 to promote feature learning. The classifier consists of components such as a max pooling layer, flattening operation, and a linear layer for disease prediction, ensuring comprehensive classification capabilities. The fit_OneCycle function drives training using one-cycle learning rate scheduling with an SGD optimizer, facilitating efficient parameter updates and gradient optimization throughout the training process. It continually monitors and adjusts losses, learning rates, and model performance metrics to enhance training efficiency and model accuracy. This holistic approach encompasses updates on training progress, parameter adjustments, and validation data evaluation, culminating in a comprehensive evaluation of the ResNet9 model's performance over multiple training epochs.



```python
# resnet architecture
class ResNet9(ImageClassificationBase):
    def __init__(self, in_channels, num_diseases):
        super().__init__()

        self.conv1 = ConvBlock(in_channels, 64)
        self.conv2 = ConvBlock(64, 128, pool=True) # out_dim : 128 x 64 x 64
        self.res1 = nn.Sequential(ConvBlock(128, 128), ConvBlock(128, 128))

        self.conv3 = ConvBlock(128, 256, pool=True) # out_dim : 256 x 16 x 16
        self.conv4 = ConvBlock(256, 512, pool=True) # out_dim : 512 x 4 x 44
        self.res2 = nn.Sequential(ConvBlock(512, 512), ConvBlock(512, 512))

        self.classifier = nn.Sequential(nn.MaxPool2d(4),
                                       nn.Flatten(),
                                       nn.Linear(512, num_diseases))

    def forward(self, xb): # xb is the loaded batch
        out = self.conv1(xb)
        out = self.conv2(out)
        out = self.res1(out) + out
        out = self.conv3(out)
        out = self.conv4(out)
        out = self.res2(out) + out
        out = self.classifier(out)
        return out

# getting summary of the model
INPUT_SHAPE = (3, 256, 256)
print(summary(model.cuda(), (INPUT_SHAPE)))
```



```python
def fit_OneCycle(epochs, max_lr, model, train_loader, val_loader, weight_decay=0,
                grad_clip=None, opt_func=torch.optim.SGD):
    torch.cuda.empty_cache()
    history = []

    optimizer = opt_func(model.parameters(), max_lr, weight_decay=weight_decay)
    # scheduler for one cycle learniing rate
    sched = torch.optim.lr_scheduler.OneCycleLR(optimizer, max_lr, epochs=epochs, steps_per_epoch=len(train_loader))


    for epoch in range(epochs):
        # Training
        model.train()
        train_losses = []
        lrs = []
        for batch in train_loader:
            loss = model.training_step(batch)
            train_losses.append(loss)
            loss.backward()

            # gradient clipping
            if grad_clip:
                nn.utils.clip_grad_value_(model.parameters(), grad_clip)

            optimizer.step()
            optimizer.zero_grad()

            # recording and updating learning rates
            lrs.append(get_lr(optimizer))
            sched.step()


        # validation
        result = evaluate(model, val_loader)
        result['train_loss'] = torch.stack(train_losses).mean().item()
        result['lrs'] = lrs
        model.epoch_end(epoch, result)
        history.append(result)

    return history
```

Figure 5.3 ResNet9 Code

The complete code can be found in : https://github.com/neerajhon/project.git

6 Evaluation and Testing

6.1 Evaluation

To evaluate the performance of the model, you would typically use a separate dataset (e.g., a validation set or a test set) that the model hasn't seen during training. This ensures an unbiased assessment of its generalization capabilities. The most common metrics for evaluating classification models include accuracy, precision, recall, F1-score, and confusion matrix.

6.2 Model Evaluation

Accuracy: Accuracy measures the proportion of correctly classified samples out of the total number of samples. While it's a straightforward metric, it might not be sufficient if the classes are imbalanced. The following Equation (6.2) represents the formula for calculation accuracy. **6.2**

$$Accuracy = no.\ of\ correct\ classifications / \text{total no. of classification attempted}$$

6.3 Testing

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub-assemblies, assemblies and/or a finished product. It is the process of exercising software with the intent of ensuring that the Software system meets its requirements and user expectations and does not fail in an unacceptable manner.

