Fertilizers Recommendation System for Disease Prediction

A Project report submitted in partial fulfilment of 7th semester in degree of

BACHELOR OF ENGINEERING
IN

ELECTRONICS AND COMMUNICATION ENGINEERING

Submitted by

Team ID: PNT2022TMID24449

Dinesh Shithik Varshan A	113319106021
Kishore H	113319106042
Venkatesh P S	113319106083
Velmurugan R	113319106082
Srikanth Y	113319106077

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING
VELAMMAL INSTITUTE OF TECHNOLOGY
THIRUVALLUR-DISTRICT

ANNA UNIVERSITY: CHENNAI 600025

NOV-2022

VELAMMAL INSTITUTE OF TECHNOLOGY
THIRUVALLUR-DISTRICT
(A Affiliated College of Anna University, Chennai)

BONAFIDE CERTIFICATE

Certified that this project report ”**Fertilizers Recommendation System For Disease Prediction”** is the bona fide record work done by **Mr. DINESH SHITHIK VARSHAN A(113319106021**), **Mr. KISHORE H(113319106042**), **Mr. SRIKANTH Y**(**113319106077**), **Mr. VENKATESH P S (113319106083) and Mr. VELMURUGAN R(113319106082)** for **“HX 8001 PROFESSIONAL READINESS FOR INNOVATION, EMPLOYABILITY AND ENTREPRENEURSHIP”** in **VII** semester of **B.E.,** degree course **in Electronics and Communication Engineering** branch during the academic year of 2022 - 2023.

ACKNOWLEDGEMENT

The satisfactions that accompany the successful completion of any task would be incomplete without mentioning the people who made it possible by their constant guidance and encouragement crowned our efforts with success. We are grateful to our **Chairman Shri M V Muthuramalingam** and our **Director Shri M V M Sasikumar** for facilitating us with this opportunity. Our sincere thanks to **Advisor's Shri K Razak** and **Shri M Vasu** , **Principal Dr N Balaji** ,**Vice Principal Dr S Soundarajan** for their support. Our respected **Head of the Department** of **Electronics and Communication Engineering Dr B Sridevi**, Project Evaluator **Dr M Vijay** and **Dr G Shanmugaraj** and our **Industrial mentor Durga Prasad** deserve a special note of thanks and gratitude for having extended their fullest cooperation and continuous suggestions to make this project successful. We also thank all the faculty members of Electronics and Communication Engineering department for their help during the course of this project. We also like to express our gratefulness to our beloved parents and our family members who have always provided backup with their unending moral support and of course momentary help too which we believe are the driving forces for the completion of the project.

Dinesh Shithik Varshan A
Venkatesh P S
Kishore H
Velmurugan R
Srikanth Y

ABSTRACT

Agriculture is the main aspect of country development. Many people lead their life from agriculture field, which gives fully related to agricultural products. Plant disease, especially on leaves, is one of the major factors of reductions in both quality and quantity of the food crops. In agricultural aspects, if the plant is affected by leaf disease then it reduces the growth of the agricultural level. Finding the leaf disease is an important role of agriculture preservation. After pre-processing using a median filter, segmentation is done by Guided Active Contour method and finally, the leaf disease is identified by using Support Vector Machine. The disease-based similarity measure is used for fertilizer recommendation.

Project Report Format

1. **INTRODUCTION**
   - 1.1 Project Overview
   - 1.2 Purpose
2. **LITERATURE SURVEY**
   - 2.1 Existing problem
   - 2.2 References
   - 2.3 Problem Statement Definition
3. **IDEATION & PROPOSED SOLUTION**
   - 3.1 Empathy Map Canvas
   - 3.2 Ideation & Brainstorming
   - 3.3 Proposed Solution
   - 3.4 Problem Solution fit
4. **REQUIREMENT ANALYSIS**
   - 4.1 Functional requirement
   - 4.2 Non-Functional requirements
5. **PROJECT DESIGN**
   - 5.1 Data Flow Diagrams
   - 5.2 Solution & Technical Architecture
   - 5.3 User Stories
6. **PROJECT PLANNING & SCHEDULING**
   - 6.1 Sprint Planning & Estimation
   - 6.2 Sprint Delivery Schedule
   - 6.3 Reports from JIRA
7. **CODING & SOLUTIONING (Explain the features added in the project along with code)**
   - 7.1 Feature 1
   - 7.2 Feature 2
   - 7.3 Database Schema (if Applicable)
8. **TESTING**
   - 8.1 Test Cases
   - 8.2 User Acceptance Testing
9. **RESULTS**
   - 9.1 Performance Metrics
10. **ADVANTAGES & DISADVANTAGES**
11. **CONCLUSION**
12. **FUTURE SCOPE**
13. **APPENDIX**

    Source Code

    GitHub & Project Demo Link

Fertilizers Recommendation System For Disease Prediction

1.INTRODUCTION

1.1 Project Overview

Detection and recognition of plant diseases using machine learning are very efficient in providing symptoms of identifying diseases at its earliest. Plant pathologists can analyze the digital images using digital image processing for diagnosis of plant diseases. Application of computer vision and image processing strategies simply assist farmers in all of the regions of agriculture. Generally, the plant diseases are caused by the abnormal physiological functionalities of plants. Therefore, the characteristic symptoms are generated based on the differentiation between normal physiological functionalities and abnormal physiological functionalities of the plants. Mostly, the plant leaf diseases are caused by Pathogens which are positioned on the stems of the plants. These different symptoms and diseases of leaves are predicted by different methods in image processing. These different methods include different fundamental processes like segmentation, feature extraction and classification and so on. Mostly, the prediction and diagnosis of leaf diseases are depending on the segmentation such as segmenting the healthy tissues from diseased tissues of leaves**.**

1.2 Purpose

Recommend the fertilizer for affected leaves based on severity level. Fertilizers may be organic or inorganic. Admin can store the fertilizers based on disease categorization with severity levels. The measurements of fertilizers suggested based on disease severity.

2. LITERATURE SURVEY

2.1 Existing problem

Leaves are affected by bacteria, fungi, virus, and other insects. Support Vector Machine (SVM) algorithm classifies the leaf image as normal or affected. Vectors are constructed based on leaf features such as color, shape, textures. Then hyperplane constructed with conditions to categorize the preprocessed leaves and also implement multiclass classifier, to predict diseases in leaf image with improved accuracy.

2.2 References

Like articles, websites, blogs.

https://www.researchgate.net/.

https://techvidvan.com/tutorials/android-news-app-project-source-code/

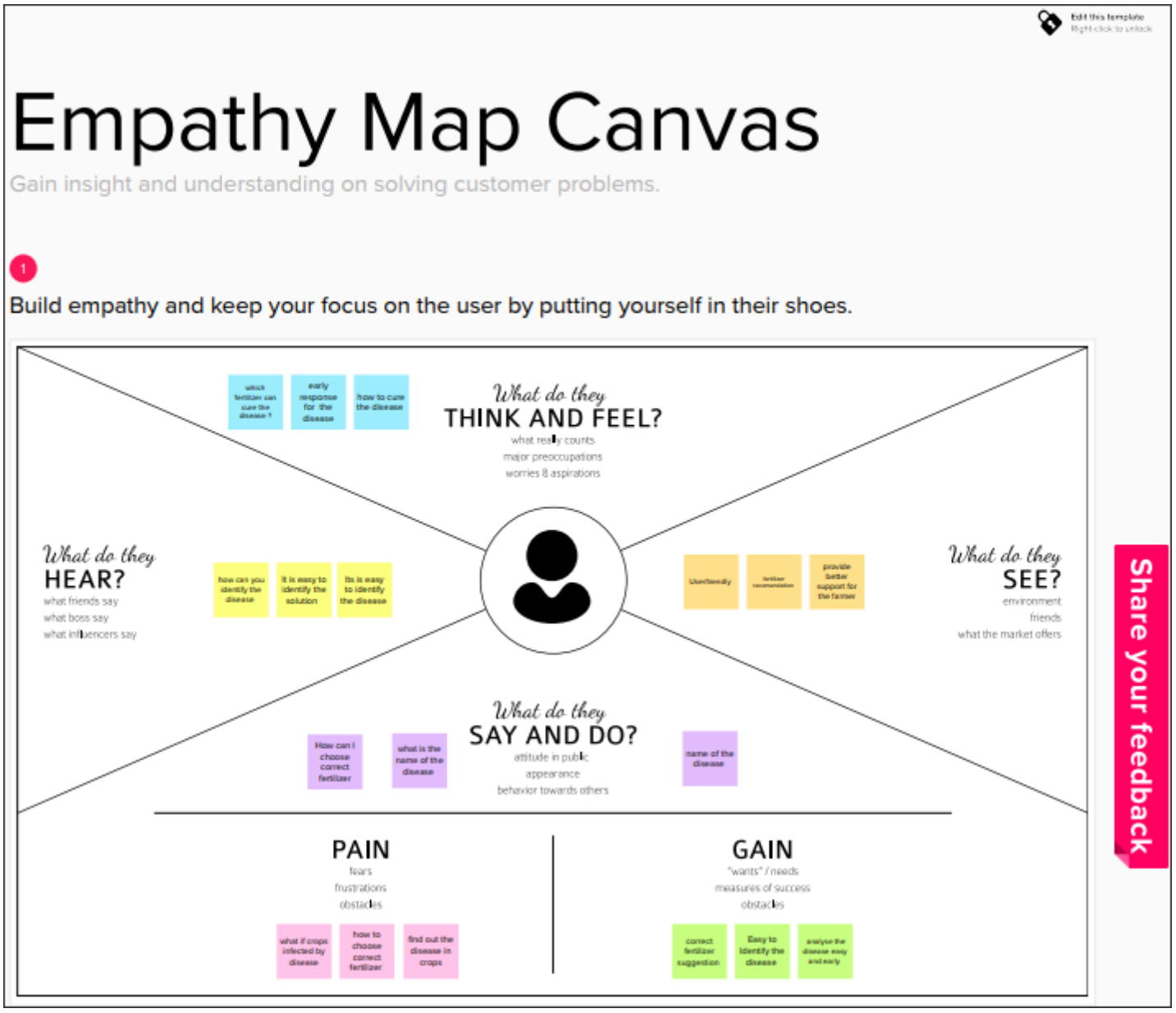
https://www.ripublication.com/

2.3 Problem Statement Definition

In today's world agriculture is very important for life and helps to save the natural resources around as. Doing agriculture is the very hard in current scenario because of many natural disaster are happening everyday. Most of the plants are affected by many disease due to pollution in water, air, soil. Identifying the disease is one of the huge hurtles in agriculture. Most of the plants are affected by leaf disease and it's hard to find to correct fertilizer to cure. Identifying the disease in early stage is very important and easy to cure that.

