Dual-Branch Plant Disease Classification (Allen + Mufti)

This notebook builds a **dual-branch CNN** that fuses a custom convolutional pipeline with a pretrained ResNet34. It trains on two datasets:

- **Allen**: 3 classes (Healthy, Powdery, Rust), 1,530 images
- **Mufti**: 38 classes across 14 plants, ~70K train / ~17K test

Features:

1. Albumentations augmentations
2. Mixed-precision training (AMP)
3. Early stopping & LR scheduling
4. Logging & reproducibility
5. Confusion matrix & classification report
6. Gradio demo for live inference

---

1. Setup & Imports

Seed everything for reproducibility, set up logging, and import all required libraries.

```
pip install gradio
```

```
import os, random, logging
from pathlib import Path

import numpy as np
import pandas as pd
from PIL import Image

import torch import torch.nn as nn import
torch.nn.functional as F from torch.utils.data import
Dataset, DataLoader from torch.optim import AdamW from
torch.optim.lr_scheduler import ReduceLROnPlateau

import albumentations as A from
albumentations.pytorch import ToTensorV2

import torchvision from torchvision.models import * from
sklearn.metrics import confusion_matrix, classification_report
import matplotlib.pyplot as plt import seaborn as sns from
tqdm.auto import tqdm import gradio as gr

# Reproducibility def
seed_everything(seed=42):
    random.seed(seed)
os.environ["PYTHONHASHSEED"] = str(seed)
np.random.seed(seed)
torch.manual_seed(seed)
torch.cuda.manual_seed_all(seed)
torch.backends.cudnn.deterministic = True
torch.backends.cudnn.benchmark = True


seed_everything(2025)

# Device device = torch.device("cuda" if torch.cuda.is_available()
else "cpu") print("Using device:", device)

# Logging logger = logging.getLogger("PlantDisease")
logger.setLevel(logging.INFO) ch = logging.StreamHandler()
ch.setFormatter(logging.Formatter("[%(asctime)s][%(levelname)s] %(message)s"))
logger.addHandler(ch) fh = logging.FileHandler('plantdisease.log')
fh.setFormatter(logging.Formatter("[%(asctime)s][%(levelname)s] %(message)s"))
logger.addHandler(fh)

import warnings
warnings.filterwarnings('ignore')
```

```
Using device: cuda
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

2. Configuration

Define all paths and hyperparameters in a single `Config` class for easy adjustment.

```
class Config:     # Dataset roots
drive_root = Path("/content/drive/MyDrive")

    # Allen dataset has train/validation/test subfolders
allen_train_dir = drive_root / "PDC/Allen/Train/Train"
allen_val_dir   = drive_root / "PDC/Allen/Validation/Validation"
allen_test_dir  = drive_root / "PDC/Allen/Test/Test"

    # Mufti dataset has an “Augmented” folder with train/valid, plus a separate test folder
mufti_aug_dir   = drive_root / "PDC/Mufti/New Plant Diseases Dataset(Augmented)"
mufti_train_dir = mufti_aug_dir / "New Plant Diseases Dataset(Augmented)/train"
mufti_val_dir   = mufti_aug_dir / "New Plant Diseases Dataset(Augmented)/valid"
mufti_test_dir  = drive_root / "PDC/Mufti/test"

    # Image size & batch
img_size = 64
batch_size = 16

    # Training hyperparameters     epochs = 20     lr
= 3e-4     weight_decay = 1e-4     patience = 3  #
for LR scheduler & early stopping cfg = Config()
```

```
logger.info(f"Allen dataset path: {cfg.allen_train_dir}")
logger.info(f"Allen train dir exists: {cfg.allen_train_dir.exists()}")
logger.info(f"Allen val dir exists:   {cfg.allen_val_dir.exists()}")
logger.info(f"Allen test dir exists:  {cfg.allen_test_dir.exists()}")
```

