UNIVERSIDADE FEDERAL DE ITAJUBÁ

MACHINE LEARNING PROJECT
PLANT DISEASE CLASSIFICATION

IARA DE CARVALHO NARES - 2024003861
LUCIANO ARAÚJO DOS SANTOS FILHO - 2024006818
PEDRO HENRIQUE CORSINI SOARES RODRIGUES - 2024004107

Repository link: https://github.com/lucianosantos23/Plant_diagnosis_app

Itajubá, October 22, 2025.

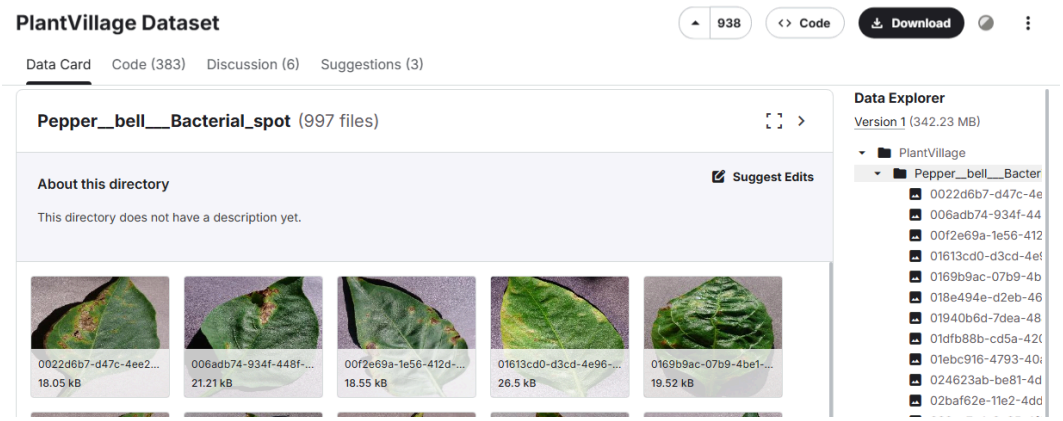
TABLE OF CONTENTS

1. INTRODUCTION

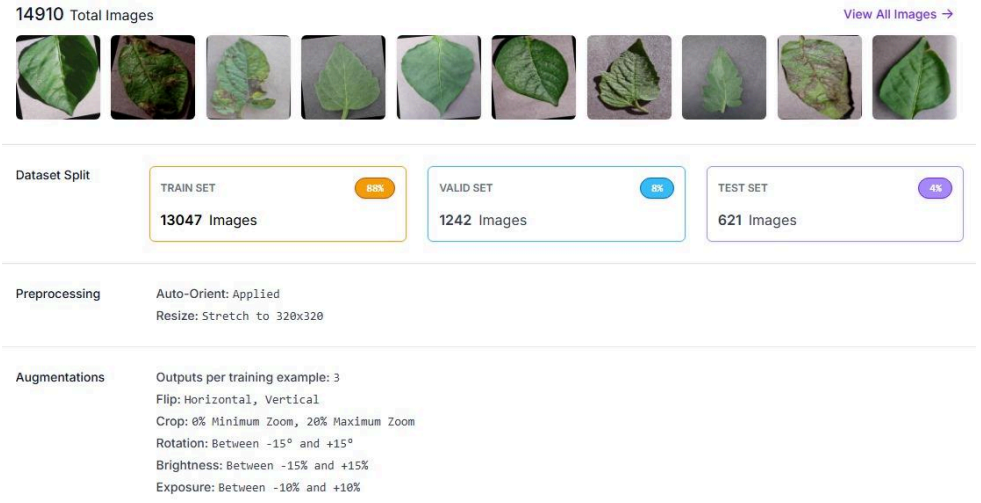
This report details the development of a custom web interface designed for classifying unhealthy plants. The process leverages machine learning, edge systems, and computer vision principles. The application development process included several key stages: goal setting to define the project's objectives and scope, dataset acquisition, labeling to annotate the acquired data for model training, pre-processing to clean and transform the data, model training to develop and optimize the AI model, rigorous testing to evaluate its performance and accuracy, and final deployment to make the application available for use. The goal is to distinguish between healthy and bacteria-degraded plant leaves using custom plant datasets.

2. DATASET ACQUISITION AND LABELING

The custom dataset used on the project was the free to use PlantVillage Dataset [Picture 1], containing a considerative amount of leaf images split between healthy, unhealthy and the kind of leaf. After downloading the dataset, we uploaded to Roboflow to do the labeling process, the pictures were labeled into two classes “Healthy-Leaf” and “Bacteria-Leaf”, after that, they were split into 70% to train, 20% to valid and 10% to test. The preprocesses applied were auto-orientation and resizing the pictures to 320x320. We also added a few augmentations that increased the final number of the dataset images. The augmentations were: flip vertically and horizontally the pics, cropping with a 20% maximum zoom, rotation between -15° and +15°, changing the brightness between -15% and +15% and also the exposure between -10% and +10%. The total of images was 14190.



