VISVESVARAYA TECHNOLOGICAL UNIVERSITY
BELGAUM-590 014

A Project Report
On

"PLANT LEAF DISEASE DETECTION SYSTEM"

Submitted in partial fulfillment of the requirement for the award of the degree of

BACHELOR OF ENGINEERING
In
COMPUTER SCIENCE AND ENGINEERING

Submitted by

UWAIZ ALI KHAN	**[1VE16CS111]**
VARSHINI RAO V	**[1VE16CS112]**
VAISHNAVI C R	**[1VE16CS115]**
RUCHI	**[1VE16CS125]**

Under the Guidance of
Dr S C LINGAREDDY
Head Of The Department
Department of Computer Science and Engineering
Sri Venkateshwara College of Engineering, Bangalore-562 157

Accredited by NBA*
SVCE BENGALURU
SRI VENKATESHWARA COLLEGE OF ENGINEERING
Affiliated to VTU, Approved by AICTE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
SRI VENKATESHWARA COLLEGE OF ENGINEERING
BANGALORE-562 157.

2019-2020

SRI VENKATESHWARA COLLEGE OF ENGINEERING
Vidyanagar, Bangalore – 562 157
Department of Computer Science and Engineering

Accredited by NBA*
SVCE BENGALURU
SRI VENKATESHWARA COLLEGE OF ENGINEERING
Affiliated to VTU, Approved by AICTE

CERTIFICATE

This is to certify that the project entitled **"PLANT LEAF DISEASE DETECTION SYSTEM"** carried out by **Mr. UWAIZ ALI KHAN [1VE16CS111], Ms. VARSHINI RAO V [1VE16CS112], Ms. VAISHNAVI C R [1VE16CS115], Ms. RUCHI[1VE16CS125],** bonafide students of Sri Venkateshwara College of Engineering, in partial fulfillment for the award of Bachelor of Engineering in Computer Science and Engineering of **Visvesvaraya Technological University, Belgaum** during the academic year 2019-2020. It is certified that all corrections/suggestions indicated for Internal Assessment have been incorporated in the Report deposited in the departmental library.

The project report has been approved as it satisfies the academic requirements in respect of Project work prescribed for the said Degree.

Signature of the HOD **Dr. S. C. Lingareddy** HOD Dept. of CSE, **SVCE, Bangalore**	**Signature of the Principal** **Dr. Suresha** Principal SVCE, Bangalore

Name of the examiners:	**Signature with date**
1. _____	_____
2. _____	_____

ABSTRACT

Crop production problems are common in India which severely effect rural farmers, agriculture sector and the country's economy as a whole. In Crops leaf plays an important role as it gives information about the quantity and quality of agriculture yield in advance depending upon the condition of leaf. In this project we proposed the system which works on pre-processing, feature extraction of leaf images from plant village dataset followed by convolution neural network for classification of disease and recommending Pesticides using Tensor flow technology. The main two processes that we use in our system is GUI screen with Python and Deep Learning. We have use Convolution Neural Network with different layers five, four & three to train our model and GUI screen as a user interface.

ACKNOWLEDGEMENT

The satisfaction that accompanies that the successful completion of any task would be incomplete without mentioning the people whose ceaseless cooperation made it possible, whose constant guidance and encouragement crown all efforts with success.

Our sincere thanks to the highly esteemed institution **SRI VENKATESHWARA COLLEGE OF ENGINEERING** for grooming up us to be budding software engineers.

We express our sincere gratitude to **Dr. SURESHA,** Principal, SVCE, Bengaluru for providing the required facility.

We wish to take this opportunity to express our sincere thanks to our project guide **Dr. S C LINGAREDDY, Professor & HOD, Department of Computer Science and Engineering, Sri Venkateshwara College of Engineering, Bangalore**, for providing constant support and encouragement to complete our project successfully.

We would also like to express our deep sense of gratitude to **Mrs. Varsha Kulkarni**, Assistant Professor, Department of Computer Science and Engineering, Sri Venkateshwara College of Engineering, Bangalore, for her timely scheduling, valuable suggestions and encouragement to bring out our project in right direction.

Finally, we would like to express our heart full thanks to parents and friends for their invaluable help, constant support and motivation for helping us to complete our project work successfully.

UWAIZ ALI KHAN [1VE16CS111]
VARSHINI RAO V [1VE16CS112]
VAISHNAVI C R [1VE16CS115]
RUCHI [1VE16CS125]

CONTENTS

LIST OF TABLES

TABLE OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 OVERVIEW

India is a cultivated country and about 70% of the population depends on agriculture. Farmers have large range of diversity for selecting various suitable crops and finding the suitable pesticides for plant. Disease on plant leads to the significant reduction in both the quality and quantity of agricultural products. The studies of plant disease refer to the studies of visually observable patterns on the plants. Monitoring of health and disease on plant plays an important role in successful cultivation of crops in the farm. In early days, the monitoring and analysis of plant diseases were done manually by the expertise person in that field.

This requires tremendous amount of work and also requires excessive processing time. The image processing techniques can be used in the plant disease detection. In most of the cases disease symptoms are seen on the leaves, stem and fruit. The plant leaf for the detection of disease is considered which shows the disease symptoms. This project gives the introduction to image processing technique used for plant disease detection. Identification of the plant diseases is the key to preventing the losses in the yield and quantity of the agricultural product. The studies of the plant diseases mean the studies of visually observable patterns seen on the plant. 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. Hence, image processing is used for the detection of plant diseases. This project discussed the methods used for the detection of plant diseases using their leaves images. Our system also sprays the pesticides for diseased plants.

Technology helps human beings in increasing the production of food. However the production of food can be affected by number of factor such as climatic change, diseases, soil fertility etc. Out of these, disease plays major role to affect the production of food. Agriculture plays an important role in Indian economy. Leaf spot diseases weaken trees and shrubs by interrupting photosynthesis, the process by which plants create energy that sustains growth and defense systems and influences survival[1].Over 58% smallholder farmer depends on agriculture as their principal means of livelihood. In the developing world, more than 80 percent of the agricultural production is generated by smallholder farmers, and reports of yield loss of more than 50% due to pests and diseases are common[2].The production is decreasing day by day with various factors and one of them is diseases on plants which are not detected early stage. There is various work is done in previous years. Bacterial disease reduces plants growth very quickly so to detect this type of diseases ,Dheeb Al Bashish, Malik Braik, and Sulieman Bani-Ahmad [3] created

system which detect the type of disease the plants have using image processing and color space transformation which creates device independent transformation. Identifying the disease at an early stage and suggesting the solution so that maximum harm can be avoided to increase the crop yield [4] have used ANN and K-means to classify the disease and grade the disease for. There is a need to design the automatic system to detect the leaf disease and recommend the proper pesticide. Pesticides on rice for controlling the disease damages the rice filed [5] created which will detect diseases at early stage. Which pesticides to use for which type of disease is the important task [6] gives the solution to which type of pesticides use.

Introduction of Deep Learning

Deep learning is specific subset of Machine Learning, which is a specific subset of Artificial Intelligence. Artificial Intelligence is the broad area of creating machines that can think intelligently. Machine learning is a part or subset of artificial intelligence (AI) which provides ability to the system to automatically learn and improve from experience without being explicitly programmed. Machine learning focuses on the development of computer's programs that can access data and use it learn for themselves. Deep learning is a collection of algorithms used in machine learning, used for top-level abstractions in data through the use of model's architectures, which are composed of multiple nonlinear transformations.

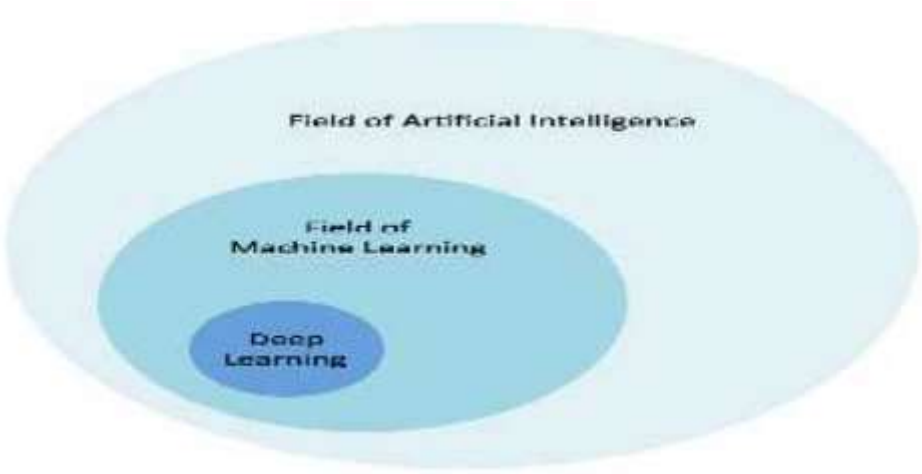


Fig 1.1. Intro of Deep Learning

Introduction of Neural Network

Neural networks are computational models and a set of algorithms which work similar to the functioning of a human nervous system and also we can say in other words like a biologically-inspired programming which enables a computer to learn from observational data. Deep learning have a powerful set of techniques for learning in neural networks. Neural networks help us cluster and classify. Deep Learning and Neural networks currently provide the good solutions to many problems in image recognition, speech recognition, and natural language

processing.

Introduction of Object Detection

Object detection has very wide area in deep learning. Object detection with the convolution neural network used in many real time fields like medical application, agriculture field's different application, identify the shapes of objects which is more helpful in manufacturing industries, face detection and people counting etc. To detect the object in deep learning, it uses the neural network of deep learning. Simply object detection in deep learning means to identify objects from images. So it identify the object based on how I train the neural network. In recent scenarios there are very wide range of applications which uses the object detection using convolutional neural network.

Introduction Of Convolutional Neural Network

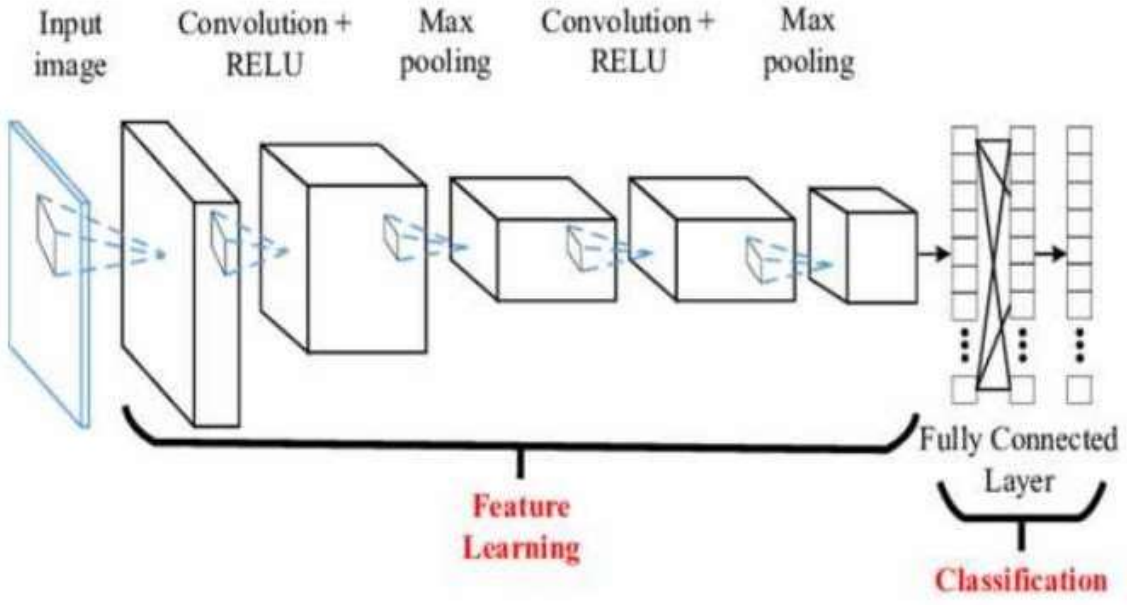


Fig 1.2 .Convolutional Neural Networks (CNN)

In Convolutional Neural Network, the image first passes through the convolutional neural layer it will convert image into a matrix of as per our size where we have used the size of the matrix as 256 x 256 x 3. Here 3 is for the color image. These image matrix multiples with the filters or kernel matrix and results in the Feature Map matrix. CNN uses the strides which defines the number of pixels to be a shift over the input matrix. If the value of stride is 1 then filter matrix shift the cell by 1 and if the value is 2 then slides with the 2 pixels. If the filters do not apply perfectly fit the input image then padding will be used and also provides the two cases, 1) pad the picture with the zero so that it fits, 2) Drop the part of the image where filter not fit, Called valid padding. It has the ReLU which stand for the Rectified linear unit for non-linear unit. The output is f(x) = max(0,x). which transform negative non-linear value with zero. Pooling layer section reduces the number of parameters when the images are too large. Pooling can be of different types as 1) Max Pooling, 2) Average Pooling, 3) Sum Pooling. Max pooling takes the largest element from the rectified feature map.Taking the largest element could also

take the average pooling and sum of all elements in the feature map call as sum pooling. Fully Connected Layer we call as FC layer, where we flattened our matrix into a vector and feed it into a fully connected layer like a neural network. At last activation function such as softmax or sigmoid to classify the outputs as the cat, dog, car, truck, identify disease from the leaf of the crop, etc.

1.2 PROBLEM STATEMENT

Less yield, higher cost of production due to labour scarcity and fertilizer cost are the major challenges before the farmers. To enhance the quality and quantity of the agriculture product there is a need to adopt the new technology. Fertilizer and pesticide management requires early and cost effective solutions which will lead to higher yield. Image processing approach is non-invasive technique which provides consistent, reasonably accurate, less time consuming and cost effective solution for farmers to manage fertilizers and pesticides.

1.3 OBJECTIVE

- Image processing is used for detection of plant diease and estimation of the fertilizers.
- Images are captured on field with digital camera in natural illumination on sunny days. As it is difficult to measure the impact of illumination at the time of image capturing in normal case, illumination effect is not considered at the time of image processing
- Image processing tool box of OpenCV and CNN for regression analysis along with other statistical tool are used for estimation of the fertilizers.

1.4 SCOPE

The most important reasons that object detection has gain more attention in the last few years is due to booming area of Artificial Intelligence. Object detection approach can be gained high accuracy rate in last few years through the deep learning domain itself.
Various model of deep learning concept are used in training and testing of networks for Object detection system. AI has many areas for research but the Object Detection through deep learning is trending in order to build robust Object detection system with different application.

1.5 ADVANTAGES

- The processing of images is faster and more cost-effective. One needs less time for processing, as well as less film and other photographing equipment is required.

- It is more ecological and economical to process images. No processing or fixing chemicals are needed to take and process digital images.
- When shooting a digital image, one can immediately see if the image is good or not.
- Copying a digital image is easy, and the quality of the image stays good unless it is compressed. For instance, saving an image as jpg format compresses the image. By resaving the image as jpg format, the compressed image will be recompressed, and the quality of the image will get worse with every saving.
- Fixing and retouching of images has become easier. It is possible to smoothen distorted pixels by making use of python packages such as imutils and opencv.
- The expensive reproduction is faster and cheaper.
- By changing the image format and resolution, the image can be used in a number of media.

- Important features such as edges can be extracted from images which can be used in industry.
- Images can be given more sharpness and better visual appearance.
- Minor errors can be rectified.
- Images sizes can be increased or decreased.
- Images can be compressed and decompressed for faster images transfer over the network.
- Images can be automatically sorted depending on the contents they have unrecognisable features can be made prominent.

CHAPTER 2

LITERATURE SURVEY

The system by Melike Sardogan , Adem Tuncer ,Yunus Ozen [1] focuses on Convolutional Neural Network (CNN) model and Learning Vector Quantization (LVQ) algorithm based method for tomato leaf disease detection and classification. The dataset contains 500 images of tomato leaves with four symptoms of diseases. We have modeled a CNN for automatic feature extraction and classification.

The paper by L. Sherly Puspha Annabel, T. Annapoorani, P. Deepalakshmi [2] presents an overview on various types of plant diseases and different classification techniques in machine learning that are used for identifying diseases in different plant leaves.

[3] Santhosh S. Kumar, B.K. Raghavendra "Diseases Detection of Various Plant Leaf Using Image Processing Techniques: A Review" To detect the plant diseases many fast techniques need to be adopted.This paper shows a survey on different plants disease and various advance techniques to detect these diseases.

In [4]. Amrita S. Tulshan, Nataasha Raul focus on steps like image pre-processing, image segmentation, feature extraction. Furthur K Nearest Neighbor (KNN) classification is applied on the outcome of these three stages. Proposed implementation has shown 98.56% of accuracy in predicting plant leaf diseases. It also presents other information regarding a plant leaf disease that is Affected Area, Disease Name, Total Accuracy, Sensitivity and Elapsed Time

[5] This paper presents, a survey on different technologies of leaf disease detection using image processing approach and classified them based on the type of analysis tool and applications. Almost, prevailing technologies used in leaf disease detection system are critically reviewed and discussed in brief; comparison of available approaches are examined and presented. The key issues and challenges in leaf diseases detection are highlighted.