6.3.1 Levels of testing

The different levels of testing that are to be conducted are:

• Code Testing

• Program Testing

• System Testing

Code Testing: The code test has been conducted to test the logic of the program. Here, we have tested with all possible combinations of data to find out logical errors. The code testing is done thoroughly with all possible data available with library.

Program Testing: Program testing is also called unit testing. The modules in the system are integrated to perform the specific function. The modules have been tested independently, later Assembled and tested thoroughly for integration between different modules.

System Testing: System testing has been conducted to test the integration of each module in the system. It is used to find discrepancies between the system and its original objective. It is found that there is an agreement between current specifications and system documentation. Software Testing is carried out in three steps.

6.3.2 Unit Testing

In the unit testing we test each module individually and integrate with the overall system. Unit testing focuses verification efforts on the smallest unit of software design in the module. This is also known as module testing. The module of the system is tested separately. This testing is carried out during programming stage itself. In the testing step each module is found to work satisfactorily as regard to expected output from the module. There are some validation checks for fields also. For example, the validation check is done for varying the user input given by the user which validity of the data entered. It is very easy to find error debut the system.

7 Results

The proposed ResNet50 and ResNet9 models were trained and tested using the New Plant Disease dataset. The dataset was split into training, validation and testing sets with 67K RGB images, respectively, and were labelled with 39 different classes of diseased and healthy plant leaves and background images. These images were taken from the “new plant disease dataset” that was just recently published on Kaggle and are related to plant diseases.

When recreating this dataset using offline augmentation, the original dataset, which was known as the plant village dataset, served as the point of departure. Over eighty-seven thousand RGB photographs of plant leaves, depicting both healthy and diseased conditions, are included in this dataset. These photographs have been organized into a total of 38 different classifications for your viewing pleasure. There are a total of 14 different kinds of plants that are taken into consideration within the parameters of this dataset. The following plants are taken into consideration in this dataset: orange, grape, raspberry, tomato, peach, apple, cherry (including sour), soybean, pepper bell, corn (maize), potato, blueberry, squash, and strawberry. This particular dataset takes into account a total of 26 distinct diseases that can be found in plants.

This research demonstrates conclusively that ResNet50 and ResNet9 may be used to strengthen small-scale farmers with their battle against plant disease. The intensity of crop diseases varies over time; hence, deep learning models must be enhanced/modified to identify and categorize diseases throughout their whole occurrence cycle. The Deep Learning model/architecture must be effective across a variety of light circumstances; hence the datasets must not only represent the actual environment, but also include photographs captured in various field scenarios. A detailed investigation is necessary to comprehend the aspects that influence the identification of plant illnesses, such as the categories and quantity of datasets, the learning rate, as well as the amount of light.

Python programming language is used to perform the comparative study. Keras and Tensorflow libraries are used for building the ResNet50 and ResNet9 models. Experiments carried out have to be evaluated using the success measurement metrics. The metric used in this work is accuracy. It measures the ratio of number of correct predictions made by the model to that of the total number of predictions made by the model. Deep learning algorithms have been used to perform the experiments. The CNN models have been trained and tested using the 'New Plant Village Dataset' dataset taken from the Kaggle website.

ResNet50 model accuracy:

Below screenshot shows the validation accuracy of ResNet50 model with respect to each epoch.

