CHAPTER 1

INTRODUCTION

1.1 Overview

Plant diseases are one of major reasons behind the production and economic losses in agriculture and identifying disease correctly is a challenging task and requires expertise. Frequently the illnesses or its signs like coloured spots or streaks can be observed on the plant leaves. The plants diseases are usually caused by microbes including fungi, bacteria, and viruses. There is a wide spectrum of signs and symptoms which differ because of the cause or ethology of the plant disease. Plants have become an important source of energy and are a fundamental piece in the puzzle to solve the problem of global warming. There are several diseases that affect plants with the potential to cause economic, social and ecological losses. The detection and classification of leaf diseases accurately is the key to prevent the agriculture loss. Different plant leaf bears different diseases. There are a list of methods and classifiers to detect plant leaf diseases. There are many issues to farmer regarding the plant diseases and many times they do not get proper guidance to detect and cure the disease of plants.

In countries like India it is of utmost importance to bring technological advancement in the fields related to crop productivity. Research initiatives and tentative study process in the important domain of qualitative farming is focused towards improving the yield and food crop standard at low cost, with greater monetary outcome. Agricultural building model stands as a result of a compound interlinking of soil with seeds, and chemicals used to enhance growth. Vegetable and fruits exists as one of the present significant agricultural achieved output. In directive for getting surplus and effective worthy products, a product value examination and improvement has always been importantly imperative. Diseases are disablement to the conventional state of the plant that translates or hinders its important roles such as transpiration, photosynthesis, fertilization, pollination, germination etc. They distorting diseases are spawned by pathogens like, fungi, bacteria and viruses, because of unfavourable environmental situations. Accordingly, the preliminary stage for diagnosing of plant disease is a significant task. Farmers need periodic monitoring by professionals which might be prohibitively costly and time absorbing. Thence, looking for quick, less costly and precise ways to smartly detect the diseases from the indicators that look to be on the plant leaf is of great pragmatic importance. In our study we are proposing a system which can be used to identify the particular type of disease a tomato leaf might have. It is

of major concern to identify the type of disease an important crop like tomato can have, by implementing upbringing technologies like image recognition, which represent the application functioning visually and it is also an important reason for making digital technologies popular.

1.2 Background

The origins and continued evolution of agriculture in the face of infectious diseases and pests Perhaps the greatest technological advance that humans have ever made has been the domestication of plants during the agricultural revolution 8-12,000 years ago at multiple sites around the world. These events created civilizations through the steady, predictable supply of calories that could be obtained with lower and lower amounts of energy expended through human work. The enormous increase in human population from around 1 billion in the early 1800s to our current situation of over 7.2 billion has been made possible by an efficient and productive agricultural basis. However, this steady supply of calories is currently threatened, as many of the advances resulting from the Green Revolution of the 1950s are failing due to infectious diseases and pests spread by globalization and compounded by climate change.

Many histories have recorded the enormous impact that infectious diseases have had on growing food crops (Ainsworth 1981). The iconic example is the Irish Potato famine of 1845-49 where an overdependence on a single crop with little genetic diversity set the stage for a devastating decline in yield that resulted in 1.2 million deaths out of a population of 9 million. At the time, plant diseases were poorly studied and the Irish Potato famine is widely identified as the event that began the scientific discipline of plant disease . It also marks the formal system of extension where science based knowledge is provided to farmers by state employed extension workers. In the intervening 170 years, plant pathology has grown to become a robust discipline offering crop growers multiple solutions from chemical control to genetic engineering to integrated pest management. At the same time, formalized extension systems have likewise developed to provide knowledge to growers.

However, the fact that the global food supply is annually reduced by an average of 40% demonstrates that our collective battle against diseases and pests of crop plants is not won. Indeed, the emergence and spread of novel and highly virulent crop diseases like the stem rust UG99 that attacks wheat, black pod in Cocoa and viral infections of Cassava suggest that the situation may in fact be worsening. This is troubling at a time when the UN FAO recommends we must in fact increase the food supply by 70% to feed the future population.

In high-income countries, where food supply is currently less of a problem, a cursory glance at the media shows an increasingly high level of concern about food production, and many people express unease at the dominance of "Big Ag", the increased focus on growing monocultures (e.g. corn) for both meat production and biofuels, as well as over GMOs . Since the 1980s we have witnessed an increase in smallholder farmers that seek to promote diverse practices and crops. The essence of such movements were captured in Michael Pollan's book, The omnivore's dilemma: a natural history of four meal. Recognizing this, public health institutions are increasingly encouraging consumers to grow their own food, as both the process of growing food (exercise through gardening) and the yield (fresh fruit and vegetables) contribute to beneficial public health outcomes. Indeed, gardening, and urban gardening in particular, has become increasingly popular in the past two decades, reversing a long trend of consumers becoming less and less involved in growing their food. Globally, Google Trends captures this and reports a steady increase of interest in "urban gardening".

Whereas many communities in high-income countries are choosing to grow foods at small scales, such small-scale farming is the norm in poorer parts of the world. This is an imposed necessity rather than a choice. In many countries (e.g. those in Sub-Saharan Africa) as much as 80% of the population are farmers with single families growing diverse crops on small (2-5 hectare) plots with minimal mechanical or chemical (fertilizer, pesticides) inputs. At this scale the relative impact of yield gaps (the gap between potential and actual yield) is very high. A consistent wedge reducing our ability to close this gap are infectious diseases and pests. Since most subsistence agriculture today occurs around the tropics and since the biodiversity of all infectious diseases (of humans, animals and plants) is higher in the tropics then the pressure of diseases are greatest in these areas. It is commonplace for smallholder farmers to routinely lose 80-100% of a given crop to pests and diseases.

1.3 Plant village: a tool for crop health

Plant Village is a platform dedicated to crop health and crop diseases, This platform was modelled after popular online platforms in the computer programming domain, including Stack Overflow , a community driven forum where anyone can ask and answer questions related to programming. By providing answers that are being up-voted by the community, users can build an online reputation, captured by a numeric score. The higher the score, the more rights a user gets on the platform. For example, users need a certain score to be able to vote on other's contributions. An even higher score is needed in order to be able to edit other people's contributions. This model, which has worked very well in many different

contexts, has also been successful in Plant Village, and the platform has seen its traffic grow 250% year over year. In the fall of 2015, the platform welcomed the 2 millionth visitor to the site. In addition to this crowdsourced problem-solving, we have also created a library of open access information on over 150 crops and over 1,800 diseases, accessible on the same website. This content has been written by plant pathology experts, reflecting information sourced from the scientific literature. However, as the site is targeted directly to food growers, rather the professional plant pathologists, great care has been taken to write the content in a way that is easy to understand.

While human-assisted disease diagnosis is a powerful tool, we believe that the potential for machine-assisted disease diagnosis has enormous potential. Disease phenotyping, when done by humans, usually involves a visual analysis of the presentation of the disease on the plant. For some visual phenotypes, disease identification by visual cues is straightforward; for others, it may be more challenging. Nevertheless, the visual diagnosis, if possible, so far requires humans. However, despite the challenge of crop health on an increasingly crowded planet (food security), investment in training plant pathologists has not grown correspondingly, and often even decreased.

CHAPTER 2

LITERATURE SURVEY

[1]. Shima Ramesh, Mr. Ramachandra Hebba, Mr. P V Vino, Niveditha M, Pooja R, Prasad Bhat N, Shashank N, "Plant Disease Detection Using Machine Learning" 2018.

Crop diseases are a noteworthy risk to sustenance security, however their quick distinguishing proof stays troublesome in numerous parts of the world because of the non-attendance of the important foundation. Emergence of accurate techniques in the field of leaf-based image classification has shown impressive results. This paper makes use of Random Forest in identifying between healthy and diseased leaf from the data sets created. Our proposed paper includes various phases of implementation namely dataset creation, feature extraction, training the classifier and classification. The created datasets of diseased and healthy leaves are collectively trained under Random Forest to classify the diseased and healthy images. For extracting features of an image, they have used Histogram of an Oriented Gradient (HOG). Overall, using machine learning to train the large data sets available publicly gives us a clear way to detect the disease present in plants in a colossal scale. First for any image we need to convert RGB image into gray scale image. This is done just because Hu moments shape descriptor and Haralick features can be calculated over single channel only. Therefore, it is necessary to convert RGB to gray scale before computing Hu moments and Haralick features. As depicted in the figure 4. To calculate histogram the image first must be converted to HSV (hue, saturation and value), so we are converting RGB image to an HSV image as shown the figure5. Finally, the main aim of our project is to detect whether it is diseased or healthy leaf with the help of a Random Forest classifier. The objective of this algorithm is to recognize abnormalities that occur on plants in their greenhouses or natural environment. The image captured is usually taken with a plain background to eliminate occlusion. The algorithm was contrasted with other machine learning models for accuracy. Using Random Forest classifier, the model was trained using 160 images of papaya leaves. The model could classify with approximate 70 percent accuracy. The accuracy can be increased when trained with vast number of images and by using other local features together with the global features such as SIFT (Scale Invariant Feature Transform), SURF (Speed Up Robust Features).

[2]. Akshita Arora, Nazim Khan, Shefali Gupta, Samikchha Singh Maansi Gupta, "Leaf Disease Detection Using CNN and Raspberry PI", 2019.

Leaf disease detection requires huge amount of work, knowledge in the plant field. So we can use image processing for identification of leaf disease using PYTHON. This paper holds leaf diseases detection using CNN and Image Processing. It has three basic steps. Preprocessing the dataset, Training it, Validation and Testing the model using raspberry pi camera connected through raspberry pi 3. Revealing the CNN to extract the learned feature as an interpretable form not only ensures its reliability but also enables the validation of the model authenticity and the training dataset by human intervention. In this study, a variety of neuron-wise and layer-wise visualization methods were applied using a CNN, trained with a publicly available plant disease image dataset. This paper showed that neural networks can capture the textures of lesions specific to respective diseases upon diagnosis, which resembles human decision making. While several visualization methods were used as they are, others had to be optimized to target a specific layer that fully captures the features to generate consequential outputs. Moreover, by interpreting the generated attention maps, we identified several layers that were not contributing to inference and removed such layers inside the network, decreasing the number of parameters without affecting the classification accuracy. The results provide an impetus for the CNN users in the field of plant science to better understand the diagnosis process and lead to further efficient use of deep learning for plant disease diagnosis. There are essentially three fundamental kinds of Leaf ailment, Bacterial, Fungal and Viral. The precision of plant ailment recognition is essential in plant ailment location, however the procedure ought to be rapid in the meantime. The aim of this paper is to help the farmers to protect his farm from any kind of pests and disease attacks and eliminate them without disturbing the decorum of the soil and untouched parts of other plants. Mostly in India, farmersuse manual monitoring and some apps which have huge database limitations and are only bound to detection part. Since, Prevention is better than cure, this paper aims at detection of attack of pests/diseases in future thereby making farmer to prevent such attacks.

[3]. Santhosh Kumar S, and B. K. Raghavendra, "Diseases Detection of Various Plant LeafUsing Image Processing Techniques: A Review", 2019.

India's agriculture is composed of many crops and according to survey nearly 70% population is depends on agriculture. Most of Indian farmers are adopting manual cultivation due to lagging of technical knowledge. Farmers are unaware of what kind of

crops that grows well on their land. When plants are affected by heterogeneous diseases through their leaves that will effects on production of agriculture and profitable loss. Also reduction in both quality and amount of agricultural production. Leaves are important for fast growing of plant and to increase production of crops. Identifying diseases in plants leave is challenging for farmers also for researchers. Currently farmers are spraying pesticides to the plants but it effects human directly or indirectly by health or also economically. To detect these plant diseases many fast techniques need to be adopt. In this paper, they have done survey on different plants disease and various advance techniques to detect these diseases. Literature survey has detailed explanation of the importance of disease detection both to plants and to mankind. To have a meaningful impact of plant diseases & techniques in the area of agriculture, deliberation of proper input is necessary. Research issues addressed here are to develop a systematic approach to detect and recognize the plant diseases would assist farmers and pathologist in prospect exploration. The paper depicts the importance of image processingin agriculture field and considering the type of disease for further research work.

[4]. Huu Quan Cap, Katsumasa Suwa, Erika Fujita, Satoshi Kagiwada, Hiroyuki Uga, and Hitoshi Iyatomi, "A deep learning approach for on-site plant leaf detection", 2018.

This paper presents a simple and accurate leaf regions detection system with high affinity withother existing disease diagnosis systems. The objective of this system is to localize the "fully leaf" part from the input image. For clarity, the definition of "fully leaf", "not fully leaf", and "none leaf" .This paper confirmed that the performance of 78.0% in F1-score is sufficiently acceptable for this task from visual assessment. Plant diseases are the major problem in the worldwide agriculture sector. Therefore, the early detection is essential for reducing economic losses and mitigating the seriousness of the global food problem. Some fast and accurate computer-based methods have been applied to detect plant diseases. However, as far as our best knowledge, all those methodologies only accept a narrow range image, typically one or limited number of target(s) are in the image frame as their input. Thus, they are time-consuming and difficult to be applied for on site wide range images (e.g. images or videos from stationary surveillance camera). In this paper, they propose leaf localization method from on-site wide-angle images with a deep learning approach. This method achieves a detection performance of 78.0% in F1-measure at 2.0 fps.

[5]. Abirami Devaraj, Karunya Rathan, Sarvepalli Jaahnavi, and K Indira, "Identificationof Plant Disease using Image Processing Technique", 2019.

The present study deals with Alternaria Alternate, Anthracnose, Bacterial Blight and Cercosporin Leaf Spot this automatic illness detection using image processing techniques in MATLAB. It involves loading an image, image pre-processing, image segmentation, feature extraction and classification. Development of automatic detection system using advanced technology like image process facilitate to support the farmers within the identification of diseases at an early or initial stage and supply helpful data for its management. we might preferto extend our work additional on a lot of disease detection. The MATLAB image processing starts with capturing of digital high resolution pictures. Healthy and unhealthy pictures are captured and hold on for experiment. Then pictures are applied for pre-processing for image improvement. Captured leaf pictures are segmented using K-means cluster technique to create clusters. Options are extracted before applying K-means and Random Forest Classifier for training and classification. Finally diseases are recognized by this method. In this paper section one provides an introduction and importance of disease detection. Section two provides a plant disease fundamental. Section three quick literature reviews of leaf illness detection techniques. Section four describes methodology of projected system based on MATLAB image processing. Section five provides results and discussion. Section six concludes this paper along with future work.

[6]. Sharath N Payyadi, Varun S D, Satya Gururaj Kalluru, Archana R Kulkarni, "DiseaseDetection in Paddy Crop using CNN Algorithm", 2020.

This research presents a system which detects the diseases like blast and blight in paddy crop as early as possible using CNN algorithm. The Raspberry-Pi is used as the portable computer.The keras model is trained with four hundred images dataset of diseases leaf image and loadedinto Raspberry-Pi. The leaf image on the field is captured using Pi-camera and later processedby python script. The keras model is used to predict the disease in captured image. In this paper, we propose a technique for early detection of the diseases like blast and blight in paddy crop using one of the deep learning algorithms called as CNN algorithm. The entire system works on the Raspberry-pi which acts as processing unit for the proposed system. The well trained keras model to predict the disease in paddy leaf is developed using CNN algorithm. As the symptoms of the blast and blight disease are detected using feature of the paddy leaf, so the model is trained using the diseased image data set. The pi-camera will capture the image of the leaf and the captured image

will be processed by the python script running on raspberry pi. The images will be captured at different angle with respect to raspberry-pi. The trained model will predict whether the processed image of the leaf contains the symptoms of the disease. The proposed system also includes the feature which will alert the farmer about spreading of the disease by using the GSM module to send the message to the farmer of the field. Also the pesticides are suggested to farmers to control the further growth of the germs causing the disease. In India paddy is grown as the staple crop. The paddy crop is mostly damaged due to the leaf diseases called as blast and blight disease. About 20%-30% of the yield will be destroyed by the blast and blight diseases respectively. Hence, by implementing the above proposed system the disease can be detected at the early stage and loss can decreased.

CHAPTER 3

PROBLEM DEFINITION

3.1 Objectives

Leaves of a plant can be used to determine the health status of that plant. The objective of thiswork is to develop a system that capable to detect and identify the type of disease using Machine Learning. Plant diseases are one of source of obstruction in the quality and productivity of plants which can lead to the shortage of food supply. Therefore, plant disease detection is essential to the agriculture industry. The objective of this research is to classify, detect and give cures for the plant diseases by assessing the images of the leaves with the application of Machine Learning, a Machine Learning classification algorithm with a single layer feed-forward neural network. A plant disease prevents a plant from reaching its maximumpotential of production. This definition includes non-infectious and infectious diseases that pose threat to the agriculture industry by causing a decline in production and economic as well as reduction in the quality and the quantity of the plant products.

3.2 Aim of the Project

The objective is to distinguish different plant infection by checking picture out. We are using deep learning for this project because here we are working with image data. Deep learning has a Convolution neural network that is used to find features from the leaf of the plant. By utilizing CNN Algorithm we can identify the plant disease precisely.

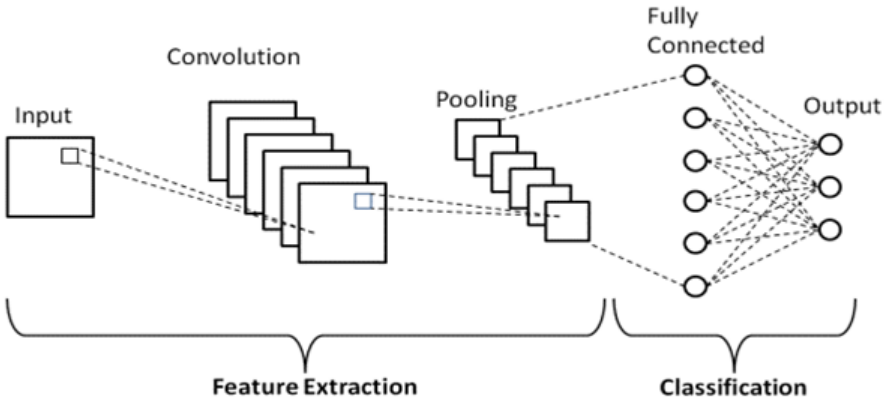

Fig.3.2 CNN architecture

The objective is to distinguish different plant disease by checking picture out.

In the web application:

1. Client should be able to upload an image of an infected plant leaf from there device.
2. CNN Model should be able to detect plant infection.
3. Finally client will get the plant disease name and also some suggestions on preventing the disease and it also suggests some supplements and fertilizers.

CHAPTER 4

SYSTEM DESIGN

4.1 Dataset

The Dataset was taken from Kaggle of Plant Village dataset existing online as such the code was also inscribed on the online kernel of Kaggle for well computation and study of training loss and validation.

4.1.1 Data Records

The data records contain 54,309 images. The images span 14 crop species: Apple, Blueberry, Cherry, Corn, Grape, Orange, Peach, Bell Pepper, Potato, Raspberry, Soybean, Squash, Strawberry, Tomato. In containes images of 17 fungal diseases, 4 bacterial diseases, 2 mold (oomycete) diseases, and 1 disease caused by a mite. 12 crop species also have images of healthy leaves that are not visibly affected by a disease. Table 4.1.1 summarizes the dataset.

	Fungi	Bacteria	Mold	Mite	Healthy
Apple (3172)	Gymnosporangium-juniperi0virginianae-(276) Venturia-insequalis-(630)-Botryospaeria-obtuse-(621)				(1645)
Blueberry (1502)					(1502)
Cherry (1906)	Podosphaera-spp-(1052)				(854)
Corn (3852)	Cercospora-zeae0maydis (513) Puccinia sorghi (1192) Exserohilum turcicum(985)				(1162)

Grape (4063)	Guignardia bidwellii (1180) Phaeomoniella spp.(1384)Pseud ocercospora vitis-(1076				(423)
Orange (5507)		Candidatus-Liberibacter-(5507)			
Peach (2657)		Xanthomona s-campestris-(2291)			(360)
Bell Pepper (2475)		Xanthomona s-campestris-(997)			(1478)
Potato (2152)	Alternaria solani (1000)		Phytopht hora-Infestans-(1000)		(152)
Raspberry (371)					(371)
Soybean (5090)					(5090)
Squash (1835)	Erysiphe-cichoracearum / Sphaerotheca fuliginea(1835)				(1835)
Strawberry (1565)	Diplocarpon earlianum(1109)				(456)
Tomato(18,162)	Alternaria solani (1000) Septorialycopersici (1771)	Xanthomona s-campestris-pv.-Vesicatoria-(2127)!	Phytopht hora-Infestans-(1910)	Tetrany chus-urticae-(1676)	(1592)

	Corynespora cassiicola (1404) Fulvia-fulva- (952)				

Table.4.1.1 List of crops and their disease status in plant village

4.2 Image Pre-processing and Labelling

Pre-processing pictures generally includes eliminating low-recurrence foundation commotion, normalizing the power of the individual particles' pictures, eliminating reflections, and veiling segments of pictures. Picture pre-processing is the strategy of improving information Furthermore, the strategy of picture pre-processing included editing of the apparent multitude of pictures physically, making the square around the leaves, to feature the district of intrigue (plant leaves). During the period of gathering the pictures for the dataset, pictures with a more modest goal and measurement not exactly 500 pixels were not considered as substantial pictures for the dataset. In expansion, just the pictures where the locale of intrigue was in higher the goal was set apart as a qualified possibility for the dataset. In that manner, it was guaranteed that pictures contain all the required data for highlight learning. Numerous assets can be found via looking over the Internet, in any case, their significance is frequently problematic. Considering a legitimate concern for affirming the exactness of classes in the dataset, at first assembled by a catchphrases search, horticultural specialists inspected leaf pictures and marked all the pictures with fitting infection abbreviations. As it is known, it is significant to utilize precisely characterized pictures for the preparation and approval dataset. Just in that manner may a fitting and solid identifying model be created. In this stage, copied pictures that were left after the underlying the emphasis of get-together and gathering pictures into classes were eliminated from the dataset.

Neural Network Training:- Preparing the profound convolutional neural organization for taking a picture grouping model from a dataset was proposed. Tensor Flow is an open-source programming library for mathematical calculation utilizing information stream diagram.

The adaptable engineering permits you to convey calculation to at least one CPUs or GPUs in a work area, worker, or on the other hand cell phone with a solitary API. Tensor Flow was initially created by scientists and designers dealing with the Google Brain Group inside Google's Machine Intelligence research association for the motivations behind directing AI

and profound neural organizations research, yet the framework is sufficiently general to be material in a wide assortment of different spaces also. In AI, a convolutional neural organization is a sort of feed-forward fake neural organization in which the network design between its neurons is roused by the association of the creature visual cortex. Individual cortical neurons react to improvements in a limited district of room known as the responsive field. The responsive fields of various neurons incompletely cover with the end goal that they tile the visual field. The reaction of an individual neuron to boosts inside its open field can be approximated numerically by a convolution activity. Convolutional networks were propelled by natural cycles and are varieties of multilayer perceptron intended to utilize negligible measures of pre-handling. They have wide applications in picture and video acknowledgment, recommender frameworks and regular language preparing. Convolutional neural organizations (CNNs) comprise of various layers of responsive fields. These are little neuron assortments which cycle parts of the info picture. The yields of these assortments are then tiled so their info districts cover, to get a higher-goal portrayal of the first picture; this is rehashed for each such layer. Tiling permits CNNs to endure interpretation of the information picture. Convolutional organizations may incorporate neighbourhood or worldwide pooling layers, which consolidate the yields of neuron bunches. They too comprise of different mixes of convolutional and completely associated layers, with point savvy nonlinearity applied toward the finish of or after each layer. A convolution procedure on little locales of info is acquainted with lessen the quantity of free boundaries and improve speculation. One significant bit of leeway of convolutional networks is the utilization of shared weight in convolutional layers, which implies that a similar channel (loads bank) is utilized for every pixel in the layer; this both lessens memory impression furthermore, improves execution. The layer's boundaries are involved a set of learnable portions which have a little open field yet stretch out through the full profundity of the info volume. Amended Linear Units (Re LU) are utilized an alternative for soaking nonlinearities. This initiation work adaptively learns the boundaries of rectifiers and improves precision at unimportant extra computational expense. In the unique situation of fake neural organizations, the rectifier is an enactment work characterized as:f(x)=max (0, x) where x is the contribution to a neuron.

