GreenEye: Intelligent Plant Species and Health Recognition System

Project Objective

GreenEye is a deep learning-based system that can **identify plant species** and **detect plant diseases** from leaf images. Users, such as farmers, gardeners, and botanists, will upload images of plant leaves and receive instant feedback regarding both the plant species and its health status.

The solution targets both **species classification** and **disease detection**, offering personalized care suggestions based on the predictions.

Core Features

- **Image Upload Interface:** Users can either capture a new image or upload an existing leaf photo.
- **Species Prediction:** The system predicts the species name (e.g., "Maple", "Apple tree").
- **Dual Model Prediction:** After prediction, the system displays of two models along-with their confidence score. One model is fine-tuned and another is completely rained from scratch by the GreenEye team.

Dataset Usage Plan

The following datasets will be utilized for model training and evaluation:

Dataset	Purpose	Justification
Pl@ntNet-300K	Plant species classification	Large-scale, real-world images covering over 1,000 species
PlantVillage	Plant species classification	Comprehensive healthy/diseased samples for fine-tuning models
Plant Leaves for Image Classification (Kaggle)	Prototyping	High-resolution images suitable for quick experiments and validation
plant_leaves (TensorFlow Datasets)	Prototyping	Easy integration into TensorFlow pipelines for model validation

Strategy:

Initial prototyping will be conducted using the smaller Kaggle and TensorFlow datasets. Once the pipeline is validated, full-scale training will be performed using Pl@ntNet-300K and PlantVillage datasets.

System Architecture Overview

The GreenEye system will be composed of four major architectural layers: Frontend Interface, Backend API Layer, Machine Learning Model Server, and Database Layer. These layers will communicate over secure APIs to deliver a seamless experience to users.

1. Frontend Interface

Technology: React.js (for web) or React Native (for mobile)

Main Responsibilities:

- Allow users to **upload one image**
- Provide real-time **feedback to the user** (loading indicators, success/failure notifications).
    - One input from a fine-tuned model and another from a new model.
- Display **prediction results** including:
    - Plant species name
    - Confidence score
- Perform **basic client-side validation** (e.g., acceptable image types and sizes).

Communication:

- Sends image files and metadata via HTTP POST requests to the backend API.
- Receives prediction results (species, disease status, suggestions) via JSON responses.

2. Backend API Layer

Technology: FastAPI (preferred for speed and auto-documentation) or Flask

Main Responsibilities:

- **Receive image inputs** from the frontend.
- **Preprocess images** if needed (resize, normalize before sending to ML model).
- **Route images to the correct ML model** (species classifier).

Communication:

- Acts as the bridge between the frontend, the ML Model Server, and the Database.

3. Machine Learning Model Server

Technology: TensorFlow Serving or TorchServe, depending on final model framework

Main Responsibilities:

- Host the trained **species classification model.**
- Accept **preprocessed image data** from the backend.
- Perform **real-time inference** and return:
    - Predicted species (top-k predictions if needed)
    - Confidence scores for each prediction
- Allow **model versioning** to easily deploy improved models without system downtime.

Inference Flow:

1. Receive a POST request with an image tensor.
2. Run inference through the appropriate model.
3. Return prediction results in JSON format.

Optimization Considerations:

- Models will be optimized for faster inference through quantization or pruning.
- GPU-enabled servers may be used for scaling inference under high user loads.

4. Deployment and Infrastructure

Containerization:

- Docker will be used to containerize the frontend, backend API, and model server.
- Separate containers for each component allow for scalability and easier maintenance.

Orchestration and Cloud Hosting:

- Deployed via Kubernetes or lightweight orchestration tools (e.g., Docker Compose during development).
- Hosting platforms: AWS EC2 + S3 for image storage, Google Cloud Platform, or Azure.

Continuous Integration / Continuous Deployment (CI/CD):

- GitHub Actions or GitLab CI/CD pipelines for automated testing, building, and deployment.

Security:

- HTTPS for secure communication between frontend, backend, and model server.
- JWT tokens for user authentication if profiles are implemented.
- Database access rules to enforce data security.

Machine Learning Workflow

1. **Data Preprocessing**
    - Normalize images
    - Data augmentation (rotation, zoom, crop, brightness changes)
    - Handle class imbalance if necessary
2. **Model Development**
    - Transfer learning with pre-trained ImageNet models
    - Fine-tuning on species and disease datasets
3. **Evaluation Metrics**
    - Classification Accuracy
    - Precision, Recall, and F1-Score per class
    - Confusion Matrix Analysis
    - ROC-AUC Curve for disease detection performance
4. **Deployment and Optimization**
    - Quantize models for efficiency
    - Optimize for low-latency inference
    - Set up CI/CD for automatic deployment

Why GreenEye?

GreenEye isn’t just another “plant project” — it’s a true test of our machine learning and software engineering chops. The project dives deep into a real challenge: training a system to recognize a huge variety of plant species using two different models — one pre-trained and fine-tuned, and one built from the ground up by us. This setup doesn’t just make things interesting; it gives us the chance to compare approaches, analyze results, and understand how different training strategies impact prediction accuracy.

We’ve trimmed the scope to focus purely on plant type detection — no plant doctoring this time — but that makes the core experience faster, more focused, and just as impactful. GreenEye is still scalable by design. What begins as a smart “What plant is this?” engine could grow into a fully-fledged assistant for agriculture, botany, or environmental education. Think mobile apps for farmers, tools for curious home gardeners, or even contributions to global biodiversity monitoring — all starting with a single leaf.

Most importantly, GreenEye solves a very real user problem: people often want to know what plant they’re dealing with, quickly and reliably. Lab testing? Too expensive. Expert botanists? Not always around. With GreenEye, a simple photo upload gives you a quick answer — and now, two model opinions to compare. It's fast, accessible, and empowering for farmers, hobbyists, and plant parents alike.

And let's not forget the team angle. This project spreads the work across every role: requirements, data prep, model experimentation, system architecture, frontend design, backend APIs, testing, and deployment. Each member brings their own expertise and energy to the table — and no one has to carry the jungle alone.