

WHEAT LEAF DISEASE RECOGNITION USING TENSORFLOW AND KERAS

A PROJECT REPORT

Submitted by

AAKASH KARTHIKEYAN 211418104001

ABHISHEK SUNDAR R 211418104005

AKILNANDHAN A 211418104014

in partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING



PANIMALAR ENGINEERING COLLEGE

(An Autonomous Institution, Affiliated to Anna University, Chennai)

MAY 2022

BONAFIDE CERTIFICATE

Certified that this project report on **“WHEAT LEAF DISEASE RECOGNITION USING TENSORFLOW AND KERAS”** is the bonafide work of **“AAKASH KARTHIKEYAN [REGISTER NO:211418104001], ABHISHEK SUNDAR R [REGISTER NO:211418104005], AKILNANDHAN A [REGISTER NO:211418104014]”** who carried out the project work under my supervision.

SIGNATURE

**Dr. S. MURUGAVALLI, M.E., Ph.D.,
HEAD OF THE DEPARTMENT**

DEPARTMENT OF CSE,
PANIMALAR ENGINEERING
COLLEGE,
NAZARATHPETTAI,
POONAMALLEE,
CHENNAI-600 123.

SIGNATURE

**Mrs. R. SALINI, M.Tech.,
SUPERVISOR
ASSISTANT PROFESSOR**

DEPARTMENT OF CSE,
PANIMALAR ENGINEERING
COLLEGE,
NAZARATHPETTAI,
POONAMALLEE,
CHENNAI-600 123.

Certified that the above candidate(s) was/ were examined in the Anna University Project Viva-Voce Examination held on.....

INTERNAL EXAMINER

EXTERNAL EXAMINER

DECLARATION BY THE STUDENT

We **AAKASH KARTHIKEYAN [REGISTER NO:211418104001],**
ABHISHEK SUNDAR R [REGISTER NO:211418104005],
AKILNANDHAN A [REGISTER NO:211418104014] hereby declare that
this project report titled **“WHEAT LEAF DISEASE RECOGNITION**
USING TENSORFLOW AND KERAS”, under the guidance of
Mrs. R. SALINI,M.Tech, is the original work done by us and we have
not plagiarized or submitted to any other degree in any university by
us.

ACKNOWLEDGEMENT

We would like to express our deep gratitude to our respected Secretary and Correspondent **Dr.P.CHINNADURAI, M.A., Ph.D.** for his kind words and enthusiastic motivation, which inspired us a lot in completing this project.

We express our sincere thanks to our Directors **Tmt.C.VIJAYARAJESWARI, Dr.C.SAKTHI KUMAR,M.E.,Ph.D** and **Dr.SARANYASREE SAKTHI KUMAR B.E.,M.B.A.,Ph.D.**, for providing us with the necessary facilities to undertake this project.

We also express our gratitude to our Principal **Dr.K.MANI, M.E., Ph.D.** who facilitated us in completing the project.

We thank the Head of the CSE Department, **Dr. S.MURUGAVALLI, M.E.,Ph.D.**, for the support extended throughout the project.

We would like to thank my Project Guide **Mrs.R.SALINI, M.Tech.** and all the faculty members of the Department of CSE for their advice and encouragement for the successful completion of the project.

Lastly, we would also like to take this opportunity to thank our family members, friends and well-wishers who helped in the successful completion of this project.

AAKASH KARTHIKEYAN

ABHISHEK SUNDAR R

AKILNANDHAN A

ABSTRACT

Smart farming system using necessary infrastructure is an innovative technology that helps to improve the quality and quantity of agricultural production in the country. Wheat leaf disease has long been one of the major threats to food security because it dramatically reduces the crop yield and compromises its quality. Accurate and precise diagnosis of diseases has been a significant challenge and it recent advances in computer vision made possible by deep learning has proved the way for camera-assisted disease diagnosis for Wheat leaf. It described the innovative solution that provides efficient disease detection and deep learning with convolutional neural networks (CNN's) has achieved great success in the classification of various Wheat leaf diseases. A variety of neuron-wise and layer-wise visualization methods were applied using a CNN, trained with a publicly available wheat leaf disease given image dataset. So, it observed that neural networks can capture the colors and textures of lesions specific to respective diseases upon diagnosis, which resembles human decision-making.

TABLE OF CONTENTS

CHAPTER NO.	TITLE	PAGE NO.
	ABSTRACT	i
	LIST OF TABLES	vi
	LIST OF FIGURES	vii
	LIST OF SYMBOLS AND ABBREVIATIONS	viii
1	INTRODUCTION	1
	1.1 GENERAL INTRODUCTION	1
	1.2 DOMAIN OVERVIEW	2
	1.2.1 Deep Learning	2
	1.3 OBJECTIVES OF THE PROJECT	8
	1.4 SCOPE OF THE PROJECT	8
2	LITERATURE SURVEY	9
3	SYSTEM ANALYSIS	12
	3.1 EXISTING SYSTEM	12
	3.1.1 DISADVANTAGES OF EXISTING SYSTEM	13
	3.2 PROPOSED SYSTEM	13
	3.2.1 ADVANTAGES OF PROPOSED SYSTEM	13
	3.3 FEASIBILITY STUDY	14
	3.3.1 Introduction	14
	3.3.2 Financial Feasibility	14
	3.3.3 Technical Feasibility	14
	3.3.4 Resource Feasibility	15

	3.4 SOFTWARE REQUIREMENTS	15
	3.5 HARDWARE REQUIREMENTS	15
4	SYSTEM DESIGN	16
	4.1 USE-CASE DIAGRAM	16
	4.2 ACTIVITY DIAGRAM	17
	4.3 SEQUENCE DIAGRAM	18
	4.4 CLASS DIAGRAM	18
	4.5 E.R-DIAGRAM	19
	4.6 DATA FLOW DIAGRAM	19
	4.7 COLLABORATION DIAGRAM	21
5	SYSTEM ARCHITECTURE	22
	5.1 SYSTEM ARCHITECTURE	22
	5.2 LIST OF MODULES	22
	5.2.1 Import the given image from dataset	23
	5.2.2 To train the module by given image dataset	24
	5.2.3 Working process of layers in CNN model	25
	5.3 ALEXNET	28
	5.4 LENET	31
	5.5 WHEAT LEAF CLASSIFICATIONS	33
	5.6 DEPLOY	34
6	SOFTWARE DESCRIPTION	35
	6.1 ANACONDA NAVIGATOR	35
	6.2 JUPYTER NOTEBOOK	37
	6.3 PYCHARM	39

	6.4 PYTHON	41
	6.5 DJANGO	42
	6.6 HTML	43
	6.7 CSS	43
7	SYSTEM IMPLEMENTATION	44
	7.1 MODEL-1	44
	7.2 MODEL-2	48
	7.3 MODEL-3	51
	7.4 MODEL-4	54
8	TESTING	60
	8.1 TESTING	60
	8.2 TESTING TECHNIQUES/ TESTING STRATEGIES	63
	8.3 SOFTWARE TESTING STRATEGIES	64
	8.4 TEST CASE & REPORT	67
9	CONCLUSION	68
	9.1 RESULT AND DISCUSSIONS	68
	9.1.1 RESULT OBTAINED	68
	9.2 CONCLUSION AND FUTURE WORK	72
	9.2.1 CONCLUSION	72
	9.2.2 FUTURE WORK	72
	APPENDICES	73
	A.1 SCREEN SHOTS	73
	REFERENCES	76

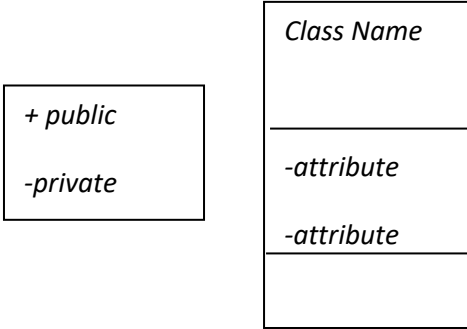
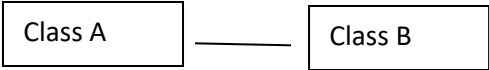
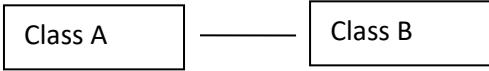
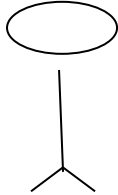
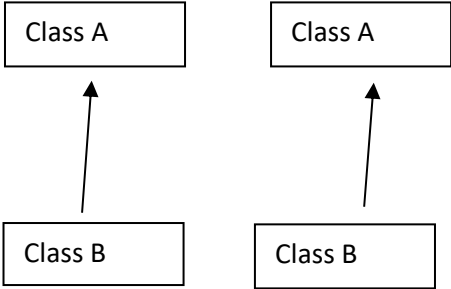
LIST OF TABLES




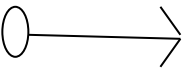
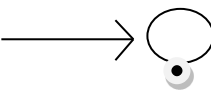
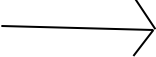
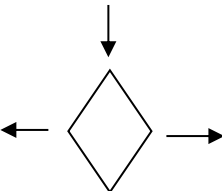
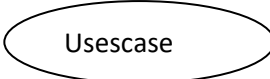
TABLE NO	TITLE	PAGE NO
8.1	Test case	67

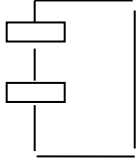
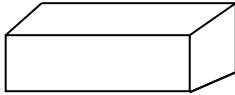
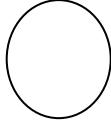

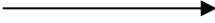

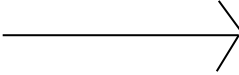
LIST OF FIGURES

FIGURE NO	NAME OF THE FIGURE	PAGE NO.
1.1	Subset of Artificial Intelligence	2
1.2	Flow of deep learning Algorithm	3
1.3	Convolutional Neural Network	6
1.4	Types of Convolutional Neural Network	7
4.1	Use-case Diagram	16
4.2	Activity Diagram	17
4.3	Sequence Diagram	18
4.4	Class Diagram	18
4.5	E.R-Diagram	19
4.6	Process of dataflow diagram	19
4.7	Data flow Diagram Level 0	20
4.8	Data flow Diagram Level 1	20
4.9	Data flow Diagram Level 2	20
4.10	Data flow Diagram Level 3	21
4.11	Collaboration Diagram	21
5.1	Architecture Diagram	22
5.2	Healthy Leaf dataset	23
5.3	Septoria Leaf dataset	23
5.4	Stripe Rust Leaf dataset	24
5.5	Training of ManualNet model	24
5.6	The Accuracy and loss curves for CNN	25
5.7	Layers of AlexNet	28
5.8	Architecture of AlexNet	29
5.9	The Accuracy and loss curves for AlexNet	30
5.10	Layers of Lenet	31
5.11	Architecture of LeNet	32
5.12	Training Lenet	33
6.1	Anaconda Navigator Screenshots	36
9.1	Dataset collected for healthy wheat leaf	68
9.2	Dataset collected for Septoria wheat leaf	69
9.3	Dataset collected for Stripe rust wheat leaf	69
9.4	Training of LeNet model	70
9.5	The training and validation loss graph(LeNet)	70
9.6	The training and validation Accuracy graph (LeNet)	70
A1.1	Home Page	65
A1.2	Prediction of healthy	66
A1.3	Prediction of septoria	67
A1.4	Prediction of stripe rust	68

LIST OF SYMBOLS AND ABBREVIATIONS

S.NO	NOTATION NAME	NOTATION	DESCRIPTION
1.	Class		Represents a collection of similar entities grouped together.
2.	Association	<p style="text-align: center;">NAME</p>  	Associations represent static relationships between classes. Roles represents the way the two classes see each other.
3.	Actor		It aggregates several classes into a single classes.
4.	Aggregation		Interaction between the system and external environment

5.	<i>Relation</i> (uses)	uses	Used for additional process communication.
6.	Relation (extends)	EXTENDS 	Extends relationship is used when one use case is similar to another use case but does a bit more.
7.	Communication		Communication between various use cases.
8.	State		State of the process.
9.	Initial State		Initial state of the object
10.	Final state		Final state of the object
11.	Control flow		Represents various control flow between the states.
12.	Decision box		Represents decision making process from a constraint
13.	Usecase		Interact ion between the system and external environment.

14.	Component		Represents physical modules which is a collection of components.
15.	Node		Represents physical modules which are a collection of components.
16.	Data Process/State		A circle in DFD represents a state or process which has been triggered due to some event or action.
17.	External entity		Represents external entities such as keyboard,sensors,etc.
18.	Transition		Represents communication that occurs between processes.
19.	Object Lifeline		Represents the vertical dimensions that the object communications.
20.	Message	Message 	Represents the message exchanged.

CHAPTER – 1

INTRODUCTION

1.1 GENERAL INTRODUCTION

In India, wheat is the second most vital food grain cultivated after rice. In many developing countries, the per hectare yield of grains is less in comparison to the developed countries. Further, this disparity can be credited to the usage of advanced tools and techniques in fields. At present, the fourth industrial revolution is disrupting everything. Artificial intelligence, machine learning, IoT, and edge computing are playing a vital role in revolutionizing the agriculture sector. Precision agriculture is aided by these technologies, resulting in reduced resource wastage and increased profits. The appropriate diagnosis of wheat diseases and timely remedial action can save the grains from wastage. Moreover, it ensures good quality of wheat yield that eventually gives maximum profit to farmers. Deep learning can solve the problems of image classification with acceptable accuracy.

Types of Disease: There are varieties of disease spots which tend to resemble each other and can easily be confused with one another by inexperienced people.

1) **Septoria:** The initial symptoms are yellowish or chlorotic flecks on leaves, especially those in contact with the soil. These flecks enlarge into irregular lesions, brown-to-reddish brown in color.

2) **Stripe Rust:** The first sign of stripe rust is the appearance of yellow streaks (pre-pustules), followed by small, bright yellow, elongated uredial pustules arranged in conspicuous rows on the leaves, leaf sheaths, glumes and awns.

1.2 DOMAIN OVERVIEW

1.2.1 Deep Learning

Deep learning is a branch of machine learning which is completely based on artificial neural networks, as neural network is going to mimic the human brain so deep learning is also a kind of mimic of human brain. It's on hype nowadays because earlier we did not have that much processing power and a lot of data. A formal definition of deep learning is- neurons Deep learning is a particular kind of machine learning that achieves great power and flexibility by learning to represent the world as a nested hierarchy of concepts, with each concept defined in relation to simpler concepts, and more abstract representations computed in terms of less abstract ones. In brain approximately 100 billion neurons all together this is a picture of an individual neuron and each neuron is connected through thousands of their neighbors. The question here is how it recreates these neurons in a computer. So, it creates an artificial structure called an artificial neural net where we have nodes or neurons. It has some neurons for input value and some for output value and in between, there may be lots of neurons interconnected in the hidden layer.

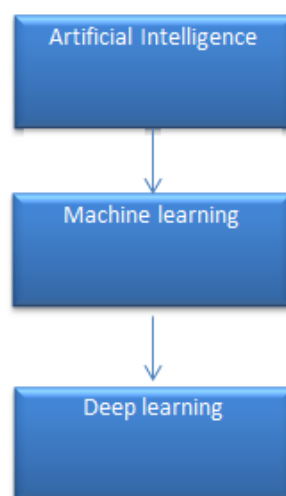


Figure 1.1 Subset of Artificial Intelligence

It need to identify the actual problem in order to get the right solution and it should be understood, the feasibility of the Deep Learning should also be checked (whether it should fit Deep Learning or not). It needs to identify the relevant data which should correspond to the actual problem and should be prepared accordingly. Choose the Deep Learning Algorithm appropriately. Algorithm should be used while training the dataset. Final testing should be done on the dataset.