Picture 1: PlantVillage Dataset.



Picture 2: Roboflow final dataset.

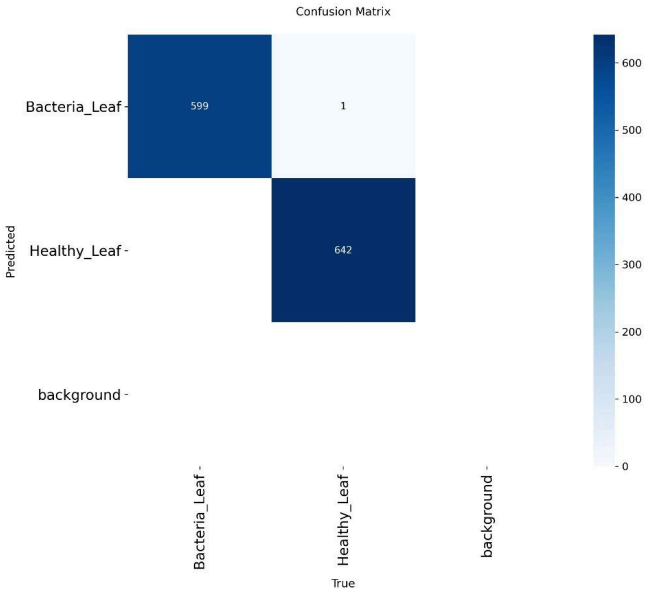
3. TRAINING THE MODEL

To get a more reliable, efficient and light model, we choose to use the Ultralytics YOLOV8n model. The training process was entirely made into a Google Collab Notebook with command lines. The notebook ran with the Nvidia T4 free GPU, ultralytics was installed using the PIP method, and the dataset was extracted from Roboflow. The image size chosen to classify the images was 224x244, and due to the amount of dataset images the epochs chosen were 25, the total time of training was about 34 minutes. The results of training were pretty good, 0.99 of accuracy for the best model.

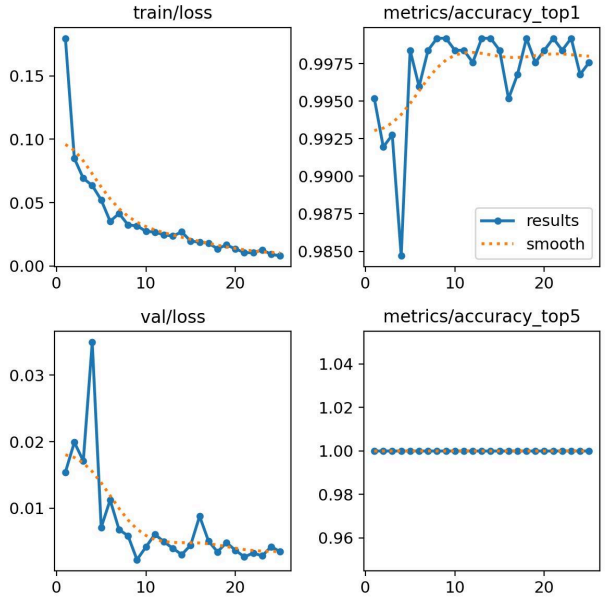
```
Validating /content/runs/classify/train/weights/best.pt...
Ultralytics 8.3.219 🚀 Python-3.12.12 torch-2.8.0+cu126 CUDA:0 (Tesla T4, 15095MiB)
YOLOv8n-cls summary (fused): 30 layers, 1,437,442 parameters, 0 gradients, 3.3 GFLOPs
train: /content/datasets/plants-classification-2/train... found 13047 images in 2 classes ✅
val: /content/datasets/plants-classification-2/valid... found 1242 images in 2 classes ✅
test: /content/datasets/plants-classification-2/test... found 621 images in 2 classes ✅
               classes   top1_acc   top5_acc: 100% ━━━━━━━━━━━━ 39/39 12.4it/s 3.1s
                   all      0.999          1
Speed: 0.1ms preprocess, 0.6ms inference, 0.0ms loss, 0.0ms postprocess per image
Results saved to /content/runs/classify/train
💡 Learn more at https://docs.ultralytics.com/modes/train
```

Picture 3: Final model trained.

The metrics generated by the model were also excellent. The confusion matrix [Picture 4] and the results graphics [Picture 5] exemplifies them.



Picture 4: Confusion Matrix.



Picture 5: Graphic of the results.

