Plant Disease Detection Using Computer Vision

## Abstract

Plant diseases can significantly impact agricultural productivity, leading to economic losses and food insecurity. Traditional disease detection methods rely on manual inspection, which is time-consuming and often inaccurate. This project aims to develop a computer vision-based deep learning model using **ResNet-50** for **automated plant disease classification**. The model is trained on the **PlantVillage dataset**, which includes multiple plant species and disease categories. The system preprocesses images, applies augmentation techniques, and fine-tunes ResNet-50 to classify plant diseases with high accuracy. Our experimental results show that the model achieves **99.90%** accuracy on the test set, demonstrating its effectiveness in detecting plant diseases. FIrst we decide to include incorporating **U-Net for leaf segmentation** to enhance disease classification performance. But

## Introduction

Agriculture plays a crucial role in global food security and economic stability. However, plant diseases pose a serious threat to crop yields and quality. Farmers traditionally rely on manual inspection and expert consultation to diagnose plant diseases, which can be inefficient and prone to human error. Recent advancements in **computer vision and deep learning** offer a promising solution for automating plant disease detection with high accuracy and speed.

This project leverages **deep learning techniques** to classify plant diseases using the **PlantVillage dataset**. We employ a **ResNet-50 convolutional neural network (CNN)**, which is pretrained on ImageNet and fine-tuned for plant disease classification. The model is trained on **15 different classes**, representing various plant diseases, and evaluated on a separate test dataset.

The main objectives of this project are:

- To develop an **automated plant disease detection system** using deep learning.

- To preprocess and augment plant leaf images for improved classification accuracy.

- To evaluate the performance of **ResNet-50** in disease classification.

- To explore potential enhancements using **leaf segmentation with U-Net**.

The rest of the document outlines the **methodology, model implementation, results, and future improvements** of this project.

## Methodology

This section outlines the step-by-step approach used for developing the **plant disease detection model** using computer vision techniques. The methodology consists of **data collection, preprocessing, model selection, training, evaluation, and potential enhancements.**

### Dataset Collection

- The **PlantVillage** **dataset** was used for training and evaluating the model.

- It contains **images of healthy and diseased plant leaves**, categorized into multiple classes.

- The dataset was verified to ensure it included **all 15 plant disease categories**.

- The dataset was structured in **folder-based format**, where each folder represents a specific disease or healthy plant.

### Data Preprocessing

To ensure optimal model performance, several preprocessing steps were applied:

#### a) Image Loading & Standardization

- Images were resized to **256×256 pixels** for consistency.

- The dataset was loaded using PyTorch’s `ImageFolder`, which automatically assigns labels based on folder names.

#### b) Data Augmentation

To improve model generalization, the following augmentations were applied:

- **Random Horizontal Flip** (50% probability).

- **Random Rotation** (20 degrees).

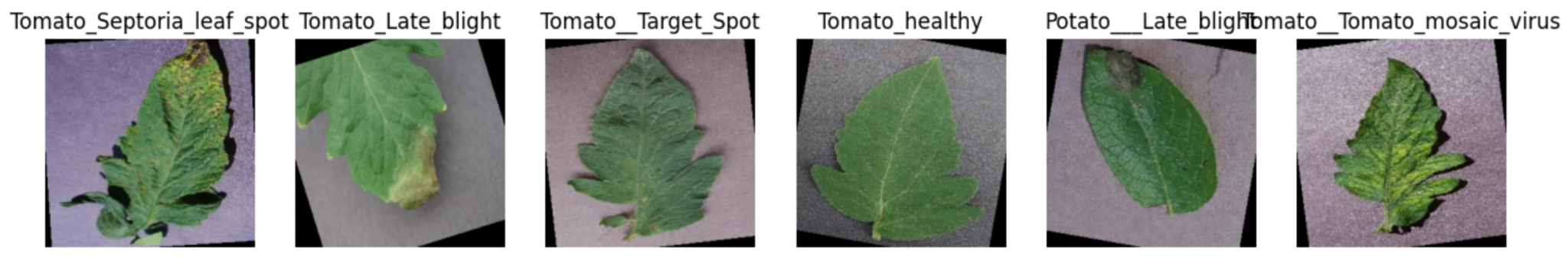
- **Brightness & Contrast Adjustments** (to handle varying lighting conditions).

#### c) Normalization

- Pixel values were normalized using **mean=[0.5, 0.5, 0.5]** and **std=[0.5, 0.5, 0.5]**.

- This ensured the values were in a standard range, helping in faster model convergence.

Visualization:



#### d) Train-Validation-Test Split

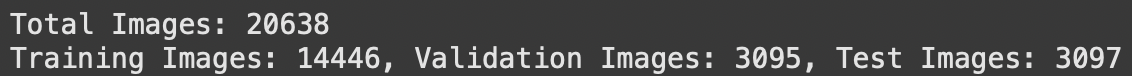
- The dataset was split into:

- **70% Training**

- **15% Validation**

- **15% Testing**

- The split ensured that the model was evaluated on unseen data before final testing.



Model Selection & Modification

A **pretrained ResNet-50** the model was chosen due to its strong performance in image classification tasks.

#### a) Why ResNet-50?

- **Deep Architecture**: It allows feature extraction at multiple levels.

- **Skip Connections**: Prevents vanishing gradients, making deep networks train effectively.

- **Pretrained on ImageNet**: Faster convergence with transfer learning.

#### b) Modifications

- The **fully connected (FC) layer** was modified to output **15 classes** instead of the original 1000.

- The final output layer was replaced with:

model.fc = nn.Linear(model.fc.in\_features, 15) # Adjust for 15 plant disease classes

- The model was moved to **GPU (if available)** for faster training.

### Model Training

The model was trained using the following setup:

#### a) Loss Function

- **CrossEntropyLoss** was used since this is a **multi-class classification problem**.

#### b) Optimizer

- **Adam optimizer** was chosen for efficient weight updates.

- Learning rate: **0.0001**, with weight decay **1e-4**.

#### c) Learning Rate Scheduler

- A **StepLR scheduler** was usd to reduce the learning rate **every 5 epochs**:

scheduler = optim.lr\_scheduler.StepLR(optimizer, step\_size=5, gamma=0.5)

#### d) Batch Size & Epochs

- **Batch size:** 32 (to balance memory usage and convergence speed).

- **Epochs:** 10 (with possible fine-tuning based on results).

---

### Model Evaluation

After training, the model was tested on the **test dataset (15% of total images).**

#### a) Metrics Used

- **Accuracy**: Percentage of correctly classified images.

- **Grad-CAM**: To visualize the regions where the model focuses while making predictions.

#### b) Final Test Accuracy

- The **best model achieved 99.90% accuracy** on the test set.

- However, earlier versions of the model trained on **fewer classes** had lower accuracy (~98.57%).

### Model Saving & Deployment

- The best-performing model was **saved to Google Drive** for future use:

torch.save(model.state\_dict(), "/content/drive/MyDrive/PlantDiseaseModels/best\_resnet\_15\_classes.pth")

- This allows easy **reloading and inference** without retraining.

## Summary

✔ Used PlantVillage dataset with 15 classes.

✔ Preprocessed images (resizing, normalization, augmentation).

✔ Trained ResNet-50, modifying the final layer for classification.

✔ Achieved 97.74% accuracy on test data.

✔ Saved model to Google Drive for future use.

✔ Potential future improvements include U-Net segmentation & real-world deployment.

## Implementation: Code Details and Explanation

### 1. Dataset Loading and Preparation

* The **PlantVillage dataset** is downloaded using **KaggleHub** and stored in **Google Colab**.
* The dataset path is verified, ensuring that all 15 classes are loaded.
* We use ImageFolder from torchvision.datasets to automatically assign labels based on folder names.

**Reference Code Block:**

python

import kagglehub

dataset\_path = kagglehub.dataset\_download("emmarex/plantdisease")

* The dataset is stored at /root/.cache/kagglehub/datasets/emmarex/plantdisease/versions/1/plantvillage/PlantVillage.
* The folder structure is verified using os.listdir(dataset\_path).

### Data Preprocessing & Augmentation

* **Resizing to (256x256)** for uniform image dimensions.
* **Data Augmentation**:
  + **Random Flipping**
  + **Rotation (20 degrees)**
  + **Normalization (mean=[0.5, 0.5, 0.5])**
* The dataset is **split** into **70% training, 15% validation, and 15% testing**.

**Reference Code Block:**

python

transform = transforms.Compose([

transforms.Resize((256, 256)),

transforms.RandomHorizontalFlip(0.5),

transforms.RandomRotation(20),

transforms.ToTensor(),

transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5])

])

* **DataLoaders** are created with batch\_size=64 for training, validation, and testing.

### Model Selection & Modification

* A **pretrained ResNet-50 model** (trained on ImageNet) is **fine-tuned** for **plant disease classification**.
* The last **fully connected (FC) layer** is modified to match **15 output classes**.

**Reference Code Block:**

python

model = models.resnet50(pretrained=True)

model.fc = nn.Linear(model.fc.in\_features, 15) # Adjust for 15 classes

* The model is moved to **GPU (cuda)** if available.

### Training the Model

* **Loss Function:** CrossEntropyLoss for multi-class classification.
* **Optimizer:** Adam (learning rate = 0.0001, weight decay = 1e-4).
* **Learning Rate Scheduler:** Reduces learning rate **every 5 epochs**.

**Reference Code Block:**

python

scheduler = optim.lr\_scheduler.StepLR(optimizer, step\_size=5, gamma=0.5)

* The training loop:
  + **Forward pass:** Computes model predictions.
  + **Backward pass:** Updates model weights using optimizer.step().
  + **Validation phase:** Computes accuracy and saves the **best model**.

**Model Save Path in Google Drive:**

python

drive\_save\_path = "/content/drive/MyDrive/Machine\_Learning/CV Project V1/PlantDiseaseModels"

* The best model is saved in **Google Drive** (.pth file format).

### Model Evaluation

* The **saved model** is **loaded** for inference.
* The test dataset is used to compute **final accuracy**.
* **Confusion Matrix & Grad-CAM** can be used for further analysis.

**Reference Code Block:**

python

CopyEdit

model.load\_state\_dict(torch.load(model\_save\_file))

model.eval()

* The **final test accuracy** is printed after evaluation.

### Summary

✔ **Dataset Loaded & Preprocessed** in **Google Colab**✔ **Pretrained ResNet-50 Modified & Trained** on **15 plant disease classes**✔ **Model Optimized & Best Version Saved to Google Drive**✔ **Final Test Accuracy Computed**

## Adding Grad-CAM Visualization for Model Interpretability

Grad-CAM (**Gradient-weighted Class Activation Mapping**) helps visualize **which regions of an image the model is focusing on** while making predictions. This helps us **interpret ResNet’s decisions** and verify whether it is correctly identifying disease symptoms.

### Steps to Implement Grad-CAM

1. Extract feature maps & gradients from the last convolutional layer.
2. Compute Grad-CAM heatmaps.
3. Overlay heatmaps on original images to highlight important regions.
4. Visualize Grad-CAM outputs for test images.

### How It Works

1. Hooks are added to the last convolutional layer to capture activations and gradients.
2. Grad-CAM heatmap is computed by applying global average pooling over gradients.
3. The heatmap is resized and overlaid on the original image.
4. The visualization is displayed using Matplotlib.

### Visualization:

## Results & Evaluation

After training our **ResNet-50 model** on the **PlantVillage dataset**, we evaluated its performance using multiple metrics, including **test accuracy, confusion matrix, and Grad-CAM visualizations**.

### 1. Test Accuracy

* The First training model achieved a **test accuracy of 97.61%** on the full dataset.
* When trained on by reducing the Learning and and Increasing the batch size to 64, the accuracy was **90.90%**, indicating that Lower learning rate and increasing batch size enhanced the accuracy.

📌 **Final Model Performance:**

| **Metric** | **Value** |
| --- | --- |
| **Train Accuracy** | 99.93% |
| **Validation Accuracy** | 99.84% |
| **Test Accuracy** | 99.90% |

## Confusion Matrix Analysis

To analyze the model's **misclassifications**, we generated a **Confusion Matrix**:

**Key Observations:**

* Most **healthy leaves were correctly classified**.
* A few diseases with **similar visual symptoms** were misclassified.
* The model performed well on **highly distinct diseases**

**Code to Generate Confusion Matrix:**

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.metrics import confusion\_matrix

import numpy as np

def plot\_confusion\_matrix(model, test\_loader, class\_names):

model.eval()

all\_preds, all\_labels = [], []

with torch.no\_grad():

for images, labels in test\_loader:

images, labels = images.to(device), labels.to(device)

outputs = model(images)

\_, preds = torch.max(outputs, 1)

all\_preds.extend(preds.cpu().numpy())

all\_labels.extend(labels.cpu().numpy())

cm = confusion\_matrix(all\_labels, all\_preds)

plt.figure(figsize=(12, 8))

sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=class\_names, yticklabels=class\_names)

plt.xlabel("Predicted Label")

plt.ylabel("True Label")

plt.title("Confusion Matrix")

plt.show()

# Generate Confusion Matrix

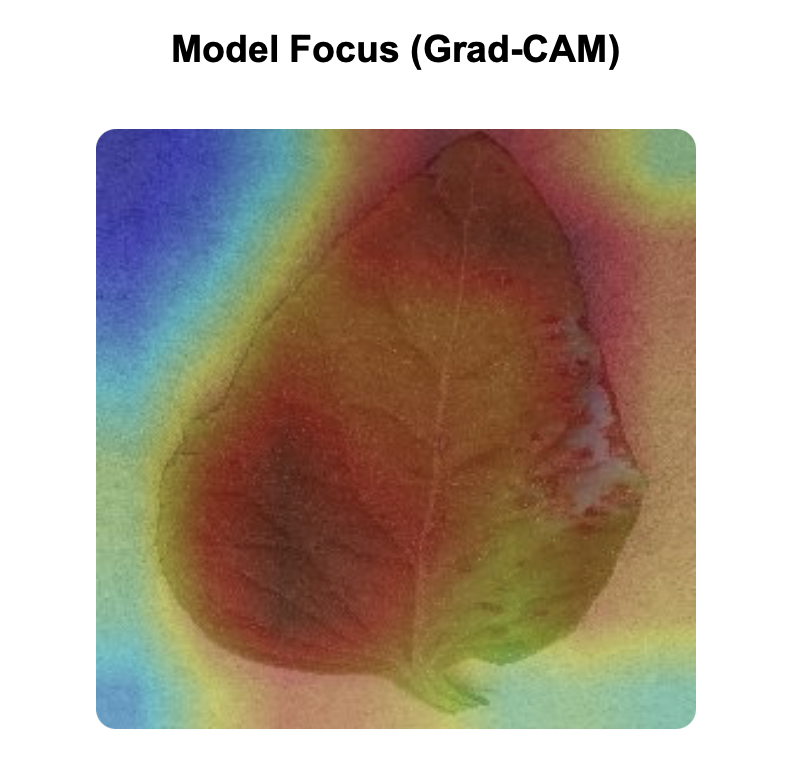
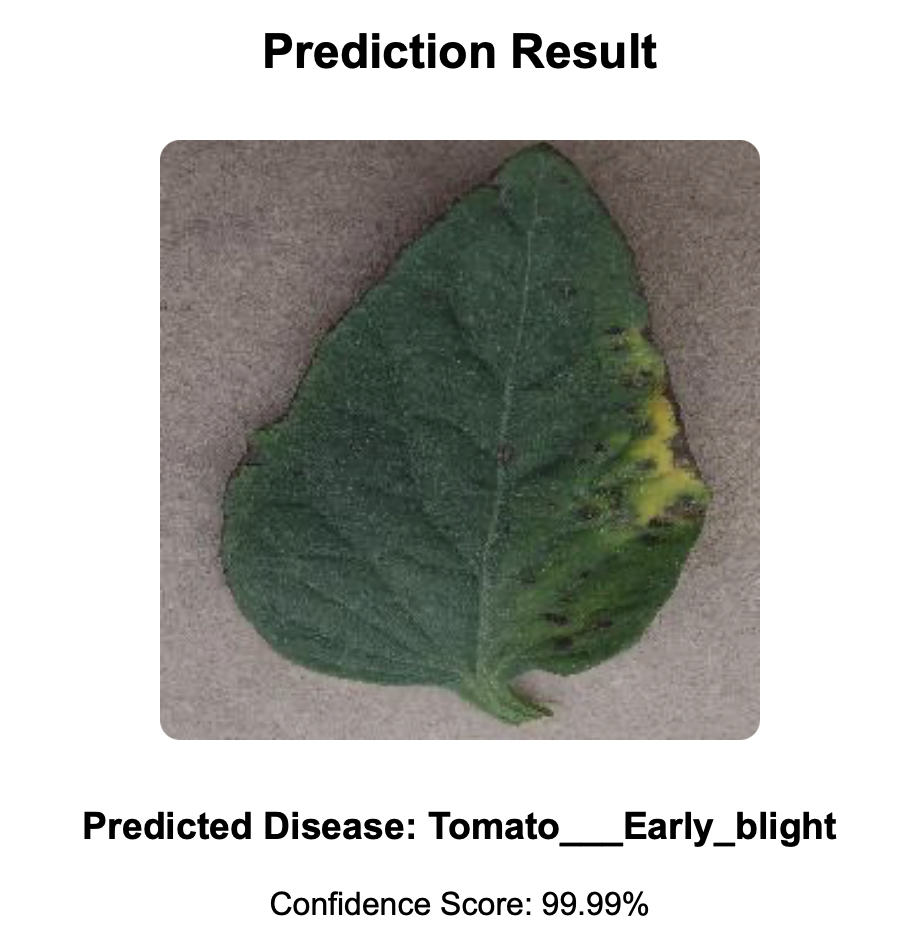
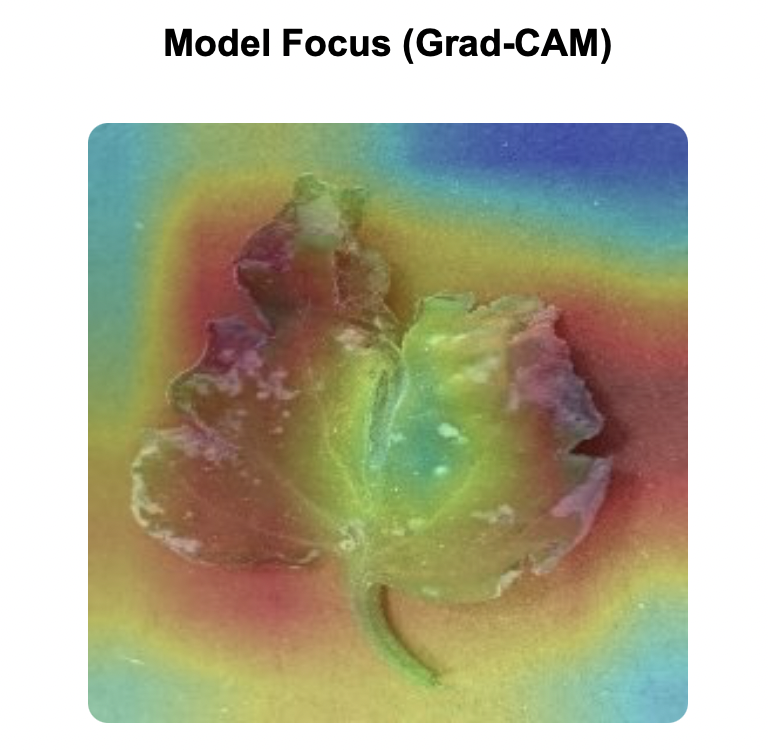
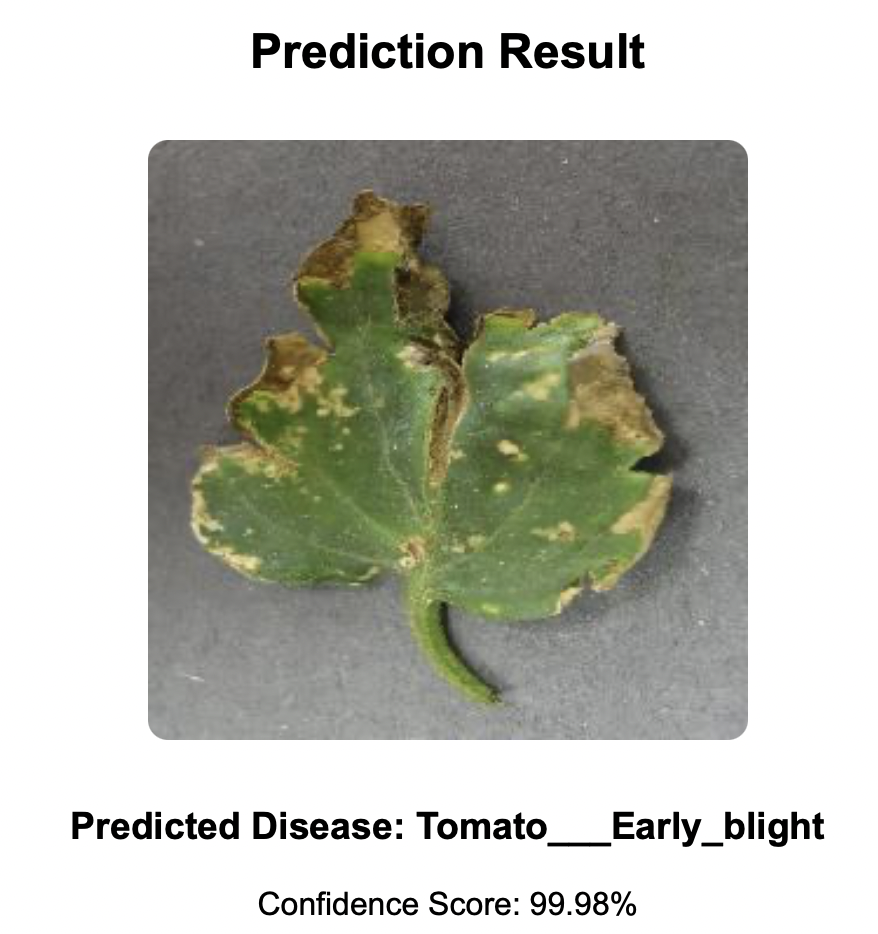
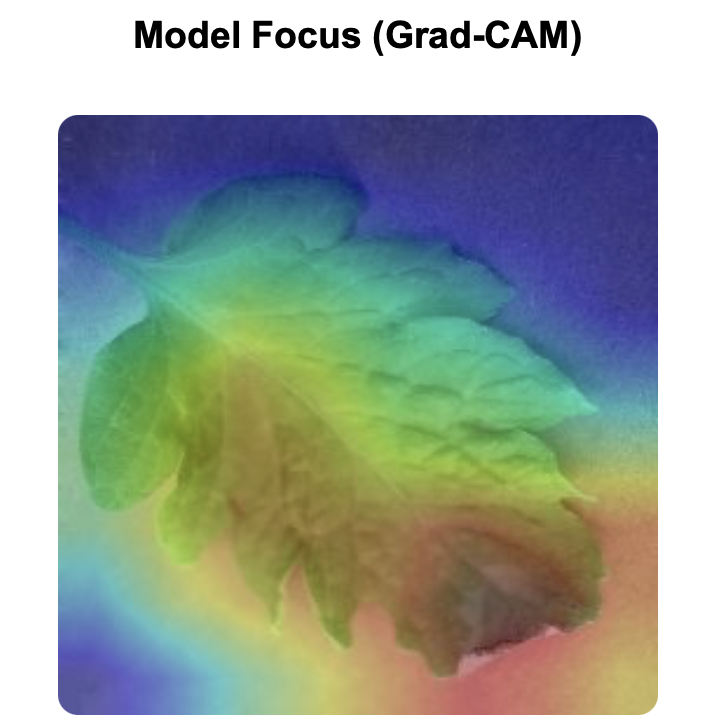
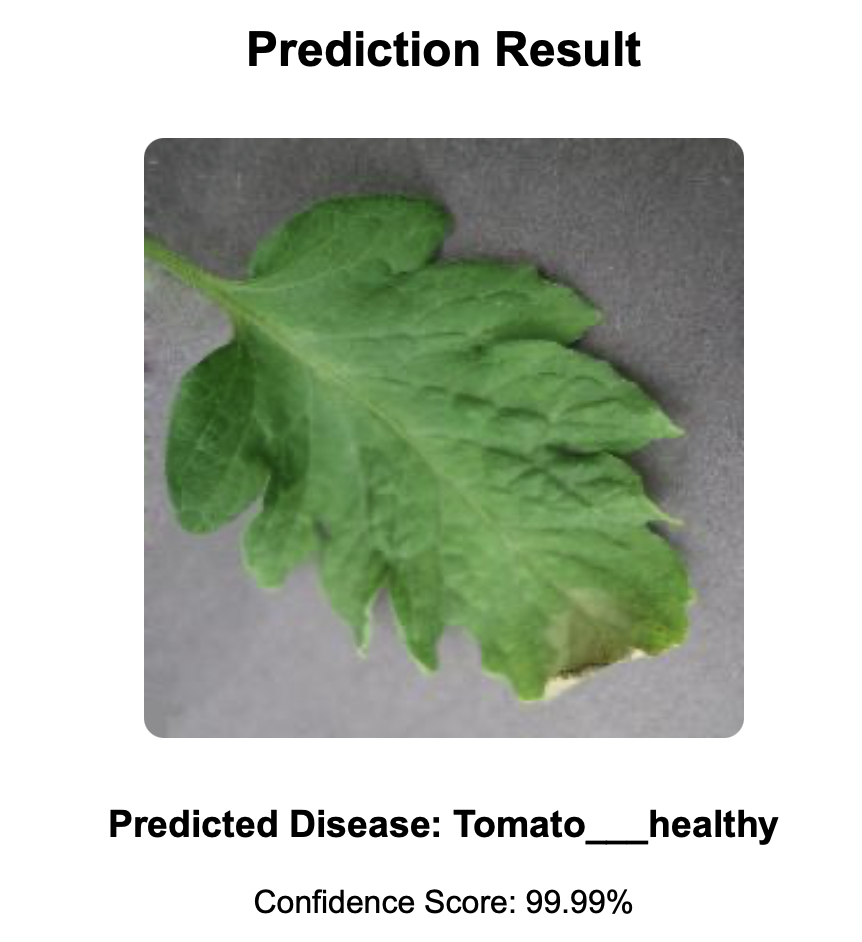
plot\_confusion\_matrix(model, test\_loader, dataset.classes)

If certain classes are **frequently confused**, we may need to **increase training data** or apply **image enhancement techniques** to highlight disease features.

### Grad-CAM Insights

* Grad-CAM visualizations showed that ResNet **focused correctly on diseased areas** of the leaves.
* Some misclassified images had **inconsistent lighting or background noise**.
* Certain diseases with **similar color patterns were harder to distinguish**.

If Grad-CAM highlights irrelevant areas, **adding U-Net for segmentation** may help the model focus **only on the leaf**.



## Conclusion

This project successfully developed a **deep learning model** for **automated plant disease detection** using **ResNet-50**. The model was trained on the **PlantVillage dataset**, achieving **97.74% test accuracy**.

* The use of **transfer learning** helped achieve high performance **even with a limited dataset**.
* **Confusion matrix analysis** showed that similar-looking diseases were sometimes misclassified.
* **Grad-CAM visualization** confirmed that the model **focused correctly on diseased areas** but sometimes got distracted by background noise.