3. IDEATION & PROPOSED SOLUTION

3.1 Empathy Map Canvas

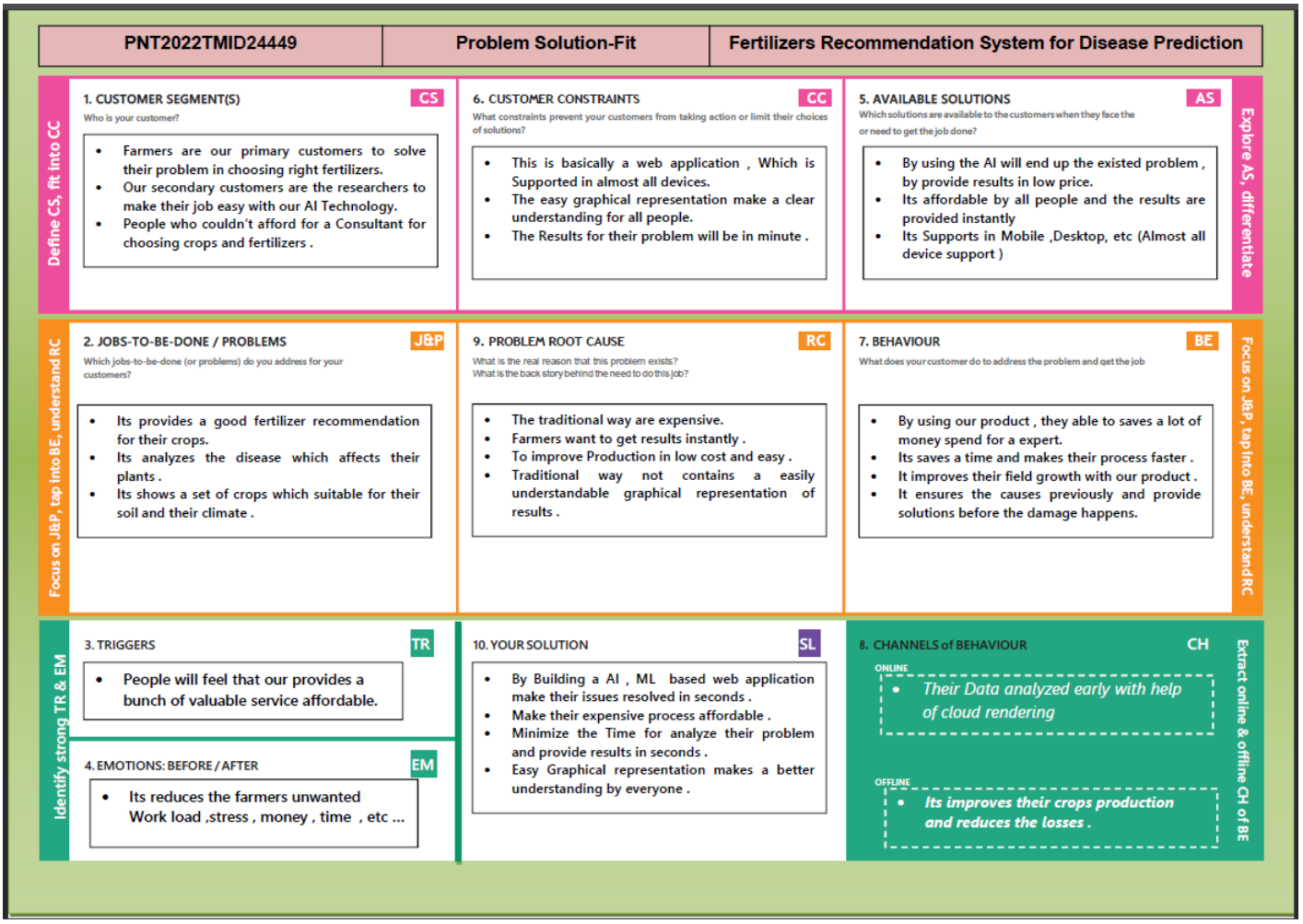


3.1.2 Ideation & Brainstorming

3.2 Proposed Solution

S.NO	PARAMETERS	DESCRIPTION
1	Problem statement (problem to be solved)	Disease in plants reduced the quantity and quality of the plants productivity. Identifying the disease in plant is hard to find.
2	Idea/solution description	One of the solution of the problem is to identifying the disease in early stage and using the correct fertilizer.
3	Novelty / uniqueness	This application can suggest good fertilizer for the disease in the plant by recognizing the images
4	Social impact/customer satisfaction	It helps the farmer by identifying the disease in the early stage and increase the quality and quantity of crops in efficient way
5	Business model(revenue model)	The application is recommends to farmer in subscription basis.
6	Scalability of the solution	This application can be improved by introducing online purchases of crops, fertilizer easily

3.3 Problem Solution fit



4. REQUIREMENT ANALYSIS

4.1 Functional requirement

Following are the functional requirements of the proposed solution.

FR No.	Functional Requirement (Epic)	Sub Requirement (Story / Sub-Task)
FR-1	User Registration	Registration through Form
FR-2	User Confirmation	Confirmation via Email Confirmation via OTP
FR-3	User Profile	Filling the profile page after logging in
FR-4	Uploading Data (Leaf)	Image of the leaves is to be uploaded
FR - 5	Requesting solution	Uploaded image is compared with the pre-defined model and solution is generated.
FR - 6	Fertilizer Recommendation	Based on the type of disease identified, suitable fertilizers are recommended.

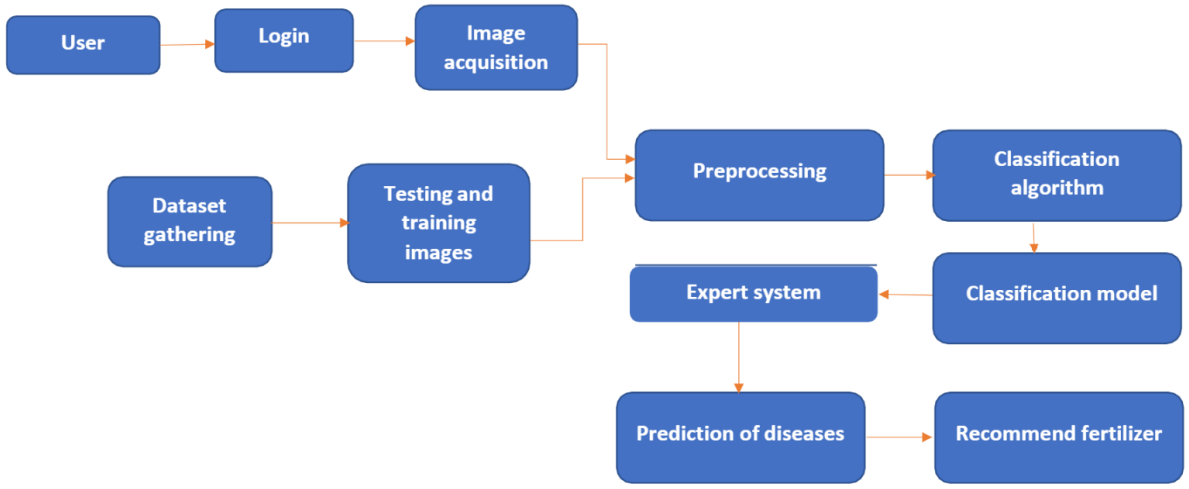
4.2 Non-functional Requirements:

Following are the non-functional requirements of the proposed solution.

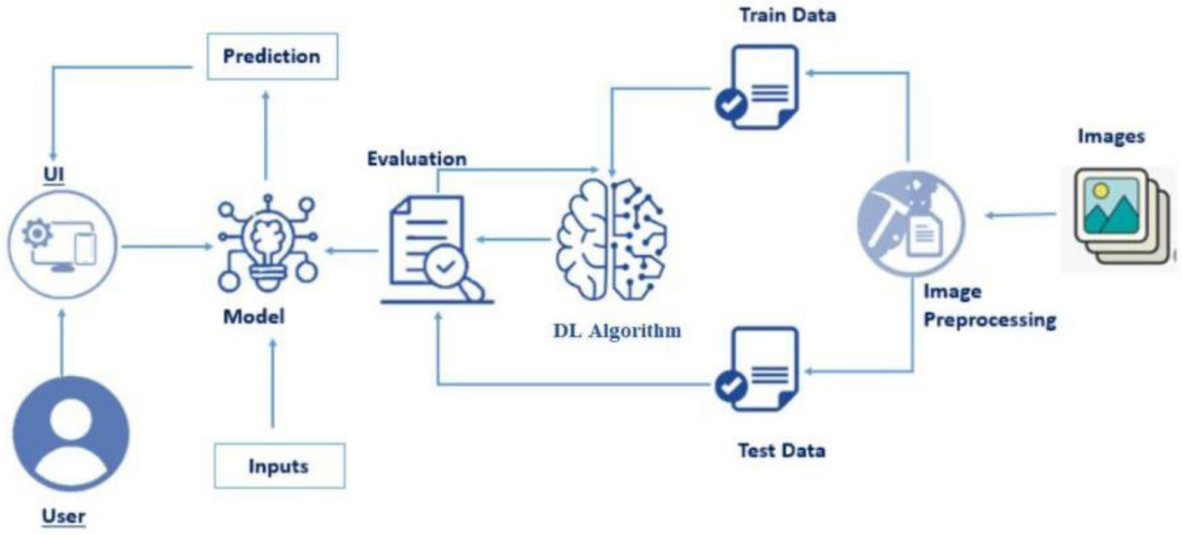
FR No.	Non-Functional Requirement	Description
NFR-1	**Usability**	The system allows the user to perform the task easily, efficiently and effectively.
NFR-2	**Security**	Information about the user and their data's are highly secured with the authorization technology
NFR-3	**Reliability**	The model deployed should be reliable and able to give accurate disease prediction and recommendation.
NFR-4	**Performance**	Response time and total processing time is fast.
NFR-5	**Availability**	The application should be available anytime and anywhere to all the registered users.
NFR-6	**Scalability**	Increase in the number of user does not affect the performance of the system.

5. PROJECT DESIGN

5.1 Data Flow Diagrams



5.2 Solution & Technical Architecture



5.3 User Stories

User Stories

Use the below template to list all the user stories for the product.

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
Customer (Mobile user)	Registration	USN-1	As a user, I can register for the application by entering my email, password, and confirming my password.	I can access my account / dashboard	High	Sprint-1
		USN-2	As a user, I will receive confirmation email once I have registered for the application	I can receive confirmation email & click confirm	High	Sprint-1
		USN-3	As a user, I can register for the application through Facebook	I can register & access the dashboard with Facebook Login	Low	Sprint-2
		USN-4	As a user, I can register for the application through Gmail		Medium	Sprint-1
	Login	USN-5	As a user, I can log into the application by entering email & password		High	Sprint-1
	Dashboard	USN-1	As a user, I can see my dashboard and go through the functions provided by the system.	I can access my dashboard	High	Sprint-1
Customer (Web user)	Registration		As a user, I can register for my account through web and login to my web page.			
Customer Care Executive	Login	USN-1	Make a call to the customer care executive and rectify the queries.	Help the user how to access the system.	High	Sprint-1
Administrator	User account control	USN-1	Responsible for carrying out the administration process.	Manage the total team	High	Sprint-1

6. PROJECT PLANNING & SCHEDULING

6.1 Sprint Planning & Estimation

TITLE	DESCRIPTION	DATE
Literature Survey & Information Gathering	Literature survey on the selected project & gathering information byreferring the technical papers, research publications etc.	6 OCTOBER 2022
Prepare Empathy Map	Prepare Empathy Map Canvas to capture the user Pains & Gains, Prepare list of problem statements	6 OCTOBER 2022
Ideation Brain Storming	List the by organizing the brainstorming session and prioritize the top 3 ideas based on the feasibility & importance.	6 OCTOBER 2022
Proposed Solution	Prepare the proposed solution document, which includes the novelty, feasibility of idea, business model, social impact, scalability of solution, etc.	6 OCTOBER 2022
Problem Solution Fit	Prepare problem solution fit document	17 OCTOBER 2022
Solution Architecture	Prepare solution architecture document.	17 OCTOBER 2022
Customer Journey	Prepare the customer journey maps to	17 OCTOBER 2022

	Understand the user interactions& experiences with the application (entry to exit).	
Data Flow Diagrams	Draw the data flow diagrams and submit for review.	17 October 2022
Technology Architecture	Prepare the technology architecture diagram.	17 October 2022
Prepare Milestone & Activity List	Prepare the milestones & activity list of the project.	3 NOVEMBER 2022
Project Development - Delivery of Sprint-1, 2, 3 & 4	Develop & submit the developed code by testing it.	19 NOVEMBER

6.2 Sprint Delivery Schedule

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-1	Image Processing.	USN-1	As a user, I can retrieve useful information aboutthe images.	1	Low	A.Dinesh Shithik Varshan, H.Kishore, Y.Srikanth, R.Velmurugan, P S.Venkatesh
Sprint-2	Model Building for Fruit DiseasePrediction.	USN-2	As a user, I can able to predict fruitdisease using this model.	1	Medium	A.Dinesh Shithik Varshan, H.Kishore, Y.Srikanth, R.Velmurugan, P S.Venkatesh
Sprint-2	Model Building for VegetableDisease Prediction.	USN-3	As a user, I can able to predict vegetabledisease using this model.	2	Medium	A.Dinesh Shithik Varshan, H.Kishore, Y.Srikanth, R.Velmurugan, P S.Venkatesh
Sprint-3	Application Building.	USN-4	As a user, I can see a web page for Fertilizers Recommendation System forDisease Prediction	2	High	A.Dinesh Shithik Varshan, H.Kishore, Y.Srikanth, R.Velmurugan, P S.Venkatesh
Sprint-4	Train The Model on IBM Cloud.	USN-5	As a user, I can save the information aboutFertilizers and crops on IBM cloud	2	High	A.Dinesh Shithik Varshan, H.Kishore, Y.Srikanth, R.Velmurugan, P S.Venkatesh

6.3 Reports from JIRA

---

7. CODING & SOLUTIONING

Forgot password-

Forpass.html

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <meta name="description" content="">
```

<meta name="author" content="Mark Otto, Jacob Thornton, and Bootstrap contributors">

<meta name="generator" content="Hugo 0.84.0">

<title>Sign In</title>

<link rel="canonical" href="https://getbootstrap.com/docs/5.0/examples/sign-in/">

<link href="https://getbootstrap.com/docs/5.0/assets/css/docs.css" rel="stylesheet">

<!-- Bootstrap core CSS -->

<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.1/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-iYQeCzEYFbKjA/T2uDLTpkwGzCiq6soy8tYaI1GyVh/UjpbCx/TYkiZhlZB6+fzT" crossorigin="anonymous">

<!-- Favicons -->

<link rel="apple-touch-icon" href="/docs/5.0/assets/img/favicons/apple-touch-icon.png" sizes="180x180">

<link rel="icon" href="/docs/5.0/assets/img/favicons/favicon-32x32.png" sizes="32x32" type="image/png">

<link rel="icon" href="/docs/5.0/assets/img/favicons/favicon-16x16.png" sizes="16x16" type="image/png">

<link rel="manifest" href="/docs/5.0/assets/img/favicons/manifest.json">

<link rel="mask-icon" href="/docs/5.0/assets/img/favicons/safari-pinned-tab.svg" color="#7952b3">

<link rel="icon" href="/docs/5.0/assets/img/favicons/favicon.ico">

<meta name="theme-color" content="#7952b3">

```
  <style>
    .bd-placeholder-img {
      font-size:  1.125rem;
      text-anchor: middle;
      -webkit-user-select: none;
      -moz-user-select: none;
      user-select: none;
    }

    @media (min-width: 768px) {
      .bd-placeholder-img-lg {
        font-size: 3.5rem;
      }
    }
  </style>


  <!-- Custom styles for this template -->
  <link href="static/sheets/signin.css" rel="stylesheet">
  <link href="static/sheets/colors.css" rel="stylesheet">
</head>
<body class="text-center">
      <nav class="navbar navbar-dark bg-dark fixed-top">
          <div class="container-fluid">
            <a class="navbar-brand" href="#">News Tracker</a>
```

```
<button class="navbar-toggler" type="button" data-bs-toggle="offcanvas" data-bs-target="#offcanvasDarkNavbar" aria-controls="offcanvasDarkNavbar">
  <span class="navbar-toggler-icon"></span>
</button>
<div class="offcanvas offcanvas-end text-bg-dark" tabindex="-1" id="offcanvasDarkNavbar" aria-labelledby="offcanvasDarkNavbarLabel">
  <div class="offcanvas-header">
    <h5 class="offcanvas-title" id="offcanvasDarkNavbarLabel">Profile</h5>
    <button type="button" class="btn-close btn-close-white" data-bs-dismiss="offcanvas" aria-label="Close"></button>
  </div>
  <div class="offcanvas-body">
    <ul class="navbar-nav justify-content-end flex-grow-1 pe-3">
      <li class="nav-item">
        <a class="nav-link active" aria-current="page" href="home.html">Home</a>
      </li>
      <li class="nav-item">
        <a class="nav-link" href="base.html">Fetch News</a>
      </li>
      <li class="nav-item">
        <a class="nav-link" href="about.html">About Us</a>
      </li>
      <li class="nav-item">
        <a class="nav-link" href="signin.html">Sign In</a>
      </li>
```

```
                <li class="nav-item">
                  <a class="nav-link" href="signup.html">Sign Up</a>
                </li>
              </div>
            </div>
          </div>
        </nav>
<main class="form-signin  ">
  <form action = "{{ url_for("getUser")}}" method="POST">
    <img class="mb-4" src="static/Images/house-user-solid.svg" alt="" width="72" height="57">
    <h1 class="h3 mb-3 fw-normal white">Change your password</h1>

    <div class="form-floating">
      <input type="email" class="form-control" id="floatingInput" name = "uname" placeholder="name@example.com">
      <label for="floatingInput">Email address</label>
    </div>
    <br>
    <button class="w-100 btn btn-lg btn-warning btn-outline-warning" type="submit"><a href="static/templates/changepass.html">Change Password</a></button><br><br>
  </form>
</main>
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.2.1/dist/js/bootstrap.bundle.min.js" integrity="sha384-u1OknCvxWvY5kfmNBILK2hRnQC3Pr17a+RTT6rIHI7NnikvbZlHgTPOOmMi466C8" crossorigin="anonymous"></script>
```