This enactment work was first acquainted with a dynamical organization by Hahn failure et al. in a 2000 paper in Nature with solid organic inspirations what is more, numerical avocations. It has been utilized in convolutional networks more successfully than the generally utilized strategic sigmoid (which is roused by likelihood hypothesis; see strategic

relapse) and its more down to earth partner, the exaggerated digression. The rectifier is, starting at 2015, the most famous actuation work for profound neural organizations. Profound CNN with ReLU's trains a few times quicker. This technique is applied to the yield of each convolutional and completely associated layer. Despite the yield, the information standardization is not needed; it is applied after ReLU nonlinearity after the first and second convolutional layer since it lessens top-1 and top-5 mistake rates. In CNN, neurons inside a covered-up layer are fragmented into "include maps." The neurons inside an element map share a similar weight and inclination. The neurons inside the component map look for a similar component. These neurons are exceptional since they are associated with various neurons in the lower layer. So, for the first concealed layer, neurons inside an element guide will be associated with various districts of the info picture. The shrouded layer is portioned into highlight maps where every neuron in an element map searches for a similar component in any case, at various places of the information picture. Essentially, the component map is the aftereffect of applying convolution over a picture. The convolutional layer is the centre structure square of a CNN. The layer's boundaries comprise of a lot of learnable channels (or pieces), which have a little open field, yet reach out through the full profundity of the information volume. During the forward pass, each channel is convolved over the width and tallness of the input volume, registering the speck item between the sections of the channel also, the info and creating a 2-dimensional actuation guide of that channel. Therefore, the organization learns channels that initiate when it recognizes some sort of highlight at some spatial situation in the info. Stacking the actuation maps for all channels along the profundity measurement structures the full yield volume of the convolution layer. Three hyper parameters control the size of the output volume of the convolutional layer: the depth, stride, and zero-padding.

Depth of the yield volume controls the quantity of neurons in the layer that interface with a similar district of the information volume. These neurons will figure out how to initiate for various highlights in the information. For model, if the primary Convolutional Layer accepts the crude picture as info, at that point various neurons along the profundity measurement may initiate in the presence of different arranged edges, or masses of shading.

Stride: It controls how depth columns around the spatial dimensions (width and height) are allocated. When the stride is 1, a new depth column of neurons is allocated to spatial positions only 1 spatial unit apart. This leads to heavily overlapping receptive fields between the columns, and to large output volumes. Conversely, if higher strides are used

then the receptive fields will overlap less and the resulting output volume will have smaller dimensions spatially.

Zero padding happens as we include a border of pixels each with value zero across the boundaries of the input pictures. This adds sort of a padding of zeros across the outside of the picture, hence the name zero padding.

4.3 System Architecture

To diagnose the reason for the symptom by using an automatic tool, therefore the image processing system is proposed to develop to automate the identification and classification of the leaf batches into specific disorders. As shown within the figure above the system consists of three main blocks: Image Analyzer, Feature Database and Classifier resp. [9]. The processing proposed to try to by these blocks is split into two phases as follows offline Phase: an outsized set of defected images are processed by a picture analyser for extracting abnormal features.

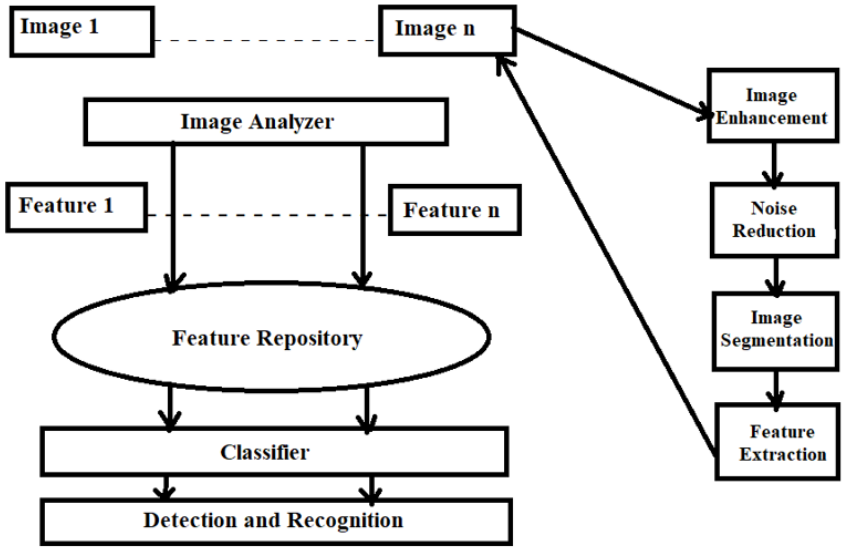

Fig.4.3 System Architecture

4.4 Flow Chart

The input test image is developed and pre-processed in the following phase and then it is transformed into array form for difference. The chosen database is appropriately separated and pre-processed and then retitled into suitable folders. The model is well trained using CNN and then classification takes position. The evaluation of the test image and the trained model take position tracked by the display of the result. If there is a flaw or infection in the plant the package displays the disease along with the remedy.

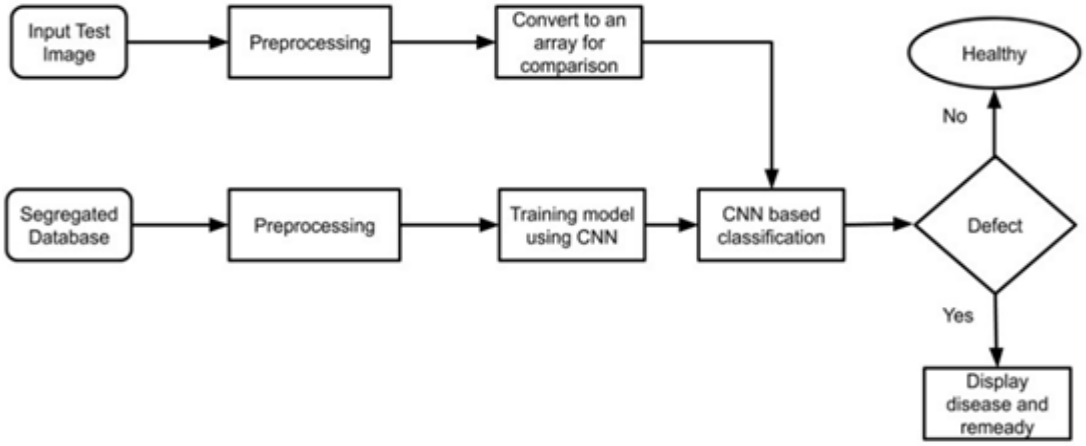

Fig.4.4 Flow Chart

4.5 Use Case Diagram

When we provide a new input image first the module extracts the leaf features. Then it goes through the CNN model. It then compares the features with already trained dataset. Then it goes through dense CNN and therefore the leaf features are extracted separately. Then the module will predict whether the plant leaf is affected by any disease or not. It shows the output from one among the38 classes which are predetermined and trained. Then the output is going to be during a textual format.

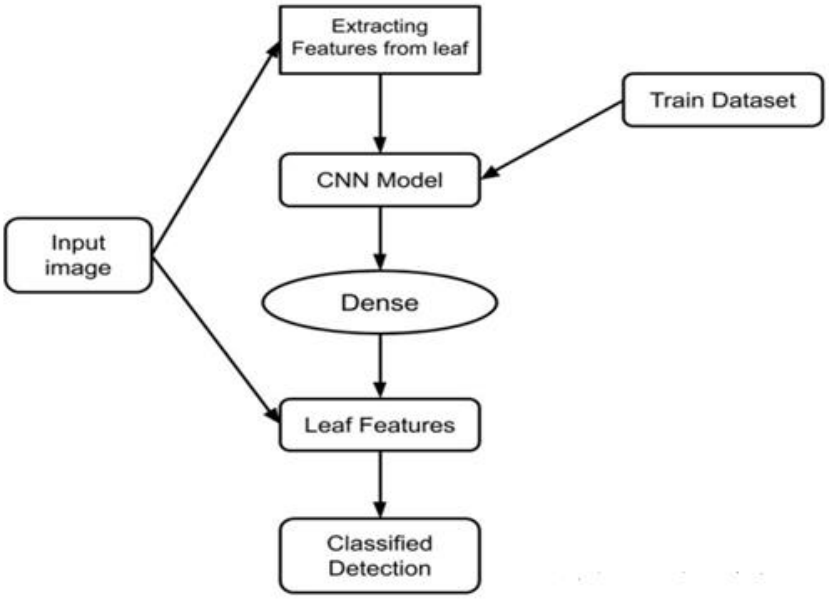

Fig.4.5 Use Case Diagram

4.6 Class Diagram

The normalization class comprises raw images and it is fed to the CNN model which comprises dense and weight. The CNN model categorizes and identifies by using the training model. The training model class contains the image dataset. Leaf recognition becomes utilized of the features.

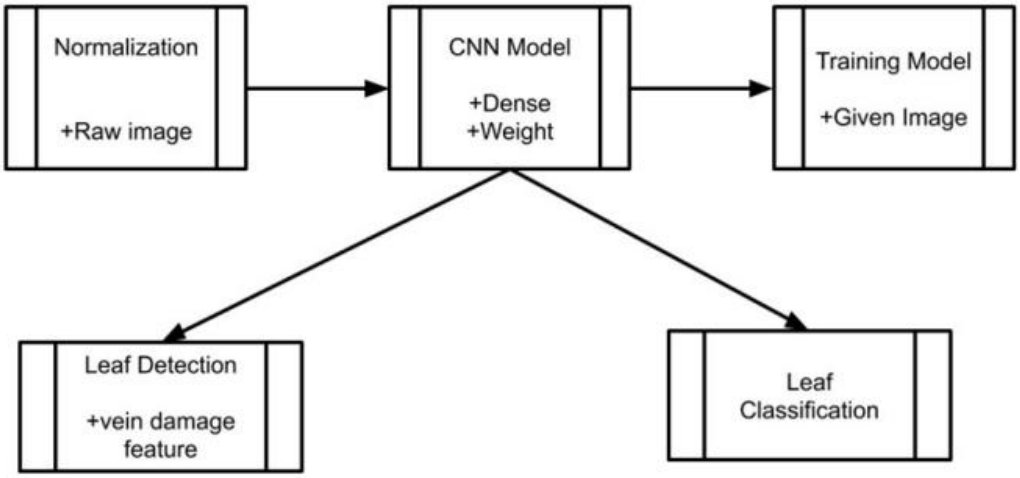

Fig.4.6 Class Diagram

Flatten In between the convolutional layer and therefore the fully connected layer, there is a 'Flatten' layer. Flattening transforms a two-dimensional matrix of features into a vector which will be fed into a totally connected neural network classifier. Image Data Generator: Image Data Generator quickly found out Python generators which will automatically turn image files on disk into batches of pre-processed tensors.

Training Process Effective training begins well before a trainer delivers a private training session and continues then training session is complete. Training is often viewed as a process comprised of 5 related stages or activities: assessment, motivation, design, delivery, and evaluation. Epochs: An epoch may be a term utilized in machine learning and indicates the amount of passes of the whole training dataset the machine learning algorithm has completed. Datasets are usually grouped into batches (especially when the quantity of knowledge is extremely large). Validation

Process Validation is mentioned because the process where a trained model is evaluated with a testing data set. The testing data set may be a separate portion of an equivalent data set from which the training set springs. the most purpose of using the testing data set is to check the generalization ability of a trained model.

4.7 Training and Testing Model

The dataset is pre-processed like Image reshaping, resizing and conversion to an array form. Similar processing is additionally done on the test image. A dataset consisting of about 38 different plant leaf diseases is obtained, out of which any image is often used as a test image for the software. The train dataset is employed to coach the model (CNN) so that it can identify the test image and therefore the disease it is CNN has different layers that are Dense, Dropout, Activation, Flatten, Convolution2D, and maxpooling2d. After the model is trained successfully, the software can identify the disease if the plant species is contained within the dataset. After successful training and pre-processing, comparison of the test image and trained model takes place to predict the disease.

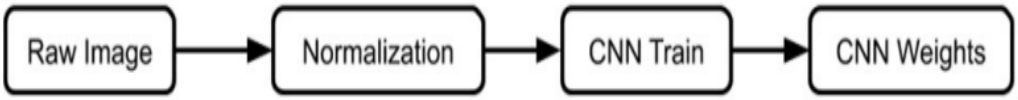

Fig.4.7.1 Training Model

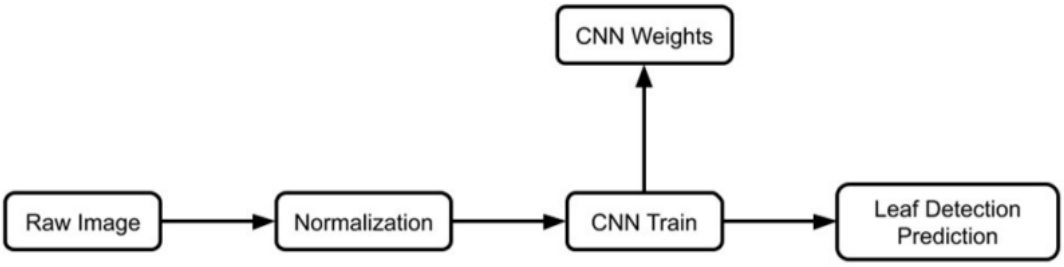

Fig.4.7.2 Testing Model

4.8 Sequence Diagram

Sequence diagrams are sometimes called event diagrams or event scenarios. A sequence diagram shows as parallel vertical lines (lifelines), different processes or objects that live simultaneously and as horizontal arrows, the messages exchanged between them in the order in which they occur.

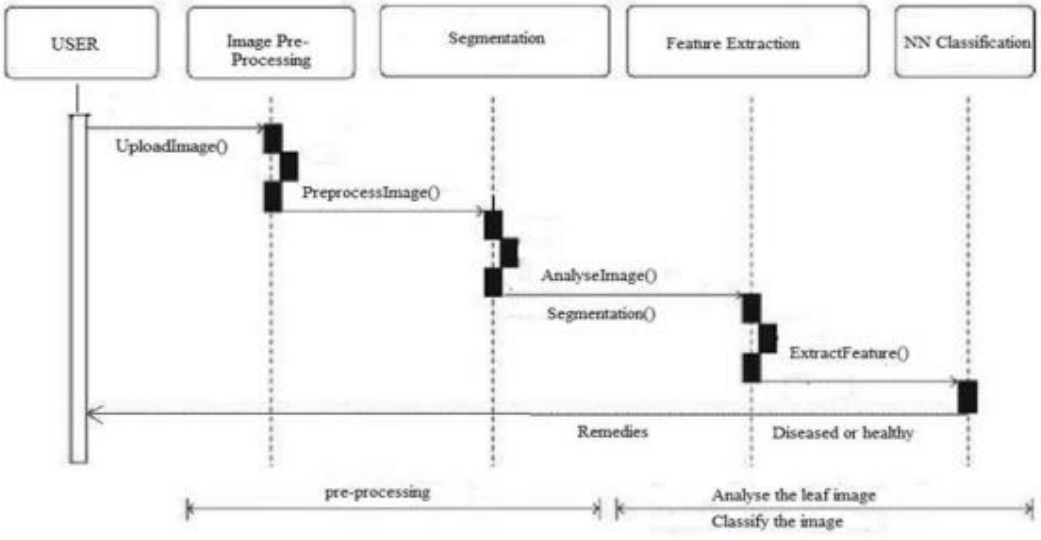

Fig.4.8. Sequence diagram

- **Usage scenarios**:

  A usage scenario is a description of a potential way your system is used. The logic of a usage scenario may be part of a use case, perhaps an alternate course. It may also be one entire pass through a use case, such as the logic described by the basic course of action or a portion of the basic course of action, plus one or more alternate scenarios. The logic of a usage scenario may also be a pass through the logic contained in several use cases.

- **The logic of methods:**

  Sequence diagrams can be used to explore the logic of a complex operation, function, or procedure. One way to think of sequence diagrams, particularly highly detailed diagrams, is as visual object code

- **The logic of services:**

  A service is effectively a high-level method, often one that can be invoked by a wide variety of clients. This includes web-services as well as business transactions implemented by a variety of technologies

CHAPTER 5

REQUIREMENTS

5.1 Software Requirement

5.1.1 Jupyter Notebook

Jupyter Notebook can connect to many kernels to allow programming in different languages. A Jupyter kernel is a program responsible for handling various types of requests (code execution, code completions, inspection), and providing a reply. Kernels talk to the other components of Jupyter using ZeroMQ, and thus can be on the same or remote machines. Unlike many other Notebook-like interfaces, in Jupyter, kernels are not aware that they are attached to a specific document, and can be connected to many clients at once. Usually kernels allow execution of only a single language, but there are a couple of exceptions. By default Jupyter Notebook ships with the IPython kernel. There are 49 Jupyter-compatible kernels for many programming languages, including Python, R, Julia and Haskell. A Jupyter Notebook can be converted to a number of open standard output formats (HTML, presentation slides, LaTeX, PDF, ReStructuredText, Markdown, Python) through "Download As" in the web interface, via the nbconvert library or "jupyter nbconvert" command line interface in a shell. To simplify visualisation of Jupyter notebook documents on the web, the nbconvert library is provided as a service through NbViewer which can take a URL to any publicly available notebook document, convert it to HTML on the fly and display it to the user.

The notebook interface was added to IPython in the 0.12 release (December 2011), renamed to Jupyter notebook in 2015 (IPython 4.0 is Jupyter 1.0). Jupyter Notebook is similar to the notebook interface of other programs such as Maple, Mathematica, and SageMath, a computational interface style that originated with Mathematica in the 1980s. Jupyter interest overtook the popularity of the Mathematica notebook interface in early 2018. JupyterLab is a newer user interface for Project Jupyter. It offers the building blocks of the classic Jupyter Notebook (notebook, terminal, text editor, file browser, rich outputs, etc.) in a flexible user interface. The first stable release was announced on February 20, 2018.

5.1.2 Python

Python 3 is a newer version of the Python programming language which was released in December 2008. This version was mainly released to fix problems that exist in Python 2.

The nature of these changes is such that Python 3 was incompatible with Python 2. It is **backward incompatible**.

Some features of Python 3 have been backported to Python 2.x versions to make the migration process easy in Python 3. As a result, for any organization who was using Python 2.x version, migrating their project to 3.x needed lots of changes. These changes not only relate to projects and applications but also all the libraries that form part of the Python ecosystem. Python is a general-purpose interpreted, interactive, object-oriented, and high-level programming language. It was created by Guido van Rossum during 1985- 1990. Like Perl, Python source code is also available under the GNU General Public License (GPL). Python is named after a TV Show called ëMonty Pythonís Flying Circusí and not after Python-the snake.

Python 3.0 was released in 2008. Although this version is supposed to be backward incompatibles, later on many of its important features have been backported to be compatible with version 2.7. This tutorial gives enough understanding on Python 3 version programming language. Please refer to this link for our Python 2 tutorial.

Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages.

5.1.3 HTML and CSS

The Hyper Text Markup Language or HTML is the standard markup language for documents designed to be displayed in a web browser. It can be assisted by technologies such as Cascading Style Sheets (CSS) and scripting languages such as JavaScript.

Web browsers receive HTML documents from a web server or from local storage and render the documents into multimedia web pages. HTML describes the structure of a web page semantically and originally included cues for the appearance of the document.HTML elements are the building blocks of HTML pages. With HTML constructs, images and other objects such as interactive forms may be embedded into the rendered page. HTML elements are delineated by tags, written using angle brackets. Browsers do not display the HTML tags but use them to interpret the content of the page.

HTML can embed programs written in a scripting language such as JavaScript, which affects the behavior and content of web pages. Inclusion of CSS defines the look and layout of content. The World Wide Web Consortium (W3C), former maintainer of the HTML and current maintainer of the CSS standards, has encouraged the use of CSS over explicit

presentational HTML since 1997. A form of HTML, known as HTML5, is used to display video and audio.

HTML (the Hypertext Markup Language) and CSS (Cascading Style Sheets) are two of the core technologies for building Web pages. HTML provides the *structure* of the page, CSS the (visual and aural) *layout,* for a variety of devices. Along with graphics and scripting, HTML and CSS are the basis of building Web pages and Web Applications.

CSS is the language for describing the presentation of Web pages, including colors, layout, and fonts. It allows one to adapt the presentation to different types of devices, such as large screens, small screens, or printers. CSS is independent of HTML and can be used with any XML-based markup language. The separation of HTML from CSS makes it easier to maintain sites, share style sheets across pages, and tailor pages to different environments. This is referred to as the *separation of structure (or: content) from presentation.* Cascading Style Sheets (CSS) is a style sheet language used for describing the presentation of a document written in a markup language such as HTML or XML (including XML dialects such as SVG, MathML or XHTML). CSS is a cornerstone technology of the World Wide Web, alongside HTML and JavaScript.

CSS is designed to enable the separation of presentation and content, including layout, colors, and fonts. This separation can improve content accessibility; provide more flexibility and control in the specification of presentation characteristics; enable multiple web pages to share formatting by specifying the relevant CSS in a separate .css file, which reduces complexity and repetition in the structural content; and enable the .css file to be cached to improve the page load speed between the pages that share the file and its formatting.

Separation of formatting and content also makes it feasible to present the same markup page in different styles for different rendering methods, such as on-screen, in print, by voice (via speech-based browser or screen reader), and on Braille-based tactile devices. CSS also has rules for alternate formatting if the content is accessed on a mobile device.

The name cascading comes from the specified priority scheme to determine which style rule applies if more than one rule matches a particular element. This cascading priority scheme is predictable.

The CSS specifications are maintained by the World Wide Web Consortium (W3C). Internet media type (MIME type) is registered for use with CSS by RFC 2318 (March 1998). The W3C operates a free CSS validation service for CSS documents.

5.1.4 MySQL

MySQL is an open-source relational database management system (RDBMS). Its name is a combination of "My", the name of co-founder Michael Widenius's daughter, and "SQL", the abbreviation for Structured Query Language. A relational database organizes data into one or more data tables in which data may be related to each other; these relations help structure the data. SQL is a language programmers use to create, modify and extract data from the relational database, as well as control user access to the database. In addition to relational databases and SQL, an RDBMS like MySQL works with an operating system to implement a relational database in a computer's storage system, manages users, allows for network access and facilitates testing database integrity and creation of backups.

MySQL is free and open-source software under the terms of the GNU General Public License, and is also available under a variety of proprietary licenses. MySQL was owned and sponsored by the Swedish company MySQL AB, which was bought by Sun Microsystems (now Oracle Corporation). In 2010, when Oracle acquired Sun, Widenius forked the open-source MySQL project to create MariaDB.

MySQL has stand-alone clients that allow users to interact directly with a MySQL database using SQL, but more often, MySQL is used with other programs to implement applications that need relational database capability. MySQL is a component of the LAMP web application software stack (and others), which is an acronym for Linux, Apache, MySQL, Perl/PHP/Python. MySQL is used by many database-driven web applications, including Drupal, Joomla, phpBB, and WordPress. MySQL is also used by many popular websites, including Facebook, Flickr, MediaWiki, Twitter, and YouTube.

Support can be obtained from the official manual. Free support additionally is available in different IRC channels and forums. Oracle offers paid support via its MySQL Enterprise products. They differ in the scope of services and in price. Additionally, a number of third party organisations exist to provide support and services.

MySQL has received positive reviews, and reviewers noticed it "performs extremely well in the average case" and that the "developer interfaces are there, and the documentation (not to mention feedback in the real world via Web sites and the like) is very, very good". It has also been tested to be a "fast, stable and true multi-user, multi-threaded SQL database server".

5.2 Libraries and Modules

5.2.1 NumPy

NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays. The ancestor of NumPy, Numeric, was originally created by Jim Hugunin with contributions from several other developers. In 2005, Travis Oliphant created NumPy by incorporating features of the competing Numarray into Numeric, with extensive modifications. NumPy is open-source software and has many contributors. NumPy is a NumFOCUS fiscally sponsored project. NumPy targets the CPython reference implementation of Python, which is a non-optimizing bytecode interpreter. Mathematical algorithms written for this version of Python often run much slower than compiled equivalents due to the absence of compiler optimization. NumPy addresses the slowness problem partly by providing multidimensional arrays and functions and operators that operate efficiently on arrays; using these requires rewriting some code, mostly inner loops, using NumPy.

