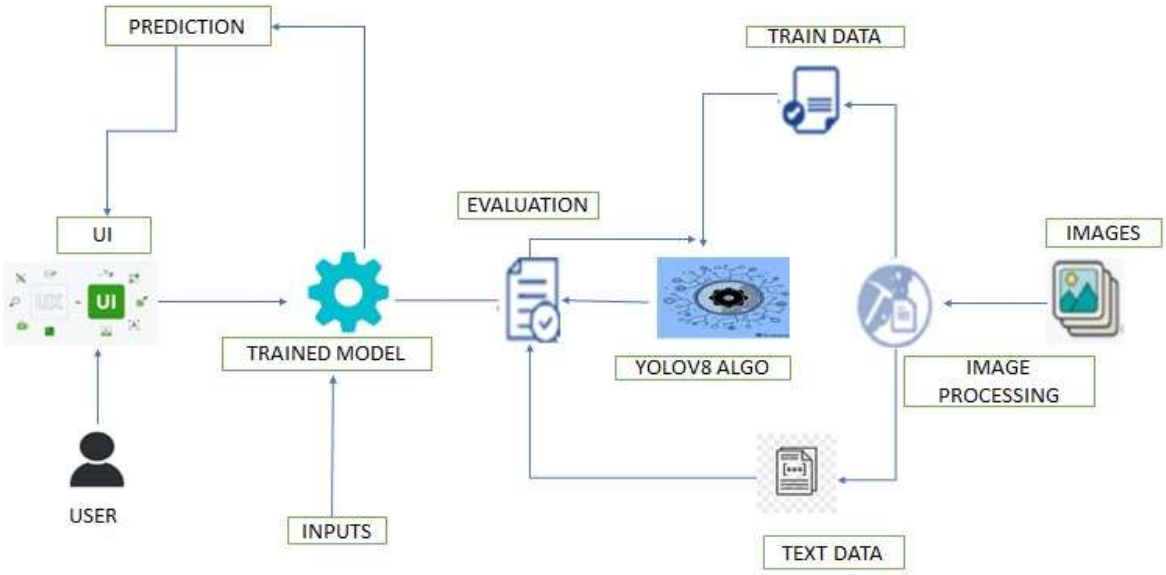
Deep Learning Model For Detecting Diseases In Tea Leaves

Introduction:

Tea, a globally appreciated beverage, holds a significant role in various cultures and economies. Nevertheless, the tea industry confronts notable challenges attributed to diseases affecting the health of tea plants. The timely and precise identification of these diseases is crucial for safeguarding crop yield and preserving the quality of tea production. Within the realm of agricultural technology, the integration of deep learning models has exhibited promising outcomes, presenting a potential solution for automating disease detection processes. This project focuses on creating and deploying a deep learning model designed for the detection of diseases in tea leaves, utilizing the YOLOv8 (You Only Look Once version-8 object detection algorithm. YOLOv8 represents a noteworthy advancement in real-time object detection, delivering improved accuracy and efficiency compared to its predecessors. The training and evaluation dataset for the model is sourced from Kaggle, a well-regarded platform for machine learning datasets. This dataset includes a wide range of images depicting various growth stages of tea plants, with instances of leaves affected by different diseases. Using a comprehensive and diverse dataset is essential for training a resilient model capable of accurately identifying and classifying diseases across various conditions encountered in real-world tea plantations. This endeavor aims to enhance precision agriculture by introducing a tool designed to facilitate the early detection of diseases in tea leaves. The YOLOv8 classifier, selected for its state-of-the-art capabilities, not only guarantees accurate disease identification but also offers real-time processing, making it a valuable asset for both farmers and researchers. The objective of this project is to improve the sustainability and productivity of tea cultivation while furthering the application of deep learning in agriculture.

Technical Architecture:



Prerequisites:

To successfully execute this project, you will need the following software, concepts, and packages.