Into the /runs/classify/train/weights the model files were saved (“last.pt” and “best.pt”), the chosen one was the “best.pt”. Before downloading it, we validated the model to check if the predictions were coherent with the results shown.

4. TESTING THE MODEL

Before developing the web application we tested the model in the Raspberry Pi Zero 2W to check if the results were going to be great and if the time inferences were going to be reasonable. We ran the model into a Jupyter Notebook and selected some pictures to make the predictions [Picture 6].

Picture 6: Model testing in Jupyter Notebook.

5. WEB APPLICATION DEVELOPMENT

5.1. GENERAL DESCRIPTION

This project is a complete web application developed with the Flask framework, designed for plant health classification through both image uploads and real-time analysis using the Raspberry Pi camera. The system employs a pretrained YOLO deep learning model (plants_healthy_bacteria.pt) capable of identifying whether a plant appears healthy or affected by a bacterial condition. When an image is submitted or captured, the application processes it through the model and returns both the predicted class label and the associated confidence level. The result is then displayed on a clean and user-friendly web interface, providing a smooth integration between artificial intelligence and web-based visualization.

5.2. APP.PY (BACK-END)

The app.py file serves as the core of the application, handling all backend operations and data flow. It is responsible for initializing the Flask environment, managing the upload directory, loading the YOLO model, and configuring the Raspberry Pi camera using the Picamera2 library. Through this script, the system processes user-submitted images or live frames, runs the AI inference, and returns the results to the web interface. It also handles real-time MJPEG video streaming, allowing users to visualize the camera feed directly in the browser.

The application uses multiple threads to ensure that the frame capturing and model inference occur simultaneously without affecting performance or responsiveness. This design allows for continuous frame analysis in the background, while the web interface remains interactive and responsive to user inputs. The main routes defined in the Flask app include the homepage for image uploads, the real-time classification page, the video feed stream, a route that provides the latest classification label in JSON

format for asynchronous updates, and a stop command that halts the live analysis and redirects users back to the initial page.

5.3. INDEX.HTML (FRONT-END – IMAGE UPLOAD)

The index.html file represents the main web interface for uploading images and viewing classification results. Users can select an image from their local device and submit it through a simple and intuitive form. Once processed, the page dynamically displays the predicted class (for example, “Healthy” or “Bacterial Infection”) and the corresponding confidence percentage. This functionality is powered by Flask’s Jinja2 templating engine, which allows the server to pass data directly into the HTML structure.

The design of this page relies on Bootstrap 5, ensuring a responsive layout that adjusts well to different screen sizes. Additionally, the interface provides access to the real-time classification mode through a clearly visible button, creating a seamless transition between static image analysis and live monitoring. Error messages are displayed in a user-friendly way, ensuring that users receive immediate feedback in cases of invalid files or failed uploads.

5.4. REALTIME.HTML (FRONT-END – REAL-TIME CLASSIFICATION)

The realtime.html file is designed specifically for real-time plant health monitoring using the Raspberry Pi camera. It is divided into two main sections: a live video stream showing the camera feed and a classification box that displays the most recent prediction and confidence level. The data in this box is automatically refreshed every half-second through a lightweight JavaScript script that sends asynchronous requests to the server’s /get_label endpoint.

This constant communication between client and server allows the classification status to update continuously without requiring a page reload, creating an efficient and interactive user experience. The layout uses soft colors and rounded elements, evoking a natural aesthetic that matches the theme of plant health and environmental care. The user can stop the real-time monitoring at any moment and return to the main page with a single click, ensuring ease of navigation and control.

5.5. STYLE.CSS (VISUAL STYLE)

The visual design of the project is defined in the style.css file, which establishes the aesthetic identity of the web interface. The background features a smooth green gradient, symbolizing the natural context of the application. Containers and images use rounded corners and soft shadows to convey a sense of modernity and lightness. Buttons are styled with transition effects that enhance interactivity, distinguishing primary actions such as “Upload” from secondary ones like “Back” or “Real-time Mode.” The chosen typography, “Segoe UI,” contributes to a clean and professional appearance, ensuring that the interface remains readable and visually balanced.

Overall, the design reflects the project’s ecological inspiration while maintaining clarity and simplicity, creating a visually pleasant and intuitive user environment.

6. CONCLUSION

In summary, this project successfully merges artificial intelligence and web development into a complete system for plant health analysis. By combining static image processing with real-time video classification, it demonstrates the versatility of computer vision applications in agriculture and environmental monitoring. The modular design, efficient use of threads, and user-friendly interface make the platform both technically sound and accessible. It serves as a practical example of how machine learning models can be integrated into web-based tools to provide immediate, actionable insights in real-world contexts.

GitHub project link: