Deep Learning Model For Detecting Diseases In Tea Leaves

Team-ID: *592127*

Team Details:

- *Medi Pavan Teja*
- *R Lokesh Kanna*

1. INTRODUCTION:

1.1 Project Overview

Tea, a widely consumed beverage globally, stands as a crucial agricultural commodity with significant economic implications worldwide. The well-being of tea plants plays a pivotal role in determining both crop yield and quality. Early detection of diseases in tea leaves is essential for the sustainable cultivation of tea. This initiative presents a deep learning approach designed to automate the recognition and categorization of diseases impacting tea leaves. The methodology leverages the robust YOLO (You Only Look Once) object detection model.

1.2 Purpose

The central objective of this project is to create a precise and efficient deep learning model capable of analyzing tea leaf images to detect potential diseases. Conventional disease detection methods often hinge on manual inspection, which can be time-intensive and may lack the sensitivity required for early identification of diseases. By utilizing deep learning, the project seeks to deliver a quicker, more dependable, and automated approach to detecting diseases in tea leaves.

2. LITERATURE SURVEY:

2.1 Existing Problem

The detection and management of diseases in agricultural crops, including tea leaves, have historically posed significant challenges. Traditional methods rely heavily on manual inspection, which is labor-intensive, time-consuming, and often subjective. These limitations can result in delayed detection of diseases, leading to reduced crop yields and economic losses for farmers. In recent years, the application of deep learning techniques to agricultural problems has shown promising results. Object detection models, particularly YOLO (You Only Look Once), have gained popularity due to their ability to process images in real-time and identify multiple objects with high accuracy. While these models have been successfully applied in various domains, their specific application to tea leaf disease detection is an area that warrants exploration and research.

2.2 References

1. Soeb, Md Janibul Alam, et al. "Tea leaf disease detection and identification based on YOLOv7 (YOLO-T)." Scientific reports 13.1 (2023): 6078.

2. Xue, Zhenyang, et al. "YOLO-tea: A tea disease detection model improved by YOLOv5." Forests 14.2 (2023): 415.

3. Hu, Gensheng, et al. "Detection and severity analysis of tea leaf blight based on deep learning." Computers & Electrical Engineering 90 (2021): 107023.

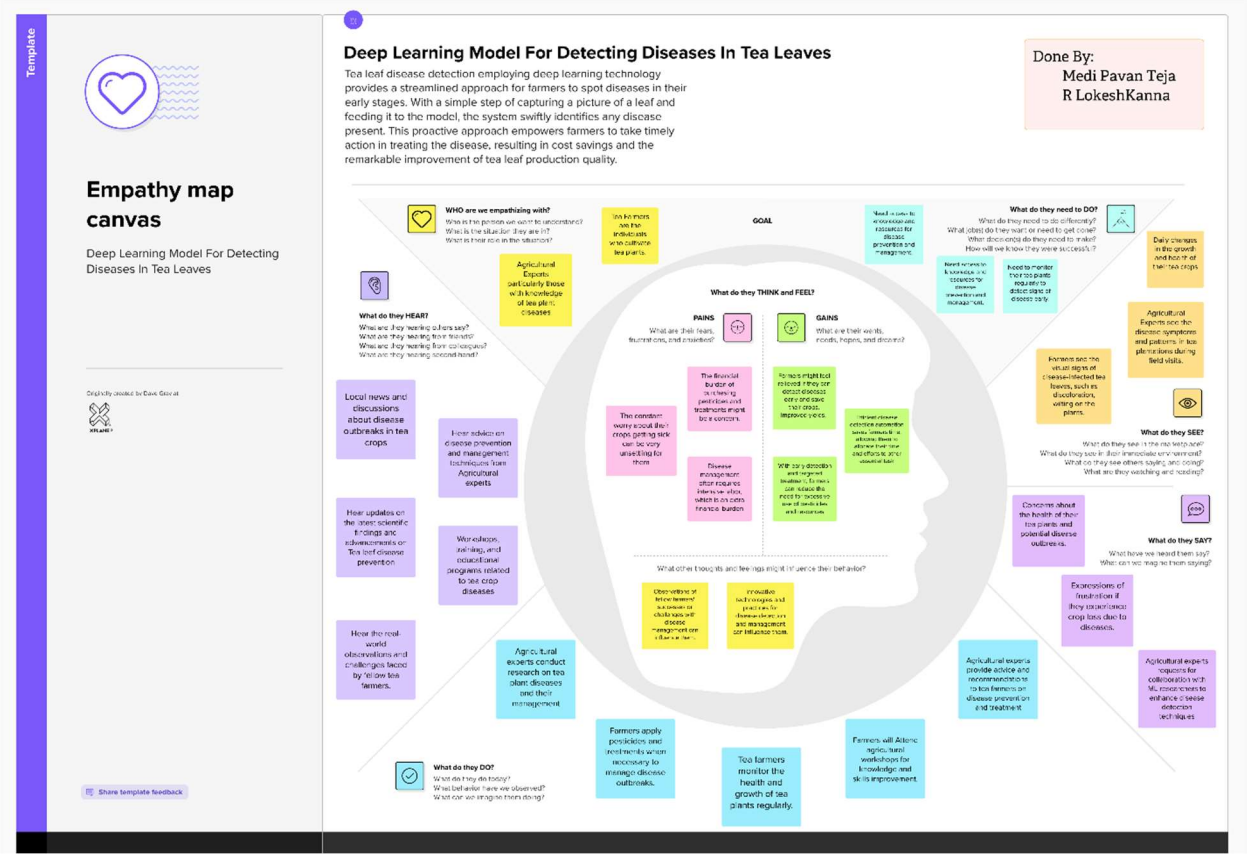
4. Gayathri, S., et al. "Image analysis and detection of tea leaf disease using deep learning." 2020 International Conference on Electronics and Sustainable Communication Systems (ICESC). IEEE, 2020.

2.3 Problem Statement Definition

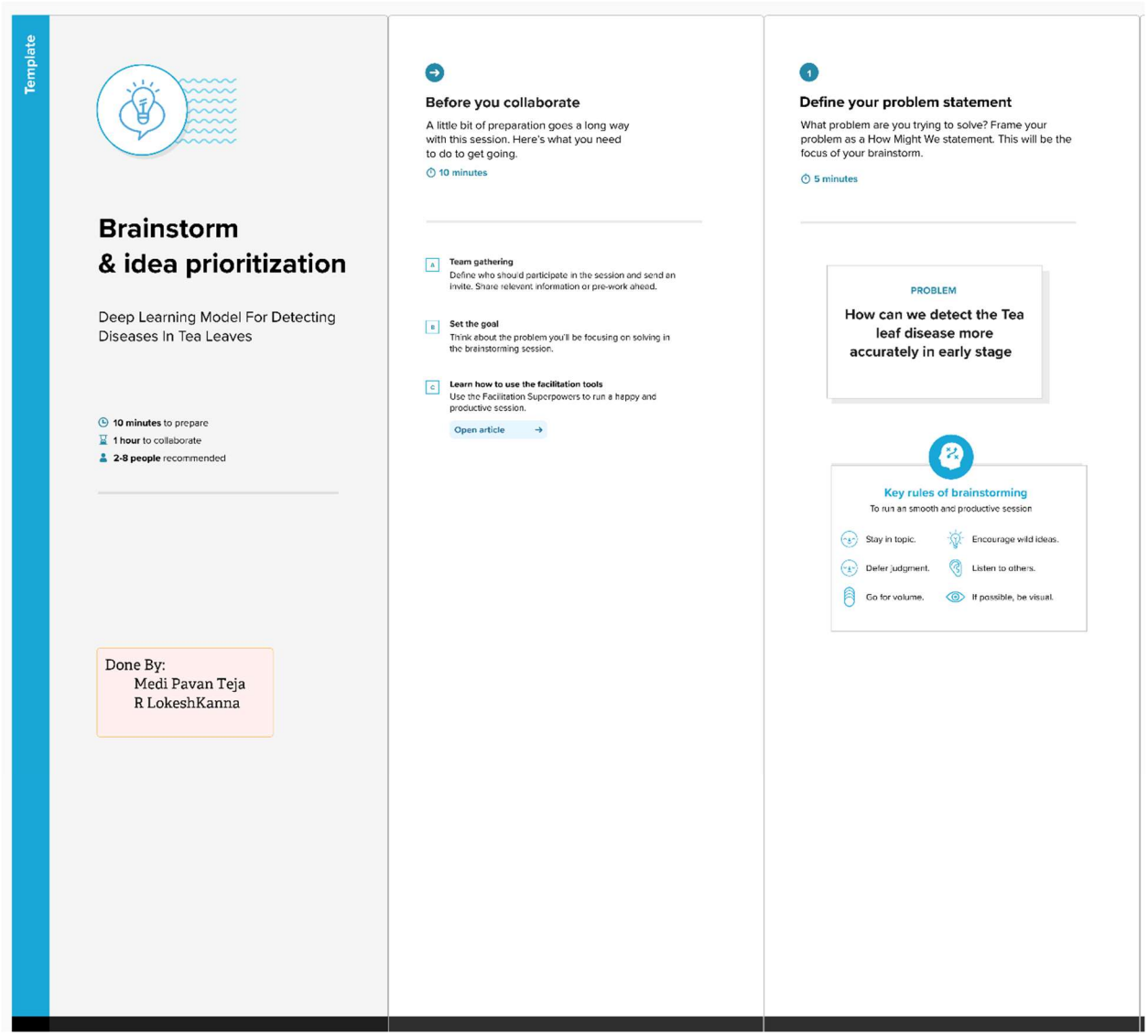
In the realm of tea cultivation, the pressing challenge lies in the prompt and accurate identification of diseases in tea leaves—a crucial element for sustaining high-quality tea production. Current methods, which rely on manual inspection, often falter in promptly detecting diseases in their early stages, impeding swift intervention. This poses a threat to the sustainability and quality of tea, with potential impacts on yield and product standards due to disease outbreaks. The overarching objective is to create an innovative and cost-effective solution that utilizes advanced deep learning techniques such as YOLO to address these challenges. Through this, the project aims to reduce reliance on chemicals, minimize environmental impact, and make the solution accessible to tea farmers of all scales. The envisioned impact encompasses not only strengthened tea plantations but also a transition toward environmentally conscious and economically viable practices in the tea industry.

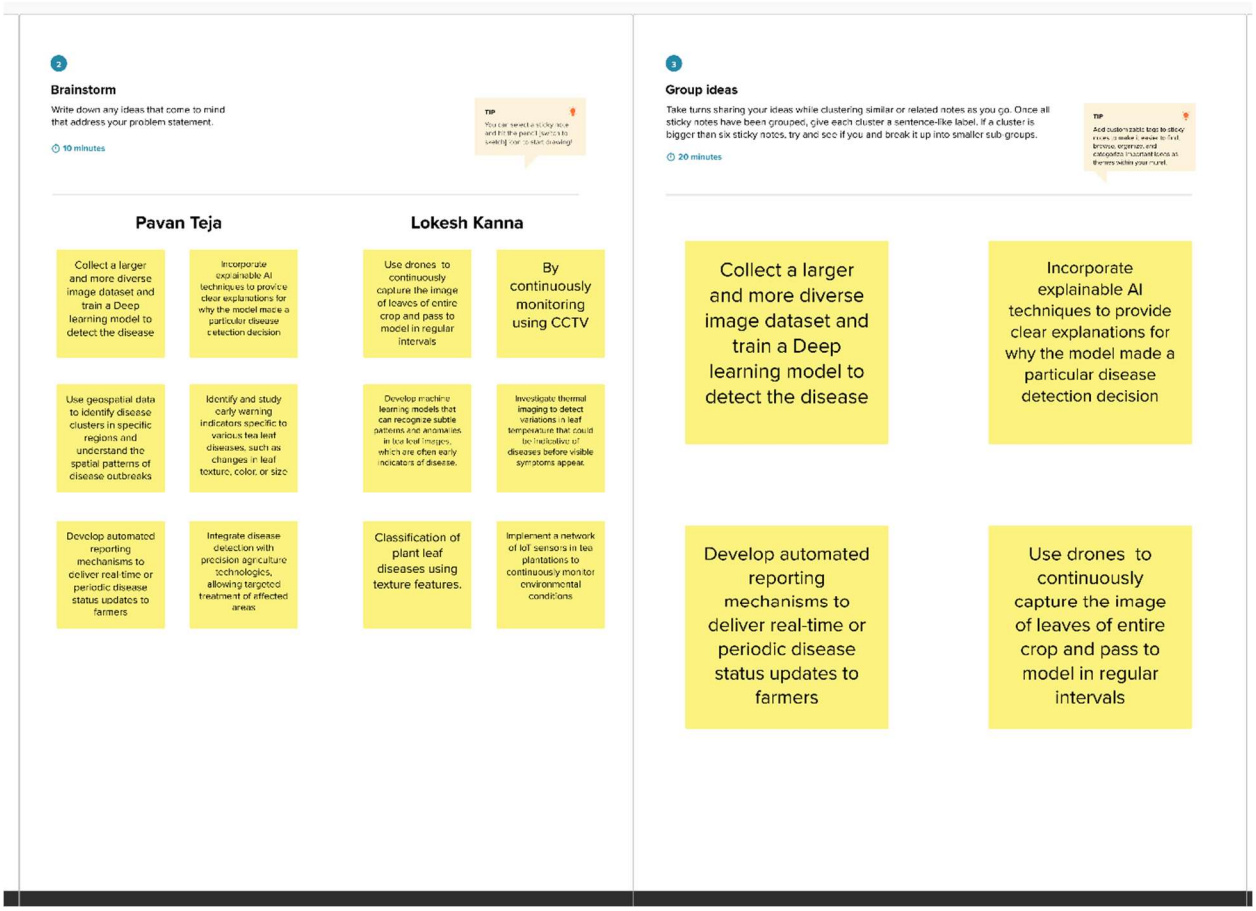
3. IDEATION & PROPOSED SOLUTION

3.1 Empathy Map Canvas



3.2 Ideation & Brainstorming




2
Brainstorm
Write down any ideas that come to mind that address your problem statement.
10 minutes
TIP
Pavan Teja
Collect a larger and more diverse image dataset and train a Deep learning model to detect the disease
Incorporate explainable AI techniques to provide clear explanations for why the model made a particular disease detection decision
Use geospatial data to identify disease clusters in specific regions and understand the spatial patterns of disease outbreaks
Identify and study early warning indicators specific to various tea leaf diseases, such as changes in leaf texture, color, or size
Develop automated reporting mechanisms to deliver real-time or periodic disease status updates to farmers
Integrate disease detection with precision agriculture technologies, allowing targeted treatment of affected areas
Lokesh Kanna
Use drones to continuously capture the image of leaves of entire crop and pass to model in regular intervals
By continuously monitoring using CCTV
Develop machine learning models that can recognize subtle patterns and anomalies in tea leaf images, which are often early indicators of disease.
Investigate thermal imaging to detect variations in leaf temperature that could be indicative of diseases before visible symptoms appear.
Classification of plant leaf diseases using texture features.
Implement a network of IoT sensors in tea plantations to continuously monitor environmental conditions
3
Group ideas
Take turns sharing your ideas while clustering similar or related notes as you go. Once all sticky notes have been grouped, give each cluster a sentence-like label. If a cluster is bigger than six sticky notes, try and see if you and break it up into smaller sub-groups.
20 minutes
TIP
Collect a larger and more diverse image dataset and train a Deep learning model to detect the disease
Incorporate explainable AI techniques to provide clear explanations for why the model made a particular disease detection decision
Develop automated reporting mechanisms to deliver real-time or periodic disease status updates to farmers
Use drones to continuously capture the image of leaves of entire crop and pass to model in regular intervals

4

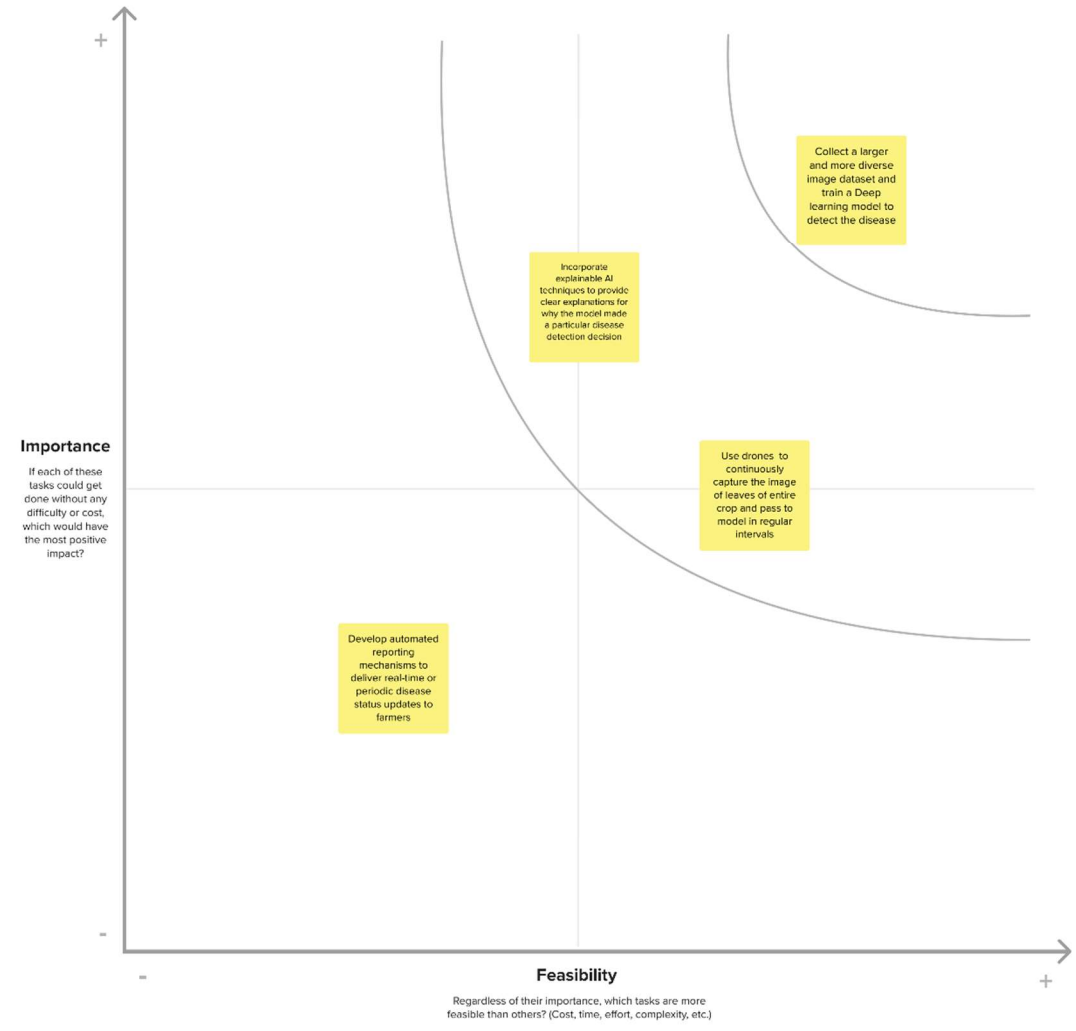
Prioritize

Your team should all be on the same page about what's important moving forward. Place your ideas on this grid to determine which ideas are important and which are feasible.

20 minutes

TIP

Participants can use their cursors to point at where sticky notes should go on the grid. The facilitator can confirm the spot by using the laser pointer holding the **H key** on the keyboard.



4. REQUIREMENT ANALYSIS

4.1 Functional requirement

a. Image Processing: The system must effectively process input images of tea leaves, extracting relevant features for disease detection.

b. Disease Classification: Implement a classification mechanism within the YOLOv8 model to accurately categorize identified regions as various diseases affecting tea leaves.

c. Real-Time Processing: Enable the system to process images in real-time or near-real-time, allowing for swift analysis in agricultural settings.

d. User Interface: Design a user-friendly interface that allows farmers to input images easily and view the results of disease detection, contributing to accessibility and usability.

e. Scalability: Ensure that the solution is scalable, capable of handling varying sizes of tea plantations and adapting to different datasets for training and testing.

4.2 Non-Functional requirements

a. Accuracy: The model must exhibit high accuracy in disease detection, ensuring reliable results to support timely interventions.

b. Robustness: The system should be robust, capable of handling variations in environmental conditions, lighting, and diverse appearances of tea leaves.

c. Resource Efficiency: Optimize the system for efficient utilization of computational resources to ensure accessibility across different computing environments.

d. Security: Implement security measures to protect the confidentiality and integrity of the data, especially considering the sensitivity of agricultural information.

e. Usability: Design the system with a focus on usability, considering the varying technical expertise of end-users, such as tea farmers.

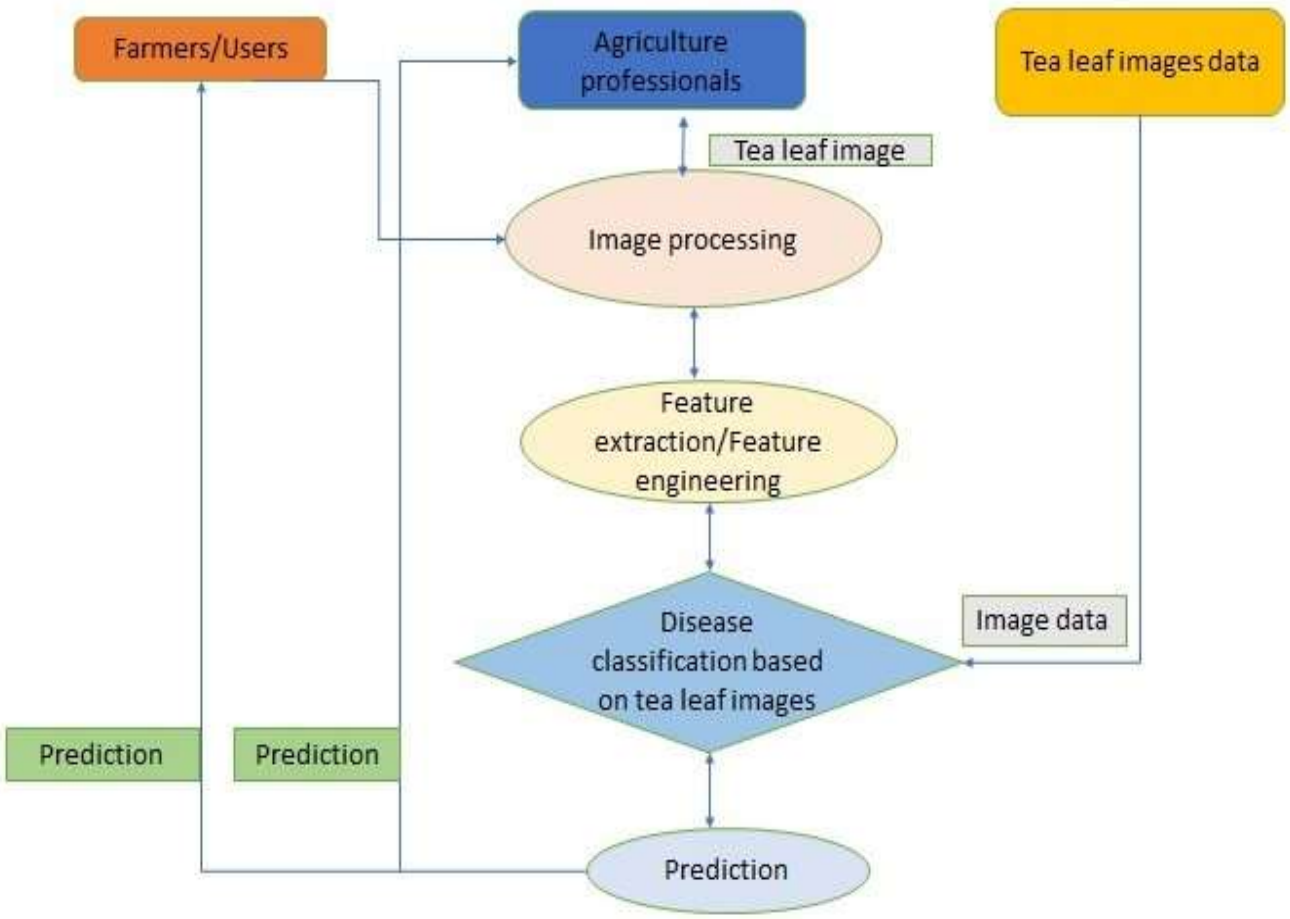
f. Compliance with Regulations: Ensure that the solution complies with relevant data protection and agricultural regulations, adhering to ethical and legal standards.

g. Adaptability: The solution should be adaptable to evolving technological advancements, allowing for updates and improvements as new methods in deep learning and disease detection emerge.

5. PROJECT DESIGN

5.1 Data Flow Diagrams & User Stories

A Data Flow Diagram (DFD) is a traditional visual representation of the information flows within a system. A neat and clear DFD can depict the right amount of the system requirement graphically. It shows how data enters and leaves the system, what changes the information, and where data is stored.

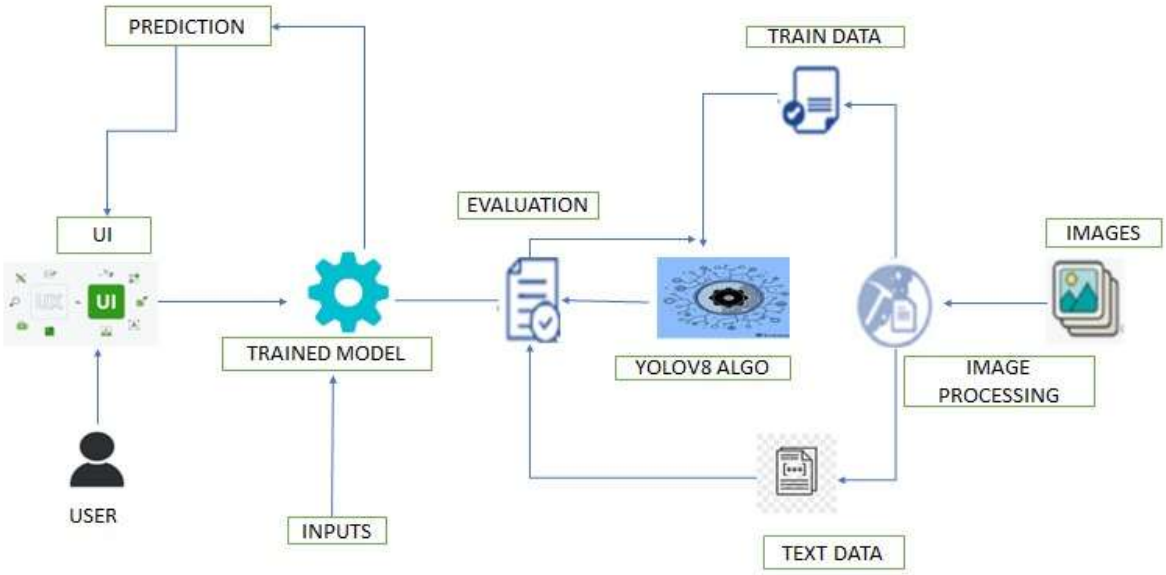


User Stories:

User Type	Functional Requirement (Epic)	User Story Num	User Story / Task	Acceptance criteria	Priority	Release
Farmers (Web User)	Registration	USN-1	As farmer, I want to fill my details like name, mail id, password to sign up so that I can have access to my account whenever I need.	I can access my account Through registered mail id.	High	Sprint -1
Farmers (Web User)	Password Recovery	USN-2	As a user, I want the option to recover my account password in case I forget it, to regain access to my account.	The system should provide a secure password recovery mechanism via email or phone verification.	High	Sprint -1
Farmers (Web User)	Access Account History	USN-3	As a user, I want to view my account activity and transaction history to track my usage and monitor any changes or activities within my account.	The system should provide a clear and comprehensive account history log, including details of transactions, logins, and other relevant activities.	High	Sprint -1
Farmers (Web User)	Results Visualization	USN-4	As farmer, I want a visual representation of the disease detection results to facilitate decision-making.	The system should display the model's predictions, highlighting areas of concern on the tea leaf images, and provide a confidence score for each prediction.	Medium	Sprint -2
System Administrators	Model Maintenance	USN-5	As a system administrator, I want tools to monitor and maintain the health of the deep learning model over time.	The system should provide logs, alerts, and tools for retraining the model periodically to ensure its effectiveness in detecting diseases in tea leaves.	Medium	Sprint -2

Farmers (Web User)	Real-time Prediction	USN-6	As a farmer, I want the ability to get real-time predictions for disease detection on live camera feed.	The system should support real-time prediction using a camera feed, providing instant feedback on the health status of tea leaves.	Low	Sprint -3
Farmers (Web User)	Mobile Accessibility	USN-7	As an agricultural researcher/farmer, I want to access the disease detection model on my mobile device for on-the-go monitoring.	The system should be accessible and user-friendly on mobile devices, allowing users to check disease predictions and results anytime, anywhere.	Medium	Sprint -2
Data Scientists	Model Customization	USN-8	As a data scientist, I want the ability to customize and fine-tune the deep learning model parameters for optimal performance.	The system should provide options for data scientists to adjust hyperparameters, model architecture, and training settings to enhance model accuracy.	High	Sprint -1

5.2 Solution Architecture



6. PROJECT PLANNING & SCHEDULING

6.1 Technical Architecture

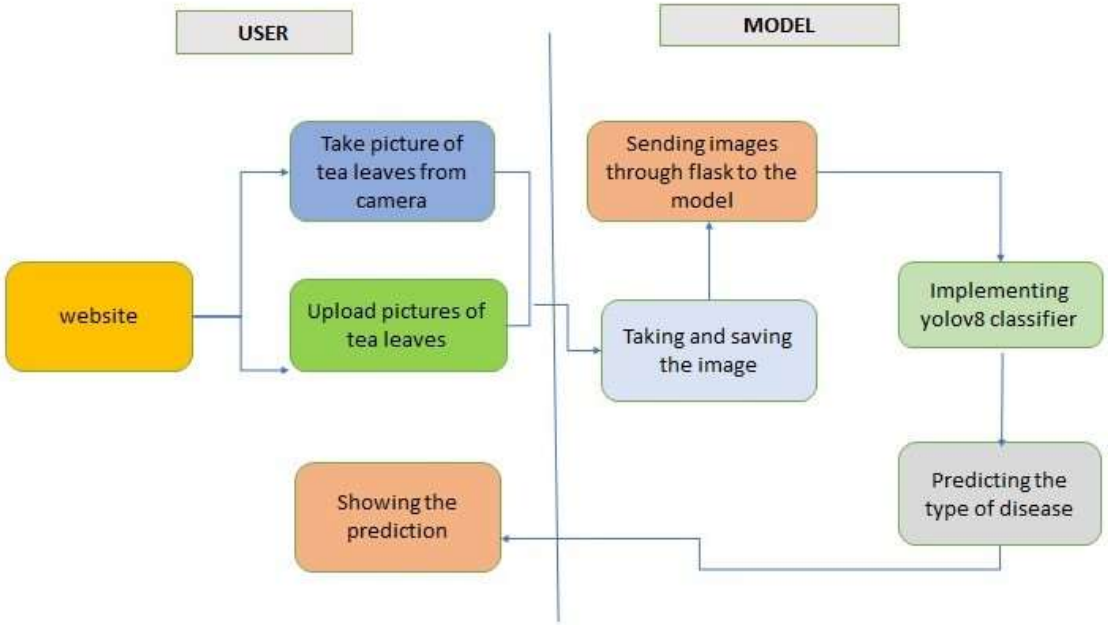


Table-1: Components & Technologies:

S.No	Characteristics	Description	Technology
1.	Open-Source Frameworks	Open-source frameworks refer to using software frameworks that are publicly accessible, can be modified.	TensorFlow, Ultralytis, Keras & Flask
2.	Security Implementations	Security implementations are essential to protect patient data and confidentiality of the image processing system. As this is a simple model it is not necessary of security implementations.	None
3.	Scalable Architecture	AWS Well-Architected/Azure Architecture provides a framework to help cloud architects build secure, high-performing, resilient, and efficient architectures.	AWS or Microsoft Azure
4.	Availability	Use of load balancers to distribute traffic, and geographically distributed servers to ensure high availability.	AWS or Microsoft Azure
5.	Performance	Performance of model involves ensures that application can handle a significant number of image processing requests per second, use of cache and CDN's efficiently.	Utilize hardware acceleration (GPU's)

Table-2: Application Characteristic

S.No	Component	Description	Technology
1.	User Interface	Web UI	HTML, CSS, JavaScript
2.	Application Logic-1	Building YOLOv8 classifier	Python
3.	Application Logic-2	Use of flask to connect from model to web application	Flask
4.	Application Logic-3	Use of Camera to capture Image	Camera
5.	Deep Learning Model	To Recognize the Pattern of the Image	YOLOv8
6.	Infrastructure (Server / Cloud)	Application Deployment on Local System Local	VS-Code

6.2 Sprint Planning & Estimation

Product Backlog, Sprint Schedule, and Estimation.

Sprint	Functional Requirement (EPIC)	User Story Number	User Story/Task	Story Points	Priority	Team Members
Sprint1	Registration	USN-1	As a user, I want to fill my details like name, mail id, password to sign up so that I can have access to my account whenever I need.	3	High	Pavan
Sprint1	Password Recovery	USN-2	As a user, I want the option to recover my account password in case I forget it, to regain access to my account.	3	High	Pavan

Sprint1	Access Account History	USN-3	As a user, I want to view my account activity and transaction history to track my usage and monitor any changes or activities within my account.	2	High	Lokesh
Sprint1	Model Customization	USN-4	As a data scientist, I want the ability to customize and fine-tune the deep learning model parameters for optimal performance.	2	High	Pavan
Sprint2	Results Visualization	USN-5	As farmer, I want a visual representation of the disease detection results to facilitate decision-making.	2	Medium	Lokesh
Sprint2	Mobile Accessibility	USN-6	As an agricultural researcher/farmer, I want to access the disease detection model on my mobile device for on-the-go monitoring.	3	Medium	Lokesh
Sprint2	Model Maintenance	USN-7	As a system administrator, I want tools to monitor and maintain the health of the deep learning model over time.	3	Medium	Pavan
Sprint3	Real-time Prediction	USN-8	As a farmer, I want the ability to get real-time predictions for disease detection on live camera feed.	1	Low	Lokesh

6.3 Sprint Delivery Schedule

Project Tracker, Velocity & Burndown Chart: (4 Marks)

Sprint	Total Story Points	Duration	Sprint Start Date	Sprint End Date (Planned)	Story Points Completed (as on Planned Date)	Sprint Release Date (Actual)
Sprint1	10	5 Days	7 Nov 2023	12 Nov 2023	10	12 Nov 2023

Sprint2	8	4 Days	13 Nov 2023	17 Nov 2023	8	17 Nov 2023
Sprint3	1	1 Day	18 Nov 2023	18 Nov 2023	1	18 Nov 2023

Velocity:

Average Velocity (Sprint 1) = Sprint Duration/velocity = 10/5 = 2

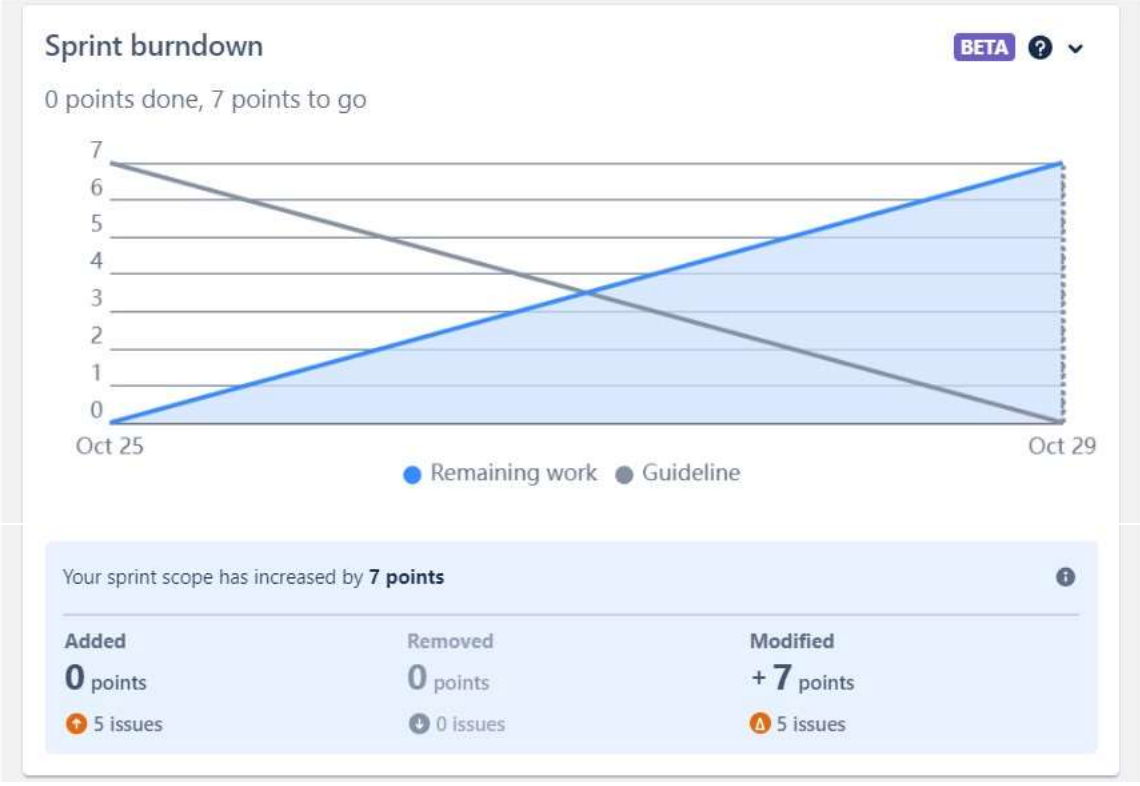
Average Velocity (Sprint 1) = Sprint Duration/velocity = 8/4 = 2

Average Velocity (Sprint 1) = Sprint Duration/velocity = 1/1 = 1

Total Average Velocity = 2

Burndown Chart:

A burn down chart is a graphical representation of work left to do versus time. It is often used in agile software development methodologies such as Scrum. However, burn down charts can be applied to any project containing measurable progress over time.



7. CODING & SOLUTIONING

7.1 Feature 1 – Model Training

a. Data Collection

https://www.kaggle.com/datasets/saikatdatta1994/tea-leaf-disease

The following dataset has been used. It was taken from Kaggle, which consists of 5867 images of total 6 classes. Each class having approximately 1000 images, so this was a well balanced dataset.

This data set was not in the required format so we need to make the data in our required format.

b. Importing necessary libraries

```
import os
import kaggle
import shutil
import ultralytics
from ultralytics import YOLO
from sklearn.model_selection import train_test_split
```

c. Download the Data

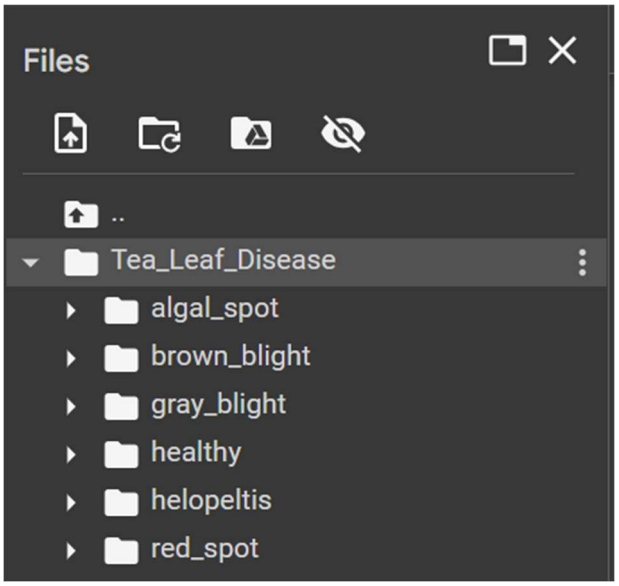
```
[4]  from google.colab import files

     # Upload the Kaggle API key
     files.upload()
```

```
Choose Files  kaggle.json
• kaggle.json(application/json) - 69 bytes, last modified: 11/3/2023 - 100% done
Saving kaggle.json to kaggle.json
{'kaggle.json': b'{"username":"pavantejamedi","key":"[redacted]"}'}
```

```
import os

# Move the Kaggle API key to the required location
os.makedirs('/root/.kaggle', exist_ok=True)
os.rename('kaggle.json', '/root/.kaggle/kaggle.json')

# Set permissions for the API key
os.chmod('/root/.kaggle/kaggle.json', 600)
```

```
[6] import kaggle

# Define the username and dataset name
username = "saikatdatta1994"
dataset_name = "tea-leaf-disease"

# Download the dataset
kaggle.api.dataset_download_files(username + '/' + dataset_name, unzip=True)
```

The downloaded data will be in this format:



Classes Detected:

```
In [3]: print(result)

[ultralytics.engine.results.Results object with attributes:

boxes: None
keypoints: None
keys: ['probs']
masks: None
names: {0: 'algal_spot', 1: 'brown_blight', 2: 'gray_blight', 3: 'healthy',
4: 'helopeltis', 5: 'red_spot'}
```

Task-4: Prepare the Data

The Data should be in the below format.

```
Dataset
|
└──train
|   └──ClassA
|       |   ClassA_1.jpg
|       |   ClassA_2.jpg
|       |   ...
|   └──ClassB
|       |   ClassB_1.jpg
|       |   ClassB_2.jpg
|       |   ...
|   └──...
```

```
└──test
|   └──ClassA
|       |   ClassA_9090.jpg
|       |   ClassA_9895.jpg
|       |   ...
|   └──ClassB
|       |   ClassB_2343.jpg
|       |   ClassB_2312.jpg
|       |   ...
|   └──...
└──val
|   └──ClassA
|       |   ClassA_3070.jpg
|       |   ClassA_2845.jpg
|       |   ...
|   └──ClassB
|       |   ClassB_2903.jpg
|       |   ClassB_2232.jpg
|       |   ...
|   └──...
```

To make it in this format we first create the folder named “dataset”

```
[ ]  !mkdir '/content/dataset'
     DATA_DIR='/content/dataset'
```

Make the subfolders train, test, val in the “dataset” folder

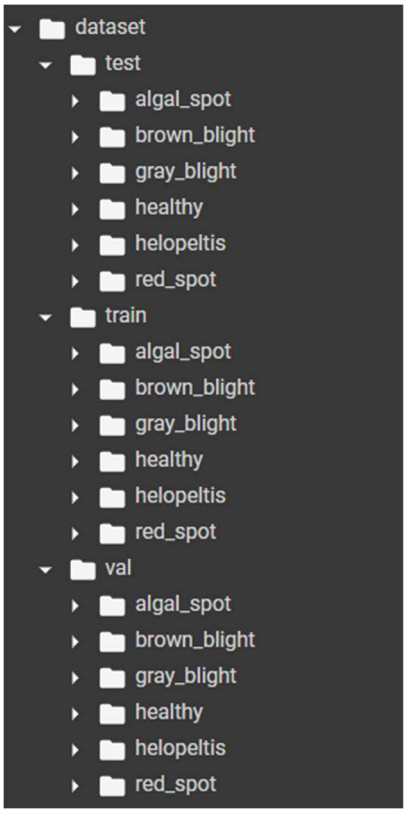
```
[ ]  source_dataset_path = '/content/Tea_Leaf_Disease'
     destination_dataset_path = '/content/dataset'
```

```
[ ]  os.makedirs(os.path.join(destination_dataset_path, 'train'), exist_ok=True)
     os.makedirs(os.path.join(destination_dataset_path, 'test'), exist_ok=True)
     os.makedirs(os.path.join(destination_dataset_path, 'val'), exist_ok=True)
```

Split the data in the source dataset into train, test, val randomly using train_test_split from sklearn and store it in the “dataset” folder.

```
[ ]  # List of classes
     classes = os.listdir(source_dataset_path)

     # Loop through each class
     for class_name in classes:
         class_dir = os.path.join(source_dataset_path, class_name)

         # Split the images for the current class into train, test, and val sets
         train_images, test_images = train_test_split(os.listdir(class_dir), test_size=0.2, random_state=42)
         val_images, test_images = train_test_split(test_images, test_size=0.5, random_state=42)

         # Create subdirectories for each class in train, test, and val
         os.makedirs(os.path.join(destination_dataset_path, 'train', class_name), exist_ok=True)
         os.makedirs(os.path.join(destination_dataset_path, 'test', class_name), exist_ok=True)
         os.makedirs(os.path.join(destination_dataset_path, 'val', class_name), exist_ok=True)

         # Move images to their respective directories
         for image in train_images:
             shutil.copy(os.path.join(class_dir, image), os.path.join(destination_dataset_path, 'train', class_name, image))
         for image in test_images:
             shutil.copy(os.path.join(class_dir, image), os.path.join(destination_dataset_path, 'test', class_name, image))
         for image in val_images:
             shutil.copy(os.path.join(class_dir, image), os.path.join(destination_dataset_path, 'val', class_name, image))
```

The final structure of dataset looks like this



Task-5: Load the Model

```
[ ]  # import YOLO model
     from ultralytics import YOLO

     # Load a model
     model = YOLO('yolov8n-cls.pt')
```

Task-6: Train the Model

```
[ ]  # Train the model
     results = model.train(data='/content/dataset', epochs=35, imgsz=64)
```

```
Downloading https://github.com/ultralytics/assets/releases/download/v0.0.0/yolov8n-cls.pt to 'yolov8n-cls.pt'...
100%|██████████| 5.28M/5.28M [00:00<00:00, 103MB/s]
Ultralytics YOLOv8.0.209 🚀 Python-3.10.12 torch-2.1.0+cu118 CUDA:0 (Tesla T4, 15102MiB)
engine/trainer: task=classify, mode=train, model=yolov8n-cls.pt, data=/content/dataset, epochs=35, patience=50,
train: /content/dataset/train... found 4693 images in 6 classes ✅
val: /content/dataset/val... found 587 images in 6 classes ✅
test: /content/dataset/test... found 587 images in 6 classes ✅
Overriding model.yaml nc=1000 with nc=6
```

```
YOLOv8n-cls summary: 99 layers, 1445974 parameters, 1445974 gradients, 3.4 GFLOPs
Transferred 156/158 items from pretrained weights
TensorBoard: Start with 'tensorboard --logdir runs/classify/train', view at http://localhost:6006/
AMP: running Automatic Mixed Precision (AMP) checks with YOLOv8n...
Downloading https://github.com/ultralytics/assets/releases/download/v0.0.0/yolov8n.pt to 'yolov8n.pt'...
100%|██████████| 6.23M/6.23M [00:00<00:00, 255MB/s]
AMP: checks passed ✅
train: Scanning /content/dataset/train... 4693 images, 0 corrupt: 100%|██████████| 4693/4693 [00:00<00:00, 5217.49it/
train: New cache created: /content/dataset/train.cache
albumentations: RandomResizedCrop(p=1.0, height=64, width=64, scale=(0.5, 1.0), ratio=(0.75, 1.3333333333333333), int
val: Scanning /content/dataset/val... 587 images, 0 corrupt: 100%|██████████| 587/587 [00:00<00:00, 3515.94it/s]
val: New cache created: /content/dataset/val.cache
optimizer: 'optimizer=auto' found, ignoring 'lr0=0.01' and 'momentum=0.937' and determining best 'optimizer', 'lr0' a
optimizer: AdamW(lr=0.000714, momentum=0.9) with parameter groups 26 weight(decay=0.0), 27 weight(decay=0.0005), 27 b
Image sizes 64 train, 64 val
Using 2 dataloader workers
Logging results to runs/classify/train
Starting training for 35 epochs...

      Epoch    GPU_mem       loss  Instances       Size
       1/35     0.384G     0.4849         16         64:  11%|▉         | 33/294 [00:02<00:16, 15.65it/s]Downloading
       1/35     0.384G     0.4563         16         64:  25%|██▍        | 73/294 [00:05<00:16, 13.26it/s]
100%|██████████| 755k/755k [00:00<00:00, 40.2MB/s]
       1/35     0.384G     0.3088          5         64: 100%|██████████| 294/294 [00:21<00:00, 13.67it/s]
                 classes   top1_acc   top5_acc: 100%|██████████| 19/19 [00:01<00:00, 15.88it/s]
                    all      0.785          1
```

```
      Epoch    GPU_mem       loss  Instances       Size
       2/35     0.363G     0.1491          5         64: 100%|██████████| 294/294 [00:15<00:00, 18.95it/s]
                 classes   top1_acc   top5_acc: 100%|██████████| 19/19 [00:00<00:00, 24.66it/s]
                    all      0.847          1

      Epoch    GPU_mem       loss  Instances       Size
       3/35     0.363G     0.1269          5         64: 100%|██████████| 294/294 [00:12<00:00, 23.23it/s]
                 classes   top1_acc   top5_acc: 100%|██████████| 19/19 [00:00<00:00, 20.81it/s]
                    all      0.862      0.998

      Epoch    GPU_mem       loss  Instances       Size
       4/35     0.363G     0.1063          5         64: 100%|██████████| 294/294 [00:12<00:00, 24.15it/s]
                 classes   top1_acc   top5_acc: 100%|██████████| 19/19 [00:00<00:00, 20.64it/s]
                    all      0.864          1

      Epoch    GPU_mem       loss  Instances       Size
       5/35     0.363G    0.09134          5         64: 100%|██████████| 294/294 [00:12<00:00, 23.64it/s]
                 classes   top1_acc   top5_acc: 100%|██████████| 19/19 [00:00<00:00, 36.35it/s]
                    all      0.882      0.998

      Epoch    GPU_mem       loss  Instances       Size
       6/35     0.363G    0.08104          5         64: 100%|██████████| 294/294 [00:13<00:00, 22.24it/s]
                 classes   top1_acc   top5_acc: 100%|██████████| 19/19 [00:00<00:00, 38.15it/s]
                    all      0.881          1
```

```
      Epoch    GPU_mem       loss  Instances       Size
      35/35     0.363G    0.01391          5         64: 100%|██████████| 294/294 [00:15<00:00, 19.01it/s]
                classes   top1_acc   top5_acc: 100%|██████████| 19/19 [00:00<00:00, 23.05it/s]
                    all      0.966          1

35 epochs completed in 0.142 hours.
Optimizer stripped from runs/classify/train/weights/last.pt, 3.0MB
Optimizer stripped from runs/classify/train/weights/best.pt, 3.0MB

Validating runs/classify/train/weights/best.pt...
Ultralytics YOLOv8.0.209 🚀 Python-3.10.12 torch-2.1.0+cu118 CUDA:0 (Tesla T4, 15102MiB)
YOLOv8n-cls summary (fused): 73 layers, 1442566 parameters, 0 gradients, 3.3 GFLOPs
train: /content/dataset/train... found 4693 images in 6 classes ✅
val: /content/dataset/val... found 587 images in 6 classes ✅
test: /content/dataset/test... found 587 images in 6 classes ✅
                classes   top1_acc   top5_acc: 100%|██████████| 19/19 [00:01<00:00, 18.30it/s]
                    all      0.973          1
Speed: 0.1ms preprocess, 1.0ms inference, 0.0ms loss, 0.0ms postprocess per image
Results saved to runs/classify/train
Results saved to runs/classify/train
```

Task-7: Save the Model

The model will be automatically saved under the runs folder with the name “best.pt” after the completion of training.

Task-8: Validate the Model

```
[ ]  from ultralytics import YOLO

     model = YOLO('/content/runs/classify/train/weights/best.pt')  # load a custom model

     # Validate the model
     metrics = model.val()  # no arguments needed, dataset and settings remembered
     metrics.top1   # top1 accuracy
     metrics.top5   # top5 accuracy
```

```
     Ultralytics YOLOv8.0.209 🚀 Python-3.10.12 torch-2.1.0+cu118 CUDA:0 (Tesla T4, 15102MiB)
     YOLOv8n-cls summary (fused): 73 layers, 1442566 parameters, 0 gradients, 3.3 GFLOPs
     train: /content/dataset/train... found 4693 images in 6 classes ✅
     val: /content/dataset/val... found 587 images in 6 classes ✅
     test: /content/dataset/test... found 587 images in 6 classes ✅
     val: Scanning /content/dataset/val... 587 images, 0 corrupt: 100%|██████████| 587/587 [00:00<?, ?it/s]
                    classes   top1_acc   top5_acc: 100%|██████████| 37/37 [00:01<00:00, 29.48it/s]
                        all      0.973          1
     Speed: 0.0ms preprocess, 1.3ms inference, 0.0ms loss, 0.0ms postprocess per image
     Results saved to runs/classify/val
     1.0
```

Task-9: Test with custom image

```
from google.colab import files

# Upload an image
uploaded = files.upload()

# Get the image name
file_name = list(uploaded.keys())[0]

# Get the file path in Colab's temporary storage
file_path = f"/content/{file_name}"

# load a custom model
model = YOLO('/content/YOLOv8_Tea lef dise_Model.pt')

# Predict with the model
results = model(file_path)
```

```
Choose Files IMG_20220...140011.jpg
• IMG_20220503_140011.jpg(image/jpeg) - 2049368 bytes, last modified: 11/8/2023 - 100% done
Saving IMG_20220503_140011.jpg to IMG_20220503_140011.jpg

image 1/1 /content/IMG_20220503_140011.jpg: 64x64 gray light 0.99, bird eye spot 0.00,
Speed: 36.2ms preprocess, 3.7ms inference, 0.1ms postprocess per image at shape (1, 3,
```

Task-10: Download the Model

```
# Trigger the download
files.download('/content/runs/classify/train/weights/best.pt')
```

7.2 Feature 2 – Web Interface

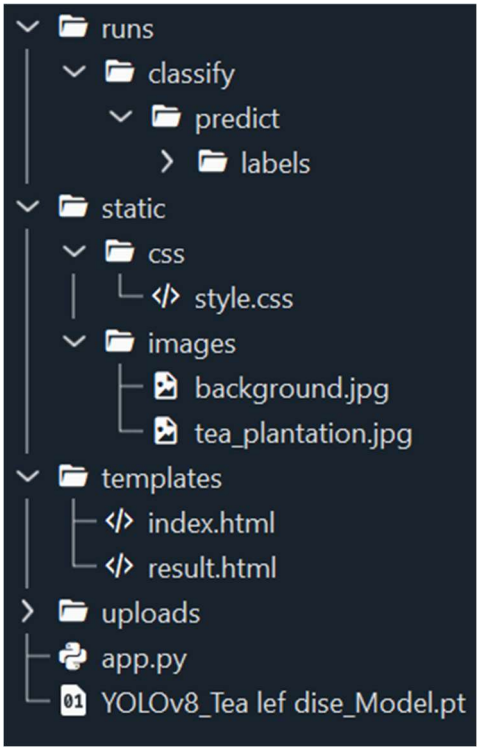
a. Application Building

- In our web development project, we employ HTML to construct the frontend of our web pages. Specifically, we have developed two HTML pages: `index.html` and `result.html`.
- The `index.html` page serves as the home page, providing users with the initial interface.

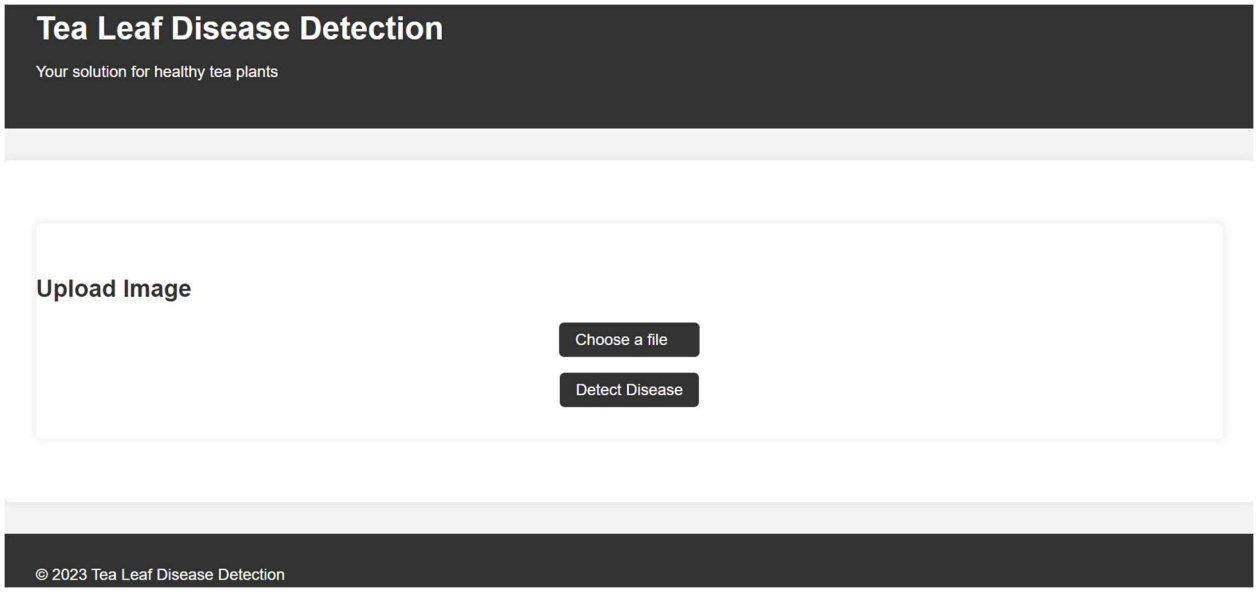
- On the other hand, `result.html` is responsible for receiving input images and presenting the corresponding predictions.
- To enhance the functionality and visual appeal of our web pages, we incorporate a CSS file named `main.css`.
- A ‘runs’ folder is used to store the model predictions, The prediction will be the classes and there confidence scores in the txt files.
- Furthermore, we leverage Flask as our web framework to deploy our Deep learning model. Flask facilitates the integration of our model into the web application, allowing users to interact with it seamlessly.

This combination of HTML, CSS, and Flask forms the backbone of our web architecture, enabling an effective and user-friendly deployment of our machine learning model.

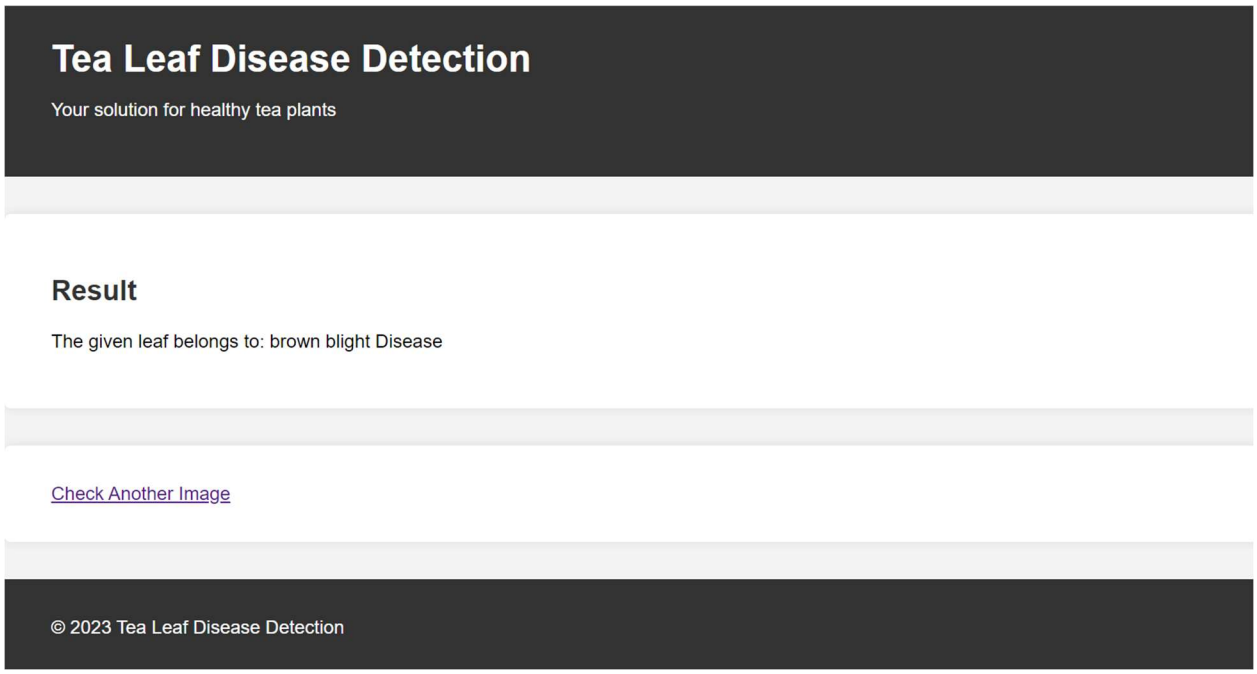
The folder structure to be maintained:



index.html



result.html



b. Python file creation

Importing necessary libraries

```
from flask import Flask, render_template, request, redirect, flash
import os
from ultralytics import YOLO
```

Loading the trained model and setting path to the upload folder to which the uploaded images will be stored

```
app = Flask(__name__)
# Set a secret key for flash messages
app.secret_key = 'your_secret_key'

# Load a pretrained YOLOv8n model
model = YOLO('D:/project/YOLOv8_Tea lef dise_Model.pt')

# Set the upload folder
UPLOAD_FOLDER = 'D:/flask_practice/uploads'
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER
```