[6]. This paper reviews the potential of the methods of plant leaves disease detection system that facilitates the advancement in agriculture. It includes various phases such as the image acquisition, image segmentation, feature extraction and classification.

The paper [7] proposes a disease detection and classification technique with the help of machine learning mechanisms and image processing tools. Initially, identifying and capturing the infected region is done and latter image preprocessing is performed. Further, the segments are obtained and the area of interest is recognized and the feature extraction is done on the same. Finally the obtained results are sent through SVM Classifiers to get the results

[8] a survey on various techniques of plant disease detection is reviewed and discussed in terms of various parameters.

[9] This paper presents an overview of using image processing methods to detect various plant diseases. Image processing provides more efficient ways to detect diseases caused by fungus, bacteria or virus on plants. Mere observations by eyes to detect diseases are not accurate. Overdose of pesticides causes harmful chronic diseases on human beings as not washed properly. Excess use also damages plants nutrient quality. It results in huge loss of production to farmer. Hence use of image processing techniques to detect and classify diseases in agricultural applications is helpful.

The paper [10] says Health monitoring and the identification of disease in plant is very difficult manually. It requires expertise in the plant disease and also it requires more processing time. Hence, image processing is used for the identification of plant diseases. Disease detection involves steps like image acquisition, image pre-processing, image segmentation, feature extraction, object recognition and classification. Based on the output obtained from the abovementioned criteria's, the disease with which the plant affected is observed.

[11] 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 we use Histogram of an Oriented Gradient (HOG).

[12] This survey paper describes plant disease identification using Machine Learning Approach and study in detail about various techniques for disease identification and classification is also done.

CHAPTER 3

REQUIREMENT SPECIFICATION

In this section we discuss about the hardware and software requirements for this project and explain the software used to build the front end and back end of the project.

3.1 HARDWARE REQUIREMENTS

Processor – Intel® Core™ i3-2348M CPU @ 2.30GHz

Hard Disk – 1TB

Memory – 6.00 GB RAM

Any desktop / Laptop system with above configuration or higher level.

3.2 SOFTWARE REQUIREMENTS

Operating system	: Windows 7 or higher
Coding Language	: Python 3.6 and higher.
Software	: Python, OpenCV, Keras, Tensorflow,Pillow, Blynk, Tkinter

3.3 FRONT END IMPLEMENTATION

3.3.1 TKINTER

- **Tkinter** commonly comes bundled with Python, using Tk and is Python's standard GUI framework. It is famous for its simplicity and graphical user interface. It is open-source and available under the Python License.
- Tkinter comes pre-installed with Python3, and you need not bother about installing it.

Now, let's build a very simple GUI with the help of Tkinter and understand it with the help of a flow diagram.

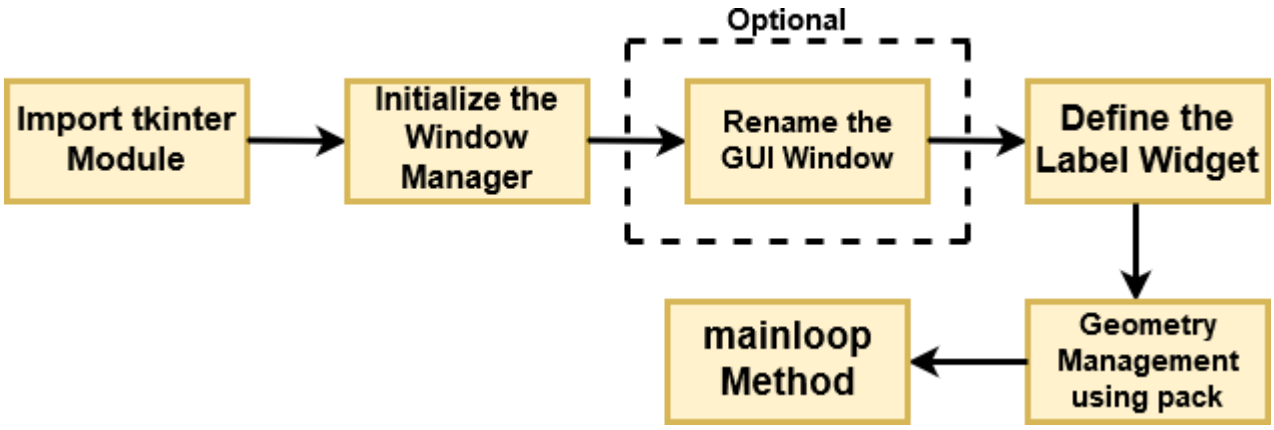


Fig 3.1. Flow Diagram for Rendering a Basic GUI

- **Tkinter** is a Python binding to the Tk GUI toolkit. It is the standard Python interface to the Tk GUI toolkit, and is Python's de facto standard GUI. Tkinter is included with standard Linux, Microsoft Windows and Mac OS X installs of Python.
- The name Tkinter comes from Tk interface. Tkinter was written by Fredrik Lundh.
- Tkinter is free software released under a Python license.

3.4 BACK END IMPLEMENTATION (PYTHON & SQL)

3.4.1 PYTHON

Python is an interpreted, high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991, Python's design philosophy emphasizes code readability with its notable use of significant whitespace. Its language constructs and object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects.

- Python is dynamically typed and garbage-collected. It supports multiple programming paradigms, including procedural, object-oriented, and functional programming. Python is often described as a "batteries included" language due to its comprehensive standard library.
- Python was conceived in the late 1980s as a successor to the ABC language. Python 2.0, released in 2000, introduced features like list comprehensions and a garbage collection system capable of collecting reference cycles. Python 3.0, released in 2008, was a major revision of the language that is not completely backward-compatible, and much Python 2 code does not run unmodified on Python 3.
- The Python 2 language, i.e. Python 2.7.x, was officially discontinued on 1 January 2020 (first planned for 2015) after which security patches and other improvements will not be released for it. With Python 2's end-of-life, only Python 3.5.x and later are supported.
- Python interpreters are available for many operating systems. A global community of programmers develops and maintains CPython, an open source reference implementation. A non-profit organization, the Python Software Foundation, manages and directs resources for Python and CPython development.

3.4.2 OPEN CV

OpenCV (*Open source computer vision*) is a library of programming functions mainly aimed at real-time computer vision.[1] Originally developed by Intel, it was later supported by Willow Garage then Itseez (which was later acquired by Intel). The library is cross-platform and free for use under the open-source BSD license.

OpenCV supports the deep learning frameworks TensorFlow, Torch/PyTorch and Caffe.

OpenCV's application areas include:

- 2D and 3D feature toolkits
- Egomotion estimation

- Facial recognition system
- Gesture recognition
- Human–computer interaction (HCI)
- Mobile robotics
- Motion understanding
- Object identification
- Segmentation and recognition
- Stereopsis stereo vision: depth perception from 2 cameras
- Structure from motion (SFM)
- Motion tracking
- Augmented reality

To support some of the above areas, OpenCV includes a statistical machine learning library that contains:

- Boosting
- Decision tree learning
- Gradient boosting trees
- Expectation-maximization algorithm
- k-nearest neighbor algorithm
- Naive Bayes classifier
- Artificial neural networks
- Random forest
- Support vector machine (SVM)
- Deep neural networks (DNN)

OpenCV is written in C++ and its primary interface is in C++, but it still retains a less comprehensive though extensive older C interface. There are bindings in Python, Java and MATLAB/OCTAVE. The API for these interfaces can be found in the online documentation. Wrappers in other languages such as C#, Perl, Ch, Haskell, and Ruby have been developed to encourage adoption by a wider audience.

3.4.3 KERAS

Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling fast experimentation.

Keras is an open-source neural-network library written in Python. It is capable of running on top of TensorFlow, Microsoft Cognitive Toolkit, R, Theano, or PlaidML. 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 also is 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.

Keras allows users to productize deep models on smartphones (iOS and Android), on the web, or on the Java Virtual Machine.[11] It also allows use of distributed training of deep-learning models on clusters of Graphics Processing Units (GPU) and Tensor processing units (TPU) principally in conjunction with CUDA.

Guiding principles

- User friendliness. Keras is an API designed for human beings, not machines. It puts user experience front and center. Keras follows best practices for reducing cognitive load: it offers consistent & simple APIs, it minimizes the number of user actions required for common use cases, and it provides clear and actionable feedback upon user error.

- Modularity. A model is understood as a sequence or a graph of standalone, fully configurable modules that can be plugged together with as few restrictions as possible. In particular, neural layers, cost functions, optimizers, initialization schemes, activation functions and regularization schemes are all standalone modules that you can combine to create new models.

- Easy extensibility. New modules are simple to add (as new classes and functions), and existing modules provide ample examples. To be able to easily create new modules allows for total expressiveness, making Keras suitable for advanced research.

- Work with Python. No separate models configuration files in a declarative format. Models are described in Python code, which is compact, easier to debug, and allows for ease of extensibility.

3.4.4 TENSORFLOW

TensorFlow is a free and open-source software library for dataflow and differentiable programming across a range of tasks. It is a symbolic math library, and is also used for machine learning applications such as neural networks.

TensorFlow is Google Brain's second-generation system. Version 1.0.0 was released on February 11, 2017. While the reference implementation runs on single devices, TensorFlow can run on multiple CPUs and GPUs (with optional CUDA and SYCL extensions for general-purpose computing on graphics processing units). TensorFlow is available on 64-bit Linux, macOS, Windows, and mobile computing platforms including Android and iOS.

Its flexible architecture allows for the easy deployment of computation across a variety of platforms (CPUs, GPUs, TPUs), and from desktops to clusters of servers to mobile and edge devices.

TensorFlow computations are expressed as stateful dataflow graphs. The name TensorFlow derives from the operations that such neural networks perform on multidimensional data arrays, which are referred to as tensors. During the Google I/O Conference in June 2016, Jeff Dean stated that 1,500 repositories on GitHub mentioned TensorFlow, of which only 5 were from Google.

Tensor processing unit (TPU)

Tensor processing unit (TPU), an application-specific integrated circuit (a hardware chip) built specifically for machine learning and tailored for TensorFlow. TPU is a programmable AI accelerator designed to provide high throughput of low-precision arithmetic (e.g., 8-bit), and oriented toward using or running models rather than training them. Google announced they had been running TPUs inside their data centers for more than a year, and had found them to deliver an order of magnitude better-optimized performance per watt for machine learning.[17]

Google announced the second-generation, as well as the availability of the TPUs in Google Compute Engine.[18] The second-generation TPUs deliver up to 180 teraflops of performance, and when organized into clusters of 64 TPUs, provide up to 11.5 petaflops.

Google announced the third-generation TPUs delivering up to 420 teraflops of performance and 128 GB high bandwidth memory (HBM). Cloud TPU v3 Pods offer 100+ petaflops of performance and 32 TB HBM.[19]

Google announced that they were making TPUs available in beta on the Google Cloud Platform.[20]

Edge TPU

In July 2018, the Edge TPU was announced. Edge TPU is Google's purpose-built ASIC chip designed to run TensorFlow Lite machine learning (ML) models on small client computing devices such as smartphones[21] known as edge computing.

TensorFlow Lite

a software stack specifically for mobile development, TensorFlow Lite.[22] In January 2019, TensorFlow team released a developer preview of the mobile GPU inference engine with OpenGL ES 3.1 Compute Shaders on Android devices and Metal Compute Shaders on iOS devices.[23] In May 2019, Google announced that their TensorFlow Lite Micro (also known as TensorFlow Lite for Microcontrollers) and ARM's uTensor would be merging.[24]

Pixel Visual Core (PVC)

Google released the Google Pixel 2 which featured their Pixel Visual Core (PVC), a fully programmable image, vision and AI processor for mobile devices. The PVC supports TensorFlow for machine learning (and Halide for image processing)

Why TensorFlow

TensorFlow is an end-to-end open source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries and community resources that lets researchers push the state-of-the-art in ML and developers easily build and deploy ML powered applications.
An entire ecosystem to help you solve challenging, real-world problems with machine learning
Easy model building

TensorFlow offers multiple levels of abstraction so you can choose the right one for your needs. Build and train models by using the high-level Keras API, which makes getting started with TensorFlow and machine learning easy.

If you need more flexibility, eager execution allows for immediate iteration and intuitive debugging. For large ML training tasks, use the Distribution Strategy API for distributed training on different hardware configurations without changing the model definition.

Robust ML production anywhere

TensorFlow has always provided a direct path to production. Whether it's on servers, edge devices, or the web, TensorFlow lets you train and deploy your model easily, no matter what language or platform you use.

Use TensorFlow Extended (TFX) if you need a full production ML pipeline. For running inference on mobile and edge devices, use TensorFlow Lite. Train and deploy models in JavaScript environments using TensorFlow.js.

Powerful experimentation for research

Build and train state-of-the-art models without sacrificing speed or performance. TensorFlow gives you the flexibility and control with features like the Keras Functional API and Model Subclassing API for creation of complex topologies. For easy prototyping and fast debugging, use eager execution.

TensorFlow also supports an ecosystem of powerful add-on libraries and models to experiment with, including Ragged Tensors, TensorFlow Probability, Tensor2Tensor and BERT.

CHAPTER 4

EXISTING SYSTEM ANALYSIS AND ITS DISADVANTAGES

Overview of Existing System and Dataset Details

In existing system mainly, they started by using Plant Village dataset. They analyse 54,306 images of plant leaves, which have a spread of 38 class labels assigned to them. Each class label is a crop-disease pair, and they make an attempt to predict the crop-disease pair given just the image of the plant leaf. They resize image the images to 256 × 256 pixels, and we perform both the model optimization and predictions on these downscaled images. They have used three types of versions of datasets. First they have started with the color images dataset. Then they have used grey-scaled version of the Plant Village dataset. Final version they have used segmented version of dataset. Extra background information of image which might have the potential to introduce some inherent bias in the dataset. For segmented they removed the background information, which is shown in below figure.
(a) Leaf 1 color, (b) Leaf 1 grayscale, (c) Leaf 1 segmented, (d) Leaf 2 color, (e) Leaf 2 gray-scale, (f) Leaf 2 segmented.

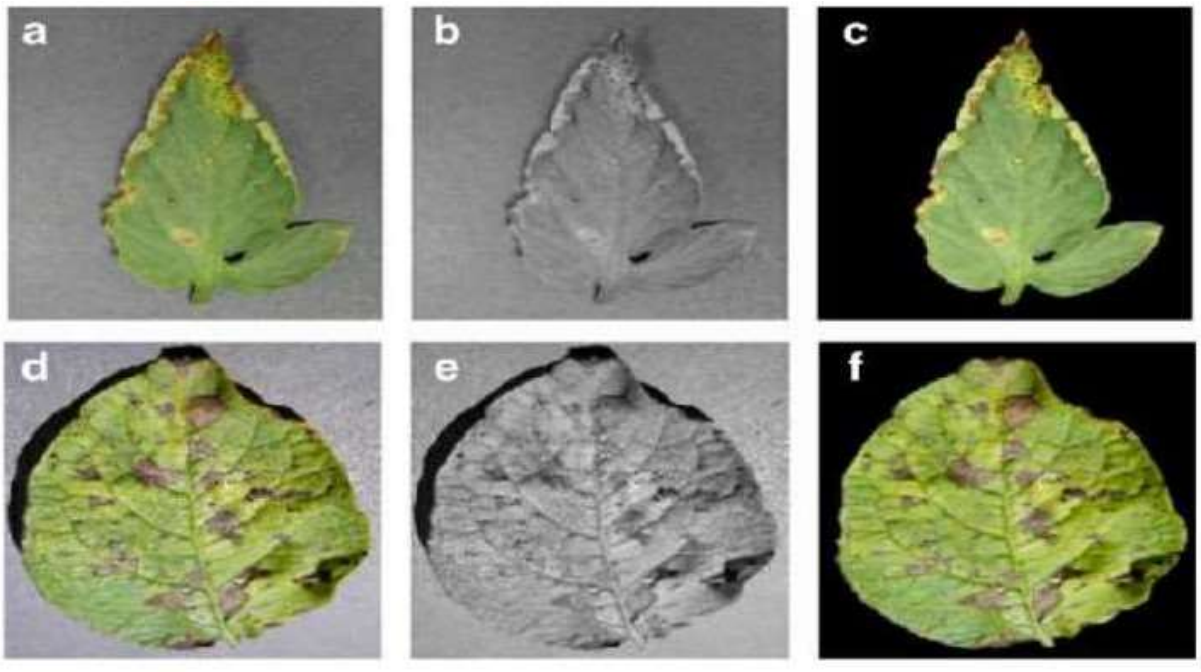


Fig 4.1: different version of crop-disease pair image

It should be notice that in many cases, the Plant Village dataset has multiple images of the

same leaf (taken from different orientations), and we have the mappings of such cases for 41,124 images out of the 54,306 images. these test-train splits, we make sure all the images of the same leaf go either in the training set or the testing set.