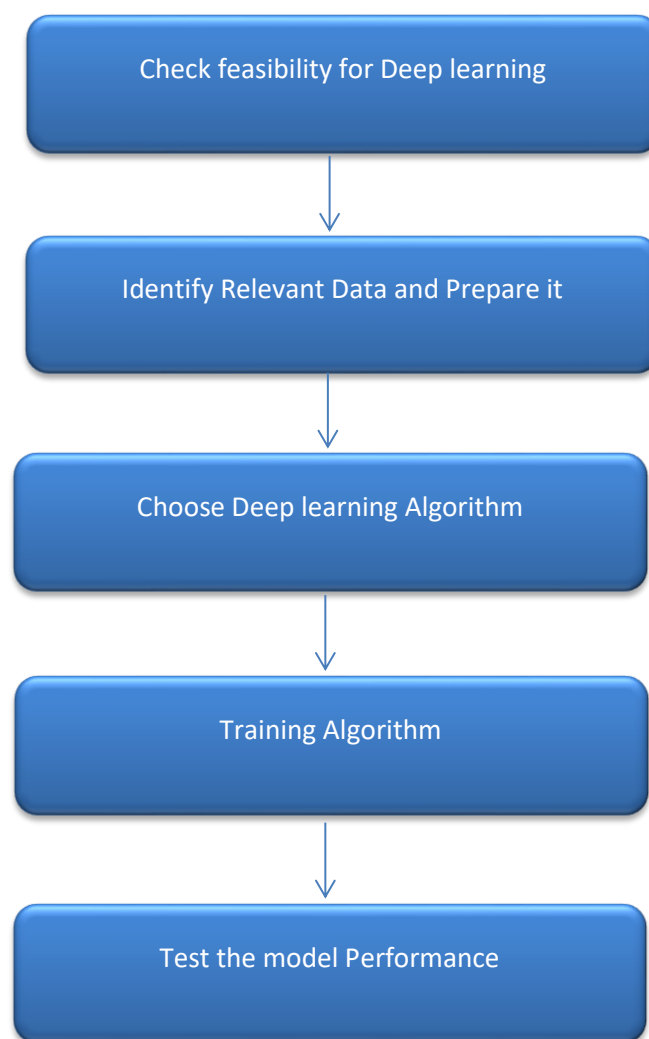


Figure 1.2 Flow of deep learning Algorithm

Deep learning (also known as deep structured learning) is part of a broader family of machine learning methods based on artificial neural networks with

representation learning. Learning can be supervised, semi-supervised or unsupervised.

Deep-learning architectures such as deep neural networks, deep belief networks, deep reinforcement learning, recurrent neural networks and convolutional neural networks have been applied to fields including computer vision, speech recognition, natural language processing, machine translation, bioinformatics, drug design, medical image analysis, material inspection and board game programs, where they have produced results comparable to and in some cases surpassing human expert performance.

Artificial neural networks (ANNs) were inspired by information processing and distributed communication nodes in biological systems. ANNs have various differences from biological brains. Specifically, neural networks tend to be static and symbolic, while the biological brain of most living organisms is dynamic (plastic) and analogue.

The adjective "deep" in deep learning refers to the use of multiple layers in the network. Early work showed that a linear perceptron cannot be a universal classifier, but that a network with a non-polynomial activation function with one hidden layer of unbounded width can. Deep learning is a modern variation which is concerned with an unbounded number of layers of bounded size, which permits practical application and optimized implementation, while retaining theoretical universality under mild conditions. In deep learning the layers are also permitted to be heterogeneous and to deviate widely from biologically informed connectionist models, for the sake of efficiency, trainability and understandability, whence the "structured" part.

Deep learning is a class of machine learning algorithms that uses multiple layers to progressively extract higher-level features from the raw input. For example,

in image processing, lower layers may identify edges, while higher layers may identify the concepts relevant to a human such as digits or letters or faces.

Interpretations:

Deep neural networks are generally interpreted in terms of the universal approximation theorem or probabilistic inference.

The classic universal approximation theorem concerns the capacity of feed-forward neural networks with a single hidden layer of finite size to approximate continuous functions. In 1989, the first proof was published by George Cybenko for sigmoid activation functions and was generalised to feed-forward multi-layer architectures in 1991 by Kurt Hornik. Recent work also showed that universal approximation also holds for non-bounded activation functions such as the rectified linear unit.

The universal approximation theorem for deep neural networks concerns the capacity of networks with bounded width but the depth is allowed to grow. It was proved that if the width of a deep neural network with ReLU activation is strictly larger than the input dimension, then the network can approximate any Lebesgue integrable function; If the width is smaller or equal to the input dimension, then deep neural network is not a universal approximator.

The probabilistic interpretation derives from the field of machine learning. It features inference, as well as the optimization concepts of training and testing, related to fitting and generalization, respectively. More specifically, the probabilistic interpretation considers the activation nonlinearity as a cumulative distribution function. The probabilistic interpretation led to the introduction of dropout as regularizer in neural networks. The probabilistic interpretation was introduced by researchers including Hopfield, Widrow and Narendra and popularized in surveys such as the one by Bishop.

CONVOLUTIONAL NEURAL NETWORKS (CNN)

CNN is a multi-layered neural network with a unique architecture designed to extract increasingly complex features of the data at each layer to determine the output. CNN's are well suited for perceptual tasks.

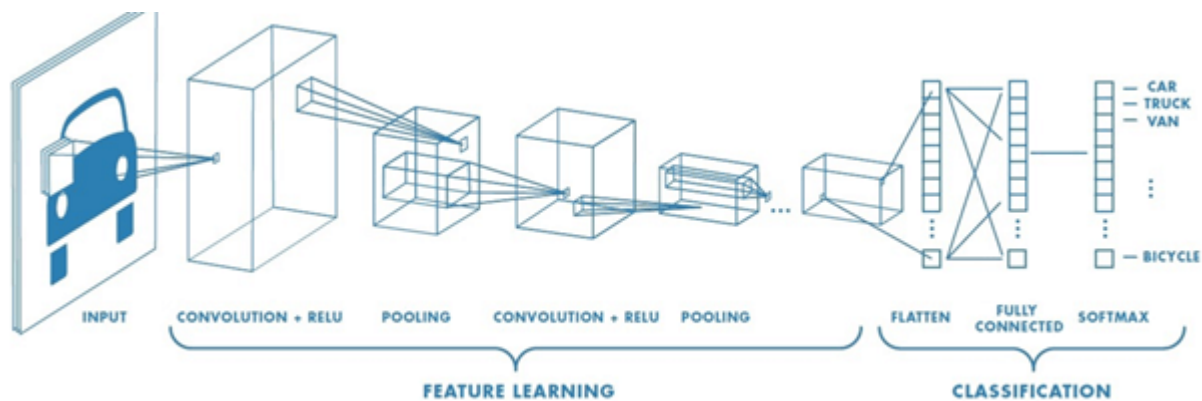


Figure 1.3 Convolutional Neural Network

CNN is mostly used when there is an unstructured data set (e.g., images) and the practitioners need to extract information from it

For instance, if the task is to predict an image caption:

- The CNN receives an image of let's say a cat, this image, in computer term, is a collection of the pixel. Generally, one layer for the greyscale picture and three layers for a color picture.
- During the feature learning (i.e., hidden layers), the network will identify unique features, for instance, the tail of the cat, the ear, etc.
- When the network thoroughly learned how to recognize a picture, it can provide a probability for each image it knows. The label with the highest probability will become the prediction of the network.

A Convolutional Neural Network (CNN, or ConvNet) are a special kind of multi-layer neural networks, designed to recognize visual patterns directly from

pixel images with minimal preprocessing. The ImageNet project is a large visual database designed for use in visual object recognition software research. The ImageNet project runs an annual software contest, the ImageNet Large Scale Visual Recognition Challenge (ILSVRC), where software programs compete to correctly classify and detect objects and scenes. Here I will talk about CNN architectures of ILSVRC top competitors.

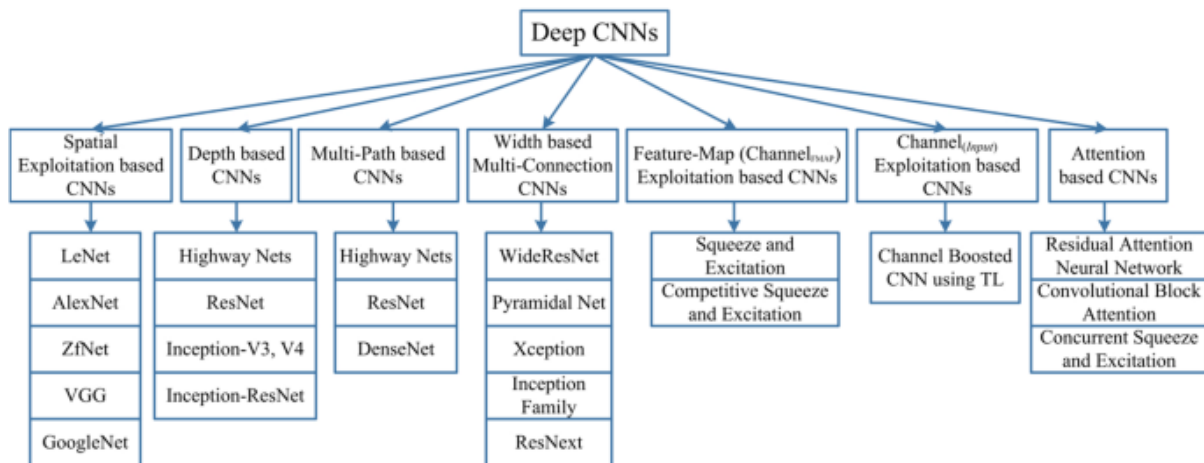


Figure 1.4 Types of Convolutional Neural Network

1.3 OBJECTIVES OF THE PROJECT

The goal is to develop a deep learning model for wheat leaf image classification by convolutional neural network algorithm for potentially classifying the results in the form of best accuracy by comparing the CNN architectures.

1.4 SCOPE OF THE PROJECT

Most of the related works classifies the disease with Three classifiers, i.e. Healthy wheat leaf and disease two diseases like Septoria, Stripe_rust. Whether. The scope in our research is to split this classes into defective and non-defective stages of wheat leaf. It can be applicable to all Farmers.

CHAPTER – 2

LITERATURE SURVEY

2.1 Yellow Rust Extraction in Wheat Crop based on Color Segmentation Techniques

Author: Amina Khatra

Year: 2013

The presented work presents a color based segmentation techniques for extraction of yellow rust in wheat crop images. Accurate segmentation of yellow rust in wheat crop images is very part of assessment of disease penetration into the wheat crop. And in turn to take the necessary preventive action for minimizing the crop damage. The jpeg images acquired from CCD camera are read into the matlab tool and a color based segmentation algorithm is performed to segment the yellow rust. The segmentation of color is performed base on k-means algorithm.

2.2 Comparative study of Leaf Disease Diagnosis system using Texture features and Deep Learning Features

Author: Ashwini T Sapka, Uday V Kulkarni

Year: 2018

The feature extraction technique plays a very critical and crucial role in automatic leaf disease diagnosis system. Many different feature extraction techniques are used by the researchers for leaf disease diagnosis which includes colour, shape, texture, HOG, SURF and SIFT features. Recently Deep Learning is giving very promising results in the field of computer vision. In this manuscript, two feature extraction techniques are discussed and compared. In first approach, the Gray Level Covariance Matrix(GLCM) is used which extracts 12 texture features for diagnosis purpose. In second

approach, the pretrained deep learning model, Alexnet is used for feature extraction purpose. There are 1000 features extracted automatically with the help of this pretrained model. Here Backpropagation neural network (BPNN) is used for the classification purpose. It is observed that the deep learning features are more dominant as compared to the texture features. It gives 93.85% accuracy which is much better than the texture feature extraction technique used here.

2.3 Various Plant Diseases Detection Using Image Processing Methods

Author: Simranjeet Kaur, Geetanjali Babbar, Navneet Sandhu, Dr.Gagan Jindal

Year: 2019

Identification of plant leaf diseases is the preventive measure for the loss happened in the yield and the overall agriculture crop quantity. Basically, the studies of the plant diseases are defined by visualizing and observing patterns observed and engraved on the leaves. So, the disease detection of any plant prior to any hazardous impact becomes very crucial factor for viable agriculture. However, it is so difficult to detect, monitor and derive conclusions from the plant leaf diseases manually because, the costs emerging in the process demands huge amount of work done, energy, expertize and last but not least the processing time. Therefore, image processing concepts comes handy and are used for disease detections. The detection process includes the phases such as, image acquisition, segmentation, image pre-processing, feature extraction from segments and then classification based on the results. This paper discusses the elementary methods that are being used for the plant disease detection based on the leaf images.

2.4 Android Application of Wheat Leaf Disease Detection and Prevention using Machine Learning

Author: Sumit Nema, Bharat Mishra and Mamta Lambert

Year: 2020

Crop quality and production plays an important role in agriculture and farmer's life. Farmer's income highly depends on crop quantity and quality in India. Wheat is the main crop in India. Wheat leaves diseases majorly affect the production rate as well as farmer's profits. An android application has designed to detect the wheat plant leaf diseases in this work. Machine learning methods are easily applied and capable to quick recognizes these diseases. Simulation results show the effectiveness of the proposed method. Real time experiment in the wheat field nearby area of Madhya Pradesh also validates the results.

2.5 Wheat Disease Detection Using SVM Classifier

Author: Er.Varinderjit Kaur , Dr.Ashish Oberoi

Year: 2018

There are many types of diseases which are present in plants. To detect these diseases pattern are required to recognize them. There are many types of pattern recognition algorithm which gives detection of disease with accuracy. Image processing Techniques for Wheat Disease Detection most important research areas in computer science for last few decades. Based on literature review ,We conclude that the eng. and research community is doing lot of work on Wheat disease detection, but the app. of this techniques to solve practical agricultural This paper presents a survey on SVM Classifier method that use digital image processing techniques to detect and classify plant diseases from digital images in the visible spectrum.

CHAPTER – 3

SYSTEM ANALYSIS

3.1 EXISTING SYSTEM

It reviews and summaries various techniques used for classifying and detecting various bacterial, fungal and viral wheat leaf diseases. The classification techniques helps in automating the detection of wheat leaf diseases and categorizing them centered on their morphological features. It focuses on identifying the wheat leaf diseases with CNN as classifier. It is also intended to focus on increasing the recognition rate and classification accuracy of severity of leaf diseases by using hybrid algorithms.

Wheat's are considered to be important as they are the source of energy supply to mankind. plant diseases can affect the wheat leaf any time between sowing and harvesting which leads to huge loss on the production of crop and economical value of market. Therefore, wheat disease detection plays a vital role in agricultural field. However, it requires huge manpower, more processing time and extensive knowledge about wheat diseases. Hence, machine learning is applied to detect diseases in wheat leaves as it analyzes the data from different aspects, and classifies it into one of the predefined set of classes. The morphological features and properties like color, intensity and dimensions of the plant leaves are taken into consideration for classification. It presents an overview on various types of wheat diseases and different classification techniques in machine learning that are used for identifying diseases in different wheat leaves.

3.1.1 DISADVANTAGES OF EXISTING SYSTEM

- It has not focused on identifying other wheat leaf diseases with CNN as classifier.
- It has not focused on increasing the recognition rate and classification accuracy of severity of leaf diseases.

3.2 PROPOSED SYSTEM

To classify the wheat leaf diseases. We planned to design deep learning technique so that a person with lesser expertise in software should also be able to use it easily. It proposed system to predicting wheat leaf diseases. It explains about the experimental analysis of our methodology. Different number of images is collected for each disease that was classified into database images and input images. The primary attributes of the image are relied upon the shape and texture oriented features. The sample screenshots displays the wheat leaf disease detection using color based segmentation model.

3.2.1 ADVANTANGES OF PROPOSED SYSTEM

- Increasing throughput & reducing subjectiveness arising from human experts in detecting the wheat leaf diseases.
- It is essential to detect a particular disease. In our country many farmers are not so educated to get correct information about all diseases.

3.3 FEASIBILITY STUDY

3.3.1 Introduction

A Feasibility Study includes a heart analysis of the necessity, value, and customary sense of a planned enterprise, like framework development. The procedure of outlining associated capital punishment record-keeping frameworks has adequate responsibility and plus suggestions for an association. the chance study can alter you to choose education and simple selection at imperative focuses throughout the biological process procedure to determine if it's operationally, economically, and indeed cheap to deliver with a selected strategy.

