

A
Major Project Report on
“PLANT DISEASE DETECTION USING CNN”

In partial fulfillment of requirements for the degree of
Bachelor of Technology (B. Tech.)

in
Computer Science and Engineering



Submitted by
Ms. Riya Sinha (170385)

Under the Guidance of
Dr. Anand Sharma

SCHOOL OF ENGINEERING AND TECHNOLOGY
Mody University of Science and Technology
Lakshmangarh, Distt. Sikar-332311

June 2021

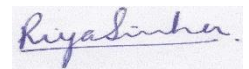
ACKNOWLEDGEMENT

I have taken efforts in this Project. However, it would not have been possible without the kind support and help of many of the individuals. I would like to extend my sincere gratitude to all of them.

I'm highly obliged to A Senthil , Assistant Dean SET and HOD (SET deptt.), Mody University of science and technology, for the amenities provided to accomplish this project.

I am indebted to Dr. Anand Sharma for his guidance and constant supervision as well as providing necessary information regarding the project topic.

I would also like to express my gratefulness towards my parents and members of Mody University for their kind cooperation and encouragement which helped me in completion of this major project. My thanks and appreciation goes to all those people who have keenly helped me with their abilities.



Riya Sinha (170385)

CERTIFICATE

This is to certify that the major project report entitled “Plant disease detection ” submitted by Ms. Riya Sinha, as a partial fulfillment for the requirement of B. Tech. VIII Semester examination of the School of Engineering and Technology, Mody University of Science and Technology, Lakshmangarh for the academic session 2020-2021 is an original project work carried out under the supervision and guidance of Dr. Anand Sharma has undergone the requisite duration as prescribed by the institution for the project work.

PROJECT GUIDE:

Approval Code:

Name: Dr. Anand Sharma

Date: 15th June 2021

HEAD OF DEPARTMENT

Signature:

Name: Dr. A. Senthil

Date: 15th June 2021

EXAMINER-I:

Name: Ms. Sonal Sharma

Dept: SET

EXAMINER-II

Name: Dr. Vikas Raina

Dept: SET

ABSTRACT

When plants & crops are affected by pests it affects the country's agricultural products. Farmers or experts often look at plants with the naked eye to detect and diagnose diseases. But this approach can be time-consuming, expensive, and inconvenient. Automatic detection using image processing techniques provides fast and accurate results. This paper is concerned with a new approach to the development of a model for the diagnosis of plant diseases, based on the separation of leaf images, through the use of deep networks. Advances in computer vision provide an opportunity to expand and enhance the practice of accurate crop protection and expand the market for computer vision applications in the precise agricultural sector. The novel training method and the method used to facilitate the implementation of a quick and easy to implement program. All the important steps required to use this diagnostic model are fully explained throughout the paper, from image collection to database, evaluated by agricultural experts, in-depth learning framework for conducting in-depth CNN training.

I use the Convolution Neural Network (CNN) which contains the various layers used to predict. The drone model is also designed that can be used for live coverage of large agricultural fields where a high-resolution camera is attached and will capture images of plants that will serve as software inputs, on which the software is based, whether the plant is healthy or not. Through This project I'll carry out these tasks:

- 1) Design such a system that can accurately detect plant and pest diseases.
- 2) Create a database of appropriate pesticides and appropriate diseases.
- 3) Provide a cure for the acquired disease.

Table of Contents

Sr.no.	Topics	Page no.
1.	Introduction	2
1.1	Present System	3
1.2	Proposed System	3-4
2.	System Design	
2.1	System flowchart	5-9
3.	Hardware and Details/ software Standards	
3.1	Hardware requirement	10
3.2	Software requirement Technology used	10-15
3.3		15-17
4.	Implementation Work Details	
4.1	Data implementation and program execution	18-29
5.	Source Code/ Simulation Code/ Commands	30-35
6.	Input/output Screens/ Model's Photograph	36-42
7.	System Testing	43-48
8.	Individual Contribution	49-54
9.	Conclusion	
9.1	Future Scope	55
10.	Bibliography	56
11.	Plagiarism Report	57

List of Figures

Sr. no.	Topics	Page No.
1.1	System flowchart	6
2.2	System diagram	9
3.1	Opening Colab	12
3.2	Creating a Python notebook	12
3.3	Setting a notebook name	13
3.4	Entering code	13
3.5	Executing Code	13
3.6	Result of executed code	14
3.7	Clearing output by cut option	14
3.8	Adding code cell	14
3.9	Add code cell	14
3.10	Deleting a cell	15
3.11	Saving work to google drive	15
3.12	Sharing a notebook	16
4.1	Fetch images from the directory	19
4.2	Function to convert image into array	19
4.3	Get size of processed image	20
4.4	Transforming image	20
4.5	Print the classes	20
4.6	Feature scaling	20
4.7	Preprocessing the data	21
4.8	Splitting dataset	22
4.8.1	Output of splitted dataset	22
4.9	Reading final dataset	26
4.10	Training network	26
4.11	Intializing batch size and epoch	26
4.12	Model.fit generator	27
4.13	Calculating model accuracy	27
4.14	Accuracy output	27

4.15	Plotting graphs	28
4.16	Training and validation accuracy graph	28
4.17	Training and validation loss vs. epoch graph	29
4.18	Predicting result	29
4.19	Saving our model using pickle	30
4.20	Saved model	30
6.1	Importing packages	38
6.2	Function to convert image into array	38
6.3	Fetching dataset	39
6.4	Output of dataset	39
6.5	Model.fit generator	33
6.6	Print the classes	40
6.8	Output of print label binariser class	40
6.9	Splitting dataset into test and train	40
6.10	Output of above operation	40
6.11	Creation of an image generator object	41
6.12	Model creation	41
6.13	Model summary	42
6.14	Output of model summary	43
6.15	Training network	43
	Output of above operation	43
6.16		

Chapter 1: Introduction

1. INTRODUCTION

When plants & crops are affected by pests it affects the country's agricultural products. Farmers or experts often look at plants with the naked eye to detect and diagnose diseases. But this approach can be time-consuming, expensive, and inconvenient. Automatic detection using image processing techniques provides fast and accurate results. This paper is concerned with a new approach to the development of a model for the diagnosis of plant diseases, based on the separation of leaf images, through the use of deep networks. Advances in computer vision provide an opportunity to expand and enhance the practice of accurate crop protection and expand the market for computer vision applications in the precise agricultural sector. The novel training method and the method used to facilitate the implementation of a quick and easy to implement program. All the important steps required to use this diagnostic model are fully explained throughout the paper, from image collection to database, evaluated by agricultural experts, in-depth learning framework for conducting in-depth CNN training.

I use the Convolution Neural Network (CNN) which contains the various layers used to predict. The drone model is also designed that can be used for live coverage of large agricultural fields where a high-resolution camera is attached and will capture images of plants that will serve as software inputs, on which the software is based, whether the plant is healthy or not. Through our code model and training, we have achieved a 97% accuracy level.

This roadmap is a new way to diagnose plant diseases using a deep and convincing network neural network that is well-trained and well-aligned to the database of plant leaves collected independently of various plant diseases. The advancement and youth of the advanced model lie in its simplicity; healthy leaves and background images are compatible with other classes, enabling the model to distinguish between sick and healthy or environmentally friendly leaves through CNN.

1.1. PRESENT SYSTEM

The only way to diagnose plant diseases is simply to look into the eyes of a specialist where there is a diagnosis of plant diseases. In doing so, a large team of experts and continuous crop monitoring is required, which is very costly when we do on large farms. At the same time, in some lands, farmers do not have the right resources, or they may not even think about consulting experts. Because sight professionals are very expensive and time consuming. In such cases, the proposed procedure proves to be helpful in monitoring large plantations.

Physical disease identification is a very difficult task and at the same time, not very accurate and can only be done in limited areas. While the automatic detection process is using it will take less effort, less time, and more precision. In plants, some of the most common diseases seen have brown and yellow spots, early and late summer temperatures, while others are fungal, bacterial and bacterial infections. Image processing is used to measure the affected area of the disease and to determine the color difference of the affected area

1.2. PROPOSED SYSTEM

We can reduce pest infestation by using pesticides and appropriate herbs. We can reduce image size by appropriate size reduction strategies and ensure that quality will not be compromised to a large extent. We can increase the projects of the authors mentioned earlier that the cure for the disease is also indicated in the program. The key purpose is to identify plant diseases using image processing. Also, after being diagnosed with the disease, suggest the name of the pesticide to be used. It also identifies insects and insects that are responsible for the epidemic. Apart from these same purposes, this drone saves a lot of time. The model budget is very high for low-level farming targets however will be a monetary value for large-scale farming. It completes each process in turn achieves each result.

The main objectives are, therefore:

- 1) Design such a system that can accurately detect plant and pest diseases.
- 2) Create a database of appropriate pesticides and appropriate diseases.
- 3) Provide a cure for the acquired disease.

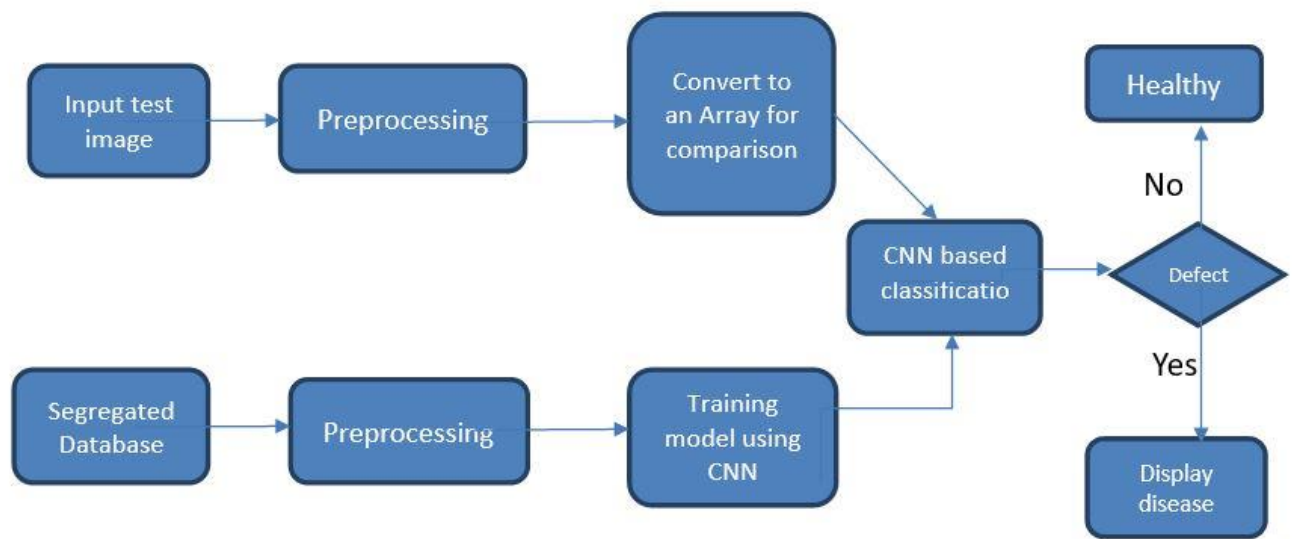


Fig 1.1 System Flow chart

Chapter 2: System Design

2.1. SYSTEM FLOWCHART

The system flow chart is shown in figure 2.2 that contains the following steps:

2.1.1. DATASET CREATION

The dataset used in this project was taken from the kaggle of the online PlantVillage database, Also the code was also written to Kaggle's online kernel for accurate calculation and to check for training and validation losses. The first step of any image-based project is finding a valid database. Most of the time a standard database is selected but, in some cases, we do not get the right database. So, it is in such cases, we may collect images and build our own database. The website is located in the AI crowd which is a challenge in the classification of plant diseases. The data found here has no label. So, the first job is to clean and label details. There is a huge database so basically, images with better resolution and angles are selected. Back selection of images, we should have in-depth knowledge about the different leaves and diseases they have. Great research performed at the final site of the plantvillage organization. Different types of plant images are studied and compatible.