```
[2025-04-14 13:53:46,581][INFO] Allen dataset path: /content/drive/MyDrive/PDC/Allen/Train/Train
 INFO:PlantDisease:Allen dataset path: /content/drive/MyDrive/PDC/Allen/Train/Train
 [2025-04-14 13:53:49,203][INFO] Allen train dir exists: True
 INFO:PlantDisease:Allen train dir exists: True
 [2025-04-14 13:53:49,205][INFO] Allen val dir exists:   True
 INFO:PlantDisease:Allen val dir exists:   True
 [2025-04-14 13:53:49,209][INFO] Allen test dir exists:  True
 INFO:PlantDisease:Allen test dir exists:  True
```

```
logger.info(f"Mufti dataset path: {cfg.mufti_train_dir}")
logger.info(f"Mufti train dir exists: {cfg.mufti_train_dir.exists()}")
logger.info(f"Mufti val dir exists:   {cfg.mufti_val_dir.exists()}")
logger.info(f"Mufti test dir exists:  {cfg.mufti_test_dir.exists()}")
```

```
[2025-04-14 13:53:49,214][INFO] Mufti dataset path: /content/drive/MyDrive/PDC/Mufti/New Plant Diseases Dataset(Augmented)/New Plant
 INFO:PlantDisease:Mufti dataset path: /content/drive/MyDrive/PDC/Mufti/New Plant Diseases Dataset(Augmented)/New Plant Diseases Data
 [2025-04-14 13:53:50,020][INFO] Mufti train dir exists: True
 INFO:PlantDisease:Mufti train dir exists: True
 [2025-04-14 13:53:50,022][INFO] Mufti val dir exists:   True
 INFO:PlantDisease:Mufti val dir exists:   True
 [2025-04-14 13:53:50,023][INFO] Mufti test dir exists:  True
 INFO:PlantDisease:Mufti test dir exists:  True
```

3. Build Combined DataFrame

Scan both Allen and Mufti folders, prefix labels so they don’t collide, and split into train/val/test.

```
def build_df(root: Path, prefix: str=""):
    """Return DataFrame with columns [filepath, label]."""
rows = []     if not root.exists():
        logger.warning(f"Path does not exist: {root}")
return pd.DataFrame(rows)

    for cls in sorted(os.listdir(root)):
        cls_path = root / cls
if not cls_path.is_dir():
            continue
        # Print the class path to check which folders it's iterating through
print(f"Checking class path: {cls_path}")         for img in
cls_path.glob("*.[jJpP][pPnNgG]*"):
            # Print the images found             print(f"Found image:
{img}")             rows.append({"filepath": str(img), "label": prefix
+ cls})
```

```
    # Print the number of rows added
print(f"Number of rows added: {len(rows)}")
return pd.DataFrame(rows)

# Allen dataset (3 classes) allen_train_df =
build_df(cfg.allen_train_dir, prefix="Allen_") allen_val_df   =
build_df(cfg.allen_val_dir,   prefix="Allen_") allen_test_df  =
build_df(cfg.allen_test_dir,  prefix="Allen_")

# Mufti dataset (38 classes) mufti_train_df =
build_df(cfg.mufti_train_dir, prefix="Mufti_") mufti_val_df   =
build_df(cfg.mufti_val_dir,   prefix="Mufti_") mufti_test_df  =
build_df(cfg.mufti_test_dir,  prefix="Mufti_")

# Combine datasets train_df = pd.concat([allen_train_df, mufti_train_df],
ignore_index=True).sample(frac=1, random_state=42) val_df   = pd.concat([allen_val_df,   mufti_val_df],
ignore_index=True).sample(frac=1, random_state=42) test_df  = pd.concat([allen_test_df,  mufti_test_df],
ignore_index=True).sample(frac=1, random_state=42)

# Sanity check: ensure DataFrames are not empty if
train_df.empty or val_df.empty or test_df.empty:
    logger.warning("One or more DataFrames are empty. Check your data paths and file formats.")
logger.warning(f"Allen train dir: {cfg.allen_train_dir} (exists={cfg.allen_train_dir.exists()})")
logger.warning(f"Allen val dir:   {cfg.allen_val_dir}   (exists={cfg.allen_val_dir.exists()})")
logger.warning(f"Allen test dir:  {cfg.allen_test_dir}  (exists={cfg.allen_test_dir.exists()})")
logger.warning(f"Mufti train dir: {cfg.mufti_train_dir} (exists={cfg.mufti_train_dir.exists()})")
logger.warning(f"Mufti val dir:   {cfg.mufti_val_dir}   (exists={cfg.mufti_val_dir.exists()})")
logger.warning(f"Mufti test dir:  {cfg.mufti_test_dir}  (exists={cfg.mufti_test_dir.exists()})")

# Encode labels: create mappings for consistency all_df =
pd.concat([train_df, val_df, test_df], ignore_index=True) labels =
sorted(all_df.label.unique()) label2idx = {l: i for i, l in
enumerate(labels)} idx2label = {i: l for l, i in
label2idx.items()}

# Map labels to indices for df_ in (train_df,
val_df, test_df):     df_["label_idx"] =
df_.label.map(label2idx)

# Log final dataset sizes
logger.info(f"Final splits ▶ Train: {len(train_df)}, Val: {len(val_df)}, Test: {len(test_df)}")
```

Show hidden output

```
logger.info(f"Train class distribution: \n{train_df['label'].value_counts()}")
logger.info(f"Val class distribution: \n{val_df['label'].value_counts()}")
logger.info(f"Test class distribution: \n{test_df['label'].value_counts()}")
```

```
     Mufti Corn (maize)   Common rust                     477
```

```
Mufti_Corn_(maize)___Common_rust_                          477
Mufti_Grape___Black_rot                                    472
Mufti_Tomato___Leaf_Mold                                   470
Mufti_Corn_(maize)___healthy                               465
Mufti_Tomato___Late_blight                                 463
Mufti_Peach___Bacterial_spot                               459
Mufti_Tomato___Target_Spot                                 457
Mufti_Strawberry___healthy                                 456
Mufti_Cherry_(including_sour)___healthy                    456
Mufti_Potato___healthy                                     456
Mufti_Blueberry___healthy                                  454
Mufti_Tomato___Tomato_mosaic_virus                         448
Mufti_Raspberry___healthy                                  445
Mufti_Strawberry___Leaf_scorch                             444
Mufti_Apple___Cedar_apple_rust                             440
Mufti_Grape___Leaf_blight_(Isariopsis_Leaf_Spot)           440
Mufti_Tomato___Septoria_leaf_spot                          436
Mufti_Tomato___Spider_mites Two-spotted_spider_mite        435
Mufti_Squash___Powdery_mildew                              434
Mufti_Grape___healthy                                      433
Mufti_Peach___healthy                                      432
Mufti_Tomato___Bacterial_spot                              425
Mufti_Cherry_(including_sour)___Powdery_mildew             421
Mufti_Corn_(maize)___Cercospora_leaf_spot Gray_leaf_spot   410
Allen_Rust                                                  20
Allen_Healthy                                               20
Allen_Powdery                                               20
Name: count, dtype: int64
[2025-04-14 13:55:22,327][INFO] Test class distribution:
label
Allen_Healthy    50
Allen_Rust       50
Allen_Powdery    50
Mufti_test       33
Name: count, dtype: int64
INFO:PlantDisease:Test class distribution:
label
Allen_Healthy    50
Allen_Rust       50
Allen_Powdery    50
Mufti_test       33
Name: count, dtype: int64
```

4. Dataset & DataLoader

Define a custom `Dataset` that applies Albumentations transforms. Then create PyTorch `DataLoader` s.

```
class PlantDataset(Dataset):
    def __init__(self, df, img_size, transforms=None):
        self.df = df.reset_index(drop=True)
        self.img_size = img_size
        self.transforms = transforms

    def __len__(self):
        return len(self.df)

    def __getitem__(self, idx):
        try:
            row = self.df.loc[idx]
            img_path = row.filepath
            # Log the image path to check for issues
            # print(f"Loading image: {img_path}")
            img = np.array(Image.open(img_path).convert("RGB"))
            if self.transforms:
                img = self.transforms(image=img)["image"]
            # Log the shape of the image after transformation
            # print(f"Image shape after transform: {img.shape}")
            return img, row.label_idx
        except Exception as e:
            # print(f"Error loading image {row.filepath}: {e}")
            # You could return a default image or raise the exception
            raise e

# Albumentations transforms
train_tfms = A.Compose([
    A.RandomResizedCrop(size=(cfg.img_size, cfg.img_size), scale=(0.8,1.0)),  # Use size=(height, width)
    A.HorizontalFlip(0.5), A.VerticalFlip(0.5),
    A.ColorJitter(0.2,0.2,0.2,0.1),
    A.Normalize(), ToTensorV2(),
])
val_tfms = A.Compose([A.Resize(cfg.img_size, cfg.img_size), A.Normalize(), ToTensorV2()])

# Datasets
train_ds = PlantDataset(train_df, cfg.img_size, transforms=train_tfms)
val_ds   = PlantDataset(val_df,   cfg.img_size, transforms=val_tfms)
```

```
test_ds  = PlantDataset(test_df,  cfg.img_size, transforms=val_tfms)

# DataLoaders train_loader = DataLoader(train_ds, batch_size=cfg.batch_size, shuffle=True,  num_workers=4,
pin_memory=True) val_loader   = DataLoader(val_ds,   batch_size=cfg.batch_size, shuffle=False, num_workers=4,
pin_memory=True) test_loader  = DataLoader(test_ds,  batch_size=cfg.batch_size, shuffle=False, num_workers=4,
pin_memory=True)
```

5. Dual-Branch Model Definition

Branch A: deep custom CNN

Branch B: frozen ResNet34 feature extractor Fusion:

concatenate features → classifier

```
import torch import torch.nn as nn
import torch.nn.functional as F
import torchvision.models as models


class DualBranchModel(nn.Module):     def __init__(self, n_classes):         super().__init__()
# Branch A: custom CNN         self.branch_a = nn.Sequential(
nn.Conv2d(3,32,3,padding=1), nn.ReLU(), nn.Conv2d(32,32,3,padding=1), nn.ReLU(),
nn.MaxPool2d(2),             nn.Conv2d(32,64,3,padding=1), nn.ReLU(),
nn.Conv2d(64,64,3,padding=1), nn.ReLU(),             nn.MaxPool2d(2),
nn.Conv2d(64,128,3,padding=1), nn.ReLU(), nn.Conv2d(128,128,3,padding=1), nn.ReLU(),
nn.AdaptiveAvgPool2d(1), nn.Flatten(),             nn.Linear(128,128), nn.ReLU()
        )         # Branch B: pretrained ResNet34
backbone = models.resnet34(pretrained=True)         for p in
backbone.parameters(): p.requires_grad=False
self.branch_b = nn.Sequential(
*list(backbone.children())[:-1],             nn.Flatten(),
nn.Linear(backbone.fc.in_features, 128),
nn.ReLU()         )         # Fusion & classifier
self.classifier = nn.Sequential(             nn.Dropout(0.5),
nn.Linear(256,256), nn.ReLU(),             nn.Dropout(0.3),
nn.Linear(256,n_classes)
        )

    def forward(self, x):         a =
self.branch_a(x)         b =
self.branch_b(x)         fused =
torch.cat([a,b], dim=1)         return
self.classifier(fused)

