

Plant Disease Detection and Classification by Deep Learning

Submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering Degree in Computer Science and Engineering

by

KH BIKASH SINGHA (Reg No: 38110251)

KRITI ANAND (Reg No: 38110271)



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

SCHOOL OF COMPUTING

SATHYABAMA

**INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)**

Accredited with Grade "A" by NAAC

JEPPIAAR NAGAR, RAJIV GANDHI SALAI, CHENNAI - 600 119

APRIL 2022

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the bonafide work of **KH BIKASH SINGHA (Reg No: 38110251)** and **KRITI ANAND (Reg No: 38110271)** who carried out the project entitled “**Plant Disease Detection and Classification by Deep Learning**” under my supervision from December 2021 to March 2022.

Internal Guide

Dr. A. Christy, M.C.A., Ph.D.

Head of the Department

Dr. S. VIGNESHWARI, M.E., Ph.D

Submitted for Viva voce Examination held on _____

Internal Examiner

External Examiner

DECLARATION

I, **KH BIKASH SINGHA** hereby declare that the project report entitled “**Plant Disease Detection and Classification by Deep Learning**” was done by me under the guidance of **Dr.A.Christy** is submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering Degree in Computer Science and Engineering.

DATE: 31/03/2022

PLACE: Chennai

SIGNATURE OF THE CANDIDATE



SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

(DEEMED TO BE UNIVERSITY)

Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE

www.sathyabama.ac.in

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to the **Board of Management of Sathyabama** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr. T. Sasikala, M.E., Ph.D, Dean, School of Computing, Dr. S. Vigneshwari, M.E., Ph.D. and Dr. L. Lakshmanan, M.E., Ph.D., Heads of the Department of Computer Science and Engineering** for providing me necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide **Dr.A.Christy** for his valuable guidance, suggestions and constant encouragement paved the way for the successful completion of my project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the **Department of Computer Science and Engineering** who were helpful in many ways for the completion of the project

ABSTRACT

Traditional farming is going out of date nowadays. Technologies are being introduced in the farming sector for the past decade and in recent years it is seen that the participation of deep learning and machine learning is playing an integral role in solving traditional problems. The introduction of new technology has increased the productivity of farmers and also increased the yields and quality of the crops too. Plant diseases are a serious concern for the consumers and the farmers too. It does not only carry some harmful bacteria within itself however it compromises the yield of the crops too. The identification of such plant diseases has been a continuous problem for cultivators and researchers. Deep learning-enabled developments in the field of computer vision have paved the path for computer-assisted plant disease diagnosis. Deep Learning has achieved great success in the categorization of a number of plant diseases by exploiting its ability to recognize objects with the help of convolutional neural networks. Various deep learning algorithm like AlexNet and LeNet-5 is applied on a publicly available dataset (plantvillage dataset) so that the neural network can capture the various features of a specific disease and diagnose it accordingly using a human-like decision making skill

TABLE OF CONTENT

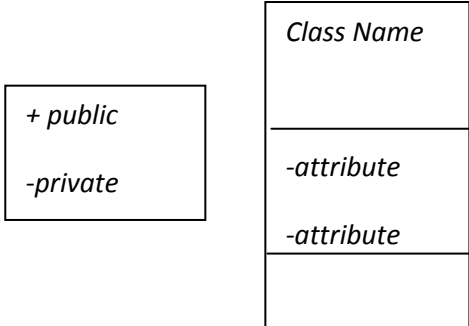
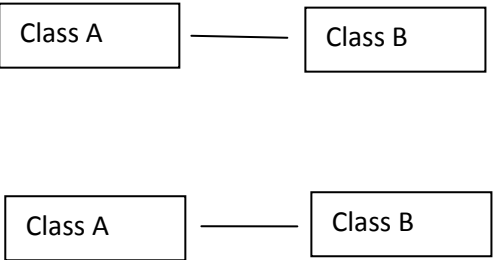
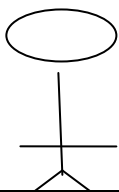
SL.NO	TITLE	PAGE.NO
01.	INTRODUCTION	1
	1.1 OUTLINE OF THE PROJECT	1
	1.2 OBJECTIVES	2
02.	LITERATURE SURVEY	3-7
03.	AIM AND SCOPE	7
	3.1 PROJECT GOAL	7
	3.2 SCOPE OF THE PROJECT	8
	3.3 OVERVIEW OF THE SYSTEM	9-11
04.	METHODOLOGY	12
	4.1 SYSTEM ARCHITECTURE	18-23
	4.2 TYPES OF CNN	23-27
05.	COMPARISION AND ANALYSIS	27-29
06.	SUMMARY, CONCLUSION AND FUTURE WORK	30-32
07.	APPENDICES	32
	7.1 SOURCE CODE	32-55
	7.2 SCREENSHOTS	56-59
	7.3 PUBLICATION AND PLAGIARISM REPORT	60-61
	7.4 REFERENCE	62-63

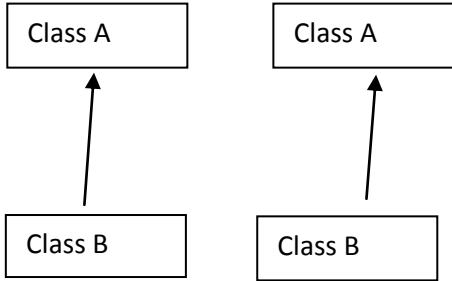
LIST OF FIGURES

SL.NO	TITLE	PAGE.NO
1.	DATAFLOW DIAGRAM	2
2.	SYSTEM ARCHITECTURE	18
3.	WORKFLOW DIAGRAM	19
4.	USECASE DIAGRAM	19
5.	CLASS DIAGRAM	20
6.	ACTIVITY DIAGRAM	21
7.	SEQUENCE DIAGRAM	21
8.	E.R – DIAGRAM	22
9.	COLLABORATION DIAGRAM	23

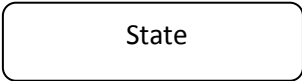
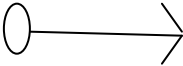
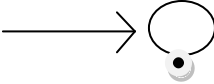
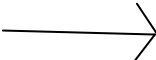
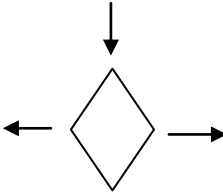
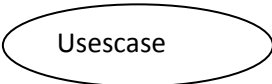
LIST OF SYMBOLS

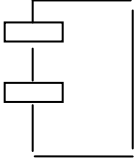
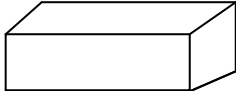
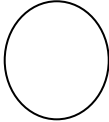
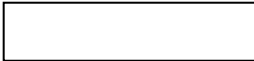
	NOTATION		
--	-----------------	--	--


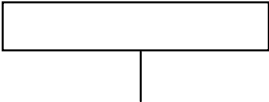
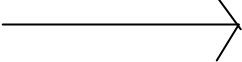
S.NO	NAME	NOTATION	DESCRIPTION
1.	Class		Represents a collection of similar entities grouped together.
2.	Association	<p>NAME</p> 	Associations represents static relationships between classes. Roles represents the way the two classes see each other.
3.	Actor		It aggregates several

			classes into a single classes.
4.	Aggregation	 <pre> graph BT B1[Class B] --> A1[Class A] B2[Class B] --> A2[Class A] </pre>	Interaction between the system and external environment

5.	<i>Relation</i> (uses)	<i>uses</i>	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 processs.
9.	Initial State		Initial state of the object
10.	Final state		F inal 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	<p>Message</p> 	Represents the message exchanged.

1. INTRODUCTION

India being an agriculture country, about 70% of the population depends on it as their main source of income and food. Agriculture plays an important part of the Indian economy as it contributes about 17% of the total GDP. Farmers have wide range in selecting their crops and finding a suitable pesticide for it but in spite of all their efforts it can all be vain if they can't identify the disease plaguing their crops. Thus, disease on crops can significantly reduce the quality and quantity of agricultural products along with economical damage to the farmers. To successfully cultivate crops without incurring much loss we need to properly identify the disease and remedy it, this requires a lot of work and processing time as detecting each and every plant can be tedious and time consuming. To lessen the burden of the farmers along with their losses we propose the use of a system which can detect infected plants so that we can curb the spread of infection and diseases at an earlier step thus reducing losses and crop failure.

In most cases symptoms like fungal infection and rot can be seen on the leaves, stem and fruit. This project provides an insight into how we deal with the problem and further discuss the challenges of our work and how we can improve upon it in future work.

1.1 OUTLINE OF THE PROJECT

Overview of the system:

- Define a problem
- Gathering image data set
- Evaluating algorithms
- Detecting results

The steps involved in Building the data model is depicted below.

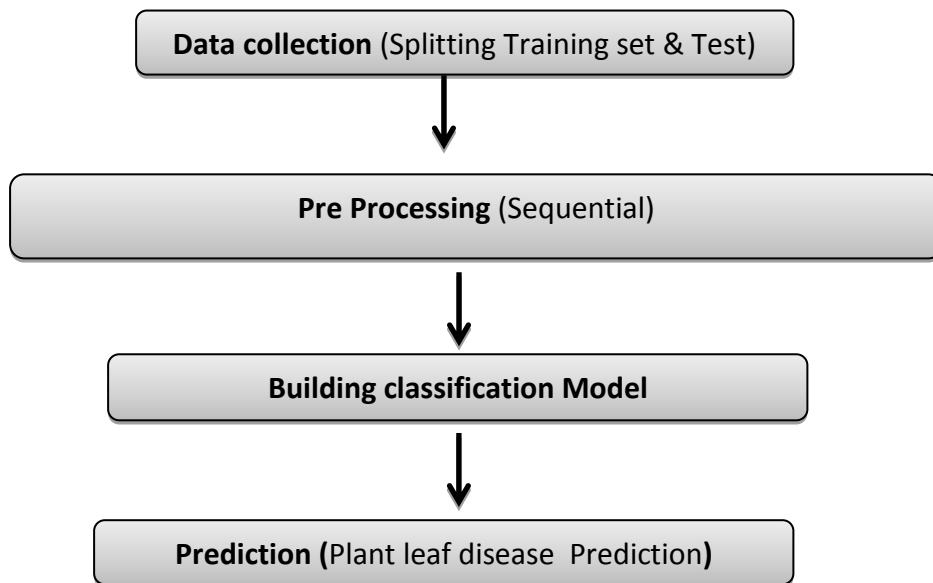


Fig 1: Data flow diagram for CNN model

1.2 OBJECTIVE :

Smart farming system using necessary infrastructure is an innovative technology that helps to improve the quality and quantity of agricultural production in the country. Disease in plants has long been one of the major threats to food security as it dramatically reduces the crop yield and compromises the quality. The identification of such diseases has been a significant challenge to cultivators and researchers. Deep learning-enabled developments in the field of computer vision have paved the path for computer-assisted plant disease diagnosis. Deep learning with convolutional neural networks (CNN) has achieved tremendous success in the categorization of a number of plant diseases by exploiting its ability to recognise objects, and the solution provides an efficient technique for detecting plant disease. Various CNN algorithm like AlexNet and LeNet-5 is applied on a publicly available dataset (plant village dataset) so that the neural network can capture the various features of specific disease and diagnose it accordingly using a human-like decision making skill.