```
<script src="https://cdn.jsdelivr.net/npm/@docsearch/js@3"></script>

<script src="https://cdn.jsdelivr.net/npm/@stackblitz/sdk@1/bundles/sdk.umd.js"></script>


<script src="/docs/5.2/assets/js/docs.min.js"></script>


<script>

  document.querySelectorAll('.btn-edit').forEach(btn => {
    btn.addEventListener('click', event => {
      const htmlSnippet = event.target.closest('.bd-code-snippet').querySelector('.bd-example').innerHTML



      const classes = Array.from(event.target.closest('.bd-code-snippet').querySelector('.bd-example').classList).join(' ')


      const jsSnippet = event.target.closest('.bd-code-snippet').querySelector('.btn-edit').getAttribute('data-sb-js-snippet')
      StackBlitzSDK.openBootstrapSnippet(htmlSnippet, jsSnippet, classes)
    })
  })


  StackBlitzSDK.openBootstrapSnippet = (htmlSnippet, jsSnippet, classes) => {
    const markup = `<!doctype html>
<html lang="en">
  <head>
```

```
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link
href="https:\/\/cdn.jsdelivr.net\/npm\/bootstrap@5.2.1\/dist\/css\/bootstrap.min.css" rel="stylesheet">
    <link href="https://getbootstrap.com/docs/5.2/assets/css/docs.css" rel="stylesheet">
    <title>Bootstrap Example</title>
    <${'script'}
src="https:\/\/cdn\.jsdelivr\.net\/npm\/bootstrap@5\.2\.1\/dist\/js\/bootstrap\.bundle\.min\.js"></${'script'}>
  </head>
  <body class="p-3 m-0 border-0 ${classes}">

    <!-- Example Code -->
${htmlSnippet.replace(/^/gm, '    ')}
    <!-- End Example Code -->
  </body>
</html>`

    const jsSnippetContent = jsSnippet ? '\/\/ NOTICE!!! Initially embedded in our docs this JavaScript\n\/\/ file contains elements that can help you create reproducible\n\/\/ use cases in StackBlitz for instance\.\n\/\/ In a real project please adapt this content to your needs\.\n\/\/ \u002b\u002b\u002b\u002b\u002b\u002b\u002b\u002b\u002b\u002b\u002b\u002b\u002b\u002b\u002b\u002b\u002b\u002b\u002b\u002b\u002b\u002b\u002b\u002b\u002b\u002b\u002b\u002b\u002b\u002b\u002b\u002b\u002b\u002b\u002b\u002b\u002b\u002b\u002b\u002b\u002b\u002b\u002b\u002b\u002b\u002b\u002b\u002b\u002b\u002b\u002b\u002b\u002b\n\n\/\*!\n \* JavaScript for Bootstrap\u0027s docs \(https:\/\/getbootstrap\.com\/\)\n \* Copyright 2011\-2022 The Bootstrap Authors\n \* Copyright 2011\-2022 Twitter, Inc\.\n \* Licensed under the Creative Commons Attribution 3\.0 Unported License\.\n \*
```

For details, see https:\/\/creativecommons\.org\/licenses\/by\/3\.0\/\.\n *\/\n\n\/* global bootstrap: false *\/\n\n\(\(\) =\u003e \{\n \u0027use strict\u0027\n\n \/\/ \-\-\-\-\-\-\-\-\-\n \/\/ Tooltips\n \/\/ \-\-\-\-\-\-\-\-\-\n \/\/ Instantiate all tooltips in a docs or StackBlitz page\n document\.querySelectorAll\(\u0027\[data\-bs\-toggle=\u0022tooltip\u0022\]\u0027\)\n \.forEach\(tooltip =\u003e \{\n new bootstrap\.Tooltip\(tooltip\)\n \}\)\n\n \/\/ \-\-\-\-\-\-\-\-\-\n \/\/ Popovers\n \/\/ \-\-\-\-\-\-\-\-\-\n \/\/ Instantiate all popovers in a docs or StackBlitz page\n document\.querySelectorAll\(\u0027\[data\-bs\-toggle=\u0022popover\u0022\]\u0027\)\n \.forEach\(popover =\u003e \{\n new bootstrap\.Popover\(popover\)\n \}\)\n\n \/\/ \-\n \/\/ Toasts\n \/\/ \-\n \/\/ Used by \u0027Placement\u0027 example in docs or StackBlitz\n const toastPlacement = document\.getElementById\(\u0027toastPlacement\u0027\)\n if \(toastPlacement\) \{\n document\.getElementById\(\u0027selectToastPlacement\u0027\)\.addEventListener\(\u0027change\u0027, function \(\) \{\n if \(!toastPlacement\.dataset\.originalClass\) \{\n toastPlacement\.dataset\.originalClass = toastPlacement\.className\n \}\n\n toastPlacement\.className = `\$\{toastPlacement\.dataset\.originalClass\} \$\{this\.value\}`\n \}\)\n \}\n\n \/\/ Instantiate all toasts in a docs page only\n document\.querySelectorAll\(\u0027\.bd\-example \.toast\u0027\)\n \.forEach\(toastNode =\u003e \{\n const toast = new bootstrap\.Toast\(toastNode, \{\n autohide: false\n \}\)\n\n toast\.show\(\)\n \}\)\n\n \/\/ Instantiate all toasts in a docs page only\n const toastTrigger = document\.getElementById\(\u0027liveToastBtn\u0027\)\n const toastLiveExample = document\.getElementById\(\u0027liveToast\u0027\)\n if \(toastTrigger\) \{\n toastTrigger\.addEventListener\(\u0027click\u0027, \(\) =\u003e \{\n const toast = new bootstrap\.Toast\(toastLiveExample\)\n\n toast\.show\(\)\n \}\)\n \}\n\n \/\/ \-\n \/\/ Alerts\n \/\/ \-\n \/\/ Used in \u0027Show live toast\u0027 example in docs or StackBlitz\n const alertPlaceholder = document\.getElementById\(\u0027liveAlertPlaceholder\u0027\)\n const alertTrigger = document\.getElementById\(\u0027liveAlertBtn\u0027\)\n\n const appendAlert = \(message, type\) =\u003e \{\n const wrapper = document\.createElement\(\u0027div\u0027\)\n wrapper\.innerHTML = \[\n

`\u003cdiv class=\u0022alert alert\-\$\{type\} alert\-dismissible\u0022 role=\u0022alert\u0022\u003e`,\n ` \u003cdiv\u003e\$\{message\}\u003c\/div\u003e`,\n \u0027 \u003cbutton type=\u0022button\u0022 class=\u0022btn\-close\u0022 data\-bs\-dismiss=\u0022alert\u0022 aria\-label=\u0022Close\u0022\u003e\u003c\/button\u003e\u0027,\n \u0027\u003c\/div\u003e\u0027\n \]\.join\(\u0027\u0027\)\n\n alertPlaceholder\.append\(wrapper\)\n \}\n\n if \(alertTrigger\) \{\n alertTrigger\.addEventListener\(\u0027click\u0027, \(\) =\u003e \{\n appendAlert\(\u0027Nice, you triggered this alert message!\u0027, \u0027success\u0027\)\n \}\)\n \}\n\n \/\/ \-\n \/\/ Checks \u0026 Radios\n \/\/ \-\n \/\/ Indeterminate checkbox example in docs and StackBlitz\n document\.querySelectorAll\(\u0027\.bd\-example\-indeterminate \[type=\u0022checkbox\u0022\]\u0027\)\n \.forEach\(checkbox =\u003e \{\n if \(checkbox\.id\.includes\(\u0027Indeterminate\u0027\)\) \{\n checkbox\.indeterminate = true\n \}\n \}\)\n\n \/\/ \-\n \/\/ Links\n \/\/ \-\n \/\/ Disable empty links in docs examples only\n document\.querySelectorAll\(\u0027\.bd\-content \[href=\u0022#\u0022\]\u0027\)\n \.forEach\(link =\u003e \{\n link\.addEventListener\(\u0027click\u0027, event =\u003e \{\n event\.preventDefault\(\)\n \}\)\n \}\)\n\n \/\/ \-\n \/\/ Modal\n \/\/ \-\n \/\/ Modal \u0027Varying modal content\u0027 example in docs and StackBlitz\n const exampleModal = document\.getElementById\(\u0027exampleModal\u0027\)\n if \(exampleModal\) \{\n exampleModal\.addEventListener\(\u0027show\.bs\.modal\u0027, event =\u003e \{\n \/\/ Button that triggered the modal\n const button = event\.relatedTarget\n \/\/ Extract info from data\-bs\-* attributes\n const recipient = button\.getAttribute\(\u0027data\-bs\-whatever\u0027\)\n\n \/\/ Update the modal\u0027s content\.\n const modalTitle = exampleModal\.querySelector\(\u0027\.modal\-title\u0027\)\n const modalBodyInput = exampleModal\.querySelector\(\u0027\.modal\-body input\u0027\)\n\n modalTitle\.textContent = `New message to \$\{recipient\}`\n modalBodyInput\.value = recipient\n \}\)\n \}\n\n \/\/ \-\-

\-\n \/\/ Offcanvas\n \/\/ \-\n \/\/ \u0027Offcanvas components\u0027 example in docs only\n const myOffcanvas = document\.querySelectorAll\(\u0027\.bd\-example\-offcanvas \.offcanvas\u0027\)\n if \(myOffcanvas\) \{\n myOffcanvas\.forEach\(offcanvas =\u003e \{\n offcanvas\.addEventListener\(\u0027show\.bs\.offcanvas\u0027, event =\u003e \{\n event\.preventDefault\(\)\n \}, false\)\n \}\)\n \}\n\}\)\(\)\n' : null

```
    const project = {
      files: {
        'index.html': markup,
        'index.js': jsSnippetContent
      },
      title: 'Bootstrap Example',
      description: `Official example from ${window.location.href}`,
      template: jsSnippet ? 'javascript' : 'html',
      tags: ['bootstrap']
    }

