**📌 Creating a Dataset for Plant Disease Detection (Train & Test)**

To train your model, you need a dataset of **plant disease images**. If you're downloading images from online sources, follow these steps to **organize them into train and test sets**.

**🔹 Step 1: Download Images from Online Sources**

You can get plant disease images from:

1. **Public Datasets**:
   * PlantVillage Dataset (Kaggle)
   * AI Crowd - Plant Disease Detection
   * Google Images (using **Web Scraping**)
2. **Web Scraping (Optional)**  
   You can use Python’s selenium or bing-image-downloader to automate downloads:

python

CopyEdit

from bing\_image\_downloader import downloader

# Download 100 images for each disease

categories = ["Healthy plant", "Powdery Mildew", "Rust", "Leaf Spot"]

for category in categories:

downloader.download(category, limit=100, output\_dir="dataset", adult\_filter\_off=True, force\_replace=False)

**🔹 Step 2: Organizing the Dataset**

**📂 Folder Structure**

After downloading, organize your dataset into the following structure:

bash

CopyEdit

plant-disease-detection/

│── dataset/

│ ├── train/ # Training dataset (80% of total images)

│ │ ├── Healthy/ # Category-wise images

│ │ ├── Powdery Mildew/

│ │ ├── Rust/

│ │ ├── Leaf Spot/

│ │ ├── ... (other diseases)

│ │

│ ├── test/ # Testing dataset (20% of total images)

│ │ ├── Healthy/

│ │ ├── Powdery Mildew/

│ │ ├── Rust/

│ │ ├── Leaf Spot/

│ │ ├── ... (other diseases)

**🔹 Step 3: Split Data into Train & Test**

Once you have downloaded and organized the images, split them into **train (80%) and test (20%)**.

**📌 Automate Splitting with Python**

python

CopyEdit

import os

import shutil

import random

# Define dataset paths

DATASET\_DIR = "dataset"

TRAIN\_DIR = "dataset/train"

TEST\_DIR = "dataset/test"

# Create train and test folders

os.makedirs(TRAIN\_DIR, exist\_ok=True)

os.makedirs(TEST\_DIR, exist\_ok=True)

# Split ratio

SPLIT\_RATIO = 0.8

# Loop through each disease category

for category in os.listdir(DATASET\_DIR):

category\_path = os.path.join(DATASET\_DIR, category)

if not os.path.isdir(category\_path): # Skip non-folder files

continue

images = os.listdir(category\_path)

random.shuffle(images) # Shuffle images

# Split into train and test

train\_size = int(len(images) \* SPLIT\_RATIO)

train\_images = images[:train\_size]

test\_images = images[train\_size:]

# Create category folders in train and test

os.makedirs(os.path.join(TRAIN\_DIR, category), exist\_ok=True)

os.makedirs(os.path.join(TEST\_DIR, category), exist\_ok=True)

# Move images

for img in train\_images:

shutil.move(os.path.join(category\_path, img), os.path.join(TRAIN\_DIR, category, img))

for img in test\_images:

shutil.move(os.path.join(category\_path, img), os.path.join(TEST\_DIR, category, img))

print("✅ Dataset split into train & test successfully!")

**🔹 Step 4: Verify Your Dataset**

Once split, check your dataset folders to ensure images are properly categorized. Run this simple **Python script** to visualize **sample images**:

python

CopyEdit

import matplotlib.pyplot as plt

import os

import random

import cv2

SAMPLE\_DIR = "dataset/train/Powdery Mildew" # Change category

sample\_images = random.sample(os.listdir(SAMPLE\_DIR), 5)

plt.figure(figsize=(10, 5))

for i, img in enumerate(sample\_images):

img\_path = os.path.join(SAMPLE\_DIR, img)

img\_data = cv2.imread(img\_path)

img\_data = cv2.cvtColor(img\_data, cv2.COLOR\_BGR2RGB)

plt.subplot(1, 5, i+1)

plt.imshow(img\_data)

plt.axis("off")

plt.show()

**🔹 Step 5: Train Your Model Using the Dataset**

Once your dataset is ready, **run your 1.py training script**:

bash

CopyEdit

python 1.py

**🚀 Final Notes**

✅ **Download** images from online sources  
✅ **Organize** dataset into train/ & test/ folders  
✅ **Use Python** to automate splitting  
✅ **Verify** dataset with sample visualization  
✅ **Train** your model

Now, your model will be trained with a properly structured dataset! 🚀 Let me know if you need further help. 😊

**📌 1. Project Structure**

Your project directory should be structured as follows:

bash

CopyEdit

plant-disease-detection/

│── static/ # Static assets (CSS, JS, images)

│ ├── styles.css # Custom CSS for frontend (optional)

│

│── templates/ # HTML templates for Flask rendering

│ ├── index.html # Main homepage (upload image, about, contact)

│ ├── result.html # Displays prediction result

│

│── uploads/ # Stores uploaded images

│

│── dataset/ # Dataset (Train & Test images)

│ ├── train/ # Training images

│ │ ├── Healthy/ # Healthy plant images

│ │ ├── Blight/ # Blight disease images

│ │ ├── Rust/ # Rust disease images

│ │ ├── ... (More classes)

│ ├── test/ # Testing images (Same structure as train)

│

│── model/ # Trained ML models

│ ├── inceptionv3\_plant\_disease\_multi\_v2.h5 # Final trained model

│

│── app.py # Flask backend for web app

│── train\_model.py # Training script for deep learning model

│── disease\_info.json # JSON file with disease descriptions and cures

│── requirements.txt # Required Python libraries

│── README.md # Project Documentation

**📌 2. Steps to Build the Project**

**Step 1: Collecting and Preparing the Dataset**

**✅ Download Images**

* Download plant disease images from sources like:
  + **PlantVillage Dataset**
  + Google Images (use web scraping)
  + Kaggle Datasets

**✅ Organize Dataset**

* Arrange images into train and test folders:

bash

CopyEdit

dataset/

├── train/

│ ├── Healthy/

│ ├── Blight/

│ ├── Rust/

│ ├── ...

