REVA UNIVERSITY
Bengaluru, India

ASSESSMENT - 1

TITLE

Tomato Leaf Disease Early Prediction - Deep Learning

Submitted by

Class and Section: ISE-A

Atharv Patil – R23EQ011

Chandan M – R23EQ015

Under the Guidance of: Ms. Bhavana

Date: 01th December 2025

Abstract

Tomato leaf diseases significantly impact crop yield and quality, making early diagnosis crucial for effective crop management. Traditional manual inspection by farmers and experts is time-consuming, subjective, and often ineffective at early stages of infection. With advances in deep learning, convolutional neural networks (CNNs) have shown strong performance in image-based disease detection.

In this project, we use a **MobileNetV2-based CNN** model with **transfer learning** for tomato leaf disease classification and early prediction. The model is trained on tomato leaf images and fine-tuned to distinguish between healthy leaves and multiple disease classes. To make the system explainable and practically useful for farmers, we integrate **Grad-CAM** visualization to highlight infected regions and estimate the **severity** of disease based on the activated area.

The trained model is deployed as a **FastAPI web service** with a simple web interface that allows users to upload a tomato leaf image and receive predictions, severity level, and a Grad-CAM heatmap. Experimental results show that the proposed approach can accurately detect diseases and provide interpretable feedback useful for early intervention.

1. Introduction

1.1 Background

Tomato is one of the most widely cultivated crops, and its yield is heavily affected by leaf diseases such as Early Blight, Late Blight, Leaf Mold, and others. Early identification of these diseases is important to prevent spread and minimize economic loss.

Traditional methods rely on:

- Manual visual inspection
- Expert consultation

These are:

- Time-consuming
- Error-prone
- Not scalable for large farms

Deep learning, especially **Convolutional Neural Networks (CNNs)**, has become a powerful tool for image-based classification tasks. Pretrained models like **MobileNetV2** can be fine-tuned on agricultural datasets to detect diseases from leaf images efficiently, even on low-resource devices.

1.2 Problem Statement

The problem is to automatically detect and classify tomato leaf diseases from images and provide **early prediction** with an estimate of severity. The system should:

- Work with real leaf images captured from a phone or camera
- Identify whether the leaf is healthy or infected
- Highlight infected regions and severity for decision support
- Be deployable as a simple web application for practical use

1.3 Objectives

- To develop a **MobileNetV2-based CNN** model for tomato leaf disease classification.
- To preprocess and augment tomato leaf images for robust training.
- To integrate **Grad-CAM** for visual explanation and severity estimation.
- To deploy the trained model using **FastAPI** and a web frontend for user interaction.
- To enable early disease prediction by detecting small infection regions and mapping them to risk levels.

2. Literature Review

2.1 Traditional Disease Detection Methods

Conventional approaches for plant disease detection include:

- Manual visual inspection by experts
- Rule-based systems using handcrafted features (color, texture, shape)
- Classical machine learning models such as SVMs, k-NN, and Random Forest

These methods require domain expertise and often fail when background, lighting, or leaf orientation changes. They also struggle with subtle and early-stage infections.

2.2 CNN-Based Approaches

CNNs automatically learn hierarchical features from images, making them highly suitable for leaf disease classification. Models like VGG, ResNet, and MobileNet have been successfully applied to:

- Classifying multiple plant diseases
- Handling complex backgrounds
- Working directly from raw images without manual feature engineering

2.3 MobileNetV2 and Explainable AI (Grad-CAM)

MobileNetV2 is a lightweight CNN model optimized for mobile and embedded devices. It uses depthwise separable convolutions and inverted residual blocks, offering:

- Low computation
- Small model size
- Good accuracy

Grad-CAM (Gradient-weighted Class Activation Mapping) is an explainability method that produces heatmaps indicating which parts of the image contributed most to the model's prediction. For agricultural use, Grad-CAM is critical to:

- Show infected regions clearly
- Provide trust and interpretability
- Estimate severity based on hotspot area

3. Methodology

3.1 Dataset and Preprocessing

- Dataset: Tomato leaf images labeled into multiple disease classes and healthy.
- Images are resized to **160×160** pixels.
- Preprocessing includes:
    - Resizing and center cropping
    - Normalization using ImageNet mean and std
    - Data augmentation (horizontal flipping, random cropping) for robustness

The dataset is split into **training** and **validation** subsets, commonly in 80/20 ratio.