After a thorough study, labeling was performed by classifying images and various diseases Diseases of different plant information:

- Pepper bell bacterial spot
- Tomatoes bacterial spot
- Potato late blight
- Potato early Blight
- Tomato plant blight



2.1.2. DATA PREPROCESSING AND LABELLING

Prefixing images includes removing low-frequency background sound, adjusting the size of individual particle images, deleting reflections, and hiding parts of the images. Image editing is a way to improve data. In addition, the image editing process involves cutting all the images by hand, forming a square around the leaves, to highlight the area of interest (plant leaves). During the data collection phase, images with minimal adjustment and size less than 500 pixels are not considered valid image data. Additionally, the only images in which the region of interest were the highest resolution were marked as eligible for the database. Thus, it was ensured that the pictures contained all the information needed to learn the features. Many resources can be found by searching across the Internet, but their suitability is often unreliable. In an effort to ensure the accuracy of the classrooms in the database, which was initially collected by keyword search, agricultural experts examined leaf images and labeled all images with appropriate disease names. As is well known, it is important to use accurately classified images for training and validation data. Only then can a suitable and reliable acquisition model be developed. At this stage, duplicate images left over after the start of compiling and collecting images by categories have been removed from the database.

2.1.3. FEATURE SCALING

Feature measurement is a method used to measure the distance of independent variables or data features. Data processing is done during the data preparation process.

- Standard Deviation is another way of measuring when the values are focused on the finish line with a standard deviation of the unit. This means that the definition of responsibility becomes zero and the distribution of the result has a standard deviation of the unit.

- Normalization is a measurement in which values are converted and redeemed to end from 0 to 1. Also known as the Min-Max Scaling.

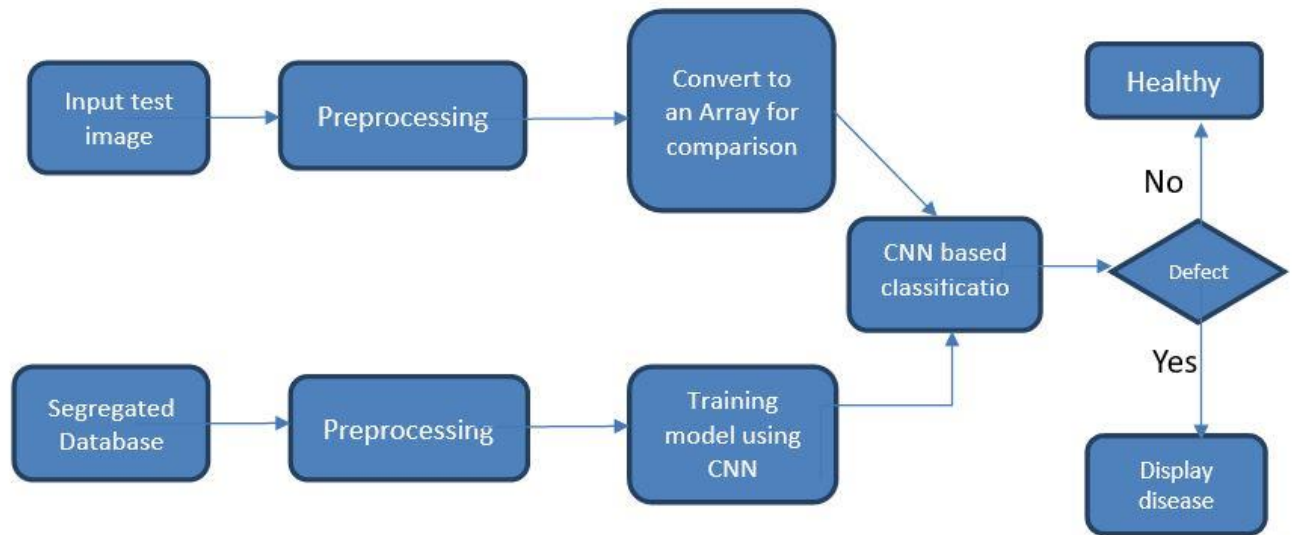


Figure 2.2 System Flowchart

2.1.4. TRAINING MODEL USING CNN

The database has been redesigned as Image reshaping, resizing, and conversion to an array form. The same analysis is also performed on the test image. Data containing approximately 32000 plant species is available, where any image can be used as a test image of the software. The training website is used for modeling training (CNN) to be able to see the diagnostic image and the disease we have. CNN has various Dense layers, Dropout, Activation, Flatten, Convolution2D, MaxPooling2D. After the model has been successfully trained, the software is able to identify the disease if the plant species are in the database. After successful training and previous practice, a comparison of the experimental image with a trained model took place to predict the disease.

2.1.5. GRAPH PLOTTING

Using matplotlib, I build a graph of Training and Validation accuracy and Training and Validation loss. By default, we can use the matplotlib library in python. The matplotlib library has many packages and functions that produce different types of graphs and sites. And it is very easy to use. It and NumPy and other built-in python functions reach the goal.

2.1.6. MODEL ACCURACY

Model accuracy is the measure used to determine which model is best at detecting relationships and patterns between data variables based on input, or training, data.

The cost of errors can be huge, but improving the accuracy of the model reduces those costs. Of course, there is a point to lower returns where the cost of making a more accurate model will not lead to a corresponding profit growth, but it is often beneficial to the rest of the board.

A better model can add ‘invisible’ data, better predictions and more insight that can produce, which also brings more business value.

2.1.7. SAVING MODEL USING PICKLE

Pickle is the standard way of serializing objects in Python. You can use pickle functionality to create learning algorithms for your machine in a different way and save the serial format to a file. You can later upload this file to disable (deserializing) your model and use it to make new predictions.

Chapter 3: Hardware and Software Details

3.1. HARDWARE REQUIREMENT

- **Operating System:** Windows 10/8.1/8/7
- **Processors:** Any Intel 64-x86 processor
- **RAM:** 4 GB is recommended

3.2. SOFTWARE REQUIREMENT

Google Colab

Colab is a free Jupyter notebook environment that runs entirely in the cloud. In particular, it doesn't require a setup and the notebooks that you make can be at the same time altered by your colleagues - simply the manner in which you alter reports in Google Docs. Colab supports many popular machine learning libraries which can be easily loaded in your notebook.

What Colab offers?

- As a developer, you can play out the accompanying utilizing Google Colab:
- Compose and execute code in Python.
- Document your code that bolsters mathematical equations.
- Make/Upload/Share notebooks.
- Import/Save notebooks from/to Google Drive.
- Import/Publish notebooks from GitHub.
- Import external datasets for example from Kaggle.
- Coordinate PyTorch, TensorFlow, Keras, OpenCV.
- Free Cloud service with free GPU.

Creating notebook in Google Colab

As Colab implicitly uses Google Drive for storing your notebooks, ensure that you are logged in to your Google Drive account before proceeding further.

Step 1:

Open the following URL in your browser – <https://colab.research.google.com> Your browser would display the following screen (assuming that you are logged into your Google Drive) –

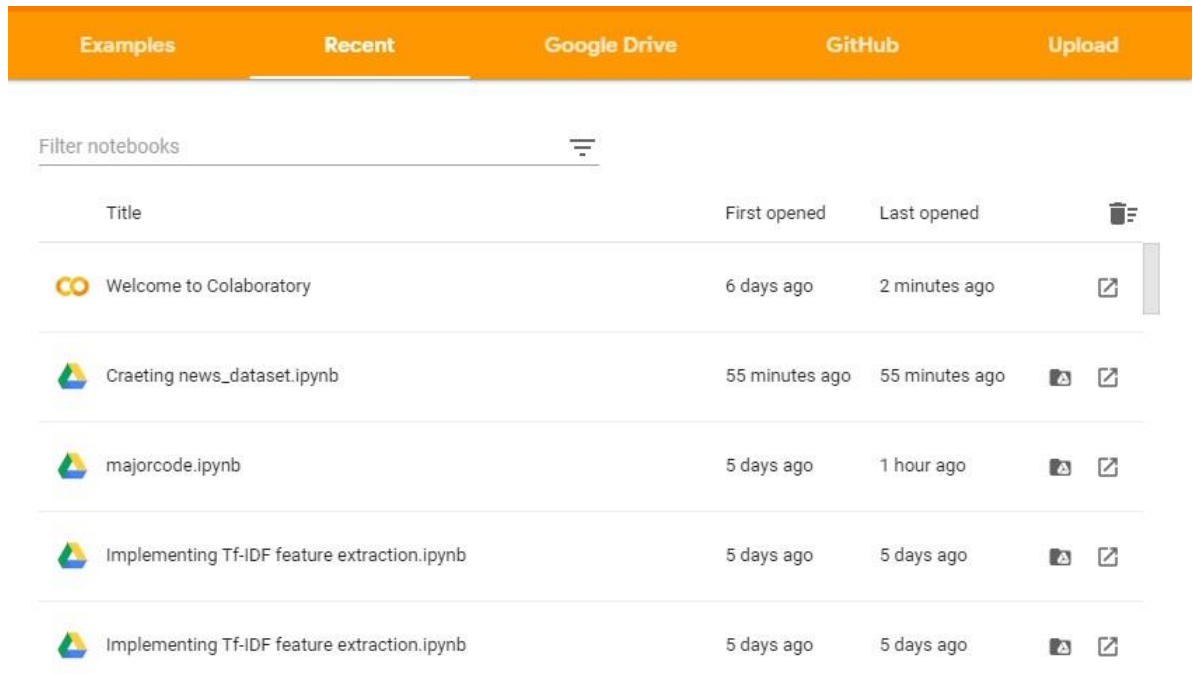


Figure 3.1 Opening Colab

Step 2:

Click on the **NEW PYTHON 3 NOTEBOOK** link at the bottom of the screen. A new notebook would open up as shown in the screen below.

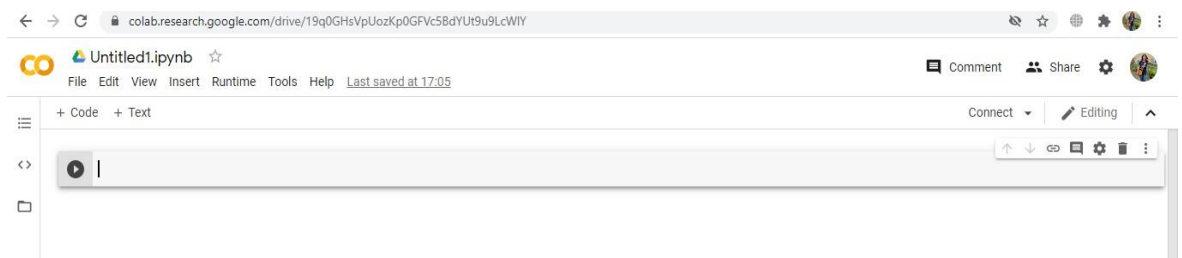


Figure 3.2 Creating Python Notebook

Setting Notebook Name:

By default, the notebook uses the naming convention UntitledXX.ipynb. To rename the notebook, click on this name and type in the desired name in the edit box as shown here –

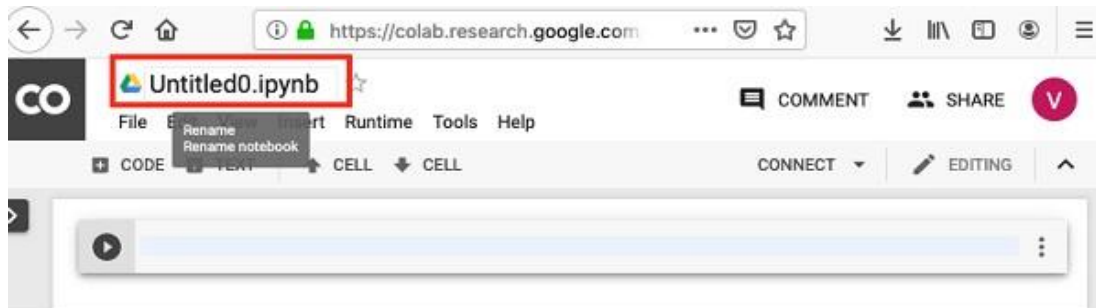


Figure 3.3 Setting Notebook Name

Entering Code:

You will now enter a trivial Python code in the code window and execute it. Enter the following two Python statements in the code window –

```
import time
print(time.ctime())
```

Figure 3.4 Entering code

Executing Code:

To execute the code, click on the arrow on the left side of the code window.

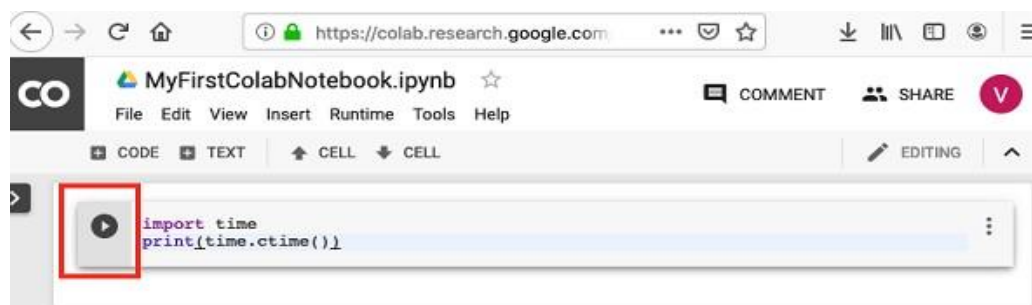


Figure 3.5 Executing code

After a while, you will see the output underneath the code window, as shown here —

Mon Jun 1 05:58:40

Figure 3.6 Result of code executed

You can clear the output anytime by clicking the icon on the left side of the output display.



Figure 3.7 Clearing output

Adding Code Cells:

To add more code to your notebook, select the following **menu** options –

Insert / Code Cell

Figure 3.8 Adding Code Cells

Alternatively, just hover the mouse at the bottom center of the Code cell. When the **CODE** and **TEXT** buttons appear, click on the **CODE** to add a new cell. This is shown in the screenshot below –

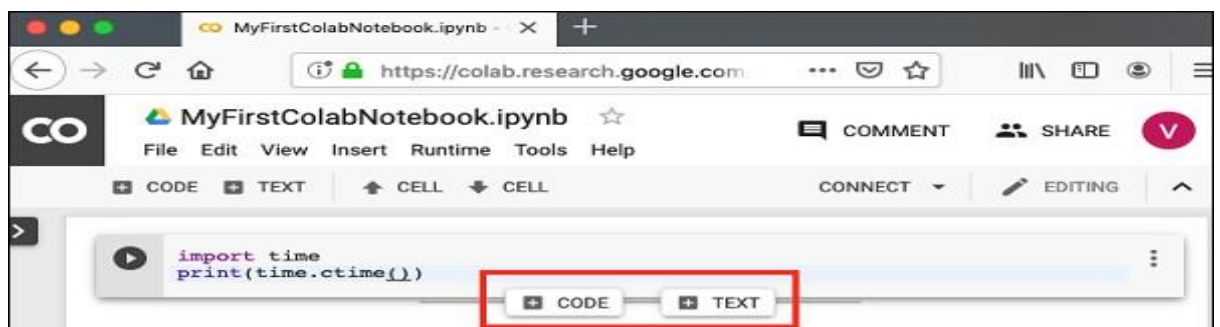


Figure 3.9 Alternate way to add code cell A new code cell will be added underneath the current cell.

Deleting Cell:

During the development of your project, you may have introduced a few now- unwanted cells in your notebook. You can remove such cells from your project easily with a single click. Click on the vertical-dotted icon at the top right corner of your code cell. Click on the **Delete cell** option and the current cell will be deleted.

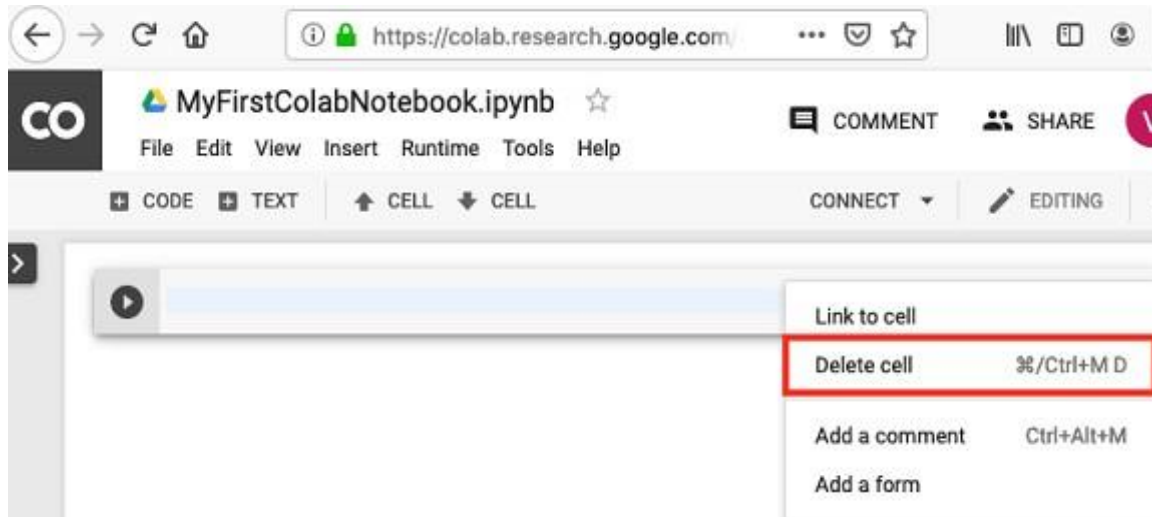


Figure 3.10 Deleting a cell

Saving Work To Google Drive:

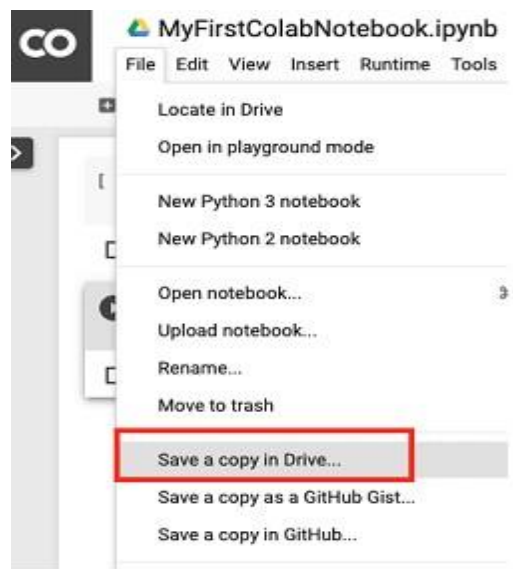


Figure 3.11 Saving work to Google Drive

Sharing Notebook:

To share the notebook that you have created with other co-developers, you may share the copy that you have made in your Google Drive. You may enter the email IDs of people with whom you would like to share the current document. You can set the kind of access by selecting from the three options shown in the above screen.

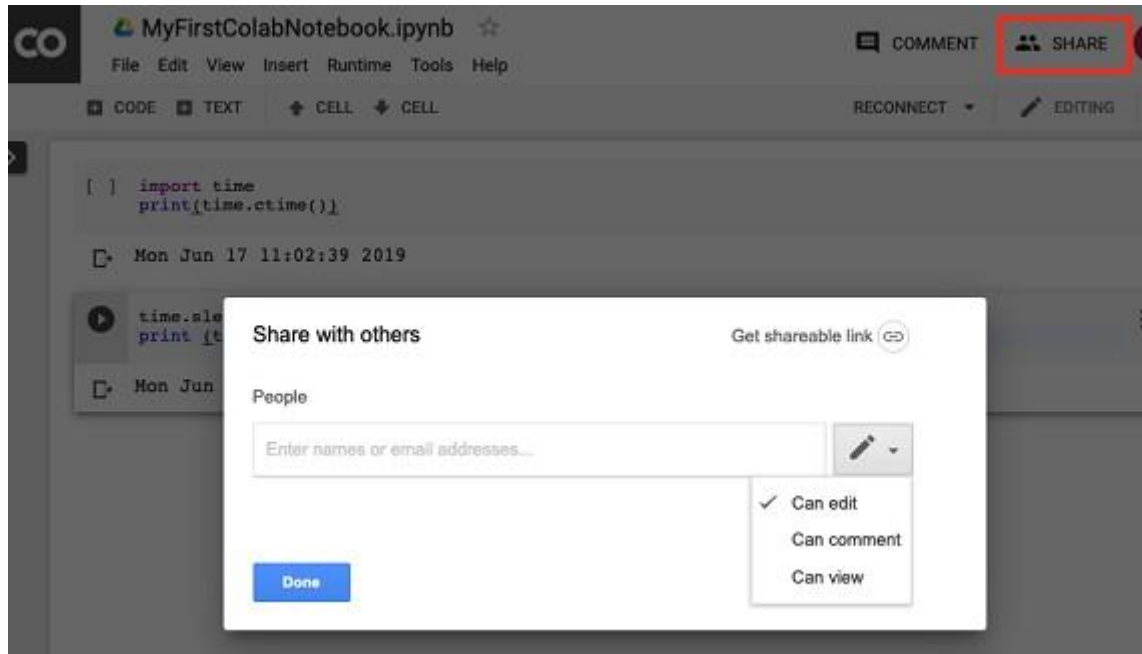


Figure 3.12 Sharing notebook

3.3. TECHNOLOGY USED

- **PYTHON**

Python is an interpreter, object-oriented, high-level programming language with dynamic semantics. It is high-level built in data structures, combined with dynamic typing and dynamic binding; make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed. Often, programmers fall in love with Python because of the increased productivity it provides. Since there is no compilation step, the edit-test-debug cycle is incredibly fast. Debugging Python programs is easy: a bug or bad input will never cause a segmentation fault. Instead, when the interpreter discovers an error, it raises an exception. When the program doesn't catch the

exception, the interpreter prints a stack trace. A source level debugger allows inspection of local and global variables, evaluation of arbitrary expressions, setting breakpoints, stepping through the code a line at a time, and so on. The debugger is written in Python itself, testifying to Python's introspective power. On the other hand, often the quickest way to debug a program is to add a few print statements to the source: the fast edit-test-debug cycle makes this simple approach very effective.

LIBRARIES USED:

- **KERAS:**

In computer programming, a Keras is an open source network Library written in Python. It can work beyond TensorFlow, Microsoft Cognitive Toolkit, or Theano. Designed to enable rapid testing of deep neural networks, focusing on ease of operation, modular, and extensible

- **NUMPY:**

It is a general-purpose array-processing package. It provides a high-performance multidimensional array object and tools for working with these arrays. It is the fundamental package for scientific computing with Python. Besides its obvious scientific uses, Numpy can also be used as an efficient multi-dimensional container of generic data.

- **MATPLOTLIB:**

Matplotlib is Python's programming language library for programming and its numerical extension NumPy Provides a site-based API in applications that use common GUI tools such as Scanter, wxPython, Qt, or GTK

- **SKLEARN:**

SciKit library provides a tool, called the Model Selection library. There's a class in the library which is, aptly, named „train_test_split.“ Using this we can easily split the dataset into the training and the testing datasets in various proportions.

Chapter 4: Implementation Work Details

4.1 DATA IMPLEMENTATION AND PROGRAM EXECUTION

The project is divided into the following steps:

1. Collection and Creation of data.
2. Preprocessing of data
3. Feature Scaling.
4. Division-of training and testing dataset.
5. Training your Dataset with different models
6. Comparing the accuracy of tradition models.
7. Plotting graphs
8. Predicting result
9. Saving model using pickl

1. DATASET COLLECTION AND CREATION

The dataset used in this project was taken from the kaggle of the online PlantVillage database, Also the code was also written to Kaggle's online kernel for accurate calculation and to check for training and validation losses. The first step of any image-based project is finding a valid database. Most of the time a standard database is selected but, in some cases, we do not get the right database. So, it is in such cases, we may collect images and build our own database. The website is located in the AI crowd which is a challenge in the classification of plant diseases. The data found here has no label. So, the first job is to clean and label details. There is a huge database so basically, images with better resolution and angles are selected. Back selection of images, we should have in-depth knowledge about the different leaves and diseases they have. Great research performed at the final site of the plantvillage organization. Different types of plant images are studied and compatible.