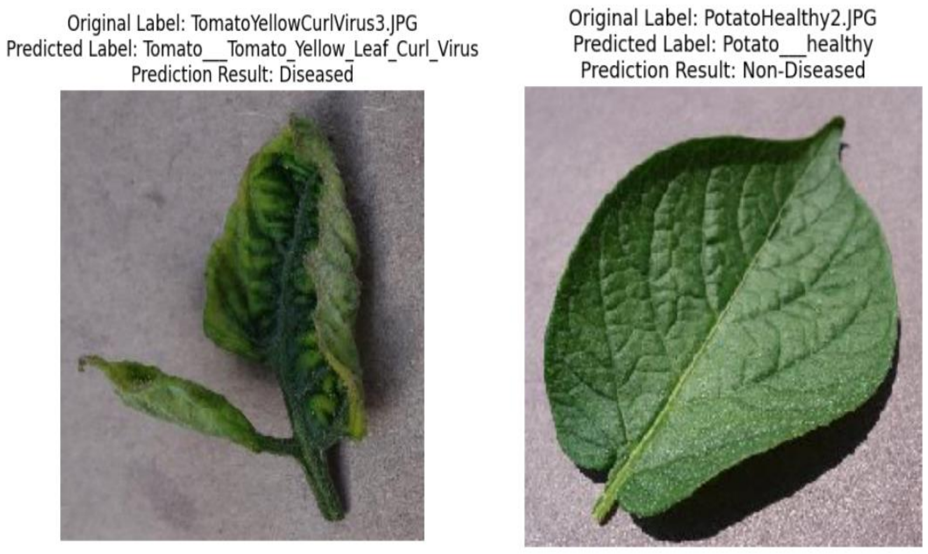
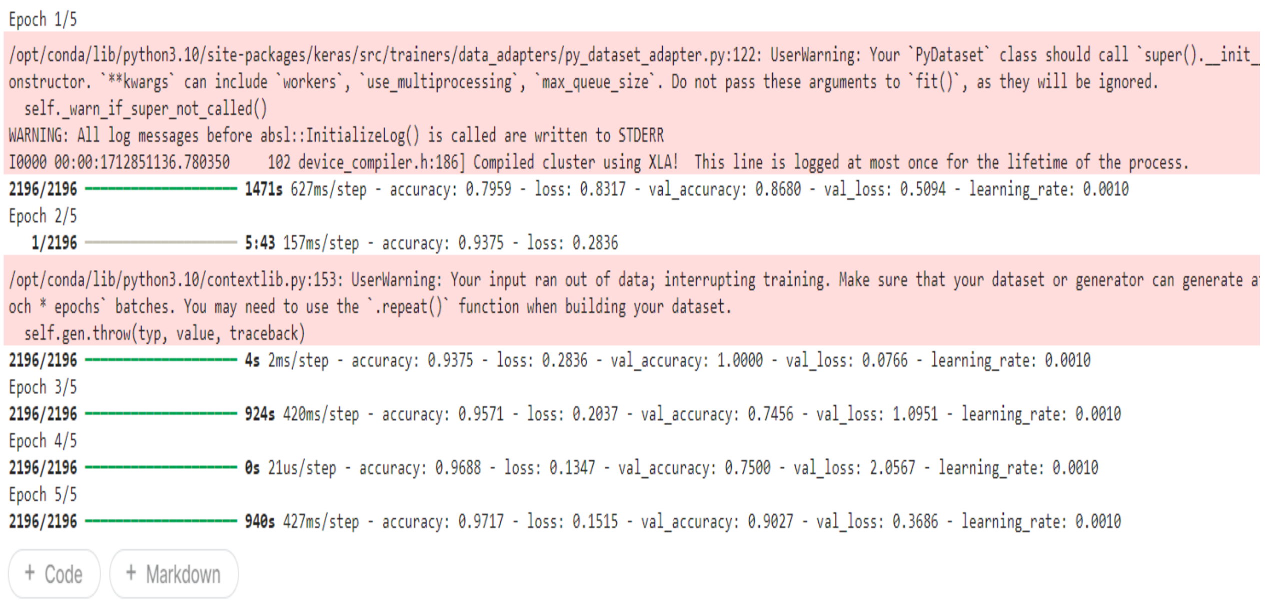