3.2 Model Architecture

- Base model: **MobileNetV2** pretrained on ImageNet
- Last classification layer is replaced to output **N** tomato classes (e.g., 11)
- Loss function: **CrossEntropyLoss**
- Optimizer: **AdamW**
- Device: Apple Silicon GPU (`mps`) or CPU depending on availability

3.3 Training

- The model is trained for a fixed number of epochs (e.g., 10).
- At each epoch:
    - Forward pass over batches
    - Loss computation
    - Backpropagation and weight update
- Validation accuracy is computed after every epoch.
- The best model (highest validation accuracy) is saved.

3.4 Inference, Grad-CAM and Severity Estimation

- Input image is preprocessed and passed through the trained model.
- `softmax` is applied to obtain class probabilities.
- Grad-CAM is computed using the last convolutional feature maps and class-specific gradients.
- The Grad-CAM heatmap is resized to original image size.
- Severity (%) is estimated from the fraction of pixels where activation is above a threshold.
- Stage and risk levels are mapped from severity (e.g., Very Early, Early, Moderate, Severe).

3.5 Deployment with FastAPI

- Model is loaded once at FastAPI startup.
- Endpoint `/predict` accepts an uploaded image file.
- The API:
    - Validates if the image is a leaf (by green color filter)
    - Runs prediction
    - Generates Grad-CAM overlay image and saves it in `/static`
    - Returns JSON with disease, confidence, severity, stage, risk and Grad-CAM URL
- A simple `index.html` frontend allows users to upload an image and see results.

4. Results and Discussion

- The MobileNetV2-based model achieved high validation accuracy on the tomato leaf dataset.
- Grad-CAM successfully highlighted disease spots, making predictions more interpretable.
- Severity mapping helped categorize infection into early, moderate, and severe stages.
- The FastAPI deployment enabled real-time prediction via a browser-based interface, making the solution practical for potential field use.

4.1 Future Work

Future work can include:

- Training on a larger and more diverse dataset with different lighting and backgrounds.
- Including more crops and disease types in a unified system.
- Integrating mobile apps for offline usage.
- Using attention-based models or Vision Transformers.
- Combining leaf image data with weather and soil data for richer prediction.

Code:

1) train.py – Model Training

```
import os
import torch
import torch.nn as nn
from torch.utils.data import DataLoader
from torchvision import datasets, transforms, models
# Device (Apple Silicon GPU if available)
DEVICE = "mps" if torch.backends.mps.is_available() else "cpu"
print("Using device:", DEVICE)
DATA_DIR = "data"          # data/train, data/valid structure
NUM_CLASSES = 11           # change as per your dataset
BATCH_SIZE = 32
EPOCHS = 10
LR = 1e-4
MODEL_PATH = "models/best_model.pth"
IMG_SIZE = 160
def get_dataloaders():
    train_tf = transforms.Compose([
        transforms.Resize((IMG_SIZE, IMG_SIZE)),
        transforms.RandomHorizontalFlip(),
        transforms.RandomResizedCrop(IMG_SIZE, scale=(0.8, 1.0)),
        transforms.ToTensor(),
        transforms.Normalize([0.485,0.456,0.406],
                             [0.229,0.224,0.225])
    ])

    valid_tf = transforms.Compose([
```

```
        transforms.Resize((IMG_SIZE, IMG_SIZE)),
        transforms.CenterCrop(IMG_SIZE),
        transforms.ToTensor(),
        transforms.Normalize([0.485,0.456,0.406],
                             [0.229,0.224,0.225])
    ])

    train_ds = datasets.ImageFolder(os.path.join(DATA_DIR, "train"), transform=train_tf)
    valid_ds = datasets.ImageFolder(os.path.join(DATA_DIR, "valid"), transform=valid_tf)

            train_dl = DataLoader(train_ds, batch_size=BATCH_SIZE, shuffle=True,
num_workers=0)
            valid_dl = DataLoader(valid_ds, batch_size=BATCH_SIZE, shuffle=False,
num_workers=0)

    return train_dl, valid_dl, train_ds.classes

def train():
    os.makedirs("models", exist_ok=True)
    train_dl, valid_dl, classes = get_dataloaders()

    # MobileNetV2 with pretrained weights
    model = models.mobilenet_v2(weights=models.MobileNet_V2_Weights.DEFAULT)
    model.classifier[1] = nn.Linear(model.classifier[1].in_features, NUM_CLASSES)
    model.to(DEVICE)

    loss_fn = nn.CrossEntropyLoss()
    optimizer = torch.optim.AdamW(model.parameters(), lr=LR)

    best_acc = 0.0
```

```
for epoch in range(EPOCHS):
    # Training
    model.train()
    running_loss = 0.0
    for x, y in train_dl:
        x, y = x.to(DEVICE), y.to(DEVICE)
        optimizer.zero_grad()
        out = model(x)
        loss = loss_fn(out, y)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
    # Validation
    model.eval()
    correct = 0
    total = 0
    with torch.no_grad():
        for x, y in valid_dl:
            x, y = x.to(DEVICE), y.to(DEVICE)
            out = model(x)
            _, pred = torch.max(out, 1)
            correct += (pred == y).sum().item()
            total += y.size(0)

    acc = correct / total
    avg_loss = running_loss / len(train_dl)
    print(f"Epoch {epoch+1}/{EPOCHS} | Loss: {avg_loss:.4f} | Valid Acc: {acc:.4f}")

    # Save best model
```

```
        if acc > best_acc:
            best_acc = acc
            torch.save({"model": model.state_dict(), "classes": classes}, MODEL_PATH)
            print("🔥 Best model saved ✔")
    print("\nTraining Finished 🎯")
    print("Best Validation Accuracy:", best_acc)
if __name__ == "__main__":
    train()
```

2) infer.py – Offline Inference + Grad-CAM + Severity

```
import torch
import torch.nn as nn
import torch.nn.functional as F
from torchvision import models, transforms