├── test/

│ ├── Healthy/

│ ├── Blight/

│ ├── Rust/

│ ├── ...

**✅ Resize Images to 299x299**

* Use **OpenCV** to resize all images:

python

CopyEdit

import os

import cv2

DATASET\_DIR = "dataset/train"

for category in os.listdir(DATASET\_DIR):

category\_path = os.path.join(DATASET\_DIR, category)

if not os.path.isdir(category\_path):

continue

for img\_name in os.listdir(category\_path):

img\_path = os.path.join(category\_path, img\_name)

img = cv2.imread(img\_path)

if img is not None:

img = cv2.resize(img, (299, 299))

cv2.imwrite(img\_path, img)

print("✅ Images resized successfully!")

**Step 2: Train the Deep Learning Model**

**✅ Create train\_model.py**

python

CopyEdit

from tensorflow.keras.applications import InceptionV3

from tensorflow.keras.models import Model

from tensorflow.keras.layers import GlobalAveragePooling2D, Dense, Dropout

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.optimizers import Adam

import os

# Load InceptionV3 without top layers

base\_model = InceptionV3(weights="imagenet", include\_top=False, input\_shape=(299, 299, 3))

# Add custom layers

x = base\_model.output

x = GlobalAveragePooling2D()(x)

x = Dense(2048, activation="relu")(x)

x = Dropout(0.4)(x)

x = Dense(1024, activation="relu")(x)

x = Dropout(0.4)(x)

output = Dense(30, activation="softmax")(x)

# Create model

model = Model(inputs=base\_model.input, outputs=output)

model.compile(optimizer=Adam(learning\_rate=0.00005), loss="categorical\_crossentropy", metrics=["accuracy"])

# Data Augmentation

datagen = ImageDataGenerator(rescale=1./255, rotation\_range=30, width\_shift\_range=0.2, height\_shift\_range=0.2,

shear\_range=0.2, zoom\_range=0.2, horizontal\_flip=True, validation\_split=0.2)

train\_generator = datagen.flow\_from\_directory("dataset/train", target\_size=(299, 299), batch\_size=32, class\_mode="categorical", subset="training")

val\_generator = datagen.flow\_from\_directory("dataset/train", target\_size=(299, 299), batch\_size=32, class\_mode="categorical", subset="validation")

# Train the model

model.fit(train\_generator, validation\_data=val\_generator, epochs=20)

# Save the model

model.save("model/inceptionv3\_plant\_disease\_multi\_v2.h5")

print("✅ Model trained and saved successfully!")

**Step 3: Build Flask Web App (app.py)**

**✅ Load Model in Flask**

python

CopyEdit

from flask import Flask, render\_template, request, jsonify

import tensorflow as tf

import numpy as np

from tensorflow.keras.preprocessing import image

import os

import json

app = Flask(\_\_name\_\_)

# Load model

model = tf.keras.models.load\_model("model/inceptionv3\_plant\_disease\_multi\_v2.h5")

# Class labels

class\_labels = ["Healthy", "Powdery Mildew", "Rust", "Leaf Spot", "Blight", "Bacterial Wilt", ...]

# Load disease info

with open("disease\_info.json", "r") as f:

disease\_info = json.load(f)

# Preprocess Image

def preprocess\_image(img\_path):

img = image.load\_img(img\_path, target\_size=(299, 299))

img\_array = image.img\_to\_array(img)

img\_array = np.expand\_dims(img\_array, axis=0) / 255.0

return img\_array

# Home Route

@app.route('/')

def home():

return render\_template('index.html')

# Prediction Route

@app.route('/predict', methods=['POST'])

def predict():

file = request.files['file']

file\_path = os.path.join("uploads", file.filename)

file.save(file\_path)

img\_array = preprocess\_image(file\_path)

prediction = model.predict(img\_array)

predicted\_index = np.argmax(prediction)

predicted\_class = class\_labels[predicted\_index]

disease\_details = disease\_info.get(predicted\_class, {"cure": "No data", "growth\_tips": "No data"})

return render\_template('result.html', prediction=predicted\_class, confidence=f"{np.max(prediction) \* 100:.2f}%",

cure=disease\_details["cure"], growth\_tips=disease\_details["growth\_tips"])

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

**Step 4: Create Frontend (HTML + CSS)**

**✅ index.html**

html

CopyEdit

<!DOCTYPE html>

<html>

<head>

<title>Plant Disease Detector</title>

</head>

<body>

<h1>Upload an Image</h1>

<form action="/predict" method="post" enctype="multipart/form-data">

<input type="file" name="file">

<button type="submit">Predict</button>

</form>

</body>

</html>

**Step 5: Run the Project**

**✅ 1. Train the model**

bash

CopyEdit

python train\_model.py

**✅ 2. Run the Flask App**

bash

CopyEdit

python app.py

**✅ 3. Open in Browser**

Go to **http://127.0.0.1:5000/**

**🎯 Final Output**

* User uploads an image ✅
* Model predicts the disease ✅
* Displays cure & growth tips ✅

🚀 **Your Plant Disease Detection Project is Complete!** 🎉