    StackBlitzSDK.openProject(project, { openFile: 'index.html' })
  }
</script>
  </body>
</html>
```

8. TESTING

8.1 Test Cases

Test case

Test case	feature	component	Test scenario	Expected result	Actual result	status	comments	bug	Executed by
Sign in	**Functional**	**Login page**	**Verify user can see the sign in option**	**can visible**	**Yes visible**	**pass**	**successful**	**-**	**Bhuvaneshwaran**
Sign up	Functional	Login page	Verify user has the option to sign up	Can visible	Yes visible	pass	Successful	-	Ashwin
Forgot password	Functional	Login page	Verify user has the option to forgot password	Yes the option is available	Option is available	pass	Successful	-	Mohan kumar

Fetch news	Functional	Home page	Verify user can get the news	News will be eed to the app	404 error	Fail	unsuccessful	App integeration problem	Ragul rathna, Ashwin
Types of news available in the fetch news page	Functional	Fetch news	Types of news available	Weather, Sport, Economy.	Hover buttons will be Shown.	Yes	successful	-	Mohan kumar, Ashwin.

Result:

The core strategy of this project is to predict the crop based on the soil nutrient content and the location where the crop is growing. This system will help he farmers to choose the right crop for their land and to give the suitable amount of fertilizer to produce the maximum yield. The Support Vector Machine algorithm helps to predict the crop the precisely based on the pre-processed crop data. This system will also help the new comers to choose the crop which will grow in their area and produce them a good profit. A decent amount of profit will attract more people towards the agriculture.

10. ADVANTAGES & DISADVANTAGES

Advantages:-

➤ The proposed model could predict the disease just from the image of a particular plant.
➤ Easy to use UI.
➤ Model has some good accuracy in detecting the plant just by taking the input(leaf).
➤ These kind of web applications can be used in the agricultural sector as well as for small house hold plants as well.

Disadvantages:-

- ❖ Prediction is limited to few plants as we havn't trained all the plants.

11. CONCLUSION

We explored the feasibility of recognizing patterns of news reading interactions and evaluated three adaptive interface designs for different news reader types. We show that from their interaction log, a specific user can be recognized as one of three kinds. The reader types emerging from the online survey are well defined and distinct. The evaluation of the three variant interfaces suggests that different news reader types need different user interfaces. We have demonstrated a method for monitoring users' news reading behaviour and inferring news reader type from it. In the future we will further explore the design of adaptive interfaces, in order to be in a position to demonstrate a complete adaptive mobile news framework providing automatic personalization of news apps.

12. FUTURE SCOPE

 As of now we have just built the web application which apparently takes the input as an image and then predict the out in the near future we can develop an application which computer vision and AI techniques to predict the infection once you keep the camera near the plant or leaf this could make our project even more usable

 This further research is implementing the proposed algorithm with the existing public datasets. Also, various segmentation algorithms can be implemented to improve accuracy. The proposed algorithm can be modified further to identify the disease that affects the various plant organs such as vegetables and fruits.

13. APPENDIX

Requirement.txt

numpy

pandas

Flask

scikit-learn

https://download.pytorch.org/whl/cpu/torch-1.7.0%2Bcpu-cp36-cp36m-linux_x86_64.whl
https://download.pytorch.org/whl/cpu/torchvision-0.8.1%2Bcpu-cp36-cp36m-linux_x86_64.whl
requests

Pillow

gunicorn == 20.0.4

asgiref==3.5.0

bcrypt==3.2.0

cffi==1.15.0

click==8.1.2

dnspython==2.2.1

email-validator==1.1.3

Flask==2.1.1

Flask-Bcrypt==1.0.1

Flask-Login==0.6.0

Flask-SQLAlchemy==2.5.1

Flask-WTF==1.0.1

greenlet==1.1.2

idna==3.3

importlib-metadata==4.11.3

itsdangerous==2.1.2

Jinja2==3.1.1

MarkupSafe==2.1.1

pycparser==2.21

six==1.16.0

SQLAlchemy==1.4.35

sqlparse==0.4.2

Werkzeug==2.1.1

WTForms==3.0.1

zipp==3.8.0

flask_sqlalchemy

flask_login

flask_wtf

wtforms

wtforms.validators

flask_bcrypt

App.py

```
# Importing essential libraries and modules
from flask import Flask, render_template, request, Markup,url_for, redirect
import numpy as np
import pandas as pd
from utils.disease import disease_dic
from utils.fertilizer import fertilizer_dic
import requests
import config
import pickle
import io
```

```
import torch

from torchvision import transforms

from PIL import Image

from utils.model import ResNet9

from flask_sqlalchemy import SQLAlchemy

from flask_login import UserMixin, login_user, LoginManager, login_required, logout_user,
current_user

from flask_wtf import FlaskForm

from wtforms import StringField, PasswordField, SubmitField

from wtforms.validators import InputRequired, Length, ValidationError

from flask_bcrypt import Bcrypt

#
==========================================================================
================ ======

# ------------------------LOADING THE TRAINED MODELS ----------------------------------------
-

# Loading plant disease classification model

disease_classes = ['Apple___Apple_scab',

'Apple___Black_rot',

 'Apple___Cedar_apple_rust',

 'Apple___healthy',

 'Blueberry___healthy',

 'Cherry_(including_sour)___Powdery_mildew',

 'Cherry_(including_sour)___healthy',

 'Corn_(maize)___Cercospora_leaf_spot Gray_leaf_spot',  'Corn_(maize)___Common_rust_',

 'Corn_(maize)___Northern_Leaf_Blight',

 'Corn_(maize)___healthy',

 'Grape___Black_rot',
```

'Grape___Esca_(Black_Measles)',

'Grape___Leaf_blight_(Isariopsis_Leaf_Spot)',

'Grape___healthy',

'Orange___Haunglongbing_(Citrus_greening)',

'Peach___Bacterial_spot',

'Peach___healthy',

'Pepper,_bell___Bacterial_spot',

'Pepper,_bell___healthy',

'Potato___Early_blight',

'Potato___Late_blight',

'Potato___healthy',

'Raspberry___healthy',

'Soybean___healthy',

'Squash___Powdery_mildew',

'Strawberry___Leaf_scorch',

'Strawberry___healthy',

'Tomato___Bacterial_spot',

'Tomato___Early_blight',

'Tomato___Late_blight',

'Tomato___Leaf_Mold',

'Tomato___Septoria_leaf_spot',

'Tomato___Spider_mites Two-spotted_spider_mite', 'Tomato___Target_Spot',

'Tomato___Tomato_Yellow_Leaf_Curl_Virus',

'Tomato___Tomato_mosaic_virus',

'Tomato___healthy']

disease_model_path = 'models/plant_disease_model.pth' disease_model = ResNet9(3, len(disease_classes))

```
disease_model.load_state_dict(torch.load(
    disease_model_path, map_location=torch.device('cpu'))) disease_model.eval()
# Loading crop recommendation model
crop_recommendation_model_path = 'models/RandomForest.pkl' crop_recommendation_model =
pickle.load(
open(crop_recommendation_model_path, 'rb'))
#
================================================================================
================ =
# Custom functions for calculations
def weather_fetch(city_name):
"""
 Fetch and returns the temperature and humidity of a city
 :params: city_name
 :return: temperature, humidity
 """
 api_key = config.weather_api_key
 base_url = "http://api.openweathermap.org/data/2.5/weather?"
 complete_url = base_url + "appid=" + api_key + "&q=" + city_name  response =
requests.get(complete_url)
 x = response.json()
 if x["cod"] != "404":
y = x["main"]
 temperature = round((y["temp"] - 273.15), 2)  humidity = y["humidity"]
  return temperature, humidity
else:
        return None
def predict_image(img, model=disease_model):
```

```
    """
    Transforms image to tensor and predicts disease label
    :params: image
    :return: prediction (string)
    """
    transform = transforms.Compose([
        transforms.Resize(256),
        transforms.ToTensor(),
    ])
    image = Image.open(io.BytesIO(img))
    img_t = transform(image)
    img_u = torch.unsqueeze(img_t, 0)

    # Get predictions from model
    yb = model(img_u)
    # Pick index with highest probability
    _, preds = torch.max(yb, dim=1)
    prediction = disease_classes[preds[0].item()]
    # Retrieve the class label
    return prediction

#
================================================================================================
=======
#-----------------------------------FLASK APP ----------------------------------

app = Flask(__name__)

db = SQLAlchemy(app)
bcrypt = Bcrypt(app)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///database.db'
app.config['SECRET_KEY'] = 'thisisasecretkey'

login_manager = LoginManager()
login_manager.init_app(app)
login_manager.login_view = 'login'

@login_manager.user_loader
def load_user(user_id):
    return User.query.get(int(user_id))

class User(db.Model, UserMixin):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(20), nullable=False, unique=True)
    password = db.Column(db.String(80), nullable=False)

class RegisterForm(FlaskForm):
    username = StringField(validators=[
                        InputRequired(), Length(min=4, max=20)], render_kw={"placeholder": "Username"})
```

```
    password = PasswordField(validators=[
                        InputRequired(), Length(min=8, max=20)], render_kw={"placeholder": "Password"})

    submit = SubmitField('Register')

    def validate_username(self, username):
        existing_user_username = User.query.filter_by(
            username=username.data).first()
        if existing_user_username:
            raise ValidationError(
                'That username already exists. Please choose a different one.')

class LoginForm(FlaskForm):
    username = StringField(validators=[
                        InputRequired(), Length(min=4, max=20)], render_kw={"placeholder": "Username"})

    password = PasswordField(validators=[
                        InputRequired(), Length(min=8, max=20)], render_kw={"placeholder": "Password"})

    submit = SubmitField('Login')

@app.route('/')
def home():
    return render_template('home.html')

@app.route('/login', methods=['GET', 'POST'])
def login():
    form = LoginForm()
    if form.validate_on_submit():
        user = User.query.filter_by(username=form.username.data).first()
        if user:
            if bcrypt.check_password_hash(user.password, form.password.data):
                login_user(user)
                return redirect (url_for('dashboard'))
    return render_template('login.html', form=form)

@app.route('/dashboard', methods=['GET', 'POST'])
@login_required
def dashboard():
    return render_template('dashboard.html')

@app.route('/logout', methods=['GET', 'POST'])
@login_required
def logout():
    logout_user()
    return redirect(url_for('login'))

@ app.route('/register', methods=['GET', 'POST'])
def register():
    form = RegisterForm()

    if form.validate_on_submit():
        hashed_password = bcrypt.generate_password_hash(form.password.data)
```

```
        new_user = User(username=form.username.data, password=hashed_password)
        db.session.add(new_user)
        db.session.commit()
        return redirect(url_for('login'))

    return render_template('register.html', form=form)

# render home page

# render crop recommendation form page

@ app.route('/crop-recommend')
def crop_recommend():
    title = 'Crop Recommendation'
    return render_template('crop.html', title=title)

# render fertilizer recommendation form page

@ app.route('/fertilizer')
def fertilizer_recommendation():
    title = 'Fertilizer Suggestion'

    return render_template('fertilizer.html', title=title)

# render disease prediction input page



#
================================================================================
=======

# RENDER PREDICTION PAGES

# render crop recommendation result page

@ app.route('/crop-predict', methods=['POST'])
def crop_prediction():
    title = 'Crop Recommendation'

    if request.method == 'POST':
        N = int(request.form['nitrogen'])
        P = int(request.form['phosphorous'])
        K = int(request.form['pottasium'])
        ph = float(request.form['ph'])
        rainfall = float(request.form['rainfall'])

        # state = request.form.get("stt")
        city = request.form.get("city")

        if weather_fetch(city) != None:
            temperature, humidity = weather_fetch(city)
            data = np.array([[N, P, K, temperature, humidity, ph, rainfall]])
            my_prediction = crop_recommendation_model.predict(data)
```

```
            final_prediction = my_prediction[0]

            return render_template('crop-result.html', prediction=final_prediction, title=title)

        else:

            return render_template('try_again.html', title=title)

# render fertilizer recommendation result page

@ app.route('/fertilizer-predict', methods=['POST'])
def fert_recommend():
    title = 'Fertilizer Suggestion'

    crop_name = str(request.form['cropname'])
    N = int(request.form['nitrogen'])
    P = int(request.form['phosphorous'])
    K = int(request.form['pottasium'])
    # ph = float(request.form['ph'])

    df = pd.read_csv('Data/fertilizer.csv')

    nr = df[df['Crop'] == crop_name]['N'].iloc[0]
    pr = df[df['Crop'] == crop_name]['P'].iloc[0]
    kr = df[df['Crop'] == crop_name]['K'].iloc[0]

    n = nr - N
    p = pr - P
    k = kr - K
    temp = {abs(n): "N", abs(p): "P", abs(k): "K"}
    max_value = temp[max(temp.keys())]
    if max_value == "N":
        if n < 0:
            key = 'NHigh'
        else:
            key = "Nlow"
    elif max_value == "P":
        if p < 0:
            key = 'PHigh'
        else:
            key = "Plow"
    else:
        if k < 0:
            key = 'KHigh'
        else:
            key = "Klow"

    response = Markup(str(fertilizer_dic[key]))

    return render_template('fertilizer-result.html', recommendation=response, title=title)

# render disease prediction result page

@app.route('/disease-predict', methods=['GET', 'POST'])
```

```
def disease_prediction():
    title = 'Disease Detection'

    if request.method == 'POST':
        if 'file' not in request.files:
            return redirect(request.url)
        file = request.files.get('file')
        if not file:
            return render_template('disease.html', title=title)
        try:
            img = file.read()

            prediction = predict_image(img)

            prediction = Markup(str(disease_dic[prediction]))
            return render_template('disease-result.html', prediction=prediction, title=title)
        except:
            pass
    return render_template('disease.html', title=title)

#
================================================================================================
=======
if __name__== '_main_':
    app.run(debug=True)
```

Notebook Code(ipynb File) :

```
# Creating final data for crop and fertilizer recommendation system
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
fertilizer_data_path = '../Data-raw/FertilizerData.csv'
merge_fert = pd.read_csv(fertilizer_data_path)
merge_fert.head()
del merge_fert['Unnamed: 0']
merge_fert.describe()
merge_fert['Crop'].unique()
plt.plot(merge_fert["N"])
plt.plot(merge_fert["P"])
plt.plot(merge_fert["K"])
sns.heatmap(merge_fert.corr(),annot=True)
merge_crop = pd.read_csv('../Data-raw/MergeFileCrop.csv')
reco_fert = merge_fert
#Add +/-3 for every NPK value
import random
temp = pd.DataFrame(columns = ['N','P','K'])
for i in range(0,merge_crop.shape[0]):
    crop = merge_crop.label.iloc[i]
    #print(crop)
    N = reco_fert[reco_fert['Crop'] == crop]["N"].iloc[0] + random.randint(-20,20)
    P = reco_fert[reco_fert['Crop'] == crop]["P"].iloc[0] + random.randint(-5,20)
```

```
    K = reco_fert[reco_fert['Crop'] == crop]["K"].iloc[0] + random.randint(-5,5)
    d = {"N":N,"P":P,"K":K}
    #print(d)
    temp = temp.append(d,ignore_index = True)
temp
merge_crop['N'] = temp['N']
merge_crop['P'] = temp['P']
merge_crop['K'] = temp['K']
merge_crop
del merge_crop['Unnamed: 0']
merge_crop
merge_crop = merge_crop[[ 'N', 'P', 'K','temperature', 'humidity', 'ph', 'rainfall', 'label']]
merge_crop.to_csv("../Data-processed/crop_recommendation.csv",index=False)
# Checking if everything went fine
df = pd.read_csv('../Data-processed/crop_recommendation.csv')
df.head()
df.shape
```

☘☘ PLANT DISEASE CLASSIFICATION USING RESNET-9 ☘☘
Corresponding Kaggle notebook can be accessed [here](https://www.kaggle.com/atharvaingle/plant-disease-classification-resnet-99-2)
☘☘☘☘☘☘DISCLAIMER: This notebook is beginner friendly, so don't worry if you don't know much about CNNs and Pytorch. Even if you have used TensorFlow in the past and are new to PyTorch, hang in there, everything is explained clearly and concisely. You will get a good overview of how to use PyTorch for imageclassification problems.
Description of the dataset
This dataset is created using offline augmentation from the original dataset. The original PlantVillage Dataset can be found [here](https://github.com/spMohanty/PlantVillage-Dataset).This dataset consists of about87K rgb images of healthy and diseased crop leaves which is categorized into 38 different classes. The total dataset is divided into 80/20 ratio of training and validation set preserving the directory structure. A new directory containing 33 test images is created later for prediction purpose.

Note: This description is given in the dataset itself# Our goal
Goal is clear and simple. We need to build a model, which can classify between healthy and diseased cropleaves and also if the crop have any disease, predict which disease is it.
Let's get started....
Importing necessary librariesLet's import required modules
!pip install torchsummary
We would require torchsummary library to print the model's summary in keras style (nicely formatted and prettyto look) as Pytorch natively doesn't support that

```
import os                       # for working with files
import numpy as np              # for numerical computationss
import pandas as pd             # for working with dataframes
import torch                    # Pytorch module
```

```
import matplotlib.pyplot as plt # for plotting informations on graph and images
using tensorsimport torch.nn as nn         # for creating neural networks
from torch.utils.data import DataLoader # for
dataloadersfrom PIL import Image         # for
checking images
import torch.nn.functional as F # for functions for calculating loss
import torchvision.transforms as transforms     # for transforming images
into tensorsfrom torchvision.utils import make_grid  # for data checking
from torchvision.datasets import ImageFolder # for working with classes
and images from torchsummary import summary       # for getting the
summary of our model

%matplotlib inline
#  Exploring the data
Loading the data
data_dir = "../input/new-plant-diseases-dataset/New Plant Diseases
Dataset(Augmented)/New Plant DiseasesDataset(Augmented)"
train_dir = data_dir +
"/train"valid_dir = data_dir
+ "/valid"
```

```
diseases =
os.listdir(train_dir)#
printing the disease names
print(diseases)
print("Total disease classes are:
{}".format(len(diseases)))plants = []
NumberOfDisease
s = 0 for plant in
diseases:
    if plant.split('_')[0] not in plants:
        plants.append(plant.split('_')[0])
    if plant.split('_')[1] != 'healthy':
        NumberOfDiseases += 1
```

The above cell extract the number of unique plants and number of unique diseases# unique plants in the dataset

```
print(f"Unique Plants are:
\n{plants}")# number of unique
plants
print("Number of plants:
{}".format(len(plants)))# number of
unique diseases
print("Number of diseases: {}".format(NumberOfDiseases))
```

So we have images of leaves of 14 plants and while excluding healthy leaves, we have 26 types of images thatshow a particular disease in a particular plant.

```
# Number of images for each
diseasenums = {}
for disease in diseases:
    nums[disease] = len(os.listdir(train_dir + '/' + disease))

# converting the nums dictionary to pandas dataframe passing index as plant name and
number of images ascolumn

img_per_class = pd.DataFrame(nums.values(), index=nums.keys(),
columns=["no. of images"])img_per_class
#### Visualizing the above information on a graph
# plotting number of images available for each
diseaseindex = [n for n in range(38)]
plt.figure(figsize=(20, 5))
plt.bar(index, [n for n in nums.values()],
width=0.3)plt.xlabel('Plants/Diseases',
fontsize=10) plt.ylabel('No of images
available', fontsize=10) plt.xticks(index,
diseases, fontsize=5, rotation=90)
plt.title('Images per each class of plant
```

disease')
We can see that the dataset is almost balanced for all classes, so we are good to go forward#### Images available for training
n_train = 0
for value in
    nums.values():
    n_train += value
print(f"There are {n_train} images for training")# Data Preparation for training
# datasets for validation and training
train = ImageFolder(train_dir, transform=transforms.ToTensor())valid = ImageFolder(valid_dir, transform=transforms.ToTensor())
`torchvision.datasets` is a class which helps in loading all common and famous datasets. It also helps in loading custom datasets. I have used subclass `torchvision.datasets.ImageFolder`
---
which helps in loading theimage data when the data is arranged in this way:
root/dog/xxx.
png
root/dog/xxy.
png
root/dog/xxz.
png

root/cat/123.png
---
root/cat/nsdf3.pn
g
root/cat/asd932_.
png
Next, after loading the data, we need to transform the pixel values of each image (0-255) to 0-1 as neural networks works quite good with normalized data. The entire array of pixel values is converted to torch [tensor](https://pytorch.org/tutorials/beginner/examples_tensor/two_layer_net_tensor.html#:~:text=A%20PyTorch%20Tensor%20is%20basically,used%20for%20arbitrary%20numeric%20computation.) and then divided by 255.If you are

not familiar why normalizing inputs help neural network, read
[this](https://towardsdatascience.com/why-data-should-be-normalized-before-training-a-neural-network-c626b7f66c7d) post.
Image shape
img, label = train[0]
print(img.shape,
label)
We can see the shape (3, 256 256) of the image. 3 is the number of channels (RGB) and 256 x 256 is the widthand height of the image
# total number of classes in train
setlen(train.classes)
# for checking some images from training
datasetdef show_image(image, label):
    print("Label :" + train.classes[label] + "(" + str(label) +
    ")")plt.imshow(image.permute(1, 2, 0))
☘ Some Images from training
dataset ☘ show_image(*train[0])
show_image(*train[70000])
show_image(*train[30000])
# Setting the seed value
random_seed = 7
torch.manual_seed(rando
m_seed)# setting the
batch size batch_size = 32
`batch_size` is the total number of images given as input at once in forward propagation of the CNN.Basically, batch size defines the number of samples that will be propagated through the network.

For instance, let's say you have 1050 training samples and you want to set up a batch_size equal to 100. Thealgorithm takes the first 100 samples (from 1st to 100th) from the training dataset and trains the network. Next, it takes the second 100 samples (from 101st to 200th) and trains the network again. We can keep doing this procedure until we have propagated all samples through of the network.
# DataLoaders for training and validation
train_dl = DataLoader(train, batch_size, shuffle=True, num_workers=2,
pin_memory=True)valid_dl = DataLoader(valid, batch_size,
num_workers=2, pin_memory=True)
- `DataLoader` is a subclass which comes from `torch.utils.data`. It helps in loading large and memory consuming datasets. It takes in `batch_size` which denotes the number of samples contained in each generated batch.

- Setting `shuffle=True` shuffles the dataset. It is heplful so that batches between epochs do not look alike. Doing so will eventually make our model more robust.

- `num_workers`, denotes the number of processes that generate batches in parallel. If you have more cores in your CPU, you can set it to number of cores in your CPU. Since, Kaggle provides a 2 core CPU, I have set it to 2

```
# helper function to show a batch of training
instancesdef show_batch(data):
    for images, labels in data:
        fig, ax = plt.subplots(figsize=(30, 30))
        ax.set_xticks([]); ax.set_yticks([])
        ax.imshow(make_grid(images,
        nrow=8).permute(1, 2, 0))break
# Images for first batch of
trainingshow_batch(train_dl)
```

☘ Modelling ☘

It is advisable to use GPU instead of CPU when dealing with images dataset because CPUs are generalized for general purpose and GPUs are optimized for training deep learning models as they can process multiple computations simultaneously. They have a large number of cores, which allows for better computation of multiple parallel processes. Additionally, computations in deep learning need to handle huge amounts of data —this makes a GPU's memory bandwidth most suitable.

To seamlessly use a GPU, if one is available, we define a couple of helper functions (`get_default_device` & `to_device`) and a helper class `DeviceDataLoader` to move our model & data to the GPU as required#### Some helper functions

```
# for moving data into GPU (if
available)def get_default_device():
    """Pick GPU if available, else
    CPU"""if
    torch.cuda.is_available:
```

```
        return
    torch.device("cuda")else:
        return torch.device("cpu")

# for moving data to device (CPU
or GPU)def to_device(data,
device):
    """Move tensor(s) to chosen
    device"""if isinstance(data,
    (list,tuple)):
        return [to_device(x, device) for x in
    data]return data.to(device,
    non_blocking=True)

# for loading in the device (GPU if available
else CPU)class DeviceDataLoader():
    """Wrap a dataloader to move data to a
    device"""def__init__(self, dl, device):
        self.dl = dl
        self.device =
        device

    def__iter__(self):
        """Yield a batch of data after moving it to
        device"""for b in self.dl:
            yield to_device(b, self.device)

    def __len__(self):
        """Number of
        batches"""return
        len(self.dl)
Checking the device we are
working withdevice =
get_default_device()
device
```

Wrap up our training and validation data loaders using `DeviceDataLoader` for automatically transferringbatches of data to the GPU (if available)

```
# Moving data into GPU
train_dl = DeviceDataLoader(train_dl,
device)          valid_dl          =
DeviceDataLoader(valid_dl, device)##
```

Building the model architecture

*We are going to use **ResNet**, which have been one of the major breakthrough in computer vision since theywere introduced in 2015.*

If you want to learn more about ResNets read the following articles:
- [Understanding and Visualizing ResNets](https://towardsdatascience.com/understanding-and-visualizing-resnets-442284831be8#:~:text=ResNet%20Layers,layers%20remains%20the%20same%20%E2%80%94%204.)
- [Overview of ResNet and its variants](https://towardsdatascience.com/an-overview-of-resnet-and-its-variants-5281e2f56035)
- [Paper with code implementation](https://paperswithcode.com/method/resnet)

In ResNets, unlike in traditional neural networks, each layer feeds into the next layer, we use a network withresidual blocks, each layer feeds into the next layer and directly into the layers about 2–3 hops away, to avoid over-fitting (a situation when validation loss stop decreasing at a point and then keeps increasing while training loss still decreases). This also helps in preventing [vanishing gradient problem](https://towardsdatascience.com/the-vanishing-gradient-problem-69bf08b15484) and allow us to train deep neural networks. Here is a simple residual block:
![image](https://www.mdpi.com/remotesensing/remotesensing-11-01896/article_deploy/html/images/remotesensing- 11-01896-g001.png)
Residual Block code implementationclass SimpleResidualBlock(nn.Module):

```
    def init (self):
        super()._init_()
        self.conv1 = nn.Conv2d(in_channels=3, out_channels=3, kernel_size=3,
        stride=1, padding=1)self.relu1 = nn.ReLU()
        self.conv2 = nn.Conv2d(in_channels=3, out_channels=3, kernel_size=3,
        stride=1, padding=1)self.relu2 = nn.ReLU()

    def  forward(self,  x):
        out =
        self.conv1(x) out
        = self.relu1(out)
        out =
        self.conv2(out)
        return self.relu2(out) + x # ReLU can be applied before or after adding the input
```

Then we define our `ImageClassificationBase` class whose functions are:

- `training_step` - To figure out how “wrong” the model is going after training or validation step.We are using this function other than just an accuracy metric that is likely not going to be differentiable (this would mean that the gradient can’t be determined, which is necessary for the model to improve during training)

A quick look at the PyTorch docs that yields the cost function: [cross_entropy](https://pytorch.org/docs/stable/nn.functional.html#cross-entropy).

- `validation_step` - Because an accuracy metric can’t be used while training the model, doesn’t mean it shouldn’t be implemented! Accuracy in this case would be measured by a threshold, and counted if the difference between the model’s prediction and the actual label is lower than that threshold.
- `validation_epoch_end` - We want to track the validation losses/accuracies and train losses after each epoch, and every time we do so we have to make sure the gradient is not being tracked.
- `epoch_end` - We also want to print validation losses/accuracies, train losses and learning rate too because we are using learning rate scheduler (which will change the learning rate after every batch of training) after each epoch.

We also define an `accuracy` function which calculates the overall accuracy of the model on an entire batch ofoutputs, so that we can use it as a metric in `fit_one_cycle`

```
# for calculating the
accuracydef
accuracy(outputs, labels):
    _, preds = torch.max(outputs, dim=1)
    return torch.tensor(torch.sum(preds == labels).item() / len(preds))


# base class for the model
class ImageClassificationBase(nn.Module):

    def training_step(self,
        batch):images, labels =
        batch
        out = self(images)                    # Generate
        predictionsloss = F.cross_entropy(out, labels) #
        Calculate loss return loss

    def validation_step(self,
        batch):images, labels =
        batch
        out = self(images)                    # Generate
        predictionloss = F.cross_entropy(out, labels)  #
        Calculate loss
        acc = accuracy(out, labels)           # Calculate
        accuracyreturn {"val_loss": loss.detach(),
        "val_accuracy": acc}

    def validation_epoch_end(self, outputs):
```

```
        batch_losses = [x["val_loss"] for x in
        outputs]
        batch_accuracy = [x["val_accuracy"] for x in outputs]
        epoch_loss = torch.stack(batch_losses).mean()        #
        Combine lossepoch_accuracy =
        torch.stack(batch_accuracy).mean()
        return {"val_loss": epoch_loss, "val_accuracy": epoch_accuracy} # Combine accuracies

    def epoch_end(self, epoch, result):
        print("Epoch [{}], last_lr: {:.5f}, train_loss: {:.4f}, val_loss: {:.4f}, val_acc:
            {:.4f}".format(epoch, result['lrs'][-1], result['train_loss'], result['val_loss'],
            result['val_accuracy']))

## Defining the final architecture of our
model # Architecture for training

# convolution block with BatchNormalization
def ConvBlock(in_channels, out_channels, pool=False):
    layers = [nn.Conv2d(in_channels, out_channels, kernel_size=3,
              padding=1),nn.BatchNorm2d(out_channels),
              nn.ReLU(inplace=True)]
    if pool:
        layers.append(nn.MaxPool
    2d(4))return
    nn.Sequential(*layers)
```

```
# resnet architecture
class ResNet9(ImageClassificationBase):
    def __init__(self, in_channels, num_diseases):
        super().__init__()

        self.conv1 = ConvBlock(in_channels, 64)
        self.conv2 = ConvBlock(64, 128, pool=True) # out_dim : 128 x 64 x 64
        self.res1 = nn.Sequential(ConvBlock(128, 128), ConvBlock(128, 128))

        self.conv3 = ConvBlock(128, 256, pool=True) # out_dim : 256 x 16 x 16
        self.conv4 = ConvBlock(256, 512, pool=True) # out_dim : 512 x 4 x 44
        self.res2 = nn.Sequential(ConvBlock(512, 512), ConvBlock(512, 512))

        self.classifier = nn.Sequential(nn.MaxPool2d(4),
                                        nn.Flatten(),
                                        nn.Linear(512, num_diseases))

    def forward(self, xb): # xb is the loaded
        batchout = self.conv1(xb)
        out = self.conv2(out)
        out  =  self.res1(out)  +
        out          out          =
        self.conv3(out)
        out = self.conv4(out)
        out  =  self.res2(out)  +
        out          out          =
        self.classifier(out)
        return out
```

Now, we define a model object and transfer it into the device with which we are working...# defining the model and moving it to the GPU

```
model = to_device(ResNet9(3, len(train.classes)),
device)model
```

Getting a nicely formatted summary of our model (like in Keras). Pytorch doesn't support it natively. So, weneed to install the `torchsummary` library (discussed earlier)

```
# getting summary of the
modelINPUT_SHAPE =
(3, 256, 256)
print(summary(model.cuda(),
(INPUT_SHAPE)))# ☘  Training
```

the model ☘

Before we train the model, Let’s define a utility functionan `evaluate` function, which will perform thevalidation phase, and a `fit_one_cycle` function which will perform the entire training process. In `fit_one_cycle`, we have use some techniques:

- **Learning Rate Scheduling**: Instead of using a fixed learning rate, we will use a learning rate scheduler, which will change the learning rate after every batch of training. There are many strategies for varying the learning rate during training, and the one we'll use is called the *"One Cycle Learning Rate Policy"*, which involves starting with a low learning rate, gradually increasing it batch-by-batch to a high learning rate for about 30% of epochs, then gradually decreasing it to a very low value for the remaining epochs.

- **Weight Decay**: We also use weight decay, which is a regularization technique which prevents the weights from becoming too large by adding an additional term to the loss function.

- **Gradient Clipping**: Apart from the layer weights and outputs, it also helpful to limit the values of gradients to a small range to prevent undesirable changes in parameters due to large gradient values. This simple yet effective technique is called gradient clipping.

We'll also record the learning rate used for each batch.# for training

```
@torch.no_grad()
def evaluate(model,
    val_loader):model.eval()
    outputs = [model.validation_step(batch) for batch in
    val_loader]return model.validation_epoch_end(outputs)


def get_lr(optimizer):
    for param_group in
        optimizer.param_groups:return
        param_group['lr']
```

```
def fit_OneCycle(epochs, max_lr, model, train_loader, val_loader,
                 weight_decay=0,grad_clip=None,
                 opt_func=torch.optim.SGD):
    torch.cuda.empty_cache
    ()history = []

    optimizer = opt_func(model.parameters(), max_lr,
    weight_decay=weight_decay)# scheduler for one cycle learniing
    rate
    sched = torch.optim.lr_scheduler.OneCycleLR(optimizer, max_lr,
epochs=epochs,steps_per_epoch=len(train_loader))


    for epoch in
        range(epochs):#
        Training
        model.train()
        train_losses = []
        lrs = []
        for batch in train_loader:
            loss =
            model.training_step(batch)
            train_losses.append(loss)
            loss.backward()

            # gradient
            clippingif
            grad_clip:
                nn.utils.clip_grad_value_(model.parameters(), grad_clip)

            optimizer.step()
            optimizer.zero_grad(
            )

            # recording and updating learning
            rateslrs.append(get_lr(optimizer))
            sched.step()


        # validation
        result = evaluate(model, val_loader)
        result['train_loss'] =
        torch.stack(train_losses).mean().item()result['lrs'] = lrs
```

```
        model.epoch_end(epoch,
        result)
        history.append(result)

    return history
```

Let's check our validation loss and accuracy

```
%%time
history = [evaluate(model,
valid_dl)]history
```

Since there are randomly initialized weights, that is why accuracy come to near 0.019 (that is 1.9% chance ofgetting the right answer or you can say model randomly chooses a class).

Now, declare some hyper parameters for the training of the model. We can change it if result is notsatisfactory.

```
epochs = 2
max_lr = 0.01
grad_clip = 0.1
weight_decay = 1e-4
opt_func =
torch.optim.Adam
```

Let's start training our model ....

Note: The following cell may take 15 mins to 45 mins to run depending on your GPU. In kaggle (P100 GPU) ittook around 20 mins of Wall Time.

```
%%time
history += fit_OneCycle(epochs, max_lr, model, train_dl, valid_dl,
                        grad_clip=grad_c
                        lip,
                        weight_decay=1e
                        -4,
```

```
                                    opt_func=opt_f
unc)### We got an accuracy of 99.2 %
#  Plotting
#### Helper functions for
plottingdef
plot_accuracies(history):
    accuracies = [x['val_accuracy'] for x in
    history]plt.plot(accuracies, '-x')
    plt.xlabel('epoch')
    plt.ylabel('accuracy')
    plt.title('Accuracy vs. No. of
    epochs');

def plot_losses(history):
    train_losses = [x.get('train_loss') for x in
    history]val_losses = [x['val_loss'] for x in
    history] plt.plot(train_losses, '-bx')
    plt.plot(val_losses, '-
    rx')plt.xlabel('epoch')
    plt.ylabel('loss')
    plt.legend(['Training',
    'Validation'])plt.title('Loss vs.
    No. of epochs');

def plot_lrs(history):
    lrs = np.concatenate([x.get('lrs', []) for x in
    history])plt.plot(lrs)
    plt.xlabel('Batch no.')
    plt.ylabel('Learning rate')
    plt.title('Learning Rate vs. Batch
    no.');
## Validation
Accuracy
plot_accuracies(histor
y)## Validation loss
plot_losses(history)
## Learning Rate
overtime
plot_lrs(history)
#  Testing model on test data
**We only have 33 images in test data, so let's check the model on all
images**test_dir = "../input/new-plant-diseases-dataset/test"
test = ImageFolder(test_dir, transform=transforms.ToTensor())
test_images = sorted(os.listdir(test_dir + '/test')) # since images in test folder are in alphabetical
```

ordertest_images

```
def predict_image(img, model):
    """Converts image to array and return the
       predicted classwith highest probability"""
    # Convert to a batch of 1
    xb = to_device(img.unsqueeze(0),
    device)# Get predictions from
    model
    yb = model(xb)
    # Pick index with highest probability
    _, preds  = torch.max(yb,
    dim=1)# Retrieve the class
    label

    return
train.classes[preds[0].item()]#
predicting first image
img, label = test[0]
plt.imshow(img.permute(1,
2, 0))
print('Label:', test_images[0], ', Predicted:', predict_image(img,
model))# getting all predictions (actual label vs predicted)
for i, (img, label) in enumerate(test):
    print('Label:', test_images[i], ', Predicted:', predict_image(img, model))
```

We can see that the model predicted all the test images perfectly!!!!# Saving the model

There are several ways to save the model in Pytorch, following are the two most common ways

1. **Save/Load `state_dict` (Recommended)**

When saving a model for inference, it is only necessary to save the trained model's learned parameters. Savingthe model's `state_dict` with the `torch.save()` function will give you the most flexibility for restoring the model later, which is why it is the recommended method for saving models.

A common PyTorch convention is to save models using either a `.pt` or `.pth` file extension.

Remember that you must call `model.eval()` to set dropout and batch normalization layers to evaluation modebefore running inference. Failing to do this will yield inconsistent inference results.

```
# saving to the
kaggle working
directoryPATH =
'./plant-disease-
model.pth'
torch.save(model.sta
te_dict(), PATH)
2. **Save/Load Entire Model**
```

This save/load process uses the most intuitive syntax and involves the least amount of code. Saving a model inthis way will save the entire module using Python's [pickle](https://docs.python.org/3/library/pickle.html) module. The disadvantage of this approach is that the serialized data is bound to the specific classes and theexact directory structure used when the model is saved. The reason for this is because pickle does not save the model class itself. Rather, it saves a path to the file containing the class, which is used during load time. Because of this, your code can break in various ways when used in other projects or after refactors.

```
# saving the entire model
to working directory
PATH = './plant-disease-
model-complete.pth'
torch.save(model, PATH)
```

Conclusion

ResNets perform significantly well for image classification when some of the parameters are tweaked and techniques like scheduling learning rate, gradient clipping and weight decay are applied. The model is able topredict every image in test set perfectly without any errors !!!!

GitHub

https://github.com/IBM-EPBL/IBM-Project-53592-1661420126