2. LITERATURE SURVEY

A literature review is a body of text that aims to review the critical points of current knowledge on and/or methodological approaches to a particular topic. It is secondary sources and discuss published information in a particular subject area and sometimes information in a particular subject area within a certain time period. Its ultimate goal is to bring the reader up to date with current literature on a topic and forms the basis for another goal, such as future research that may be needed in the area and precedes a research proposal and may be just a simple summary of sources. Usually, it has an organizational pattern and combines both summary and synthesis.

A summary is a recap of important information about the source, but a synthesis is a re-organization, reshuffling of information. It might give a new interpretation of old material or combine new with old interpretations or it might trace the intellectual progression of the field, including major debates. Depending on the situation, the literature review may evaluate the sources and advise the reader on the most pertinent or relevant of them. Loan default trends have been long studied from a socio-economic stand point. Most economics surveys believe in empirical modeling of these complex systems in order to be able to predict the loan default rate for a particular individual. The use of machine learning for such tasks is a trend which it is observing now. Some of the surveys to understand the past and present perspective of loan approval or not.

REVIEW OF LITERATURE SURVEY

Title: Yellow Rust Extraction in Wheat Crop based on Color Segmentation Techniques

Author: Amina Khatra

Year: December 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.

TITLE: 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.

TITLE: VARIOUS PLANT DISEASES DETECTION USING IMAGE PROCESSING METHODS

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

YEAR: June 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 workdone, 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

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

AUTHOR: Sumit Nema, Bharat Mishra and Mamta Lambert

YEAR: APR-2020

Crop quality and production plays an important role in agriculture and farmer's life. Famer'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.

TITLE: WHEAT DISEASE DETECTION USING SVM CLASSIFIER

AUTHOR: Er.Varinderjit Kaur , Dr.Ashish Oberoi

YEAR: AUG 2018

There are many types of diseases which are present in plants. To detect these diseases, patterns 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 engineering and research community is doing lot of work on Wheat disease detection, but the application of this techniques to solve practical agricultural This paper presents a survey on SVM Classifier method that use digital image processing techniques to detect, quantify and classify plant diseases from digital images in the visible spectrum

It reviews, and summaries various techniques used for classifying and detecting various bacterial, fungal and viral wheat leaf diseases. The classification techniques help 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

Drawback:

- It has not focused on identifying other plant diseases with CNN as classifier.
- It has not focused on increasing the recognition

3. AIM AND SCOPE

3.1 PROJECT GOAL

To classify different plant diseases, we plan to design a deep learning system so that a person without expertise in software should also be able to use it easily. The proposed system is made to predict plant diseases using the leaves as an identifying factor. It explains the analysis of our methodology along with some of the feature engineering of the data. A large number of images is collected for each disease and is classified into database images and input images. The primary attributes of the leaves that are important are the shape and texture-oriented features. The figure provided below gives us an insight into the basic principle of our system along with an idea about how the system works.

3.2 SCOPE OF THE PROJECT

India is an agriculture-based country and about 70% of the population depends on it as their main source of income and food. Farmers have wide range in selecting their crops and finding a suitable pesticide for it but in spite of all their efforts it can all be vain if they can't identify the disease plaguing their crops. Thus disease on crops can significantly reduce the quality and quantity of agricultural products along with economical damage to the farmers. To successfully cultivate crops without incurring much loss we need to properly identify the disease and remedy it, this requires a lot of work and processing time as detecting each and every plant can be tedious and time consuming. To lessen the burden of the farmers along with their losses we propose the use of a system which can detect infected plants so that we can curb the spread of infection and diseases at an earlier step thus reducing losses and crop failure.

In most cases symptoms like fungal infection and rot can be seen on the leaves, stem and fruit. This paper provides an insight into how we deal with the problem and further discuss the challenges of our work and how we can improve upon it in future work.

So, to classify different plant diseases, we plan to design a deep learning system so that a person without expertise in software should also be able to use it easily. The proposed system is made to predict plant diseases using the leaves as an identifying factor. It explains the analysis of our methodology along with some of the feature engineering of the data. A large number of images is collected for each disease and is classified into database images and input images. The primary attributes of the leaves that are important are the shape and texture-oriented features. The figure provided below gives us an insight into the basic principle of our system along with an idea about how the system works.

3.3 OVERVIEW OF THE SYSTEM

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

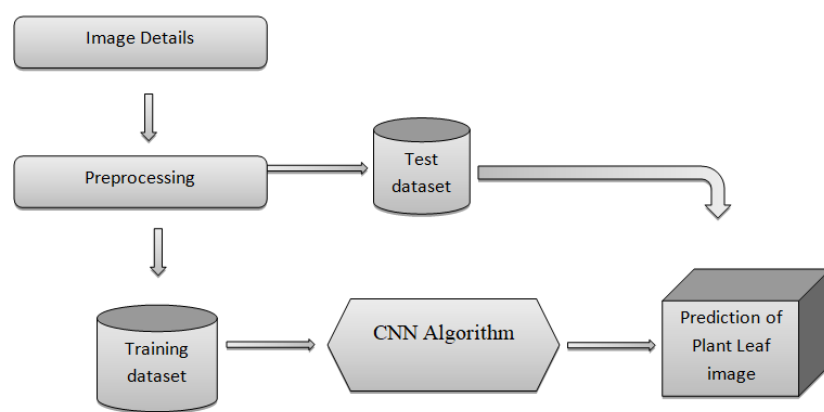


Fig 2: Overview of the System

- **DFD(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. DFDs can also be used for the visualization of data processing (structured design). A DFD shows what kind of information will be input to and output from the system, how the data will advance through the system, and where the data will be stored. It does not show information about process timing or whether processes will operate in sequence or in parallel, unlike a traditional structured flowchart which focuses on control flow, or a UML activity workflow diagram, which presents both control and data flows as a unified model. Data flow diagrams are also known as bubble charts. DFD is a designing tool used in the top down approach to Systems Design. Symbols and Notations Used in DFDs Using any

convention's DFD rules or guidelines, the symbols depict the four components of data flow diagrams.

External entity: an outside system that sends or receives data, communicating with the system being diagrammed. They are the sources and destinations of information entering or leaving the system. They might be an outside organization or person, a computer system or a business system. They are also known as terminators, sources and sinks or actors. They are typically drawn on the edges of the diagram.

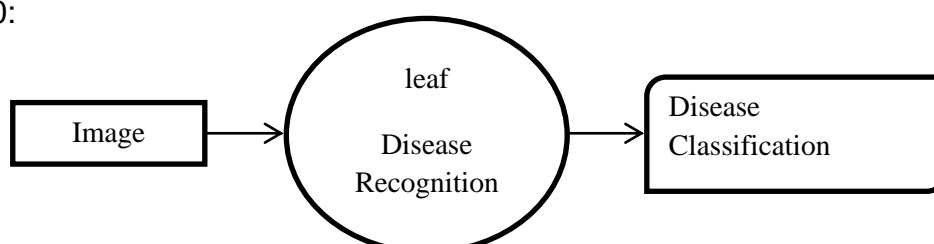
Process: any process that changes the data, producing an output. It might perform computations, or sort data based on logic, or direct the data flow based on business rules.

Data store: files or repositories that hold information for later use, such as a database table or a membership form.

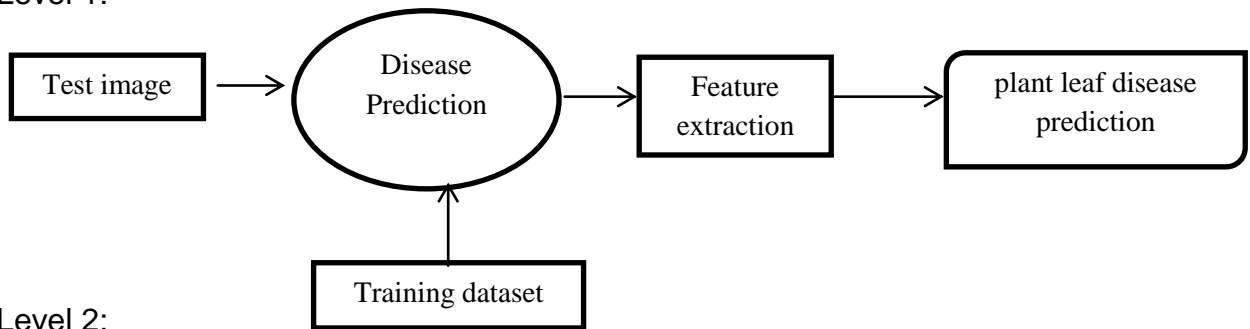
Data flow: the route that data takes between the external entities, processes and data stores. It portrays the interface between the other components and is shown with arrows, typically labeled with a short data name, like "Billing details."

DFD levels and layers A data flow diagram can dive into progressively more detail by using levels and layers, zeroing in on a particular piece. DFD levels are numbered 0, 1 or 2, and occasionally go to even Level 3 or beyond. The necessary level of detail depends on the scope of what you are trying to accomplish. 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, data analysts and developers.

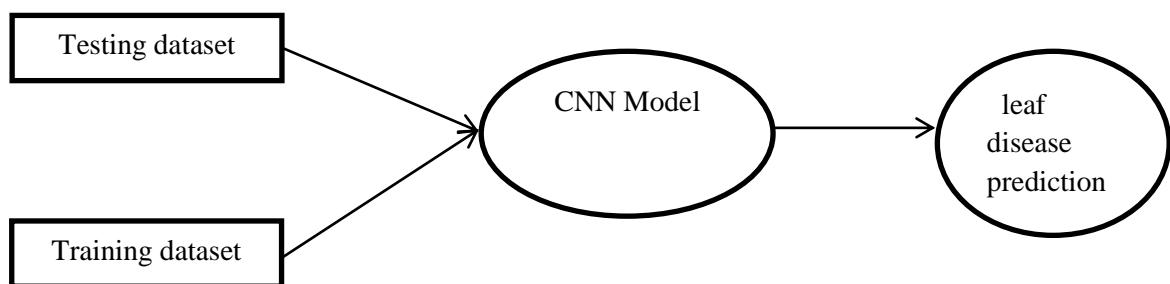
Level 0:



Level 1:



Level 2:



Level 3:

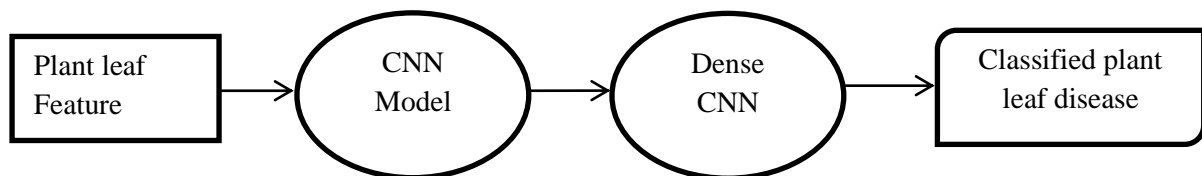


Fig 3: Data Flow Diagram

DESIGN ENGINEERING

General

Design is meaningful engineering representation of something that is to be built. Software design is a process design is the perfect way to accurately translate requirements in to a finished software product. Design creates a representation or model, provides detail about software data structure, architecture, interfaces and components that are necessary to implement a system.