DISADVANTAGES

- The existing method for plant disease detection is simply naked eye observation by experts through which identification and detection of plant diseases is done.
- For doing so, a large team of experts as well as continuous monitoring of plants is required, which costs very high when we do with large farms.
- At the same time, in some countries, farmers do not have proper facilities or even idea as to how they can contact experts with regard to the type of disease. Due to which consulting experts even cost high as well as time consuming too.
- But when we think about an electronic expert system for leaf based plant diseases detection, in current scenario this kind of system is not available for day to day use.
- Other features like online marketplace, weather reports, soil information,etc are present but may not be available in a single package, means end users use this features from different service providers.

CHAPTER 5

PROPOSED SYSTEM DESIGN

- Our project is to detect the plant diseases and provide the solutions to recover from the leaf diseases. We planned to design our project with image processing system so that a person with lesser expertise in software should also be able to use it easily.
- In our proposed system we are providing a solution to recover from the leaf diseases and also show the affected part of the leaf by image processing technique along with concepts of Machine Learning.
- The existing system can only identify the type of diseases which affects the leaf. We will provide a result within seconds.
- Several images are collected for each disease that was classified into database images and input images. The primary attributes of the images are relied upon the shape and texture oriented features. The sample screenshots displays the plant disease detection using colour based segmentation model.
- The project gives suggestions as to which pesticide shall be used to treat the detected disease.

5.1 USE CASE DIAGRAM

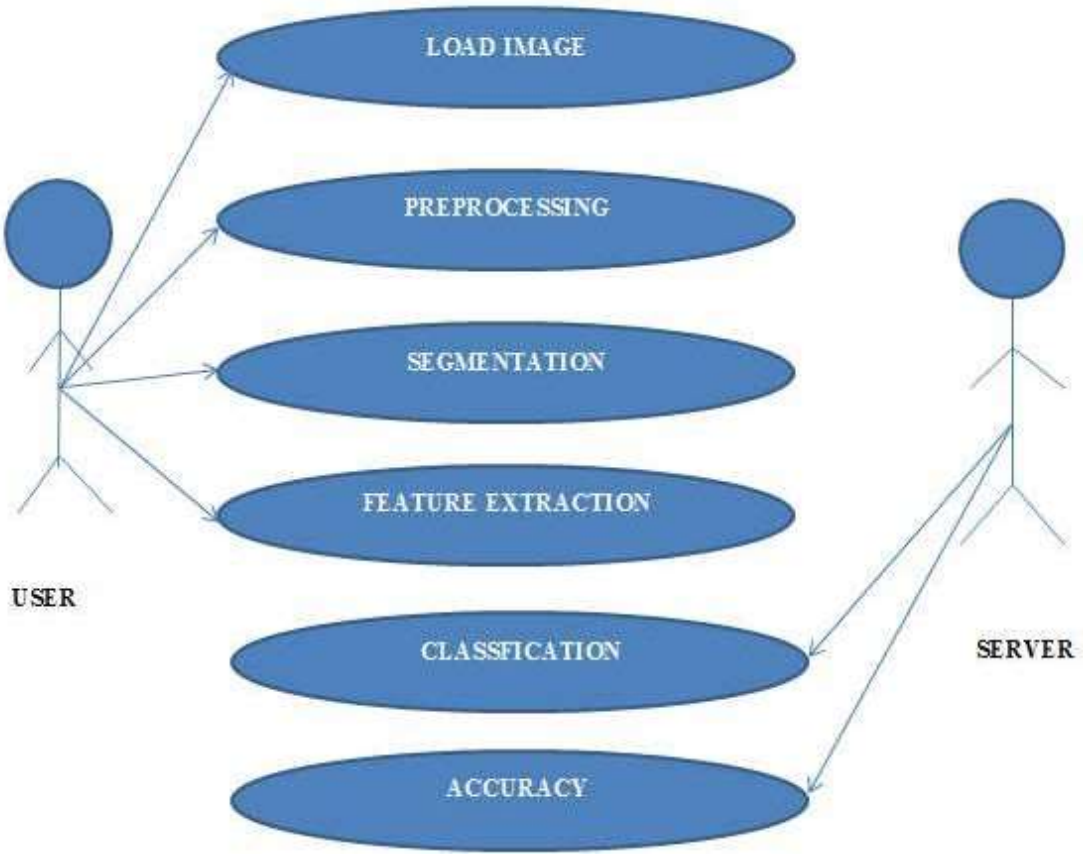


Fig 5.1: Use Case Diagram of leaf disease detection

5.2 OUTLINE OF THE SYSTEM

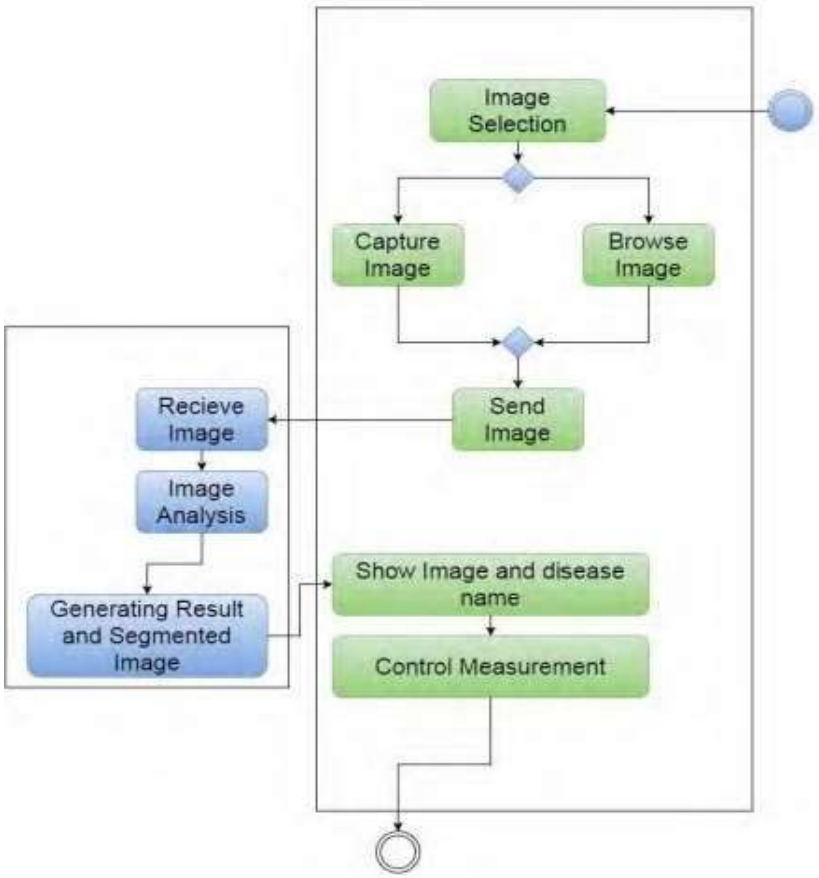
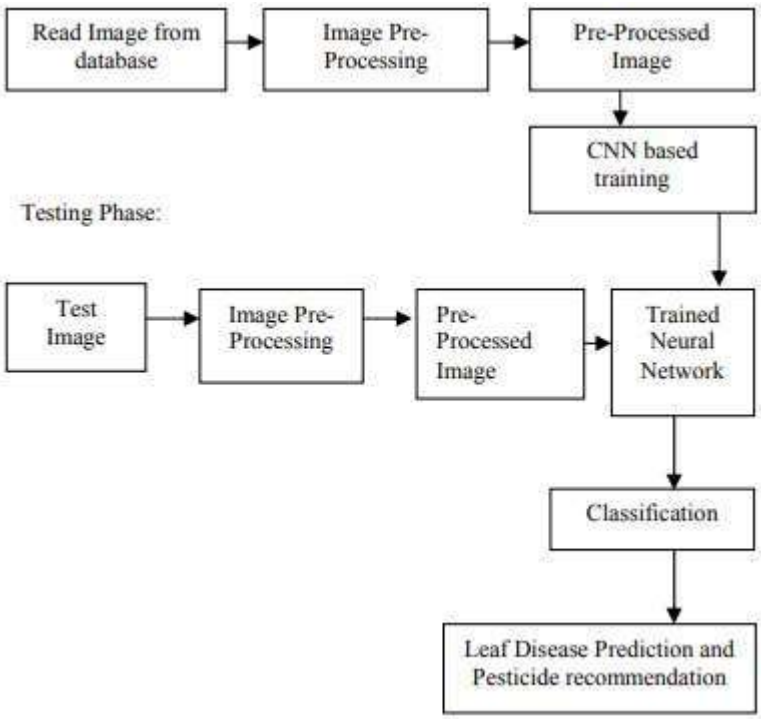


Fig 5.2: Outline of the system

5.3 SYSTEM ARCHITECTURE



5.4 WORKING

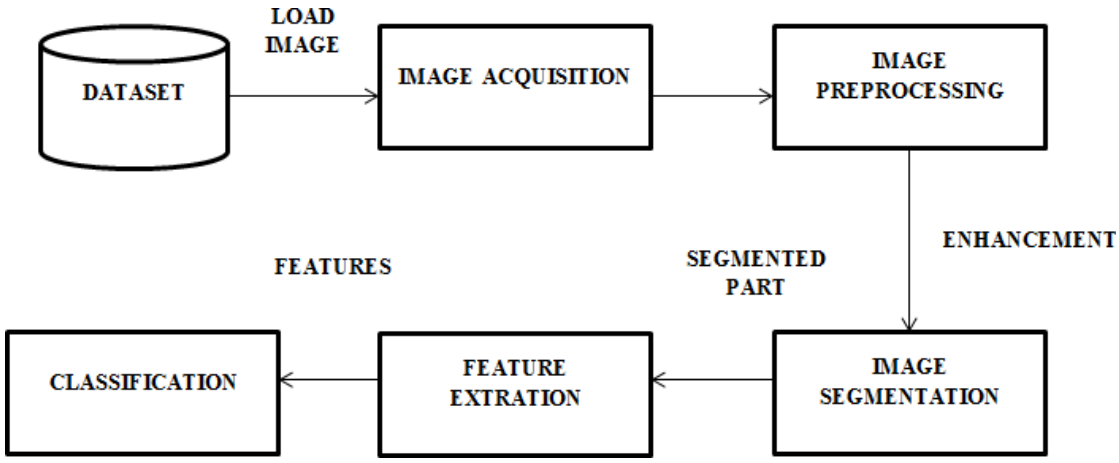


Fig 5.3 Working

IMAGE ACQUISITION

For training, Image is taken from database. And for testing, you can take image from camera at real time but in this project, we made a particular folder on desktop from that image will be fetched by GUI screen and send through java web services i.e. tomcat server to server side system on which pre-processing is done and later on algorithm test that particular image.

Image Pre-Processing

Image should be processed before sending to the algorithm for testing and training purpose. For that purpose, in this project image is scaled or resize into 150 x 150 dimensions. As we used color image so that we don't need any color conversion techniques and that pre-processed image is directly passed to algorithm for training and testing purpose.

Image Classification

We used CNN for image classification. CNNs are biologically inspired models inspired by research by D. H. Hubel and T. N. Wiesel. They proposed an explanation for the way in which mammals visually perceive the world around them using a layered architecture of neurons in the brain, and these in turn inspired engineers to attempt to develop similar pattern recognition mechanisms in computer vision.

For the implementation of CNN, we have used keras library on top of tensorflow. CNN receives the image as a matrix of pixel values. A sequence of convolution, maxpooling and normalizationis done in several layers of CNN and is finally regularized.

Image datasets

Every image is a matrix of pixel values. The range of values that can be encoded in each pixel depends upon its bit size. Most commonly, we have 8 bit or 1 Byte-sized pixels. Thus the possible range of values a single pixel can represent is [0, 255]. However, with coloured images, particularly RGB (Red, Green, Blue)-based images, the presence of separate colour channels (3 in the case of RGB images) introduces an additional 'depth' field to the data, making the input

3-dimensional. Hence, for a given RGB image of size, say 255×255 (Width x Height) pixels, we'll have 3 matrices associated with each image, one for each of the color channels. Thus the image in it's entirety, constitutes a 3-dimensional structure called the Input Volume (255x255x3).

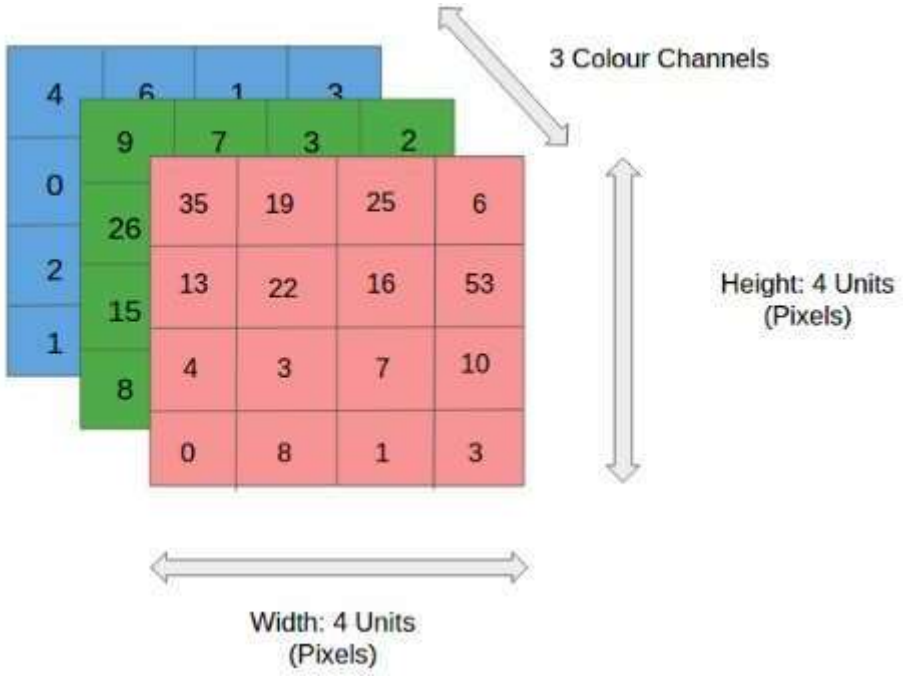


Fig 5.4: Image as pixel matrix

Convolution

A convolution is an orderly procedure where two sources of information are intertwined. A kernel (also called a filter) is a smaller-sized matrix in comparison to the input dimensions of the image, that consists of real valued entries.

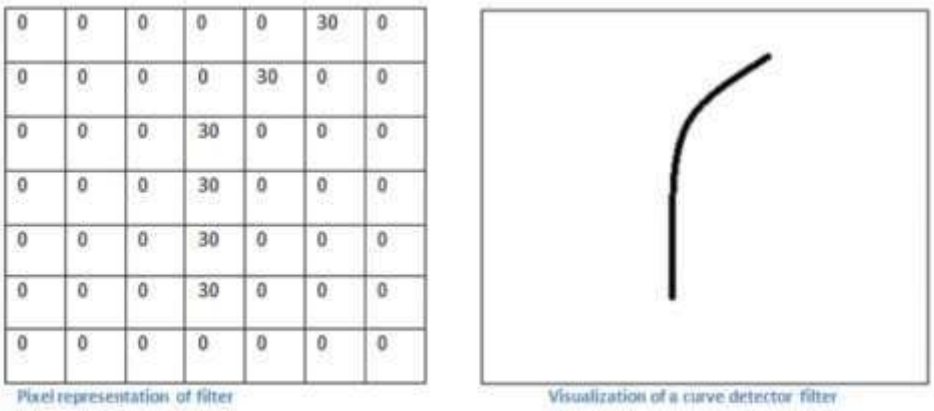


Fig 5.5: Pixel representation of filter

Kernels are then convolved with the input volume to obtain so-called 'activation maps' (also called feature maps). We compute the dot product between the kernel and the input matrix. The convolved value obtained by summing the resultant terms from the dot product forms a single

entry in the activation matrix. The patch selection is then slided (towards the right, or downwards when the boundary of the matrix is reached) by a certain amount called the 'stride' value, and the process is repeated till the entire input image has been processed.