3.3.2 Financial Feasibility

The resources required by the proposed system are available easily in the internet for free. This procedure is to determine the benefits and savings that is expected from a system and compare with cost. Otherwise further justification or alterations in a proposed system that has to be made if it is having a change of being approved. This is an ongoing effort that improves inaccuracy of each phase of the system lifecycle. From these it's clear that this project is financially feasible.

3.3.3 Technical feasibility

This project is desktop-based application. The main technologies that are associated with project are

- Deep Learning Model-Lenet, AlexNet
- Django for creating web UI
- Python for libraries
- Pycharm

- Anaconda Navigator

Each of the technologies are freely on the internet and also the technical skills needed are manageable. This requires less information measure to transfer knowledge that holds on in information. From this, it's clear that our project is technically feasible.

3.3.4 Resource Feasibility

Relates to whether the participants will be able to handle the new system. Moreover, 3 - 5 is enough for this proposed system.

Resources that are required for our project are:

- Programming Devices which adhere hard requirements
- Programming Tools such as Deep Learning Framework –Lenet, Alexnet(CNN model), Pycharm ,Python, Anaconda Navigator
- Programming Individuals who have required technical knowledge.

So, it's clear that project has required resource feasibility.

3.4 SOFTWARE REQUIREMENTS

Operating System : Windows 10

Simulation Tool : Anaconda with Jupyter Notebook and Pycharm

3.5 HARDWARE REQUIREMENTS

Processor : Pentium IV/III

Hard disk : minimum 80 GB

RAM : minimum 2 GB

CHAPTER – 4

SYSTEM DESIGN

4.1 USE CASE DIAGRAM

Use case diagrams give a graphic overview of the actors involved in a system, different functions needed by those actors and how these different functions interact.

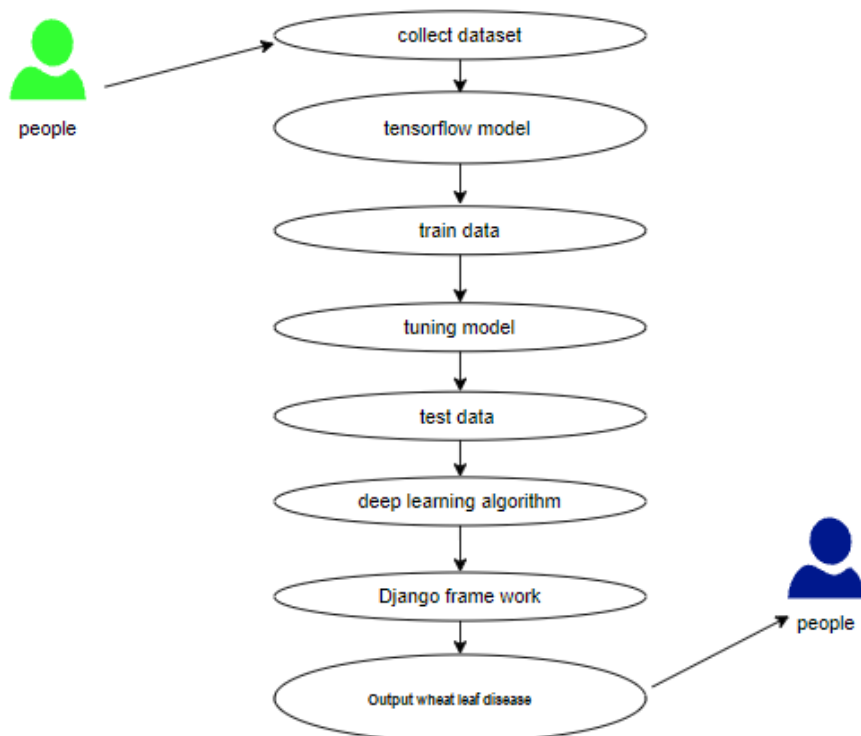


Figure 4.1 Use-case Diagram

4.2 ACTIVITY DIAGRAM

Activity is a particular operation of the system. Activity diagrams are not only used for visualizing dynamic nature of a system but they are also used to construct the executable system by using forward and reverse engineering techniques. The only missing thing in activity diagram is the message part. It does not show any message flow from one activity to another. Activity diagrams some time considered as the flow chart. Although the diagrams looks like a flow chart but it is not. It shows different flow like parallel, branched, concurrent and single.

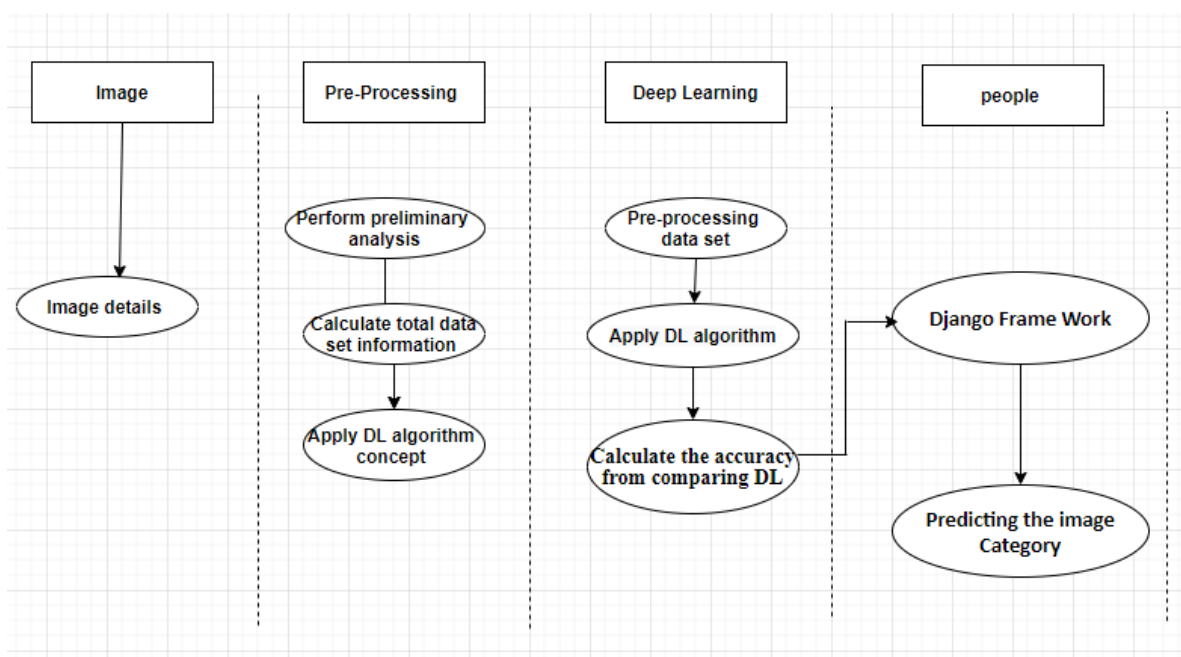


Figure 4.2 Activity Diagram

4.3 SEQUENCE DIAGRAM

Sequence diagrams in UML show how objects interact with each other and the order those interactions occur. It's important to note that they show the interactions for a particular scenario. The processes are represented vertically and interactions are shown as arrows.

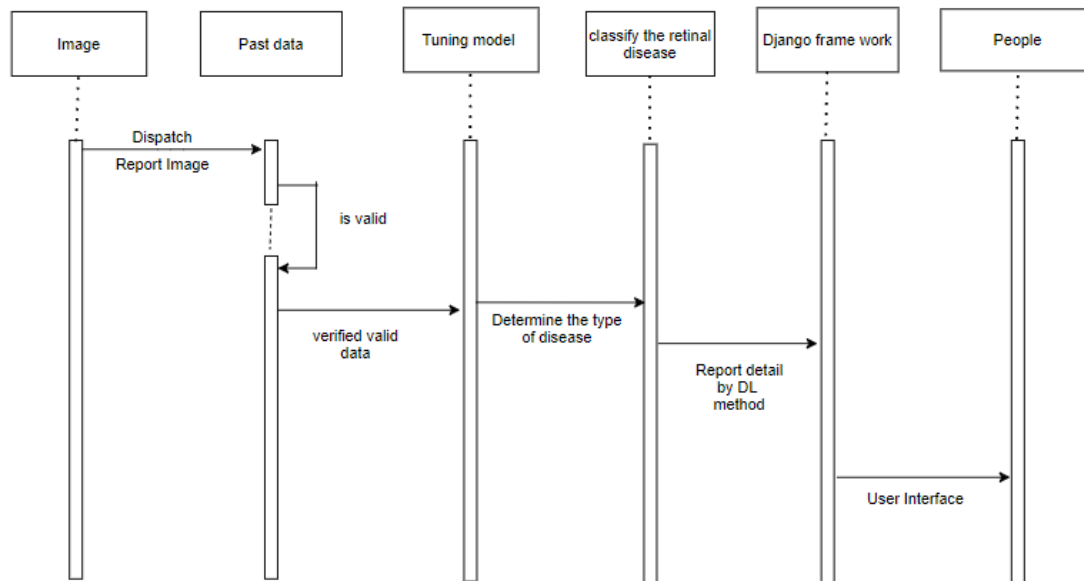


Figure 4.3 Sequence Diagram

4.4 CLASS DIAGRAM

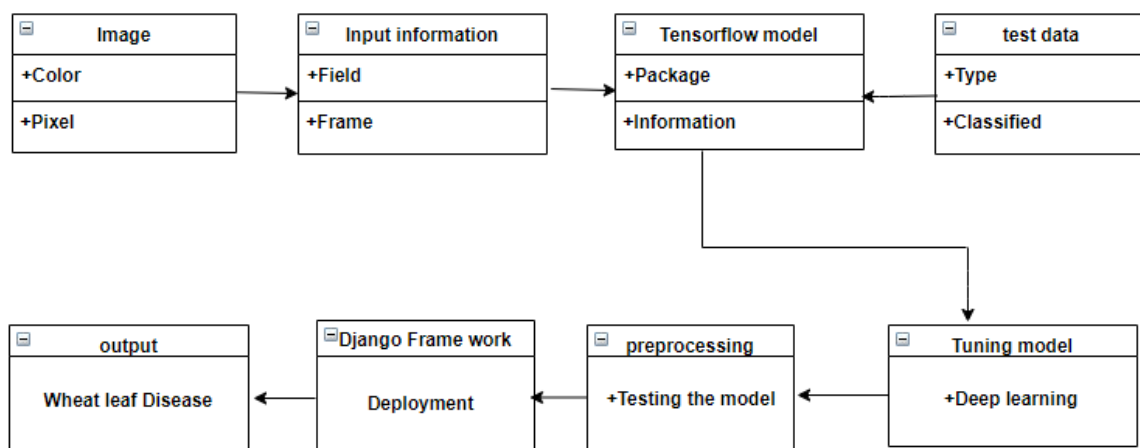


Figure 4.4 Class Diagram

4.5 E.R – DIAGRAM

E-R model is a popular high level conceptual data model. This model and its variations are frequently use for the conceptual design of database application and many database design tools employ its concept. A database that confirms to an E-R diagram can be represented by a collection of tables in the relational system.

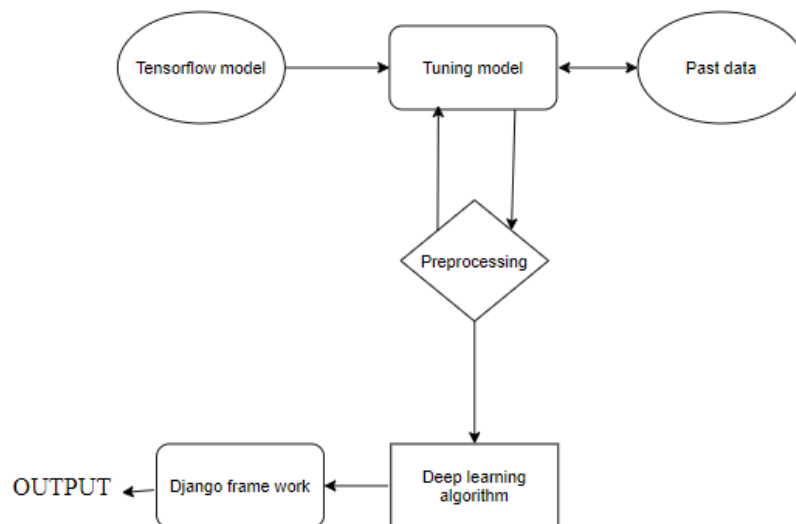


Figure 4.5 E.R-Diagram

4.6 DATA FLOW DIAGRAM

A data flow diagram (DFD) is a graphical representation of the “flow” of data through an information system, modeling its process aspects. A DFD is often used as a preliminary step to create an overview of the system without going into great detail, which can later be elaborated.

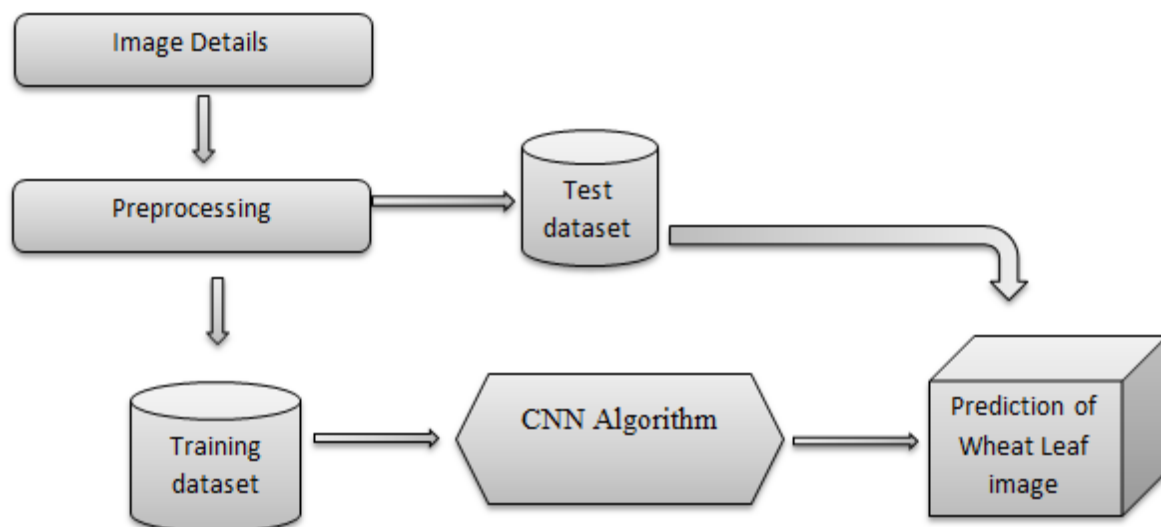


Figure 4.6 Process of dataflow diagram

DFD Level 0 is also called a Context Diagram. It’s a basic overview of the whole system or process being analyzed or modeled. It’s designed to be an at-a-

glance view, showing the system as a single high-level process, with its relationship to external entities. It should be easily understood by a wide audience, including stakeholders, business analysts, and developers.