Original Label: TomatoYellowCurlVirus3.JPG
Predicted Label: Tomato___Tomato_Yellow_Leaf_Curl_Virus
Prediction Result: Diseased
Original Label: PotatoHealthy2.JPG
Predicted Label: Potato___healthy
Prediction Result: Non-Diseased

Figure 7.1model detecting diseased leaf and Non-diseased leaves

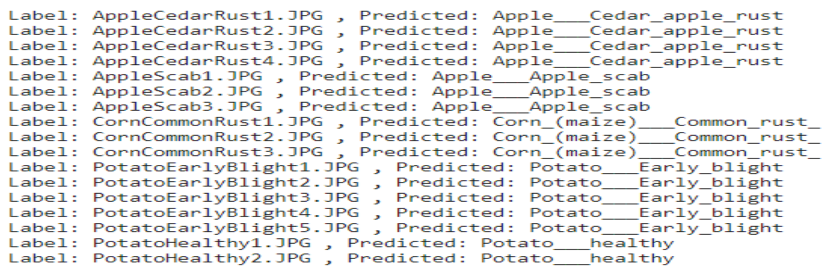


```
Epoch 1/5
/opt/conda/lib/python3.10/site-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:122: UserWarning: Your `PyDataset` class should call `super().__init_
onstructor. `**kwargs` can include `workers`, `use_multiprocessing`, `max_queue_size`. Do not pass these arguments to `fit()`, as they will be ignored.
  self._warn_if_super_not_called()
WARNING: All log messages before absl::InitializeLog() is called are written to STDERR
I0000 00:00:1712851136.780350     102 device_compiler.h:186] Compiled cluster using XLA!  This line is logged at most once for the lifetime of the process.
2196/2196 ━━━━━━━━━━━━━━━━━━━━ 1471s 627ms/step - accuracy: 0.7959 - loss: 0.8317 - val_accuracy: 0.8680 - val_loss: 0.5094 - learning_rate: 0.0010
Epoch 2/5
   1/2196 ━━━━━━━━━━━━━━━━━━━━ 5:43 157ms/step - accuracy: 0.9375 - loss: 0.2836
/opt/conda/lib/python3.10/contextlib.py:153: UserWarning: Your input ran out of data; interrupting training. Make sure that your dataset or generator can generate a
och * epochs` batches. You may need to use the `.repeat()` function when building your dataset.
  self.gen.throw(typ, value, traceback)
2196/2196 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9375 - loss: 0.2836 - val_accuracy: 1.0000 - val_loss: 0.0766 - learning_rate: 0.0010
Epoch 3/5
2196/2196 ━━━━━━━━━━━━━━━━━━━━ 924s 420ms/step - accuracy: 0.9571 - loss: 0.2037 - val_accuracy: 0.7456 - val_loss: 1.0951 - learning_rate: 0.0010
Epoch 4/5
2196/2196 ━━━━━━━━━━━━━━━━━━━━ 0s 21us/step - accuracy: 0.9688 - loss: 0.1347 - val_accuracy: 0.7500 - val_loss: 2.0567 - learning_rate: 0.0010
Epoch 5/5
2196/2196 ━━━━━━━━━━━━━━━━━━━━ 940s 427ms/step - accuracy: 0.9717 - loss: 0.1515 - val_accuracy: 0.9027 - val_loss: 0.3686 - learning_rate: 0.0010
```

Figure 7.2 ResNet50 Result

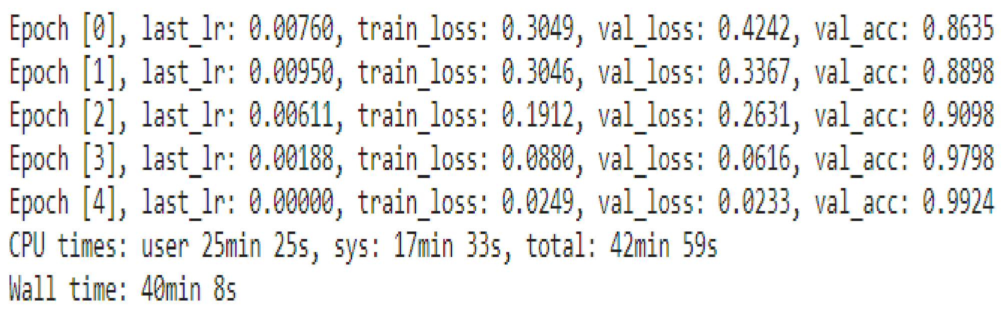
ResNet9 model accuracy:

Below screenshot shows the validation accuracy of of ResNet9 model with respect to each epoch.



```
Label:  AppleCedarRust1.JPG ,  Predicted:  Apple___Cedar_apple_rust
Label:  AppleCedarRust2.JPG ,  Predicted:  Apple___Cedar_apple_rust
Label:  AppleCedarRust3.JPG ,  Predicted:  Apple___Cedar_apple_rust
Label:  AppleCedarRust4.JPG ,  Predicted:  Apple___Cedar_apple_rust
Label:  AppleScab1.JPG ,  Predicted:  Apple___Apple_scab
Label:  AppleScab2.JPG ,  Predicted:  Apple___Apple_scab
Label:  AppleScab3.JPG ,  Predicted:  Apple___Apple_scab
Label:  CornCommonRust1.JPG ,  Predicted:  Corn_(maize)___Common_rust_
Label:  CornCommonRust2.JPG ,  Predicted:  Corn_(maize)___Common_rust_
Label:  CornCommonRust3.JPG ,  Predicted:  Corn_(maize)___Common_rust_
Label:  PotatoEarlyBlight1.JPG ,  Predicted:  Potato___Early_blight
Label:  PotatoEarlyBlight2.JPG ,  Predicted:  Potato___Early_blight
Label:  PotatoEarlyBlight3.JPG ,  Predicted:  Potato___Early_blight
Label:  PotatoEarlyBlight4.JPG ,  Predicted:  Potato___Early_blight
Label:  PotatoEarlyBlight5.JPG ,  Predicted:  Potato___Early_blight
Label:  PotatoHealthy1.JPG ,  Predicted:  Potato___healthy
Label:  PotatoHealthy2.JPG ,  Predicted:  Potato___healthy
```

Figure 7.3 ResNet9 model classifying the leaves



```
Epoch [0], last_lr: 0.00760, train_loss: 0.3049, val_loss: 0.4242, val_acc: 0.8635
Epoch [1], last_lr: 0.00950, train_loss: 0.3046, val_loss: 0.3367, val_acc: 0.8898
Epoch [2], last_lr: 0.00611, train_loss: 0.1912, val_loss: 0.2631, val_acc: 0.9098
Epoch [3], last_lr: 0.00188, train_loss: 0.0880, val_loss: 0.0616, val_acc: 0.9798
Epoch [4], last_lr: 0.00000, train_loss: 0.0249, val_loss: 0.0233, val_acc: 0.9924
CPU times: user 25min 25s, sys: 17min 33s, total: 42min 59s
Wall time: 40min 8s
```

Figure 7.4 Model accuracy for each epoch

Table 7.1 model and their accuracies

S.NO.	MODEL	ACCURACY
1	ResNet9	99.24%
2	ResNet50	97%

Plots:

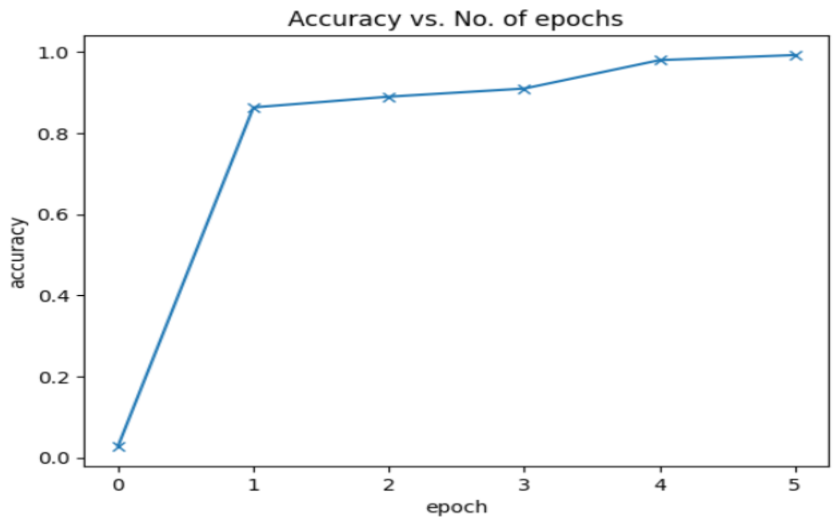


Figure 7.5 epoch vs accuracy plot

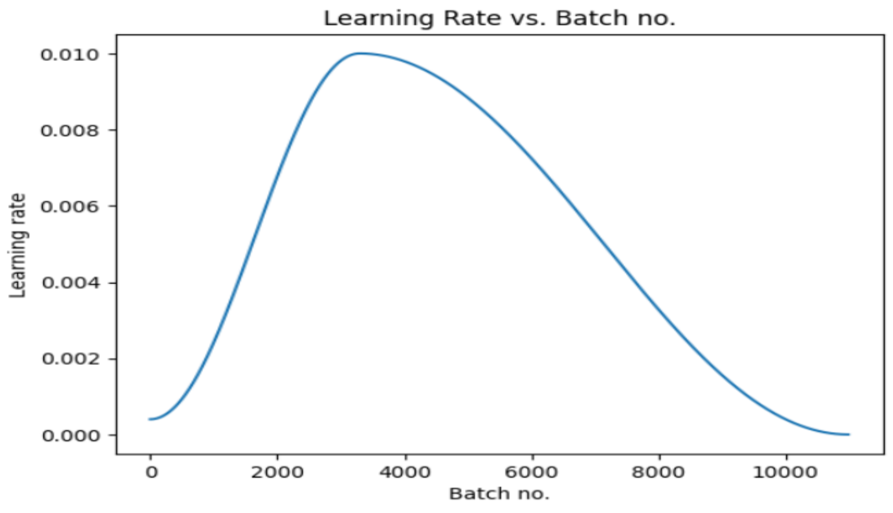


Figure 7.6 batch no. vs learning rate

8 Conclusions and future work

The ResNet50 model proved to be a powerful tool for the detection project, as it leveraged transfer learning from the pre-trained ResNet50 architecture on the ImageNet dataset. By extracting features and fine-tuning the model, it effectively identified and classified disease patterns within the plant images. The utilization of techniques such as global average pooling, custom dense layers, and dropout regularization contributed to the model's robustness and accuracy. The use of early stopping and learning rate scheduling during training further optimized the model's performance. Overall, the ResNet50 model demonstrated its efficacy in detecting and classifying plant diseases, showcasing the potential for real-world agricultural applications.

After thorough experimentation and evaluation, the ResNet9 model has demonstrated its efficacy in accurately classifying plant leaf diseases, leveraging the rich dataset containing augmented images of various plant diseases. The training process, learning rate scheduling, and validation results have been effectively managed and monitored, leading to a well-optimized model. Overall, the ResNet9 model, developed for plant leaf disease classification, exhibits promising potential for real-world applications in agriculture and plant disease management, providing valuable insights for researchers and practitioners in the field.

The results and lessons learned from the comparative pave the way for promising future work. One research direction is to improve and adapt models to address specific challenges related to plant disease detection. This could include hyperparameter optimization, exploring various data augmentation techniques, and exploring transfer learning approaches to increase the adaptability of models to different plant species and disease types. Another area of future work is the scalability and implementation of these models in real agricultural contexts. In addition, by creating intuitive interfaces and mobile applications using these templates, farmers can get easily accessible tools for quickly and efficiently diagnosing diseases. In the future, we aim to refine our project by individually focusing on each plant - tomato, bell pepper, and potato. This would allow for more specialized and accurate disease detection for each plant type.