from PIL import Image
import numpy as np
import cv2

DEVICE = "mps" if torch.backends.mps.is_available() else "cpu"
MODEL_PATH = "models/best_model.pth"
IMG_SIZE = 160

pre_tf = transforms.Compose([
    transforms.Resize((IMG_SIZE, IMG_SIZE)),
    transforms.CenterCrop(IMG_SIZE),
    transforms.ToTensor(),
    transforms.Normalize([0.485,0.456,0.406],
                         [0.229,0.224,0.225]),
])

def load_model():
    ckpt = torch.load(MODEL_PATH, map_location=DEVICE)
    classes = ckpt["classes"]

    model =
models.mobilenet_v2(weights=models.MobileNet_V2_Weights.DEFAULT)
    model.classifier[1] = nn.Linear(model.classifier[1].in_features, len(classes))
    model.load_state_dict(ckpt["model"])
```

```
    model.to(DEVICE).eval()
    return model, classes

def gradcam(model, x, idx):
    # Hook to capture features
    handle = model.features[18].register_forward_hook(
        lambda m, i, o: setattr(model, "feat", o)
    )

    model.zero_grad()
    out = model(x)
    out[0, idx].backward()

    fmap = model.feat.detach().cpu().numpy()[0]     # [C,H,W]
    grad = model.classifier[1].weight[idx].detach().cpu().numpy()  # [C]

    cam = np.tensordot(grad, fmap, axes=(0, 0))    # [H,W]
    cam = np.maximum(cam, 0)
    cam = cam / (cam.max() + 1e-8)

    handle.remove()
    return cam
def stage_and_severity(cam_resized):
    infected_pixels = np.sum(cam_resized > 0.3)
    total_pixels = cam_resized.size
    frac = infected_pixels / (total_pixels + 1e-8)
    severity = frac * 100.0
    if severity < 5:
        stage = "VERY EARLY / MINIMAL"
        risk = "LOW to MEDIUM"
    elif severity < 20:
        stage = "EARLY"
        risk = "HIGH"
    elif severity < 30:
        stage = "MODERATE"
        risk = "VERY HIGH"
    else:
        stage = "SEVERE"
        risk = "CRITICAL"
    return severity, stage, risk
def infer(img_path):
    model, classes = load_model()
    img = Image.open(img_path).convert("RGB")
```

```
    x = pre_tf(img).unsqueeze(0).to(DEVICE)
    with torch.no_grad():
        out = model(x)
        prob = F.softmax(out, 1)[0]
    idx = prob.argmax().item()
    disease = classes[idx]
    conf = float(prob[idx])

    # Grad-CAM
    cam = gradcam(model, x, idx)
    cam_resized = cv2.resize(cam, (img.size[0], img.size[1]))
    severity, stage, risk = stage_and_severity(cam_resized)
    # Early warning logic for healthy label
    is_healthy = disease.lower().startswith("healthy")
    early_note = "No"
    if is_healthy and severity > 3.0:
        stage = "POSSIBLE EARLY INFECTION"
        risk = "ELEVATED"
        early_note = "YES – Possible early infection region detected"
    # Visualization
    heatmap = cv2.applyColorMap(np.uint8(255 * cam_resized),
cv2.COLORMAP_JET)

  overlay = cv2.addWeighted(np.array(img), 0.6, heatmap, 0.4, 0)
    save_path = "gradcam_output.jpg"
    cv2.imwrite(save_path, cv2.cvtColor(overlay, cv2.COLOR_RGB2BGR))

    print("\n🧠 Predicted Label:", disease)
    print("📊 Confidence:", round(conf, 4))
    print("🔥 Severity:", f"{severity:.2f}%")
    print("🔎 Stage:", stage)
    print("🚨 Future Risk:", risk)
    if is_healthy:
        print("🟢 Healthy Leaf Early Warning:", early_note)
    print("📌 Grad-CAM Saved:", save_path)
if __name__ == "__main__":
    infer("sample_leaf.jpg")
```

3) `api.py` – FastAPI Backend

```
from fastapi import FastAPI, UploadFile, File
from fastapi.responses import HTMLResponse
from fastapi.staticfiles import StaticFiles

import torch
import torch.nn as nn
import torch.nn.functional as F
from torchvision import models, transforms
from PIL import Image
import numpy as np
import cv2
import io
import os

DEVICE = "mps" if torch.backends.mps.is_available() else "cpu"
MODEL_PATH = "models/best_model.pth"
IMG_SIZE = 160

app = FastAPI()

# Serve static files (Grad-CAM images)
app.mount("/static", StaticFiles(directory="static"), name="static")

# Serve homepage
@app.get("/", response_class=HTMLResponse)
def serve_homepage():
    with open("index.html", "r") as f:
        html_content = f.read()
    return HTMLResponse(content=html_content)

pre_tf = transforms.Compose([
    transforms.Resize((IMG_SIZE, IMG_SIZE)),
    transforms.CenterCrop(IMG_SIZE),
    transforms.ToTensor(),
    transforms.Normalize([0.485,0.456,0.406],
                         [0.229,0.224,0.225]),
])

model = None
classes = None

@app.on_event("startup")
def load_model():
    global model, classes
    ckpt = torch.load(MODEL_PATH, map_location=DEVICE)
    classes = ckpt["classes"]
    model_local = models.mobilenet_v2(weights=models.MobileNet_V2_Weights.DEFAULT)
```

```
    model_local.classifier[1] = nn.Linear(model_local.classifier[1].in_features,
                                          len(classes))
    model_local.load_state_dict(ckpt["model"])
    model_local.to(DEVICE).eval()
    print("Model loaded with classes:", classes)
    # assign to global
    globals()["model"] = model_local

def gradcam(x, idx):
    handle = model.features[18].register_forward_hook(
        lambda m, i, o: setattr(model, "feat", o)
    )
    model.zero_grad()
    pred = model(x)
    pred[0, idx].backward()

    fmap = model.feat.detach().cpu().numpy()[0]
    grad = model.classifier[1].weight[idx].detach().cpu().numpy()
    cam = np.tensordot(grad, fmap, axes=(0, 0))
    cam = np.maximum(cam, 0)
    cam = cam / (cam.max() + 1e-8)
    handle.remove()
    return cam

@app.post("/predict")
async def predict(file: UploadFile = File(...)):
    if model is None:
        return {"error": True, "message": "Model not loaded yet."}
    # Read and preprocess image
    img = Image.open(io.BytesIO(await file.read())).convert("RGB")
    img_np = np.array(img)
    # Quick leaf validation using green color
    hsv = cv2.cvtColor(img_np, cv2.COLOR_RGB2HSV)
    lower_green = np.array([25, 40, 40])
    upper_green = np.array([95, 255, 255])
    mask_green = cv2.inRange(hsv, lower_green, upper_green)
    green_percentage = (mask_green > 0).mean() * 100
    if green_percentage < 15:
        return {
            "error": True,
            "message": "❌ Not a tomato leaf! Upload a clear tomato leaf image."
        }
    x = pre_tf(img).unsqueeze(0).to(DEVICE)
with torch.no_grad():
        out = model(x)
        prob = F.softmax(out, 1)[0]
    confidence, idx = torch.max(prob, 0)
    confidence = float(confidence)
    disease = classes[idx]
```

```
# Reject low confidence images
 if confidence < 0.60:
    return {
       "error": True,
       "message": "⚠️ Image unclear — retake a clear picture of a leaf."
    }

 # Grad-CAM
 cam = gradcam(x, idx)
 cam_resized = cv2.resize(cam, (img.size[0], img.size[1]))
 heatmap = cv2.applyColorMap(np.uint8(255 * cam_resized), cv2.COLORMAP_JET)
 overlay = cv2.addWeighted(img_np, 0.6, heatmap, 0.4, 0)
 os.makedirs("static", exist_ok=True)
 save_path = "static/gradcam_output.jpg"
 cv2.imwrite(save_path, cv2.cvtColor(overlay, cv2.COLOR_RGB2BGR))

 # Severity estimation
 infected_pixels = np.sum(cam_resized > 0.3)
 total_pixels = cam_resized.size
 severity = (infected_pixels / (total_pixels + 1e-8)) * 100

 # Simple stage & risk logic
 if disease.lower().startswith("healthy") and confidence > 0.90:
    stage = "SAFE"
    risk = "LOW"
    severity = max(severity, 1.0)  # very small
 else:
    if severity < 20:
       stage = "EARLY"
       risk = "MEDIUM"
    elif severity < 50:
       stage = "MODERATE"
       risk = "HIGH"
    else:
       stage = "SEVERE"
       risk = "CRITICAL"
 return {
    "error": False,
    "disease": disease,
    "confidence": round(confidence, 4),
    "severity": round(severity, 2),
    "stage": stage,
    "future_risk": risk,
    "gradcam": "/static/gradcam_output.jpg"
 }
```

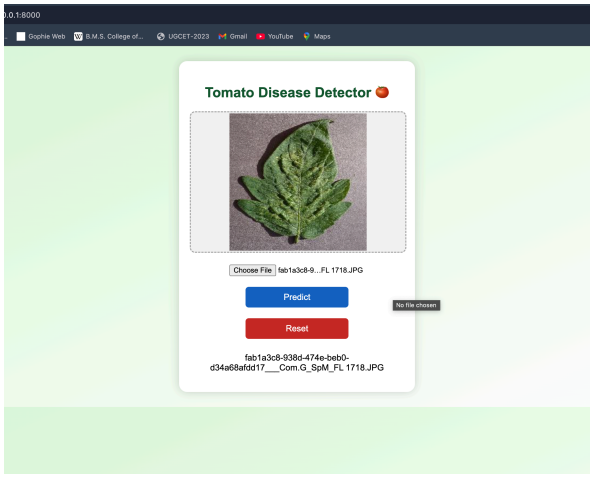
4) `index.html` – Simple Frontend

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Tomato Leaf Disease Early Prediction</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      max-width: 800px;
      margin: 0 auto;
      padding: 20px;
      background: #f5f5f5;
    }
    h1 {
      text-align: center;
    }
    .card {
      background: #fff;
      padding: 20px;
      margin-top: 20px;
      border-radius: 10px;
      box-shadow: 0 2px 8px rgba(0,0,0,0.1);
    }
    .result {
      margin-top: 20px;
    }
    img {
      max-width: 100%;
      border-radius: 10px;
      margin-top: 10px;
    }
    button {
      padding: 10px 20px;
      border: none;
      background: #2e7d32;
      color: #fff;
      border-radius: 5px;
      cursor: pointer;
    }
    button:disabled {
      background: #999;
    }
  </style>
</head>
<body>
  <h1>Tomato Leaf Disease Early Prediction 🌿</h1>
```

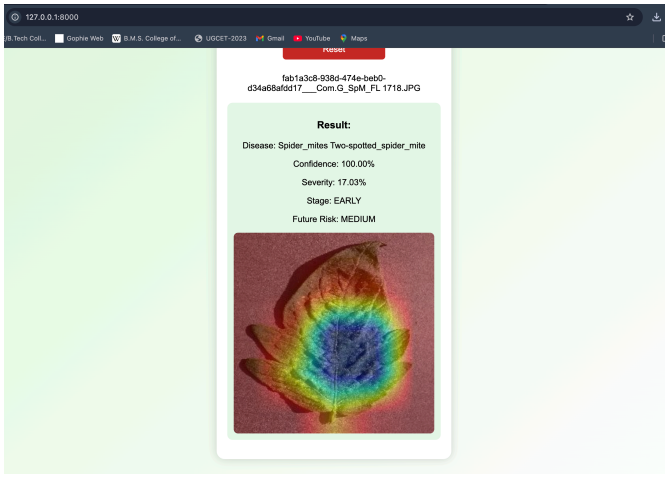
```
<div class="card">
  <form id="upload-form">
    <label>Select a tomato leaf image:</label><br><br>
    <input type="file" id="file-input" name="file" accept="image/*" required />
    <br><br>
    <button type="submit" id="submit-btn">Predict</button>
  </form>

  <div class="result" id="result"></div>
</div>

<script>
  const form = document.getElementById("upload-form");
  const resultDiv = document.getElementById("result");
  const submitBtn = document.getElementById("submit-btn");

  form.addEventListener("submit", async (e) => {
    e.preventDefault();

    const fileInput = document.getElementById("file-input");
    if (!fileInput.files[0]) {
      alert("Please select an image.");
      return;
    }

    const formData = new FormData();
    formData.append("file", fileInput.files[0]);

    submitBtn.disabled = true;
    submitBtn.textContent = "Predicting...";

    try {
      const res = await fetch("/predict", {
        method: "POST",
        body: formData
      });

      const data = await res.json();
      console.log(data);

      if (data.error) {
        resultDiv.innerHTML = `<p style="color:red;">${data.message}</p>`;
      } else {
        resultDiv.innerHTML = `
          <h3>Prediction Result</h3>
          <p><strong>Disease:</strong> ${data.disease}</p>
          <p><strong>Confidence:</strong> ${(data.confidence * 100).toFixed(2)}%</p>
          <p><strong>Severity:</strong> ${data.severity}%</p>
```

```
        <p><strong>Stage:</strong> ${data.stage}</p>
        <p><strong>Future Risk:</strong> ${data.future_risk}</p>
        <h4>Grad-CAM (Infected Region Highlighted)</h4>
        <img src="${data.gradcam}" alt="Grad-CAM"
      }