Using NumPy in Python gives functionality comparable to MATLAB since they are both interpreted,[21] and they both allow the user to write fast programs as long as most operations work on arrays or matrices instead of scalars. In comparison, MATLAB boasts a large number of additional toolboxes, notably Simulink, whereas NumPy is intrinsically integrated with Python, a more modern and complete programming language. Moreover, complementary Python packages are available; SciPy is a library that adds more MATLAB-like functionality and Matplotlib is a plotting package that provides MATLAB-like plotting functionality. Internally, both MATLAB and NumPy rely on BLAS and LAPACK for efficient linear algebra computations.

Python bindings of the widely used computer vision library OpenCV utilize NumPy arrays to store and operate on data. Since images with multiple channels are simply represented as three-dimensional arrays, indexing, slicing or masking with other arrays are very efficient ways to access specific pixels of an image. The NumPy array as universal data structure in OpenCV for images, extracted feature points, filter kernels and many more vastly simplifies the programming workflow and debugging.

5.2.2 Pickle

Python pickle module is used for serializing and de-serializing python object structures. The process to converts any kind of python objects (list, dict, etc.) into byte streams (0s and 1s) is called pickling or serialization or flattening or marshalling. We can converts the byte

stream (generated through pickling) back into python objects by a process called as unpickling.

5.2.3 Sklearn

It features various classification, regression and clustering algorithms including support-vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy. Scikit-learn is a NumFOCUS fiscally sponsored project. The scikit-learn project started as scikits.learn, a Google Summer of Code project by French data scientist David Cournapeau. Its name stems from the notion that it is a "SciKit" (SciPy Toolkit), a separately-developed and distributed third-party extension to SciPy. The original codebase was later rewritten by other developers. In 2010 Fabian Pedregosa, Gael Varoquaux, Alexandre Gramfort and Vincent Michel, all from the French Institute for Research in Computer Science and Automation in Rocquencourt, France, took leadership of the project and made the first public release on February the 1st 2010. Of the various scikits, scikit-learn as well as scikit-image were described as "well-maintained and popular" in November 2012. Scikit-learn is one of the most popular machine learning.

5.2.4 Keras

Keras is an open-source software library that provides a Python interface for artificial neural networks. Keras acts as an interface for the TensorFlow library.Up until version 2.3, Keras supported multiple backends, including TensorFlow, Microsoft Cognitive Toolkit, Theano, and PlaidML. As of version 2.4, only TensorFlow is supported. Designed to enable fast experimentation with deep neural networks, it focuses on being user-friendly, modular, and extensible. It was developed as part of the research effort of project ONEIROS (Open-ended Neuro-Electronic Intelligent Robot Operating System), and its primary author and maintainer is François Chollet, a Google engineer. Chollet is also the author of the Xception deep neural network model.

Keras contains numerous implementations of commonly used neural-network building blocks such as layers, objectives, activation functions, optimizers, and a host of tools to make working with image and text data easier to simplify the coding necessary for writing deep neural network code. The code is hosted on GitHub, and community support forums include the GitHub issues page, and a Slack channel. In addition to standard neural networks, Keras has support for convolutional and recurrent neural networks. It supports other common utility layers like dropout, batch normalization, and pooling. Karas allows users to productize deep models on smartphones (iOS and Android), on the web, or on

the Java Virtual Machine. It also allows use of distributed training of deep-learning models on clusters of Graphics processing units (GPU) and tensor processing units (TPU).

5.2.5 Matplotlib

Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK. There is also a procedural "pylab" interface based on a state machine (like OpenGL), designed to closely resemble that of MATLAB, though its use is discouraged. SciPy makes use of Matplotlib. Matplotlib was originally written by John D. Hunter. Since then it has an active development community and is distributed under a BSD-style license. Michael Droettboom was nominated as matplotlib's lead developer shortly before John Hunter's death in August 2012 and was further joined by Thomas Caswell. Matplotlib is a NumFOCUS fiscally sponsored project.

5.3 Hardware Requirements

- Processor : 2.5 gigahertz (GHz) frequency or above.
- RAM : A minimum of 4 GB of RAM.
- Hard disk : A minimum of 20 GB of available space.
- Input Device : High resolution camera
- Monitor : Minimum Resolution 1024 X 768.

5.4 Functional requirements

Functional requirements describe the system functionality, while the non-functional requirements describe system properties and constraints. Functional requirements capture the intended behaviour of the system. This behaviour may be expressed as services, tasks, or the functions the system is required to perform. This lays out important concepts and discusses capturing functional requirements in such a way they can drive architectural decisions and be used to validate the architecture. Features may be additional functionality, or differ from basic functionality along some quality attribute. In the proposed system, concert assesses the compliance of a workflow by analysing the five established elements required to check for the rule adherence in workflows: activities, data, location, resources, and time limits. A rule describes which activities may, must or must not be performed on

what objects by which roles. In addition, a rule can further prescribe the order of activities i.e. which activities have to happen before or after other activities.

5.5 Non-functional requirements

- **Security**
  - System needs to control the user access and session
  - It needs to store the data in a secure location and stored in a secure format
  - It requires a secure communication channel for the data.
- **Concurrency and Capacity**

  System should be able to handle multiple computations executing simultaneously, and potentially interacting with each other.
- **Performance**

  Performance is generally perceived as a time expectation. This is one of the most important considerations especially when the project is in the architecture phase.
- **Reliability**

  It is necessary to ensure and notify about the system transactions and processing as simple as keep a system log will increase the time and effort to get it done from the very beginning. Data should be transferred in a reliable way and using trustful protocols.
- **Maintainability**

  Well-done system is meant to be up and running for long time. Therefore, it will regularly need preventive and corrective maintenance. Maintenance might signify scalability to grow and improve the system features and functionalities.
- **Usability**

  End user satisfaction and acceptance is one of the key pillars that support a project success. Considering the user experience requirements from the project conception is a win bet, and it will especially save a lot of time at the project release, as the user will not ask for changes or even worst misunderstandings.
- **Documentation**

  All projects require a minimum of documentation at different levels. In many cases the users might even need training on it, so keeping good documentation practices and standards will do this task spread along the project development; but as well

CHAPTER 6

IMPLEMENTATION

6.1 Libraries

We import all the necessary libraries required to process the data and build the classification model.

```
[ ]  import numpy as np
     import pickle
     import cv2
     import os
     import matplotlib.pyplot as plt
     from os import listdir
     from sklearn.preprocessing import LabelBinarizer
     from keras.models import Sequential
     from keras.layers.normalization import BatchNormalization
     from keras.layers.convolutional import Conv2D
     from keras.layers.convolutional import MaxPooling2D
     from keras.layers.core import Activation, Flatten, Dropout, Dense
     from keras import backend as K
     from keras.preprocessing.image import ImageDataGenerator
     from keras.optimizers import Adam
     from keras.preprocessing import image
     from keras.preprocessing.image import img_to_array
     from sklearn.preprocessing import MultiLabelBinarizer
     from sklearn.model_selection import train_test_split
```

Fig.6.1 Importing libraries

6.2 Data Pre-processing

First, let's define a couple of variables required to perform operations on the dataset images. As the images are acquired from the real field it may contain dust, spores and water spots as noise. The purpose of data pre-processing is to eliminate the noise in the image, so as to adjust the pixel values. It enhances the quality of the image. The main aim of Pre-processing is to suppress unwanted image data and to enhance some important image features. It includes RGB to Gray conversion, image resizing and median filtering. Here color image is converted to Gray scale image to make the image device independent. The image is then resized to a size of 256*256. Then median filtering is performed on the image to remove the noise. The digital version of the rotten leaf sample consists of about 30% of leaf area and rest 70% is the background. Thus the redundant background requires high disk storage space and utilizes CPU time in the segmentation process. In order to have efficient disk storage and achieve fast processing speed the digital image of the leaf sample is cropped

into a smaller dimension of size 16x20sq, cm. Thus the pre-processing step introduced saves about 30% of disk storage space and increases the CPU processing 1.4 times.

The cropping process does not introduce any loss in the region of interest, i.e. the selected leaf sample. After pre-processing stage, the digital version of the sample leaf image consists about 70% of leaf area part and rest 30% as background. For getting better results in further steps, image pre-processing is required because dust, dewdrops, insect's excrements may be present on the plant; these things are considered as image noise. Furthermore, captured images may have distortion of some water drops and shadow effect, which could create problems in the segmentation and feature extraction stages. Effect of such distortion can be weakened or removed using different noise removal filters. There may be low contrast in captured images; for such images contrast enhancement algorithms can be used. Sometimes background removal techniques may also be needed in case of region of interest needs to be extracted. In case of noise such as salt and pepper, a median filter can be used. Weiner filter can be used to remove a blurring effect. In case of the images captured using high definition cameras, the size of the pictures might be very large, for that reduction of image size is required. Also, image reduction helps in reducing the computing memory power.

```
[ ]  # Dimension of resized image
     DEFAULT_IMAGE_SIZE = tuple((256, 256))

     # Number of images used to train the model
     N_IMAGES = 100

     # Path to the dataset folder
     root_dir = './PlantVillage'

     train_dir = os.path.join(root_dir, 'train')
     val_dir = os.path.join(root_dir, 'val')
```

Fig.6.2.1 Defining image dataset variables

1. We need to resize the raw dataset images to the **DEFAULT_IMAGE_SIZE** so that it would match the input shape of the primary layer of the neural network.
2. Each directory of the plant disease dataset folder varies in the number of images. Instead of taking them all, we select the first **N_IMAGES** from each directory to train our model.
3. Finally, we set the path of the dataset in the **root_dir** to access plant images.

Now, we write a function to convert or resize the input dataset images so that they can be fit for training.

```
[ ]  def convert_image_to_array(image_dir):
         try:
             image = cv2.imread(image_dir)
             if image is not None:
                 image = cv2.resize(image, DEFAULT_IMAGE_SIZE)
                 return img_to_array(image)
             else:
                 return np.array([])
         except Exception as e:
             print(f"Error : {e}")
             return None
```

Fig.6.2.2 Image resizing function

1. **cv2.imread()** loads an image from the specified file, if the image exists, otherwise it returns an empty matrix if an image cannot be read.
2. **cv2.resize()** changes the dimensions of an image, be it width alone, height alone, or both. Also, the aspect ratio of the original image could be preserved in the resized image.

We then iterate through the plant disease dataset folder, resize the images from each of the folders and convert or load them into a NumPy array.

```
[ ]  image_list, label_list = [], []

     try:
         print("[INFO] Loading images ...")
         plant_disease_folder_list = listdir(train_dir)

         for plant_disease_folder in plant_disease_folder_list:
             print(f"[INFO] Processing {plant_disease_folder} ...")
             plant_disease_image_list = listdir(f"{train_dir}/{plant_disease_folder}/")

             for image in plant_disease_image_list[:N_IMAGES]:
                 image_directory = f"{train_dir}/{plant_disease_folder}/{image}"
                 if image_directory.endswith(".jpg")==True or image_directory.endswith(".JPG")==True:
                     image_list.append(convert_image_to_array(image_directory))
                     label_list.append(plant_disease_folder)

         print("[INFO] Image loading completed")
     except Exception as e:
         print(f"Error : {e}")

     # Transform the loaded training image data into numpy array
     np_image_list = np.array(image_list, dtype=np.float16) / 225.0
     print()

     # Check the number of images loaded for training
     image_len = len(image_list)
     print(f"Total number of images: {image_len}")
```