After a thorough study, labeling was performed by classifying images and various diseases

Diseases of different plant information:

- Pepper bell bacterial spot
- Tomatoes bacterial spot
- Potato late blight
- Potato early Blight
- Tomato plant blight

```

image_list, label_list = [], []
try:
    print("[INFO] Loading images ...")
    root_dir = listdir(directory_root)
    for directory in root_dir :
        # remove .DS_Store from list
        if directory == ".DS_Store" :
            root_dir.remove(directory)

    for plant_folder in root_dir :
        plant_disease_folder_list = listdir(f"{directory_root}/{plant_folder}")

        for disease_folder in plant_disease_folder_list :
            # remove .DS_Store from list
            if disease_folder == ".DS_Store" :
                plant_disease_folder_list.remove(disease_folder)

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

            for single_plant_disease_image in plant_disease_image_list :
                if single_plant_disease_image == ".DS_Store" :
                    plant_disease_image_list.remove(single_plant_disease_image)

    ##I picked just 200 images from each folder.

    for image in plant_disease_image_list[:200]:
        image_directory = f"{directory_root}/{plant_folder}/{plant_disease_folder}/{image}"
        if image_directory.endswith(".jpg") == True or image_directory.endswith(".JPG") == True:
            image_list.append(convert_image_to_array(image_directory))

            ##convert image to array using the function declared above.

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

```

Figure 4.1 Fetch images from the directory

```

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

```

Figure 4.2 Function to convert images into array

2. PRE-PROCESSING OF DATA AND LABELLING

Prefixing images includes removing low-frequency background sound, adjusting the size of individual particle images, deleting reflections, and hiding parts of the images. Image editing is a way to improve data. In addition, the image editing process involves cutting all the images by hand, forming a square around the leaves, to highlight the area of interest (plant leaves). During the data collection phase, images with minimal adjustment and size less than 500 pixels are not considered valid image data. Additionally, the only images in which the region of interest were the highest resolution were marked as eligible for the database. Thus, it was ensured that the pictures contained all the information needed to learn the features. Many resources can be found by searching across the Internet, but their suitability is often unreliable. In an effort to ensure the accuracy of the classrooms in the database, which was initially collected by keyword search, agricultural experts examined leaf images and labeled all images with appropriate disease names. As is well known, it is important to use accurately classified images for training and validation data. Only then can a suitable and reliable acquisition model be developed. At this stage, duplicate images left over after the start of compiling and collecting images by categories have been removed from the database.

```
image_size = len(image_list)
```

Figure 4.3 Get size of processed image

```
label_binarizer = LabelBinarizer()
image_labels = label_binarizer.fit_transform(label_list)
pickle.dump(label_binarizer, open('label_transform.pkl', 'wb'))
n_classes = len(label_binarizer.classes_)
```

Figure 4.4 Transform Image Labels using Scikit LabelBinarize

```
print(label_binarizer.classes_)
```

```
['Pepper__bell___Bacterial_spot' 'Pepper__bell___healthy'
 'Potato___Early_blight' 'Potato___Late_blight' 'Potato___healthy'
 'Tomato_Bacterial_spot' 'Tomato_Early_blight' 'Tomato_Late_blight'
 'Tomato_Leaf_Mold' 'Tomato_Septoria_leaf_spot'
 'Tomato_Spider_mites_Two_spotted_spider_mite' 'Tomato__Target_Spot'
 'Tomato__Tomato_YellowLeaf__Curl_Virus' 'Tomato__Tomato_mosaic_virus'
 'Tomato_healthy']
```

Figure 4.5 Print the classes

3. FEATURE SCALING

Feature measurement is a method used to measure the distance of independent variables or data features. Data processing is done during the data preparation process.

- Standard Deviation is another way of measuring when the values are focused on the finish line with a standard deviation of the unit. This means that the definition of responsibility becomes zero and the distribution of the result has a standard deviation of the unit.
- Normalization is a measurement in which values are converted and redeemed to end from 0 to 1. Also known as the Min-Max Scaling.

```
np_image_list = np.array(image_list, dtype=np.float16) / 225.0
```

Figure 4.6 pre-process the input data by scaling the data points

4. DIVISION OF TRAINING AND TESTING DATASET

Splitting data into Train and Test. I performed a training/testing split on the data using 80% of the images for training and 20% for testing. Train test separation is a method of testing the performance of a machine learning algorithm.

It can be used to distinguish or reverse problems and can be used for any supervised learning algorithm. The process involves taking a database and dividing it into two subsets. The first subset is used to match the model and is referred to as a training database. The second subset is not used for model training; instead, the database input is given to the model, and then the prediction is made and compared to the expected values. This second database is called a test database.

Train Database: Used to match the machine learning model.

Test Database: Used to test the appropriate machine learning model.

The aim was to measure the performance of a machine learning model in new data: data not used for model training.

In this way we expect to use the model in practice. That is, to measure it on available data by entering the known and results, and then predicting new examples in the future when we do not have the expected output or target values.

The train test procedure is appropriate if there is a large enough database available.

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

Figure 4.7 splitting dataset

```
[INFO] Splitting data to train, test
```

Figure 4.8 output of splitted dataset

5. TRAINING MODEL USING CNN

Training the deep convolutional neural network for making an image classification model from a dataset was proposed. Tensor Flow is an open source software library for numerical computation using data flow graphs. Nodes in the graph represent mathematical operations, while the graph edges represent the multidimensional data arrays (tensors) communicated between them. The flexible architecture allows you to deploy computation to one or more CPUs or GPUs in a desktop, server, or mobile device with a single API. Tensor Flow was originally developed by researchers and engineers working on the Google Brain Team within Google's Machine Intelligence research organization for the purposes of conducting machine learning and deep neural networks research, but the system is general enough to be applicable in a wide variety of other domains as well. In machine learning, a convolutional neural network is a type of feed-forward artificial neural network in which the connectivity pattern between its neurons is inspired by the organization of the animal visual cortex. Individual cortical neurons respond to stimuli in a restricted region of space known as the receptive field. The receptive fields of different neurons partially overlap such that they tile the visual field. The response of an individual neuron to stimuli within its receptive field can be approximated mathematically by a convolution operation. Convolutional networks were inspired by biological processes and are variations of multilayer perceptron designed to use minimal amounts of pre-processing. They have wide applications in image and video recognition, recommender systems and natural language processing. Convolutional neural networks (CNNs) consist of multiple layers of receptive fields. These are small neuron collections which process portions of the input image. The outputs of these collections are then tiled so that their input regions overlap, to obtain a higher-resolution representation of the original image; this is repeated for every such layer. Tiling allows CNNs to tolerate translation of the input image. Convolutional networks may include local or global pooling layers, which combine the outputs of neuron clusters. They also consist of various combinations of convolutional and fully connected layers, with point wise nonlinearity applied at the end of or

after each layer. A convolution operation on small regions of input is introduced to reduce the number of free parameters and improve generalization. One major advantage of convolutional networks is the use of shared weight in convolutional layers, which means that the same filter (weights bank) is used for each pixel in the layer; this both reduces memory footprint and improves performance. The layer's parameters are comprised of a set of learnable kernels which possess a small receptive field but extend through the full depth of the input volume. Rectified Linear Units (ReLU) are used as substitute for saturating nonlinearities. This activation function adaptively learns the parameters of rectifiers and improves accuracy at negligible extra computational cost. In the context of artificial neural networks, the rectifier is an activation function defined as:

$$f(x) = \max(0, x)$$

, where x is the input to a neuron. This is also known as a ramp function and is analogous to half-wave rectification in electrical engineering. This activation function was first introduced to a dynamical network by Hahnloser et al. in a 2000 paper in *Nature* with strong biological motivations and mathematical justifications. It has been used in convolutional networks more effectively than the widely used logistic sigmoid (which is inspired by probability theory; see logistic regression) and its more practical counterpart, the hyperbolic tangent. The rectifier is, as of 2015, the most popular activation function for deep neural networks. Deep CNN with ReLUs trains several times faster. This method is applied to the output of every convolutional and fully connected layer. Despite the output, the input normalization is not required; it is applied after ReLU nonlinearity after the first and second convolutional layer because it reduces top-1 and top-5 error rates. In CNN, neurons within a hidden layer are segmented into "feature maps." The neurons within a feature map share the same weight and bias. The neurons within the feature map search for the same feature. These neurons are unique since they are connected to different neurons in the lower layer. So for the first hidden layer, neurons within a feature map will be connected to different regions of the input image. The hidden layer is segmented into feature maps where each neuron in a feature map looks for the same feature but at different positions of the input image. Basically, the feature map is the result of applying convolution across an image. The convolutional layer is the core building block of a CNN. The layer's parameters consist of a set of learnable filters (or kernels), which have a small receptive field, but extend through the full depth of the input volume. During the forward pass, each filter is convolved across the width and height of the input volume, computing the dot product between the entries of the filter and the input and producing a 2-dimensional activation map of that filter. As a result, the network learns filters that activate

when it detects some specific type of feature at some spatial position in the input. Stacking the activation maps for all filters along the depth dimension forms the full output volume of the convolution layer. Every entry in the output volume can thus also be interpreted as an output of a neuron that looks at a small region in the input and shares parameters with neurons in the same activation map. When dealing with high-dimensional inputs such as images, it is impractical to connect neurons to all neurons in the previous volume because such network architecture does not take the spatial structure of the data into account. Convolutional networks exploit spatially local correlation by enforcing a local connectivity pattern between neurons of adjacent layers: each neuron is connected to only a small region of the input volume. The extent of this connectivity is a hyper parameter called the receptive field of the neuron. The connections are local in space (along width and height), but always extend along the entire depth of the input volume. Such architecture ensures that the learnt filters produce the strongest response to a spatially local input pattern. Three hyper parameters control the size of the output volume of the convolutional layer: the depth, stride and zero-padding.

1. **Depth** of the output volume controls the number of neurons in the layer that connect to the same region of the input volume. All of these neurons will learn to activate for different features in the input. For example, if the first Convolutional Layer takes the raw image as input, then different neurons along the depth dimension may activate in the presence of various oriented edges, or blobs of color.

2. **Stride** controls how depth columns around the spatial dimensions (width and height) are allocated. When the stride is 1, a new depth column of neurons is allocated to spatial positions only 1 spatial unit apart. This leads to heavily overlapping receptive fields between the columns, and also to large output volumes. Conversely, if higher strides are used then the receptive fields will overlap less and the resulting output volume will have smaller dimensions spatially.

3. **Stride controls** how depth columns around the spatial dimensions (width and height) are allocated. When the stride is 1, a new depth column of neurons is allocated to spatial positions only 1 spatial unit apart. This leads to heavily overlapping receptive fields between the columns, and also to large output volumes. Conversely, if higher strides are used then the receptive fields will overlap less and the resulting output volume will have smaller dimensions spatially.

Parameter sharing scheme is used in convolutional layers to control the number of free parameters. It relies on one reasonable assumption: That if one patch feature is useful to compute at some spatial position, then it should also be useful to compute at a different

position. In other words, denoting a single 2-dimensional slice of depth as a depth slice, we constrain the neurons in each depth slice to use the same weights and bias. Since all neurons in a single depth slice are sharing the same parameterization, then the forward pass in each depth slice of the CONV layer can be computed as a convolution of the neuron's weights with the input volume (hence the name: convolutional layer). Therefore, it is common to refer to the sets of weights as a filter (or a kernel), which is convolved with the input. The result of this convolution is an activation map, and the set of activation maps for each different filter are stacked together along the depth dimension to produce the output volume. Parameter Sharing contributes to the translation invariance of the CNN architecture. It is important to notice that sometimes the parameter sharing assumption may not make sense. This is especially the case when the input images to a CNN have some specific centred structure, in which we expect completely different features to be learned on different spatial locations. One practical example is when the input is faces that have been centred in the image: we might expect different eye-specific or hair-specific features to be learned in different parts of the image. In that case it is common to relax the parameter sharing scheme, and instead simply call the layer a locally connected layer. Another important layer of CNNs is the pooling layer, which is a form of nonlinear down sampling.