model = DualBranchModel(len(labels)).to(device)
logger.info(model)
```

```
          (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
```

```
        )
      )
      (5): Sequential(
        (0): BasicBlock(         (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2),
  padding=(1, 1), bias=False)         (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1,
  affine=True, track_running_stats=True)
          (relu): ReLU(inplace=True)
          (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          (downsample): Sequential(
            (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
            (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
        )
        (1): BasicBlock(         (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1),
  padding=(1, 1), bias=False)         (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1,
  affine=True, track_running_stats=True)
          (relu): ReLU(inplace=True)
          (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        )
        (2): BasicBlock(         (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1),
  padding=(1, 1), bias=False)         (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1,
  affine=True, track_running_stats=True)
          (relu): ReLU(inplace=True)
          (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        )
        (3): BasicBlock(         (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1),
  padding=(1, 1), bias=False)         (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1,
  affine=True, track_running_stats=True)
          (relu): ReLU(inplace=True)
          (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        )
```

6. Loss, Optimizer, Scheduler & AMP

Use label-smoothed cross-entropy, AdamW, ReduceLROnPlateau, and mixed precision.

```
criterion = nn.CrossEntropyLoss(label_smoothing=0.1) optimizer = AdamW(model.parameters(),
lr=cfg.lr, weight_decay=cfg.weight_decay) scheduler = ReduceLROnPlateau(optimizer, mode="min",
patience=cfg.patience, factor=0.5, verbose=True) scaler = torch.cuda.amp.GradScaler()
```

7. Training Loop with Early Stopping

Train for up to `cfg.epochs` , save the best model, and stop early if no improvement.

```
best_val_loss = float("inf") early_stop = 0 history =

{"train_loss":[], "train_acc":[], "val_loss":[], "val_acc":[]}

for epoch in range(1, cfg.epochs+1):
    # — Training     model.train()     tloss, tcorrect = 0, 0     for imgs,
lbls in tqdm(train_loader, desc=f"Train {epoch}/{cfg.epochs}"):
        imgs, lbls = imgs.to(device),
lbls.to(device)         optimizer.zero_grad()
with torch.cuda.amp.autocast():             out =
model(imgs)             loss = criterion(out, lbls)
scaler.scale(loss).backward()
scaler.step(optimizer)         scaler.update()

        preds = out.argmax(1)         tloss
+= loss.item() * imgs.size(0)
tcorrect += (preds==lbls).sum().item()

    # — Validation     model.eval()     vloss, vcorrect = 0, 0     with
torch.no_grad():         for imgs, lbls in tqdm(val_loader, desc=f" Val
{epoch}/{cfg.epochs}"):
            imgs, lbls = imgs.to(device),
lbls.to(device)             out = model(imgs)
loss = criterion(out, lbls)             preds =
out.argmax(1)             vloss += loss.item() *
imgs.size(0)
            vcorrect += (preds==lbls).sum().item()

    # — Metrics     train_loss =
tloss/len(train_ds)     train_acc  =
tcorrect/len(train_ds)     val_loss   =
vloss/len(val_ds)     val_acc    =
vcorrect/len(val_ds)

    history["train_loss"].append(train_loss)
history["train_acc"].append(train_acc)
history["val_loss"].append(val_loss)
history["val_acc"].append(val_acc)

    logger.info(f"Epoch {epoch} ▶ Train: {train_loss:.4f}/{train_acc:.4f} | Val: {val_loss:.4f}/{val_acc:.4f}")
scheduler.step(val_loss)

    # Early stopping     if val_loss < best_val_loss:
best_val_loss = val_loss
torch.save(model.state_dict(), "best_model.pth")
early_stop = 0     else:
        early_stop += 1         if
early_stop >= cfg.patience:
            logger.info("Early stopping triggered.")
break
```

Plot Training Curves

```
plt.figure(figsize=(12,4)) plt.subplot(1,2,1)
plt.plot(history["train_loss"], '-o', label="Train Loss")
plt.plot(history["val_loss"],   '-o', label="Val Loss")
plt.title("Loss"); plt.legend()

plt.subplot(1,2,2) plt.plot(history["train_acc"], '-o',
label="Train Acc") plt.plot(history["val_acc"],   '-o',
label="Val Acc") plt.title("Accuracy"); plt.legend()
plt.show()
```

8. Test Set Evaluation

Load the best model, compute a confusion matrix and classification report.

