### *[Group 51 - Topic 1] Advanced Training Program - AI125*

CropCare AI: Plant Leaf Disease Detection

## 1. Business Understanding

### 1.1 Problem Definition

Plant diseases significantly impact global agriculture, leading to reduced crop yield, compromised quality, and economic losses, especially for smallholder farmers with limited access to expert diagnosis. Manual inspection is often slow, subjective, and inaccessible in rural or under-resourced areas. This project aims to develop an **AI-powered image classification system** that can **automatically detect plant diseases from leaf images**, enabling faster, more accurate diagnoses.

### 1.2 Project Goals

The core objectives of this project are to:

* **Reduce crop loss** through early and accurate disease detection.
* **Support farmers** by providing accessible, real-time diagnostic tools.
* **Improve crop yield and quality**, contributing to food security and sustainable farming.

These goals align with global efforts to modernize agriculture using smart technologies.

### 1.3 Pain Points

Key challenges that motivate this project include:

* **Late detection** of diseases, often due to subtle early symptoms.
* **Lack of expert access**, especially in rural areas.
* **Limitations of manual diagnosis** include subjectivity, inconsistency, and inefficiency.

### 1.4 Success Criteria

The project will be considered successful if it delivers:

* **High model accuracy** (precision, recall, F1-score)
* **Early-stage detection** capability
* **Practical usability** across varying lighting and image conditions
* **Mobile deployment readiness** for real-world use by farmers

## 

## 2. Data Understanding

### 2.1 Explore the Dataset(s)

This project uses publicly available datasets, particularly **PlantVillage**, which provides a large collection of labeled plant leaf images. The dataset includes:

* **Tens of thousands** of images covering various crop species
* **Multiple disease categories**, often specific to each crop
* **Healthy and diseased examples**, enabling both binary and multiclass classification
* **Metadata** such as crop type, disease name, and health status

The images come from both **controlled lab conditions** and **field environments**, offering diverse lighting, backgrounds, and leaf appearances.

### 2.2 Identify Key Data Points

The most relevant attributes include:

* **Crop Type**: Disease symptoms differ by plant species.
* **Disease Type**: Labeled classes help the model learn unique disease patterns.
* **Health Status**: A binary feature (healthy/diseased) essential for basic classification.
* **Image Quality**: Sharp, well-lit images improve feature learning; low-quality inputs hinder performance.

Understanding these variables is essential for preprocessing and model training.

### 2.3 Look for Patterns

Data analysis reveals important trends:

* **Crop–Disease Relationships**: Some diseases, like late blight, are crop-specific.
* **Symptom Diversity**: Visual signs vary by disease and infection stage.
* **Class Imbalance**: Uneven sample distribution can bias the model; balancing techniques are needed.

These insights help guide augmentation and model tuning strategies.

### 2.4 Note External Factors

Disease visibility is influenced by:

* **Seasonality**: Some pathogens are climate-dependent.
* **Weather Conditions**: Rain, glare, or shadow can obscure visual symptoms.
* **Growth Stage**: Early symptoms are subtle, and late-stage symptoms are more visible.

Robust models must generalize across these environmental variations, or optionally use contextual metadata to improve predictions.

## 3. Data Preparation

### 3.1 Clean the Data: Remove Poor-Quality Images, Incorrect Labels, and Duplicates

A crucial first step in preparing data for machine learning is ensuring its **integrity and consistency**. In this project, the image dataset underwent a detailed cleaning process to eliminate noise and potential sources of error:

* **Poor-quality images** those that are blurry, poorly lit, overexposed, underexposed, or partially obscured, were identified and removed. These images can distort the training process and reduce the model’s ability to extract meaningful features.
* **Incorrect labels** often resulting from human annotation errors or inconsistencies in public datasets, were manually reviewed and corrected or excluded. Maintaining accurate labels is essential for supervised learning tasks.
* **Duplicate images** were detected and removed to avoid model overfitting and artificial inflation of dataset size.

The dataset used in this project (**PlantVillage**) is already pre-cleaned and well-organized. It contains high-quality, labeled images across a wide range of plant species and diseases. Therefore, no additional cleaning was necessary. Tasks such as removing duplicates, correcting labels, or filtering low-quality images were not required during this phase.

### 3.2 Resize and Normalize Images to Standard Formats (e.g., 224×224, Pixel Scaling)

To ensure compatibility with **convolutional neural network (CNN)** architectures, all images were resized to a standard resolution, typically **224×224 pixels**. This size maintains sufficient visual detail while enabling efficient computation and alignment with pre-trained models like ResNet, VGG, or MobileNet.

Following resizing, pixel values were **normalized**—either by scaling intensities to a [0, 1] range or by standardizing using the dataset’s mean and standard deviation. Normalization ensures that input features have consistent distributions, which:

* Accelerates training convergence
* Stabilizes gradient updates
* Enhances overall model performance

Together, resizing and normalization transform raw image data into a **uniform, well-conditioned input set**, ready for robust deep learning workflows.

### 3.3 Apply Data Augmentation: Rotation, Flipping, and Zooming for Generalization

To increase the model's robustness and adaptability to real-world scenarios, **data augmentation** techniques were applied to artificially expand the training dataset. This step is critical for avoiding overfitting, especially when working with imbalanced or limited data.

The augmentation strategies included:

* **Random rotations**, simulating different leaf orientations commonly seen in the field
* **Horizontal and vertical flips**, mimicking the natural variability of plant positioning

These transformations help the model focus on **invariant features** such as disease-specific patterns rather than surface-level details like angle or position. As a result, the trained model is better equipped to perform accurately across a wide variety of field conditions, lighting environments, and image qualities.

## 4. Modeling

### 4.1 Choose Model Types

We selected **ResNet-18** as the primary model due to its strong performance and efficient architecture. Its **residual connections** address vanishing gradients, and its smaller size makes it ideal for deployment on **resource-constrained devices** like smartphones or embedded systems used in agriculture.

We used **pretrained weights from ImageNet**, replacing the final classification layer to match our disease categories. Lightweight models like **MobileNet** and **EfficientNet** were considered for future work due to their speed and size, but were not implemented in this phase.

### 4.2 Transfer Learning Using Pretrained Models

To improve training speed and accuracy, we applied **transfer learning** with pretrained **ResNet-18** weights:

* **All convolutional layers** were retained to preserve useful feature extraction.
* The **final fully connected layer** was replaced with a custom layer for our classification task.
* The **entire network was fine-tuned**, enabling adaptation to domain-specific leaf patterns and textures.