Pooling operation gives the form of translation invariance; it operates independently on every depth slice of the input and resizes it spatially. Overlapping pooling is beneficially applied to lessen over fitting. Also in favour of reducing over fitting, a dropout layer is used in the first two fully connected layers. But the shortcoming of dropout is that it increases training time 2-3 times comparing to a standard neural network of the exact architecture. Bayesian optimization experiments also proved that ReLUs and dropout have synergy effects, which means that it is advantageous when they are used together. The advance of CNNs refers to their ability to learn rich mid-level image representations as opposed to hand-designed low-level features used in other image classification methods

```

model = Sequential()
inputShape = (height, width, depth)
chanDim = -1
if K.image_data_format() == "channels_first":
    inputShape = (depth, height, width)
    chanDim = 1
model.add(Conv2D(32, (3, 3), padding="same", input_shape=inputShape))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=chanDim))
model.add(MaxPooling2D(pool_size=(3, 3)))
model.add(Dropout(0.25))
model.add(Conv2D(64, (3, 3), padding="same"))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=chanDim))
model.add(Conv2D(64, (3, 3), padding="same"))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=chanDim))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Conv2D(128, (3, 3), padding="same"))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=chanDim))
model.add(Conv2D(128, (3, 3), padding="same"))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=chanDim))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(1024))
model.add(Activation("relu"))
model.add(BatchNormalization())
model.add(Dropout(0.5))
model.add(Dense(n_classes))
model.add(Activation("softmax"))

```

Figure 4.9 Reading final dataset

```

opt = Adam(lr=INIT_LR, decay=INIT_LR / EPOCHS)
# distribution
model.compile(loss="binary_crossentropy", optimizer=opt, metrics=["accuracy"])
# train the network
print("[INFO] training network...")

```

Figure 4.10 Training network

```

EPOCHS = 15
INIT_LR = 1e-3
BS = 32
default_image_size = tuple((256, 256))
image_size = 0
directory_root = '../input/plantdisease'
width=256
height=256
depth=3

```

Figure 4.11 Initializing the Batch size and Epochs

```

history = model.fit_generator(
    aug.flow(x_train, y_train, batch_size=BS),
    validation_data=(x_test, y_test),
    steps_per_epoch=len(x_train) // BS,
    epochs=EPOCHS, verbose=1
)

```

Figure 4.12 model.fit_generator

6. COMPARING THE ACCURACY

Model accuracy is the measure used to determine which model is best at detecting relationships and patterns between data variables based on input, or training, data. The cost of errors can be huge, but improving the accuracy of the model reduces those costs. Of course, there is a point to lower returns where the cost of making a more accurate model will not lead to a corresponding profit growth, but it is often beneficial to the rest of the board. A better model can add ‘invisible’ data, better predictions and more insight that can produce, which also brings more business value. I calculated the accuracy of my model using the verification data (x_test and y_test) generated earlier. I got an accurate score of 96.77%.

```

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

```

Figure 4.13 Calculating Model accuracy

```

[INFO] Calculating model accuracy
591/591 [=====] - 1s 2ms/step
Test Accuracy: 96.44670223221561

```

Figure 4.14 Accuracy output

7. PLOTTING GRAPHS

```
acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(acc) + 1)
#Train and validation accuracy
plt.plot(epochs, acc, 'b', label='Training accuracy')
plt.plot(epochs, val_acc, 'r', label='Validation accuracy')
plt.title('Training and Validation accuracy')
plt.legend()

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

Figure 4.15 Plotting graphs

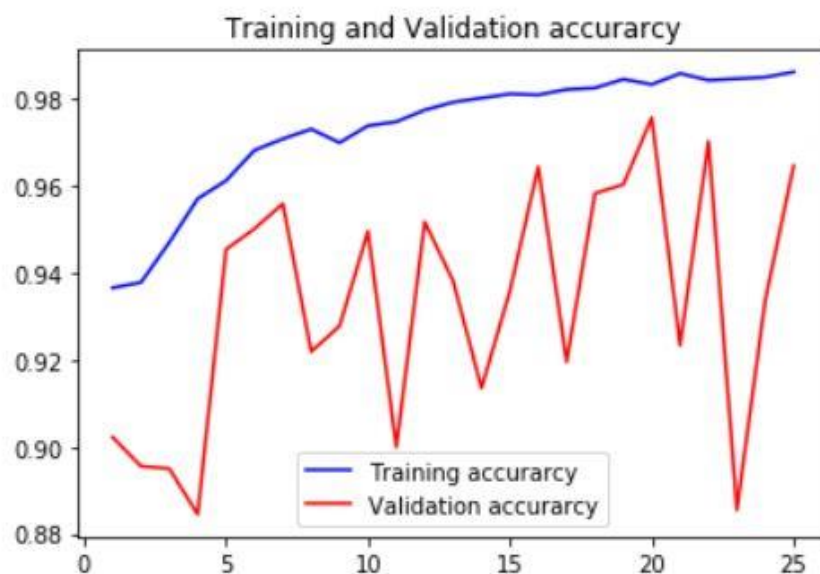


Figure 4.16 Training and validation accuracy graph

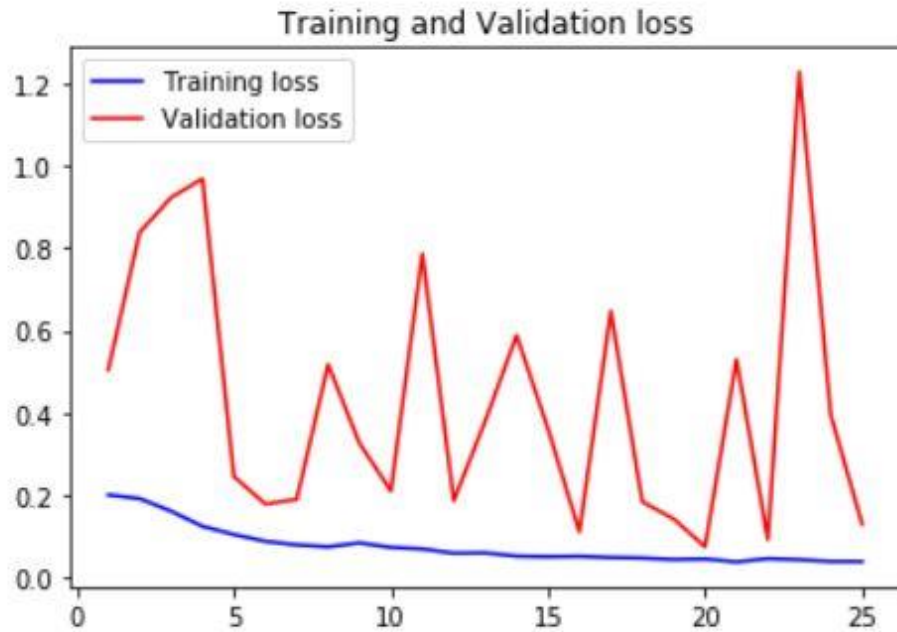


Figure 4.17 Training and validation loss graph

8. PREDICTING RESULT

The Model.PREDICT function can be used to predict results using a model. Prediction can be made during model creation, after model creation, or after failure (as long as at least one iteration is completed). Model.PREDICT always uses model weights from a successful final iteration.

The output of the Model.PREDICT function has many lines as the input table, and includes all columns in the input table and all output columns from the model.

```
loaded_model = pickle.load(open('cnn_model.pkl', 'rb'))

image_dir="../../input/plantdisease/PlantVillage/Pepper__bell___Bacterial_spot/0022d6b7-d47c-4ee2-ae9a-392a53f48647___JR_B.Spot 85

im=convert_image_to_array(image_dir)
np_image_li = np.array(im, dtype=np.float16) / 225.0
npp_image = np.expand_dims(np_image_li, axis=0)

result=model.predict(npp_image)
print(result)
```

Figure 4.18 Predicting result

9. SAVING MODEL USING PICKL

Pickle is the standard way of serializing objects in Python. You can use pickle functionality to create learning algorithms for your machine in a different way and save the serial format to a file. You can later upload this file to disable (deserializing) your model and use it to make new predictions

```
# save the model to disk
print("[INFO] Saving model...")
pickle.dump(model, open('cnn_model.pkl', 'wb'))
```

Figure 4.19 Saving model using pickl

```
[INFO] Saving model...
```

Figure 4.20 Saved model

Chapter 5: Source Code/ Simulation Code

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

#Function to convert images to array

def convert_image_to_array(image_dir):
    try:
        image = cv2.imread(image_dir)
        if image is not None :
            image = cv2.resize(image, default_image_size)
            return img_to_array(image)
        else :
            return np.array([])
    except Exception as e:
```

```

        print(f"Error : {e}")
        return None

# Fetch images from directory, load the dataset — images of diseased plants

image_list, label_list = [], []

try:
    print("[INFO] Loading images ...")
    root_dir = listdir(directory_root)
    for directory in root_dir :
        # remove .DS_Store from list
        if directory == ".DS_Store" :
            root_dir.remove(directory)

    for plant_folder in root_dir :
        plant_disease_folder_list = listdir(f"{directory_root}/{plant_folder}")

        for disease_folder in plant_disease_folder_list :
            # remove .DS_Store from list
            if disease_folder == ".DS_Store" :
                plant_disease_folder_list.remove(disease_folder)

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

            for single_plant_disease_image in plant_disease_image_list :
                if single_plant_disease_image == ".DS_Store" :
                    plant_disease_image_list.remove(single_plant_disease_image)

##I picked just 200 images from each folder.

    for image in plant_disease_image_list[:200]:
        image_directory = f"{directory_root}/{plant_folder}/{plant_disease_folder}/{image}"

```

```

        if image_directory.endswith(".jpg") == True or image_directory.endswith(".JPG") == True:
            image_list.append(convert_image_to_array(image_directory))

        ##convert image to array using the function declared above.

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

# Get Size of Processed Image

image_size = len(image_list)

# Transform Image Labels using Scikit LabelBinarizer

label_binarizer = LabelBinarizer()
image_labels = label_binarizer.fit_transform(label_list)
pickle.dump(label_binarizer, open('label_transform.pkl', 'wb'))
n_classes = len(label_binarizer.classes_)
print(label_binarizer.classes_)

np_image_list = np.array(image_list, dtype=np.float16) / 225.0

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

```

I created an image generator object which performs random rotations, shifts, flips, crops, and sheers on our image dataset. This allows us to use a smaller dataset and still achieve high results.