```
# Load best weights
model.load_state_dict(torch.load("best_model.pth"))
model.eval()

y_true, y_pred = [], [] with torch.no_grad():     for
imgs, lbls in tqdm(test_loader, desc="Testing"):
        imgs = imgs.to(device)         out
= model(imgs)         preds =
out.argmax(1).cpu().numpy()
y_pred.extend(preds)
y_true.extend(lbls.numpy())
# Confusion matrix cm =
confusion_matrix(y_true, y_pred)
plt.figure(figsize=(10,8)) sns.heatmap(cm,
cmap="Blues", fmt="d") plt.title("Test
Confusion Matrix") plt.ylabel("True");
plt.xlabel("Predicted") plt.show()

# Classification report
print(classification_report(y_true, y_pred, target_names=labels))
```

10. Gradio Demo

Launch a simple web interface so judges can upload images and see top-3 predictions live.

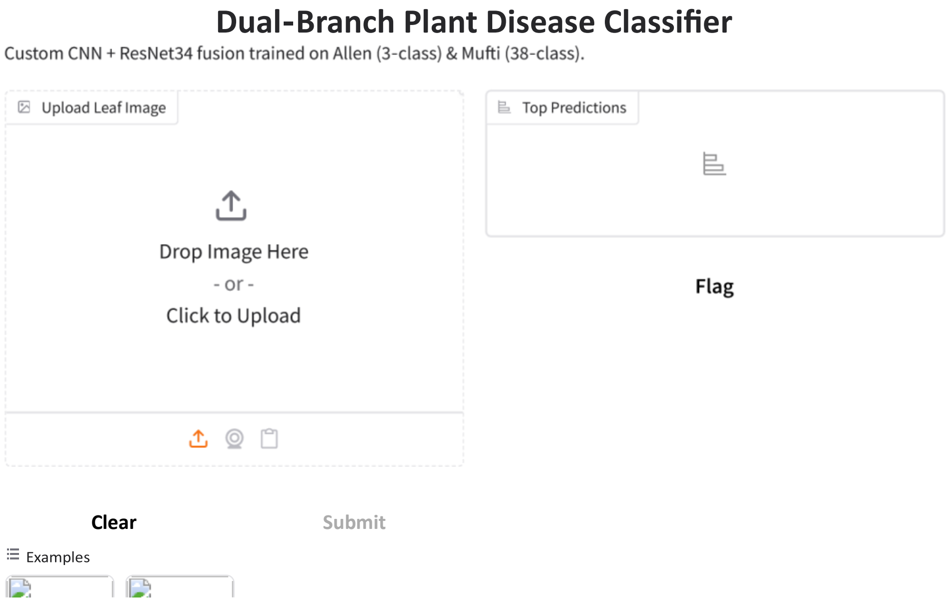
```
def—predict(img:—np.ndarray):
———img_t—=—val_tfms(image=img)["image"].unsqueeze(0).to(device)
with—torch.no_grad():
————————logits—=—model(img_t) ————————probs—=—F.softmax(logits,
dim=1)[0].cpu().numpy() ————return—{—idx2label[i]:—float(probs[i])—for—i
in—range(len(labels))—}

demo—=—gr.Interface( ————fn=predict, ————inputs=gr.Image(type="numpy",—label="Upload—Leaf
Image"), ————outputs=gr.Label(num_top_classes=3,—label="Top—Predictions"),     title="Dual-
Branch Plant Disease Classifier",     description="Custom CNN + ResNet34 fusion trained on
Allen (3-class) & Mufti (38-class).",     examples=[["example1.jpg"], ["example2.jpg"]]
)
demo.launch()
```

```
Running Gradio in a Colab notebook requires sharing enabled. Automatically setting `share=True` (you can turn this off by setting `s

Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
* Running on public URL: https://1f02f169bcce485b23.gradio.live

This share link expires in 72 hours. For free permanent hosting and GPU upgrades, run `gradio deploy` from the terminal in the worki
```



Dual-Branch Plant Disease Classifier

Custom CNN + ResNet34 fusion trained on Allen (3-class) & Mufti (38-class).