    } catch (err) {
      console.error(err);
      resultDiv.innerHTML = `<p style="color:red;">Error occurred. Check console.</p>`;
    } finally {
      submitBtn.disabled = false;
      submitBtn.textContent = "Predict";
    }
  });
 </script>
</body>
</html>
```

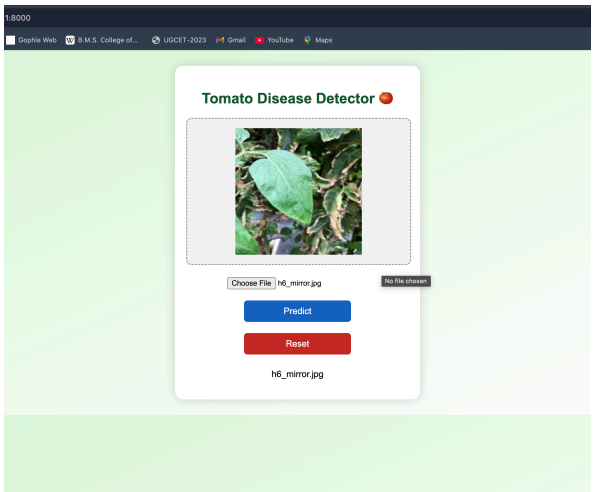
5. Output



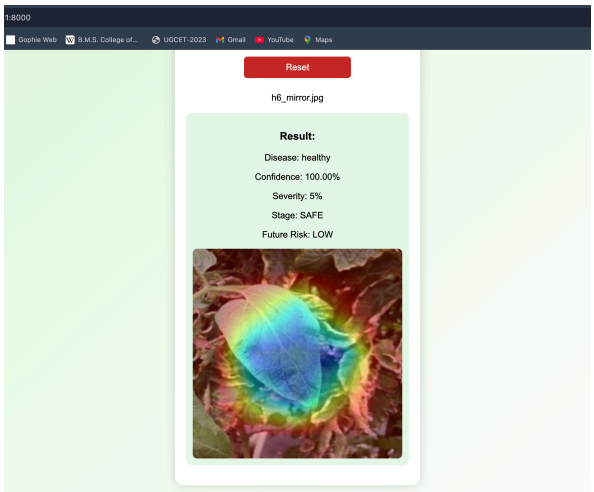
Input



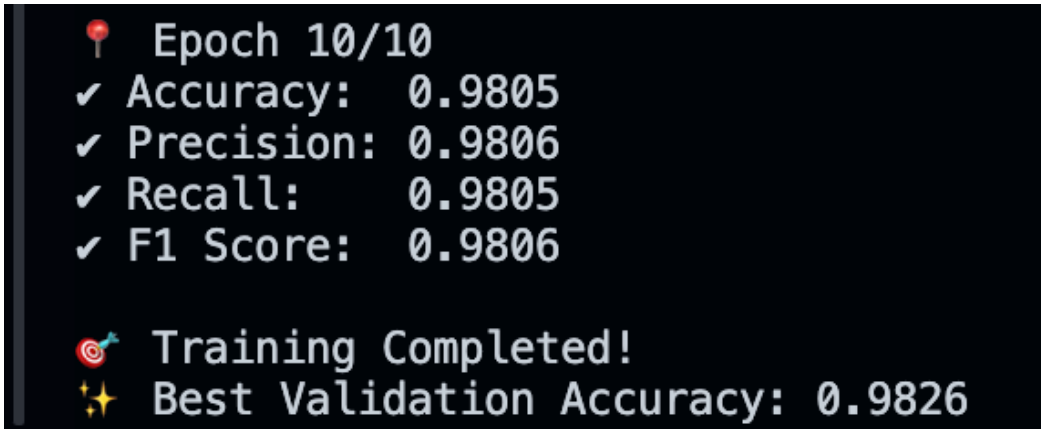
Output



Input



Output



Performance Measure

6. Conclusion

This project successfully demonstrates an efficient and practical approach for **early prediction of tomato leaf diseases** using **deep learning** and **computer vision**. By leveraging **MobileNetV2** with **transfer learning**, the model learns to accurately classify diseases from tomato leaf images even with a relatively small dataset. The integration of **Grad-CAM** visualization allows the system to highlight infected regions on the leaf, making the predictions interpretable and trustworthy.

Additionally, the introduction of **severity estimation** and **risk-level assessment** enables early-stage disease monitoring — providing valuable decision-making support to farmers and agricultural experts. The successful deployment of the model using **FastAPI** and a simple web-based interface makes the system accessible on real devices, demonstrating its potential for real-world field applications.

Overall, the project shows that the combination of lightweight CNN architecture, explainable AI, and severity analysis can significantly improve precision and early detection in agricultural disease management. With future improvements such as larger datasets, mobile integration, and multi-crop support, this system can become a powerful tool for **smart agriculture and crop protection**.