4. METHODOLOGY

Preprocessing and Training the model (CNN): The dataset is preprocessed such as Image reshaping, resizing and conversion to an array form. Similar processing is also done on the test image. A dataset consisting of about 10 different class of leaf, out of which any image can be used as a test image for the software.

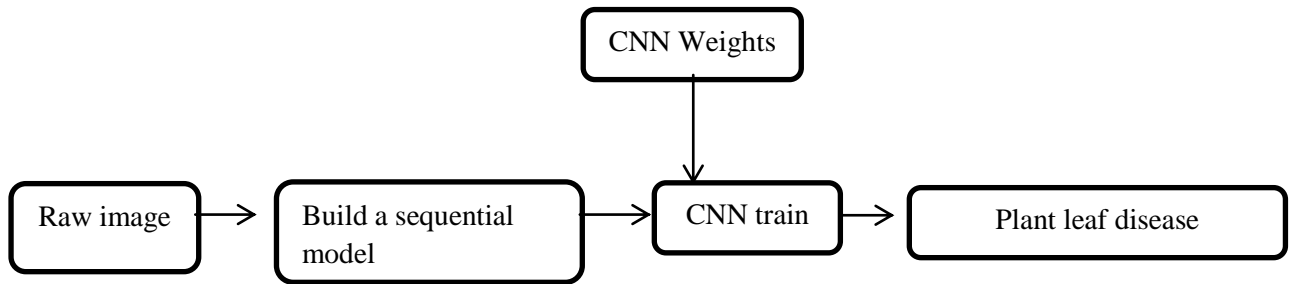


Fig 4: Methodology of the system

The train dataset is used to train the model (CNN) so that it can identify the test image and the disease it has. CNN has different layers that are Dense, Dropout, Activation, Flatten, Convolution2D, and MaxPooling2D. After the model is trained successfully, the software can identify the Plant leaf disease prediction image contained in the dataset. After successful training and preprocessing, comparison of the test image and trained model takes place to predict the Sign language.

CNN Model steps:

Conv2d:

The 2D convolution is a fairly simple operation at heart: you start with a kernel, which is simply a small matrix of weights. This kernel “slides” over the 2D input data, performing an elementwise multiplication with the part of the input it is currently on, and then summing up the results into a single output pixel.

The kernel repeats this process for every location it slides over, converting a 2D matrix of features into yet another 2D matrix of features. The output features are essentially, the weighted sums (with the weights being the values of the kernel itself) of the input features located roughly in the same location of the output pixel on the input layer.

Whether or not an input feature falls within this “roughly same location”, gets determined directly by whether it’s in the area of the kernel that produced the output or not. This means the size of the kernel directly determines how many (or few) input features get combined in the production of a new output feature.

This is all in pretty stark contrast to a fully connected layer. In the above example, we have $5 \times 5 = 25$ input features, and $3 \times 3 = 9$ output features. If this were a standard fully connected layer, you’d have a weight matrix of $25 \times 9 = 225$ parameters, with every output feature being the weighted sum of every single input feature. Convolutions allow us to do this transformation with only 9 parameters, with each output feature, instead of “looking at” every input feature, only getting to “look” at input features coming from roughly the same location. Do take note of this, as it’ll be critical to our later discussion.

MaxPooling2D layer

Downsamples the input along its spatial dimensions (height and width) by taking the maximum value over an input window (of size defined by `pool_size`) for each channel of the input. The window is shifted by `strides` along each dimension.

The resulting output, when using the "valid" padding option, has a spatial shape (number of rows or columns) of: $\text{output_shape} = \text{math.floor}((\text{input_shape} - \text{pool_size}) / \text{strides}) + 1$ (when $\text{input_shape} \geq \text{pool_size}$)

The resulting output shape when using the "same" padding option is: $\text{output_shape} = \text{math.floor}((\text{input_shape} - 1) / \text{strides}) + 1$

Arguments

- `pool_size`: integer or tuple of 2 integers, window size over which to take the maximum. (2, 2) will take the max value over a 2x2 pooling window. If only one integer is specified, the same window length will be used for both dimensions.
- `strides`: Integer, tuple of 2 integers, or None. Strides values. Specifies how far the pooling window moves for each pooling step. If None, it will default to `pool_size`.

- padding: One of "valid" or "same" (case-insensitive). "valid" means no padding. "same" results in padding evenly to the left/right or up/down of the input such that output has the same height/width dimension as the input.
- data_format: A string, one of channels_last (default) or channels_first. The ordering of the dimensions in the inputs. channels_last corresponds to inputs with shape (batch, height, width, channels) while channels_first corresponds to inputs with shape (batch, channels, height, width). It defaults to the image_data_format value found in your Keras config file at ~/.keras/keras.json. If you never set it, then it will be "channels_last".

Input shape

- If data_format='channels_last': 4D tensor with shape (batch_size, rows, cols, channels).
- If data_format='channels_first': 4D tensor with shape (batch_size, channels, rows, cols).

Output shape

- If data_format='channels_last': 4D tensor with shape (batch_size, pooled_rows, pooled_cols, channels).
- If data_format='channels_first': 4D tensor with shape (batch_size, channels, pooled_rows, pooled_cols).

Flatten layer

It is used to flatten the dimensions of the image obtained after convolving it. Dense: It is used to make this a fully connected model and is the hidden layer. Dropout: It is used to avoid over fitting on the dataset and dense is the output layer contains only one neuron which decide to which category image belongs.

Flatten is used to flatten the input. For example, if flatten is applied to layer having input shape as (batch_size, 2,2), then the output shape of the layer will be (batch_size, 4)

Flatten has one argument as follows

```
keras.layers.Flatten(data_format = None)
```

data_format is an optional argument and it is used to preserve weight ordering when switching from one data format to another data format. It accepts either channels_last or channels_first as value. channels_last is the default one and it identifies the input shape as (batch_size, ..., channels) whereas channels_first identifies the input shape as (batch_size, channels, ...)

Dense layer

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

Note: If the input to the layer has a rank greater than 2, then Dense computes the dot product between the inputs and the kernel along the last axis of the inputs and axis 0 of the kernel (using tf.tensordot). For example, if input has dimensions (batch_size, d0, d1), then we create a kernel with shape (d1, units), and the kernel operates along axis 2 of the input, on every sub-tensor of shape (1, 1, d1) (there are batch_size * d0 such sub-tensors). The output in this case will have shape (batch_size, d0, units).

Besides, layer attributes cannot be modified after the layer has been called once (except the trainable attribute). When a popular kwarg input_shape is passed, then keras will create an input layer to insert before the current layer. This can be treated equivalent to explicitly defining an InputLayer.

Arguments

- units: Positive integer, dimensionality of the output space.

- `activation`: Activation function to use. If you don't specify anything, no activation is applied (ie. "linear" activation: $a(x) = x$).
- `use_bias`: Boolean, whether the layer uses a bias vector.
- `kernel_initializer`: Initializer for the kernel weights matrix.
- `bias_initializer`: Initializer for the bias vector.
- `kernel_regularizer`: Regularizer function applied to the kernel weights matrix.
- `bias_regularizer`: Regularizer function applied to the bias vector.
- `activity_regularizer`: Regularizer function applied to the output of the layer (its "activation").
- `kernel_constraint`: Constraint function applied to the kernel weights matrix.
- `bias_constraint`: Constraint function applied to the bias vector.

Input shape

N-D tensor with shape: `(batch_size, ..., input_dim)`. The most common situation would be a 2D input with shape `(batch_size, input_dim)`.

Output shape

N-D tensor with shape: `(batch_size, ..., units)`. For instance, for a 2D input with shape `(batch_size, input_dim)`, the output would have shape `(batch_size, units)`.

Dropout layer

The Dropout layer randomly sets input units to 0 with a frequency of rate at each step during training time, which helps prevent overfitting. Inputs not set to 0 are scaled up by $1/(1 - \text{rate})$ such that the sum over all inputs is unchanged.

Note that the Dropout layer only applies when training is set to True such that no values are dropped during inference. When using `model.fit`, training will be appropriately set to True automatically, and in other contexts, you can set the kwarg explicitly to True when calling the layer.

(This is in contrast to setting `trainable=False` for a Dropout layer. `trainable` does not affect the layer's behavior, as Dropout does not have any variables/weights that can be frozen during training.)

Arguments

- **rate**: Float between 0 and 1. Fraction of the input units to drop.
- **noise_shape**: 1D integer tensor representing the shape of the binary dropout mask that will be multiplied with the input. For instance, if your inputs have shape (batch_size, timesteps, features) and you want the dropout mask to be the same for all timesteps, you can use `noise_shape=(batch_size, 1, features)`.

seed: A Python integer to use as random seed.

Image Data Generator:

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

Training Process:

`train_datagen.flow_from_directory` is the function that is used to prepare data from the `train_dataset` directory. `Target_size` specifies the target size of the image. `Test_datagen.flow_from_directory` is used to prepare test data for the model and all

is similar as above. `fit_generator` is used to fit the data into the model made above, other factors used are `steps_per_epochs` tells us about the number of times the model will execute for the training data.