Data flow Diagram Level 0

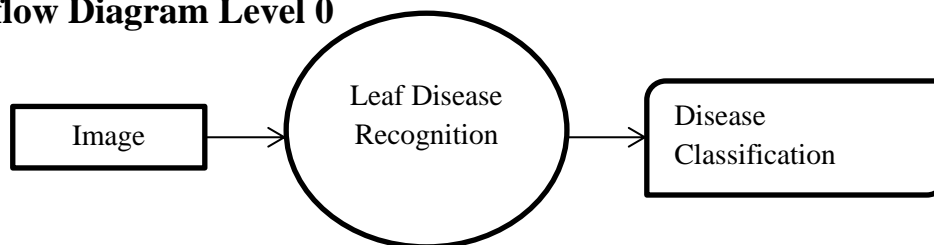


Figure 4.7 Data flow Diagram Level 0

Data flow Diagram Level 1

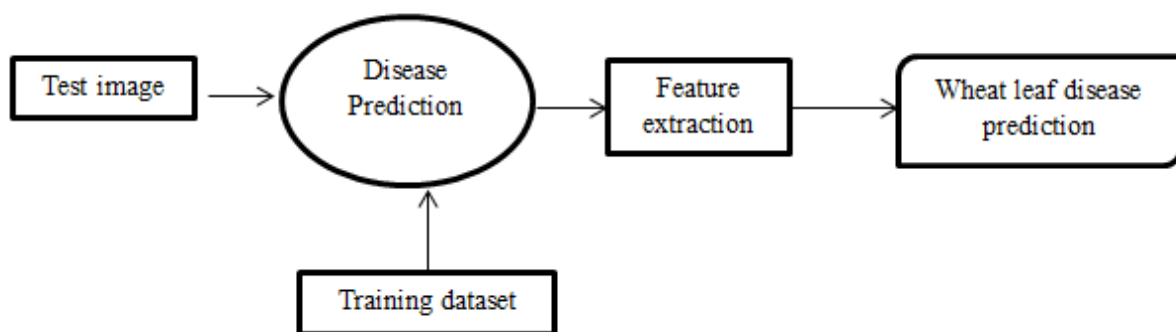


Figure 4.8 Data flow Diagram Level 1

Data flow Diagram Level 2

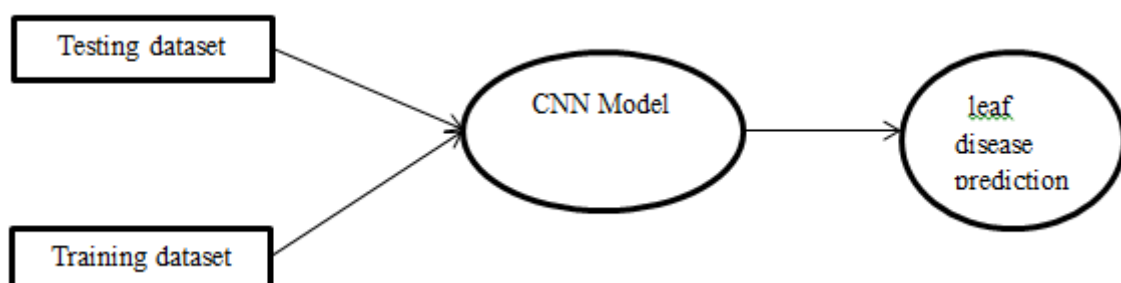


Figure 4.9 Data flow Diagram Level 2

Data flow Diagram Level 3

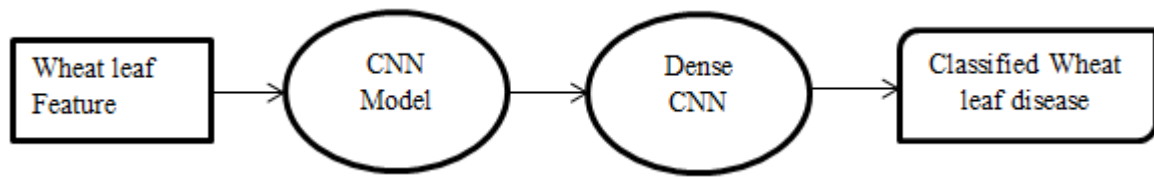


Figure 4.10 Data flow Diagram Level 3

4.7 COLLABORATION DIAGRAM

A collaboration diagram, also known as a communication diagram, is an illustration of the relationships and interactions among software objects in the Unified Modeling Language (UML). These diagrams can be used to portray the dynamic behavior of a particular use case and define the role of each object.

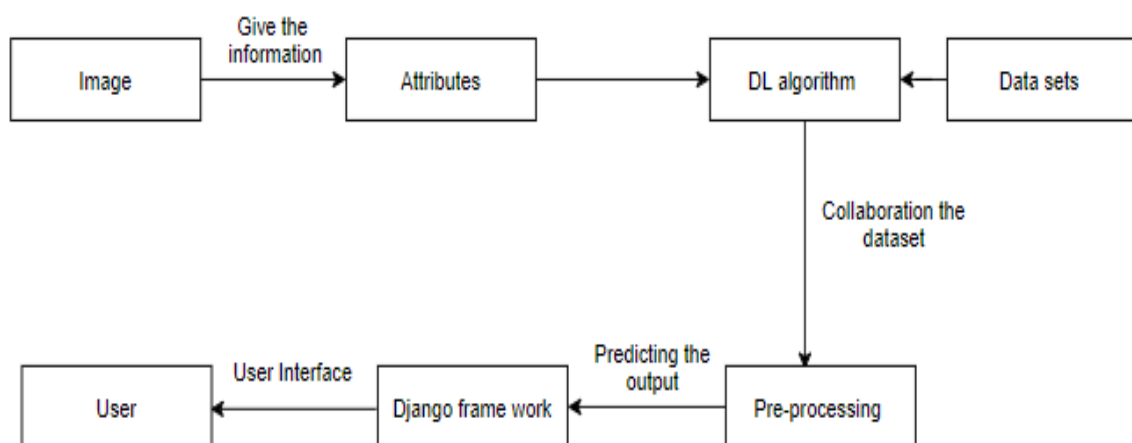


Figure 4.11 Collaboration Diagram

CHAPTER – 5

SYSTEM ARCHITECTURE

5.1 SYSTEM ARCHITECTURE

The Figure 5.1 is nothing but a simple description of all the entities that have been incorporated into the system. The diagram represents the relations between each of them and involves a sequence of decision-making processes and steps. You can simply call it a visual or the whole process and its implementation. All functional correspondences are explained in this diagram.

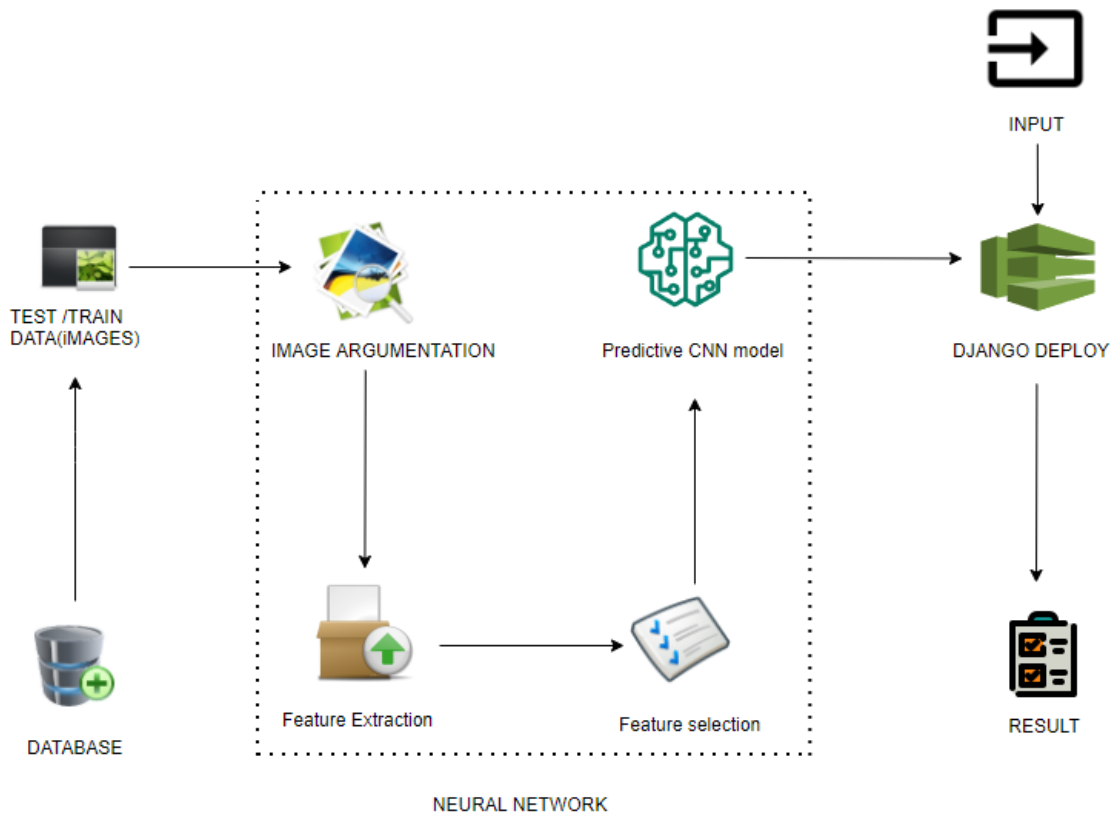


Figure 5.1 Architecture Diagram

5.2 LIST OF MODULES

- Import the given image from dataset
- To train the module by given image dataset

- Working process of layers in CNN model
- Classification identification
- Deploy

5.2.1 IMPORT THE GIVEN IMAGE FROM DATASET

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

Healthy:

Trained data for Healthy:

```
===== images in: dataset/train/Healthy
images_count      84
min_width         179
max_width         2988
min_height        168
max_height        2988
```



Figure 5.2 Healthy Leaf dataset

Septoria:

Trained data for septoria:

```
===== images in: dataset/train/septoria
images_count      97
min_width         1267
max_width         5999
min_height        2770
max_height        5999
```

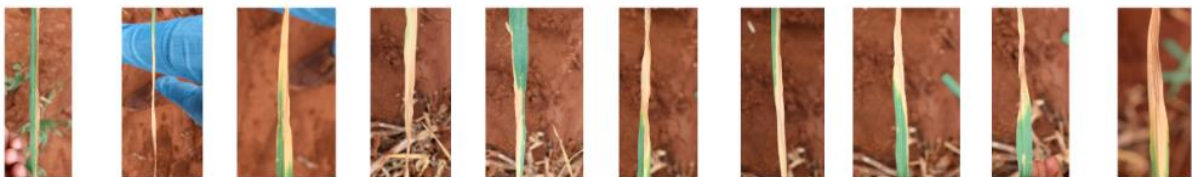


Figure 5.3 Septoria Leaf dataset

Stripe_rust:

Trained data for Stripe Rust:

```
===== images in: dataset/train/stripe_rust
images_count      102
min_width         1748
max_width         6000
min_height        3203
max_height        5999
```

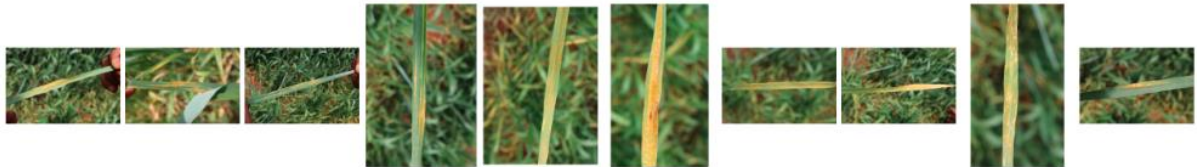


Figure 5.4 Stripe Rust Leaf dataset

5.2.2 TO TRAIN THE MODULE BY GIVEN IMAGE DATASET

To train our dataset using classifier and fit generator function also we make training steps per epoch's then total number of epochs, validation data and validation steps using this data we can train our dataset.

```
Epoch 1/10
8/8 [=====] - 92s 11s/step - loss: 40.6297 - accuracy: 0.3944 - val_loss: 11.8876 - val_accuracy: 0.62
50
Epoch 2/10
8/8 [=====] - 85s 11s/step - loss: 4.7086 - accuracy: 0.6641 - val_loss: 1.1567 - val_accuracy: 0.8125
Epoch 3/10
8/8 [=====] - 82s 11s/step - loss: 0.2997 - accuracy: 0.9203 - val_loss: 1.7294 - val_accuracy: 0.7500
Epoch 4/10
8/8 [=====] - 87s 11s/step - loss: 2.4066 - accuracy: 0.7410 - val_loss: 1.0218 - val_accuracy: 0.8125
Epoch 5/10
8/8 [=====] - 81s 10s/step - loss: 1.1518 - accuracy: 0.8327 - val_loss: 0.2818 - val_accuracy: 0.9062
Epoch 6/10
8/8 [=====] - 77s 10s/step - loss: 0.6843 - accuracy: 0.8486 - val_loss: 3.3290 - val_accuracy: 0.6719
Epoch 7/10
8/8 [=====] - 78s 10s/step - loss: 0.8407 - accuracy: 0.8247 - val_loss: 0.6063 - val_accuracy: 0.8750
Epoch 8/10
8/8 [=====] - 77s 10s/step - loss: 0.4545 - accuracy: 0.8884 - val_loss: 2.9757 - val_accuracy: 0.6875
Epoch 9/10
8/8 [=====] - 79s 10s/step - loss: 0.7911 - accuracy: 0.8725 - val_loss: 0.0752 - val_accuracy: 0.9844
Epoch 10/10
8/8 [=====] - 76s 10s/step - loss: 0.1244 - accuracy: 0.9402 - val_loss: 0.0776 - val_accuracy: 0.9688
```

Figure 5.5 Training of ManualNet model

5.2.3 WORKING PROCESS OF LAYERS IN CNN MODEL

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

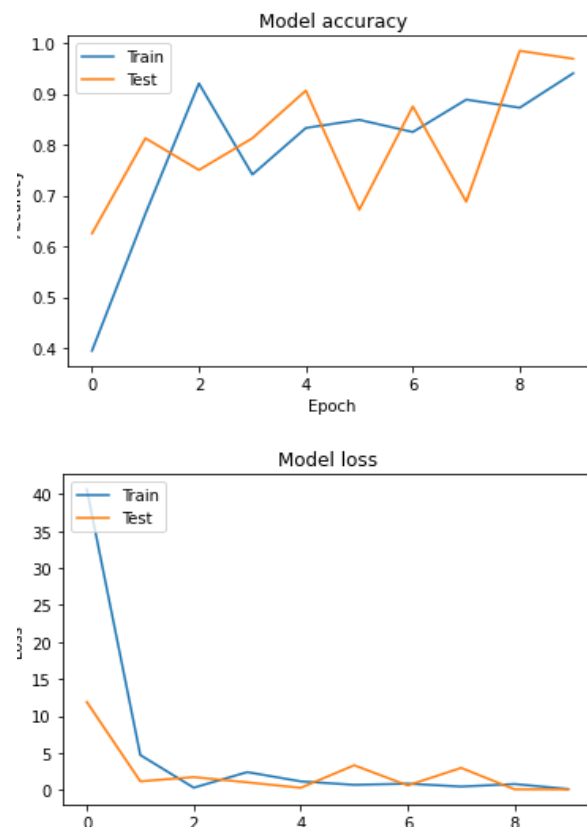


Figure 5.6 The Accuracy and loss curves for CNN

The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics. The architecture of a ConvNet is analogous to that of the connectivity pattern of Neurons in the Human Brain and was inspired by the organization of the Visual Cortex. Individual neurons respond to stimuli only in a restricted region of the visual field known as the Receptive Field. Their network consists of four layers with 1,024 input units, 256 units in the first hidden layer, eight units in the second hidden layer, and two output units.

Input Layer:

Input layer in CNN contain image data. Image data is represented by three dimensional matrixes. It needs to reshape it into a single column. Suppose you have image of dimension $28 \times 28 = 784$, it need to convert it into 784×1 before feeding into input.

Convo Layer:

Convo layer is sometimes called feature extractor layer because features of the image are get extracted within this layer. First of all, a part of image is connected to Convo layer to perform convolution operation as we saw earlier and calculating the dot product between receptive field (it is a local region of the input image that has the same size as that of filter) and the filter. Result of the operation is single integer of the output volume. Then the filter over the next receptive field of the same input image by a Stride and do the same operation again. It will repeat the same process again and again until it goes through the whole image. The output will be the input for the next layer.

Pooling Layer:

Pooling layer is used to reduce the spatial volume of input image after convolution. It is used between two convolution layers. If it applies FC after Convo layer without applying pooling or max pooling, then it will be computationally expensive. So, the max pooling is only way to reduce the spatial volume of input image. It has applied max pooling in single depth slice with Stride of 2. It can observe the 4×4 dimension input is reducing to 2×2 dimensions.

Fully Connected Layer (FC):

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

Softmax / Logistic Layer:

Softmax or Logistic layer is the last layer of CNN. It resides at the end of FC layer. Logistic is used for binary classification and softmax is for multi-classification.

Output Layer:

Output layer contains the label which is in the form of one-hot encoded. Now you have a good understanding of CNN.

5.3 ALEXNET

AlexNet is the name of a convolutional neural network which has had a large impact on the field of machine learning, specifically in the application of deep learning to machine vision. AlexNet was the first convolutional network which used GPU to boost performance.

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 60, 60, 96)	34944
activation (Activation)	(None, 60, 60, 96)	0
max_pooling2d (MaxPooling2D)	(None, 30, 30, 96)	0
conv2d_1 (Conv2D)	(None, 20, 20, 256)	2973952
activation_1 (Activation)	(None, 20, 20, 256)	0
max_pooling2d_1 (MaxPooling2D)	(None, 10, 10, 256)	0
conv2d_2 (Conv2D)	(None, 8, 8, 384)	885120
activation_2 (Activation)	(None, 8, 8, 384)	0
conv2d_3 (Conv2D)	(None, 6, 6, 384)	1327488
activation_3 (Activation)	(None, 6, 6, 384)	0
conv2d_4 (Conv2D)	(None, 4, 4, 256)	884992
activation_4 (Activation)	(None, 4, 4, 256)	0
max_pooling2d_2 (MaxPooling2D)	(None, 2, 2, 256)	0
flatten (Flatten)	(None, 1024)	0
dense (Dense)	(None, 4096)	4198400
activation_5 (Activation)	(None, 4096)	0
dropout (Dropout)	(None, 4096)	0
dense_1 (Dense)	(None, 4096)	16781312
activation_6 (Activation)	(None, 4096)	0
dropout_1 (Dropout)	(None, 4096)	0
dense_2 (Dense)	(None, 1000)	4097000
activation_7 (Activation)	(None, 1000)	0
dropout_2 (Dropout)	(None, 1000)	0
dense_3 (Dense)	(None, 3)	3003
activation_8 (Activation)	(None, 3)	0
Total params: 31,186,211		
Trainable params: 31,186,211		
Non-trainable params: 0		

Figure 5.7 Layers of AlexNet

AlexNet architecture consists of 5 convolutional layers, 3 max-pooling layers, 2 normalization layers, 2 fully connected layers, and 1 softmax layer. Each convolutional layer consists of convolutional filters and a nonlinear activation function ReLU. The pooling layers are used to perform max pooling.

Architecture of AlexNet:

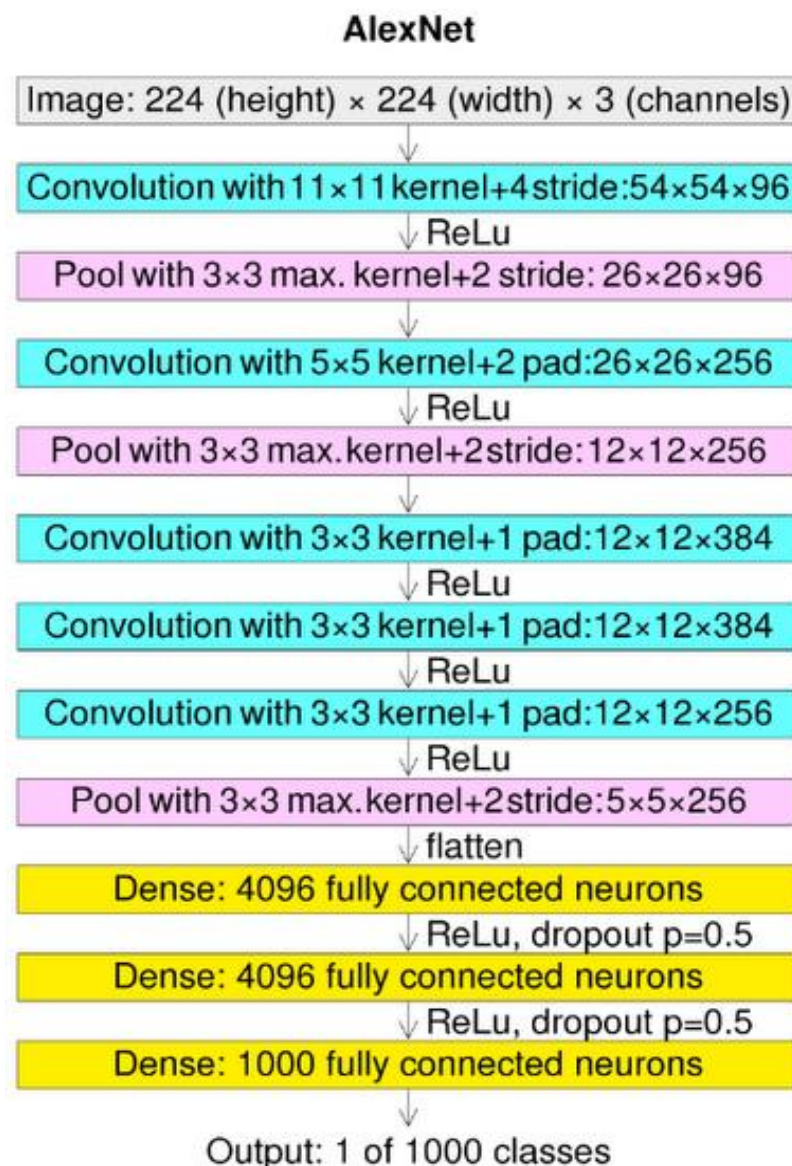


Figure 5.8 Architecture of AlexNet

Convolutional layers:

Convolutional layers are the layers where filters are applied to the original image, or to other feature maps in a deep CNN. This is where most of the user-specified parameters are in the network. The most important parameters are the number of kernels and the size of the kernels.

Pooling layers:

Pooling layers are similar to convolutional layers, but they perform a specific function such as max pooling, which takes the maximum value in a certain filter region, or average pooling, which takes the average value in a filter region. These are typically used to reduce the dimensionality of the network.

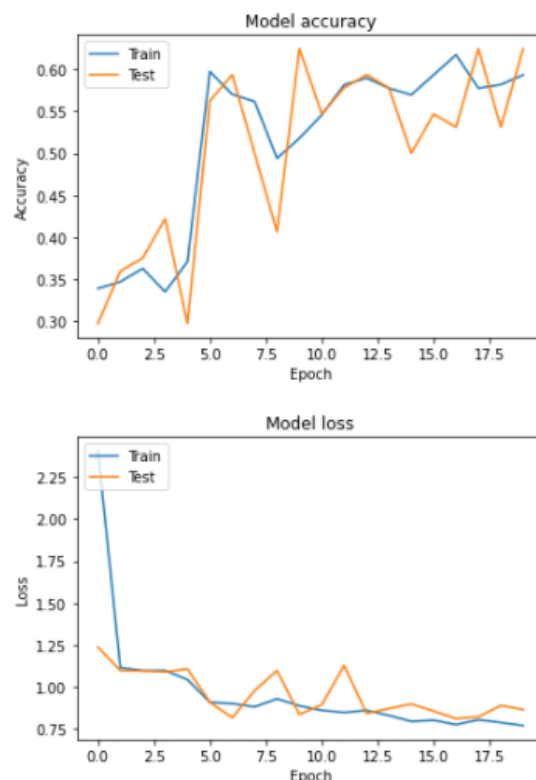


Figure 5.9 The Accuracy and loss curves for AlexNet

Dense or Fully connected layers:

Fully connected layers are placed before the classification output of a CNN and are used to flatten the results before classification. This is similar to the output layer of an MLP.

5.4 LENET

LeNet was one among the earliest convolutional neural networks which promoted the event of deep learning. After innumerable years of analysis and plenty of compelling iterations, the end result was named LeNet.

```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 83, 83, 32)	896
max_pooling2d (MaxPooling2D)	(None, 41, 41, 32)	0
conv2d_1 (Conv2D)	(None, 13, 13, 128)	36992
max_pooling2d_1 (MaxPooling2D)	(None, 6, 6, 128)	0
flatten (Flatten)	(None, 4608)	0
dense (Dense)	(None, 256)	1179904
dense_1 (Dense)	(None, 3)	771

```
Total params: 1,218,563  
Trainable params: 1,218,563  
Non-trainable params: 0
```

Figure 5.10 Layers of Lenet

Architecture of LeNet-5:

LeNet-5 CNN architecture is made up of 7 layers. The layer composition consists of 3 convolutional layers, 2 subsampling layers and 2 fully connected layers.

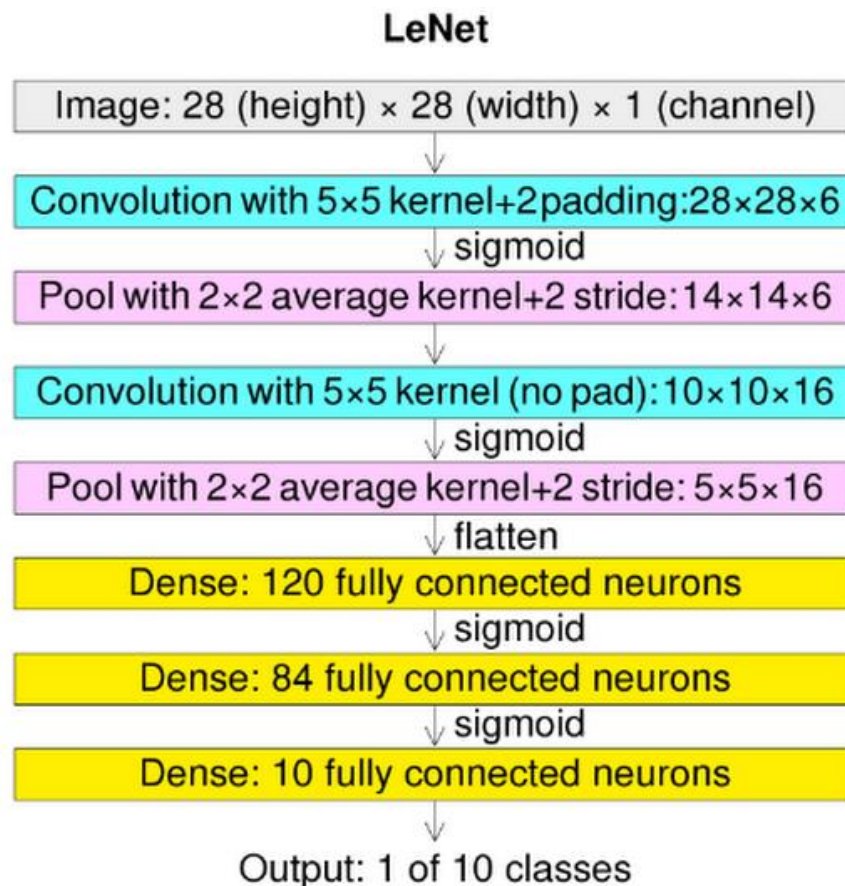


Figure 5.11 Architecture of LeNet

Convolutional layers:

Convolutional layers are the layers where filters are applied to the original image, or to other feature maps in a deep CNN. This is where most of the user-specified parameters are in the network. The most important parameters are the number of kernels and the size of the kernels.

Pooling layers:

Pooling layers are similar to convolutional layers, but they perform a specific function such as max pooling, which takes the maximum value in a certain filter region, or average pooling, which takes the average value in a filter region. These are typically used to reduce the dimensionality of the network.

```

Epoch 35/40
8/8 [=====] - 77s 10s/step - loss: 0.0383 - accuracy: 0.9801 - val_loss: 0.1066 - val_accuracy: 0.93
75
Epoch 36/40
8/8 [=====] - 77s 10s/step - loss: 0.3229 - accuracy: 0.9141 - val_loss: 0.0324 - val_accuracy: 1.00
00
Epoch 37/40
8/8 [=====] - 70s 9s/step - loss: 0.0269 - accuracy: 0.9920 - val_loss: 0.0241 - val_accuracy: 1.000
0
Epoch 38/40
8/8 [=====] - 81s 11s/step - loss: 0.0111 - accuracy: 1.0000 - val_loss: 0.0424 - val_accuracy: 0.98
44
Epoch 39/40
8/8 [=====] - 77s 10s/step - loss: 0.0600 - accuracy: 0.9766 - val_loss: 0.0144 - val_accuracy: 1.00
00
Epoch 40/40
8/8 [=====] - 73s 10s/step - loss: 0.0054 - accuracy: 1.0000 - val_loss: 0.0487 - val_accuracy: 0.98
44
Out[18]: <keras.callbacks.History at 0x19fceb08d60>

```

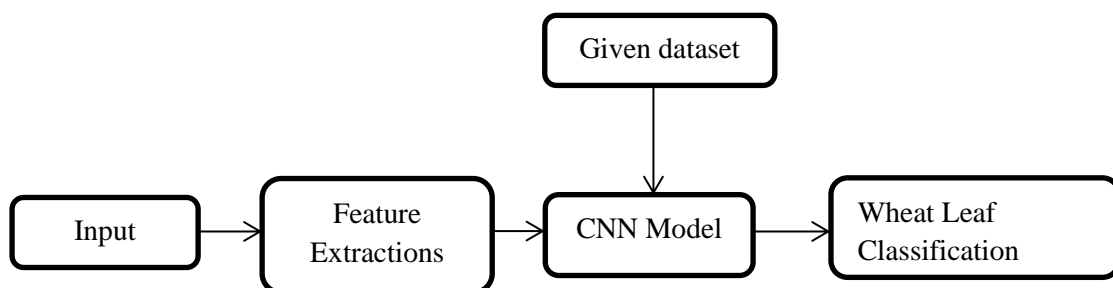
Figure 5.12 Training LeNet

Dense or Fully connected layers:

Fully connected layers are placed before the classification output of a CNN and are used to flatten the results before classification. This is similar to the output layer of an MLP.

5.5 WHEAT LEAF CLASSIFICATIONS

We give input image using keras preprocessing package. That input Image converted into array value using pillow and image to array function package. We have already classified Wheat leaf in our dataset. It classifies what are the leafs. Then we have to predict our leafs using predict function.



The Wheat leaf recognition method is based on a two-channel architecture that is able to recognize . The wheat leaf parts are cropped and extracted and then used as the input into the inception layer of the CNN. The

Training phase involves the feature extraction and classification using convolution neural network.

Libraries Required:

- ✓ **tensorflow**: Just to use the tensor board to compare the loss and adam curve our result data or obtained log.
- ✓ **keras**: To pre-process the image dataset.
- ✓ **matplotlib**: To display the result of our predictive outcome.
- ✓ **os**: To access the file system to read the image from the train and test directory from our machines

5.6 DEPLOY

Deploying the model in Django Framework and predicting output

In this module the trained deep learning model is converted into hierarchical data format file (.h5 file) which is then deployed in our django framework for providing better user interface and predicting the output whether the given leaf image is SEPTORIA / STRIP RUST / HEALTHY.

CHAPTER – 6

SOFTWARE DESCRIPTION

The purpose of the Software Requirement Specification is to produce the specification of the analysis task and also to establish complete information about the requirement, behavior and also the other constraint like functional performance and so on. The main aim of the Software Requirement Specification is to completely specify the technical requirements for the software product in a concise and in unambiguous manner.

6.1 ANACONDA NAVIGATOR

Anaconda Navigator is a desktop graphical user interface (GUI) included in Anaconda® distribution that allows you to launch applications and easily manage conda packages, environments, and channels without using command-line commands. Navigator can search for packages on Anaconda.org or in a local Anaconda Repository.

Anaconda. Now, if you are primarily doing data science work, Anaconda is also a great option. Anaconda is created by Continuum Analytics, and it is a Python distribution that comes preinstalled with lots of useful python libraries for data science.

Anaconda is a distribution of the Python and R programming languages for scientific computing (data science, machine learning applications, large-scale data processing, predictive analytics, etc.), that aims to simplify package management and deployment.

In order to run, many scientific packages depend on specific versions of other packages. Data scientists often use multiple versions of many packages and use multiple environments to separate these different versions.

The command-line program conda is both a package manager and an environment manager. This helps data scientists ensure that each version of each package has all the dependencies it requires and works correctly.

Navigator is an easy, point-and-click way to work with packages and environments without needing to type conda commands in a terminal window. You can use it to find the packages you want, install them in an environment, run the packages, and update them – all inside Navigator.

The following applications are available by default in Navigator:

- JupyterLab
- Jupyter Notebook
- Spyder
- PyCharm
- VSCode
- Anaconda Prompt (Windows only)
- Anaconda PowerShell (Windows only)

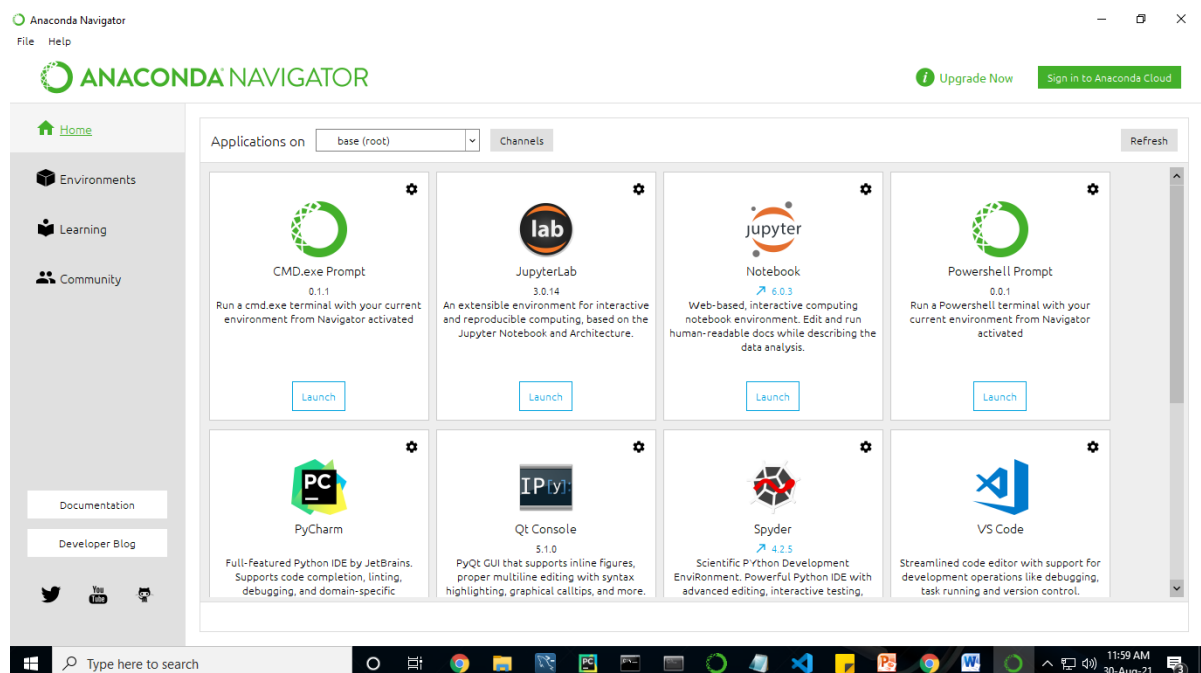


Figure 6.1 Anaconda Navigator Screenshots

Anaconda Navigator is a desktop graphical user interface (GUI) included in Anaconda distribution.

Navigator allows you to launch common Python programs and easily manage conda packages, environments, and channels without using command-line commands. Navigator can search for packages on Anaconda Cloud or in a local Anaconda Repository.

Anaconda comes with many built-in packages that you can easily find with `conda list` on your anaconda prompt. As it has lots of packages (many of which are rarely used), it requires lots of space and time as well. If you have enough space, time and do not want to burden yourself to install small utilities like JSON, YAML, you better go for Anaconda.

6.2 JUPYTER NOTEBOOK

This website acts as “meta” documentation for the Jupyter ecosystem. It has a collection of resources to navigate the tools and communities in this ecosystem, and to help you get started.

Project Jupyter is a project and community whose goal is to "develop open-source software, open-standards, and services for interactive computing across dozens of programming languages". It was spun off from IPython in 2014 by Fernando Perez.

Notebook documents are documents produced by the Jupyter Notebook App, which contain both computer code (e.g. python) and rich text elements (paragraph, equations, figures, links, etc). Notebook documents are both human-readable documents containing the analysis description and the results (figures, tables, etc.) as well as executable documents which can be run to perform data analysis.

Installation: The easiest way to install the Jupyter Notebook App is installing a scientific python distribution which also includes scientific python packages. The most common distribution is called Anaconda

Purpose: To support interactive data science and scientific computing across all programming languages.

File Extension: An IPYNB file is a notebook document created by Jupyter Notebook, an interactive computational environment that helps scientists manipulate and analyze data using Python.

The best way to get started using Python for machine learning is to complete a project.

- It will force you to install and start the Python interpreter (at the very least).
- It will give you a bird's eye view of how to step through a small project.
- It will give you confidence, maybe to go on to your own small projects.

When you are applying machine learning to your own datasets, you are working on a project. A machine learning project may not be linear, but it has a number of well-known steps:

- Define Problem.
- Prepare Data.
- Evaluate Algorithms.
- Improve Results.
- Present Results.

The best way to really come to terms with a new platform or tool is to work through a machine learning project end-to-end and cover the key steps. Namely, from loading data, summarizing data, evaluating algorithms and making some predictions.

Here is an overview of what we are going to cover:

1. Installing the Python anaconda platform.
2. Loading the dataset.
3. Summarizing the dataset.
4. Visualizing the dataset.
5. Evaluating some algorithms.
6. Making some predictions.

6.3 PYCHARM

PyCharm is a dedicated Python Integrated Development Environment (IDE) providing a wide range of essential tools for Python developers, tightly integrated to create a convenient environment for productive Python, web, and data science development. Code faster and with more easily in a smart and configurable editor with code completion, snippets, code folding and split windows support.

PyCharm Features:

- **Intelligent Coding Assistance** - PyCharm provides smart code completion, code inspections, on-the-fly error highlighting and quick-fixes, along with automated code refactorings and rich navigation capabilities.
- **Intelligent Code Editor** - PyCharm's smart code editor provides first-class support for Python, JavaScript, CoffeeScript, TypeScript, CSS, popular template languages and more. Take advantage of language-aware code completion, error detection, and on-the-fly code fixes!
- **Fast and Safe Refactorings** - Refactor your code the intelligent way, with safe Rename and Delete, Extract Method, Introduce Variable, Inline

Variable or Method, and other refactorings. Language and framework-specific refactorings help you perform project-wide changes.

- **Built-in Developer Tools** - PyCharm's huge collection of tools out of the box includes an integrated debugger and test runner; Python profiler; a built-in terminal; integration with major VCS and built-in database tools; remote development capabilities with remote interpreters; an integrated ssh terminal; and integration with Docker and Vagrant.
- **Web Development** - In addition to Python, PyCharm provides first-class support for various Python web development frameworks, specific template languages, JavaScript, CoffeeScript, TypeScript, HTML/CSS, AngularJS, Node.js, and more.
- **Python Web frameworks** - PyCharm offers great framework-specific support for modern web development frameworks such as Django, Flask, Google App Engine, Pyramid, and web2py, including Django templates debugger, manage.py and appcfg.py tools, special autocompletion and navigation, just to name a few.
- **JavaScript & HTML** - PyCharm provides first-class support for JavaScript, CoffeeScript, TypeScript, HTML and CSS, as well as their modern successors. The JavaScript debugger is included in PyCharm and is integrated with the Django server run configuration.
- **Live Edit** - Live Editing Preview lets you open a page in the editor and the browser and see the changes being made in code instantly in the browser. PyCharm auto-saves your changes, and the browser smartly updates the page on the fly, showing your edits.
- **Scientific Tools** - PyCharm integrates with IPython Notebook, has an interactive Python console, and supports Anaconda as well as multiple scientific packages including Matplotlib and NumPy.

- **Conda Integration** - Keep your dependencies isolated by having separate Conda environments per project, PyCharm makes it easy for you to create and select the right environment.
- **Cross-platform IDE** - PyCharm works on Windows, Mac OS or Linux. You can install and run PyCharm on as many machines as you have, and use the same environment and functionality across all your machines

6.4 PYTHON

In this project python is used as a programming language for development.

In technical terms, Python is an object-oriented, high-level programming language with integrated dynamic semantics primarily for web and app development. It is extremely attractive in the field of Rapid Application Development because it offers dynamic typing and dynamic binding options.

Python is relatively simple, so it's easy to learn since it requires a unique syntax that focuses on readability. Developers can read and translate Python code much easier than other languages. In turn, this reduces the cost of program maintenance and development because it allows teams to work collaboratively without significant language and experience barriers.

Additionally, Python supports the use of modules and packages, which means that programs can be designed in a modular style and code can be reused across a variety of projects. Once a module or package is developed by a user, it can be scaled for use in other projects, and it's easy to import or export these modules.

One of the most promising benefits of Python is that both the standard library and the interpreter are available free of charge, in both binary and source form. There is no exclusivity either, as Python and all the necessary tools are available

on all major platforms. Therefore, it is an enticing option for developers who don't want to worry about paying high development costs.

6.5 DJANGO

Django is a high-level Python web framework that enables rapid development of secure and maintainable websites. Built by experienced developers, Django takes care of much of the hassle of web development, so you can focus on writing your app without needing to reinvent the wheel. It is free and open source, has a thriving and active community, great documentation, and many options for free and paid-for support.

Django helps you write software that is:

Versatile

Django can be (and has been) used to build almost any type of website — from content management systems and wikis, through to social networks and news sites. It can work with any client-side framework, and can deliver content in almost any format (including HTML, RSS feeds, JSON, XML, etc). The site you are currently reading is built with Django! Internally, while it provides choices for almost any functionality you might want (e.g. several popular databases, templating engines, etc.), it can also be extended to use other components if needed.

Maintainable

Django code is written using design principles and patterns that encourage the creation of maintainable and reusable code. In particular, it makes use of the Don't Repeat Yourself (DRY) principle so there is no unnecessary duplication, reducing the amount of code. Django also promotes the grouping of related functionality into reusable "applications" and, at a

lower level, groups related code into modules (along the lines of the Model View Controller (MVC) pattern).

Portable

Django is written in Python, which runs on many platforms. That means that you are not tied to any particular server platform, and can run your applications on many flavors of Linux, Windows, and Mac OS X. Furthermore, Django is well-supported by many web hosting providers, who often provide specific infrastructure and documentation for hosting Django sites.

6.6 HTML

HTML stands for Hyper Text Markup Language. It is used to design web pages using a markup language. HTML is the combination of Hypertext and Markup language. Hypertext defines the link between the web pages. A markup language is used to define the text document within tag which defines the structure of web pages. This language is used to annotate (make notes for the computer) text so that a machine can understand it and manipulate text accordingly. Most markup languages (e.g. HTML) are human-readable. The language uses tags to define what manipulation has to be done on the text.

6.7 CSS

CSS stands for Cascading Style Sheets. It is the language for describing the presentation of Web pages, including colours, layout, and fonts, thus making our web pages presentable to the users.

CSS is designed to make style sheets for the web. It is independent of HTML and can be used with any XML-based markup language.

CHAPTER – 7

SYSTEM IMPLEMENTATION

7.1 Model-1

To build a model for training and testing:

```
import os
import numpy as np
import matplotlib.pyplot as plt
# Dl framwork - tensorflow, keras a backend
import tensorflow as tf
import tensorflow.keras.backend as K
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.layers import Input, Dense, Flatten, Dropout,
BatchNormalization
from tensorflow.keras.layers import Conv2D, SeparableConv2D, MaxPool2D,
LeakyReLU, Activation
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import ModelCheckpoint,
ReduceLROnPlateau, EarlyStopping
from IPython.display import display
from os import listdir
from os.path import isfile, join
from PIL import Image
import glob
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import Convolution2D
from tensorflow.keras.layers import MaxPooling2D
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense
import warnings
warnings.filterwarnings('ignore')
dir_name_train_Healthy='dataset/train/Healthy'
dir_name_train_septoria='dataset/train/septoria'
dir_name_tran_stripe_rust='dataset/train/stripe_rust'
def plot_images(item_dir, n=6):
    all_item_dir = os.listdir(item_dir)
    item_files = [os.path.join(item_dir, file) for file in all_item_dir][:n]
```

```

plt.figure(figsize=(35, 10))
for idx, img_path in enumerate(item_files):
    plt.subplot(2, n, idx+1)
    img = plt.imread(img_path)
    plt.imshow(img, cmap='gray')
    plt.axis('off')

plt.tight_layout()
def Images_details_Print_data(data, path):
    print("==== images in: " ,path)
    for k, v in data.items():
        print(" %s \t %s" % (k,v))

def Images_details(path):
    files = [f for f in glob.glob(path + "**/*.*", recursive=True)]
    data = {}
    data['images_count'] = len(files)
    data['min_width'] = 10**100 # No image will be bigger than that
    data['max_width'] = 0
    data['min_height'] = 10**100 # No image will be bigger than that
    data['max_height'] = 0

    for f in files:
        im = Image.open(f)
        width, height = im.size
        data['min_width'] = min(width, data['min_width'])
        data['max_width'] = max(width, data['max_height'])
        data['min_height'] = min(height, data['min_height'])
        data['max_height'] = max(height, data['max_height'])

    Images_details_Print_data(data, path)
    print("")
    print("Trained data for Healthy type disease:")
    print("")

    Images_details(dir_name_train_Healthy)
    print("")
    plot_images(dir_name_train_Healthy, 10)
    print("")
    print("Trained data for Septoria type disease:")
    print("")

```

```

Images_details(dir_name_train_septoria)
print("")
plot_images(dir_name_train_septoria, 10)
print("")
print("Trained data for Stripe_rust type disease:")
print("")

Images_details(dir_name_tran_stripe_rust)
print("")
plot_images(dir_name_tran_stripe_rust, 10)
Classifier=Sequential()
Classifier.add(Convolution2D(32,(3,3),input_shape=(250,250,3),activation="relu"))
Classifier.add(MaxPooling2D(pool_size=(2,2)))
Classifier.add(Flatten())
Classifier.add(Dense(38,activation="relu"))
Classifier.add(Dense(3,activation="softmax"))
Classifier.compile(optimizer="rmsprop",
loss="categorical_crossentropy",metrics=['accuracy'])
train_datagen=ImageDataGenerator(rescale=1./255,shear_range=0.2,zoom_range=0.2,horizontal_flip=True)
test_datagen= ImageDataGenerator(rescale=1./255)
training_set=train_datagen.flow_from_directory('Dataset/train',target_size=(250,250),batch_size=32,class_mode='categorical')
test_set=
test_datagen.flow_from_directory('Dataset/test',target_size=(250,250),batch_size=32,class_mode='categorical')
img_dims =150
epochs = 10
batch_size = 32
from PIL import ImageFile
ImageFile.LOAD_TRUNCATED_IMAGES = True
#### Fitting the model
history = Classifier.fit_generator(
    training_set, steps_per_epoch=training_set.samples // batch_size,
    epochs=epochs,
    validation_data=test_set,validation_steps=test_set.samples // batch_size)
def graph():
    #Plot training & validation accuracy values
    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.title('Model accuracy')
    plt.ylabel('Accuracy')

```

```

plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()

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

```

7.2 Model-2

```
# Dl framwork - tensorflow, keras a backend
import tensorflow as tf
import tensorflow.keras.backend as k
from tensorflow.keras.models import Model
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import MaxPooling2D
from tensorflow.keras.layers import Input
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import LeakyReLU
from tensorflow.keras.layers import Activation
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.callbacks import ReduceLROnPlateau
from tensorflow.keras.callbacks import EarlyStopping
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
# 1st Convolutional Layer
model.add(Conv2D(filters=96, input_shape=(250,250,3), kernel_size=(11,11),
strides=(4,4), padding='valid'))
model.add(Activation('relu'))
# Max Pooling
model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2), padding='valid'))
# 2nd Convolutional Layer
model.add(Conv2D(filters=256, kernel_size=(11,11), strides=(1,1),
padding='valid'))
model.add(Activation('relu'))
# Max Pooling
model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2), padding='valid'))
# 3rd Convolutional Layer
model.add(Conv2D(filters=384, kernel_size=(3,3), strides=(1,1),
padding='valid'))
```

```

model.add(Activation('relu'))
# 4th Convolutional Layer
model.add(Conv2D(filters=384, kernel_size=(3,3), strides=(1,1),
padding='valid'))
model.add(Activation('relu'))
# 5th Convolutional Layer
model.add(Conv2D(filters=256, kernel_size=(3,3), strides=(1,1),
padding='valid'))
model.add(Activation('relu'))
# Max Pooling
model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2), padding='valid'))
# Passing it to a Fully Connected layer
model.add(Flatten())
# 1st Fully Connected Layer
model.add(Dense(4096, input_shape=(224*224*3,)))
model.add(Activation('relu'))
# Add Dropout to prevent overfitting
model.add(Dropout(0.4))
# 2nd Fully Connected Layer
model.add(Dense(4096))
model.add(Activation('relu'))
# Add Dropout
model.add(Dropout(0.4))
# 3rd Fully Connected Layer
model.add(Dense(1000))
model.add(Activation('relu'))
# Add Dropout
model.add(Dropout(0.4))
# Output Layer
model.add(Dense(3))
model.add(Activation('softmax'))
model.summary()

# Compile the model
model.compile(loss = 'categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])

train_datagen=ImageDataGenerator(rescale=1./255,shear_range=0.2,zoom_range=0.2,horizontal_flip=True)
test_datagen=ImageDataGenerator(rescale=1./255)

training_set=train_datagen.flow_from_directory('dataset/train',target_size=(250,
250),batch_size=32,class_mode='categorical')

```

```

test_set=test_datagen.flow_from_directory('dataset/test',target_size=(250,250),b
atch_size=32,class_mode='categorical')

img_dims = 150
epochs = 20
batch_size = 32

#### Fitting the model
history = model.fit(
    training_set, steps_per_epoch=training_set.samples // batch_size,
    epochs=epochs,
    validation_data=test_set,validation_steps=test_set.samples // batch_size)

from PIL import ImageFile
ImageFile.LOAD_TRUNCATED_IMAGES = True

def graph():
    #Plot training & validation accuracy values
    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.title('Model accuracy')
    plt.ylabel('Accuracy')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Test'], loc='upper left')
    plt.show()

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

```

7.3 Model-3

```
from tensorflow.keras.callbacks import ModelCheckpoint,
ReduceLROnPlateau, EarlyStopping
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Convolution2D
from tensorflow.keras.layers import MaxPooling2D
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense
import warnings
warnings.filterwarnings('ignore')
Classifier=Sequential()
Classifier.add(Convolution2D(32,3,3,input_shape=(250,250,3),activation='relu'
))
Classifier.add(MaxPooling2D(pool_size=(2,2)))
Classifier.add(Convolution2D(128,3,3,activation='relu'))
Classifier.add(MaxPooling2D(pool_size=(2,2)))
Classifier.add(Flatten())
Classifier.add(Dense(256, activation='relu'))
Classifier.add(Dense(3, activation='softmax'))

Classifier.compile(optimizer='rmsprop',loss='categorical_crossentropy',metrics=
['accuracy'])
Classifier.summary()

from tensorflow.keras.preprocessing.image import ImageDataGenerator
train_datagen=ImageDataGenerator(rescale=1./255,shear_range=0.2,zoom_ran
ge=0.2,horizontal_flip='True')
test_datagen=ImageDataGenerator(rescale=1./255)
training_set=train_datagen.flow_from_directory('dataset/train',target_size=(250,
250),batch_size=32,class_mode='categorical')
test_set =
train_datagen.flow_from_directory('dataset/test',target_size=(250,250),batch_si
ze=32,class_mode='categorical')

from IPython.display import display
from PIL import ImageFile
ImageFile.LOAD_TRUNCATED_IMAGES = True
img_dims=150
epochs=40
```



```

batch_size=32
history = Classifier.fit_generator( training_set,
steps_per_epoch=training_set.samples // batch_size,
    epochs=epochs,
    validation_data=test_set,validation_steps=test_set.samples // batch_size)

import h5py
Classifier.save('leaf_.h5')

from keras.models import load_model
model=load_model('leaf_.h5')

import numpy as np
from tensorflow.keras.preprocessing import image
test_image=image.load_img('Log_0_1140.jpg',target_size=(250,250))

import matplotlib.pyplot as plt
img = plt.imshow(test_image)
print("")
test_image=image.img_to_array(test_image)
test_image=np.expand_dims(test_image,axis=0)
result=model.predict(test_image)
prediction = result[0]
classes=training_set.class_indices
classes
prediction=list(prediction)
prediction
classes=['Healthy','septoria','stripe_rust']
output=zip(classes,prediction)
output=dict(output)
output
if output['Healthy']==1.0 :
    print('Healthy')
elif output['septoria']==1.0:
    print('septoria')
elif output['stripe_rust']==1.0:
    print('stripe_rust')

plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')

```

```
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```

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

```
graph()
```

7.4 Main Page (.html)

```
<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <title>image upload example</title>

</head>

<style>

    label

    {

font-size: 20px;

color:red;

font-family: Algerian;

    }

    body

    {

        background: url(../static/image/classy.jpg);

        background-repeat: no-repeat;

        background-position: center;

        background-size: 100% 100%;
```

```
    min-height: 100vh;
}

.button {
    background-color: #4CAF50; /* Green */
    border: none;
    color: white;
    padding: 16px 32px;
    text-align: center;
    text-decoration: none;
    display: inline-block;
    font-size: 16px;
    margin: 4px 2px;
    transition-duration: 0.4s;
    cursor: pointer;
}

.button1 {
    background-color: white;
    color: black;
    border: 2px solid #4CAF50;
    border-radius: 5px;
```

```

}

#ss

{

font-size: 20px;

color:#20fc03;

background-color: black;

font-family: Times new roman;

}

.button1:hover {

    background-color: #4CAF50;

color: white;

}

h2

{

font-size: 20px;

color:black;

background-color: white;

font-family: Times new roman;

}

```

```

a
{
font-size: 20px;

color:yellow;

font-family: Times new roman;

}

.alerts-border {

border: 4px #ff0000 dashed;

animation-iteration-count: infinite;

}

@keyframes blink { 50% { border-color:yellow ; }

}

button

{

font-family: Algerian;

font-size: 35px;

font-weight: bold;

}

.alerts-border

{

```


</div>

</form>

</div>

</div>

</div>

</body>

</html>

CHAPTER – 8

TESTING

TESTING METHODOLOGIES

There are many different types of testing methods or techniques used as part of the software testing methodology. Some of the important testing methodologies are:

8.1 SYSTEM TESTING

Testing is performed to identify errors. It is used for quality assurance. Testing is an integral part of the entire development and maintenance process. The goal of the testing during this phase is to verify that the specification has been accurately and completely incorporated into the design, as well as to ensure the correctness of the design itself. For example, the design must not have any logic faults in the design before coding commences, otherwise, the cost of fixing the faults will be considerably higher as reflected. Detection of design faults can be achieved by means of inspection as well as a walkthrough.

Testing is one of the important steps in the software development phase. Testing checks for the errors, as a whole of the project testing involves the following test cases:

- Static analysis is used to investigate the structural properties of the Source code.
- Dynamic testing is used to investigate the behavior of the source code by executing the program on the test data.

Unit Testing

Unit testing is conducted to verify the functional performance of each modular component of the software. Unit testing focuses on the smallest unit of the

software design (i.e.), the module. The white-box testing techniques were heavily employed for unit testing.

Functional Tests

Functional test cases involved exercising the code with nominal input values for which the expected results are known, as well as boundary values and special values, such as logically related inputs, files of identical elements, and empty files.

Three types of tests in Functional test:

- Performance Test
- Stress Test
- Structure Test

Performance Test

It determines the amount of execution time spent in various parts of the unit, program throughput, and response time and device utilization by the program unit.

Stress Test

Stress Test is a test designed to intentionally break the unit. A Great deal can be learned about the strength and limitations of a program by examining the manner in which a programmer in which a program unit breaks.

Structure Test

Structure Tests are concerned with exercising the internal logic of a program and traversing particular execution paths. The way in which White-Box test strategy was employed to ensure that the test cases could Guarantee that all independent paths within a module have been exercised at least once.

- Exercise all logical decisions on their true or false sides.

- Execute all loops at their boundaries and within their operational bounds.
- Exercise internal data structures to assure their validity.
- Checking attributes for their correctness.
- Handling end of file conditions, I/O errors, buffer problems, and textual errors in the output information

Integration Testing

Integration testing is a systematic technique for constructing the program structure while at the same time conducting tests to uncover errors associated with interfacing. i.e., integration testing is the complete testing of the set of modules that makes up the product. The objective is to take untested modules and build a program structure tester should identify critical modules. Critical modules should be tested as early as possible. One approach is to wait until all the units have passed testing, and then combine them and then test. This approach evolved from the unstructured testing of small programs. Another strategy is to construct the product in increments of tested units. A small set of modules are integrated together and tested, to which another module is added and tested in combination. And so on. The advantages of this approach are that interface dispenses can be easily found and corrected.

The major error that was faced during the project is a linking error. When all the modules are combined the link is not set properly with all support files. Then we checked out for interconnection and the links. Errors are localized to the new module and its intercommunications. The product development can be staged, and modules integrated as they complete unit testing. Testing is completed when the last module is integrated and tested.

8.2 TESTING TECHNIQUES / TESTING STRATEGIES

Testing is the process of executing a program with the intent of finding an error. A good test case is one that has a high probability of finding an as-yet-undiscovered error. A successful test is one that uncovers an as-yet-undiscovered error. System testing is the stage of implementation, which is aimed at ensuring that the system works accurately and efficiently as expected before live operation commences. It verifies that the whole set of programs hangs together. System testing requires a test consisting of several key activities and steps for running a program, string, and system and is important in adopting a successful new system. This is the last chance to detect and correct errors before the system is installed for user acceptance testing.

The software testing process commences once the program is created and the documentation and related data structures are designed. Software testing is essential for correcting errors. Otherwise, the program or the project is not said to be complete. Software testing is the critical element of software quality assurance and represents the ultimate review of specification design and coding. Testing is the process of executing the program with the intent of finding the error. A good test case design is one that has a probability of finding a yet undiscovered error. A successful test is one that uncovers a yet undiscovered error. Any engineering product can be tested in one of the two ways:

White-box testing

This testing is also called Glass box testing. In this testing, by knowing the specific functions that a product has been designed to perform tests can be conducted that demonstrate each function is fully operational and at the same time search for errors in each function. It is a test case design method that uses the control structure of the procedural design to derive test cases. Basis path testing is white box testing.

Basis path testing:

- Flow graph notation
- Cyclomatic complexity
- Deriving test cases
- Graph matrices Control

Black box testing

In this testing by knowing the internal operation of a product, a test can be conducted to ensure that “all gears mesh”, that is the internal operation performs according to specification and all internal components have been adequately exercised. It fundamentally focuses on the functional requirements of the software.

The steps involved in black-box test case design are:

- Graph-based testing methods
- Equivalence partitioning
- Boundary value analysis
- Comparison testing

8.3 SOFTWARE TESTING STRATEGIES:

A software testing strategy provides a road map for the software developer. Testing is a set activity that can be planned in advance and conducted systematically. For this reason, a template for software testing a set of steps into which we can place specific test case design methods should be strategy should have the following characteristics:

- Testing begins at the module level and works “outward” toward the integration of the entire computer-based system.
- Different testing techniques are appropriate at different points in time.

- The developer of the software and an independent test group conducts testing.
- Testing and Debugging are different activities but debugging must be accommodated in any testing strategy.

Integration Testing

Integration testing is a systematic technique for constructing the program structure while at the same time conducting tests to uncover errors associated with it. Individual modules, which are highly prone to interface errors, should not be assumed to work instantly when we put them together. The problem, of course, is “putting them together”- interfacing. There may be the chances of data loss across another’s sub-functions when combined may not produce the desired major function; individually acceptable impressions may be magnified to unacceptable levels; global data structures can present problems.

Program Testing

The logical and syntax errors have been pointed out by program testing. A syntax error is an error in a program statement that violates one or more rules of the language in which it is written. An improperly defined field dimension or omitted keywords are common syntax errors. These errors are shown through error messages generated by the computer. A logic error on the other hand deals with the incorrect data fields, out-of-range items, and invalid combinations. Since the compiler will not deduct logical errors, the programmer must examine the output. Condition testing exercises the logical conditions contained in a module. The possible types of elements in a condition include a Boolean operator, Boolean variable, a pair of Boolean parentheses A relational operator or an arithmetic expression. The condition testing method focuses on testing each condition in the program. The purpose of the condition test is to deduct not only errors in the condition of a program but also other errors in the program.

Security Testing

Security testing attempts to verify the protection mechanisms built into a system well, in fact, protect it from improper penetration. The system security must be tested for invulnerability from frontal attacks must also be tested for invulnerability from rear attacks. During security, the tester places the role of the individual who desires to penetrate the system.

Validation Testing

At the culmination of integration testing, the software is completely assembled as a package. Interfacing errors have been uncovered and corrected and a final series of software test-validation testing begins. Validation testing can be defined in many ways, but a simple definition is that validation succeeds when the software functions in a manner that is reasonably expected by the customer. Software validation is achieved through a series of black-box tests that demonstrate conformity with requirements. After the validation test has been conducted, one of two conditions exists.

- The function or performance characteristics conform to specifications and are accepted.
- A validation from the specification is uncovered and a deficiency is created.

Deviations or errors discovered at this step in this project are corrected prior to completion of the project with the help of the user by negotiating to establish a method for resolving deficiencies. Thus, the proposed system under consideration has been tested by using validation testing and found to be working satisfactorily. Though there were deficiencies in the system they were not catastrophic.

User Acceptance Testing

User acceptance of the system is a key factor for the success of any system. The system under consideration is tested for user acceptance by constantly keeping in touch with prospective systems and users at the time of developing and making changes whenever required. This is done in regard to the following points.

- Input screen design.
- Output screen design.

8.4 TEST CASES& REPORT:

S.NO	ACTION	INPUTS	EXPECTED OUTPUT	ACTUAL OUTPUT	TEST RESULT
1.	HEALTHY WHEAT LEAF DETECTION	Input any HEALTHY WHEAT LEAF image	Prediction to be HEALTHY	Predicted HEALTHY	Pass
2.	SEPTORIA WHEAT LEAF DETECTION	Input any SEPTORIA DETECTION image	Prediction to be SEPTORIA	Predicted SEPTORIA	Pass
3.	STRIPE RUST WHEAT LEAF DETECTION	Input any STRIPE RUST image	Prediction to be STRIPE RUST	Predicted STRIPE RUST	Pass

Tab 8.1 Test case

CHAPTER – 9

CONCLUSION

9.1 RESULTS AND DISCUSSIONS

This chapter discusses about the practical results obtained while implementing the project.

9.1.1 RESULTS OBTAINED

To begin with, testing of the trained model, we can split our project into modules of implementation that is done.

Dataset collection involves the process of collecting Healthy, Septoria and stripe rust image dataset.