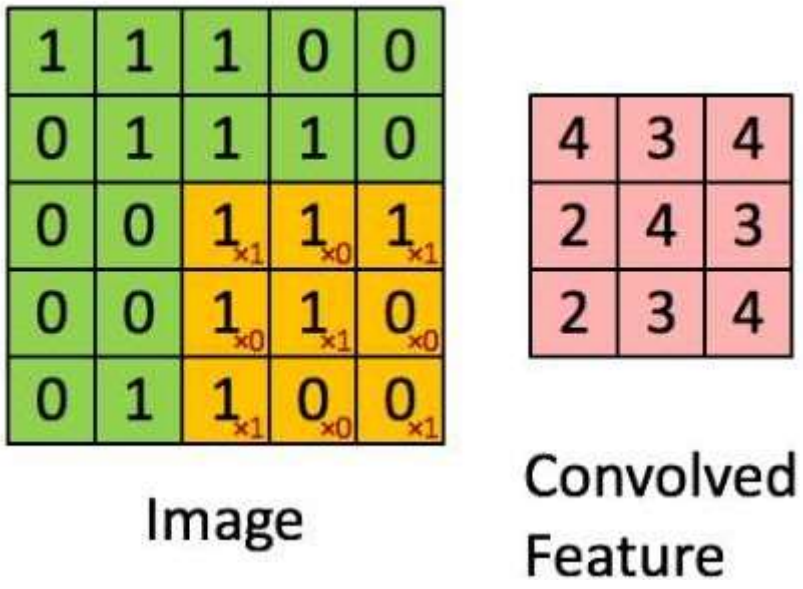


Fig 5.6: Convolved Feature of Image

Pooling

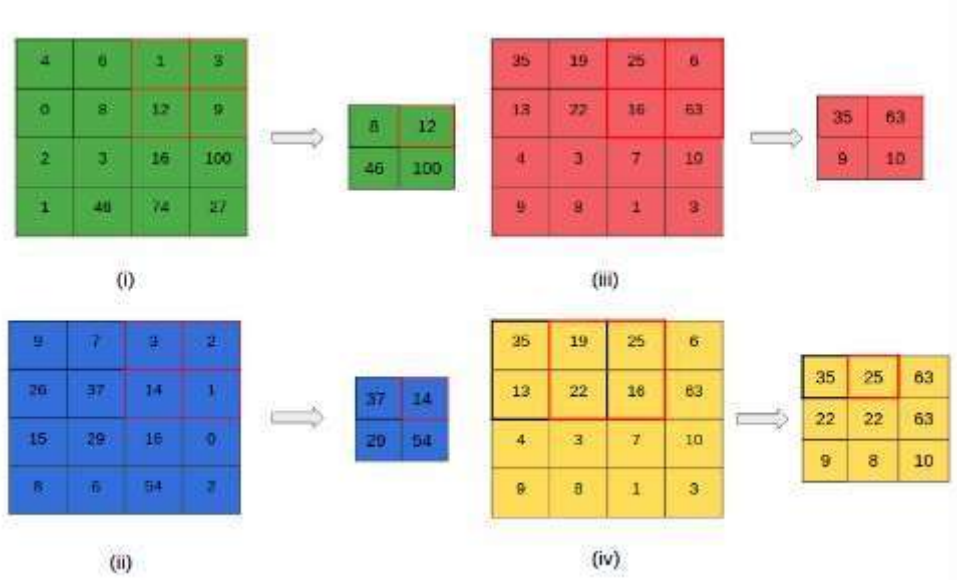


Fig 5.7: Pooling of features

Pooling reduces the spatial dimensions (Width x Height) of the Input Volume for the next Convolutional Layer. It does not affect the depth dimension of the Volume. The transformation is either performed by taking the maximum value from the values observable in the window

(called ‘max pooling’), or by taking the average of the values. Max pooling has been favored over others due to its better performance characteristics.

Normalization

Normalization turns all the negative values to 0 so that a matrix have no negative values. We have used ReLU activation in our case. A stack of images becomes a stack of images with no negative values.

Regularization

Regularization is a vital feature in almost every state-of-the-art neural network implementation. To perform dropout on a layer, you randomly set some of the layer's values to 0 during forward propagation. Dropout forces an artificial neural network to learn multiple independent representations of the same data by alternately randomly disabling neurons in the learning phase.

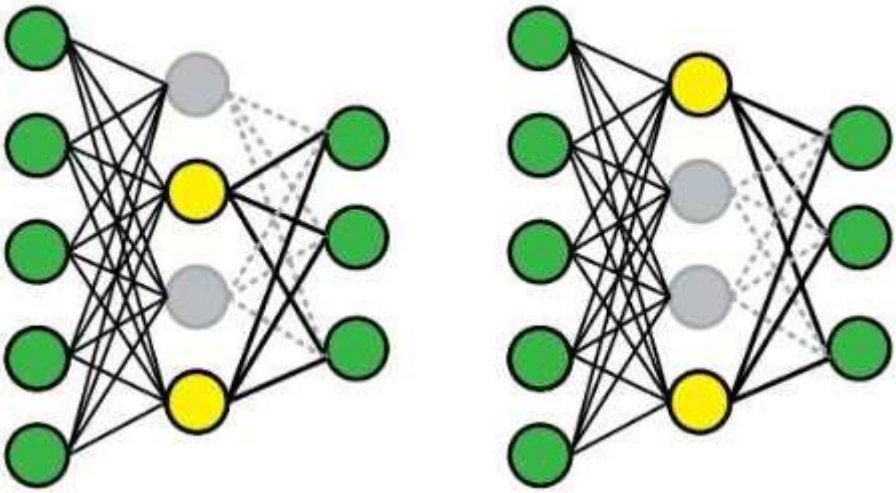


Fig 5.8: Regularization

Convolutional Neural Network

Once pre-processing is done, then CNN is used for training purpose and after that we get trained model. That CNN method is written with help of tensor flow. By using this model, we classify the image that system is getting after pre-processing of testing image. Then we get particular disease name or healthy leaf name if there is no disease on that leaf and that disease name is send to GUI screen and with the help of that disease name we get particular pesticide name which help farmer to take respective action in order to decrease percentage of disease.

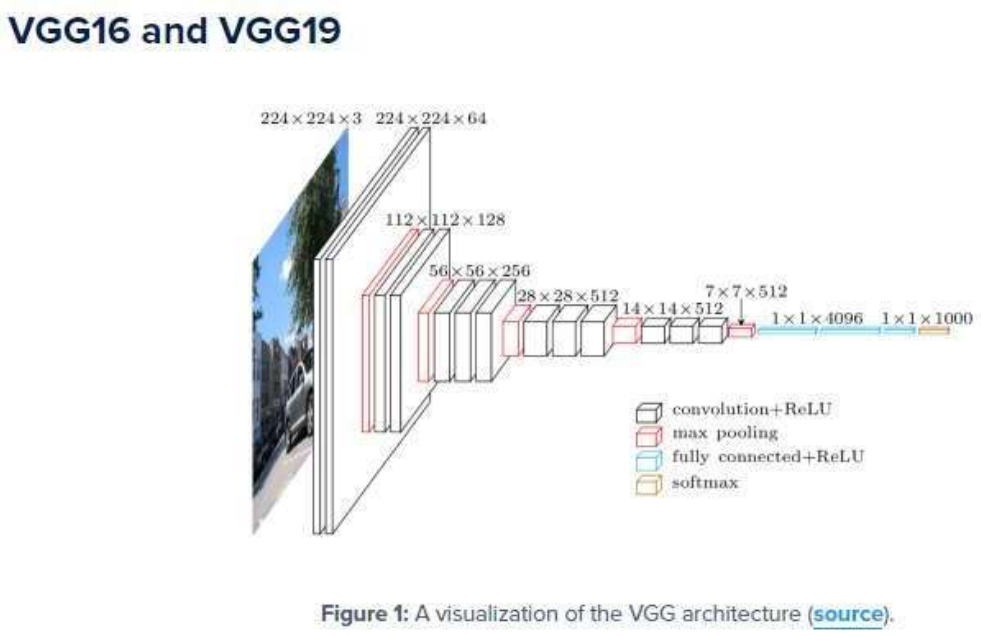


Figure 1: A visualization of the VGG architecture (source).

Each layer in a CNN applies a different set of filters, typically hundreds or thousands of them, and combines the results, feeding the output into the next layer in the network. During training, a

CNN automatically learns the values for these filters.

In the context of image classification, our CNN may learn to:

- Detect edges from raw pixel data in the first layer.
- Use these edges to detect shapes (i.e., "blobs") in the second layer.
- Use these shapes to detect higher-level features such as facial structures, parts of a car, etc.in the highest layers of the network.

The last layer in a CNN uses these higher-level features to make predictions regarding the Contents of the image.

In terms of deep learning, an (image) convolution is an element-wise multiplication of two matrices followed by a sum.

1. Take two matrices (which both have the same dimensions).
2. Multiply them, element-by-element (i.e., not the dot product, just a simple multiplication).
3. Sum the elements together.

Kernels

Again, let's think of an image as a big matrix and a kernel as a tiny matrix (at least in respect to the

original "big matrix" image), depicted in Figure 11.1. As the figure demonstrates, we are sliding

the kernel (red region) from left-to-right and top-to-bottom along the original image. At each (x;y)-coordinate of the original image, we stop and examine the neighborhood of pixels located at

the center of the image kernel. We then take this neighborhood of pixels, convolve them with the
kernel, and obtain a single output value. The output value is stored in the output image at the same
(x;y)-coordinates as the center of the kernel.
If this sounds confusing, no worries, we'll be reviewing an example in the next section. But before we dive into an example, let's take a look at what a kernel looks like (Figure 11.3):

$$K = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{11.3}$$

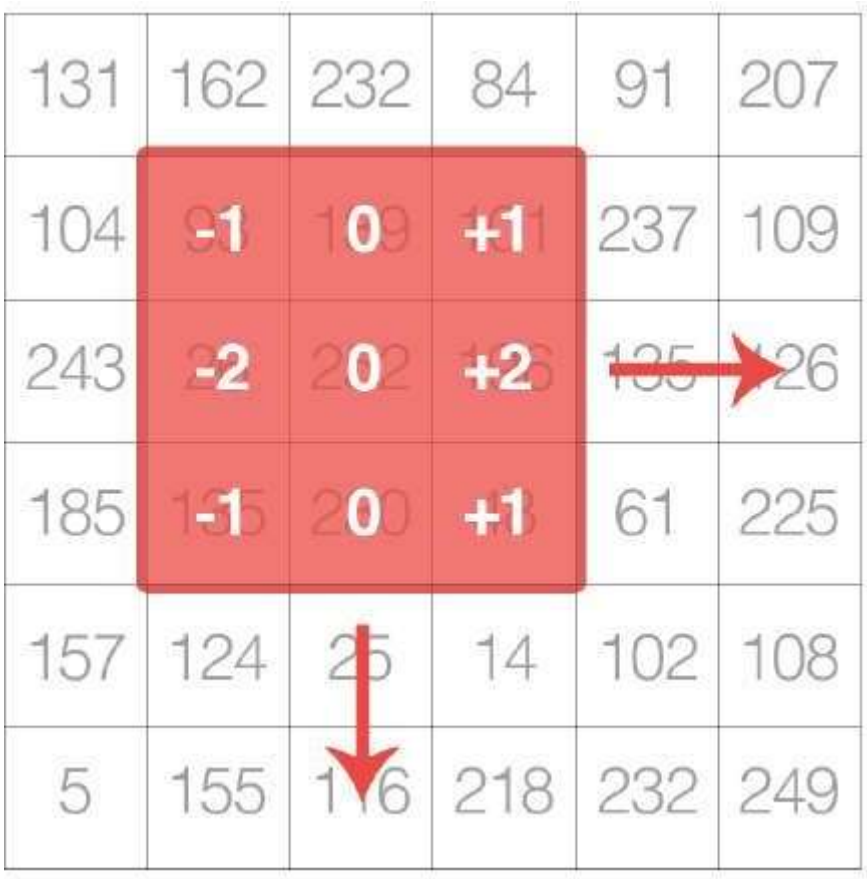


Figure 11.1: A kernel can be visualized as a small matrix that slides across, from left-to-right and
top-to-bottom, of a larger image. At each pixel in the input image, the neighborhood of the image
is convolved with the kernel and the output stored.

We use an odd kernel size to ensure there is a valid integer (x;y)-coordinate at the center of the image (Figure 11.2). On the left, we have a 3_3 matrix. The center of the matrix is located at

x = 1;y = 1 where the top-left corner of the matrix is used as the origin and our coordinates are zero-indexed. But on the right, we have a 2_2 matrix. The center of this matrix would be located

at x = 0:5;y = 0:5.

But as we know, without applying interpolation, there is no such thing as pixel location (0:5;0:5) – our pixel coordinates must be integers! This reasoning is exactly why we use odd kernel

sizes: to always ensure there is a valid (x;y)-coordinate at the center of the kernel.

Example of Convolution

Now that we have discussed the basics of kernels, let's discuss the actual convolution operation and

see an example of it actually being applied to help us solidify our knowledge. In image processing,

a convolution requires three components:

1. An input image.
2. A kernel matrix that we are going to apply to the input image.

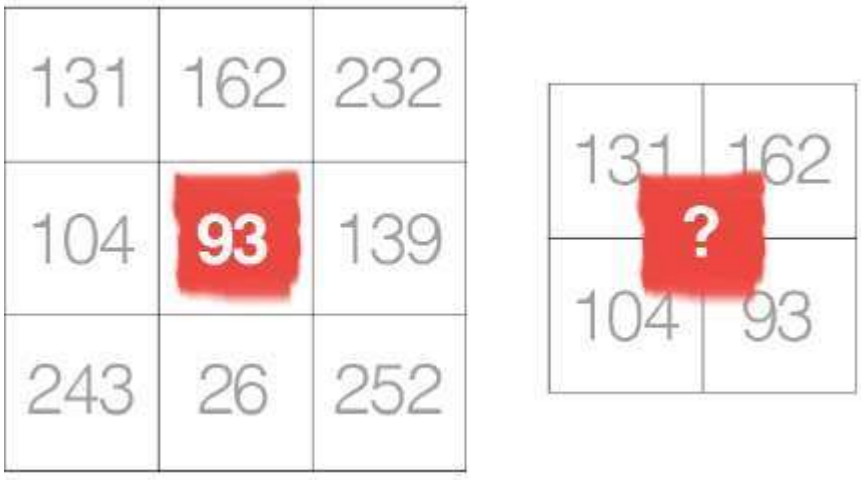


Figure 11.2: Left: The center pixel of a 3_3 kernel is located at coordinate (1;1) (highlighted in red). Right: What is the center coordinate of a kernel of size 2_2?

3. An output image to store the output of the image convolved with the kernel.

Convolution (i.e., cross-correlation) is actually very easy. All we need to do is:

1. Select an (x;y)-coordinate from the original image.
2. Place the center of the kernel at this (x;y)-coordinate.
3. Take the element-wise multiplication of the input image region and the kernel, then sum up the values of these multiplication operations into a single value. The sum of these multiplications is called the kernel output.
4. Use the same (x;y)-coordinates from Step #1, but this time, store the kernel output at the

same (x;y)-location as the output image.

Below you can find an example of convolving (denoted mathematically as the ? operator) a 3_3 region of an image with a 3_3 kernel used for blurring:

$$O_{i,j} = \frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix} * \begin{bmatrix}93 & 139 & 101\\ 26 & 252 & 196\\ 135 & 230 & 18\end{bmatrix} = \begin{bmatrix}1/9\times 93 & 1/9\times 139 & 1/9\times 101\\ 1/9\times 26 & 1/9\times 252 & 1/9\times 196\\ 1/9\times 135 & 1/9\times 230 & 1/9\times 18\end{bmatrix} \quad (11.4)$$

Therefore,

$$O_{i,j} = \sum \begin{bmatrix}10.3 & 15.4 & 11.2\\ 2.8 & 28.0 & 21.7\\ 15.0 & 25.5 & 2.0\end{bmatrix} \approx 132. \quad (11.5)$$

After applying this convolution, we would set the pixel located at the coordinate (i; j) of the output image O to Oi; j = 132.

That’s all there is to it! Convolution is simply the sum of element-wise matrix multiplication between the kernel and neighborhood that the kernel covers of the input image.

Layer Types

There are many types of layers used to build Convolutional Neural Networks, but the ones you are most likely to encounter include:

- Convolutional (CONV)
- Activation (ACT or RELU, where we use the same of the actual activation function)
- Pooling (POOL)
- Fully-connected (FC)
- Batch normalization (BN)
- Dropout (DO)

Stacking a series of these layers in a specific manner yields a CNN. We often use simple text diagrams to describe a CNN: INPUT => CONV => RELU => FC => SOFTMAX

Here we define a simple CNN that accepts an input, applies a convolution layer, then an activation layer, then a fully-connected layer, and, finally, a softmax classifier to obtain the output classification probabilities. The SOFTMAX activation layer is often omitted from the network diagram as it is assumed it directly follows the final FC.

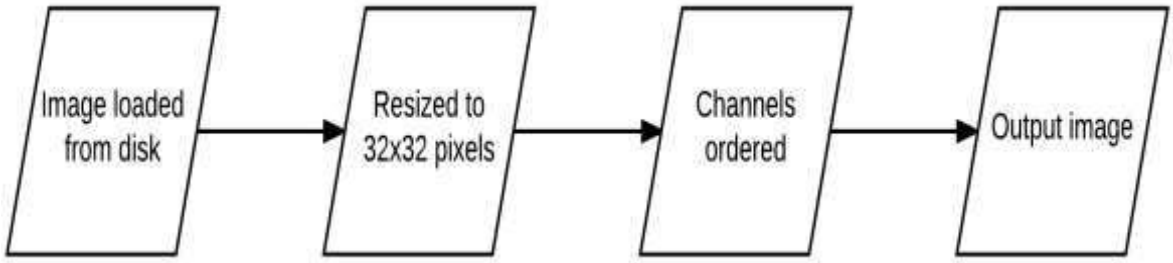
Of these layer types, CONV and FC, (and to a lesser extent, BN) are the only layers that contain parameters that are learned during the training process. Activation and dropout layers are not considered true “layers" themselves, but are often included in network diagrams to make the architecture explicitly clear. Pooling layers (POOL), of equal importance as CONV and FC, are also included in network diagrams as they have a substantial impact on the spatial dimensions of an image as it moves through a CNN.