Default home page

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

Upon clicking the "Detect" button, the uploaded image is saved in the "uploads" folder. Subsequently, the Python code captures the path of the uploaded image in the variable named file_path. This path is then relayed to the model for prediction. The resulting prediction is stored in a text file within the "Labels" directory, residing under the "predict" folder, itself situated in the "runs" directory. Now the final result will be extracted from the txt file and displays it in the result.html page.

```python
@app.route('/upload', methods=['POST'])
def upload_file():
    create_upload_folder()

    if 'file' not in request.files:
        flash('No file part', 'error')
        return redirect(request.url)

    file = request.files['file']

    if file.filename == '':
        flash('No selected file', 'error')
        return redirect(request.url)

    if file:
        filename = os.path.join(app.config['UPLOAD_FOLDER'], file.filename)
        file.save(filename)

        file_path = filename.replace('\\', '/')

        model.predict(file_path, save_txt=True, hide_conf=True, save=True)

        txt_files = os.listdir('runs/classify/predict/labels')
        txt_files.sort(key=lambda x: os.path.getmtime(os.path.join('runs/classify/predict/labels', x)), reverse=True)

        if txt_files:
            txt_file_path = os.path.join('runs/classify/predict/labels', txt_files[0])

            # Read the content of the first line excluding the first four characters
            with open(txt_file_path, 'r') as txt_file:
                first_line = txt_file.readline()[4:].strip()

            # Adjust the output b ased on the prediction
            if first_line.lower() == 'healthy':
                result = "The tea leaf is healthy!"
            else:
                result = f'The given leaf belongs to: {first_line} Disease'

            return render_template('result.html', result=result)

    flash('No files found in the "labels" directory.', 'error')
    return redirect(request.url)
```

This is used to run the application in a localhost.

```python
if __name__ == '__main__':
    app.run(debug=True)
```

This message will be displayed in the conssole

```
In [1]: runfile('D:/project/app.py', wdir='D:/project')
 * Serving Flask app "app" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it
a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Restarting with watchdog (windowsapi)
```

You can access the application by navigating to the following URL in your web browser:

'http://localhost:5000/'

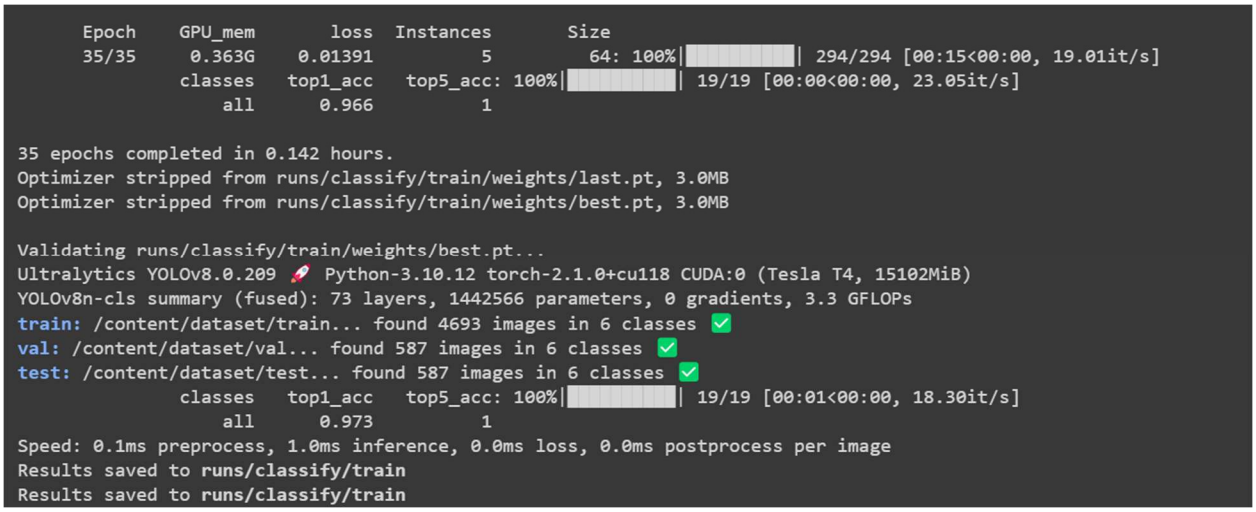
8. PERFORMANCE TESTING

Model Performance Testing:

S.No.	Parameter	Values
1.	Accuracy (Top-1)	Training Accuracy - 97.3 Validation Accuracy – 97.3
2.	Accuracy (Top-5)	Training Accuracy - 100 Validation Accuracy – 100

Screenshots

Training Accuracy:



Validation Accuracy:

```
[ ]  from ultralytics import YOLO

     model = YOLO('/content/runs/classify/train/weights/best.pt')  # load a custom model

     # Validate the model
     metrics = model.val()  # no arguments needed, dataset and settings remembered
     metrics.top1   # top1 accuracy
     metrics.top5   # top5 accuracy
```

```
Ultralytics YOLOv8.0.209 🚀 Python-3.10.12 torch-2.1.0+cu118 CUDA:0 (Tesla T4, 15102MiB)
YOLOv8n-cls summary (fused): 73 layers, 1442566 parameters, 0 gradients, 3.3 GFLOPs
train: /content/dataset/train... found 4693 images in 6 classes ✅
val: /content/dataset/val... found 587 images in 6 classes ✅
test: /content/dataset/test... found 587 images in 6 classes ✅
val: Scanning /content/dataset/val... 587 images, 0 corrupt: 100%|██████████| 587/587 [00:00<?, ?it/s]
               classes   top1_acc   top5_acc: 100%|██████████| 37/37 [00:01<00:00, 29.48it/s]
                   all      0.973          1
Speed: 0.0ms preprocess, 1.3ms inference, 0.0ms loss, 0.0ms postprocess per image
Results saved to runs/classify/val
1.0
```

Classes Detected:

```
In [3]: print(result)
```

```
[ultralytics.engine.results.Results object with attributes:

boxes: None
keypoints: None
keys: ['probs']
masks: None
names: {0: 'algal_spot', 1: 'brown_blight', 2: 'gray_blight', 3: 'healthy',
4: 'helopeltis', 5: 'red_spot'}
```

Entire Training Epochs:

```
Downloading https://github.com/ultralytics/assets/releases/download/v0.0.0/yolov8n-cls.pt to 'yolov8n-cls.pt'...
100%|██████████| 5.28M/5.28M [00:00<00:00, 103MB/s]
Ultralytics YOLOv8.0.209 🚀 Python-3.10.12 torch-2.1.0+cu118 CUDA:0 (Tesla T4, 15102MiB)
engine/trainer: task=classify, mode=train, model=yolov8n-cls.pt, data=/content/dataset, epochs=35, patience=50,
train: /content/dataset/train... found 4693 images in 6 classes ✅
val: /content/dataset/val... found 587 images in 6 classes ✅
test: /content/dataset/test... found 587 images in 6 classes ✅
Overriding model.yaml nc=1000 with nc=6
```

```
YOLOv8n-cls summary: 99 layers, 1445974 parameters, 1445974 gradients, 3.4 GFLOPs
Transferred 156/158 items from pretrained weights
TensorBoard: Start with 'tensorboard --logdir runs/classify/train', view at http://localhost:6006/
AMP: running Automatic Mixed Precision (AMP) checks with YOLOv8n...
Downloading https://github.com/ultralytics/assets/releases/download/v0.0.0/yolov8n.pt to 'yolov8n.pt'...
100%|██████████| 6.23M/6.23M [00:00<00:00, 255MB/s]
AMP: checks passed ✅
train: Scanning /content/dataset/train... 4693 images, 0 corrupt: 100%|██████████| 4693/4693 [00:00<00:00, 5217.49it/
train: New cache created: /content/dataset/train.cache
albumentations: RandomResizedCrop(p=1.0, height=64, width=64, scale=(0.5, 1.0), ratio=(0.75, 1.3333333333333333), int
val: Scanning /content/dataset/val... 587 images, 0 corrupt: 100%|██████████| 587/587 [00:00<00:00, 3515.94it/s]
val: New cache created: /content/dataset/val.cache
optimizer: 'optimizer=auto' found, ignoring 'lr0=0.01' and 'momentum=0.937' and determining best 'optimizer', 'lr0' a
optimizer: AdamW(lr=0.000714, momentum=0.9) with parameter groups 26 weight(decay=0.0), 27 weight(decay=0.0005), 27 b
Image sizes 64 train, 64 val
Using 2 dataloader workers
Logging results to runs/classify/train
Starting training for 35 epochs...

      Epoch    GPU_mem       loss  Instances       Size
       1/35     0.384G     0.4849         16         64:  11%|▉         | 33/294 [00:02<00:16, 15.65it/s]Downloading
       1/35     0.384G     0.4563         16         64:  25%|██▍        | 73/294 [00:05<00:16, 13.26it/s]
100%|██████████| 755k/755k [00:00<00:00, 40.2MB/s]
       1/35     0.384G     0.3088          5         64: 100%|██████████| 294/294 [00:21<00:00, 13.67it/s]
              classes   top1_acc   top5_acc: 100%|██████████| 19/19 [00:01<00:00, 15.88it/s]
                  all      0.785          1
```

```
      Epoch    GPU_mem       loss  Instances       Size
       2/35     0.363G     0.1491          5         64: 100%|██████████| 294/294 [00:15<00:00, 18.95it/s]
              classes   top1_acc   top5_acc: 100%|██████████| 19/19 [00:00<00:00, 24.66it/s]
                  all      0.847          1

      Epoch    GPU_mem       loss  Instances       Size
       3/35     0.363G     0.1269          5         64: 100%|██████████| 294/294 [00:12<00:00, 23.23it/s]
              classes   top1_acc   top5_acc: 100%|██████████| 19/19 [00:00<00:00, 20.81it/s]
                  all      0.862      0.998

      Epoch    GPU_mem       loss  Instances       Size
       4/35     0.363G     0.1063          5         64: 100%|██████████| 294/294 [00:12<00:00, 24.15it/s]
              classes   top1_acc   top5_acc: 100%|██████████| 19/19 [00:00<00:00, 20.64it/s]
                  all      0.864          1

      Epoch    GPU_mem       loss  Instances       Size
       5/35     0.363G    0.09134          5         64: 100%|██████████| 294/294 [00:12<00:00, 23.64it/s]
              classes   top1_acc   top5_acc: 100%|██████████| 19/19 [00:00<00:00, 36.35it/s]
                  all      0.882      0.998

      Epoch    GPU_mem       loss  Instances       Size
       6/35     0.363G    0.08104          5         64: 100%|██████████| 294/294 [00:13<00:00, 22.24it/s]
              classes   top1_acc   top5_acc: 100%|██████████| 19/19 [00:00<00:00, 38.15it/s]
                  all      0.881          1
```

```
      Epoch    GPU_mem       loss  Instances       Size
       7/35     0.363G    0.07359          5         64: 100%|██████████| 294/294 [00:13<00:00, 21.55it/s]
              classes   top1_acc   top5_acc: 100%|██████████| 19/19 [00:00<00:00, 36.75it/s]
                  all      0.893          1

      Epoch    GPU_mem       loss  Instances       Size
       8/35     0.363G    0.06513          5         64: 100%|██████████| 294/294 [00:13<00:00, 21.08it/s]
              classes   top1_acc   top5_acc: 100%|██████████| 19/19 [00:00<00:00, 36.20it/s]
                  all      0.917          1

      Epoch    GPU_mem       loss  Instances       Size
       9/35     0.363G    0.05679          5         64: 100%|██████████| 294/294 [00:13<00:00, 21.62it/s]
              classes   top1_acc   top5_acc: 100%|██████████| 19/19 [00:00<00:00, 35.75it/s]
                  all      0.927          1
```

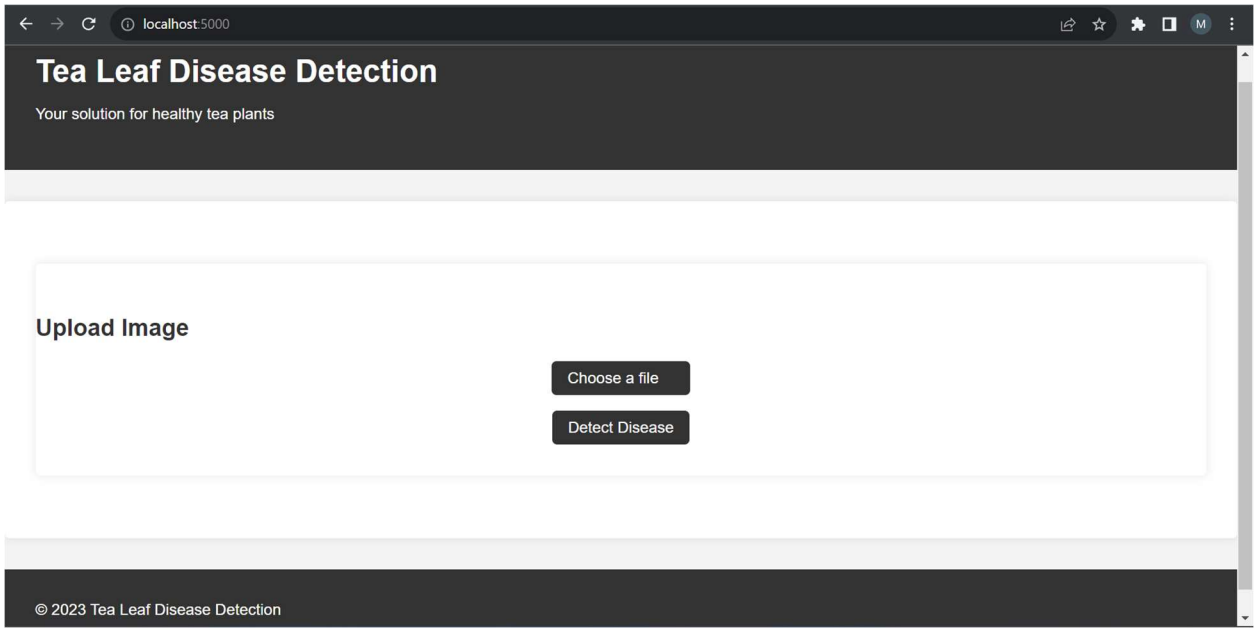
```
      Epoch    GPU_mem       loss  Instances       Size
      10/35     0.363G    0.05083          5         64: 100%|██████████| 294/294 [00:15<00:00, 18.69it/s]
                classes   top1_acc   top5_acc: 100%|██████████| 19/19 [00:00<00:00, 35.70it/s]
                    all      0.923          1

      Epoch    GPU_mem       loss  Instances       Size
      11/35     0.363G    0.04987          5         64: 100%|██████████| 294/294 [00:13<00:00, 21.83it/s]
                classes   top1_acc   top5_acc: 100%|██████████| 19/19 [00:00<00:00, 37.02it/s]
                    all      0.923          1

      Epoch    GPU_mem       loss  Instances       Size
      12/35     0.363G    0.04463          5         64: 100%|██████████| 294/294 [00:13<00:00, 21.59it/s]
                classes   top1_acc   top5_acc: 100%|██████████| 19/19 [00:00<00:00, 36.82it/s]
                    all       0.94          1

      Epoch    GPU_mem       loss  Instances       Size
      13/35     0.363G    0.04684          5         64: 100%|██████████| 294/294 [00:13<00:00, 22.12it/s]
                classes   top1_acc   top5_acc: 100%|██████████| 19/19 [00:00<00:00, 29.22it/s]
                    all      0.947          1

      Epoch    GPU_mem       loss  Instances       Size
      14/35     0.363G    0.04144          5         64: 100%|██████████| 294/294 [00:12<00:00, 23.48it/s]
                classes   top1_acc   top5_acc: 100%|██████████| 19/19 [00:00<00:00, 20.52it/s]
                    all       0.92          1

      Epoch    GPU_mem       loss  Instances       Size
      15/35     0.363G    0.03656          5         64: 100%|██████████| 294/294 [00:12<00:00, 23.37it/s]
                classes   top1_acc   top5_acc: 100%|██████████| 19/19 [00:00<00:00, 20.62it/s]
                    all      0.939          1

      Epoch    GPU_mem       loss  Instances       Size
      16/35     0.363G    0.03804          5         64: 100%|██████████| 294/294 [00:12<00:00, 23.05it/s]
                classes   top1_acc   top5_acc: 100%|██████████| 19/19 [00:00<00:00, 35.31it/s]
                    all      0.944      0.998

      Epoch    GPU_mem       loss  Instances       Size
      17/35     0.363G    0.03226          5         64: 100%|██████████| 294/294 [00:13<00:00, 21.68it/s]
                classes   top1_acc   top5_acc: 100%|██████████| 19/19 [00:00<00:00, 37.12it/s]
                    all      0.944          1

      Epoch    GPU_mem       loss  Instances       Size
      18/35     0.363G     0.0311          5         64: 100%|██████████| 294/294 [00:13<00:00, 21.78it/s]
                classes   top1_acc   top5_acc: 100%|██████████| 19/19 [00:00<00:00, 35.70it/s]
                    all      0.939          1

      Epoch    GPU_mem       loss  Instances       Size
      19/35     0.363G    0.02792          5         64: 100%|██████████| 294/294 [00:13<00:00, 21.98it/s]
                classes   top1_acc   top5_acc: 100%|██████████| 19/19 [00:00<00:00, 36.49it/s]
                    all      0.947          1

      Epoch    GPU_mem       loss  Instances       Size
      20/35     0.363G    0.02527          5         64: 100%|██████████| 294/294 [00:13<00:00, 21.84it/s]
                classes   top1_acc   top5_acc: 100%|██████████| 19/19 [00:00<00:00, 36.38it/s]
                    all      0.949          1

      Epoch    GPU_mem       loss  Instances       Size
      21/35     0.363G    0.02352          5         64: 100%|██████████| 294/294 [00:13<00:00, 21.88it/s]
                classes   top1_acc   top5_acc: 100%|██████████| 19/19 [00:00<00:00, 36.44it/s]
                    all      0.956          1

      Epoch    GPU_mem       loss  Instances       Size
      22/35     0.363G    0.02413          5         64: 100%|██████████| 294/294 [00:13<00:00, 21.71it/s]
                classes   top1_acc   top5_acc: 100%|██████████| 19/19 [00:00<00:00, 37.25it/s]
                    all      0.952          1
```

```
      Epoch    GPU_mem       loss  Instances       Size
      23/35     0.363G    0.02429          5         64: 100%|██████████| 294/294 [00:13<00:00, 22.53it/s]
                classes   top1_acc   top5_acc: 100%|██████████| 19/19 [00:00<00:00, 24.83it/s]
                    all      0.961          1

      Epoch    GPU_mem       loss  Instances       Size
      24/35     0.363G    0.02163          5         64: 100%|██████████| 294/294 [00:12<00:00, 23.59it/s]
                classes   top1_acc   top5_acc: 100%|██████████| 19/19 [00:00<00:00, 23.03it/s]
                    all      0.957          1

      Epoch    GPU_mem       loss  Instances       Size
      25/35     0.363G     0.0205          5         64: 100%|██████████| 294/294 [00:11<00:00, 24.58it/s]
                classes   top1_acc   top5_acc: 100%|██████████| 19/19 [00:00<00:00, 21.05it/s]
                    all      0.954          1

      Epoch    GPU_mem       loss  Instances       Size
      26/35     0.363G    0.01876          5         64: 100%|██████████| 294/294 [00:13<00:00, 22.38it/s]
                classes   top1_acc   top5_acc: 100%|██████████| 19/19 [00:00<00:00, 24.25it/s]
                    all      0.954          1

      Epoch    GPU_mem       loss  Instances       Size
      27/35     0.363G    0.01728          5         64: 100%|██████████| 294/294 [00:13<00:00, 22.58it/s]
                classes   top1_acc   top5_acc: 100%|██████████| 19/19 [00:00<00:00, 36.02it/s]
                    all      0.961          1

      Epoch    GPU_mem       loss  Instances       Size
      28/35     0.363G    0.01947          5         64: 100%|██████████| 294/294 [00:13<00:00, 21.20it/s]
                classes   top1_acc   top5_acc: 100%|██████████| 19/19 [00:00<00:00, 35.90it/s]
                    all      0.954          1
      Epoch    GPU_mem       loss  Instances       Size
      29/35     0.363G    0.01651          5         64: 100%|██████████| 294/294 [00:13<00:00, 21.60it/s]
                classes   top1_acc   top5_acc: 100%|██████████| 19/19 [00:00<00:00, 36.96it/s]
                    all      0.959          1

      Epoch    GPU_mem       loss  Instances       Size
      30/35     0.363G    0.01784          5         64: 100%|██████████| 294/294 [00:13<00:00, 21.62it/s]
                classes   top1_acc   top5_acc: 100%|██████████| 19/19 [00:00<00:00, 33.68it/s]
                    all      0.963          1

      Epoch    GPU_mem       loss  Instances       Size
      31/35     0.363G    0.01503          5         64: 100%|██████████| 294/294 [00:13<00:00, 21.52it/s]
                classes   top1_acc   top5_acc: 100%|██████████| 19/19 [00:00<00:00, 37.58it/s]
                    all      0.961          1

      Epoch    GPU_mem       loss  Instances       Size
      32/35     0.363G    0.01415          5         64: 100%|██████████| 294/294 [00:13<00:00, 21.85it/s]
                classes   top1_acc   top5_acc: 100%|██████████| 19/19 [00:00<00:00, 34.88it/s]
                    all      0.973          1

      Epoch    GPU_mem       loss  Instances       Size
      33/35     0.363G    0.01237          5         64: 100%|██████████| 294/294 [00:13<00:00, 21.56it/s]
                classes   top1_acc   top5_acc: 100%|██████████| 19/19 [00:00<00:00, 36.41it/s]
                    all      0.966          1

      Epoch    GPU_mem       loss  Instances       Size
      34/35     0.363G    0.01272          5         64: 100%|██████████| 294/294 [00:13<00:00, 22.61it/s]
                classes   top1_acc   top5_acc: 100%|██████████| 19/19 [00:00<00:00, 21.23it/s]
                    all      0.964          1
      Epoch    GPU_mem       loss  Instances       Size
      35/35     0.363G    0.01391          5         64: 100%|██████████| 294/294 [00:15<00:00, 19.01it/s]
                classes   top1_acc   top5_acc: 100%|██████████| 19/19 [00:00<00:00, 23.05it/s]
                    all      0.966          1
```

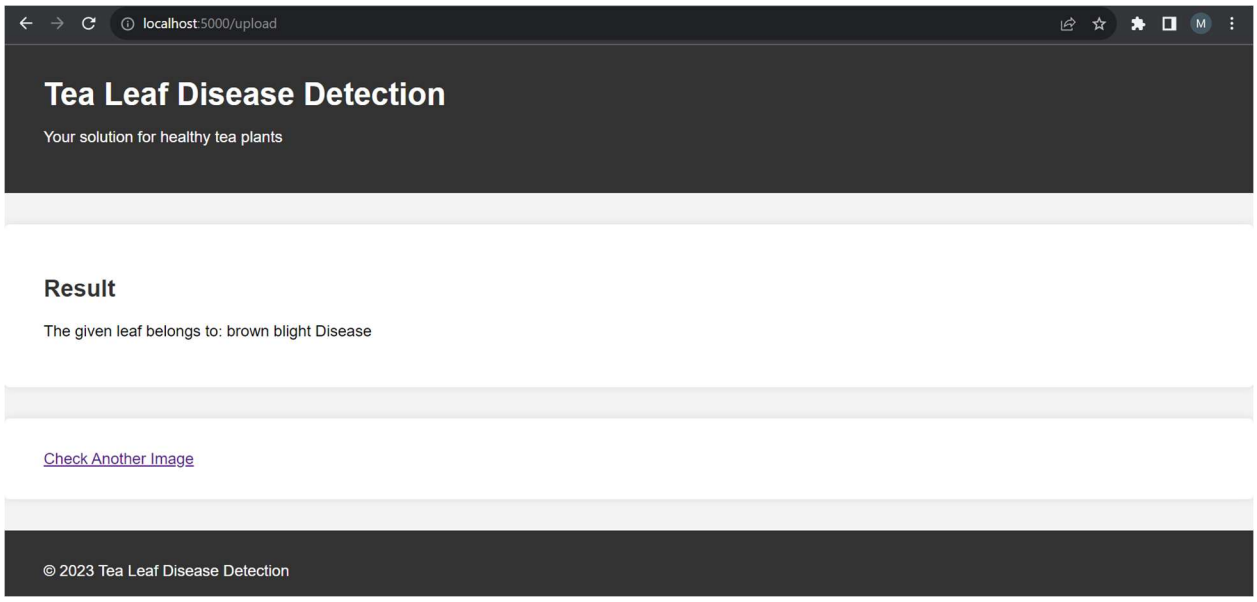
```
35 epochs completed in 0.142 hours.
Optimizer stripped from runs/classify/train/weights/last.pt, 3.0MB
Optimizer stripped from runs/classify/train/weights/best.pt, 3.0MB

Validating runs/classify/train/weights/best.pt...
Ultralytics YOLOv8.0.209 🚀 Python-3.10.12 torch-2.1.0+cu118 CUDA:0 (Tesla T4, 15102MiB)
YOLOv8n-cls summary (fused): 73 layers, 1442566 parameters, 0 gradients, 3.3 GFLOPs
train: /content/dataset/train... found 4693 images in 6 classes ✅
val: /content/dataset/val... found 587 images in 6 classes ✅
test: /content/dataset/test... found 587 images in 6 classes ✅
               classes   top1_acc   top5_acc: 100%|██████████| 19/19 [00:01<00:00, 18.30it/s]
                   all      0.973          1
Speed: 0.1ms preprocess, 1.0ms inference, 0.0ms loss, 0.0ms postprocess per image
Results saved to runs/classify/train
Results saved to runs/classify/train
```

9. RESULTS

9.1 Output Screenshots



When You upload image and click detect.