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

Next I created my model. In the model we are defaulting to “channel_last” architecture but also creating a switch for backends that support “channel_first” on the fourth line. Then I created the first CONV => RELU => POOL. Our CONV layer has 32 filters with a 3 x 3 kernel and RELU activation (Rectified Linear Unit). We apply batch normalization, max pooling, and 25% (0.25) dropout. Dropout is a regularization technique for reducing overfitting in neural networks by preventing complex co-adaptations on training data. It is a very efficient way of performing model averaging with neural networks. Next I created two sets of (CONV => RELU) * 2 => POOL blocks. Then only one set of FC (Fully Connected Layer)=> RELU layers.

```
model = Sequential()
inputShape = (height, width, depth)
chanDim = -1
if K.image_data_format() == "channels_first":
    inputShape = (depth, height, width)
    chanDim = 1
model.add(Conv2D(32, (3, 3), padding="same", input_shape=inputShape))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=chanDim))
model.add(MaxPooling2D(pool_size=(3, 3)))
model.add(Dropout(0.25))
model.add(Conv2D(64, (3, 3), padding="same"))
model.add(Activation("relu"))
```

```

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

```

```

model.summary()

```

```

EPOCHS = 25
INIT_LR = 1e-3
BS = 32
default_image_size = tuple((256, 256))
image_size = 0
directory_root = '../input/plantdisease'
width=256
height=256

```

```
depth=3
```

```
opt = Adam(lr=INIT_LR, decay=INIT_LR / EPOCHS)
# distribution
model.compile(loss="binary_crossentropy", optimizer=opt, metrics=["accuracy"])
# train the network
print("[INFO] training network...")
```

```
history = model.fit_generator(
    aug.flow(x_train, y_train, batch_size=BS),
    validation_data=(x_test, y_test),
    steps_per_epoch=len(x_train) // BS,
    epochs=EPOCHS, verbose=1
)
```

```
# Plot the train and val curve
```

```
acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(acc) + 1)
#Train and validation accuracy
plt.plot(epochs, acc, 'b', label='Training accuracy')
plt.plot(epochs, val_acc, 'r', label='Validation accuracy')
plt.title('Training and Validation accuracy')
plt.legend()
```

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



```

plt.legend()
plt.show()
#model accuracy

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

# save the model to disk
print("[INFO] Saving model...")
pickle.dump(model,open('cnn_model.pkl', 'wb'))

loaded_model = pickle.load(open('cnn_model.pkl', 'rb'))

image_dir="../input/plantdisease/PlantVillage/Pepper__bell__Bacterial_spot/0022d6b7-d47c-4ee2-ae9a-392a53f48647__JR_B.Spot 8964.JPG"

im=convert_image_to_array(image_dir)
np_image_li = np.array(im, dtype=np.float16) / 225.0
npp_image = np.expand_dims(np_image_li, axis=0)

result=model.predict(npp_image)
print(result)

```

Chapter 6: Input/output Screens/ Model's Photograph

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

Figure 6.1 Importing packages

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

Figure 6.2 Function to convert images into array

```

image_list, label_list = [], []
try:
    print("[INFO] Loading images ...")
    root_dir = listdir(directory_root)
    for directory in root_dir :
        # remove .DS_Store from list
        if directory == ".DS_Store" :
            root_dir.remove(directory)

    for plant_folder in root_dir :
        plant_disease_folder_list = listdir(f"{directory_root}/{plant_folder}")

        for disease_folder in plant_disease_folder_list :
            # remove .DS_Store from list
            if disease_folder == ".DS_Store" :
                plant_disease_folder_list.remove(disease_folder)

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

            for single_plant_disease_image in plant_disease_image_list :
                if single_plant_disease_image == ".DS_Store" :
                    plant_disease_image_list.remove(single_plant_disease_image)

            ##I picked just 200 images from each folder.

            for image in plant_disease_image_list[:200]:
                image_directory = f"{directory_root}/{plant_folder}/{plant_disease_folder}/{image}"
                if image_directory.endswith(".jpg") == True or image_directory.endswith(".JPG") == True:
                    image_list.append(convert_image_to_array(image_directory))

                ##convert image to array using the function declared above.

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

```

Figure 6.3 Fetch images from directory load the dataset — images of diseased plants

```

[INFO] Loading images ...
[INFO] Processing Pepper__bell___Bacterial_spot ...
[INFO] Processing Potato___healthy ...
[INFO] Processing Tomato_Leaf_Mold ...
[INFO] Processing Tomato__Tomato_YellowLeaf__Curl_Virus ...
[INFO] Processing Tomato_Bacterial_spot ...
[INFO] Processing Tomato_Septoria_leaf_spot ...
[INFO] Processing Tomato_healthy ...
[INFO] Processing Tomato_Spider_mites_Two_spotted_spider_mite ...
[INFO] Processing Tomato_Early_blight ...
[INFO] Processing Tomato__Target_Spot ...
[INFO] Processing Pepper__bell___healthy ...
[INFO] Processing Potato___Late_blight ...
[INFO] Processing Tomato_Late_blight ...
[INFO] Processing Potato___Early_blight ...
[INFO] Processing Tomato__Tomato_mosaic_virus ...
[INFO] Processing PlantVillage ...
[INFO] Image loading completed

```

Figure 6.4 Output of loading the dataset

```

image_size = len(image_list)

```

Figure 6.5 Get size of processed image

```
label_binarizer = LabelBinarizer()
image_labels = label_binarizer.fit_transform(label_list)
pickle.dump(label_binarizer, open('label_transform.pkl', 'wb'))
n_classes = len(label_binarizer.classes_)
```

Figure 6.6 Transform Image Labels using Scikit LabelBinarizer

```
print(label_binarizer.classes_)
```

Figure 6.7 Print the classes

```
['Pepper__bell___Bacterial_spot' 'Pepper__bell___healthy'
 'Potato___Early_blight' 'Potato___Late_blight' 'Potato___healthy'
 'Tomato_Bacterial_spot' 'Tomato_Early_blight' 'Tomato_Late_blight'
 'Tomato_Leaf_Mold' 'Tomato_Septoria_leaf_spot'
 'Tomato_Spider_mites_Two_spotted_spider_mite' 'Tomato__Target_Spot'
 'Tomato__Tomato_YellowLeaf__Curl_Virus' 'Tomato__Tomato_mosaic_virus'
 'Tomato_healthy']
```

Figure 6.8 Output of print(label_binariser.classes_)

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

Figure 6.9 Splitting data into test and train sets

```
[INFO] Splitting data to train, test
```

Figure 6.10 Output of above operation


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

Figure 6.11 Creation of an image generator object

```
model = Sequential()
inputShape = (height, width, depth)
chanDim = -1
if K.image_data_format() == "channels_first":
    inputShape = (depth, height, width)
    chanDim = 1
model.add(Conv2D(32, (3, 3), padding="same", input_shape=inputShape))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=chanDim))
model.add(MaxPooling2D(pool_size=(3, 3)))
model.add(Dropout(0.25))
model.add(Conv2D(64, (3, 3), padding="same"))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=chanDim))
model.add(Conv2D(64, (3, 3), padding="same"))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=chanDim))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Conv2D(128, (3, 3), padding="same"))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=chanDim))
model.add(Conv2D(128, (3, 3), padding="same"))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=chanDim))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(1024))
model.add(Activation("relu"))
model.add(BatchNormalization())
model.add(Dropout(0.5))
model.add(Dense(n_classes))
model.add(Activation("softmax"))
```

Figure 6.12 Model creation

```
model.summary()
```

Figure 6.13 Model summary

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 256, 256, 32)	896
activation_1 (Activation)	(None, 256, 256, 32)	0
batch_normalization_1 (Batch Normalization)	(None, 256, 256, 32)	128
max_pooling2d_1 (MaxPooling2D)	(None, 85, 85, 32)	0
dropout_1 (Dropout)	(None, 85, 85, 32)	0
conv2d_2 (Conv2D)	(None, 85, 85, 64)	18496
activation_2 (Activation)	(None, 85, 85, 64)	0
batch_normalization_2 (Batch Normalization)	(None, 85, 85, 64)	256
conv2d_3 (Conv2D)	(None, 85, 85, 64)	36928
activation_3 (Activation)	(None, 85, 85, 64)	0
batch_normalization_3 (Batch Normalization)	(None, 85, 85, 64)	256
max_pooling2d_2 (MaxPooling2D)	(None, 42, 42, 64)	0
dropout_2 (Dropout)	(None, 42, 42, 64)	0
conv2d_4 (Conv2D)	(None, 42, 42, 128)	73856
activation_4 (Activation)	(None, 42, 42, 128)	0
batch_normalization_4 (Batch Normalization)	(None, 42, 42, 128)	512
conv2d_5 (Conv2D)	(None, 42, 42, 128)	147584
activation_5 (Activation)	(None, 42, 42, 128)	0
batch_normalization_5 (Batch Normalization)	(None, 42, 42, 128)	512
max_pooling2d_3 (MaxPooling2D)	(None, 21, 21, 128)	0
dropout_3 (Dropout)	(None, 21, 21, 128)	0
flatten_1 (Flatten)	(None, 56448)	0
dense_1 (Dense)	(None, 1024)	57803776
activation_6 (Activation)	(None, 1024)	0
batch_normalization_6 (Batch Normalization)	(None, 1024)	4096
dropout_4 (Dropout)	(None, 1024)	0
dense_2 (Dense)	(None, 15)	15375
activation_7 (Activation)	(None, 15)	0
Total params: 58,102,671		
Trainable params: 58,099,791		
Non-trainable params: 2,880		

Figure 6.14 Output of model.summary()

```

opt = Adam(lr=INIT_LR, decay=INIT_LR / EPOCHS)
# distribution
model.compile(loss="binary_crossentropy", optimizer=opt, metrics=["accuracy"])
# train the network
print("[INFO] training network...")

```

Figure 6.15 Training network

```
[INFO] training network...
```

Figure 6.16 Output of above operation

Figure 6.17 Initializing the epochs and batch size

```

EPOCHS = 15
INIT_LR = 1e-3
BS = 32
default_image_size = tuple((256, 256))
image_size = 0
directory_root = '../input/plantdisease'
width=256
height=256
depth=3

```

```

history = model.fit_generator(
    aug.flow(x_train, y_train, batch_size=BS),
    validation_data=(x_test, y_test),
    steps_per_epoch=len(x_train) // BS,
    epochs=EPOCHS, verbose=1
)

```

Figure 6.18 Model fitting function


```

Epoch 1/25
73/73 [=====] - 43s 583ms/step - loss: 0.2812 - acc: 0.9365 - val_
loss: 0.5047 - val_acc: 0.9822
Epoch 2/25
73/73 [=====] - 35s 479ms/step - loss: 0.1928 - acc: 0.9378 - val_
loss: 0.8381 - val_acc: 0.8955
Epoch 3/25
73/73 [=====] - 36s 491ms/step - loss: 0.1617 - acc: 0.9469 - val_
loss: 0.9227 - val_acc: 0.8958
Epoch 4/25
73/73 [=====] - 36s 490ms/step - loss: 0.1249 - acc: 0.9569 - val_
loss: 0.9686 - val_acc: 0.8845
Epoch 5/25
73/73 [=====] - 34s 478ms/step - loss: 0.1852 - acc: 0.9611 - val_
loss: 0.2461 - val_acc: 0.9453
Epoch 6/25
73/73 [=====] - 36s 487ms/step - loss: 0.8884 - acc: 0.9688 - val_
loss: 0.1789 - val_acc: 0.9508
Epoch 7/25
73/73 [=====] - 34s 468ms/step - loss: 0.8798 - acc: 0.9787 - val_
loss: 0.1982 - val_acc: 0.9558
Epoch 8/25
73/73 [=====] - 35s 483ms/step - loss: 0.8752 - acc: 0.9729 - val_
loss: 0.5176 - val_acc: 0.9218
Epoch 9/25
73/73 [=====] - 35s 476ms/step - loss: 0.8848 - acc: 0.9698 - val_
loss: 0.3263 - val_acc: 0.9277
Epoch 10/25
73/73 [=====] - 35s 482ms/step - loss: 0.8737 - acc: 0.9736 - val_
loss: 0.2185 - val_acc: 0.9495
Epoch 11/25
73/73 [=====] - 35s 481ms/step - loss: 0.8699 - acc: 0.9746 - val_
loss: 0.7867 - val_acc: 0.8999
Epoch 12/25
73/73 [=====] - 35s 473ms/step - loss: 0.8598 - acc: 0.9772 - val_
loss: 0.1873 - val_acc: 0.9516
Epoch 13/25
73/73 [=====] - 35s 478ms/step - loss: 0.8684 - acc: 0.9791 - val_
loss: 0.3881 - val_acc: 0.9381
Epoch 14/25
73/73 [=====] - 34s 468ms/step - loss: 0.8526 - acc: 0.9808 - val_
loss: 0.5882 - val_acc: 0.9135
Epoch 15/25
73/73 [=====] - 36s 487ms/step - loss: 0.8512 - acc: 0.9818 - val_
loss: 0.3619 - val_acc: 0.9357
Epoch 16/25
73/73 [=====] - 35s 482ms/step - loss: 0.8528 - acc: 0.9808 - val_
loss: 0.1112 - val_acc: 0.9642
Epoch 17/25
73/73 [=====] - 34s 472ms/step - loss: 0.8491 - acc: 0.9828 - val_
loss: 0.6471 - val_acc: 0.9195
Epoch 18/25
73/73 [=====] - 35s 484ms/step - loss: 0.8478 - acc: 0.9824 - val_
loss: 0.1848 - val_acc: 0.9582
Epoch 19/25
73/73 [=====] - 34s 469ms/step - loss: 0.8443 - acc: 0.9843 - val_
loss: 0.1428 - val_acc: 0.9682
Epoch 20/25
73/73 [=====] - 35s 483ms/step - loss: 0.8455 - acc: 0.9832 - val_
loss: 0.8754 - val_acc: 0.9755
Epoch 21/25
73/73 [=====] - 35s 476ms/step - loss: 0.8384 - acc: 0.9857 - val_
loss: 0.5299 - val_acc: 0.9233
Epoch 22/25
73/73 [=====] - 35s 481ms/step - loss: 0.8463 - acc: 0.9842 - val_
loss: 0.8925 - val_acc: 0.9781
Epoch 23/25
73/73 [=====] - 35s 481ms/step - loss: 0.8448 - acc: 0.9844 - val_
loss: 1.2281 - val_acc: 0.8855
Epoch 24/25
73/73 [=====] - 34s 472ms/step - loss: 0.8396 - acc: 0.9847 - val_
loss: 0.3955 - val_acc: 0.9337
Epoch 25/25
73/73 [=====] - 35s 483ms/step - loss: 0.8394 - acc: 0.9868 - val_
loss: 0.1388 - val_acc: 0.9645