The experiments in this research were conducted on a Lenovo Ideapad 3 laptop running Windows 11 operating system with the following specifications:

- 16GB Ram
- 512GB SSD
- Intel Core i5 Processor
- Network Speed of 50 Mbps

The following Python(3.9.16) libraries were used in the research:

The model was devoleped on the Colab platform provided by Google, which offers a Jupyter Notebook environment and provides access to high-performance computing resources like GPUs and TPUs. The following Colab configurations were used:

Runtime type. GPU

Runtime shape. High Ram

Anaconda Navigator. is a freely available and open-source distribution that supports Python and R programming languages, specifically tailored for data science and machine learning applications. Anaconda offers a range of useful tools, including Jupyter Notebook, Spyder, Rstudio, and VS Code. In the context of this project, we will utilize Spyder and flask to buld the application.

Necessary Python Libraries

- **OS:** It serves as a versatile interface for interacting with the operating system, offering functionalities for tasks such as file and directory operations
- **Kaggle:** A Python library that allows users to interact with Kaggle platform and perform operations like download data.
- **Shutil:** The "shutil" module in Python streamlines file operations by offering a high-level interface for tasks like copying, removal, and archiving.
- **Scikit-learn:** A comprehensive machine learning library in Python, facilitates various tasks including classification, regression, and clustering.
- **Flask:** A lightweight and flexible web framework for Python, eases the process of building web applications.

Python packages installation: Open colab and execuite the folloing commands
“ !pip install os ”
“ !pip install Kaggle ”
“ !pip install scikit-learn ”
“ !pip install Flask ”

Deep Learning Concepts:

- YOLOv8 Classifier: The YOLOv8 Classifier stands out as an advanced image classification model built upon the YOLO (You Only Look Once) architecture. Engineered for speed, accuracy, and user-friendliness, it emerges as an exceptional option for various image classification tasks.
- Ultralytis: It is a Python package that serves as a robust tool for creating and deploying applications related to object detection, segmentation, classification, tracking, and pose estimation. This package offers an extensive range of functionalities tailored for working with YOLO models.

“ !pip install ultralytics ”
“ from ultralytics import YOLO ”

Project Objectives:

- Implement and train a YOLOv8 deep learning model for accurate and efficient detection of diseases in tea leaves.
- Ensure effective preprocessing of the dataset from Kaggle, including image normalization, resizing, and accurate annotation of tea leaf diseases.
- Assess the accuracy, Top_1 and Top_5 of the trained YOLOv8 model on a validation or test dataset to ensure reliable disease detection.
- Design a user-friendly interface for farmers or end-users, facilitating easy interaction with the model for disease detection.
- Maintain comprehensive documentation detailing the model architecture, hyperparameters, training strategies, and any challenges encountered during the development process.

By achieving these objectives, the project aims to deliver a robust and effective solution for detecting diseases in tea leaves, contributing to the sustainability and quality of tea production.

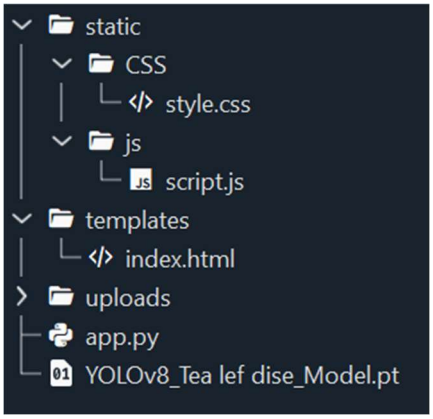
Project Flow:

To accomplish this, we have to complete all the activities and tasks listed below

1. Data Collection
2. Importing necessary libraries
3. Download the Data
4. Prepare the Data
5. Import the Model Building Libraries
6. Load the Model
7. Model Training
8. Model Validation
9. Save the Model
10. Download the trained model from colab
11. HTML file creation
12. CSS file creation
13. JS file creation
14. Python Flask code building
15. Application execuition

Create a Project folder which contains files as shown below

- Flask Application Files: Within the "Project Website" folder, the following files are essential for the Flask application:
- A folder named "templates" that houses the following HTML pages: index.html
- A Python script named "app.py" responsible for server-side scripting.
- Model File: Store the saved model, "YOLOv8_Tea dise_Model.pt" within the "Project Website" folder
- A folder named “uploads” to store the uploaded images.
- Within the "Project Website" folder, the following additional subfolders are required
- A folder named "static" which includes the following subfolders:
    1. A "css" folder containing the "main.css" file.
    2. A "js" folder.



Task-1: 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.

Task-2: Importing necessary libraries

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

Task-3: Download the Data

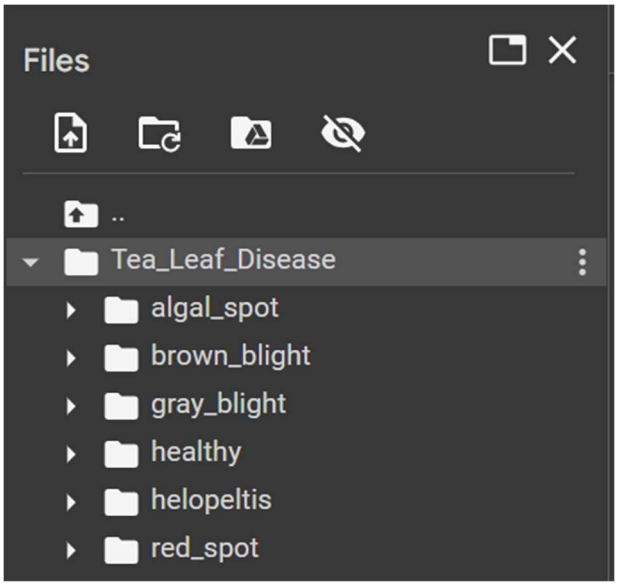
```
[4] from google.colab import files

    # 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:

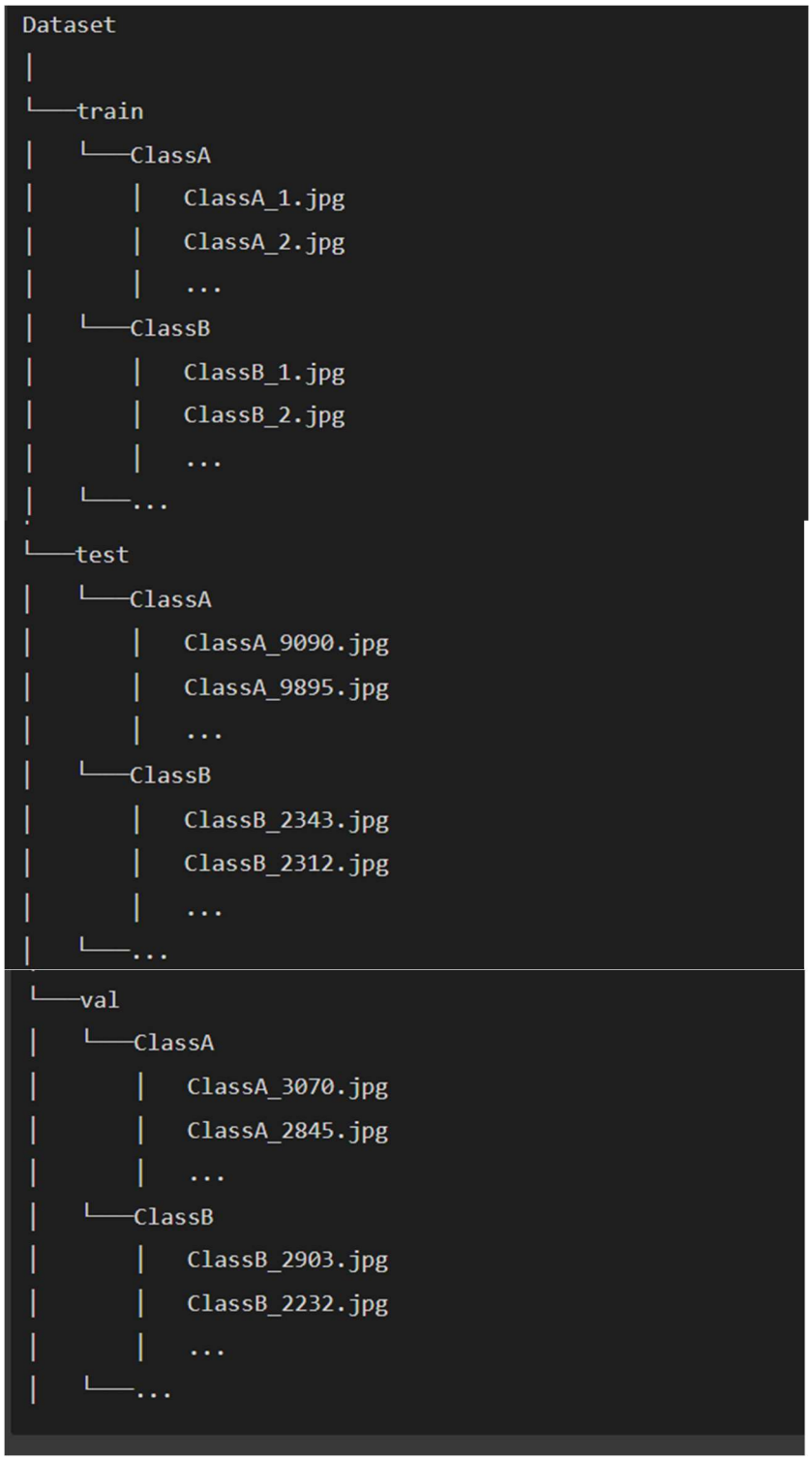


Classes Detected:

```
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.



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

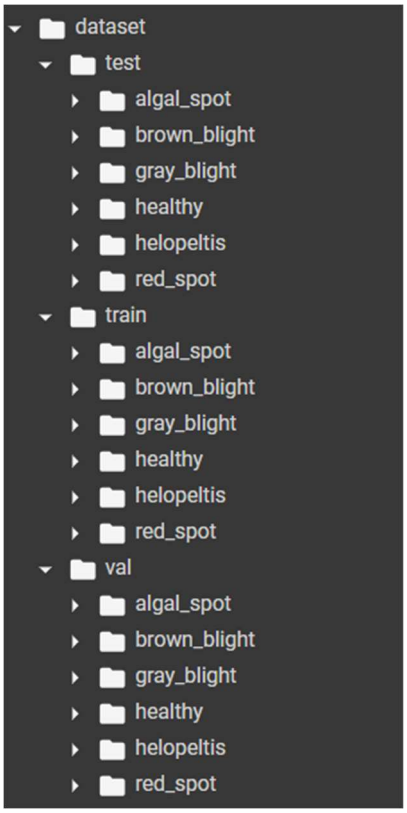
```
[ ]  source_dataset_path = '/content/Tea_Leaf_Disease'
     destination_dataset_path = '/content/dataset'
```

```
[ ]  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')
```

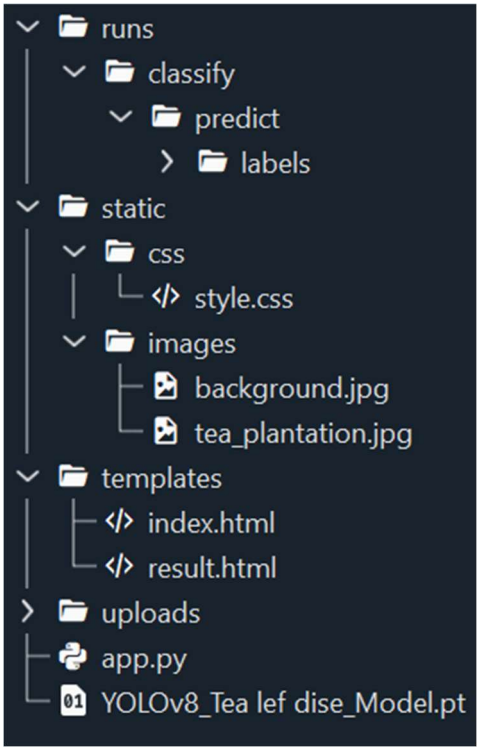
Task-11: Application Building

- 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.