CONV, POOL, RELU, and FC are the most important when defining your actual network architecture. That's not to say that the other layers are not critical, but take a backseat to this critical set of four as they define the actual architecture itself.

Activation functions themselves are practically assumed to be part of the architecture, When defining CNN architectures we often omit the activation layers from a table/diagram to save space; however, the activation layers are implicitly assumed to be part of the architecture.

- **INPUT:**
  Plant leaf image
- **OUTPUT:**
  The result of input image segmentation for a plant disease detection system is to preserve only the infected area in the output image for detection purpose . The disease name along with the suggestions to cure the disease will be displayed as output..

5.5 DEEP LEARNING ARCHITECTURE AND WORKING



Scaling and Aspect Ratios

Scaling, or simply resizing, is the process of increasing or decreasing the size of an image in terms of width and height. When resizing an image, it's important to keep in mind the aspect ratio image pre-processing pipeline that (1) loads an image from disk, (2)
resizes it to 32_32 pixels, (3) orders the channel dimensions, and (4) outputs the image.

THE FOUR STEPS TO CONSTRUCTING A DEEP LEARNING MODEL

Step 1: Gather Your Dataset

The first component of building a deep learning network is to gather our initial dataset. We need the images themselves as well as the labels associated with each image. These labels should come from a finite set of categories, such as: categories = dog, cat, panda.

Furthermore, the number of images for each category should be approximately uniform (i.e., the same number of examples per category). If we have twice the number of cat images than dog images, and five times the number of panda images than cat images, then our classifier will become naturally biased to overfitting into these heavily-represented categories.

Class imbalance is a common problem in machine learning and there exist a number of ways to overcome it. We'll discuss some of these methods later, but keep in mind the best method to avoid learning problems due to class imbalance is to simply avoid class imbalance entirely.

Step 2: Split Your Dataset

Now that we have our initial dataset, we need to split it into two parts:

1. A training set
2. A testing set

A training set is used by our classifier to "learn" what each category looks like by making predictions on the input data and then correct itself when predictions are wrong. After the classifier has been trained, we can evaluate the performing on a testing set.

It's extremely important that the training set and testing set are independent of each other and do not overlap! If you use your testing set as part of your training data, then your classifier has an unfair advantage since it has already seen the testing examples before and "learned" from them. Instead, you must keep this testing set entirely separate from your training process and use it only to evaluate your network.

Common split sizes for training and testing sets include 66:6%33:3%, 75%=25%, and 90%=10%, respectively. (Figure 4.7):

Figure 5.9: Examples of common training and testing data splits.

These data splits make sense, but what if you have parameters to tune? Neural networks have a number of knobs and levers (ex., learning rate, decay, regularization, etc.) that need to be tuned

and dialed to obtain optimal performance. We'll call these types of parameters hyperparameters, and it's critical that they get set properly. In practice, we need to test a bunch of these hyperparameters and identify the set of parameters that works the best. You might be tempted to use your testing data to tweak these values, but again, this is a major no-no! The test set is only used in evaluating the performance of your network. Instead, you should create a third data split called the validation set. This set of the data (normally) comes from the training data and is used as "fake test data" so we can tune our hyperparameters. Only after have we determined the hyperparameter values using the validation set do we move on to collecting final accuracy results in the testing data. We normally allocate roughly 10-20% of the training data for validation. If splitting your data into chunks sounds complicrated, it's actually not.

Step 3: Train Your Network

Given our training set of images, we can now train our network. The goal here is for our network to learn how to recognize each of the categories in our labeled data. When the model makes a mistake, it learns from this mistake and improves itself. So, how does the actual "learning" work? In general, we apply a form of gradient descent.

Step 4: Evaluate

Last, we need to evaluate our trained network. For each of the images in our testing set, we present them to the network and ask it to predict what it thinks the label of the image is. We then tabulate the predictions of the model for an image in the testing set.

Finally, these model predictions are compared to the ground-truth labels from our testing set. The ground-truth labels represent what the image category actually is. From there, we can compute the number of predictions our classifier got correct and compute aggregate reports such as precision, recall, and f-measure, which are used to quantify the performance of our network as a whole.

CHAPTER 6

MECHANISMS AND FLOW SEQUENCE

6.1 BASIC WORKFLOW OF THE SYSTEM

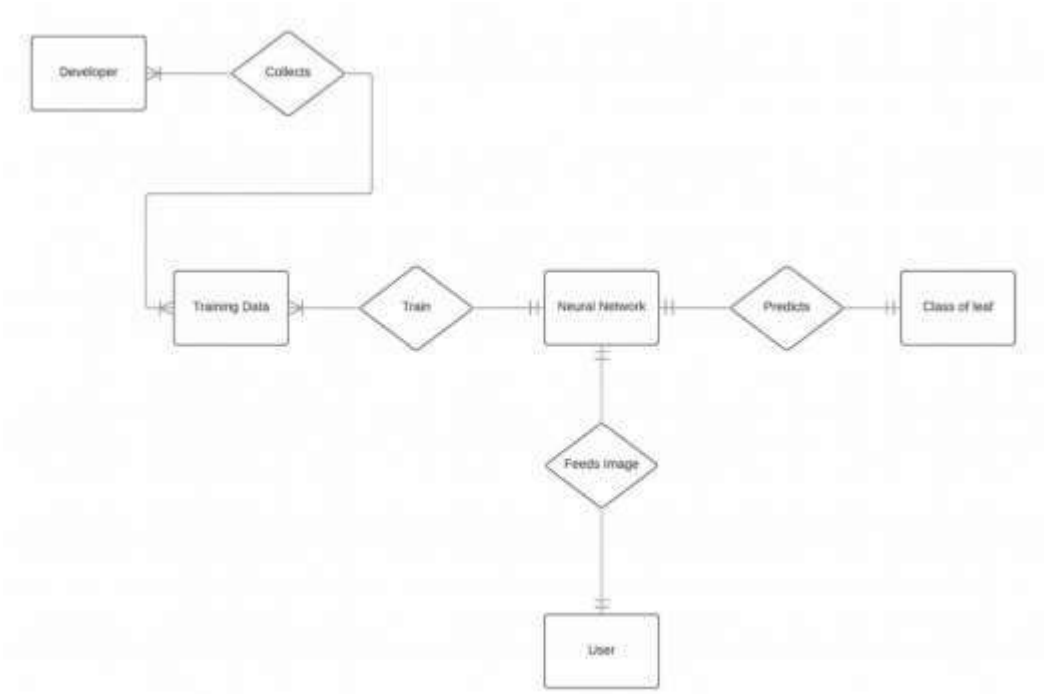


Fig 6.1: Work Flow Diagram Of the Leaf Disease Detection System as a whole

- A data flow diagram is the graphical representation of the flow of data through an information system. DFD is very useful in understanding a system and can be efficiently used during analysis.
- So, the Data Flow Diagrams can be successfully used for visualization of data processing or structured design.

6.2 CONVOLUTION LAYERS

The CONV layer is the core building block of a Convolutional Neural Network. The CONV layer parameters consist of a set of K learnable filters (i.e., “kernels”), where each filter has a width and a height, and are nearly always square. These filters are small (in terms of their spatial dimensions) but extend throughout the full depth of the volume.

For inputs to the CNN, the depth is the number of channels in the image (i.e., a depth of three when working with RGB images, one for each channel). For volumes deeper in the network,the depth will be the number of filters applied in the previous layer. To make this concept more clear, let’s consider the forward-pass of a CNN, where we convolve each of the K filters across the width and height of the input volume, just like we did in Section

11.1.5 above. More simply, we can think of each of our K kernels sliding across the input region,

computing an element-wise multiplication, summing, and then storing the output value in a 2-dimensional activation map, such as in Figure 11.6.

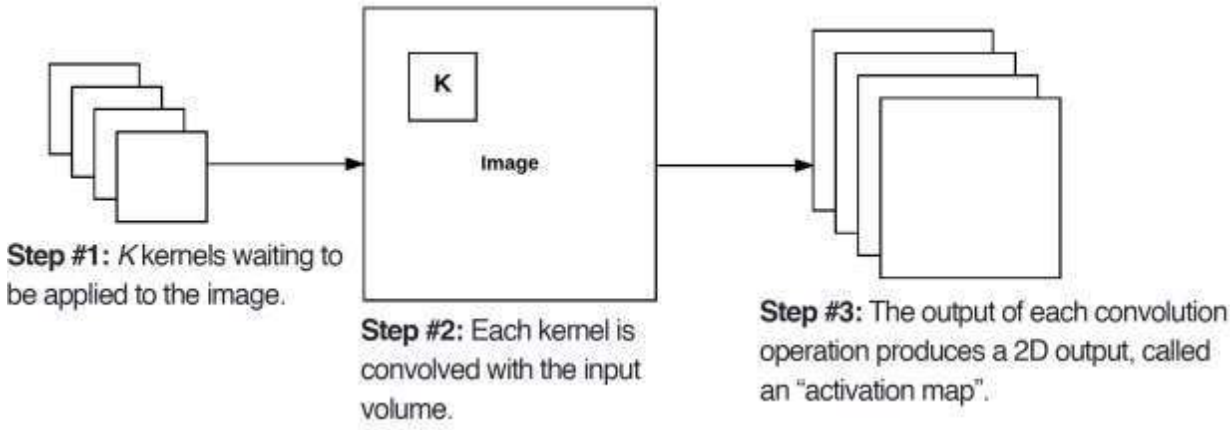


Figure 6.2: Left: At each convolutional layer in a CNN, there are K kernels applied to the input volume. Middle: Each of the K kernels is convolved with the input volume. Right: Each kernel produces an 2D output, called an activation map.

After applying all K filters to the input volume, we now have K, 2-dimensional activation maps. We then stack our K activation maps along the depth dimension of our array to form the final output
volume (Figure 11.7).

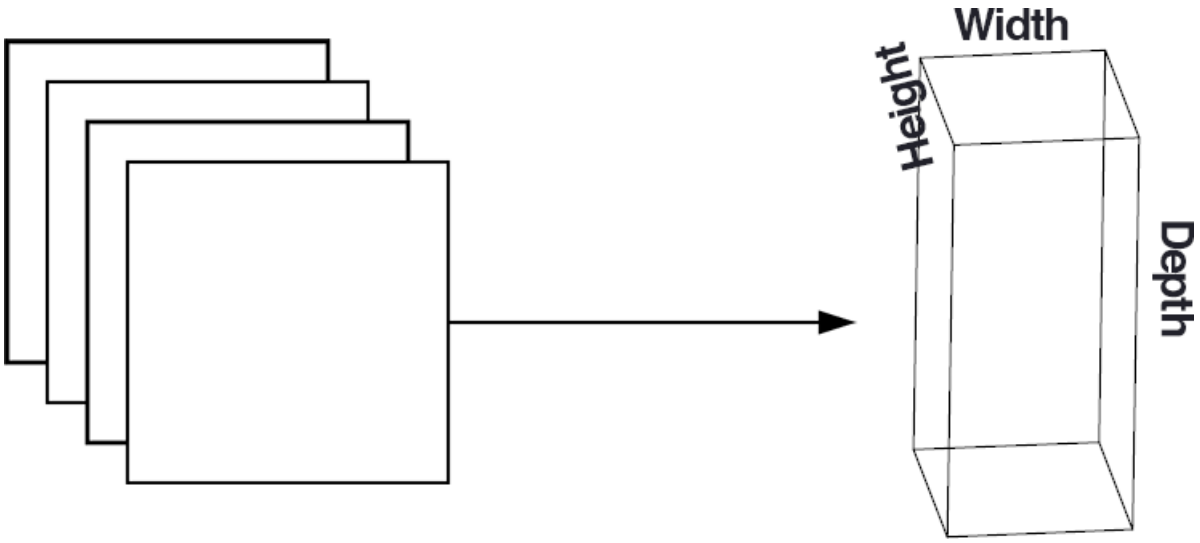


Figure 6.3: After obtaining the K activation maps, they are stacked together to form the input volume to the next layer in the network.

Every entry in the output volume is thus an output of a neuron that “looks” at only a small region of the input. In this manner, the network “learns” filters that activate when they see a specific type of feature at a given spatial location in the input volume. In lower layers of the network, filters may activate when they see edge-like or corner-like regions.

Then, in the deeper layers of the network, filters may activate in the presence of high-level features, such as parts of the face, the paw of a dog, the hood of a car, etc. This activation concept goes back to our neural network analogy in Chapter 10 – these neurons are becoming “excited” and “activating” when they see a particular pattern in an input image.

The concept of convolving a small filter with a large(r) input volume has special meaning in Convolutional Neural Networks – specifically, the local connectivity and the receptive field of a neuron. When working with images, it’s often impractical to connect neurons in the current volume to all neurons in the previous volume – there are simply too many connections and too many weights, making it impossible to train deep networks on images with large spatial dimensions.

Instead, when utilizing CNNs, we choose to connect each neuron to only a local region of the input volume – we call the size of this local region the receptive field (or simply, the variable F) of the neuron.

To make this point clear, let’s return to our CIFAR-10 dataset where the input volume as an input size of 32_32_3. Each image thus has a width of 32 pixels, a height of 32 pixels, and a depth of 3 (one for each RGB channel). If our receptive field is of size 3_3, then each neuron in the CONV layer will connect to a 3_3 local region of the image for a total of 3_3_3 = 27 weights (remember, the depth of the filters is three because they extend through the full depth of the input image, in this case, three channels).

Now, let’s assume that the spatial dimensions of our input volume have been reduced to a smaller size, but our depth is now larger, due to utilizing more filters deeper in the network, such that the volume size is now 16_16_94. Again, if we assume a receptive field of size 3_3, then every neuron in the CONV layer will have a total of 3_3_94=846 connections to the input volume.

Simply put, the receptive field F is the size of the filter, yielding an F _F kernel that is convolved with the input volume.

At this point we have explained the connectivity of neurons in the input volume, but not the arrangement or size of the output volume. There are three parameters that control the size of an output volume: the depth, stride, and zero-padding size, each of which we’ll review below.

Depth

The depth of an output volume controls the number of neurons (i.e., filters) in the CONV layer that connect to a local region of the input volume. Each filter produces an activation map that "activate" in the presence of oriented edges or blobs or color.

For a given CONV layer, the depth of the activation map will be K, or simply the number of filters we are learning in the current layer. The set of filters that are "looking at" the same (x;y) location of the input is called the depth column.

Zero-padding

As we know from Section 11.1.5, we need to "pad" the borders of an image to retain the original image size when applying a convolution – the same is true for filters inside of a CNN. Using zero-padding, we can "pad" our input along the borders such that our output volume size matches our input volume size. The amount of padding we apply is controlled by the parameter P.

This technique is especially critical when we start looking at deep CNN architectures that apply multiple CONV filters on top of each other. To visualize zero-padding, again refer to Table 11.1 where we applied a 3_3 Laplacian kernel to a 5_5 input image with a stride of S = 1.

We can see in Table 11.3 (left) how the output volume is smaller (3_3) than the input volume (5_5) due to the nature of the convolution operation. If we instead set P = 1, we can pad our input volume with zeros (middle) to create a 7_7 volume and then apply the convolution operation, leading to an output volume size that matches the original input volume size of 5_5 (right).

Without zero padding, the spatial dimensions of the input volume would decrease too quickly, and we wouldn't be able to train deep networks (as the input volumes would be too tiny to learn any useful patterns from).

Putting all these parameters together, we can compute the size of an output volume as a function of the input volume size (W, assuming the input images are square, which they nearly always are),

the receptive field size F, the stride S, and the amount of zero-padding P. To construct a valid CONV

Activation Layers

After each CONV layer in a CNN, we apply a nonlinear activation function, such as ReLU, ELU, or

any of the other Leaky ReLU variants mentioned in Chapter 10. We typically denote activation layers as RELU in network diagrams as since ReLU activations are most commonly used, we may

also simply state ACT – in either case, we are making it clear that an activation function is being applied inside the network architecture.

Activation layers are not technically "layers" (due to the fact that no parameters/weights are learned inside an activation layer) and are sometimes omitted from network architecture diagrams as it's assumed that an activation immediately follows a convolution.

In this case, authors of publications will mention which activation function they are using after each CONV layer somewhere in their paper. As an example, consider the following network architecture: INPUT => CONV => RELU => FC.

To make this diagram more concise, we could simply remove the RELU component since it's assumed that an activation always follows a convolution: INPUT => CONV => FC. I personally do

not like this and choose to explicitly include the activation layer in a network diagram to make it clear when and what activation function I am applying in the network.

An activation layer accepts an input volume of sizeWinput _Hinput _Dinput and then applies the given activation function (Figure 11.9). Since the activation function is applied in an element-wise

manner, the output of an activation layer is always the same as the input dimension,Winput =Wout put ,

Hinput = Hout put , Dinput = Dout put .