Epochs:

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

Validation process:

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

4.1 SYSTEM ARCHITECTURE:

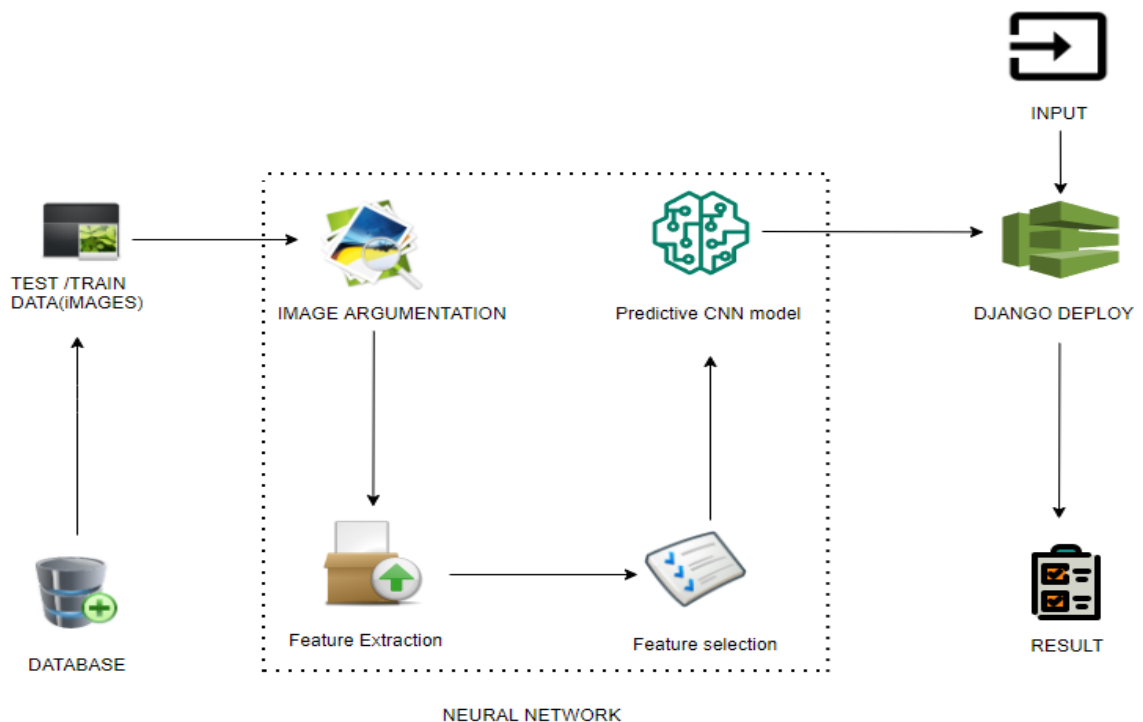


Fig 5: System Architecture

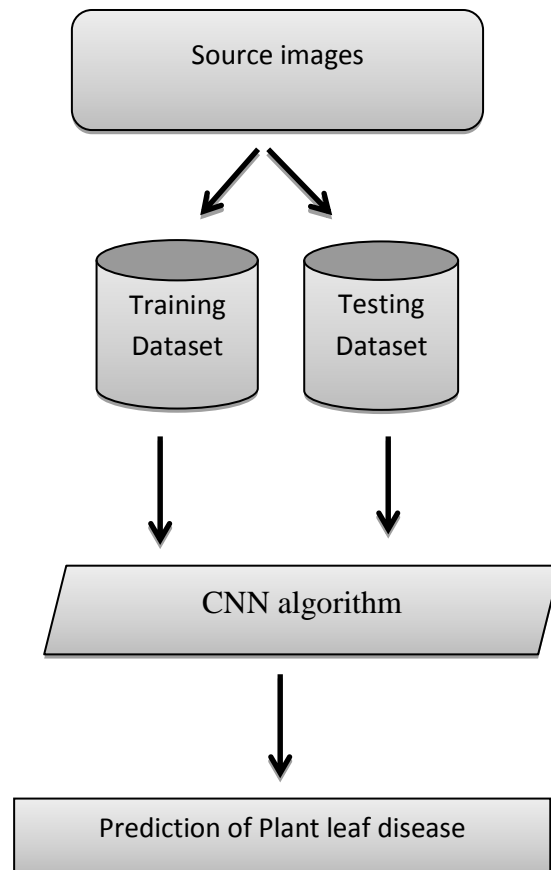


Fig 6: Workflow Diagram

4.1.1 USE CASE DIAGRAM:

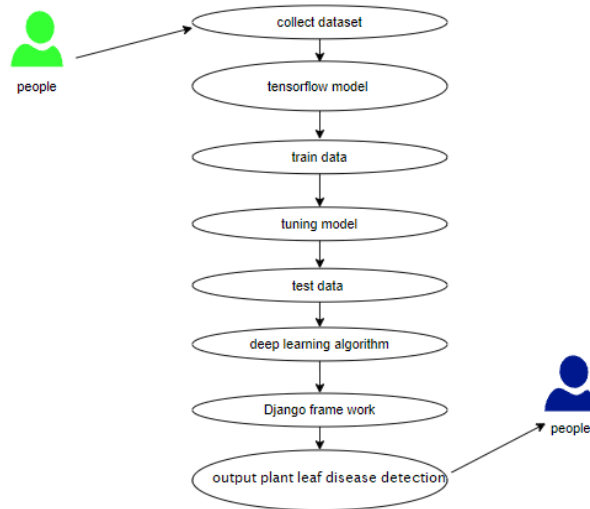


Fig 7: Use Case Diagram

Use case diagrams are considered for high level requirement analysis of a system. So, when the requirements of a system are analyzed the functionalities are captured in use cases. So, it can say that uses cases are nothing but the system functionalities written in an organized manner.

4.1.2. CLASS DIAGRAM:

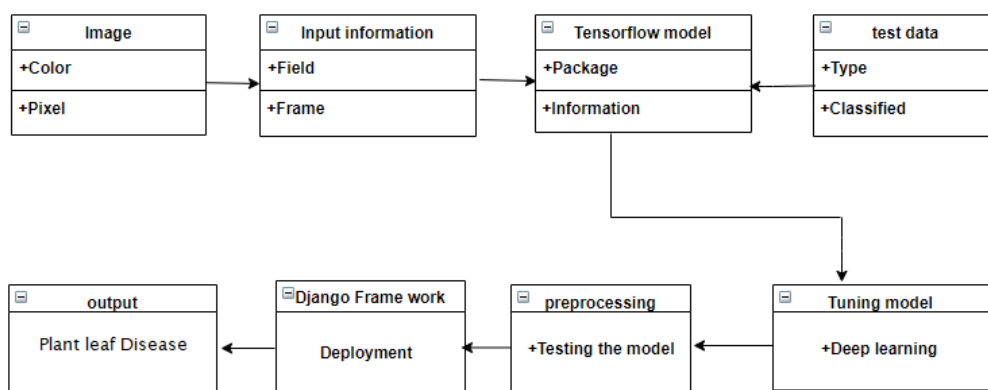


Fig 8: Class Diagram

Class diagram is basically a graphical representation of the static view of the system and represents different aspects of the application. So, a collection of class diagrams represent the whole system. The name of the class diagram should be meaningful to describe the aspect of the system. Each element and their relationships should be identified in advance Responsibility (attributes and methods) of each class should be clearly identified for each class minimum number of properties should be specified and because, unnecessary properties will make the diagram complicated. Use notes whenever required to describe some aspect of the diagram and at the end of the drawing it should be understandable to the developer/coder. Finally, before making the final version, the diagram should be drawn on plain paper and rework as many times as possible to make it correct.

4.1.3. ACTIVITY DIAGRAM:

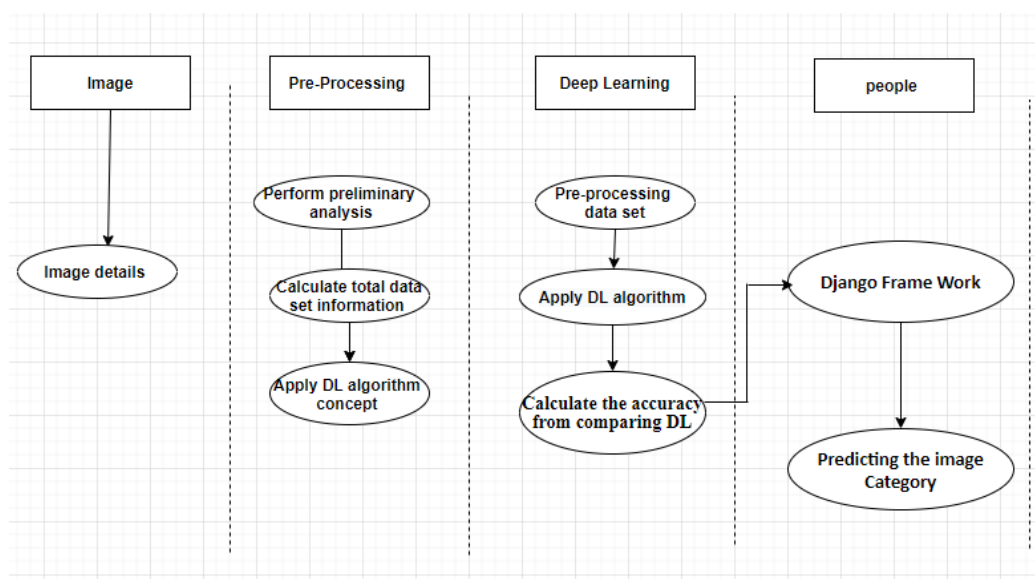


Fig 9: Activity Diagram

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 diagram is some time considered

as the flow chart. Although the diagram looks like a flow chart but it is not. It shows different flow like parallel, branched, concurrent and single.

4.1.4. SEQUENCE DIAGRAM:

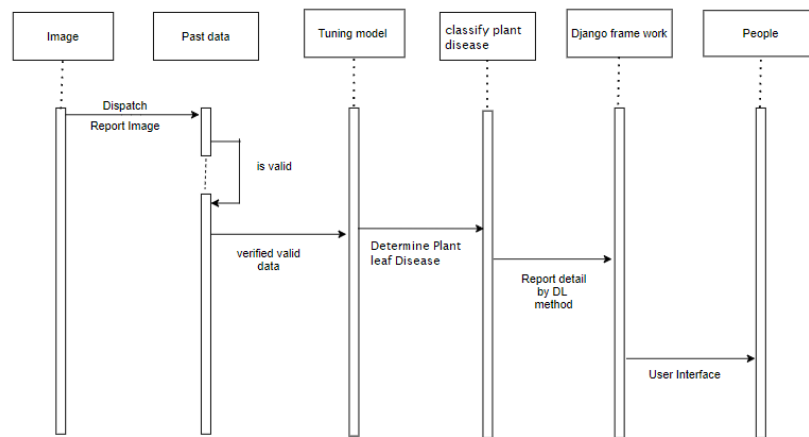


Fig 10: Sequence Diagram

Sequence diagrams model the flow of logic within your system in a visual manner, enabling you both to document and validate your logic, and are commonly used for both analysis and design purposes. Sequence diagrams are the most popular UML artifact for dynamic modelling, which focuses on identifying the behavior within your system. Other dynamic modelling techniques include activity diagramming, communication diagramming, timing diagramming, and interaction overview diagramming. Sequence diagrams, along with class diagrams and physical data models are in my opinion the most important design-level models for modern business application development.