The dataset has been collected for the project and the below figure can be seen as follows:

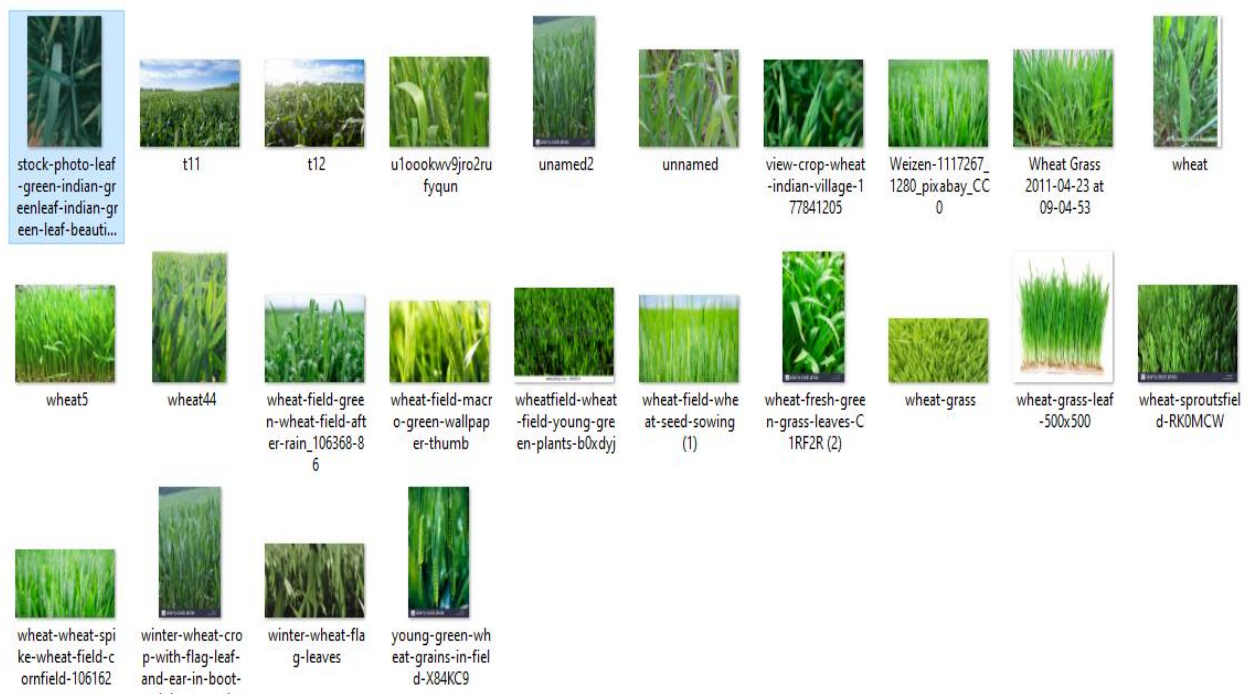


Figure 9.1 Dataset collected for healthy wheat leaf



Figure 9.2 Dataset collected for Septoria wheat leaf



Figure 9.3 Dataset collected for Stripe rust wheat leaf

These datasets are then augmented, to increase dataset size.

Then these datasets are pre-processed from convert the images into required size format so that it can be made ready for training with the model.

After pre-processing, training is performed using deep learning algorithms.

The algorithm used for training is AlexNet, and LeNet . The LeNet Algorithm outperformed the AlexNet algorithm and the results obtained can be seen below.

The below figure shows the LeNet Model training.

```
Epoch 1/40
8/8 [=====] - 42s 5s/step - loss: 1.1215 - accuracy: 0.4821 - val_loss: 0.6219 - val_accuracy: 0.703
1
Epoch 2/40
8/8 [=====] - 34s 4s/step - loss: 0.5915 - accuracy: 0.7171 - val_loss: 0.5248 - val_accuracy: 0.765
6
Epoch 3/40
8/8 [=====] - 38s 5s/step - loss: 0.4386 - accuracy: 0.8526 - val_loss: 0.3917 - val_accuracy: 0.843
8
Epoch 4/40
8/8 [=====] - 41s 5s/step - loss: 0.2787 - accuracy: 0.8884 - val_loss: 0.8153 - val_accuracy: 0.671
9
Epoch 5/40
8/8 [=====] - 42s 5s/step - loss: 0.4358 - accuracy: 0.8287 - val_loss: 0.3260 - val_accuracy: 0.890
6
Epoch 6/40
8/8 [=====] - 39s 5s/step - loss: 0.2580 - accuracy: 0.9044 - val_loss: 0.1931 - val_accuracy: 0.921
9
```

Figure 9.4 Training of LeNet model

The below figure shows the training and validation loss of LeNet Model.

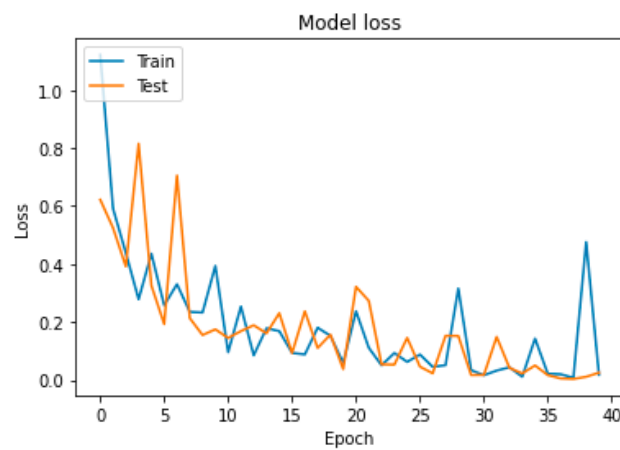


Figure 9.5 The training and validation loss graph (LeNet)

The below figure shows the training and validation accuracy of LeNet Model

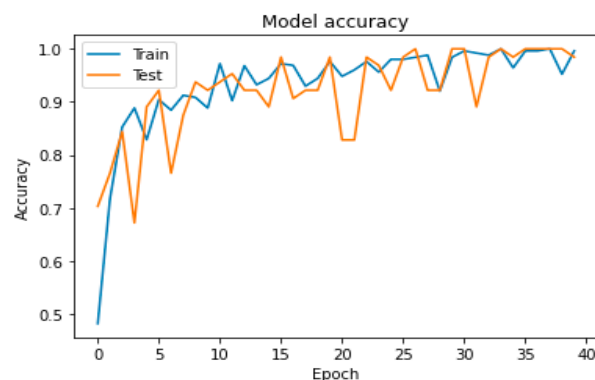


Figure 9.6 The training and validation Accuracy graph (LeNet)

Output prediction done with the deployed (LeNet algorithm)



[Result](#) [Go Back!!!](#)



[Result](#) [Go Back!!!](#)



[Result](#) [Go Back!!!](#)



Figure 9.7 Prediction

9.2 CONCLUSION & FUTURE WORK

9.2.1 CONCLUSION

In this project, a research to classify Wheat leaf Disease Classification over static images using deep learning techniques was developed. This is a complex problem that has already been approached several times with different techniques. While good results have been achieved using feature engineering, this project focused on feature learning, which is one of DL promises. While feature engineering is not necessary, image pre-processing boosts classification accuracy. Hence, it reduces noise on the input data. Nowadays, Agriculture based AI wheat leaf disease includes is heavily required. The solution totally based on feature learning does not seem close yet because of a major limitation. Thus, leaf Disease classification could be achieved by means of deep learning techniques.

9.2.2 FUTURE WORK

Further improvement on the network's accuracy and generalization can be achieved through the following practices. The first one is to use the whole dataset during the optimization. Using batch optimization is more suitable for larger datasets. Another technique is to evaluate wheat leaf one by one. This can lead to detect which classification are more difficult to classify. Finally, using a larger dataset for training seems beneficial. However, such a dataset might not exist nowadays. Using several datasets might be a solution, but a careful procedure to normalize them is required. Finally, using full dataset for training, pre-training on each categories, and using a larger dataset seem to have the possibility to improve the network's performance. Thus, they should be addressed in future research on this topic.

APPENDICES

A.1 Screenshots

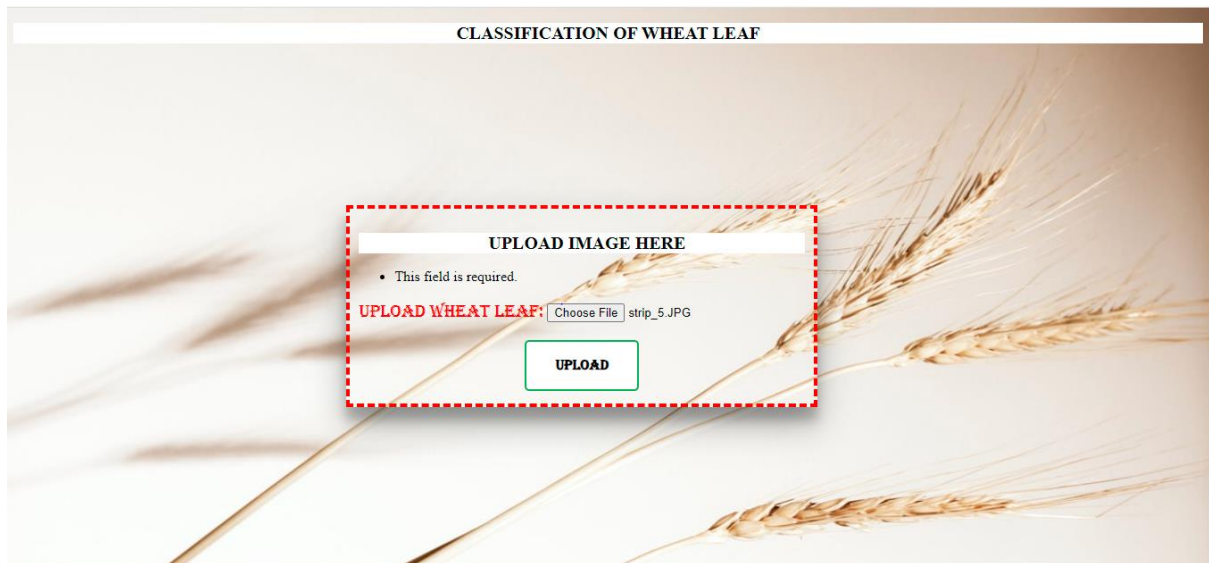


Figure A1.1. Home Page

The below figure shows the prediction of healthy.



Figure A1.2 Prediction of healthy

The below figure shows the prediction of septoria.

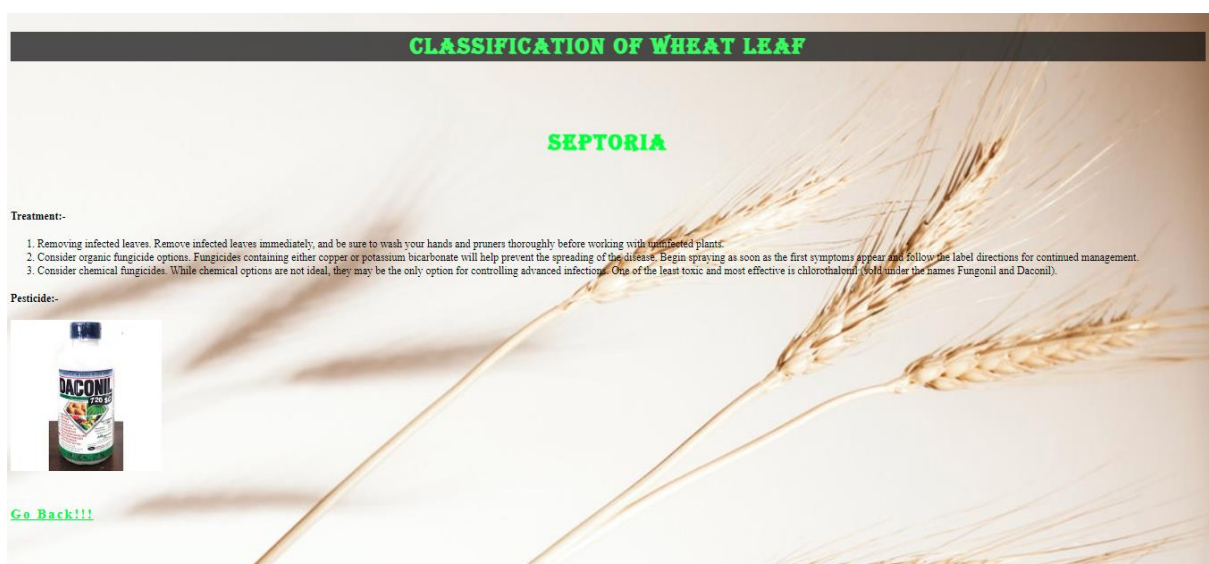


Figure A1.3 Prediction of septoria

The below figure shows the prediction of stripe rust.

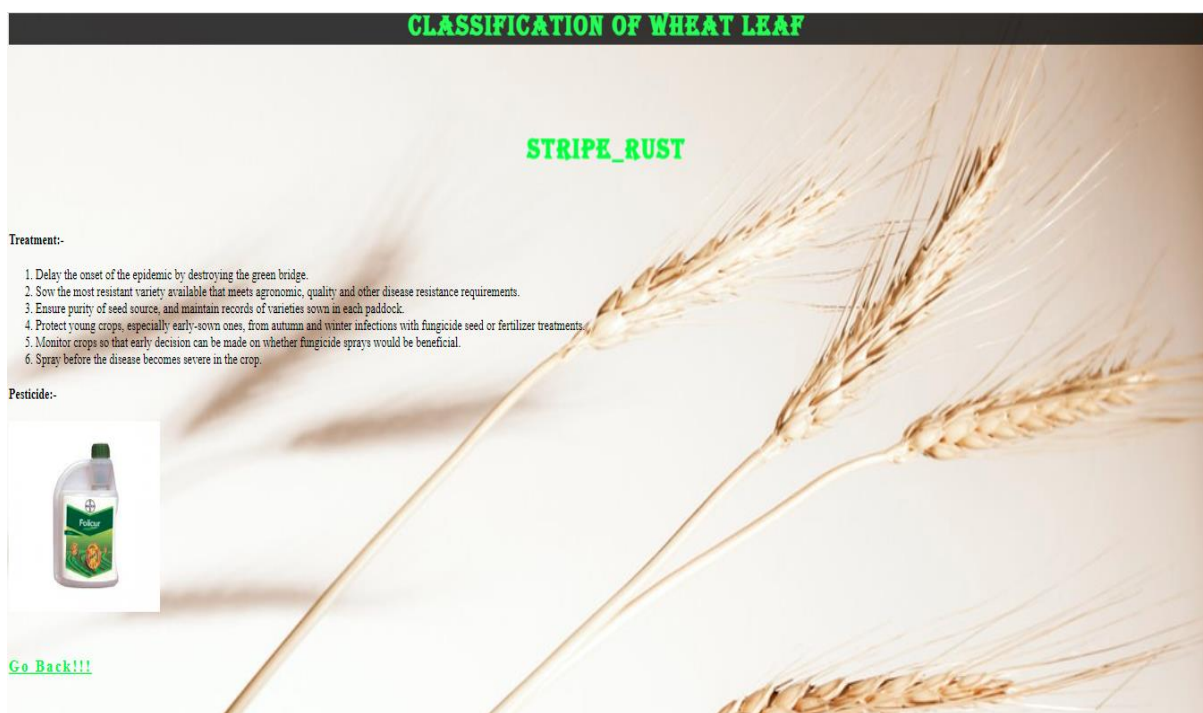


Figure A1.4 Prediction of stripe rust

REFERENCE

- [1] Amina Khatra, “Yellow Rust Extraction in Wheat Crop based on Color Segmentation Techniques IOSR Journal of Engineering, vol. 3, PP 56-58, December 2013.
- [2] Sumit Nema, Bharat Mishra and Mamta Lambert “Android Application of Wheat Leaf Disease Detection and Prevention using Machine Learning”, International Journal of Trend in Research and Development., vol. 7(2), April 2020, ISSN: 2394-9333.
- [3] Er.Varinderjit Kaur and Dr.Ashish Oberoi, “Wheat Disease Detection Using SVM Classifier,” JETIR, August 2018, Volume 5, Issue 8, ISSN-2349-5162.
- [4] Simranjeet Kaur, Geetanjali Babbar, Navneet Sandhu and Dr. Gagan Jindal,” Various Plant Diseases Detection Using Image Processing Methods,” IJSDR, June 2019, Volume 4, Issue 6, ISSN: 2455-2631.
- [5] Akshai KP and J.Anitha,” Plant disease classification using deep learning,” 2021 3rd International Conference on Signal Processing and Communication (ICPSC) | 13 – 14 May 2021 | Coimbatore.
- [6] Singh V and Misra, A.K. Detection of plant leaf diseases using image segmentation and soft computing techniques. Inf. Process. Agric.2017, 4, 41–49.
- [7] Sangeetha, R.; Rani, M. Tomato Leaf Disease Prediction Using Transfer Learning. In Proceedings of the International Advanced Computing Conference 2020, Panaji, India, 5–6 December 2020.
- [8] Hasan, M.; Tanawala, B.; Patel, K.J. Deep learning precision farming: Tomato leaf disease detection by transfer learning. In Proceeding of the 2nd International Conference on Advanced Computing and Software Engineering (ICACSE), Sultanpur, India, 8–9 February 2019.
- [9] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems, pp. 1097–1105. 2012.