Pooling Layers

There are two methods to reduce the size of an input volume – CONV layers with a stride > 1 (which

we've already seen) and POOL layers. It is common to insert POOL layers in-between consecutive

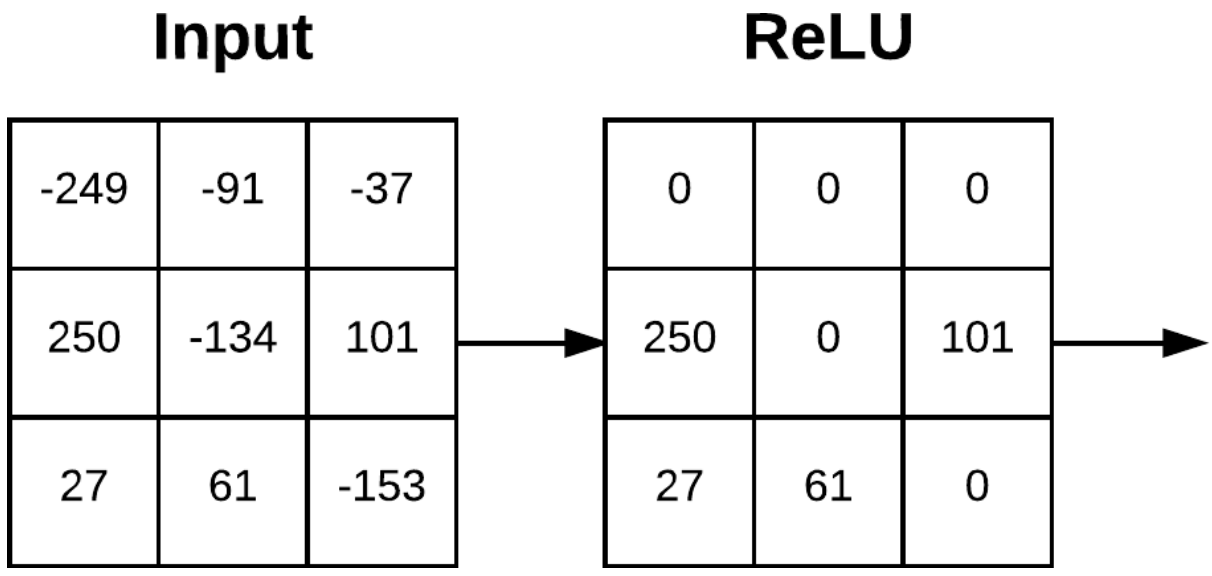


Figure 6.4: An example of an input volume going through a ReLU activation, max(0;x).

Activations

are done in-place so there is no need to create a separate output volume although it is easy to visualize the flow of the network in this manner.

CONV layers in a CNN architectures:

INPUT => CONV => RELU => POOL => CONV => RELU => POOL => FC

The primary function of the POOL layer is to progressively reduce the spatial size (i.e., width and height) of the input volume. Doing this allows us to reduce the amount of parameters and computation in the network – pooling also helps us control overfitting.

POOL layers operate on each of the depth slices of an input independently using either the max or average function. Max pooling is typically done in the middle of the CNN architecture to reduce spatial size, whereas average pooling is normally used as the final layer of the network

(e.x., GoogLeNet, SqueezeNet, ResNet) where we wish to avoid using FC layers entirely. The most

common type of POOL layer is max pooling, although this trend is changing with the introduction of more exotic micro-architectures.

Typically we'll use a pool size of 2_2, although deeper CNNs that use larger input images (> 200 pixels) may use a 3_3 pool size early in the network architecture. We also commonly set the stride to either S = 1 or S = 2. Figure 11.10 (heavily inspired by Karpathy et al. [121]) follows an example of applying max pooling with 2_2 pool size and a stride of S = 1. Notice for every 2_2 block, we keep only the largest value, take a single step (like a sliding window), and apply the operation again – thus producing an output volume size of 3_3.

We can further decrease the size of our output volume by increasing the stride – here we apply S = 2 to the same input (Figure 11.10, bottom). For every 2_2 block in the input, we keep only the largest value, then take a step of two pixels, and apply the operation again. This pooling allows

us to reduce the width and height by a factor of two, effectively discarding 75% of activations from

the previous layer.

In summary, POOL layers Accept an input volume of size Winput _Hinput _Dinput . They then require two parameters:

- The receptive field size F (also called the "pool size").
- The stride S.

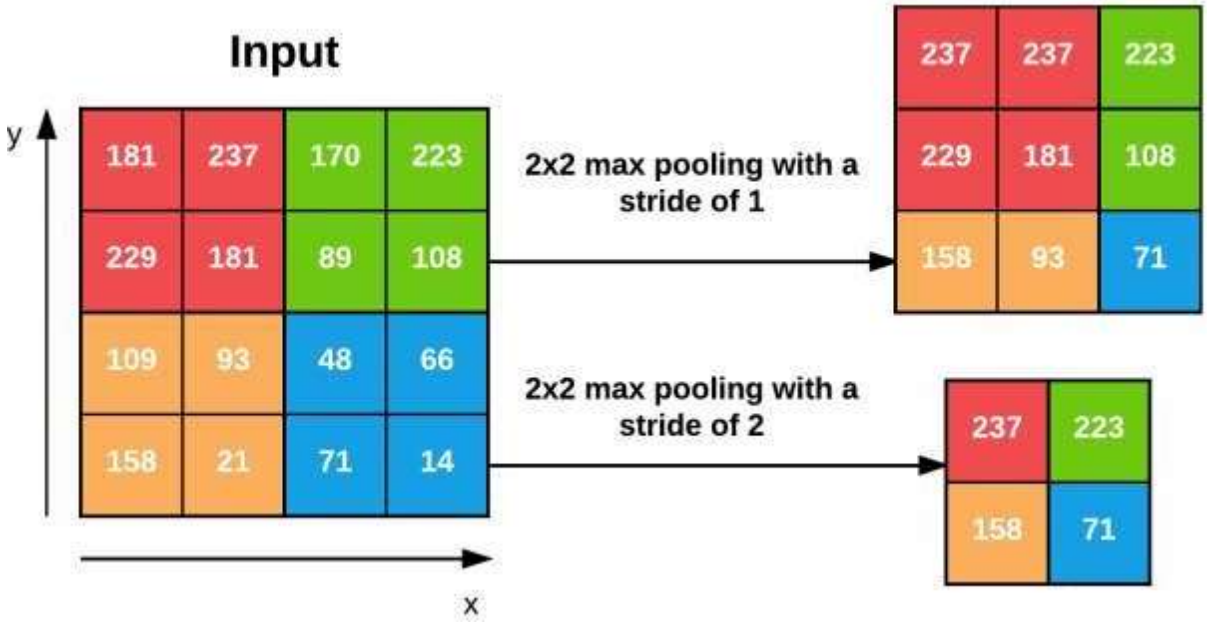
Applying the POOL operation yields an output volume of sizeWout put _Hout put _Dout put , where:

- Wout put = ((Winput F)=S)+1

- Hout put = ((Hinput F)=S)+1
- Dout put = Dinput

In practice, we tend to see two types of max pooling variations:

_ Type #1: F = 3;S = 2 which is called overlapping pooling and normally applied to images/ input volumes with large spatial dimensions.



S = 1. Bottom: Applying 2_2 max pooling with S = 2 – this dramatically reduces the spatial dimensions of our input.

Fully-connected Layers

Neurons in FC layers are fully-connected to all activations in the previous layer, as is the standard for feedforward neural networks that we've been discussing in Chapter 10. FC layers are always placed at the end of the network (i.e., we don't apply a CONV layer, then an FC layer, followed by another CONV) layer. It's common to use one or two FC layers prior to applying the softmax classifier, as the following (simplified) architecture demonstrates:

INPUT => CONV => RELU => POOL => CONV => RELU => POOL => FC => FC

Here we apply two fully-connected layers before our (implied) softmax classifier which will compute our final output probabilities for each class.

Batch Normalization

First introduced by Ioffe and Szgedy in their 2015 paper, Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift [123], batch normalization layers (or BN for short), as the name suggests, are used to normalize the activations of a given input volume before passing it into the next layer in the network. If we consider x to be our mini-batch of activations, then we can compute the normalized ˆ x via the following equation

We set e equal to a small positive value such as 1e-7 to avoid taking the square root of zero. Applying this equation implies that the activations leaving a batch normalization layer will have approximately zero mean and unit variance (i.e., zero-centered).

At testing time, we replace the mini-batch mb and sb with running averages of mb and sb computed during the training process. This ensures that we can pass images through our network and still obtain accurate predictions without being biased by the mb and sb from the final mini-batch passed through the network at training time. Batch normalization has been shown to be extremely effective at reducing the number of epochs it takes to train a neural network. Batch normalization also has the added benefit of helping "stabilize" training, allowing for a larger variety of learning rates and regularization strengths. Using batch normalization doesn't alleviate the need to tune these parameters of course, but it will make your life easier by making learning rate and regularization less volatile and more straightforward to tune. You'll also tend to notice lower final loss and a more stable loss curve when using batch normalization in your networks.

The biggest drawback of batch normalization is that it can actually slow down the wall time it takes to train your network (even though you'll need fewer epochs to obtain reasonable accuracy) by 2-3x due to the computation of per-batch statistics and normalization.

That said, I recommend using batch normalization in nearly every situation as it does make a significant difference. As we'll see later in this book, applying batch normalization to our network architectures can help us prevent overfitting and allows us to obtain significantly higher classification accuracy in fewer epochs compared to the same network architecture without batch normalization.

Dropout

The last layer type we are going to discuss is dropout. Dropout is actually a form of regularization that aims to help prevent overfitting by increasing testing accuracy, perhaps at the expense of training accuracy. For each mini-batch in our training set, dropout layers, with probability p, randomly disconnect inputs from the preceding layer to the next layer in the network architecture. Figure 11.11 visualizes this concept where we randomly disconnect with probability p=0:5 the connections between two FC layers for a given mini-batch. Again, notice how half of the connections are severed for this mini-batch. After the forward and backward pass are computed for the minibatch, we re-connect the dropped connections, and then sample another set of connections to drop.

The reason we apply dropout is to reduce overfitting by explicitly altering the network architecture at training time. Randomly dropping connections ensures that no single node in the network is responsible for "activating" when presented with a given pattern. Instead, dropout ensures there are multiple, redundant nodes that will activate when presented with similar inputs – this in turn helps our model to generalize.

It is most common to place dropout layers with p = 0:5 in-between FC layers of an architecture where the final FC layer is assumed to be our softmax classifier:

Serializing a Model to Disk

Using the Keras library, model serialization is as simple as calling **model.save** on a trained model

and then loading it via the **load_model** function.

The LeNet Architecture

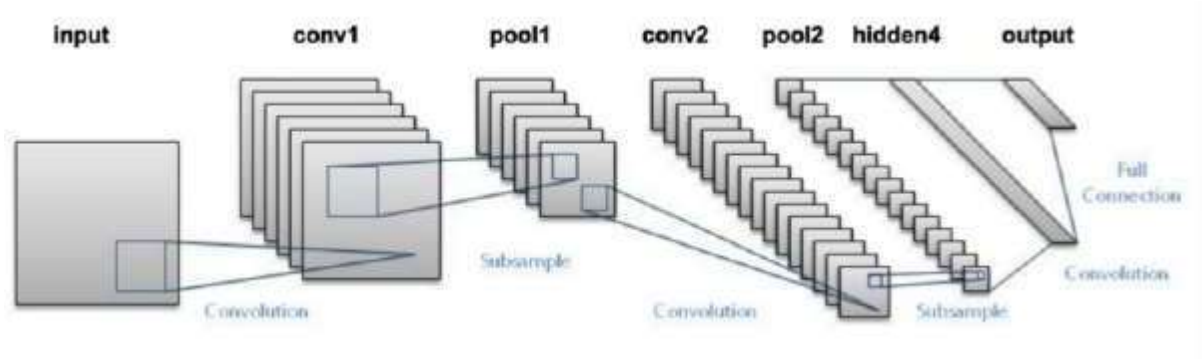


Figure 6.5: The LeNet architecture consists of two series of CONV => TANH => POOL layer sets

followed by a fully-connected layer and softmax output.

6.3 TRAINING MECHANISM

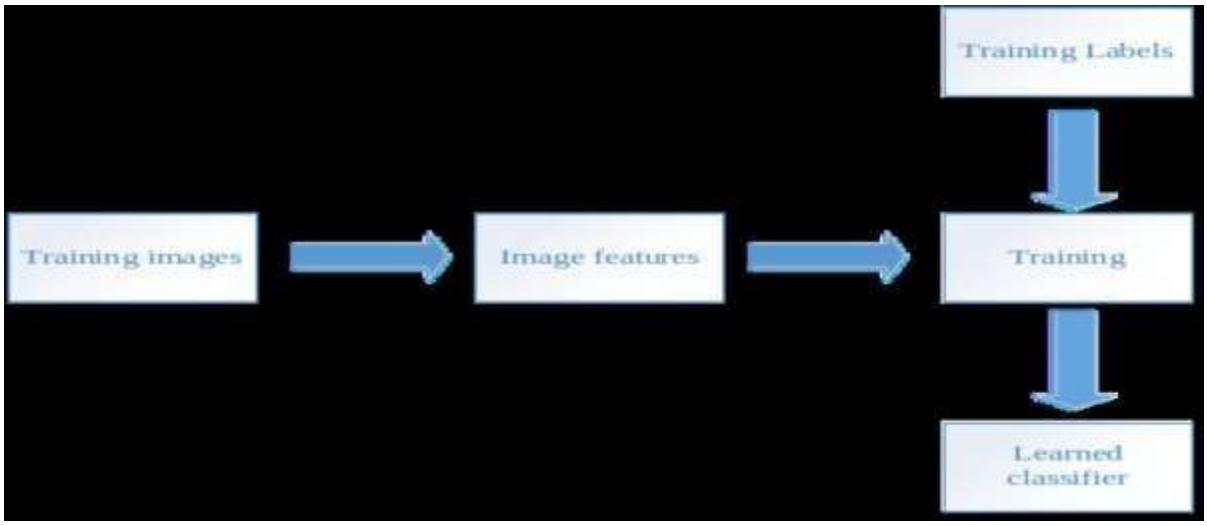


Fig 6.6: Workflow diagram for the training

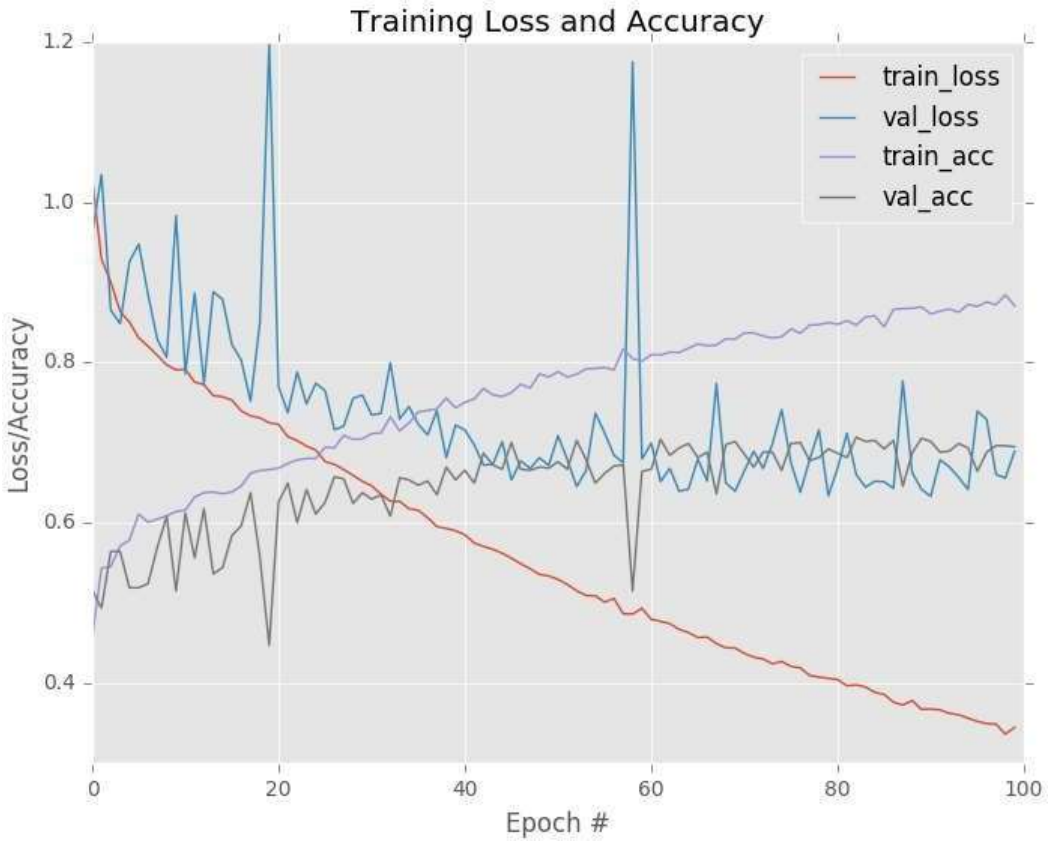
Before we train our model, let's build a basic Fully Connected Neural Network for the dataset. The basic steps to build an image classification model using a neural network are:

- Flatten the input image dimensions to 1D (width pixels x height pixels)
- Normalize the image pixel values (divide by 255)
- One-Hot Encode the categorical column
- Build a model architecture (Sequential) with Dense layers
- Train the model and make predictions

To get real accuracy in our model, it is necessary to

- Line up the feature of the image.
- Multiply each image pixel by corresponding feature pixel.
- Add the values and find the sum.
- Divide the sum by the total number of pixels in the feature.