4.1.5. E.R – DIAGRAM

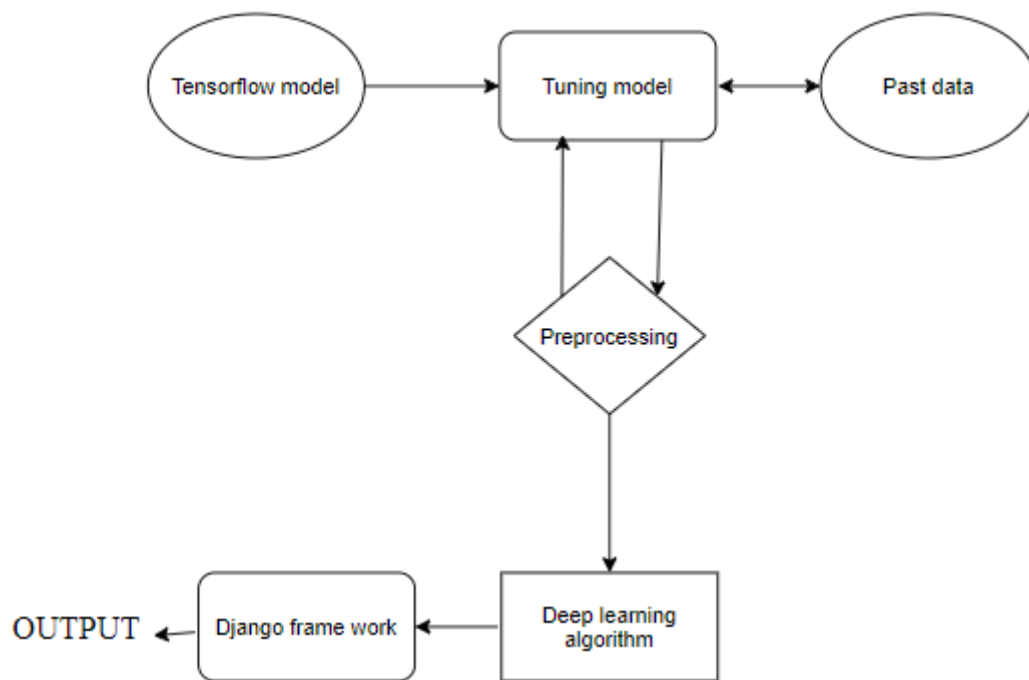


Fig 11: E.R. Diagram

An entity relationship diagram (ERD), also known as an entity relationship model, is a graphical representation of an information system that depicts the relationships among people, objects, places, concepts or events within that system. An ERD is a data modeling technique that can help define business processes and be used as the foundation for a relational database. Entity relationship diagrams provide a visual starting point for database design that can also be used to help determine information system requirements throughout an organization. After a relational database is rolled out, an ERD can still serve as a referral point, should any debugging or business process re-engineering be needed later.

4.1.6. COLLABORATION DIAGRAM:

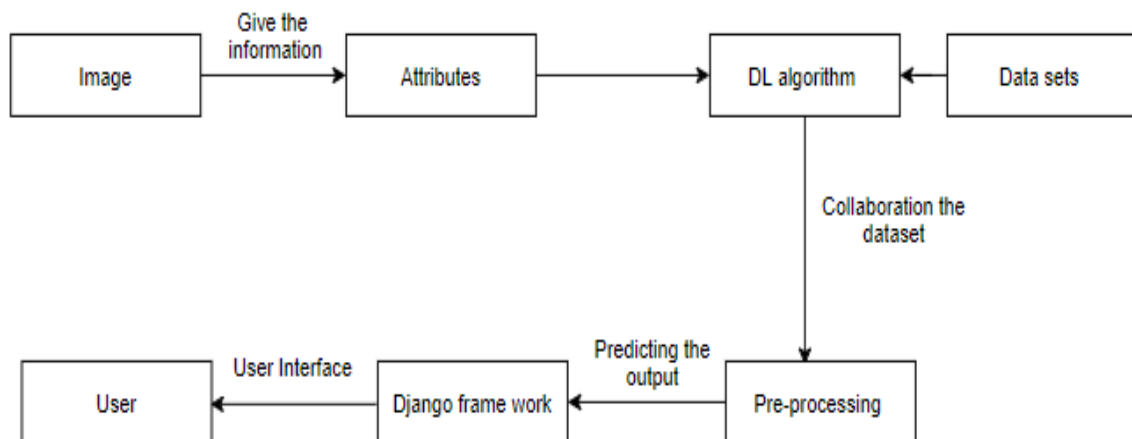


Fig 12: Collaboration Diagram

4.2 TYPES OF CNN:

- AlexNet
- LeNet

4.2.1 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.

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:



Fig 13: 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.

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.

4.2.2 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.

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.

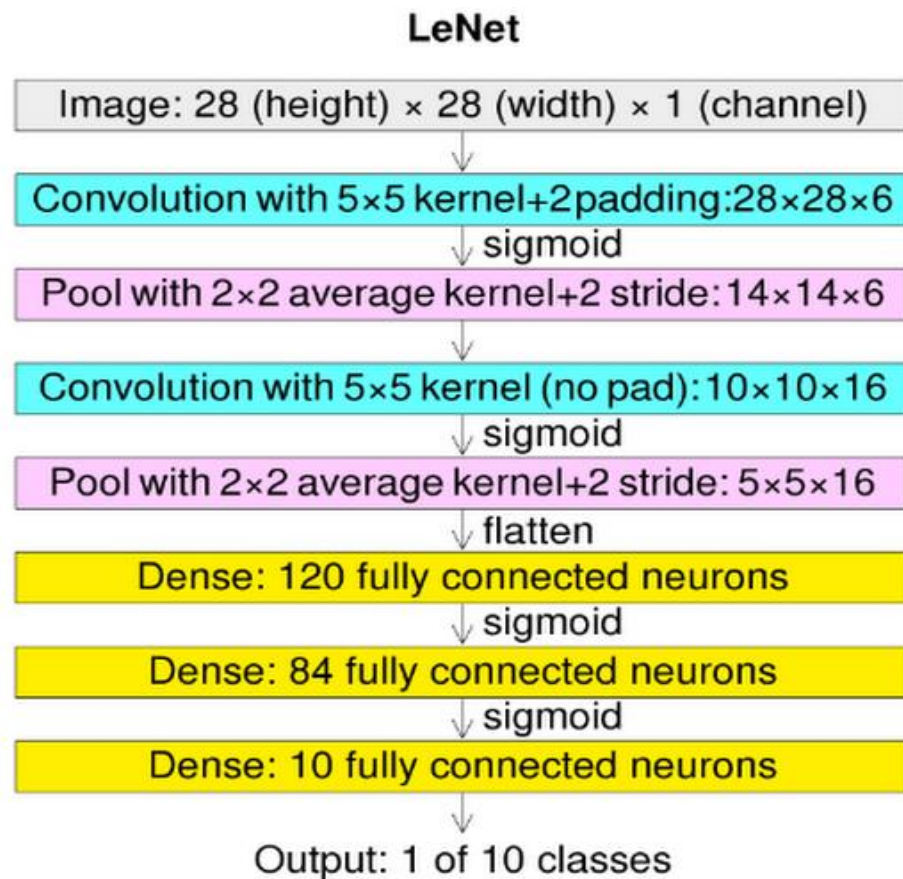


Fig 14: Architecture of LeNet

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.

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. COMPARISION AND ANALYSIS

In this paper, we have discussed about the working of different types of Convnet and its implementation. The use of the relative variety of techniques gives us a decent perspective into the working of our model and helps us finalize on a particular algorithm by comparing them against each other. We measure the performance using metrics like the overall accuracy and loss against the training data and testing data. The overall accuracy we achieve at the end of the model training gives us a good idea of the performance, along with the graph of the overall accuracy as it progresses through the training also gives us an idea about the outliers and not just the end result. The graph provides us with the insights which we could not see clearly with just a glance over the module and presents us with more in-depth idea about how we progress through our data with all the deviations in our sight for better analysis of our model.

To better understand the performance of the different algorithms we have used, the model loss and accuracy graph is given below.

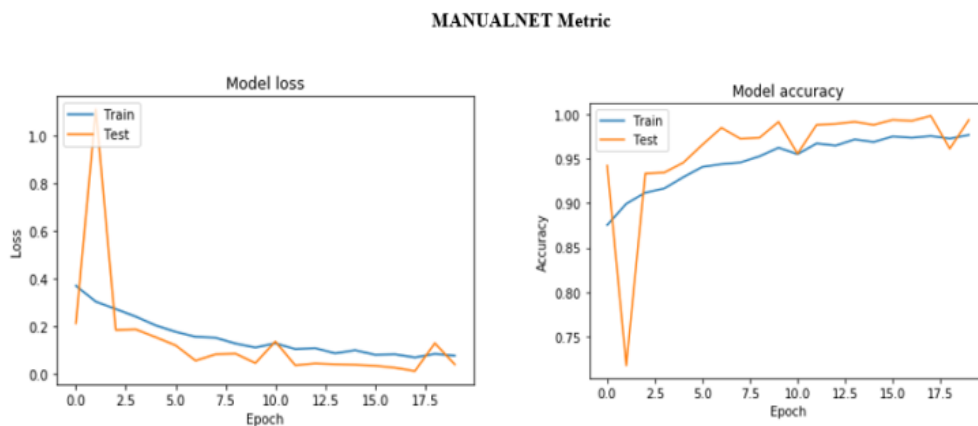


Fig 15: ManualNet Metrics Graph

In the initial phase we used a CNN which we made manually with layers and activation functions which we estimated were good for our data. The metrics tells us that at the end of our training we have a model accuracy of 92% which states that our model is performing very well for a deep learning model, but the initial outliers in our test data tells us that the model is having some problem achieving the desired results due to some discrepancy in our test data.

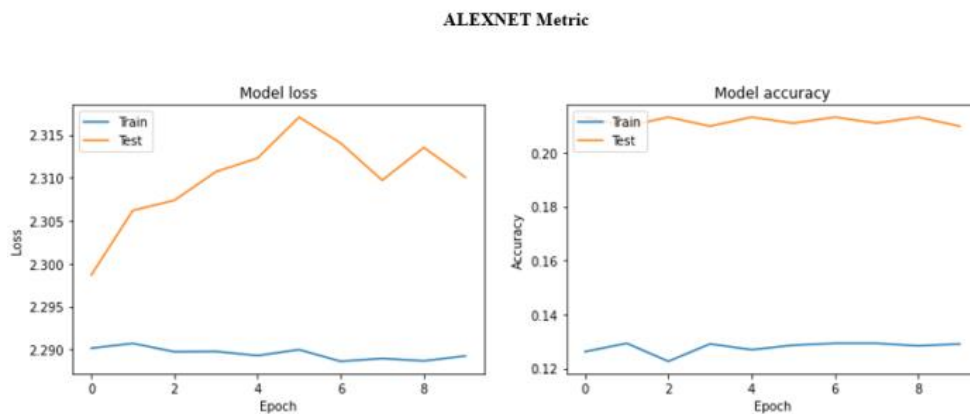


Fig 16: AlexNet Metrics Graph

To evaluate our initial model against other models which are available we made the use of AlexNet since its one of the most relevant models in the field of CNN. But as we train our model we learn that it does not converge on our desired results. The model loss on our test data keeps on increasing as we train our model and our model accuracy for the training data also flat lines with very little accuracy. This tells us that AlexNet is a very bad algorithm for our given data with both our test and train data not converging and flat lining with both of them parallel to each other.