Fig.6.2.3 Loading image dataset

```
[INFO] Loading images ...
[INFO] Processing Soybean___healthy ...
[INFO] Processing Cherry_(including_sour)___Powdery_mildew ...
[INFO] Processing Corn_(maize)___Northern_Leaf_Blight ...
[INFO] Processing Tomato___healthy ...
[INFO] Processing Tomato___Target_Spot ...
[INFO] Processing Potato___Early_blight ...
[INFO] Processing Pepper,_bell___Bacterial_spot ...
[INFO] Processing Peach___healthy ...
[INFO] Processing Corn_(maize)___Cercospora_leaf_spot Gray_leaf_spot ...
[INFO] Processing Corn_(maize)___healthy ...
[INFO] Processing Cherry_(including_sour)___healthy ...
[INFO] Processing Tomato___Leaf_Mold ...
[INFO] Processing Apple___Cedar_apple_rust ...
[INFO] Processing Potato___healthy ...
[INFO] Processing Squash___Powdery_mildew ...
[INFO] Processing Strawberry___healthy ...
[INFO] Processing Orange___Haunglongbing_(Citrus_greening) ...
[INFO] Processing Apple___healthy ...
[INFO] Processing Grape___Leaf_blight_(Isariopsis_Leaf_Spot) ...
[INFO] Processing Strawberry___Leaf_scorch ...
[INFO] Processing Tomato___Spider_mites Two-spotted_spider_mite ...
[INFO] Processing Tomato___Tomato_Yellow_Leaf_Curl_Virus ...
[INFO] Processing Pepper,_bell___healthy ...
[INFO] Processing Potato___Late_blight ...
[INFO] Processing background ...
[INFO] Processing Raspberry___healthy ...
[INFO] Processing Corn_(maize)___Common_rust_ ...
[INFO] Processing Apple___Black_rot ...
[INFO] Processing Peach___Bacterial_spot ...
[INFO] Processing Grape___healthy ...
[INFO] Processing Tomato___Septoria_leaf_spot ...
[INFO] Processing Apple___Apple_scab ...
[INFO] Processing Blueberry___healthy ...
[INFO] Processing Grape___Black_rot ...
[INFO] Processing Tomato___Bacterial_spot ...
[INFO] Processing Grape___Esca_(Black_Measles) ...
[INFO] Processing Tomato___Late_blight ...
[INFO] Processing Tomato___Tomato_mosaic_virus ...
[INFO] Processing Tomato___Early_blight ...
[INFO] Image loading completed

Total number of images: 3900
```

Fig.6.2.4 Plant disease classes

After loading the image dataset, we map each label or class of each plant disease to a unique value for the training task. Also, saving this transform to a pickle file will help us later in predicting a label or class of plant disease from the output of the classification model.

```
[7]  label_binarizer = LabelBinarizer()
     image_labels = label_binarizer.fit_transform(label_list)

     pickle.dump(label_binarizer,open('plant_disease_label_transform.pkl', 'wb'))
     n_classes = len(label_binarizer.classes_)

     print("Total number of classes: ", n_classes)

     Total number of classes:  39
```

Fig.6.2.5 Creating label transform

Finally, we split the loaded image dataset into two sets, namely train and test sets with a **0.2 split ratio**. Train set to train the classification model and test set to validate the model while training.

```
[9]  print("[INFO] Splitting data to train and test...")
     x_train, x_test, y_train, y_test = train_test_split(np_image_list, image_labels, test_size=0.2, random_state = 42)

     [INFO] Splitting data to train and test...
```

Fig.6.2.6 Splitting data into training and validation set

6.3 Data Augmentation

The data augmentation technique is used to significantly increase the number of images in a dataset. We perform various operations such as shift, rotation, zoom, and flip on the image dataset to diversify our dataset.

Providing augmented images to a model helps it efficiently learn features from different areas of the same image and thus perform better on unseen image data.

```
[ ]  augment = ImageDataGenerator(rotation_range=25, width_shift_range=0.1,
                                  height_shift_range=0.1, shear_range=0.2,
                                  zoom_range=0.2, horizontal_flip=True,
                                  fill_mode="nearest")
```

Fig.6.3 Data augmentation

Note: We just make an object of the ImageDataGenerator at the moment, which will later be passed to the model while training.

6.4 Model

Here, we define all the hyperparameters of our plant disease classification model. Executing them initially in a separate cell makes it easy for us to tweak them later if needed.

```
[10] EPOCHS = 25
     STEPS = 100
     LR = 1e-3
     BATCH_SIZE = 32
     WIDTH = 256
     HEIGHT = 256
     DEPTH = 3
```

Fig.6.4.1 Hyperparameters of neural network

Now, we create a sequential model for the classification task. In the model, we are not only defaulting to "**channel_last**" architecture but also creating a switch for backends that support "**channel_first**" on the fourth line.

For the model, we first create a **2D Convolutional layer** with **32 filters** of **3 x 3 kernel** and a **ReLU** (Rectified Linear Unit) **activation**. We then perform **batch normalization, max pooling**, and 25**% (0.25) dropout** operation in the following layers.

Next, we create two blocks of **2D Convolutional layer** with **64 filters** and **ReLU activation** followed by a **pooling** and **dropout layer**. We repeat this step for the last set

of **FC** (Fully Connected) layers with **128 filters** in the Conv2D layer being the only difference.

```
[ ] model = Sequential()
    inputShape = (HEIGHT, WIDTH, DEPTH)
    chanDim = -1

    if K.image_data_format() == "channels_first":
        inputShape = (DEPTH, HEIGHT, WIDTH)
        chanDim = 1

    model.add(Conv2D(32, (3, 3), padding="same",input_shape=inputShape))
    model.add(Activation("relu"))
    model.add(BatchNormalization(axis=chanDim))
    model.add(MaxPooling2D(pool_size=(3, 3)))
    model.add(Dropout(0.25))
    model.add(Conv2D(64, (3, 3), padding="same"))
    model.add(Activation("relu"))
    model.add(BatchNormalization(axis=chanDim))
    model.add(Conv2D(64, (3, 3), padding="same"))
    model.add(Activation("relu"))
    model.add(BatchNormalization(axis=chanDim))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))
    model.add(Conv2D(128, (3, 3), padding="same"))
    model.add(Activation("relu"))
    model.add(BatchNormalization(axis=chanDim))
    model.add(Conv2D(128, (3, 3), padding="same"))
    model.add(Activation("relu"))
    model.add(BatchNormalization(axis=chanDim))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))
    model.add(Flatten())
    model.add(Dense(1024))
    model.add(Activation("relu"))
    model.add(BatchNormalization())
    model.add(Dropout(0.5))
    model.add(Dense(n_classes))
    model.add(Activation("softmax"))

    model.summary()
```

Fig.6.4.2 Building classification model

6.5 Training

Before starting the training of our model, we initialize our optimizer with the learning rate and decay parameters we defined above. We select the **Adam optimization** technique as it nearly always performs faster and better global minimum convergence as compared to the other optimization techniques.

```
[ ] # Initialize optimizer
    opt = Adam(lr=LR, decay=LR / EPOCHS)

    model.compile(loss="binary_crossentropy", optimizer=opt, metrics=["accuracy"])

    print("[INFO] Training network...")
    history = model.fit_generator(augment.flow(x_train, y_train, batch_size=BATCH_SIZE),
                                  validation_data=(x_test, y_test),
                                  steps_per_epoch=len(x_train) // BATCH_SIZE,
                                  epochs=EPOCHS,
                                  verbose=1)
```

Fig. 6.5.1 Compiling classification model

```
[INFO] Training network...
Epoch 1/25
97/97 [==============================] - 792s 8s/step - loss: 0.0964 - accuracy: 0.9739 - val_loss: 0.3557 - val_accuracy: 0.9499
Epoch 2/25
97/97 [==============================] - 778s 8s/step - loss: 0.0852 - accuracy: 0.9749 - val_loss: 0.3076 - val_accuracy: 0.9669
Epoch 3/25
97/97 [==============================] - 793s 8s/step - loss: 0.0718 - accuracy: 0.9775 - val_loss: 0.3322 - val_accuracy: 0.9549
Epoch 4/25
97/97 [==============================] - 788s 8s/step - loss: 0.0576 - accuracy: 0.9813 - val_loss: 0.1837 - val_accuracy: 0.9626
Epoch 5/25
97/97 [==============================] - 791s 8s/step - loss: 0.0575 - accuracy: 0.9809 - val_loss: 0.0760 - val_accuracy: 0.9767
Epoch 6/25
97/97 [==============================] - 802s 8s/step - loss: 0.0500 - accuracy: 0.9830 - val_loss: 0.1289 - val_accuracy: 0.9714
Epoch 7/25
97/97 [==============================] - 795s 8s/step - loss: 0.0520 - accuracy: 0.9828 - val_loss: 0.1049 - val_accuracy: 0.9765
Epoch 8/25
97/97 [==============================] - 802s 8s/step - loss: 0.0422 - accuracy: 0.9854 - val_loss: 0.0811 - val_accuracy: 0.9787
Epoch 9/25
97/97 [==============================] - 796s 8s/step - loss: 0.0395 - accuracy: 0.9860 - val_loss: 0.0859 - val_accuracy: 0.9787
Epoch 10/25
97/97 [==============================] - 804s 8s/step - loss: 0.0359 - accuracy: 0.9875 - val_loss: 0.2368 - val_accuracy: 0.9668
Epoch 11/25
97/97 [==============================] - 809s 8s/step - loss: 0.0345 - accuracy: 0.9881 - val_loss: 0.1852 - val_accuracy: 0.9674
Epoch 12/25
97/97 [==============================] - 802s 8s/step - loss: 0.0323 - accuracy: 0.9887 - val_loss: 0.1122 - val_accuracy: 0.9758
Epoch 13/25
97/97 [==============================] - 806s 8s/step - loss: 0.0328 - accuracy: 0.9883 - val_loss: 0.2544 - val_accuracy: 0.9608
Epoch 14/25
97/97 [==============================] - 804s 8s/step - loss: 0.0373 - accuracy: 0.9868 - val_loss: 0.0958 - val_accuracy: 0.9783
Epoch 15/25
97/97 [==============================] - 810s 8s/step - loss: 0.0305 - accuracy: 0.9892 - val_loss: 0.1213 - val_accuracy: 0.9722
Epoch 16/25
97/97 [==============================] - 796s 8s/step - loss: 0.0289 - accuracy: 0.9897 - val_loss: 0.0630 - val_accuracy: 0.9845
Epoch 17/25
97/97 [==============================] - 807s 8s/step - loss: 0.0273 - accuracy: 0.9903 - val_loss: 0.0800 - val_accuracy: 0.9822
Epoch 18/25
97/97 [==============================] - 829s 9s/step - loss: 0.0256 - accuracy: 0.9909 - val_loss: 0.1230 - val_accuracy: 0.9761
Epoch 19/25
97/97 [==============================] - 822s 8s/step - loss: 0.0247 - accuracy: 0.9909 - val_loss: 0.1299 - val_accuracy: 0.9765
Epoch 20/25
97/97 [==============================] - 814s 8s/step - loss: 0.0243 - accuracy: 0.9913 - val_loss: 0.2022 - val_accuracy: 0.9696
Epoch 21/25
97/97 [==============================] - 820s 8s/step - loss: 0.0210 - accuracy: 0.9922 - val_loss: 0.0584 - val_accuracy: 0.9862
Epoch 22/25
97/97 [==============================] - 824s 8s/step - loss: 0.0198 - accuracy: 0.9926 - val_loss: 0.0557 - val_accuracy: 0.9838
Epoch 23/25
97/97 [==============================] - 825s 9s/step - loss: 0.0203 - accuracy: 0.9925 - val_loss: 0.0264 - val_accuracy: 0.9916
Epoch 24/25
97/97 [==============================] - 818s 8s/step - loss: 0.0197 - accuracy: 0.9925 - val_loss: 0.0481 - val_accuracy: 0.9887
Epoch 25/25
97/97 [==============================] - 813s 8s/step - loss: 0.0187 - accuracy: 0.9933 - val_loss: 0.0497 - val_accuracy: 0.9875
```

Fig.6.5.2 Training classification model

6.6 Evaluation

We plot a graph to compare the maximum accuracy achieved by the model while minimizing the loss during the training phase.

```
[ ]  acc = history.history['accuracy']
     val_acc = history.history['val_accuracy']
     loss = history.history['loss']
     val_loss = history.history['val_loss']
     epochs = range(1, len(acc) + 1)

     # Train and validation accuracy
     plt.plot(epochs, acc, 'b', label='Training accurarcy')
     plt.plot(epochs, val_acc, 'r', label='Validation accurarcy')
     plt.title('Training and Validation accurarcy')
     plt.legend()

     plt.figure()

     # Train and validation loss
     plt.plot(epochs, loss, 'b', label='Training loss')
     plt.plot(epochs, val_loss, 'r', label='Validation loss')
     plt.title('Training and Validation loss')
     plt.legend()
     plt.show()
```

Fig.6.6.1 Evaluation

We can obtain better results by tweaking the **learning rate** or by training on more images or just by simply training the model for more epochs.

To check the actual (test) accuracy of the model we trained, we use the evaluate() method and obtain a **test accuracy** of **98.75%.**

```
[ ]  print("[INFO] Calculating model accuracy")
     scores = model.evaluate(x_test, y_test)
     print(f"Test Accuracy: {scores[1]*100}")

     [INFO] Calculating model accuracy
     780/780 [==============================] - 52s 66ms/step
     Test Accuracy: 98.74754548072815
```

Fig.6.6.2 Test accuracy

6.7 Testing

We write the following predict_disease function to predict the class or disease of a plant image. We just need to provide the complete path to the image and it displays the image along with its prediction class or the plant disease.

```
[ ]  def predict_disease(image_path):
       image_array = convert_image_to_array(image_path)
       np_image = np.array(image_array, dtype=np.float16) / 225.0
       np_image = np.expand_dims(np_image,0)
       plt.imshow(plt.imread(image_path))
       result = model.predict_classes(np_image)
       print((image_labels.classes_[result][0]))
```

Fig.6.7 Predict function

6.8 Convolutional Neural Network (ConvNet) Algorithm

A convolutional neural network is a class of deep neural network, most commonly applied to analysing visual imagery.

A Convolutional Neural Network is a Machine Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other.

- **INPUT LAYER**

• In the figure below, we have an RGB image which has been separated by its three color planes—Red, Green, and Blue. There are a number of such color spaces in which images exist—Grayscale, RGB, HSV, CMYK, etc.

• The role of the ConvNet is to reduce the images into a form which is easier to process, without losing features which are critical for getting a good prediction.

• This is important when we are to design an architecture which is not only good at learning features but also is scalable to massive datasets.

- **CONVOLUTION LAYER--The Kernel**

• In the below demonstration, the green section resembles our 5x5x1 input image. The element involved in carrying out the convolution operation in the first part of a Convolutional Layer is called the Kernel/Filter, K, represented in the color yellow. We have selected K as a 3x3x1 matrix{{1,0,1},{0,1,0},{1,0,1}}.

• The Kernel shifts 9 times because of Stride Length = 1, every time performing a matrix multiplication operation between K and the portion P of the image.

• The filter moves to the right with a certain Stride Value till it parses the complete width. Moving on, it hops down to the beginning (left) of the image with the same Stride Value and repeats the process until the entire image is traversed.

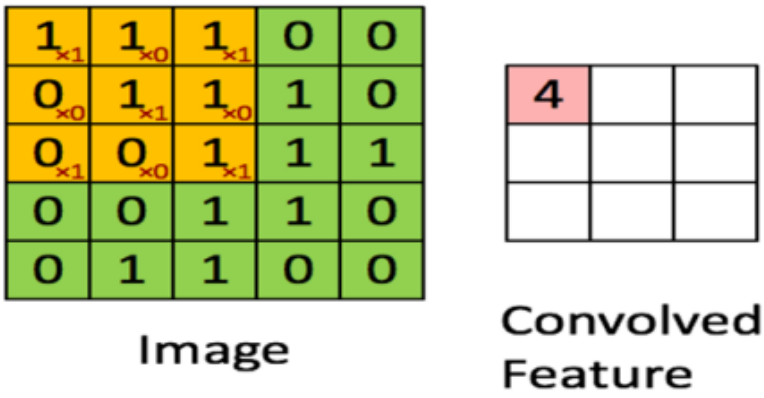

Fig.6.8.1 convoluting a 5x5x1 image with a 3x3x1 kernel

➢ POOLING LAYER

• The Pooling layer is responsible for reducing the spatial size of the Convolved Feature.

• It is even useful for extracting dominant features which are rotational and positional invariant, thus maintaining the process of effectively training of the model.

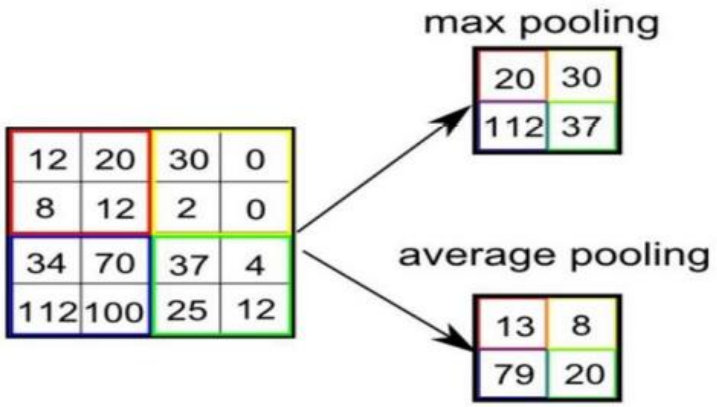

Fig.6.8.2 selecting maximum and average pooling matrix

- There are two types of Pooling: Max Pooling and Average Pooling.
- Max Pooling returns the maximum value from the portion of the image covered by the Kernel.
- On the other hand, Average Pooling returns the average of all the values from the portion of the image covered by the Kernel.

CHAPTER 7

TEST CASES

The purpose of testing is to discover errors. Testing is a 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. Software testing is an important element of the software quality assurance and represents the ultimate review of specification, design and coding. The increasing feasibility of software as a system and the cost associated with the software failures are motivated forces for well-planned through testing.

7.1 Testing Objectives

There are several rules that can serve as testing objectives they are:

- Testing is a process of executing program with the intent of finding an error.
- A good test case is the one that has a high probability of finding an undiscovered error.

7.2 Types of Testing

In order to make sure that the system does not have errors, the different levels of testing strategies that are applied at different phases of software development are:

1. Unit Testing: Unit testing is done on individual models as they are completed and becomes executable. It is confined only to the designer's requirements. Unit testing is different from and should be preceded by other techniques, including:
    - Inform Debugging
    - Code Inspection
2. Black Box testing: In this strategy some test cases are generated as input conditions that fully execute all functional requirements for the program. This testing has been used to find error in the following categories: Incorrect or missing functions
    - Interface errors
    - Errors in data structures are external database access
    - Performance error
    - Initialisation and termination of errors
    - In this testing only the output is checked for correctness

- The logical flow of data is not checked

3. White Box testing: In this the test cases are generated on the logic of each module by drawing flow graphs of that module and logical decisions are tested on all the cases. It has been used to generate the test cases in the following cases:
    - Guarantee that all independent paths have been executed
    - Execute all loops at their boundaries and within their operational bounds.
    - Execute internal data structures to ensure their validity.
4. Integration Testing: Integration testing ensures that software and subsystems work together a whole. It tests the interface of all the modules to make sure that the modules behave properly when integrated together. It is typically performed by developers, especially at the lower, module to module level. Testers become involved in higher levels
5. System Testing: Involves in house testing of the entire system before delivery to the user. The aim is to satisfy the user the system meets all requirements of the client's specifications. It is conducted by the testing organization if a company has one. Test data may range from and generated to production. Requires test scheduling to plan and organize:
    - Inclusion of changes/fixes.
    - Test data to use

    One common approach is graduated testing: as system testing progresses and (hopefully) fewer and fewer defects are found, the code is frozen for testing for increasingly longer time periods.

6. Acceptance Testing: It is a pre-delivery testing in which entire system is tested at client's site on real world data to find errors.
    - User Acceptance Test (UAT):
      "Beta testing": Acceptance testing in the customer environment.
    - Requirements traceability:
        1. Match requirements to test cases.
        2. Every requirement has to be cleared by at least one test case.
        3. Display in a matrix of requirements vs. test cases.

ID	Test Case	Input Description	Expected Output	Test Status
1	Loading model	Initializing trained model and load it.	Loaded model without errors	Pass
2	Converting images into arrays	Converting images into numpy arrays	Conversion into array done without errors	Pass
3	Extracting key features from the images	Image frame that contains plant leaf	Extracting the key features and saving into a numpy array of the respective disease	Pass
3	Recognize plant disease	Image frame that contains plant leaf	Label (disease name)	Pass

Table 7.2 Verification of testcases

CHAPTER 8

RESULTS

8.1 Screenshots

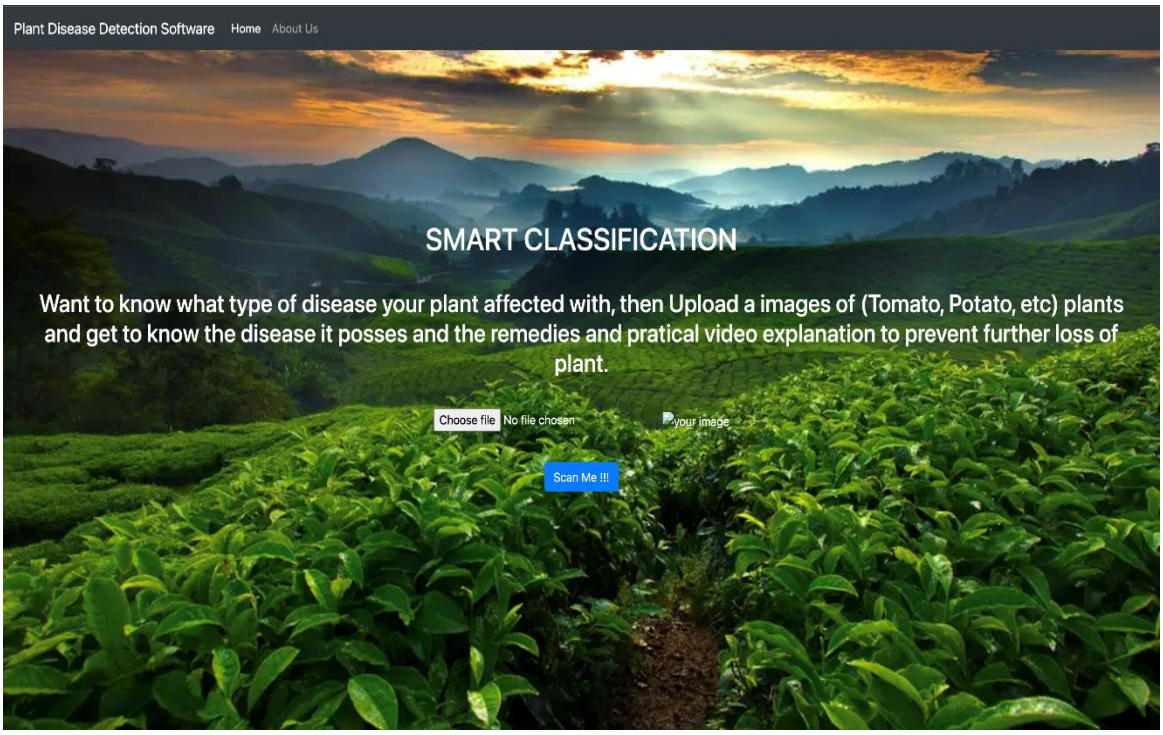

Fig.8.1.1 Home Page

Figure 8.1.1 depicts the first look of our front end. We have a text message called “Choose file” to choose picture for testing so that a user can understand to click the below button. This button can be used to browse the images on the system’s hard disk.

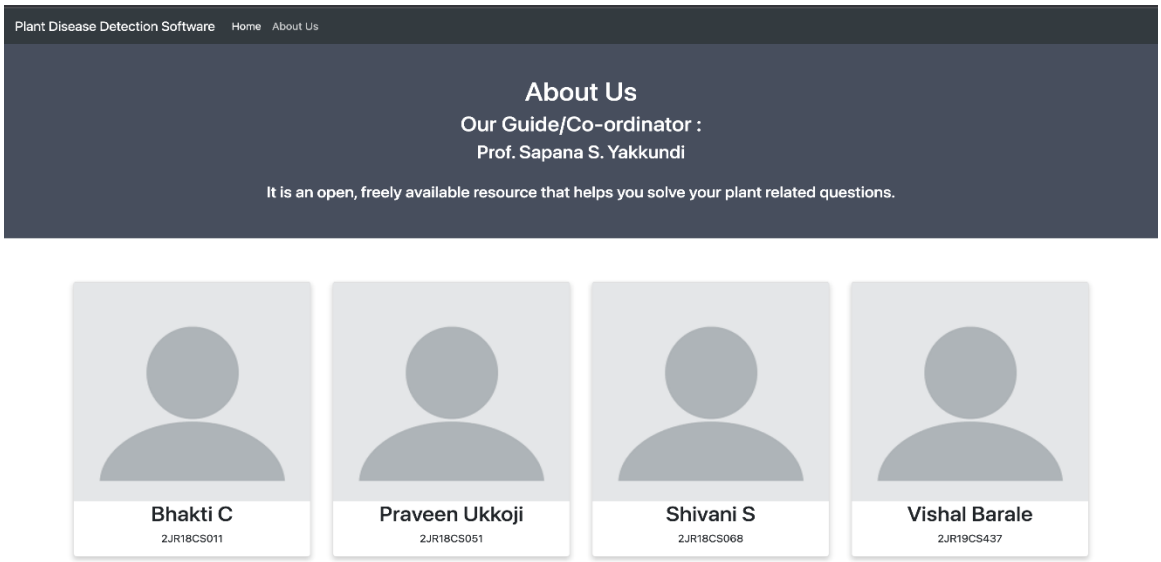

Fig.8.1.2 About Us

Figure 8.1.2 shows the details about our project guide or co-ordinators along with the four project team members.

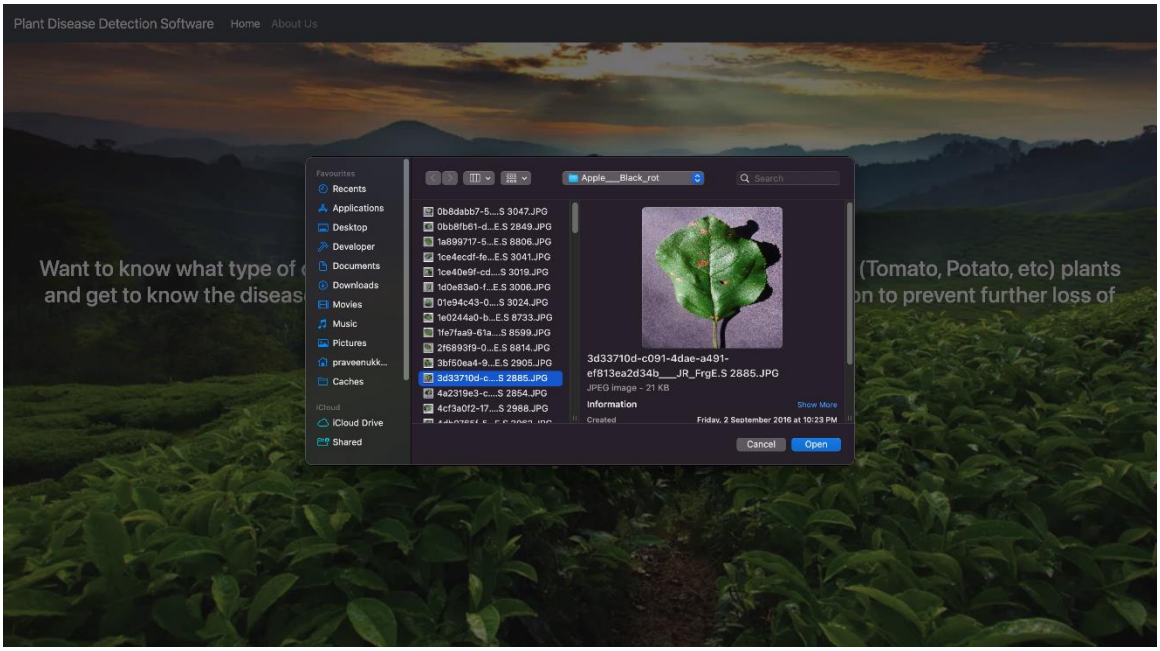

Fig.8.1.3 Selecting Image

Figure 8.1.3 shows the popup which appears when user clicks on 'chose file' button. The popup window will be having number of input images to be selected, this action should be confirmed with a double click or an open button.

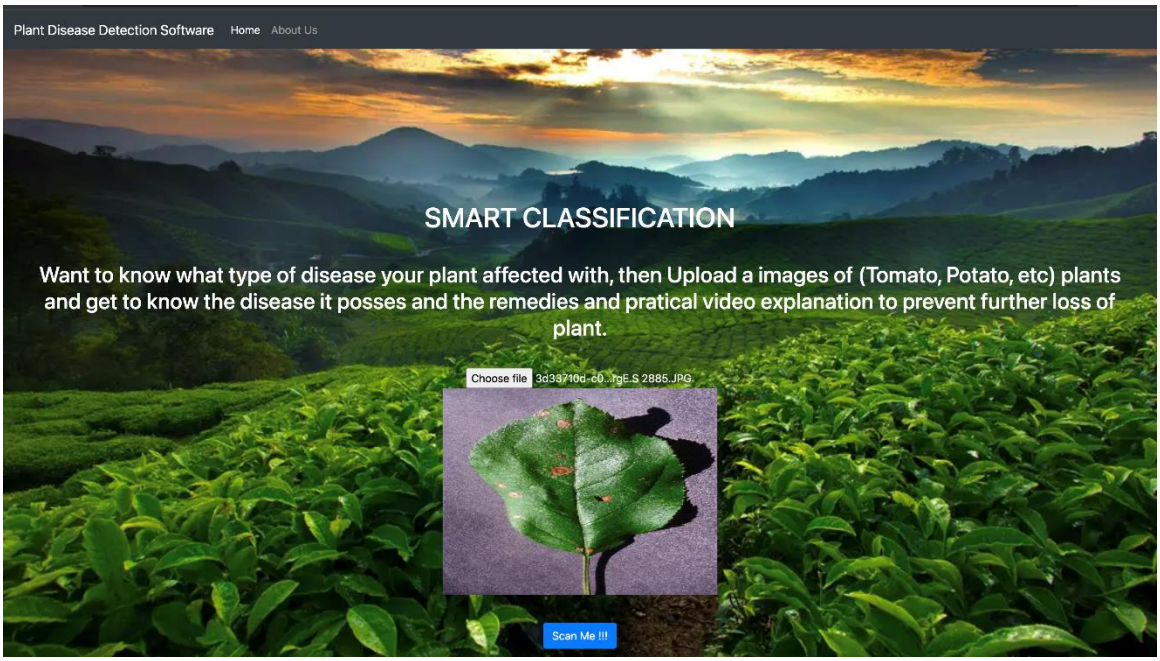

Fig.8.1.4 Scan Image

Figure 8.1.4 shows the selected image from the system directory, there will be a button provided called as 'Scan me' for analysing the input image to detect the condition of the leaf ,once the picture has been selected ,it image shows on the home screen .After selecting the image click on “Scan Me” button which detects the disease type with its cure.

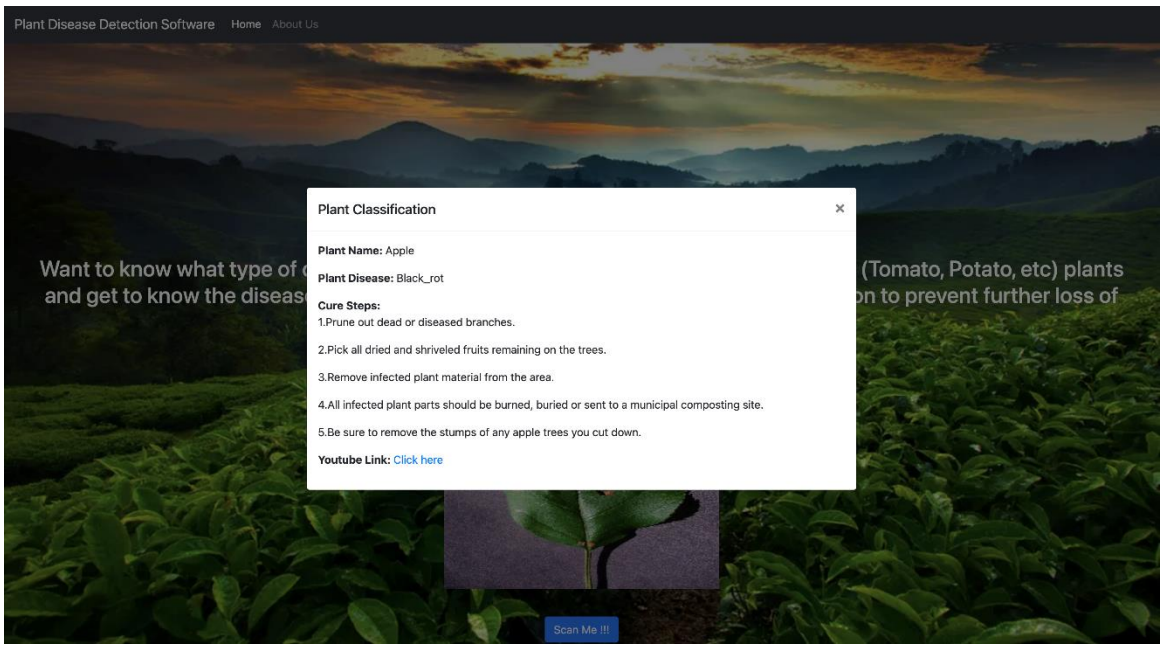

Fig.8.1.5 Result of Scanned Image

Figure 8.1.5 displays the status of leaf i.e. Healthy or Unhealthy, if it is unhealthy it displays the particular disease name.The result of scanned image shows the plant name,plant disease and suggest the Remedies for the disease found in the leaf and it also shows the youtube video link related to leaf disease .

CHAPTER 9

CONCLUSION

In our country around approx. 60% of farmer are illiterate and poor, they do not know which pesticides can be used to save the plant from diseases and the plant infection. Our naked eyes can't detect the disease and also people may not know which fertilizer to be used to prevent the plant from diseases, so we use computer-based programming data to detect the disease and provide remedies. Health monitoring and disease detection on plant is very critical for sustainable agriculture. It is very difficult to monitor the plant diseases manually. It requires tremendous amount of work, expertise in the plant diseases, and also require the excessive processing time. To have a meaningful impact of plant diseases & techniques in the area of agriculture, deliberation of proper input is necessary. Research issues addressed here are to develop a systematic approach to detect and recognize the plant diseases would assist farmers and pathologist in prospect exploration. The paper depicts the importance of image processing in agriculture field and considering the type of disease for further research work. Our aim of the project is to detect the plant diseases and provide the solutions to recover from the plant diseases. We planned to design a software where it will detect the type of plant, disease and also provide its cure of disease.

FUTURE SCOPE

The proposed Plant Disease Detection Using Machine Learning System used to recognize disease of plants can be further extended to provide valuable information about the plants. Instead of just displaying disease name and cure steps it will be more appropriate to provide some video links to follow cure steps. This also gives user proper understanding of cure steps and why it has to be done. The scope of different plants can be increased in this project. More training data can be added to detect the disease with more accuracy. This project can further be extended to cover large number of plants for disease detection.

REFERENCES

[1] Shima Ramesh, Mr. Ramachandra Hebba, Mr. P V Vino, Niveditha M, Pooja R, Prasad Bhat N, Shashank N, "Plant Disease Detection Using Machine Learning" International Conference on Design Innovations for 3Cs Compute Communicate Control 2018.

[2] Akshita Arora, Nazim Khan, Shefali Gupta, Samikchha Singh Maansi Gupta. "Leaf Disease Detection Using CNN and Raspberry PI." In computing, communications and Networking Technologies (ICCCNT), 2019 Fourth International conference on, pp. 1-5 IEEE, 2019.

[3] Santhosh Kumar S, and B. K. Raghavendra. Diseases Detection of Various Plant Leaf Using Image Processing Techniques: A Review. 2019 5th International Conference on Advanced Computing & Communication Systems (ICACCS).

[4] Huu Quan Cap, Katsumasa Suwa, Erika Fujita, Satoshi Kagiwada, Hiroyuki Uga, and Hitoshi Iyatomi. A deep learning approach for on-site plant leaf detection. 2018 IEEE 14th International Colloquium on Signal Processing & Its Applications (CSPA).

[5] Abirami Devaraj, Karunya Rathan, Sarvepalli Jaahnavi, and K Indira. Identification of Plant Disease using Image Processing Technique. 2019 International Conference on Communication and Signal Processing (ICCSP).

[6] Sharath N Payyadi, Varun S D, Satya Gururaj Kalluru, Archana R Kulkarni "Disease Detection in Paddy Crop using CNN Algorithm" international Journal of Recent Technology and Engineering (IJRTE)ISSN: 2277-3878, April 2020.