6.4 TRAINING ACCURACY



One of the main problems of image processing is their need for large amounts of training data, as commented in improving disease detection results. As mentioned there, our project will be able to recognize a specific region of interest only if a big number of images related to it are also included. Until now, this process has been carried out in a manual way that was clearly inefficient and time-consuming.

To overcome this we have trained it with large number of dataset which will help overcome accuracy and precision problem during training of our model.

CHAPTER 7

PROJECT OUTCOMES

- This project presents a survey on different diseases classification techniques used for plant leaf disease detection and an algorithm for image segmentation technique that can be used for automatic detection as well as classification of plant leaf diseases later.
- Related diseases for the plants were taken for identification. With very less computational efforts the optimum results were obtained, which also shows the efficiency of proposed algorithm in recognition and classification of the leaf diseases. Another advantage of using this method is that the plant diseases can be identified at early stage or the initial stage.
- To improve recognition rate in classification process Convolution Neural Network can be used.

CHAPTER 8

APPLICATIONS OF PROPOSED SYSTEM

- It helps in quick detection of disease.
- Computer vision and machine-learning solutions offer great opportunities for the automatic recognition of sick plants by visual inspection of damaged leaves.
- A plant disease recognition system can work as a universal detector, recognizing general abnormalities on the leaves, such as scorching or mold.
- The main advantages of our solution include high processing speed and high classification accuracy.
- Bio Farm
- Bio Pesticides

CHAPTER 9

SAMPLE CODE

CODE:

- **Main.py**

```
import tkinter as tk
from PIL import Image, ImageTk
from tkinter.filedialog import askopenfilename
from keras.preprocessing import image
from keras.models import load_model
import numpy as np
import cv2
import shutil
import time
import imutils


li = ['Apple___Apple_scab\n\n\nPesticides:\nBonide Sulfur Plant
Fungicide\nrganocide\nBonide Orchard Spray', 'Apple___Black_rot\n\n\nPesticides:\nPhysan
20\nNEEM oil\nCopper sprays\nOrchard Spray',
'Apple___Cedar_apple_rust\n\n\nPesticides:\nSERENADE Garden\nOrchard Spray\nSulfur
Plant Fungicide',
        'Apple___healthy', 'Blueberry___healthy',
'Cherry___Powdery_mildew\n\n\nPesticides:\norganic compost\nNeem oil and PM
Wash\nZero Tolerance Fungicide',
        'Cherry___healthy', 'Corn_(maize)___Cercospora_leaf_spot\n\n\nPesticides:\nsulfur
sprays\ncopper-based fungicides\nGarden Dust',
        'Corn_(maize)___Common_rust\n\n\nPesticides:\nSERENADE Garden\nOrchard
Spray\nSulfur Plant Fungicide', 'Corn_(maize)___Northern_Leaf_Blight\n\n\nPesticides:\n',
'Corn_(maize)___healthy',
        'Grape___Black_rot\n\n\nPesticides:\nPhysan 20\nNEEM oil\nCopper
sprays\nOrchard Spray', 'Grape___Esca_(Black_Measles)\n\n\nPesticides:\n',
'Grape___Leaf_blight\n\n\nPesticides:\n',
        'Grape___healthy', 'Orange___Haunglongbing\n\n\nPesticides:\n',
'Peach___Bacterial_spot\n\n\nPesticides:\n',
```

'Peach___healthy', 'Pepper,_bell___Bacterial_spot\n\n\nPesticides:\n', 'Pepper,_bell___healthy', 'Potato___Early_blight\n\n\nPesticides:\nFontelis\nEndura, Lance WDG\nCabrio\nReason',

'Potato___Late_blight\n\n\nPesticides:\ncopper based fungicide\nOrganocide\nLiquid Copper', 'Potato___healthy', 'Raspberry___healthy', 'Soybean___healthy',

'Squash___Powdery_mildew\n\n\nPesticides:\norganic compost\nNeem oil and PM Wash\nZero Tolerance Fungicide', 'Strawberry___Leaf_scorch\n\n\nPesticides:\n', 'Strawberry___healthy', 'Tomato___Bacterial_spot\n\n\nPesticides:',

'Tomato___Early_blight\n\n\nPesticides:\nFontelis\nEndura, Lance WDG\nCabrio\nReason', 'Tomato___Late_blight\n\n\nPesticides:\ncopper based fungicide\nOrganocide\nLiquid Copper', 'Tomato___Leaf_Mold\n\n\nPesticides:', 'Tomato___Septoria_leaf_spot\n\n\nPesticides:',

'Tomato___Spider_mites\n\n\nPesticides:\n', 'Tomato___Target_Spot\n\n\nPesticides:\n', 'Tomato___Tomato_Yellow_Leaf_Curl_Virus\n\n\nPesticides:\nsulfur or copper-based fungicides\nGarden Dust\norganic fertilizers high in nitrogen\nLiquid Copper',

'Tomato___Tomato_mosaic_virus\n\n\nPesticides:\nSafer Soap, Bon-Neem\nHarvest-Guard row cover\nleast-toxic herbicides\nAllDown', 'Tomato___healthy']

```
classifier = load_model('PLANT_MODEL.hdf5')

diseasename = None

root = tk.Tk()
root.title("Plant_Leaf")

root.geometry("800x600")
root.configure(background ="white")

title = tk.Label(text="Select An Image To Process", background = "white", fg="Brown", font=("", 15))
title.grid(row=0, column=2, padx=10, pady = 10)


def exit():
```

```
        root.destroy()

    def clear():
        cv2.destroyAllWindows()
        disease = tk.Label(text='                                                    \n\n\n\n\n',
background="white",
                           fg="Black", font=("", 20))
        disease.grid(column=3, row=3, padx=10, pady=10)

    def cap_process():
        maxid = 0
        image_path = path
        img = cv2.imread(image_path)
        imputimg = imutils.resize(img, width=300, height=300)
        cv2.imshow('original',imputimg)
        img = imutils.resize(img, width=120, height=120)
        original = img.copy()
        neworiginal = img.copy()

        #Calculating number of pixels with shade of white(p) to check if exclusion of these
pixels is required or not (if more than a fixed %) in order to differentiate the white background
or white patches in image caused by flash, if present.
        p = 0
        for i in range(img.shape[0]):
            for j in range(img.shape[1]):
                B = img[i][j][0]
                G = img[i][j][1]
                R = img[i][j][2]
                if (B > 110 and G > 110 and R > 110):
                    p += 1

        #finding the % of pixels in shade of white
        totalpixels = img.shape[0]*img.shape[1]
        per_white = 100 * p/totalpixels

        #excluding all the pixels with colour close to white if they are more than 10% in
the image
```

```
            if per_white > 10:
                img[i][j] = [200,200,200]
                #cv2.imshow('color change', img)



            #Guassian blur
            blur1 = cv2.GaussianBlur(img,(3,3),1)



            #mean-shift algo
            newimg = np.zeros((img.shape[0], img.shape[1],3),np.uint8)
            criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER ,
10 ,1.0)

            img = cv2.pyrMeanShiftFiltering(blur1, 20, 30, newimg, 0, criteria)
            #cv2.imshow('means shift image',img)



            #Guassian blur
            blur = cv2.GaussianBlur(img,(11,11),1)

            #Canny-edge detection
            canny = cv2.Canny(blur, 160, 290)

            canny = cv2.cvtColor(canny,cv2.COLOR_GRAY2BGR)

            #contour to find leafs
            bordered = cv2.cvtColor(canny,cv2.COLOR_BGR2GRAY)
            contours,hierarchy = cv2.findContours(bordered, cv2.RETR_TREE,
cv2.CHAIN_APPROX_NONE)

            maxC = 0
            for x in range(len(contours)):
                                        #if take max or one less than max then will not
work in
                if len(contours[x]) > maxC:
                                            # pictures with zoomed leaf images
```

```
            maxC = len(contours[x])
            maxid = x

perimeter = cv2.arcLength(contours[maxid],True)
#print perimeter
Tarea = cv2.contourArea(contours[maxid])
cv2.drawContours(neworiginal,contours[maxid],-1,(0,0,255))
#cv2.imshow('Contour',neworiginal)
#cv2.imwrite('Contour complete leaf.jpg',neworiginal)



#Creating rectangular roi around contour
height, width, _ = canny.shape
min_x, min_y = width, height
max_x = max_y = 0
frame = canny.copy()

# computes the bounding box for the contour, and draws it on the frame,
for contour, hier in zip(contours, hierarchy):
    (x,y,w,h) = cv2.boundingRect(contours[maxid])
    min_x, max_x = min(x, min_x), max(x+w, max_x)
    min_y, max_y = min(y, min_y), max(y+h, max_y)
    if w > 80 and h > 80:
        roi = img[y:y+h , x:x+w]
        originalroi = original[y:y+h , x:x+w]

if (max_x - min_x > 0 and max_y - min_y > 0):
    roi = img[min_y:max_y , min_x:max_x]
    originalroi = original[min_y:max_y , min_x:max_x]

#cv2.imshow('ROI', frame)
#cv2.imshow('rectangle ROI', roi)
img = roi



#Changing colour-space
```

```
        #imghsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
        imghls = cv2.cvtColor(roi, cv2.COLOR_BGR2HLS)
        #cv2.imshow('HLS', imghls)
        imghls[np.where((imghls==[30,200,2]).all(axis=2))] = [0,200,0]
        #cv2.imshow('new HLS', imghls)

        #Only hue channel
        huehls = imghls[:,:,0]
        #cv2.imshow('img_hue hls',huehls)
        #ret, huehls = cv2.threshold(huehls,2,255,cv2.THRESH_BINARY)

        huehls[np.where(huehls==[0])] = [35]
        #cv2.imshow('img_hue with my mask',huehls)



        #Thresholding on hue image
        ret, thresh = cv2.threshold(huehls,28,255,cv2.THRESH_BINARY_INV)
        #cv2.imshow('thresh', thresh)



        #Masking thresholded image from original image
        mask = cv2.bitwise_and(originalroi,originalroi,mask = thresh)
        masked = imutils.resize(mask, width=300, height=300)
        cv2.imshow('masked out img',masked)



        #Finding contours for all infected regions
        contours,heirarchy = cv2.findContours(thresh, cv2.RETR_TREE,
cv2.CHAIN_APPROX_NONE)

        Infarea = 0
        for x in range(len(contours)):
            cv2.drawContours(originalroi,contours[x],-1,(0,0,255))
            #cv2.imshow('Contour masked',originalroi)

            #Calculating area of infected region
            Infarea += cv2.contourArea(contours[x])
```

```
        if Infarea > Tarea:
            Tarea = img.shape[0]*img.shape[1]

        #Finding the percentage of infection in the leaf



        try:
            per = 100 * Infarea/Tarea
        except ZeroDivisionError:
            per = 0

        per = round(per,2)

        print (f'Infected area:{Infarea}')
        print (f'Percentage of infection region:{per}')

        fiimg = imutils.resize(original, width=300, height=300)
        cv2.imshow('final',fiimg)
        disease = tk.Label(text='Infected area:' + str(per), background="white",
                    fg="Black", font=("", 15))
        disease.grid(column=3, row=3, padx=10, pady=10)
        button4 = tk.Button(text="clear", command=clear)
        button4.grid(row=6, column=2, padx=10, pady = 10)
        button5 = tk.Button(text="Exit", command=exit)
        button5.grid(row=7, column=2, padx=10, pady = 10)

    def analysis():
        image_path = path
        new_img = image.load_img(image_path, target_size=(224, 224))
        img = image.img_to_array(new_img)
        img = np.expand_dims(img, axis=0)
        img = img/255

        print("Following is our prediction:")
        prediction = classifier.predict(img)
        # decode the results into a list of tuples (class, description, probability)
```

```
    # (one such list for each sample in the batch)
    d = prediction.flatten()
    j = d.max()
    for index,item in enumerate(d):
       if item == j:
          class_name = li[index]
    print(class_name)
    diseasename = class_name
    disease = tk.Label(text='Status: ' + diseasename, background="white",
                       fg="Black", font=("", 15))
    disease.grid(column=3, row=3, padx=10, pady=10)
    button3 = tk.Button(text="Clear", command=clear)
    button3.grid(row=6, column=2, padx=10, pady = 10)
    button4 = tk.Button(text="Exit", command=exit)
    button4.grid(row=7, column=2, padx=10, pady = 10)

def openphoto():
    global path
    path=askopenfilename(filetypes=[("Image File",")])
    im = Image.open(path)
    tkimage = ImageTk.PhotoImage(im)
    myvar=tk.Label(root,image = tkimage, height="224", width="224")
    myvar.image = tkimage
    myvar.place(x=1, y=0)
    myvar.grid(row=3, column=2 , padx=10, pady = 10)
    button2 = tk.Button(text="Analyse Image", command=analysis)
    button2.grid(row=4, column=2, padx=10, pady = 10)
    button7 = tk.Button(text="Process Image", command=cap_process)
    button7.grid(row=5, column=2, padx=10, pady = 10)

def capture():
    global path
    cam = cv2.VideoCapture(0)
    time.sleep(0.5)
    ret, img = cam.read()
    captured = cv2.imwrite("./Captured_images/Captured.jpg", img)
    cam.release()
```

```
        path = "./Captured_images/Captured.jpg"
        frame = cv2.imread(path)
        cv2image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGBA)
        cv2image = imutils.resize(cv2image, width=250)
        img = Image.fromarray(cv2image)
        tkimage = ImageTk.PhotoImage(img)
        myvar=tk.Label(root,image = tkimage, height="224", width="224")
        myvar.image = tkimage
        myvar.place(x=1, y=0)
        myvar.grid(row=3, column=2 , padx=10, pady = 10)
        button2 = tk.Button(text="Analyse Image",command=analysis)
        button2.grid(row=4, column=2, padx=10, pady = 10)
        button7 = tk.Button(text="Process Image",command=cap_process)
        button7.grid(row=5, column=2, padx=10, pady = 10)

    button1 = tk.Button(text="Select Image", command = openphoto)
    button1.grid(row=1, column=2, padx=10, pady = 10)

    capbut = tk.Button(text="Capture", command = capture)
    capbut.grid(row=2, column=2, padx=10, pady = 10)

    root.mainloop()
```

CHAPTER 10

TESTING

Software and hardware testing is a critical element of the ultimate review of specification design and coding. Testing of software and hardware leads to the uncovering of errors in the software functional and performance requirements are met. Testing also provides a good indication of software- hardware reliability and quality as a whole to function well in any critical missions and projects. The result of different phases of testing are evaluated and then compared with the expected results. If the errors are uncovered they are debugged and corrected and if any hardware fault then the hardware can be repaired or replaced with the brand new one with a finer quality to work for longer time duration. A strategy approach to software testing has the generic characteristics:

10.1.1 Different testing techniques are appropriate at different points of time.

10.1.2 Testing and debugging are different activities, but debugging must be accommodated in the testing strategy.

Some hardware products are tested very rigorously or completely because they are critical to the operation of the environment they're going into or the process variation on the manufacturing process is so tight that it's necessary to test every part.

For the types of test systems we develop at Viewpoint, this often involves:

- Stimulating the UUT (electrically and/or mechanically)
- Sequencing through various test steps
- Taking various physical property measurements (e.g., vibration, temperature, pressure, load, voltage, current, etc)
- Analyzing the acquired data to calculate pass/fail for various tests
- And logging, report generation, and/or displaying the results on a UI.
- Also check if the wire ports are proper and the wired connections are working well when the whole circuit is built up.

10.1 VALIDATION TESTING

Validation Testing ensures that the auto aiming machine actually meets the valid client's needs. It can also be defined as to demonstrate that the system fulfills its intended use when deployed on appropriate environment for authorized and legal usage.

Validation Testing Variations include:-

- **Unit testing –** Checks if the sensor, buzzer, display and trigger and other basic components works properly in the system.
- **Integration testing –** Checks if the all the components like sensor, buzzer, display, trigger works together properly when used with other components integrated as a whole.
- **System Testing –** Checks if the weapon system works perfectly when integrated with the module. Manual or automated user testing comes into play here, where it is checked whether the whole system is working well when the authorized user is handling it.
- **User Acceptance Testing –** Finally, customer representing end users test the feature from their perspective and report if any anomaly is found.