```

Figure 6.19 Output of above function

Chapter 7: System Testing

```
opt = Adam(lr=INIT_LR, decay=INIT_LR / EPOCHS)
# distribution
model.compile(loss="binary_crossentropy", optimizer=opt, metrics=["accuracy"])
# train the network
print("[INFO] training network...")
```

Figure 7.1 Training Network

```
[INFO] training network...
```

Figure 7.2 Output of training network

```
EPOCHS = 15
INIT_LR = 1e-3
BS = 32
default_image_size = tuple((256, 256))
image_size = 0
directory_root = '../input/plantdisease'
width=256
height=256
depth=3
```

Figure 7.3 Initializing the epochs and batch size

```
history = model.fit_generator(
    aug.flow(x_train, y_train, batch_size=BS),
    validation_data=(x_test, y_test),
    steps_per_epoch=len(x_train) // BS,
    epochs=EPOCHS, verbose=1
)
```

Figure 7.4 Model fit generator

```

Epoch 1/25
73/73 [=====] - 43s 583ms/step - loss: 0.2812 - acc: 0.9365 - val_
loss: 0.5847 - val_acc: 0.9822
Epoch 2/25
73/73 [=====] - 35s 479ms/step - loss: 0.1928 - acc: 0.9378 - val_
loss: 0.8381 - val_acc: 0.8955
Epoch 3/25
73/73 [=====] - 36s 491ms/step - loss: 0.1617 - acc: 0.9469 - val_
loss: 0.9227 - val_acc: 0.8958
Epoch 4/25
73/73 [=====] - 36s 498ms/step - loss: 0.1249 - acc: 0.9569 - val_
loss: 0.9686 - val_acc: 0.8845
Epoch 5/25
73/73 [=====] - 34s 478ms/step - loss: 0.1052 - acc: 0.9611 - val_
loss: 0.2461 - val_acc: 0.9453
Epoch 6/25
73/73 [=====] - 36s 487ms/step - loss: 0.0884 - acc: 0.9688 - val_
loss: 0.1789 - val_acc: 0.9588
Epoch 7/25
73/73 [=====] - 34s 468ms/step - loss: 0.0798 - acc: 0.9787 - val_
loss: 0.1982 - val_acc: 0.9558
Epoch 8/25
73/73 [=====] - 35s 483ms/step - loss: 0.0752 - acc: 0.9729 - val_
loss: 0.5176 - val_acc: 0.9218
Epoch 9/25
73/73 [=====] - 35s 476ms/step - loss: 0.0848 - acc: 0.9698 - val_
loss: 0.3263 - val_acc: 0.9277
Epoch 10/25
73/73 [=====] - 35s 482ms/step - loss: 0.0737 - acc: 0.9736 - val_
loss: 0.2185 - val_acc: 0.9495
Epoch 11/25
73/73 [=====] - 35s 481ms/step - loss: 0.0699 - acc: 0.9746 - val_
loss: 0.7867 - val_acc: 0.8999
Epoch 12/25
73/73 [=====] - 35s 473ms/step - loss: 0.0598 - acc: 0.9772 - val_
loss: 0.1873 - val_acc: 0.9516
Epoch 13/25
73/73 [=====] - 35s 478ms/step - loss: 0.0684 - acc: 0.9791 - val_
loss: 0.3881 - val_acc: 0.9381
Epoch 14/25
73/73 [=====] - 34s 468ms/step - loss: 0.0526 - acc: 0.9888 - val_
loss: 0.5882 - val_acc: 0.9135
Epoch 15/25
73/73 [=====] - 36s 487ms/step - loss: 0.0512 - acc: 0.9818 - val_
loss: 0.3619 - val_acc: 0.9357
Epoch 16/25
73/73 [=====] - 35s 482ms/step - loss: 0.0528 - acc: 0.9888 - val_
loss: 0.1112 - val_acc: 0.9642
Epoch 17/25
73/73 [=====] - 34s 472ms/step - loss: 0.0491 - acc: 0.9828 - val_
loss: 0.6471 - val_acc: 0.9195
Epoch 18/25
73/73 [=====] - 35s 484ms/step - loss: 0.0478 - acc: 0.9824 - val_
loss: 0.1848 - val_acc: 0.9582
Epoch 19/25
73/73 [=====] - 34s 469ms/step - loss: 0.0443 - acc: 0.9843 - val_
loss: 0.1428 - val_acc: 0.9682
Epoch 20/25
73/73 [=====] - 35s 483ms/step - loss: 0.0455 - acc: 0.9832 - val_
loss: 0.0754 - val_acc: 0.9755
Epoch 21/25
73/73 [=====] - 35s 476ms/step - loss: 0.0384 - acc: 0.9857 - val_
loss: 0.5299 - val_acc: 0.9233
Epoch 22/25
73/73 [=====] - 35s 481ms/step - loss: 0.0463 - acc: 0.9842 - val_
loss: 0.0925 - val_acc: 0.9781
Epoch 23/25
73/73 [=====] - 35s 481ms/step - loss: 0.0448 - acc: 0.9844 - val_
loss: 1.2281 - val_acc: 0.8855
Epoch 24/25
73/73 [=====] - 34s 472ms/step - loss: 0.0396 - acc: 0.9847 - val_
loss: 0.3955 - val_acc: 0.9337
Epoch 25/25
73/73 [=====] - 35s 483ms/step - loss: 0.0394 - acc: 0.9868 - val_
loss: 0.1308 - val_acc: 0.9645

```

Figure 7.5 output of training model

```

acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(acc) + 1)
#Train and validation accuracy
plt.plot(epochs, acc, 'b', label='Training accuracy')
plt.plot(epochs, val_acc, 'r', label='Validation accuracy')
plt.title('Training and Validation accuracy')
plt.legend()

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

```

Figure 7.6 Plotting the train and validation curve

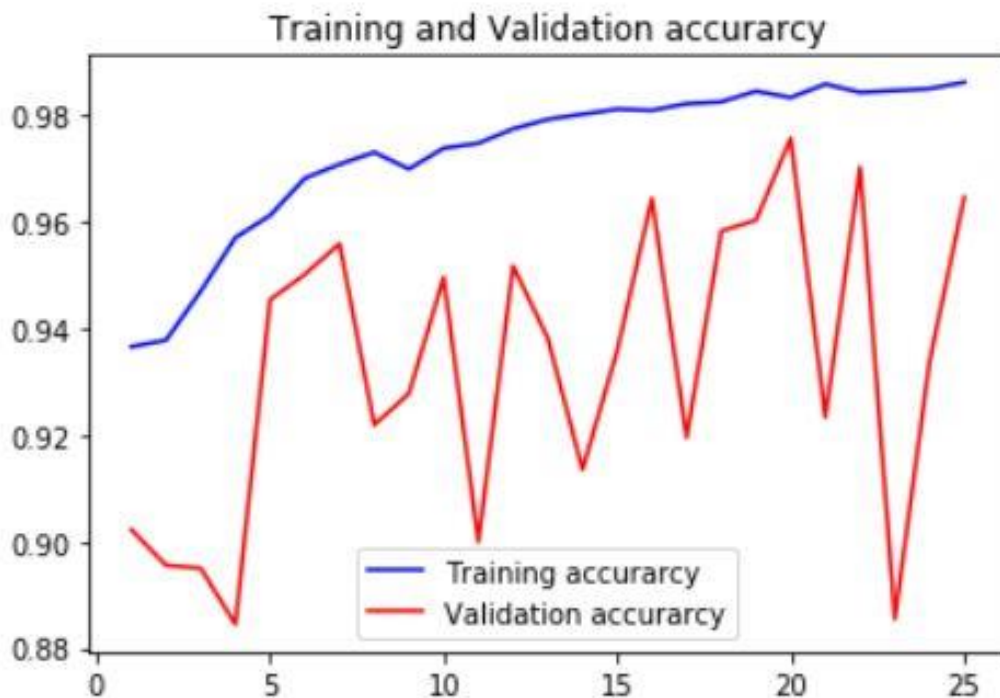


Figure 7.7 Training and validation Accuracy

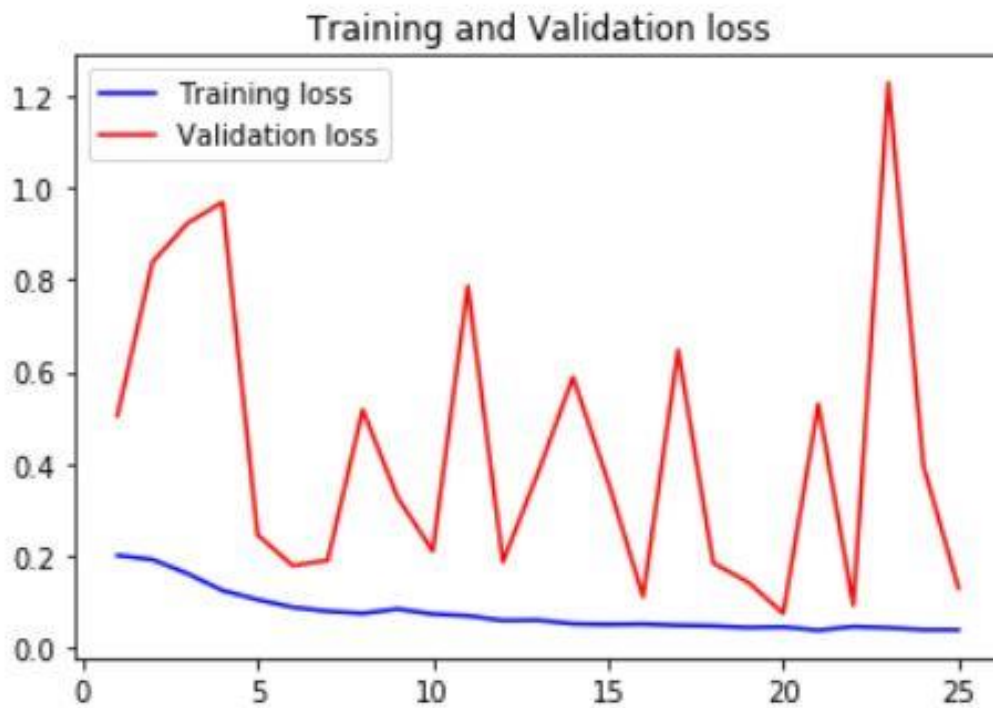


Figure 7.8 Training and validation loss

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

Figure 7.9 Model Accuracy

```
[INFO] Calculating model accuracy
591/591 [=====] - 1s 2ms/step
Test Accuracy: 96.44670223221561
```

Figure 7.10 Model accuracy output

```
# save the model to disk  
print("[INFO] Saving model...")  
pickle.dump(model, open('cnn_model.pkl', 'wb'))
```

Figure 7.11 Saving model using Pickl

```
[INFO] Saving model...
```

Figure 7.12 output of saving model

Chapter 8: Individual Contribution

For completion of project I have performed the following steps:

1. Dataset creation.
2. Preprocessing and labelling.
3. Feature Scaling.
4. Training model using CNN.
5. Graph plotting
6. Model accuracy
7. Saving model using Pickl
8. Prediction

8.1 DATASET CREATION

The dataset used in this project was taken from the kaggle of the online PlantVillage database, Also the code was also written to Kaggle's online kernel for accurate calculation and to check for training and validation losses. The first step of any image-based project is finding a valid database. Most of the time a standard database is selected but, in some cases, we do not get the right database. So, it is in such cases, we may collect images and build our own database. The website is located in the AI crowd which is a challenge in the classification of plant diseases. The data found here has no label. So, the first job is to clean and label details. There is a huge database so basically, images with better resolution and angles are selected. Back selection of images, we should have in-depth knowledge about the different leaves and diseases they have. Great research performed at the final site of the plantvillage organization. Different types of plant images are studied and compatible.