This approach reduced training time, improved convergence, and helped avoid overfitting.

### 4.3 Ensemble Models

No ensemble methods were applied in this phase. We focused solely on a single ResNet-18 model. However, ensemble learning is a promising direction for future improvements to boost robustness and reduce prediction variance.

### 4.4 Train the Model with Augmented Data

We trained the model in **PyTorch** using the following setup:

* **Epochs**: 15
* **Optimizer**: Adam
* **Learning Rate**: 0.001
* **Batch Size**: 64
* **Input Size**: 224×224
* **Data Split**: 80% training, 20% testing
  + From the training set, 20% was reserved for validation

**Data augmentation** (rotation, flipping, zooming) was applied to improve generalization. Throughout training:

* Accuracy and loss were tracked on training and validation sets
* The **best-performing model** was saved based on validation accuracy
* Final weights were stored for deployment

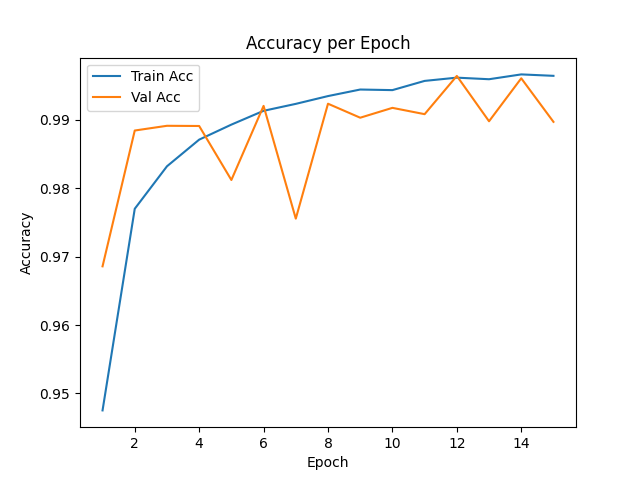
This setup ensured efficient training and strong generalization with relatively low resource demands.

Below is a detailed summary of the training process:

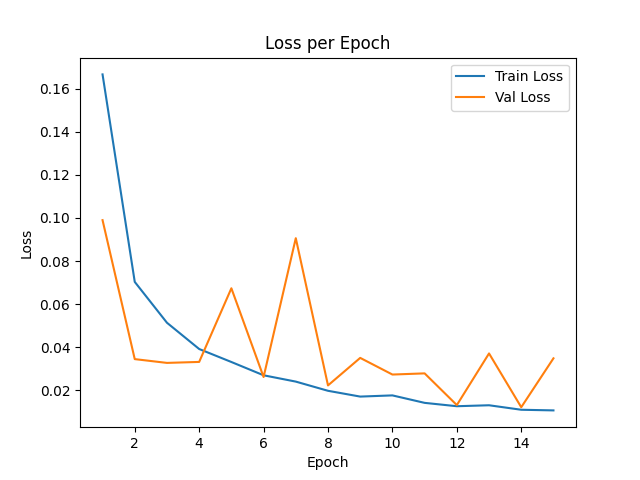
*Table 1: ResNet18 - Training and Validation Metrics*

| **Epoch** | **Train Loss** | **Train Acc** | **Val Loss** | **Val Acc** |
| --- | --- | --- | --- | --- |
| 1 | 0.16660 | 0.94753 | 0.09899 | 0.96857 |
| 2 | 0.07035 | 0.97697 | 0.03452 | 0.98840 |
| 3 | 0.05140 | 0.98318 | 0.03277 | 0.98909 |
| 4 | 0.03922 | 0.98707 | 0.03322 | 0.98906 |
| 5 | 0.03319 | 0.98926 | 0.06739 | 0.98118 |
| 6 | 0.02702 | 0.99128 | 0.02622 | 0.99200 |
| 7 | 0.02410 | 0.99229 | 0.09062 | 0.97553 |
| 8 | 0.01983 | 0.99343 | 0.02233 | 0.99231 |
| 9 | 0.01715 | 0.99439 | 0.03509 | 0.99027 |
| 10 | 0.01770 | 0.99430 | 0.02737 | 0.99171 |
| 11 | 0.01423 | 0.99565 | 0.02791 | 0.99079 |
| 12 | 0.01267 | 0.99612 | 0.01320 | 0.99637 |
| 13 | 0.01312 | 0.99589 | 0.03716 | 0.98975 |
| 14 | 0.01103 | 0.99660 | 0.01221 | 0.99603 |
| 15 | 0.01074 | 0.99639 | 0.03489 | 0.98967 |

To better visualize the numerical results, the training and validation performance are illustrated using the following plots:



*Figure 1: ResNet18 - Training and Validation Accuracy per Epoch*



*Figure 2: ResNet18 - Training and Validation Loss per Epoch*

### 4.5. Tune Hyperparameters

### After reviewing training and validation metrics, we identified signs of mild overfitting. While training accuracy exceeded 99.6% by epoch 15, validation accuracy plateaued and showed fluctuations at specific points (e.g., epochs 5, 7, and 13). Additionally, the validation loss spiked intermittently, even as training loss steadily declined.

### These trends suggest the model was starting to memorize training data, rather than learning features that generalize well to unseen samples.

### To address this, we introduced a series of targeted hyperparameter and architecture adjustments to improve generalization and stability:

#### 4.5.1 Reduce Batch Size

### Change: Batch size was reduced from 64 to 32. Reason: A smaller batch size introduces stochasticity in gradient updates, which acts as a form of regularization. This helps the model avoid sharp minima and generalizes better to new data.

#### 4.5.2 Improve the Final Fully Connected Layer

**Change**: Replaced the single-layer classification head with a **multi-layer structure**, consisting of:

Linear → BatchNorm → ReLU → Dropout → Final Linear layer.

**Reason**: This deeper head increases model capacity while integrating **Batch Normalization** for stable training and **Dropout** for regularization. These layers reduce overfitting and help the model learn more robust features.

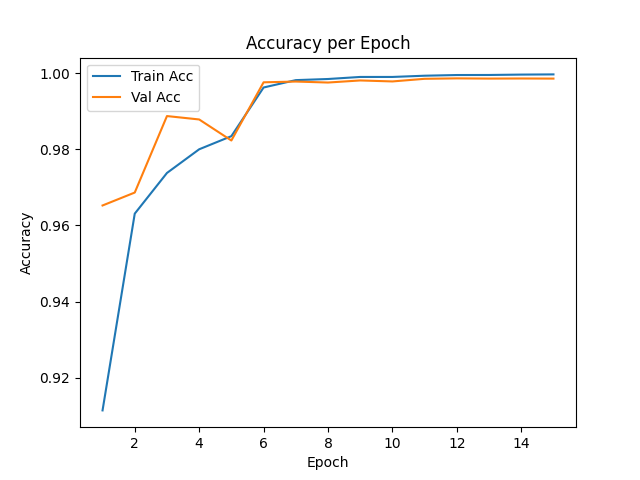
#### 

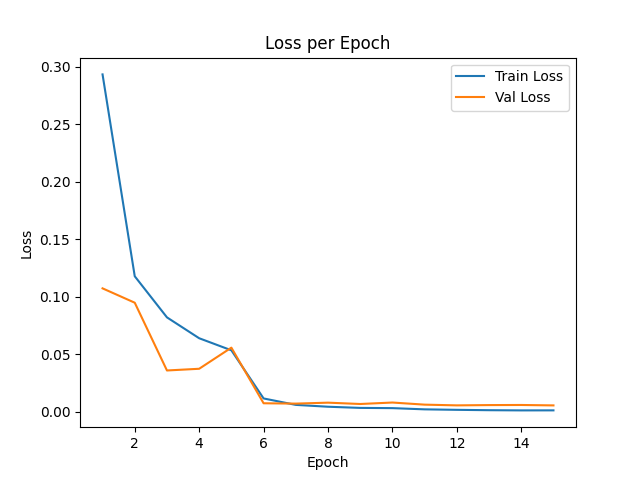
#### 4.5.3 Add Learning Rate Scheduler

**Change**: Introduced a scheduler to **reduce the learning rate by a factor of 10 every 5 epochs**.  
**Reason**: Gradually reducing the learning rate allows the model to take large optimization steps early in training and finer adjustments later on. This strategy prevents the model from overshooting optimal minima and encourages smoother convergence. Especially in the later epochs where overfitting typically starts to emerge, a smaller learning rate helps in fine-tuning the model more precisely without causing significant over-adaptation to the training data.

*Table 2: Training and Validation Results After Tuning Hyperparameters and Model*

| **Epoch** | **Train Loss** | **Train Accuracy** | **Val Loss** | **Val Accuracy** |
| --- | --- | --- | --- | --- |
| 1 | 0.29320 | 0.91140 | 0.10726 | 0.96523 |
| 2 | 0.11777 | 0.96309 | 0.09479 | 0.96863 |
| 3 | 0.08205 | 0.97377 | 0.03590 | 0.98872 |
| 4 | 0.06391 | 0.97999 | 0.03738 | 0.98782 |
| 5 | 0.05343 | 0.98344 | 0.05575 | 0.98230 |
| 6 | 0.01160 | 0.99622 | 0.00741 | 0.99758 |
| 7 | 0.00600 | 0.99816 | 0.00711 | 0.99778 |
| 8 | 0.00439 | 0.99845 | 0.00793 | 0.99752 |
| 9 | 0.00338 | 0.99899 | 0.00677 | 0.99807 |
| 10 | 0.00318 | 0.99899 | 0.00804 | 0.99781 |
| 11 | 0.00212 | 0.99932 | 0.00621 | 0.99850 |
| 12 | 0.00170 | 0.99950 | 0.00558 | 0.99862 |
| 13 | 0.00138 | 0.99951 | 0.00583 | 0.99856 |
| 14 | 0.00121 | 0.99961 | 0.00592 | 0.99859 |
| 15 | 0.00124 | 0.99966 | 0.00555 | 0.99856 |



*Figure 3: Training and Validation Accuracy After Tuning Hyperparameters and Model*  


*Figure 4: Training and Validation Loss After Tuning Hyperparameters and Model*

The refinements made to the model architecture and training configuration have contributed to improved overall performance. Compared to the previous results, the updated training exhibits:

* A significant reduction in validation loss, reaching as low as 0.00555 in the final epoch, indicates a much better generalization capability.
* Higher and more stable validation accuracy, consistently above 99.7% from epoch 6 onwards, peaking at 99.86%.
* Minimal gap between training and validation accuracy, suggesting that the model no longer suffers from the mild overfitting observed previously.
* Early and smooth convergence, with the model stabilizing around optimal performance after just 6 epochs.

In summary, these enhancements have effectively addressed the earlier signs of overfitting, leading to a more robust and well-generalized model that achieves excellent accuracy while maintaining stability across epochs.

## 5. Evaluation

### 5.1 Use Standard Metrics: Accuracy, F1-Score, and Confusion Matrix

To evaluate the performance of the plant disease classification model, we employed standard classification metrics—**Accuracy**, **Precision**, **Recall**, **F1-Score**, and the **Confusion Matrix**—to provide both global and class-specific insights.

* **Accuracy** measures the proportion of correctly predicted samples among all samples, offering a high-level summary of the model's performance. For our 38-class dataset, a high accuracy indicates the model effectively distinguishes between various plant diseases.
* The **Confusion Matrix** provides a detailed breakdown of predictions. Each row represents the actual class, and each column shows the predicted class. Diagonal entries indicate correct predictions, while off-diagonal elements highlight misclassifications.

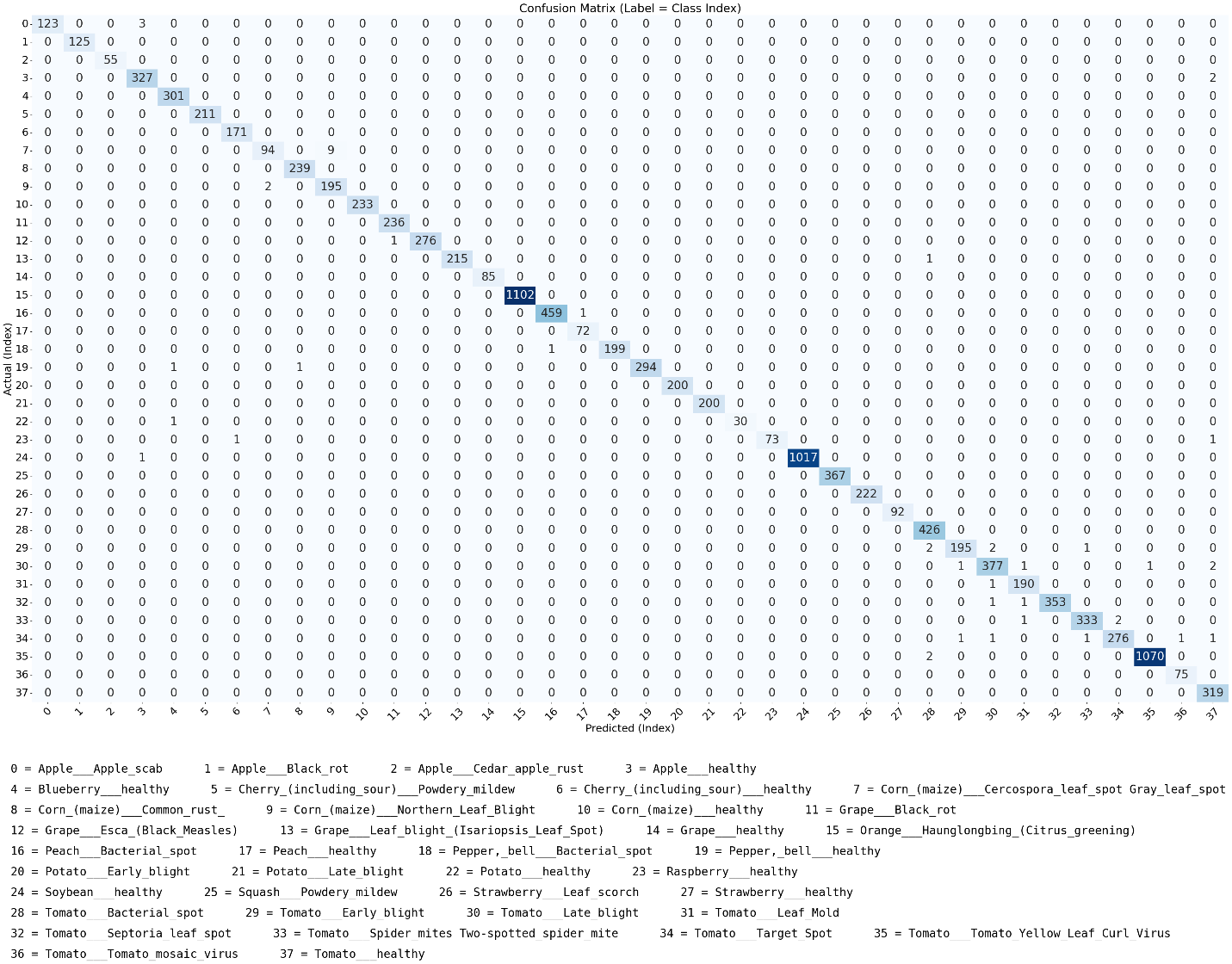
By analyzing the confusion matrix, we can:

* Identify which diseases are frequently misclassified.
* Detect potential class imbalance in predictions.
* Prioritize refinements for specific class pairs with high confusion.

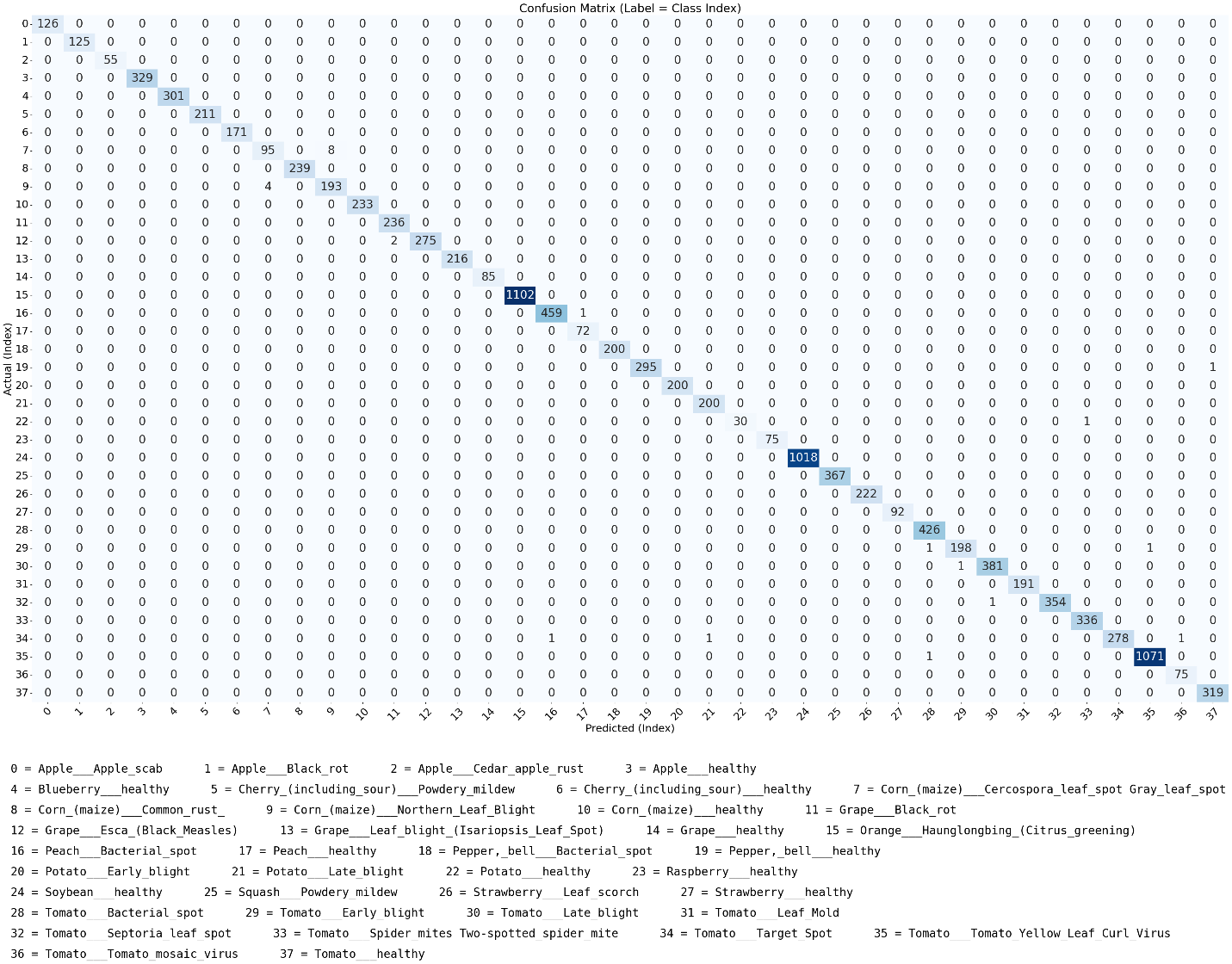
These metrics complement each other: **accuracy** offers a summary of overall model effectiveness, while the **confusion matrix** gives granular feedback on individual class performance.

As reported in the **Modeling** section, the final **validation accuracy reached 99.85%**, reflecting strong overall classification capability.

The confusion matrix (shown below) demonstrates the model’s performance across all 38 disease classes by comparing predicted vs. true labels.



*Figure 5: Confusion Matrix*



*Figure 6: Confusion Matrix After Tuning Hyperparameters and Model*

### 5.2. Analyze Misclassifications and Confusion Between Diseases

A deeper examination of the confusion matrices from both training phases reveals how the model evolved in terms of class-wise accuracy and robustness.

#### Before Hyperparameter and Model Tuning:

In the initial phase, the model showed strong convergence with a peak validation accuracy of approximately **99.64%**. The confusion matrix indicated that the model could generally differentiate between the 38 plant disease classes effectively.

However, signs of **mild overfitting** were evident, seen in the widening gap between training and validation accuracy in later epochs. While misclassifications were relatively few and scattered, some class pairs showed **isolated confusion**, suggesting room for improvement.

#### After Hyperparameter and Model Tuning:

Following modifications—such as reducing the batch size, expanding the final fully connected layer, and implementing a learning rate scheduler—the updated confusion matrix showed noticeable improvements:

* A **reduction in total misclassifications** across all classes.
* Formerly confused class pairs were now **more accurately separated**.
* The matrix displayed a **stronger diagonal alignment**, indicating consistent and confident predictions across the validation set.

These improvements demonstrate the impact of tuning on both generalization and class-specific performance. Enhancements to the classifier head likely helped the model learn more **discriminative features**, while the learning rate schedule allowed more **stable convergence** and finer weight adjustments.

Ultimately, the reduction in inter-class confusion improves **diagnostic reliability**—an essential factor for deploying the model in real-world agricultural settings, where precise identification directly affects crop treatment strategies.

### 5.3. Compare Different Models and Configurations

To evaluate the effectiveness of different model architectures, we compared **ResNet-18** with **MobileNetV2**, a lightweight CNN designed for mobile and embedded applications. Both models were trained on the same dataset using identical hyperparameters to ensure a fair comparison:

* **Optimizer**: Adam
* **Learning Rate**: 0.001
* **Batch Size**: 32
* **Epochs**: 15
* **Input Size**: 224×224

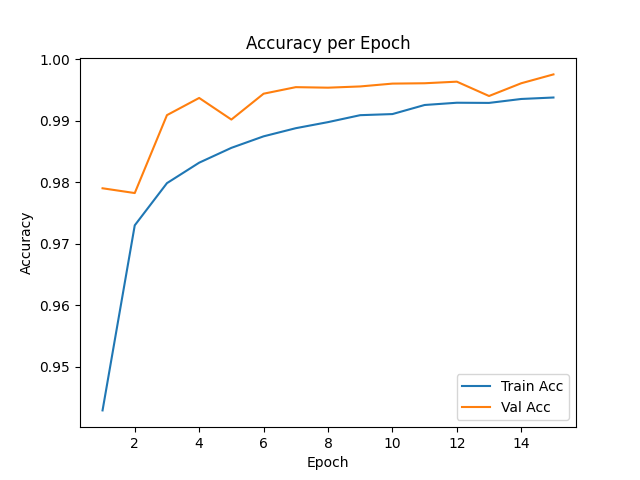
#### Training Observations:

* **ResNet-18** achieved slightly higher final validation accuracy (**99.86%**) compared to **MobileNetV2** (**99.75%**).
* ResNet-18 showed more **stable convergence**, with lower validation loss and minimal fluctuations in the later epochs.
* MobileNetV2 performed **competitively**, converging well and demonstrating strong generalization, especially given its **smaller model size and lower computational cost**.

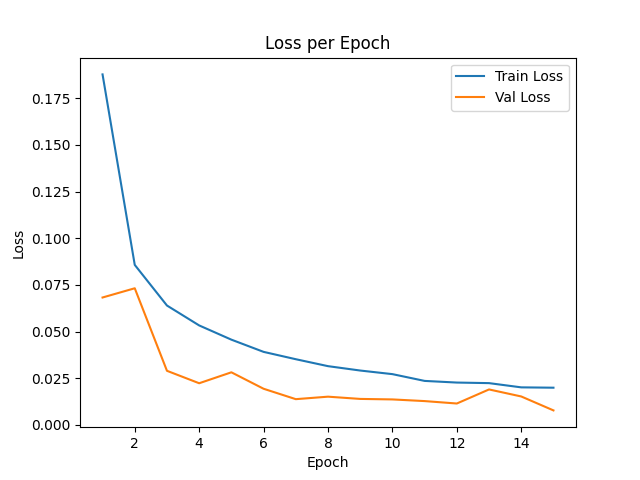
The training and validation results for MobileNetV2 are summarized below:

*Table 3: MobileNet - Training and Validation Metrics*

| **Epoch** | **Train Loss** | **Train Accuracy** | **Val Loss** | **Val Accuracy** |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
| 1 | 0.1878 | 94.29% | 0.0682 | 97.90% |
| 2 | 0.0857 | 97.30% | 0.0732 | 97.82% |
| 3 | 0.0639 | 97.99% | 0.0289 | 99.09% |
| 4 | 0.0533 | 98.32% | 0.0223 | 99.37% |
| 5 | 0.0456 | 98.56% | 0.0281 | 99.02% |
| 6 | 0.0391 | 98.75% | 0.0193 | 99.44% |
| 7 | 0.0351 | 98.88% | 0.0137 | 99.55% |
| 8 | 0.0314 | 98.98% | 0.0151 | 99.54% |
| 9 | 0.0291 | 99.09% | 0.0139 | 99.56% |
| 10 | 0.0271 | 99.11% | 0.0136 | 99.60% |
| 11 | 0.0235 | 99.26% | 0.0127 | 99.61% |
| 12 | 0.0226 | 99.29% | 0.0114 | 99.63% |
| 13 | 0.0223 | 99.29% | 0.0189 | 99.40% |
| 14 | 0.0201 | 99.35% | 0.0152 | 99.61% |
| 15 | 0.0199 | 99.38% | 0.0077 | 99.75% |



*Figure 7: MobileNet - Training and Validation Accuracy per Epoch*



*Figure 8: MobileNet - Training and Validation Loss per Epoch*

Both **ResNet-18** and **MobileNetV2** delivered excellent performance on the plant disease classification task, achieving very high training and validation accuracy. However, ResNet-18 consistently outperformed MobileNetV2 across several key areas.

ResNet-18 achieved a slightly higher final validation accuracy of **99.86%**, compared to **99.75%** for MobileNetV2, and showed more stable convergence with minimal fluctuations in validation loss during the final epochs. It also reached a **lower overall validation loss**, indicating greater confidence in its predictions.

Additionally, ResNet-18 achieved **lower loss values earlier in training**, suggesting superior feature extraction and generalization capabilities from the outset.

That said, **MobileNetV2 remains a strong and practical alternative**, especially under resource constraints. With a significantly smaller model size and reduced computational demand, it offers competitive accuracy while being highly suitable for **real-time deployment on edge devices or mobile platforms**.

## 6. Deployment

### 6.1 Visualize Detection Results with Image Previews and Labels

To validate offline performance, predictions were visualized using a custom script (predict\_resnet18.py). This script processes an input image through the trained ResNet-18 model and outputs:

{"prediction": "Strawberry\_\_\_Leaf\_scorch", "confidence": 99.98}

Below is a result from predicting an image of a **strawberry leaf** infected with **Leaf Scorch**:



*Figure 9: Sample input image showing a diseased strawberry leaf*

**Result Visualization:**

**Description**: The model accurately identified the disease as Strawberry\_\_\_Leaf\_scorch with a high confidence score of **99.98%**, clearly displayed in the prediction overlay.

### 6.2 Model Deployment on Web Platform

The trained **ResNet-18** model was deployed using a **Gradio web interface**, allowing users to upload plant leaf images and receive real-time disease predictions. The interface includes:

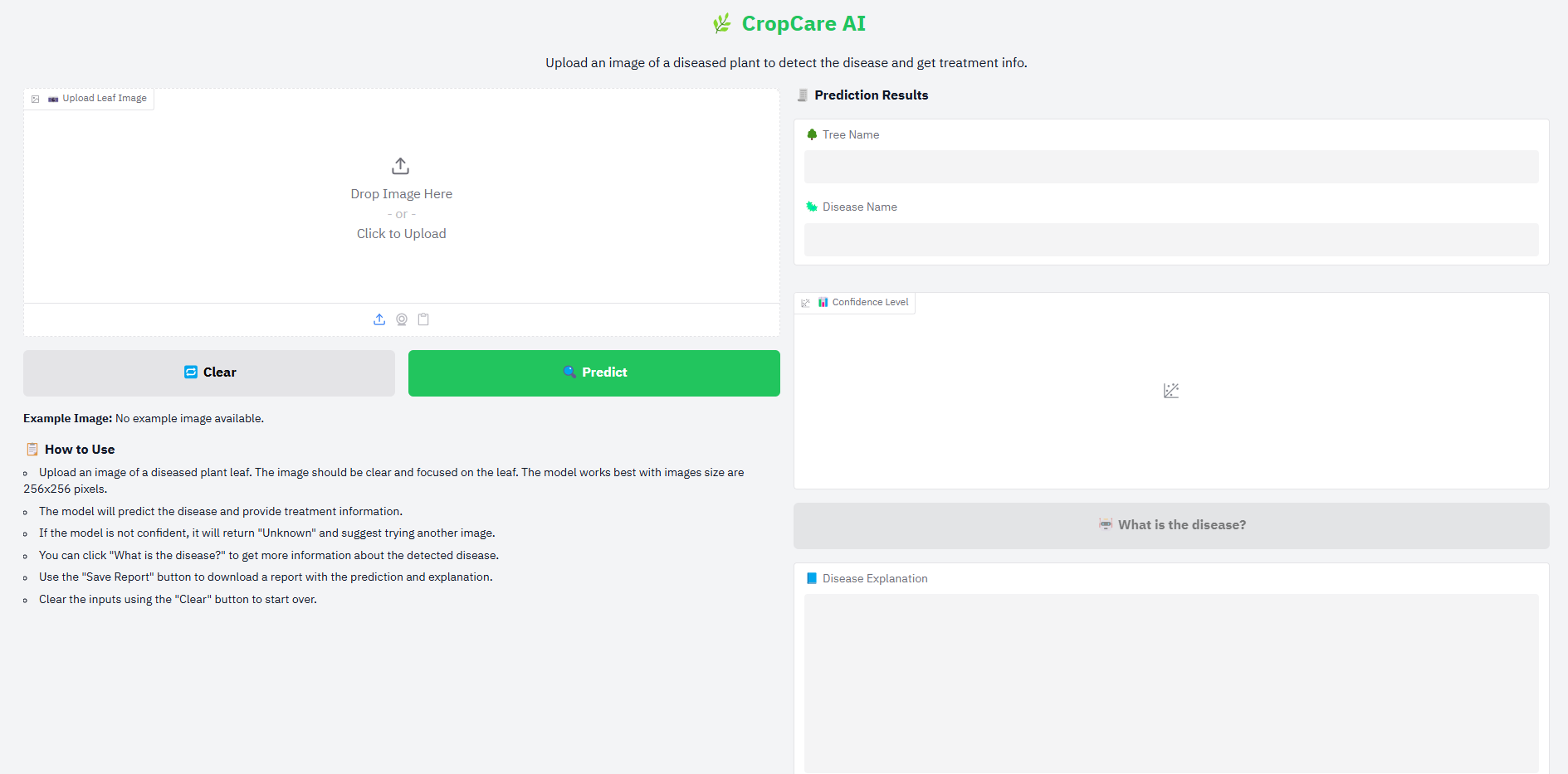
* Image upload field
* Disease prediction with confidence gauge
* GPT-powered treatment suggestions
* Option to save the prediction report

#### How to Use the Application:

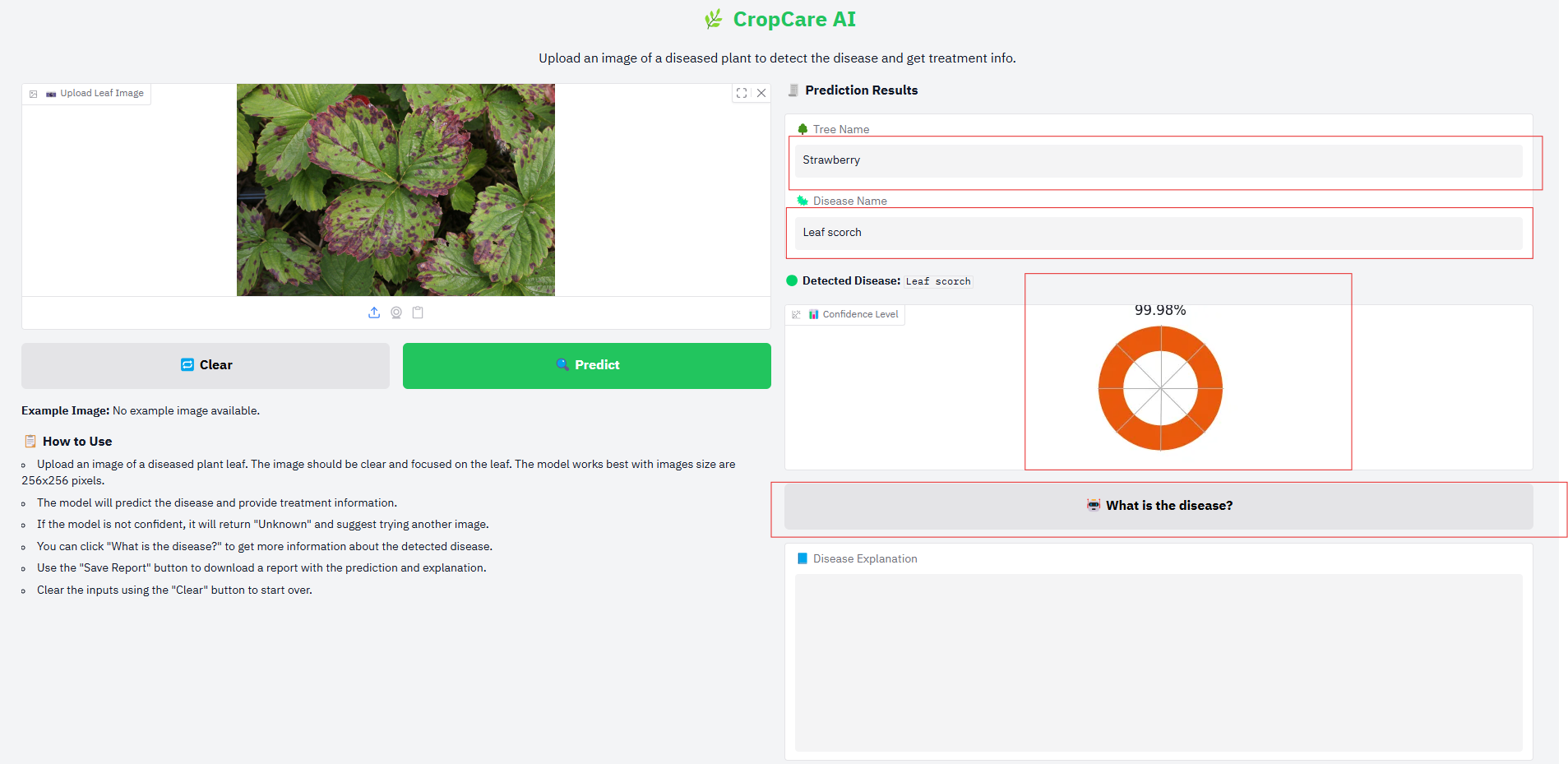
1. Run the Python script to launch the app.
2. Open the local URL provided in the terminal.
3. Upload a clear image of a plant leaf.
4. Click **“Predict”** to see the predicted disease and confidence score.
5. *(Optional)* Click **“What is the disease?”** to receive care and treatment advice via GPT.
6. Click **“Save Report”** to download the prediction and explanation.
7. Click **“Clear”** to reset the interface.

This deployment approach ensures ease of access for non-technical users and provides both prediction and actionable insights in a single, user-friendly interface.

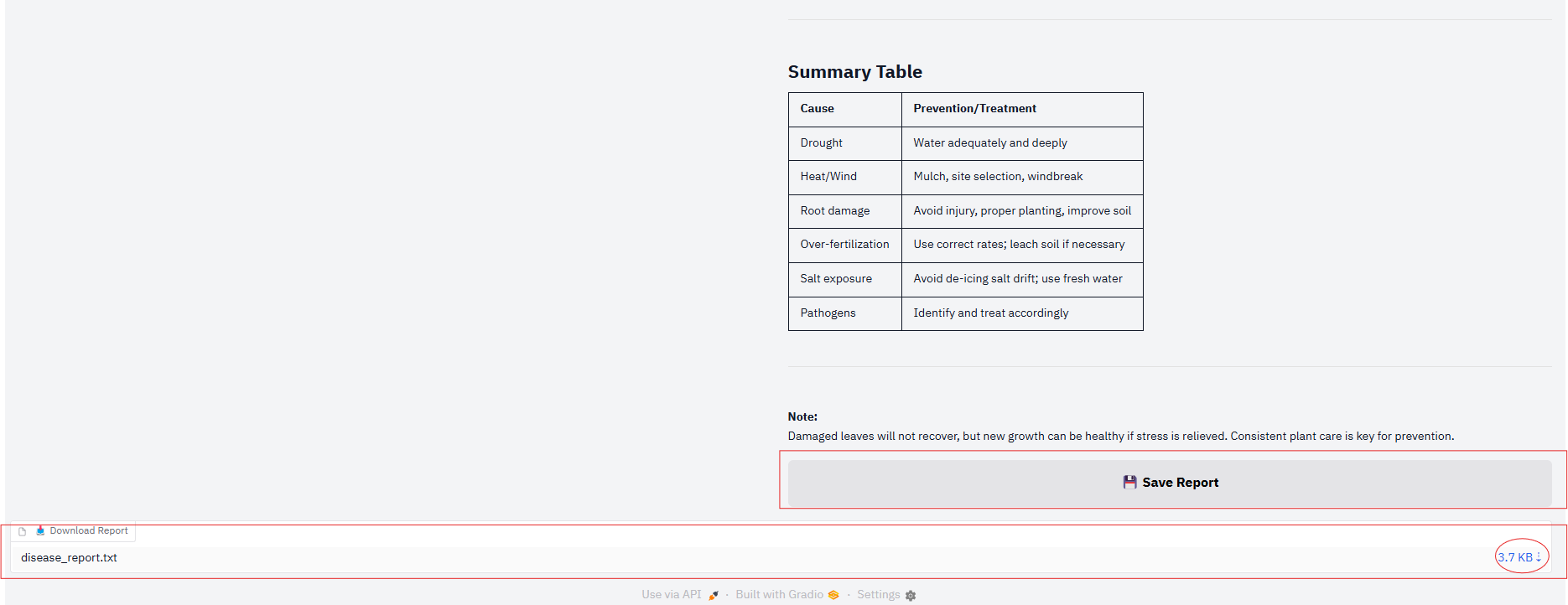
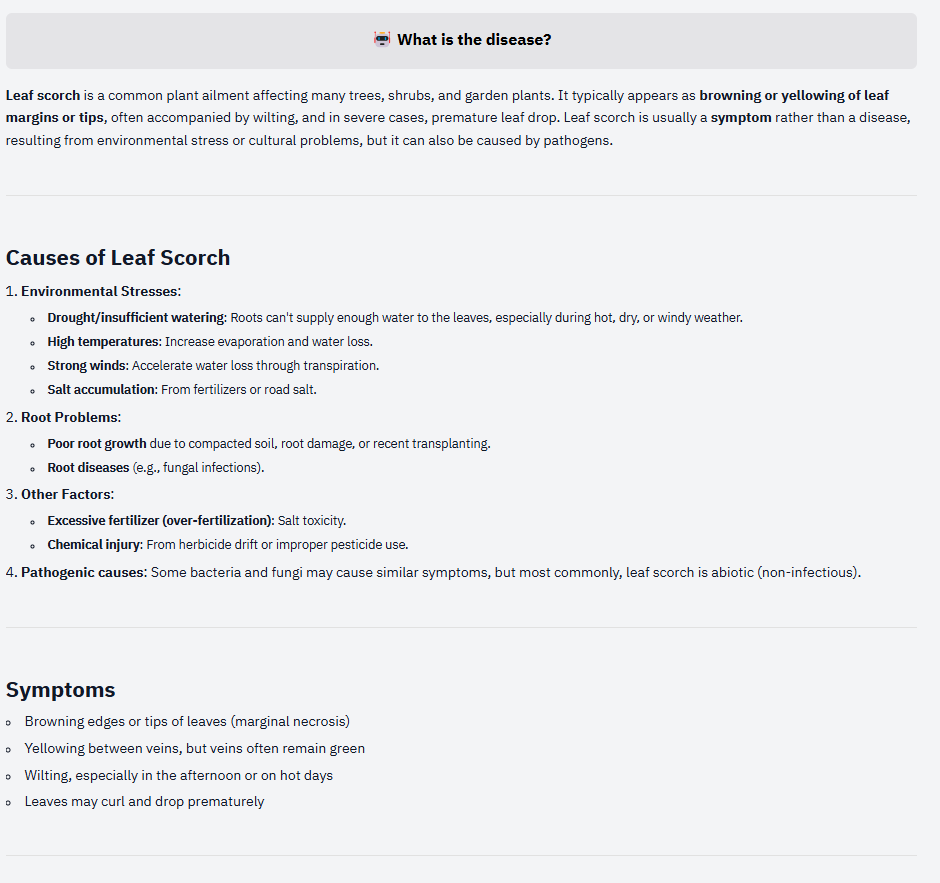
### 6.3 Example Output and User Experience Description

****

*Figure 10: Screenshot of the CropCare AI web interface showing prediction and user options*



*Figure 11: Screenshot of prediction result displayed by the web application*



*Figures 12 & 13: Screenshot of GPT-generated explanation and treatment suggestion*

### 6.4 Suggest Future Improvements

To further enhance the system's usability and reach, a key next step is deploying **CropCare AI on mobile devices**.

* This would enable **real-time disease detection** directly through the smartphone camera.
* Farmers and gardeners could **diagnose plant health instantly in the field**, without needing internet access.
* By leveraging **edge-device inference** and **lightweight model architectures** like MobileNet or EfficientNet-Lite, the system could deliver **fast, offline predictions** with minimal resource requirements.

Such mobile deployment would make the tool more **accessible, practical, and scalable** for use in diverse agricultural environments, especially in rural areas with limited connectivity.

## Appendix: Dataset and GitHub Repository Links

#### 1. Dataset Used in the Project

* Dataset Name: PlantVillage Dataset
* Description: Contains over 50,000 labeled images of healthy and diseased crop leaves across multiple plant species.
* Source: Kaggle (public domain)
* Dataset link: [GitHub - spMohanty/PlantVillage-Dataset: Dataset of diseased plant leaf images and corresponding labels](https://github.com/spMohanty/PlantVillage-Dataset)

#### 2. Project GitHub repository

* Project Repository: [GitHub - huyluongme/TMA\_AI\_Advance\_Group\_51: TMA AI Advance Group 51 - Plant Disease Detection](https://github.com/huyluongme/TMA_AI_Advance_Group_51?tab=readme-ov-file#1-demo)
* Contains:
  + Model training script: train\_resnet18.py
  + Preprocessing and augmentation code
  + Inference scripts: predict\_resnet18.py
  + Web deployment app built using Gradio
  + Sample images and documentation
  + Demo video: [demo.mp4](https://github.com/huyluongme/TMA_AI_Advance_Group_51/blob/master/demo.mp4)

## References:

[1] T. Liu, et al., “Theoretical Understanding of Convolutional Neural Networks,” *MDPI Mathematics*, vol. 11, no. 3, 2023. [Online]. Available: <https://www.mdpi.com/2079-3197/11/3/52>

[2] Z. Li, X. Yang, H. Liu, and X. Sun, “A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects,” *arXiv preprint*, arXiv:2004.02806, 2020. [Online]. Available: <https://arxiv.org/abs/2004.02806>

[3] L. L. Ankile, H. O. Omidiora, and T. O. Olabiyisi, “Deep Convolutional Neural Networks: A survey of the foundations, selected improvements, and some current applications,” *arXiv preprint*, arXiv:2011.12960, 2020. [Online]. Available: <https://arxiv.org/abs/2011.12960>

[4] A. Khan, A. Sohail, U. Zahoora, and A. S. Qureshi, “A Survey of the Recent Architectures of Deep Convolutional Neural Networks,” *arXiv preprint*, arXiv:1901.06032, 2019. [Online]. Available: <https://arxiv.org/abs/1901.06032>

[5] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” *arXiv preprint*, arXiv:1512.03385, 2015. [Online]. Available: <https://arxiv.org/abs/1512.03385>

[6] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L. Chen, “MobileNetV2: Inverted Residuals and Linear Bottlenecks,” *arXiv preprint*, arXiv:1801.04381, 2018. [Online]. Available: <https://arxiv.org/abs/1801.04381>