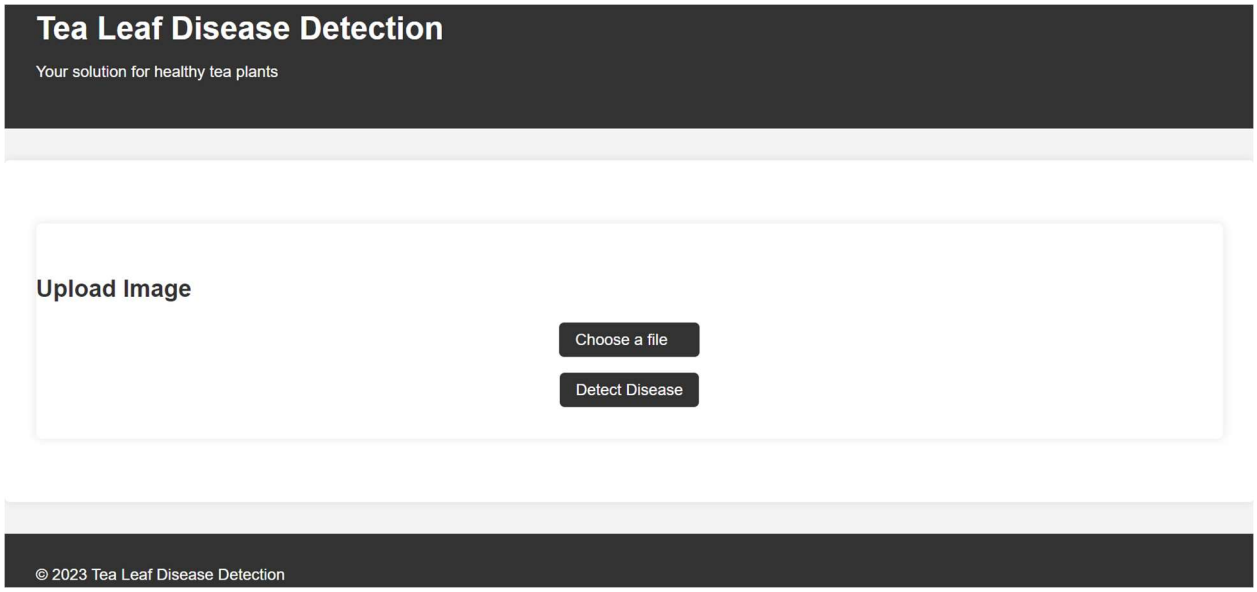
- On the other hand, `result.html` is responsible for receiving input images and presenting the corresponding predictions.
- To enhance the functionality and visual appeal of our web pages, we incorporate a CSS file named `main.css`.
- A ‘runs’ folder is used to store the model predictions, The prediction 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.

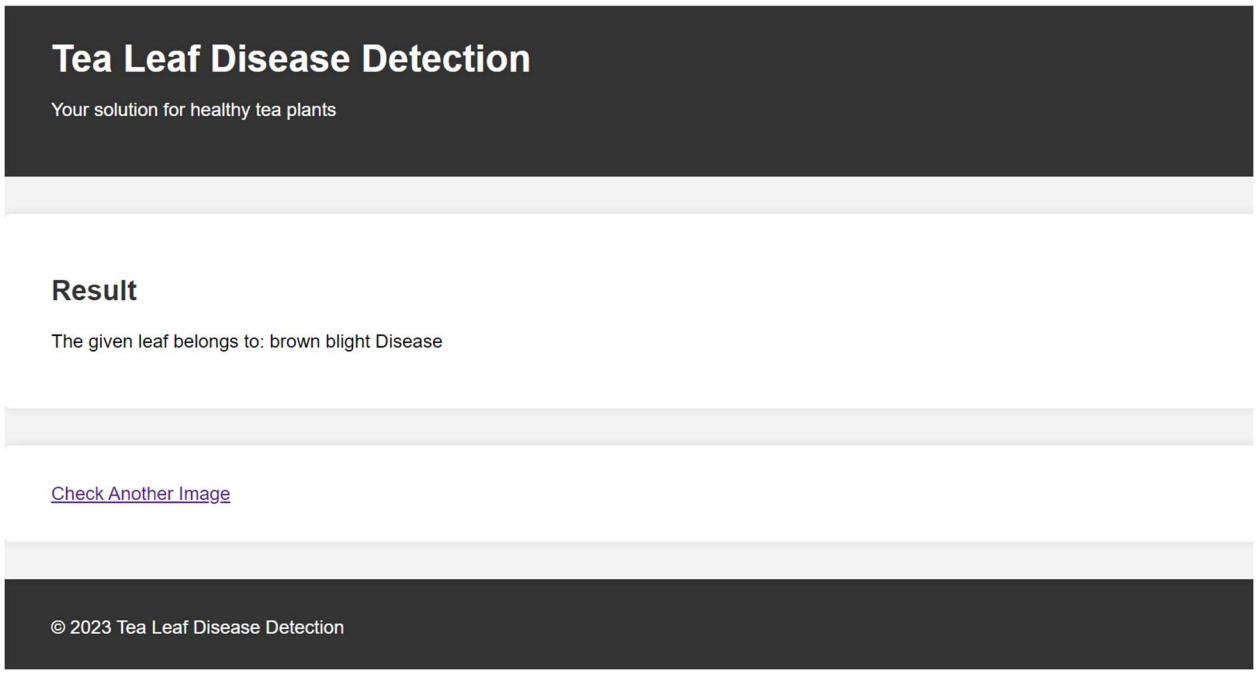
The folder structure to be maintained:



index.html



result.html



Task-12: 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)
```

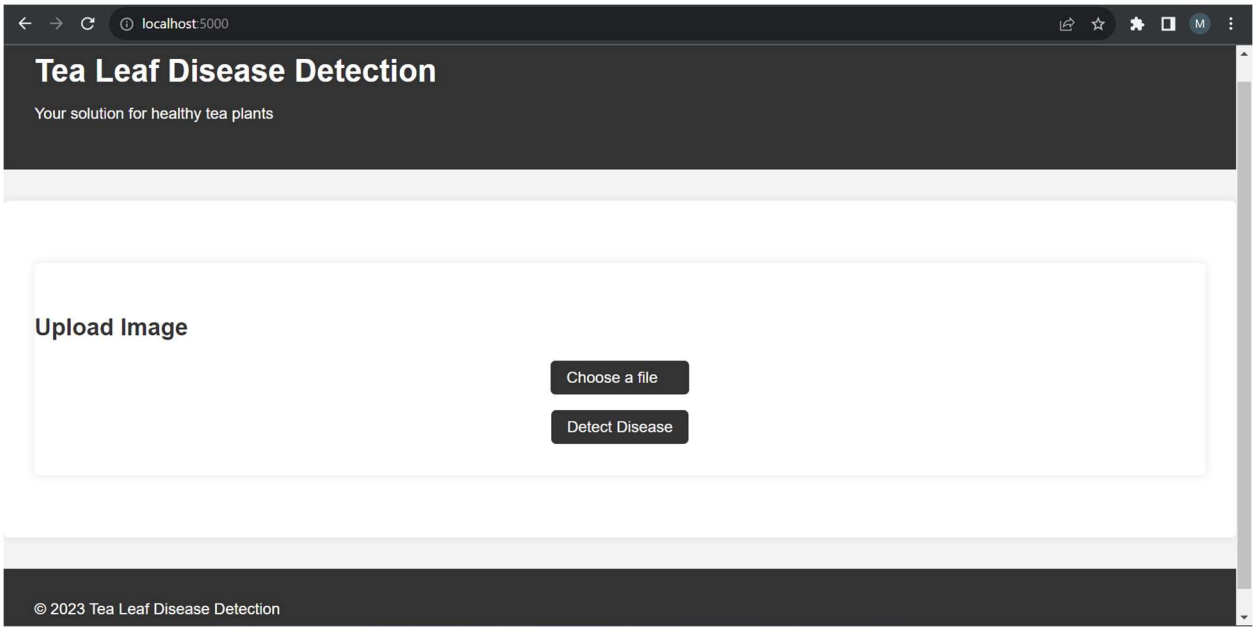
This message will be displayed in the conssole

```
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/’

Final Output:



When You upload image and click detect.