10. ADVANTAGES & DISADVANTAGES

10.1 Advantages:

a. Real-Time Processing: Efficiency in real-time object detection is a notable strength of YOLOv8. This trait is particularly advantageous in agricultural settings, where swift disease detection is paramount.

b. Object Detection: YOLOv8 excels in object detection tasks, allowing it to identify and pinpoint multiple diseases in a single tea leaf image. This capability proves beneficial in scenarios involving the simultaneous presence of various diseases.

c. Versatility: The versatility of YOLOv8 extends its applicability to a range of object detection tasks, making it adaptable to different crops and diverse agricultural applications beyond detecting diseases in tea leaves.

d. Single Forward Pass: The YOLO architecture's ability to perform detection in a single forward pass is advantageous, striking a balance between speed and accuracy. This stands in contrast to two-stage detectors, providing efficiency in processing.

e. Open-Source and Community Support: Being open-source, YOLOv8 benefits from an active community that offers continuous support, updates, and enhancements.

This collaborative environment facilitates engagement and provides access to a wealth of resources for users.

10.2 Disadvantages:

a. Resource-Intensive: The drawback lies in the demand for substantial computing resources during the training of YOLOv8, necessitating powerful GPUs. This constraint may pose challenges for individuals or organizations with limited computational capabilities.

b. Limited Spatial Resolution: One limitation of YOLO models is their potential struggle in detecting small or closely spaced objects due to constrained spatial resolution. This becomes particularly concerning when tea leaf diseases manifest as minute lesions.

c. Risk of Overfitting: There is a susceptibility to overfitting with YOLOv8, especially in cases where the dataset is small or lacks diversity. Overfitting can lead to a model performing well on training data but poorly on new, unseen data.

d. Difficulty in Handling Class Imbalance: A challenge arises when there is a significant class imbalance in the dataset, potentially causing the model to struggle in effectively detecting diseases in less represented classes.

e. Complexity for Novice Users: Implementing and fine-tuning YOLOv8 may pose complexity, especially for users with limited experience in deep learning. This intricacy could act as a deterrent to adoption, particularly among farmers or practitioners with less technical expertise.

11. CONCLUSION

In summary, employing YOLOv8 for disease detection in tea leaves offers a robust and efficient solution, boasting significant advantages. Its real-time processing capabilities and ability to identify multiple diseases in a single image make it well-suited for dynamic agricultural settings, demonstrating adaptability across various crops.

However, careful consideration is warranted for certain drawbacks, including resource-intensive training requirements and spatial resolution limitations. The potential

for overfitting, difficulties in handling class imbalances, and the complexity for novice users highlight the need for a nuanced approach.

Notwithstanding these challenges, the open-source nature of YOLOv8 and the supportive community contribute to continual enhancements and accessibility. As deep learning evolves, addressing these challenges can lead to improved models, positioning YOLOv8 as a valuable tool for efficient and reliable disease detection in tea leaves and beyond. Success lies in a balanced approach, harnessing the strengths of YOLOv8 while mitigating its limitations to advance sustainable and technologically empowered agriculture.

12. FUTURE SCOPE

The future prospects of utilizing YOLOv8 for disease detection in tea leaves are optimistic, offering potential opportunities for advancements and expansion. Several key aspects define the potential scope for the future:

1. Improved Model Performance: Ongoing research and development endeavors hold the potential to refine the YOLOv8 model, addressing challenges like resource intensity and spatial resolution limitations. This ongoing work aims to cultivate a more efficient and accurate model for detecting diseases in tea leaves.

2. Integration of Advanced Techniques: Incorporating emerging deep learning techniques, such as attention mechanisms and transfer learning, offers avenues for further elevating the model's performance. These advancements are poised to enhance generalization and adaptability across diverse agricultural conditions.

3. Dataset Expansion and Diversity: Future efforts may concentrate on broadening and diversifying the dataset utilized for model training. An enriched dataset, encompassing a variety of tea leaf diseases and environmental conditions, has the potential to bolster the model's capacity for effective disease detection and generalization.

4. Expansion to Other Crops: The adaptable nature of YOLOv8 to various crops implies its potential application beyond tea leaves. Subsequent research could assess its effectiveness in detecting diseases in diverse crops, contributing to a broader and more impactful role in global agriculture.

13. APPENDIX

GitHub Repo Link:

https://github.com/smartinternz02/SI-GuidedProject-613818-1699079718

Project Demo Link:

https://drive.google.com/file/d/1QIK_qf1WifQ0nNmlq84sRbT72wziQING/view?usp=sharing

Source Code:

app.py

```
from flask import Flask, render_template, request, redirect, flash
import os
from ultralytics import YOLO

app = Flask(__name__)
# Set a secret key for flash messages
app.secret_key = 'your_secret_key'

# Load a pretrained YOLOv8n model
model = YOLO('D:/project/YOLOv8_Tea lef dise_Model.pt')

# Set the upload folder
UPLOAD_FOLDER = 'D:/flask_practice/uploads'
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER

def create_upload_folder():
    if not os.path.exists(UPLOAD_FOLDER):
        os.makedirs(UPLOAD_FOLDER)

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

```
@app.route('/upload', methods=['POST'])
def upload_file():
    create_upload_folder()

    if 'file' not in request.files:
        flash('No file part', 'error')
        return redirect(request.url)
    file = request.files['file']
    if file.filename == '':
        flash('No selected file', 'error')
        return redirect(request.url)
    if file:
        filename = os.path.join(app.config['UPLOAD_FOLDER'], file.filename)
        file.save(filename)
        file_path = filename.replace('\\', '/')
        model.predict(file_path, save_txt=True, hide_conf=True, save=True)
        txt_files = os.listdir('runs/classify/predict/labels')
        txt_files.sort(key=lambda x:
os.path.getmtime(os.path.join('runs/classify/predict/labels', x)), reverse=True)

        if txt_files:
            txt_file_path = os.path.join('runs/classify/predict/labels', txt_files[0])
            # Read the content of the first line excluding the first four characters
            with open(txt_file_path, 'r') as txt_file:
                first_line = txt_file.readline()[4:].strip()
            # Adjust the output b ased on the prediction
            if first_line.lower() == 'healthy':
                result = "The tea leaf is healthy!"
            else:
                result = f'The given leaf belongs to: {first_line} Disease'
            return render_template('result.html', result=result)
    flash('No files found in the "labels" directory.', 'error')
    return redirect(request.url)

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

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="{{ url_for('static', filename='css/style.css')
}}">
    <title>Tea Leaf Disease Detection</title>
</head>
<body>
    <div id="background-container">
        <header>
            <div class="container">
                <h1>Tea Leaf Disease Detection</h1>
                <p>Your solution for healthy tea plants</p>
            </div>
        </header>

        <section id="content">
            <div class="container">
                <section id="upload">
                    <h2>Upload Image</h2>
                    <form action="/upload" method="post" enctype="multipart/form-
data">
                        <label for="file-upload" class="file-label">
                            <span>Choose a file</span>
                            <input type="file" id="file-upload" name="file"
accept="image/*" required>
                        </label>
                        <button type="submit">Detect Disease</button>
                    </form>
                </section>
            </div>
        </section>

        <footer>
            <div class="container">
                <p>&copy; 2023 Tea Leaf Disease Detection</p>
            </div>
        </footer>
    </div>
</body>
</html>
```

result.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="{{ url_for('static', filename='css/style.css')
}}">
    <title>Tea Leaf Disease Detection Result</title>
</head>
<body>
    <header>
        <div class="container">
            <h1>Tea Leaf Disease Detection</h1>
            <p>Your solution for healthy tea plants</p>
        </div>
    </header>

    <section id="result">
        <div class="container">
            <h2>Result</h2>
            <div class="result-container">
                <p>{{ result }}</p>
            </div>
        </div>
    </section>

    <section id="upload-again">
        <div class="container">
            <a href="/" class="upload-btn">Check Another Image</a>
        </div>
    </section>

    <footer>
        <div class="container">
            <p>&copy; 2023 Tea Leaf Disease Detection</p>
        </div>
    </footer>
</body>
</html>
```

style.css

```
body {
    font-family: 'Arial', sans-serif;
    margin: 0;
    padding: 0;
    background-color: #f4f4f4;
}

.container {
    max-width: 1200px;
    margin: 0 auto;
    padding: 0 20px;
}

header {
    background-color: #333;
    color: #fff;
    padding: 2em 0;
}

header h1 {
    margin: 0;
}

section {
    background-color: #fff;
    margin: 2em 0;
    padding: 2em 0;
    border-radius: 5px;
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
}

h2 {
    color: #333;
}

form {
    display: flex;
    flex-direction: column;
    align-items: center;
}

.label {
```

```
    display: flex;
    flex-direction: column;
    align-items: center;
    margin-top: 1em;
}

#file-upload {
    display: none;
}

.file-label {
    cursor: pointer;
    background-color: #333;
    color: #fff;
    padding: 0.5em 1em;
    border-radius: 5px;
    transition: background-color 0.3s;
    display: flex;
    align-items: center;
}

.file-label span {
    margin-right: 1em;
}

.file-label:hover {
    background-color: #555;
}

button {
    margin-top: 1em;
    padding: 0.5em 1em;
    font-size: 1em;
    cursor: pointer;
    background-color: #333;
    color: #fff;
    border: none;
    border-radius: 5px;
    transition: background-color 0.3s;
}

button:hover {
    background-color: #555;
}
```

```
footer {
    background-color: #333;
    color: #fff;
    padding: 1em 0;
}
```

YOLOv8 Classifier

```
# Pip install method
!pip install ultralytics
from IPython import display
display.clear_output()
import ultralytics
ultralytics.checks()

import os
import kaggle
import shutil
import ultralytics
from ultralytics import YOLO
from IPython.display import display, Image
from sklearn.model_selection import train_test_split

# Download the dataset
# https://www.kaggle.com/datasets/saikatdatta1994/tea-leaf-disease
files.upload()
os.makedirs('/root/.kaggle', exist_ok=True)
os.rename('kaggle.json', '/root/.kaggle/kaggle.json')
os.chmod('/root/.kaggle/kaggle.json', 600)
username = "saikatdatta1994"
dataset_name = "tea-leaf-disease"
kaggle.api.dataset_download_files(username + '/' + dataset_name,
unzip=True)

# prepare the data
!mkdir '/content/dataset'
DATA_DIR='/content/dataset'
source_dataset_path = '/content/Tea_Leaf_Disease'
destination_dataset_path = '/content/dataset'
os.makedirs(os.path.join(destination_dataset_path, 'train'),
exist_ok=True)
os.makedirs(os.path.join(destination_dataset_path, 'test'), exist_ok=True)
os.makedirs(os.path.join(destination_dataset_path, 'val'), exist_ok=True)
```

```
classes = os.listdir(source_dataset_path)
for class_name in classes:
    class_dir = os.path.join(source_dataset_path, class_name)
    train_images, test_images = train_test_split(os.listdir(class_dir),
test_size=0.2, random_state=42)
    val_images, test_images = train_test_split(test_images, test_size=0.5,
random_state=42)
    os.makedirs(os.path.join(destination_dataset_path, 'train',
class_name), exist_ok=True)
    os.makedirs(os.path.join(destination_dataset_path, 'test',
class_name), exist_ok=True)
    os.makedirs(os.path.join(destination_dataset_path, 'val', class_name),
exist_ok=True)
    for image in train_images:
        shutil.copy(os.path.join(class_dir, image),
os.path.join(destination_dataset_path, 'train', class_name, image))
    for image in test_images:
        shutil.copy(os.path.join(class_dir, image),
os.path.join(destination_dataset_path, 'test', class_name, image))
    for image in val_images:
        shutil.copy(os.path.join(class_dir, image),
os.path.join(destination_dataset_path, 'val', class_name, image))

# Training
model = YOLO('yolov8n-cls.pt')
results = model.train(data='/content/dataset', epochs=35, imgsz=64)

# Download the Model
files.download('/content/runs/classify/train/weights/best.pt')

# Validate the model
model = YOLO('/content/runs/classify/train/weights/best.pt')  # load a
metrics = model.val()
metrics.top1   # top1 accuracy
metrics.top5   # top5 accuracy
```

**** THE END ****