LENET-5 Metrics

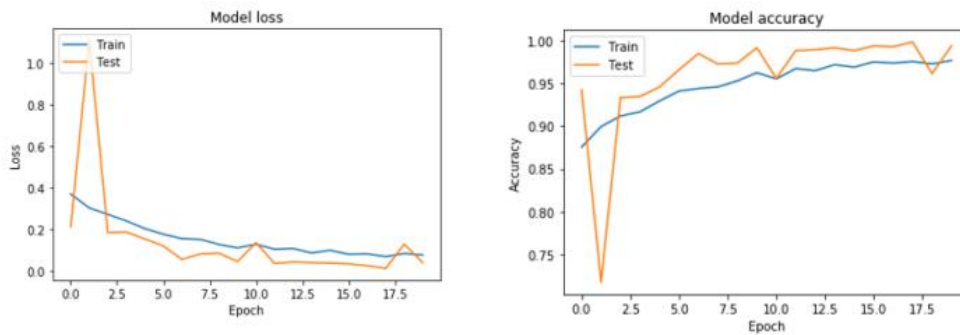


Fig 17: LEnet-5 Metrics Graph

Since our second model gave no insight into our initial model, we make use of another popular Deep Learning model called LeNet-5. In this we can see that our model performs very well on our data and it as we progress through the training phase the model loss flatlines under 10%, also our model accuracy has a gradual increase as we crunch through our dataset and it flatlines above 95% as we come to the end of our model training. There are certain outliers in our test data, but we can rule it out as a discrepancy in our dataset as we had the same outlier in our initial model.

At the end of our analysis we conclude that our initial model along with the final model i.e. LeNet-5 has worked as we desired. But at the end we choose LeNet-5 as our working model as it has a higher accuracy and lower loss percentage. In the initial model i.e., our ManualNet we have a model loss of around 25% which is not reliable in comparison to LeNet-5 which has an overall model loss of 5%. Thus, we conclude that LeNet-5 would be the best fit model for our final proposal.

6. SUMMARY

After comparing and analyzing the above we deployed our model in Django framework and designed our web app using HTML and CSS. We have tried to design our interface in a way so that everyone can use the resource without having the proper about deep learning. The Choose button helps you in providing the input whereas the upload button starts the processing of the image. After successful importation of image, we get the result that the leaf is healthy or not.

CONCLUSION

The proposed system for classifying the different crop disease has a good accuracy and gives us a good result. The system has potential to reduce the burden of the farmers as well as researchers as it acts as an early detector for the crops. This application can also reduce the loss of crops as it can pre-emptively give warnings as well as help new farmers and researchers from making a mistake by double checking their doubts. Further, future iteration can add more diseases and better detection algorithm.

In this project, a research to classify Plant leaf Disease Classification over static facial 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 Plant 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.

OUTPUT

```
M1.ipynb M2.ipynb M3.ipynb about.md X
C:\Users> vikas > Desktop > PLANT LEAF > doc > about.md > ## Model
9
10 Trained to identify 14 classes for Disease Detection and 10 classes for Disease Classification
11
12 - Disease Classification Classes
13
14     - Cherry_healthy
15     - Cherry_Powdery_mildew
16     - Grape_Black_rot
17     - Grape_Esca_Black_Measles
18     - Grape_healthy
19     - Grape_Leaf_blight_Isariopsis_Leaf_Spot
20     - Apple_Disease
21     - Apple_healthy
22     - Peach_Disease
23     - Peach_Healthy
24     - Strawberry_healthy
25     - Strawberry_Disease
26
27
28
29 - Disease Detection Classes
30
31     - Apple__Apple_scab
32     - Apple__Black_rot
33     - Apple__Cedar_apple_rust
34     - Apple__healthy
35     - Cherry__healthy
36     - Cherry__Powdery_mildew
37     - Grape__Black_rot
38     - Grape__Esca_Black_Measles
39     - Grape__healthy
40     - Grape__Leaf_blight_Isariopsis_Leaf_Spot
41     - Peach__Bacterial_spot
42     - Peach__healthy
43     - Strawberry__healthy
44     - Strawberry__Leaf_scorch
45
46 ---
Ln 26, Col 1 Tab Size: 4 UTF-8 CRLF Markdown Go Live kites: unsup
```

Fig 18: Output Classes

FUTURE WORK

This work can be further improved by adding more data into our dataset. Furthermore, with the advancement and proposal of more algorithms we can improve our model and update it if required. Also, for our web application we can make it more interactive by providing important knowledge about the plant disease along with measures to prevent them. Lastly, making our model accessible to more people can make it such that they can further develop on our work.

7. APPENDICES:

7.1 SOURCE CODE:

To build a model for training and testing:

```
import os
import numpy as np # linear algebra
import matplotlib.pyplot as plt

# DL framework - 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, MaxPooling2D, 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_Apple leaf black rot = 'Dataset/train/Apple leaf black rot '
dir_name_train_Apple leaf Healthy = 'Dataset/train/Apple leaf Healthy'
```

```
dir_name_train_Cherry leaf Powdery mildew = 'Dataset/train/Cherry leaf  
Powdery mildew'
```

```
dir_name_train_Cherry leaf Healthy = 'Dataset/train/Cherry leaf Healthy'
```

```
dir_name_train_Grape leaf black rot = 'Dataset/train/Grape leaf black rot'
```

```
dir_name_train_Grape leaf Healthy = 'Dataset/train/Grape leaf Healthy'
```

```
dir_name_train_Peach leaf bacterial spot = 'Dataset/train/Peach leaf  
bacterial spot'
```

```
dir_name_train_Peach leaf Healthy = 'Dataset/train/Peach leaf Healthy'
```

```
dir_name_train_Strawberry leaf scorch = 'Dataset/train/Strawberry leaf  
scorch'
```

```
dir_name_train_Strawberry leaf Healthy = 'Dataset/train/Strawberry leaf  
Healthy'
```

```
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=(80,40))  
    for idx,img_path in enumerate(item_files):  
        plt.subplot(7,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=[for 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

```

forinfiles:

```

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 Apple leaf black rot:")
print("")
Images_details(dir_name_train_Apple leaf black rot)
print("")
plot_images(dir_name_train_Apple leaf black rot, 10)

```

```

print("")
print("Trained data for Apple leaf Healthy:")
print("")
Images_details(dir_name_train_Apple leaf Healthy)
print("")
plot_images(dir_name_train_Apple leaf Healthy, 10)

```

```

print("")
print("Trained data for Cherry leaf Powdery mildew:")
print("")
Images_details(dir_name_train_Cherry leaf Powdery mildew)
print("")
plot_images(dir_name_train_Cherry leaf Powdery mildew, 10)

```

```

print("")
print("Trained data for Cherry leaf Healthy:")
print("")
Images_details(dir_name_train_Cherry leaf Healthy)
print("")
plot_images(dir_name_train_Cherry leaf Healthy, 10)

```



```

print("")
print("Trained data for Grape leaf black rot:")
print("")
Images_details(dir_name_train_Grape leaf black rot)
print("")
plot_images(dir_name_train_Grape leaf black rot, 10)

print("")
print("Trained data for Grape leaf Healthy:")
print("")
Images_details(dir_name_train_Grape leaf Healthy)
print("")
plot_images(dir_name_train_Grape leaf Healthy, 10)

print("")
print("Trained data for Peach leaf bacterial spot:")
print("")
Images_details(dir_name_train_Peach leaf bacterial spot)
print("")
plot_images(dir_name_train_Peach leaf bacterial spot, 10)

print("")
print("Trained data for Peach leaf Healthy:")
print("")
Images_details(dir_name_train_Peach leaf Healthy)
print("")
plot_images(dir_name_train_Peach leaf Healthy, 10)

print("")
print("Trained data for Strawberry leaf scorch:")
print("")
Images_details(dir_name_train_Strawberry leaf scorch)
print("")
plot_images(dir_name_train_Strawberry leaf scorch, 10)

print("")
print("Trained data for Strawberry leaf Healthy:")
print("")
Images_details(dir_name_train_Strawberry leaf Healthy)
print("")
plot_images(dir_name_train_Strawberry leaf Healthy, 10)

```

```

Classifier=Sequential()
Classifier.add(Convolution2D(32,(3,3),input_shape=(128,128,3),activation='relu'))
Classifier.add(MaxPooling2D(pool_size=(2,2)))
Classifier.add(Flatten())
Classifier.add(Dense(38,activation='relu'))

```

```

Classifier.add(Dense(4,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=(128,128),batch_size=32,class_mode='categorical')
test_set=test_datagen.flow_from_directory('dataset/Test',target_size=(128,128),batch_size=32,class_mode='categorical')

```

```

img_dims=150
epochs=10
batch_size=32

```

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()
```

Module 2:

DL framework - 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 Input
```

```
from tensorflow.keras.layers import Dense
```

```
from tensorflow.keras.layers import Flatten
```

```
from tensorflow.keras.layers import Conv2D
```

```
from tensorflow.keras.layers import MaxPooling2D
```

```
from tensorflow.keras.layers import Dropout
```

```
from tensorflow.keras.layers import LeakyReLU
```

```
from tensorflow.keras.layers import Activation
```

```
from tensorflow.keras.optimizers import Adam
```

```
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 warnings  
warnings.filterwarnings('ignore')
```

```
model = Sequential()  
# 1st Convolutional Layer  
model.add(Conv2D(filters=96, input_shape=(224,224,3),  
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(4))
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,zo
om_range=0.2,horizontal_flip=True)
test_datagen=ImageDataGenerator(rescale=1./255)

training_set=train_datagen.flow_from_directory('dataset/Train',target_siz
e=(224,224),batch_size=32,class_mode='categorical')
test_set=test_datagen.flow_from_directory('dataset/Test',target_size=(2
24,224),batch_size=32,class_mode='categorical')

img_dims = 150
epochs = 1
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)

```

```

import matplotlib.pyplot as plt

```

```

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()

```

```

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

```

Module 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=(225,225,3),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(4,activation='softmax'))
```

```
Classifier.compile(optimizer='rmsprop',loss='categorical_crossentropy',m  
etrics=['accuracy'])
```

```
Classifier.summary()
```

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

```
train_datagen=ImageDataGenerator(rescale=1./255,shear_range=0.2,zo  
om_range=0.2,horizontal_flip=True)
```

```
test_datagen=ImageDataGenerator(rescale=1./255)
```

```
training_set=train_datagen.flow_from_directory('dataset/Train',target_size=(225,225),batch_size=32,class_mode='categorical')
```

```
test_set=test_datagen.flow_from_directory('dataset/Test',target_size=(225,225),batch_size=32,class_mode='categorical')
```

```
from IPython.display import display
```

```
img_dims=150  
epochs=60  
batch_size=32
```

```
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('e.h5')
```

```
from keras.models import load_model
```

```
model=load_model('e.h5')
```

```
import numpy as np
```

```
from tensorflow.keras.preprocessing import image  
test_image=image.load_img('c5.jpg',target_size=(225,225))
```

```
import matplotlib.pyplot as plt  
img=plt.imshow(test_image)
```



```
test_image=image.img_to_array(test_image)
```

```
test_image=np.expand_dims(test_image,axis=0)
```

```
result=model.predict(test_image)
```

```
result
```

```
prediction=result[0]
```

```
classes=training_set.class_indices
```

```
classes
```

```
prediction=list(prediction)
```

```
prediction
```

```
classes=['Apple leaf black rot ','Apple leaf Healthy','Cherry leaf Powdery  
mildew','Cherry leaf Healthy','Grape leaf black rot','Grape leaf  
Healthy','Peach leaf bacterial spot','Peach leaf Healthy','Strawberry leaf  
scorch','Strawberry leaf Healthy']
```

```
    output = zip(classes, prediction)
```

```
    output = dict(output)
```

```
    if output['Apple leaf black rot '] == 1.0:
```

```
        a="Apple leaf black rot "
```

```
    elif output['Apple leaf Healthy'] == 1.0:
```

```
        a="Apple leaf Healthy"
```

```
    elif output['Cherry leaf Powdery mildew'] == 1.0:
```

```
        a="Cherry leaf Powdery mildew"
```

```
elif output['Cherry leaf Healthy'] == 1.0:
    a="Cherry leaf Healthy"
elif output['Grape leaf black rot'] == 1.0:
    a="Grape leaf black rot"
elif output['Grape leaf Healthy'] == 1.0:
    a="Grape leaf Healthy"
elif output['Peach leaf bacterial spot'] == 1.0:
    a="Peach leaf bacterial spot"
elif output['Peach leaf Healthy'] == 1.0:
    a="Peach leaf Healthy"
elif output['Strawberry leaf scorch'] == 1.0:
    a="Strawberry leaf scorch"
elif output['Strawberry leaf Healthy'] == 1.0:
    a="Strawberry leaf Healthy"
```

PyCharm:

Views.py

```
from django.shortcuts import render

from django.http import HttpResponseRedirect
from django.urls import reverse_lazy
from django.views.generic import TemplateView
from employee.forms import EmployeeForm

from django.views.generic import DetailView
from employee.models import Employee
```

```

class EmployeeImage(TemplateView):
    form = EmployeeForm
    template_name = 'emp_image.html'

    def post(self, request, *args, **kwargs):
        form = EmployeeForm(request.POST, request.FILES)

        if form.is_valid():
            obj = form.save()

            return HttpResponseRedirect(reverse_lazy('emp_image_display',
kwargs={'pk': obj.id}))

        context = self.get_context_data(form=form)
        return self.render_to_response(context)

    def get(self, request, *args, **kwargs):
        return self.post(request, *args, **kwargs)

class EmpImageDisplay(DetailView):
    model = Employee
    template_name = 'emp_image_display.html'
    context_object_name = 'emp'

def sign language(request):
    result1 = Employee.objects.latest('id')
    import numpy as np
    import tensorflow as tf
    from tensorflow import keras
    import h5py
    models = keras.models.load_model('C:/Users/SPIRO73-
PYTHON/Desktop/smb/Deep_Learning/Plant leaf disease
prediction/Deploy/employee/e.h5')
    from tensorflow.keras.preprocessing import image
    test_image = image.load_img('C:/Users/SPIRO73-

```

```

PYTHON/Desktop/smb/Deep_Learning/Plant leaf disease
prediction/Deploy/media/" + str(result1), target_size=(225, 225))
test_image = image.img_to_array(test_image)
test_image = np.expand_dims(test_image, axis=0)
result = models.predict(test_image)
prediction = result[0]
prediction = list(prediction)
classes=['Apple leaf black rot ', 'Apple leaf Healthy', 'Cherry leaf Powdery
mildew', 'Cherry leaf Healthy', 'Grape leaf black rot', 'Grape leaf
Healthy', 'Peach leaf bacterial spot', 'Peach leaf Healthy', 'Strawberry leaf
scorch', 'Strawberry leaf Healthy']

output = zip(classes, prediction)
output = dict(output)

if output['Apple leaf black rot '] == 1.0:
    a="Apple leaf black rot "
elif output['Apple leaf Healthy'] == 1.0:
    a="Apple leaf Healthy"
elif output['Cherry leaf Powdery mildew'] == 1.0:
    a="Cherry leaf Powdery mildew"
elif output['Cherry leaf Healthy'] == 1.0:
    a="Cherry leaf Healthy"
elif output['Grape leaf black rot'] == 1.0:
    a="Grape leaf black rot"
elif output['Grape leaf Healthy'] == 1.0:
    a="Grape leaf Healthy"
elif output['Peach leaf bacterial spot'] == 1.0:
    a="Peach leaf bacterial spot"
elif output['Peach leaf Healthy'] == 1.0:

```

```

    a="Peach leaf Healthy"

elif output['Strawberry leaf scorch'] == 1.0:

    a="Strawberry leaf scorch"

elif output['Strawberry leaf Healthy'] == 1.0:

    a="Strawberry leaf Healthy"
return render(request, "result.html", {"out": a})

```

emp_image.html

```

<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>image upload example</title>
</head>
<style>
    label
    {
font-size: 20px;
color:purple;
font-family: Algerian;
    }
body
    {
    background: url(../static/image/emo.png);
    background-repeat: no-repeat;
    background-position: center;
    background-size: 100% 100%;
    min-height: 100vh;
    }
<!-- button-->
<!--{-->

```

```

<!--font-size: 20px;-->
<!--font-family: wide latin;-->
<!--color:black;-->
<!--background-color: green;-->
<!-- box-shadow: 5px 5px blue, 10px 10px red, 15px 15px green;-->
<!--}-->
.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;
}
#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: blink 0.2s;
animation-iteration-count: infinite;
}

@keyframes blink { 50% { border-color:yellow ; }
}
button
{
font-family: Algerian;
font-size: 35px;
font-weight: bold;
}
.alerts-border
{
width:500px;
margin-top:7%;
margin-left:28%;
padding: 10px 10px 10px 10px;
</style>
<body style="background-color: lightblue;">
<center><h2 class="blink_me">PLANT LEAF DISEASE PREDICTION
USING DEEP LEARNING

```



```

background: url(../static/image/emo.gif));
background-repeat: no-repeat;
background-position: center;
background-size: cover;
-webkit-background-size: cover;
-moz-background-size: cover;
-o-background-size: cover;
min-height: 100vh;
}
button
{
font-size: 20px;
font-family: wide latin;
color:black;
background-color: green;
box-shadow: 5px 5px blue, 10px 10px red, 15px 15px green;
}
#ss
{
font-size: 20px;
color:#20fc03;
background-color: black;
font-family: Times new roman;
}
h2
{
font-size: 20px;
color:black;
background-color: white;

```

```

font-family: Times new roman;
}
a
{
font-size: 20px;
color:#FA2204 ;
font-family: Times new roman;
}
.alerts-border {
    border: 4px #ff0000 dashed;

    animation: blink 0.2s;
    animation-iteration-count: infinite;
}
@keyframes blink { 50% { border-color:yellow ; }
}
</style>
<body>
{% load static %}
<center>

<br>
<a href="{% url 'sign language' %}">Result</a>&#160;&#160;&#160;
<a href="{% url 'home' %}">Go Back!!!</a>
</center>
</body>
</html>

```

result.html

```
{% load static %}
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>PLANT LEAF DISEASE PREDICTION OUTPUT RESULT</title>
</head>
<style>
  label
  {
font-size: 20px;
color:red;
font-family: Algerian;
}
body
{
  background: url(../static/image/emo1.png);
  background-repeat: no-repeat;
  background-position: center;
  background-size: cover;
  -webkit-background-size: cover;
  -moz-background-size: cover;
  -o-background-size: cover;
  min-height: 100vh;
}
  button
  {
font-size: 20px;
font-family: wide latin;
color:black;
background-color: green;
  box-shadow: 5px 5px blue, 10px 10px red, 15px 15px green;
}
  #ss
  {
font-size: 20px;
```

```

color:#20fc03;
background-color: black;
font-family: Times new roman;
}
.blink_me {
  animation: blinker 2s linear infinite;
}
@keyframes blinker {
  50% {
    opacity: 0;
  }
}
h2
{
background-color: #101010;
font-family: Algerian;
font-size: 33px;
letter-spacing: 3px;
color:#56ff00;
}
h1,h3
{
font-family: Algerian;
font-size: 33px;
letter-spacing: 3px;
color:red;
}
a
{
font-size: 20px;
color:black ;
font-family: Times new roman;
}
</style>
<body>
<center><h2 class="blink_me">PLANT LEAF DISEASE PREDICTION
USING ARTIFICIAL NEURAL

```

```
NETWORK</h2></center><br><br><br><br>
<marquee
direction="down"><center><b><h1>{{out}}</h1></b></center></marquee>
e>

<h3><a href="{% url 'home' %}">Go Back!!!</a></h3>

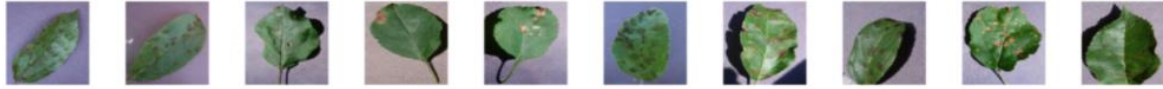
</body>
</html>
```

7.2 SCREENSHOTS:

Apple leaf black rot

Trained data for Apple Disease:

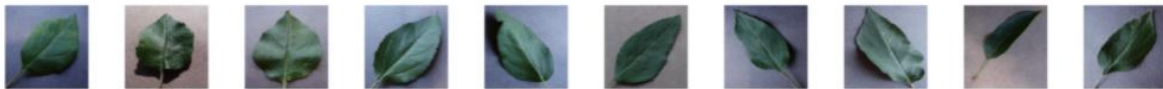
```
===== Images in: dataset/Train/Apple Disease
images_count: 415
min_width: 256
max_width: 256
min_height: 256
max_height: 256
```



Apple leaf Healthy

Trained data for Apple healthy:

```
===== Images in: dataset/Train/Apple healthy
images_count: 264
min_width: 256
max_width: 256
min_height: 256
max_height: 256
```



Cherry leaf Powdery mildew

Trained data for Cherry Disease:

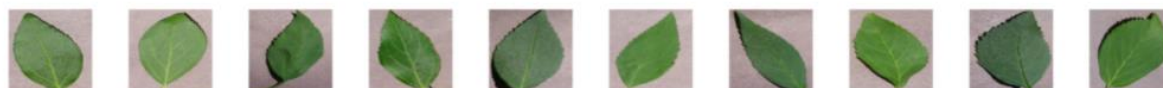
```
===== Images in: dataset/Train/Cherry Disease
images_count: 490
min_width: 256
max_width: 256
min_height: 256
max_height: 256
```



Cherry leaf Healthy

Trained data for Cherry healthy:

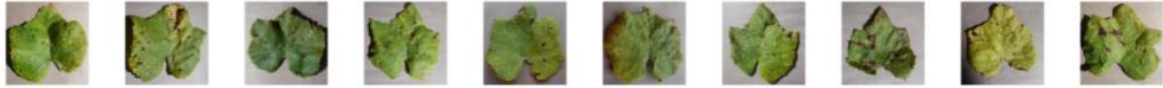
```
===== Images in: dataset/Train/Cherry healthy
images_count: 542
min_width: 256
max_width: 256
min_height: 256
max_height: 256
```



Grape leaf black rot

Trained data for Grape Disease:

```
===== Images in: dataset/Train/Grape Disease
images_count: 454
min_width: 256
max_width: 256
min_height: 256
max_height: 256
```



Grape leaf Healthy

Trained data for Grape healthy:

```
===== Images in: dataset/Train/Grape healthy
images_count: 423
min_width: 256
max_width: 256
min_height: 256
max_height: 256
```



Peach leaf bacterial spot

Trained data for Peach Bacterial:

```
===== Images in: dataset/Train/Peach Bacterial
images_count: 385
min_width: 256
max_width: 256
min_height: 256
max_height: 256
```



Peach leaf Healthy

Trained data for Peach healthy:

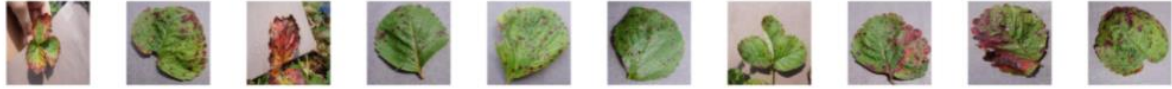
```
===== Images in: dataset/Train/Peach healthy
images_count: 360
min_width: 256
max_width: 256
min_height: 256
max_height: 256
```



Strawberry leaf scorch

Trained data for Strawberry Disease:

```
===== Images in: dataset/Train/Strawberry Disease
images_count: 427
min_width: 256
max_width: 256
min_height: 256
max_height: 256
```



Strawberry leaf Healthy

Trained data for Strawberry healthy:

```
===== Images in: dataset/Train/Strawberry healthy
images_count: 456
min_width: 256
max_width: 256
min_height: 256
max_height: 256
```



Fig 19: Dataset Screenshots

OUTPUT SCREENSHOTS:

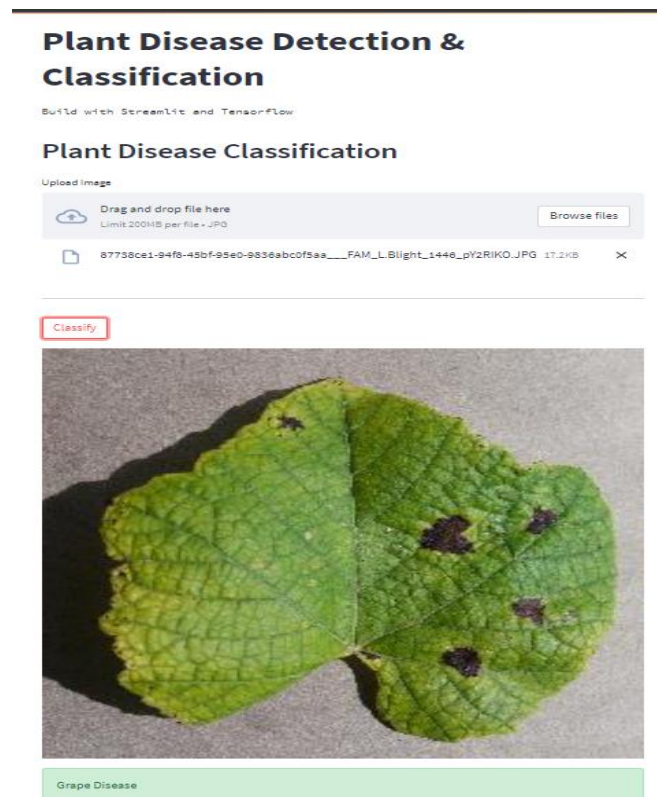
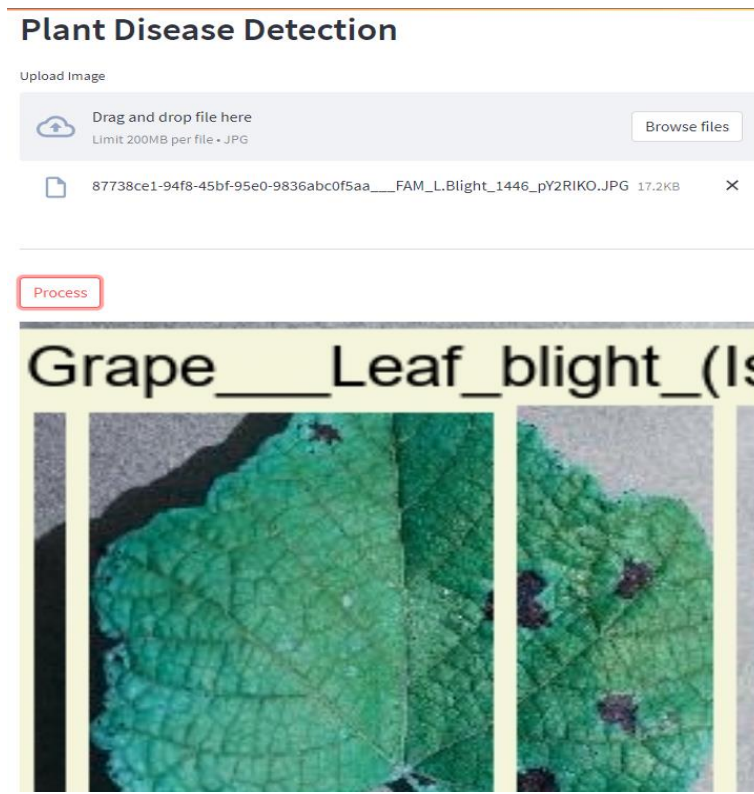


Fig 20: Output Screenshots

7.1 PUBLICATION AND PLAGIARISM REPORT



KL17

ORIGINALITY REPORT

4%

SIMILARITY INDEX

3%

INTERNET SOURCES

2%

PUBLICATIONS

1%

STUDENT PAPERS

PRIMARY SOURCES

1

Prathima Devadas, G. Kalaiarasi, M. Selvi.
"Intensity based Image Registration on Brain
MRI Images", 2020 Second International
Conference on Inventive Research in
Computing Applications (ICIRCA), 2020
Publication

1%

2

www.ijert.org
Internet Source

1%

3

Submitted to University of East London
Student Paper

<1%

4

"Proceeding of First Doctoral Symposium on
Natural Computing Research", Springer
Science and Business Media LLC, 2021
Publication

<1%

5

Lili Li, Shujuan Zhang, Bin Wang. "Plant
Disease Detection and Classification by Deep
Learning—A Review", IEEE Access, 2021
Publication

<1%

6

nitech.repo.nii.ac.jp
Internet Source

<1%

7.3 REFERENCE:

1. Simraneet Kaur, Getanajali Babar, Navneet Sandhu, Dr. Gagan Jindal, 'Various Plant disease detection using Image Processing Methods', June 2019.
2. Er. Varinderjit Kaur, Dr. Ashish Oberoi, 'Wheat disease detection using svm classifier', Aug 2018.
3. Barbedo, 'Plant Disease Identification from individual lesions spots using deep learning', Apr 2019.
4. K. Naga Subramanian, A.K. Singh, A. Singh, S. Sarkar and Ganpath Subramanian, "Usefulness of interpretability methods to explain deep learning-based plant stress phenotyping", Jul 2020.
5. W. Yang, C. Yang, Z. Hao, C. Xie, and M. Li, "Diagnosis of plant cold damage based on hyperspectral imaging and convolutional neural network," IEEE Access, vol. 7, pp. 118239–118248, 2019.
6. K. Nagasubramanian, S. Jones, A. K. Singh, S. Sarkar, A. Singh, and B. Ganapathysubramanian, "Plant disease identification using explainable 3D deep learning on hyperspectral images".
7. N. Zhang, G. Yang, Y. Pan, X. Yang, L. Chen, and C. Zhao, "A review of advanced technologies and development for hyperspectral-based plant disease detection in the past three decades," Remote Sens., vol. 12, no. 19, Sep. 2020, Art. no. 3188.
8. J. Chen, J. Chen, D. Zhang, Y. Sun, and Y. A. Nanekharan, "Using deep transfer learning for image-based plant disease identification," Comput. Electron. Agricult., vol. 173, Jun. 2020, Art. no. 105393.
9. M. Agarwal, A. Singh, S. Arjaria, A. Sinha, and S. Gupta, "ToLeD: Tomato leaf disease detection using convolution neural network," Procedia Comput. Sci., vol. 167, pp. 293–301, Jan. 2020.
10. G. Hu, H. Wu, Y. Zhang, and M. Wan, "A low shot learning method for tea leaf's disease identification," Comput. Electron. Agricult., vol. 163, Aug. 2019, Art. no. 104852.
11. D. Das and C. S. G. Lee, "A two-stage approach to few-shot learning for image recognition," IEEE Trans. Image Process., vol. 29, no. 5, pp. 3336–3350, Dec. 2020.
12. J.-H. Li, L.-J. Lin, and K. Tian, "Detection of leaf diseases of balsam pear in the field based on improved faster R-CNN," Trans. Chin. Soc. Agricult. Eng., vol. 36, no. 12, pp. 179–185, Jun. 2020.
13. K. Nagasubramanian, A. K. Singh, A. Singh, S. Sarkar, and B. Ganapathysubramanian, "Usefulness of interpretability methods to explain deep learning-based plant stress phenotyping," Comput. Sci., vol. 4, pp. 18–32, Jul. 2020.
14. J. G. A. Barbedo, "Plant disease identification from individual lesions and spots using deep learning," Biosyst. Eng., vol. 180, pp. 96–107, Apr. 2019.

15. S. H. Lee, H. Goëau, P. Bonnet, and A. Joly, "New perspectives on plant disease characterization based on deep learning," *Comput. Electron. Agricult.*, vol. 170, Mar. 2020, Art. no. 105220.
16. X.-R. Li, S.-Q. Li, and B. Liu, "Apple leaf disease detection method based on improved faster R-CNN," *Comput. Eng.*, vol. 46, no. 11, pp. 59–64, Nov. 2020.
17. J. Chen, J. Chen, D. Zhang, Y. Sun, and Y. A. Nanekaran, "Using deep transfer learning for image-based plant disease identification," *Comput. Electron. Agricult.*, vol. 173, Jun. 2020, Art. no. 105393.
18. S.-N. Ren, Y. Sun, H.-Y. Zhang, and L.-X. Guo, "Plant disease identification for small sample based on one-shot learning," *Jiangsu J. Agricult. Sci.*, vol. 35, no. 5, pp. 1061–1067, May 2019.