Looking ahead, there are several avenues for further development and enhancement of this system:

Improved plant disease detection: In this project we have aimed to build and deploy a plant disease detection system that covers the most commonly facing diseases, and this dataset was so huge and it takes longer hours for the model to be trained. Also, we have attained an accuracy of 97.34%, which is remarkable and can be improved by improving the quality of the dataset. And even more types of plant diseases can be added to the dataset for better precision.

Real-Time Processing Enhancements: While the system performs efficiently, there is always room for improvement in processing speed and real-time analysis. Exploring advanced computational techniques or hardware accelerations could yield faster processing times.

User Interface Customization: Further customization options in the user interface, tailored to specific user requirements or industry standards, could enhance the system's usability.

Scalability and Cloud Integration: Enhancing the system's scalability and potentially integrating cloud-based services could facilitate handling larger datasets and enable remote access and analysis.

Integration with IoT and Sensor Data: Incorporating data from IoT devices and sensors, such as soil moisture sensors, weather stations, and crop health monitoring systems, can provide real-time data inputs to the project. This integration can enable more accurate decision-making, predictive analytics, and proactive alerts for farmers regarding irrigation, fertilization, and crop health management

9 References

[1] R. G. de Luna, E. P. Dadios, and A. A. Bandala, "Automated Image Capturing System for Deep Learning based Tomato Plant Leaf Disease Detection and Recognition," IEEE Region 10 Annual International Conference, Proceedings/TENCON, vol. 2018-October, pp. 1414–1419, Feb. 2019, doi:10.1109/TENCON.2018.8650088.

[2] X.-P. Fan, J.-P. Zhou, and Y. Xu, ‘‘Recognition of field maize leaf diseases based on improved regional convolutional neural network,’’ J. South China Agricult. Univ., vol. 41, no. 6, pp. 82–91, Jun. 2020.

[3] P. Jiang, Y. Chen, B. Liu, D. He, and C. Liang, "Real-Time Detection of Apple Leaf Diseases Using Deep Learning Approach Based on Improved Convolutional

Neural Networks," IEEE Access, vol. 7, pp. 59069–59080, 2019, doi:

10.1109/ACCESS.2019.2914929.

[4] P. Soni and R. Chahar, "A segmentation improved robust PNN model for disease identification in different leaf images," 1st IEEE International Conference on Power Electronics, Intelligent Control and Energy Systems, ICPEICES 2016, Feb. 2017, doi:10.1109/ICPEICES.2016.7853301.

[5] Lee, S.H., Chan, C.S., Mayo, S.J. and Remagnino, P., 2017. How deep learning extracts and learns leaf features for plant classification. Pattern Recognition, 71, pp.1-13

[6] D. Venkataraman and N. Mangayarkarasi. Computer vision based feature extraction of leaves for identification of medicinal values of plants. In 2016 IEEE International Conference on Computational Intelligence and Computing Research (ICCIC), pages 1–5, Dec 2016 V. Suma, R. A. Shetty, R. F. Tated, S. Rohan, and T. S. Pujar, "CNN based Leaf Disease Identification and Remedy Recommendation System," Proceedings of the 3rd International Conference on Electronics and Communication and

Aerospace Technology, ICECA 2019, pp. 395– 399, Jun. 2019, doi: 10.1109/ICECA.2019.8821872.

[7] "New Plant Disease." https://www.kaggle.com/datasets/vipoooool/new-plant-diseases-dataset

[8] T. Subetha, R. Khilar, M. C.-M. T. Proceedings, and undefined 2021, "A comparative analysis on plant pathology classification using deep learning architecture–Resnet and VGG19," Elsevier.

[9] S. Vallabhajosyula, V. Sistla, V. K.-J. of P. D. and, and undefined 2022, "Transfer learningbased deep ensemble neural network for plant leaf disease detection," Springer.