Slno.	Test case Description	Expected result	Actual Result	Remarks
1.	Check if software is installed properly in the laptop to run the Project	All the software modules installed with compatible dependencies to work well without showing any error during the download of software.	All the software modules installed with compatible dependencies to work well without showing any error during the download of software.	Pass
2.	User Input: Capture an image Through WebCam	Intent: Infected Area Response from Project: Infected Area: 0.0	Intent: Infected Area Response from Project: Infected Area: 0.0	Pass

3.	User Input: Leaf Image From Dataset	Status: Potato Blight Pesticide : Orchard Spray	Status: Potato Blight Pesticide : Orchard Spray	Pass
4.	User Input: Leaf Image From Dataset	Status: Corn Maize common rust Pesticide : Orchard Spray	Status: Corn Maize common rust Pesticide : Orchard Spray	PASS

Table 10.1.1 Test case for Validation

CHAPTER 11

SCREENSHOTS

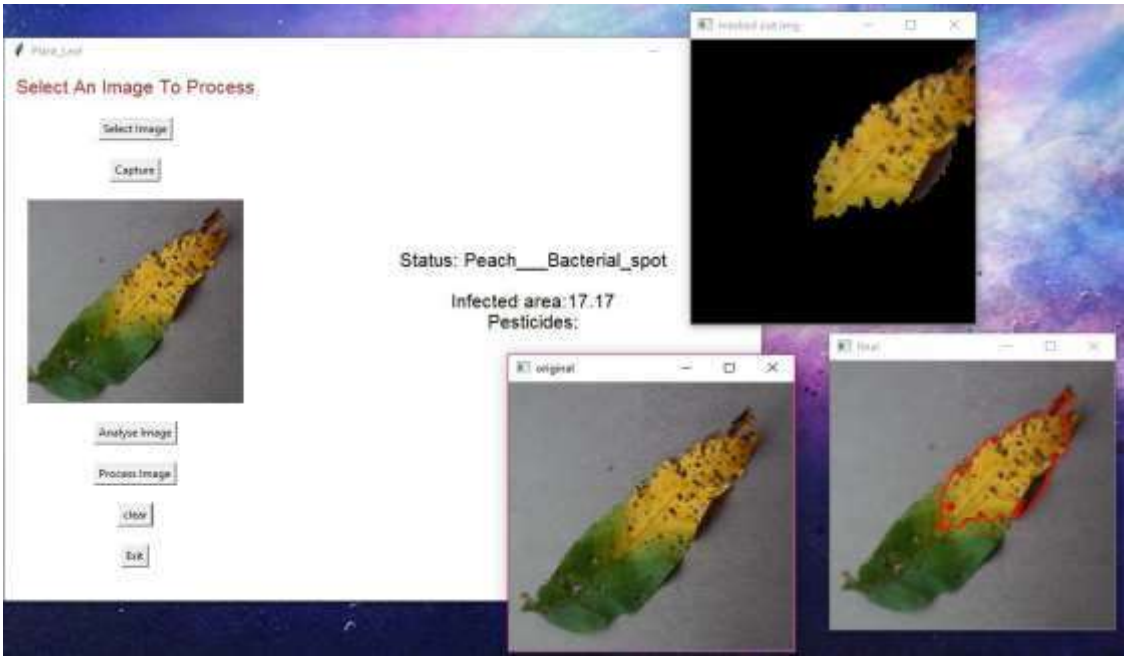


Fig 11.1: Peach Bacterial Spot

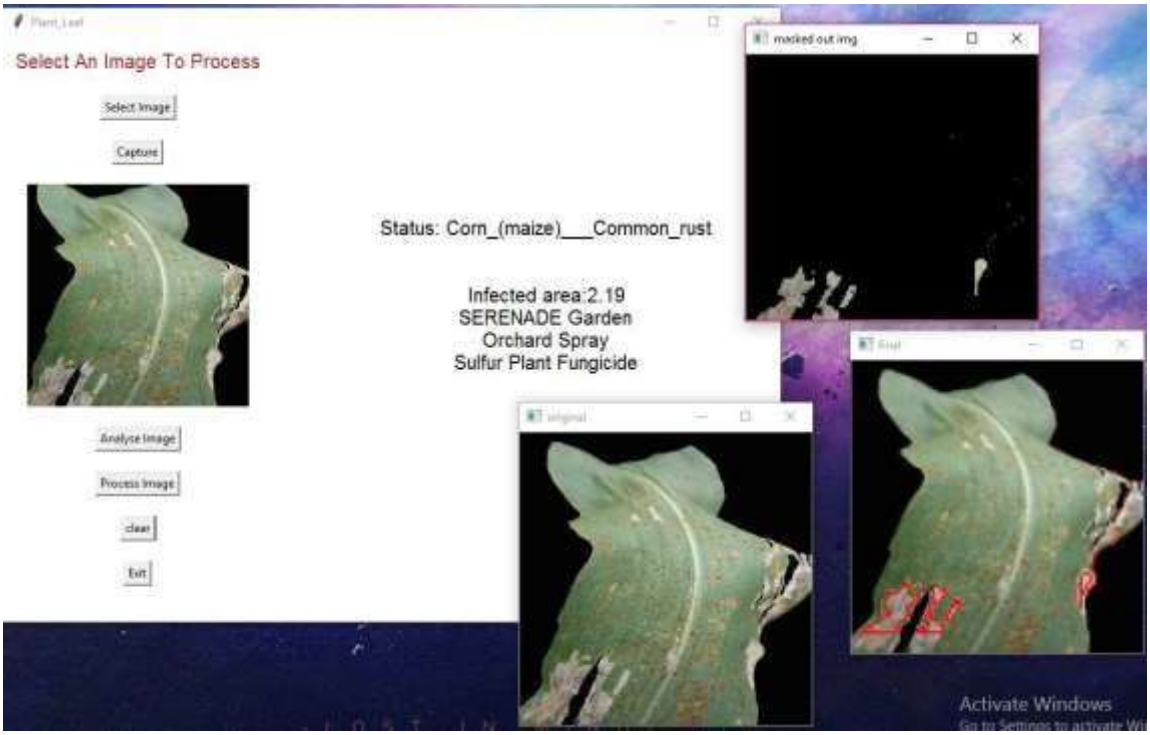


Fig 11.2: Corn Common Rust

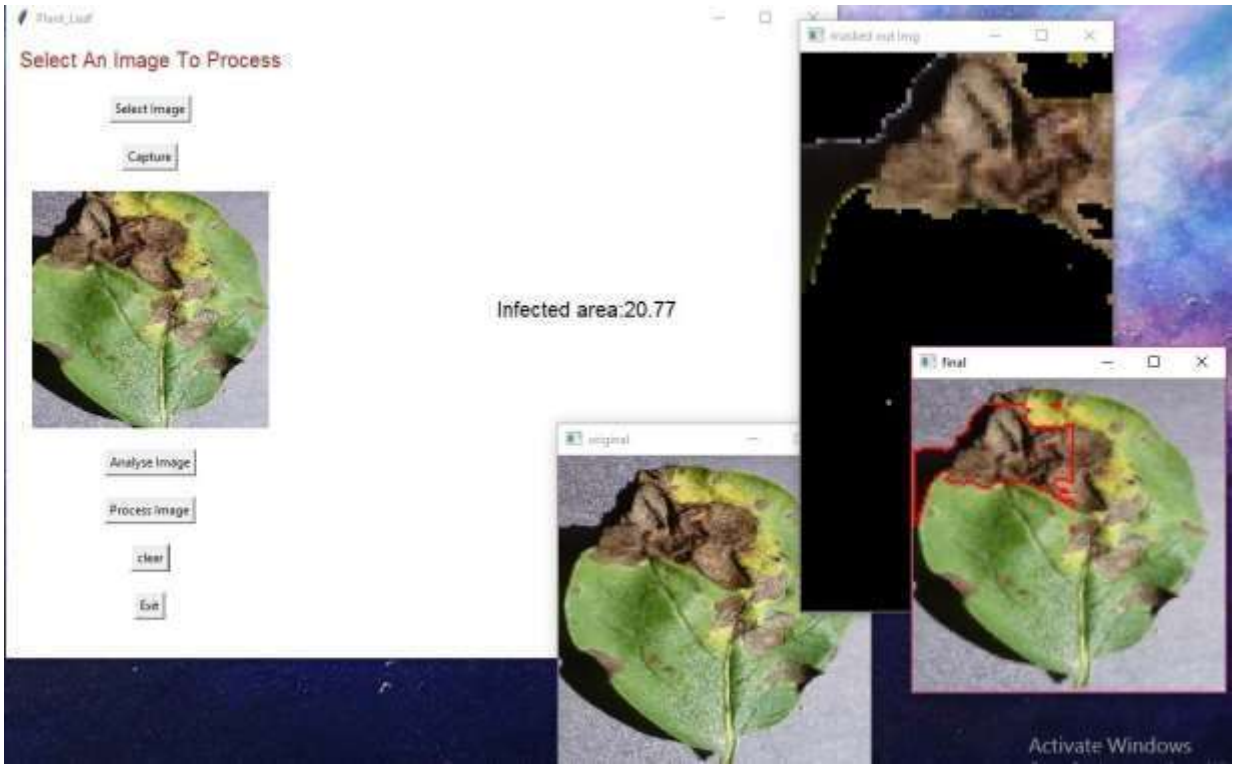


Fig 11.3: Potato Early Blight

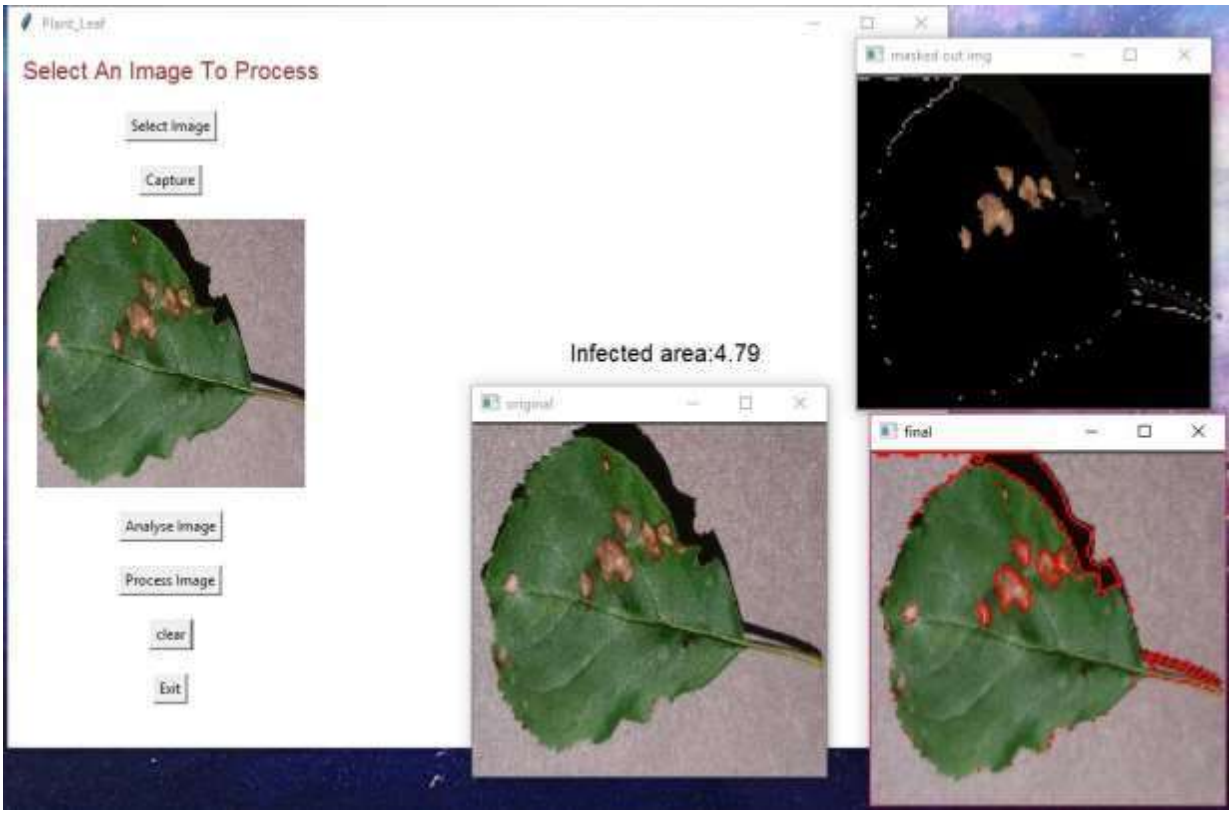


Fig 11.4: Apple Black Rot

Fig 11.5: Potato Early Blight

Fig 11.6: Leaves captured using web camera

FUTURE ENHANCEMENT

The future plan of this project is to improve its design, add more effective features and make the documentation more flexible so that anybody can use the software without facing any issues.

Following few future enhancements can be implemented on this project to make the system more accurate.

- Removal of shadow from the image
- Improvement of accuracy rate
- Implementing Advanced self-training features
- Database updating features without human intervention in the system
- Adding a vide variety of leaf types for detection

CONCLUSION

- The main aim of this approach is to recognise the leaf diseases. Speed and accuracy are the main characteristics which are taken into consideration.
- The results can recognise the leaf diseases with little computational efforts as Keras and tensorflow are used which reduces the number of steps required to run simple and common use cases.
- Another advantage of using this method is that the plant diseases can be identified at early stage or the initial stage.

REFERENCES

[1] Melike Sardogan , Adem Tuncer ,Yunus Ozen “Plant Leaf Disease Detection and Classification Based on CNN with LVQ Algorithm”, IEEE 2018 3rd International Conference on Computer Science and Engineering (UBMK)

[2] L. Sherly Puspha Annabel, T. Annapoorani, P. Deepalakshmi “Machine Learning for Plant Leaf Disease Detection and Classification – A Review”, IEEE 2019 International Conference on Communication and Signal Processing (ICCSP)

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

[4] Amrita S. Tulshan, Nataasha Raul “Plant Leaf Disease Detection using Machine Learning”, IEEE 2019 10th International Conference on Computing, Communication and Networking Technologies (ICCCNT)

[5] Bharat Mishra, Sumit Nema ,Mamta Lambert, Swapnil Nema “Recent technologies of leaf disease detection using image processing approach” IEEE 2017 International Conference on Innovations in Information, Embedded and Communication Systems (ICIIECS)

[6] Gurleen Kaur Sandhu, Rajbir Kaur “Plant Disease Detection Techniques” IEEE 2019 International Conference on Automation, Computational and Technology Management (ICACTM)

[7] V Pooja, Rahul Das, V Kanchana “Identification of plant leaf diseases using image processing techniques” IEEE 2017 Technological Innovations in ICT for Agriculture and Rural Development (TIAR)

[8] Jaskaran Singh, Harpreet Kaur “ Various techniques of plant leaf disease detection” IEEE 2018 2nd International Conference on Inventive Systems and Control (ICISC)

[9] Shivani K. Tichkule, Dhanashri. H. Gawali “Plant diseases detection using image processing techniques” IEEE 2017 Online International Conference on Green Engineering and Technologies (IC-GET)

[10] Sharath D.M., Akhilesh, S. Arun Kumar, Rohan M.G., Prathap C. “Image based Plant Disease Detection in Pomegranate Plant for Bacterial Blight” IEEE 2019 International Conference on Communication and Signal Processing (ICCSP)

[11] Shima Ramesh, Ramachandra Hebbar, Niveditha M., Pooja R., Prasad Bhat N., Shashank N., Vinod P.V. “Plant Disease Detection Using Machine Learning” IEEE 2018 International Conference on Design Innovations for 3Cs Compute Communicate Control (ICDI3C)

[12] Jyoti Shirahatti, Rutuja Patil, Pooja Akulwar “A Survey Paper on Plant Disease Identification Using Machine Learning Approach” IEEE 2018 3rd International Conference on Communication and Electronics Systems (ICCES)

QS I·GAUGE INDIAN COLLEGE RATING GOLD

Accredited by NBA*
SVCE BENGALURU
SRI VENKATESHWARA COLLEGE OF ENGINEERING
Affiliated to VTU, Approved by AICTE, Recognised by UGC u/s 2(f) & 12(B)

INSTITUTION'S INNOVATION COUNCIL (Ministry of HRD Initiative)

E-LEAD

SRI VENKATESHWARA COLLEGE OF ENGINEERING

Vidyanagar, Kempegowda International Airport Road, Bengaluru - 562 157

SVCE IQAC Internal Quality Assurance Cell

CERTIFICATE OF PARTICIPATION

AN ONLINE TECHNICAL FEST

UDYUKTA – 2020

This is to Certify that

Mr./Ms. Varshini Rao V,Uwaiz Ali Khan,Vaishnavi C R,Ruchi .....

Department of Computer Science and engineering has actively participated in the Technical/~~Non-Technical~~ Event titled Paper Presentation held on 20th & 21st July 2020.

Dr. S C Lingareddy	Dr. Shoba M	Dr. H N Rajakumara	Dr. Suresha	Dr. Shashidhar Muniyappa
HoD-CSE	HoD-ISE	HoD-CE	Principal	CED
SVCE	SVCE	SVCE	SVCE	SVGI