After a thorough study, labeling was performed by classifying images and various diseases

Diseases of different plant information:

- Pepper bell bacterial spot
- Tomatoes bacterial spot
- Potato late blight
- Potato early Blight
- Tomato plant blight



Figure 8.1 Dataset of infected leaves of various plant

8.2 DATA PREPROCESSING AND LABELLING

Prefixing images includes removing low-frequency background sound, adjusting the size of individual particle images, deleting reflections, and hiding parts of the images. Image editing is a way to improve data. In addition, the image editing process involves cutting all the images by hand, forming a square around the leaves, to highlight the area of interest (plant leaves). During the data collection phase, images with minimal adjustment and size less than 500 pixels are not considered valid image data. Additionally, the only images in which the region of interest were the highest resolution were marked as eligible for the database. Thus, it was ensured that the pictures contained all the information needed to learn the features. Many resources can be found by searching across the Internet, but their suitability is often unreliable. In an effort to ensure the accuracy of the classrooms in the database, which was initially collected by keyword search, agricultural experts examined leaf images and labeled

all images with appropriate disease names. As is well known, it is important to use accurately classified images for training and validation data. Only then can a suitable and reliable acquisition model be developed. At this stage, duplicate images left over after the start of compiling and collecting images by categories have been removed from the database.

8.3 FEATURE SCALING

Feature measurement is a method used to measure the distance of independent variables or data features. Data processing is done during the data preparation process.

- Standard Deviation is another way of measuring when the values are focused on the finish line with a standard deviation of the unit. This means that the definition of responsibility becomes zero and the distribution of the result has a standard deviation of the unit.

- Normalization is a measurement in which values are converted and redeemed to end from 0 to 1. Also known as the Min-Max Scaling.

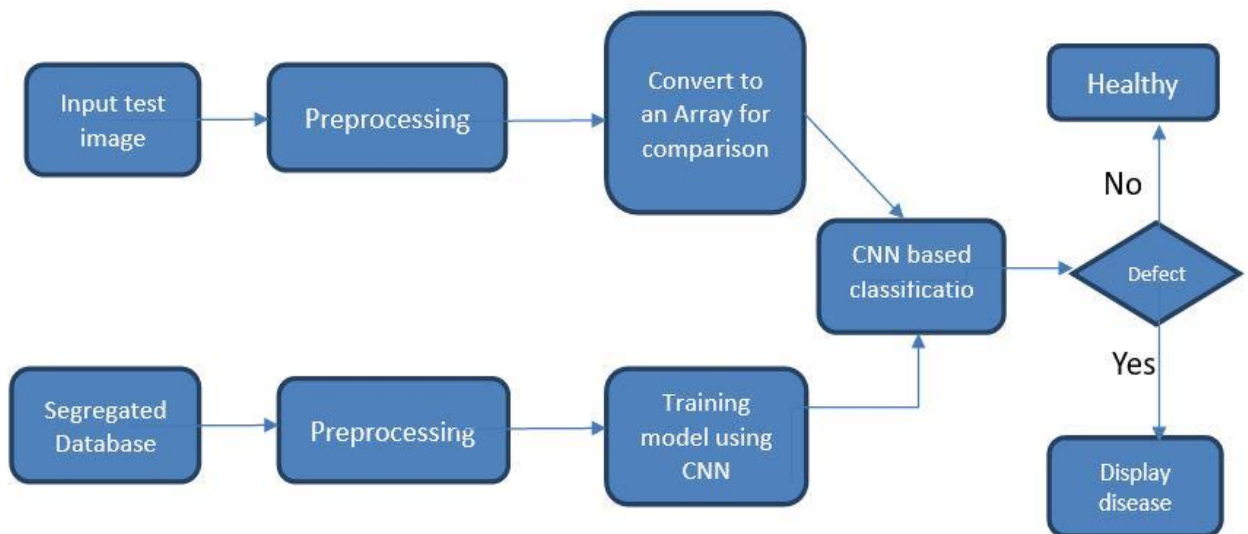


Figure 8.2 System Flowchart

8.4 TRAINING MODEL USING CNN

The database has been redesigned as Image reshaping, resizing, and conversion to an array form. The same analysis is also performed on the test image. Data containing approximately 32000 plant species is available, where any image can be used as a test image of the software. The training website is used for modeling training (CNN) to be able to see the diagnostic image and the disease we have. CNN has various Dense layers, Dropout, Activation, Flatten,

Convolution2D, MaxPooling2D. After the model has been successfully trained, the software is able to identify the disease if the plant species are in the database. After successful training and previous practice, a comparison of the experimental image with a trained model took place to predict the disease.

8.5 GRAPH PLOTTING

Using matplotlib, I build a graph of Training and Validation accuracy and Training and Validation loss. By default, we can use the matplotlib library in python. The matplotlib library has many packages and functions that produce different types of graphs and sites. And it is very easy to use. It and NumPy and other built-in python functions reach the goal.

8.6 MODEL ACCURACY

Model accuracy is the measure used to determine which model is best at detecting relationships and patterns between data variables based on input, or training, data.

The cost of errors can be huge, but improving the accuracy of the model reduces those costs. Of course, there is a point to lower returns where the cost of making a more accurate model will not lead to a corresponding profit growth, but it is often beneficial to the rest of the board. A better model can add 'invisible' data, better predictions and more insight that can produce, which also brings more business value.

8.7 SAVING MODEL USING PICKLE

Pickle is the standard way of serializing objects in Python. You can use pickle functionality to create learning algorithms for your machine in a different way and save the serial format to a file. You can later upload this file to disable (deserializing) your model and use it to make new predictions.

8.8 PREDICTION

Once the training and testing is completed, the model is ready to classify the image files into the predefined categories. For this the model is first loaded with its entire configuration and then the dataset is given as input to it. Once you choose and fit final machine learning model in scikit-learn, you can use it to make predictions on new data instances.

A class prediction is: given the finalized model and one or more data instances, predict the class for the data instances.

We do not know the outcome classes for the new data. That is why we need the model in the first place. We can predict the class for new data instances using our finalized classification model in scikit-learn using the **predict()** function.

Chapter 9: Conclusion

The results presented in this section are related to training with a whole database that contains both real and augmented images. Since it is known that convolutional networks are able to learn features when trained on large data sets, the results obtained when trained with only real images will not be tested. After adjusting the network parameters, a total accuracy of 96.77% was reached. In addition, a trained model was tested in each class individually. Tests are performed on the whole image from the verification set. As suggested by good performance standards, the results achieved should be compared with other results. In addition, there are no commercial solutions on the market, other than those for the recognition of plant species according to leaf patterns. In this paper, a study of how to use an in-depth study method to automatically detect and detect plant diseases in leaf pictures was explored. The complete process is described, respectively, from image collection used for training and validation to pre-image processing and enlargement and finally the CNN training process in-depth and fine-tuning. Various tests were performed to evaluate the performance of the newly developed model. Since the presented method has not been abused, to our knowledge, in the field of plant disease recognition, there has been no comparison of the corresponding results, using a straightforward process.

9.1 FUTURE SCOPE

In future we can work upon the drone system that can capture images on a large-scale basis and of particular dimension as well. We can take that images for further use. We can pass that image in CNN based trained model and can get proper disease detection. Also, we can work upon getting the proper remedy in future.

I'm also planning to deploy my model for use with an android application via an API. Using Custom REST-API with Django or Flask we can carry out this process and deploy it on higher levels.

Chapter 10: Bibliography

[1] Detection and classification of plant leaf diseases using image processing techniques: a review

Int J Recent Adv Eng Technol, 2 (3) (2014), pp. 2347-2812

ISSN (Online)

[Google Scholar](#)

Sanjay B. Dhaygude, Nitin P. Kumbhar

[2] Agricultural plant leaf disease detection using image processing

Int J Adv Res Electr Electron Instrum Eng, 2 (1) (2013)

[Google Scholar](#)

R. Badnakhe Mrunalini, Prashant R. Deshmukh

[3] An application of K-means clustering and artificial intelligence in pattern recognition for crop diseases

Int Conf Adv Inf Technol, 20 (2011)

2011 IPCSIT

[Google Scholar](#)

S. Arivazhagan, R. Newlin Shebiah, S. Ananthi, S. Vishnu Varthini

[4] Detection of unhealthy region of plant leaves and classification of plant leaf diseases using texture features

Agric Eng Int CIGR, 15 (1) (2013), pp. 211-217

[View Record in Scopus](#)[Google Scholar](#)

Anand H. Kulkarni, R.K. Ashwin Patil

[5] Applying image processing technique to detect plant diseases

Int J Mod Eng Res, 2 (5) (2012), pp. 3661-3664

[View Record in Scopus](#)[Google Scholar](#)

Sabah Bashir, Navdeep Sharma

[6] Remote area plant disease detection using image processing

IOSR J Electron Commun Eng, 2 (6) (2012), pp. 31-34

ISSN: 2278-2834

[CrossRef](#)[View Record in Scopus](#)[Google Scholar](#)

Smita Naikwadi, Niket Amoda

[7] Advances in image processing for detection of plant diseases

Int J Appl Innov Eng Manage, 2 (11) (2013)

[Google Scholar](#)

Sanjay B. Patil, *et al.*

[8] Leaf disease severity measurement using image processing

Int J Eng Technol, 3 (5) (2011), pp. 297-301

[View Record in Scopus](#)[Google Scholar](#)

Piyush Chaudhary, *et al.*

Plagiarism Report



Anand Sharma

to me ▾

22:53 (5 minutes ago)



Document : riya.pdf[D108764173]

About 15% of this document consists of text similar to text found in 54sources. The largest marking is 180 words long and is 28% similar to its primary source.

PLEASE NOTE that the above figures do not automatically mean that there is plagiarism in the document. There may be good reasons as to why parts of a text also appear in other sources. For a reasonable suspicion of academic dishonesty to present itself, the analysis, possibly found sources and the original document need to be examined closely.

Click here to open the analysis:

<https://secure.orkund.com/view/103691943-752591-893361>

Click here to download the document:

<https://secure.ouriginal.com/archive/download/108764173-614193-134087>

--

Dr. ANAND SHARMA

Asst.Prof. (CSE Deptt.)

SET, MUST, Lakshmangarh,

Sikar-332311 (Rajasthan)

Contact no. +91-9649012214

Go Green....

Please don't print this e-mail unless this is absolutely necessary.