In [ ]:
```
import kagglehub

# Download latest version
path = kagglehub.dataset_download("sohansakib75/cotton-kaggle")

print("Path to dataset files:", path)
```

```
Path to dataset files: /kaggle/input/cotton-kaggle
```

In [ ]:
```
import os
import kagglehub

# Download dataset
path = kagglehub.dataset_download("sohansakib75/cotton-kaggle")

print("Dataset root path:", path)

# List folders and files inside
print("Contents inside dataset folder:")
for item in os.listdir(path):
    item_path = os.path.join(path, item)
    if os.path.isdir(item_path):
        print(f" {item}/")
    else:
        print(f" {item}")
```

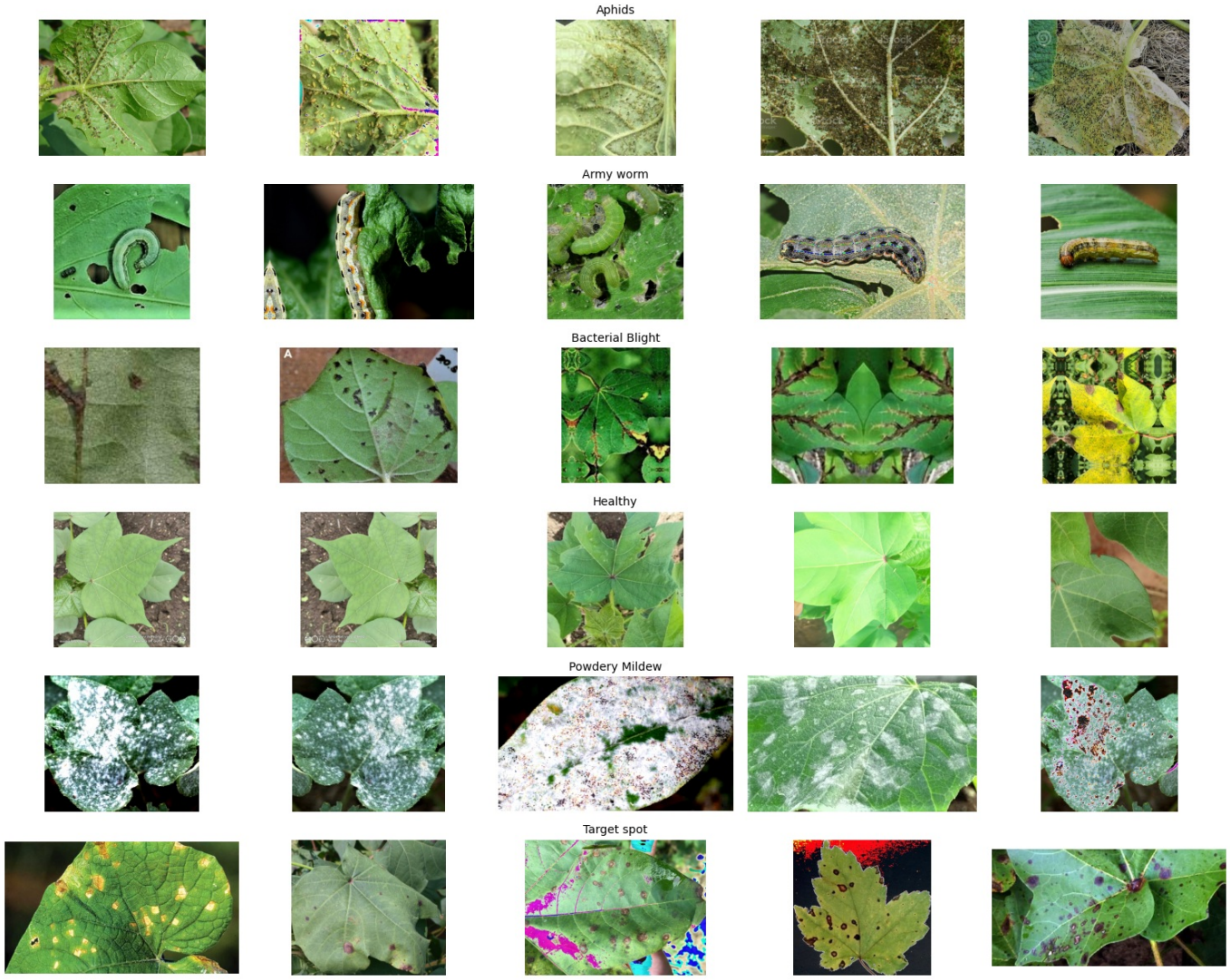
```
Dataset root path: /kaggle/input/cotton-kaggle
Contents inside dataset folder:
 Dataset/
```

In [ ]:
```
import os
import glob

dataset_path = os.path.join(path, "Dataset")

print("Path to Dataset folder:", dataset_path)


for subdir in sorted(os.listdir(dataset_path)):
    subpath = os.path.join(dataset_path, subdir)
    if os.path.isdir(subpath):
        # Count images by common formats
        image_files = glob.glob(os.path.join(subpath, "*.jpg")) + \
                      glob.glob(os.path.join(subpath, "*.jpeg")) + \
                      glob.glob(os.path.join(subpath, "*.png"))
        print(f"{subdir}: {len(image_files)} images")
```

```
Path to Dataset folder: /kaggle/input/cotton-kaggle/Dataset
Aphids: 400 images
Army worm: 400 images
Bacterial Blight: 400 images
Healthy: 400 images
Powdery Mildew: 400 images
Target spot: 400 images
```

In [ ]:
```
import os
import glob
import matplotlib.pyplot as plt
import random

dataset_path = "/kaggle/input/cotton-kaggle/Dataset"

# Classes
classes = sorted(os.listdir(dataset_path))

# Plot 5 images per class
fig, axes = plt.subplots(len(classes), 5, figsize=(15, 12))

for i, cls in enumerate(classes):
    cls_path = os.path.join(dataset_path, cls)
    image_files = glob.glob(os.path.join(cls_path, "*.jpg")) + \
                  glob.glob(os.path.join(cls_path, "*.jpeg")) + \
                  glob.glob(os.path.join(cls_path, "*.png"))

    # Randomly pick 5 images
    sample_files = random.sample(image_files, 5)

    for j, img_path in enumerate(sample_files):
        img = plt.imread(img_path)
        axes[i, j].imshow(img)
        axes[i, j].axis("off")
        if j == 2:  # center column
            axes[i, j].set_title(cls, fontsize=10)
```

```
plt.tight_layout()
plt.show()
```



In [ ]:

```
import os
import cv2
import numpy as np
import matplotlib.pyplot as plt
import random

# Paths
input_dir = "/kaggle/input/cotton-kaggle/Dataset"
output_dir = "/kaggle/working/preprocessed_dataset"
os.makedirs(output_dir, exist_ok=True)

img_size = (224, 224)
num_samples = 5

classes = sorted(os.listdir(input_dir))
fig, axes = plt.subplots(len(classes), num_samples, figsize=(15, 3*len(classes)))


clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))

for i, cls in enumerate(classes):
    cls_path = os.path.join(input_dir, cls)
    if os.path.isdir(cls_path):
        save_cls_path = os.path.join(output_dir, cls)
        os.makedirs(save_cls_path, exist_ok=True)

        files = [f for f in os.listdir(cls_path) if f.lower().endswith((".jpg", ".jpeg", ".png"))]
        sample_files = random.sample(files, min(num_samples, len(files)))

        for j, file in enumerate(files):
            img_path = os.path.join(cls_path, file)
            img = cv2.imread(img_path)

            # 1️ Resize
            img_resized = cv2.resize(img, img_size)

            # 2️ CLAHE on Y channel
```

```
            yuv = cv2.cvtColor(img_resized, cv2.COLOR_BGR2YUV)
            yuv[:,:,0] = clahe.apply(yuv[:,:,0])
            img_clahe = cv2.cvtColor(yuv, cv2.COLOR_YUV2BGR)

            # 3️ Non-Local Means Denoising
            img_denoised = cv2.fastNlMeansDenoisingColored(img_clahe, None, h=10, hColor=10, templateWindowSize=

            # 4️ Normalize to [0,1]
            img_final = img_denoised.astype(np.float32) / 255.0

            # Save preprocessed image
            save_img = (img_final * 255).astype(np.uint8)
            cv2.imwrite(os.path.join(save_cls_path, file), save_img)

            # Plot sample images
            if file in sample_files:
                axes[i, sample_files.index(file)].imshow(cv2.cvtColor(save_img, cv2.COLOR_BGR2RGB))
                axes[i, sample_files.index(file)].axis("off")
                if sample_files.index(file) == num_samples // 2:
                    axes[i, sample_files.index(file)].set_title(cls, fontsize=12)

plt.tight_layout()
plt.show()
print("\n Preprocessed images saved in:", output_dir)
```


```
Preprocessed images saved in: /kaggle/working/preprocessed_dataset
```

In [ ]:

```python
import os
import shutil
import random

# Preprocessed dataset path
preprocessed_dir = "/kaggle/working/preprocessed_dataset"

# Split paths
split_base = "/kaggle/working/cotton_split"
train_dir = os.path.join(split_base, "train")
val_dir   = os.path.join(split_base, "val")
test_dir  = os.path.join(split_base, "test")

# Create split folders
for d in [train_dir, val_dir, test_dir]:
    os.makedirs(d, exist_ok=True)

# Split ratios
```

```
train_ratio = 0.8
val_ratio = 0.1
test_ratio = 0.1

classes = sorted(os.listdir(preprocessed_dir))

for cls in classes:
    cls_path = os.path.join(preprocessed_dir, cls)
    files = [f for f in os.listdir(cls_path) if f.lower().endswith((".jpg", ".jpeg", ".png"))]
    random.shuffle(files)

    n_total = len(files)
    n_train = int(train_ratio * n_total)
    n_val = int(val_ratio * n_total)
    n_test = n_total - n_train - n_val

    splits = {
        train_dir: files[:n_train],
        val_dir: files[n_train:n_train+n_val],
        test_dir: files[n_train+n_val:]
    }

    for split_folder, split_files in splits.items():
        cls_split_path = os.path.join(split_folder, cls)
        os.makedirs(cls_split_path, exist_ok=True)
        for f in split_files:
            shutil.copy(os.path.join(cls_path, f), os.path.join(cls_split_path, f))

print(" Dataset split into 80-10-10 and saved in:", split_base)
```

```
Dataset split into 80-10-10 and saved in: /kaggle/working/cotton_split
```

In [ ]:
```
import os

split_base = "/kaggle/working/cotton_split"
splits = ["train", "val", "test"]

for split in splits:
    split_path = os.path.join(split_base, split)
    print(f"\n {split.capitalize()} Split:")
    for cls in sorted(os.listdir(split_path)):
        cls_path = os.path.join(split_path, cls)
        num_images = len([f for f in os.listdir(cls_path) if f.lower().endswith((".jpg", ".jpeg", ".png"))])
        print(f"{cls}: {num_images} images")
```

```
 Train Split:
Aphids: 320 images
Army worm: 320 images
Bacterial Blight: 320 images
Healthy: 320 images
Powdery Mildew: 320 images
Target spot: 320 images

 Val Split:
Aphids: 40 images
Army worm: 40 images
Bacterial Blight: 40 images
Healthy: 40 images
Powdery Mildew: 40 images
Target spot: 40 images

 Test Split:
Aphids: 40 images
Army worm: 40 images
Bacterial Blight: 40 images
Healthy: 40 images
Powdery Mildew: 40 images
Target spot: 40 images
```

In [ ]:
```
import os
import cv2
import numpy as np
import random

train_dir = "/kaggle/working/cotton_split/train"
aug_train_dir = "/kaggle/working/cotton_train_aug"
os.makedirs(aug_train_dir, exist_ok=True)

# Augmentation functions
def random_flip(img):
    flip_code = random.choice([-1, 0, 1])
    return cv2.flip(img, flip_code)

def random_rotate(img):
```

```
    angle = random.uniform(-25, 25)
    h, w = img.shape[:2]
    M = cv2.getRotationMatrix2D((w//2, h//2), angle, 1)
    return cv2.warpAffine(img, M, (w, h), borderMode=cv2.BORDER_REFLECT)

def random_zoom(img):
    zoom_factor = random.uniform(0.8, 1.2)
    h, w = img.shape[:2]
    new_h, new_w = int(h*zoom_factor), int(w*zoom_factor)
    img_resized = cv2.resize(img, (new_w, new_h))
    if zoom_factor < 1:
        pad_h = (h - new_h) // 2
        pad_w = (w - new_w) // 2
        img_padded = cv2.copyMakeBorder(img_resized, pad_h, h-new_h-pad_h,
                                        pad_w, w-new_w-pad_w, cv2.BORDER_REFLECT)
        return img_padded
    else:
        start_h = (new_h - h)//2
        start_w = (new_w - w)//2
        return img_resized[start_h:start_h+h, start_w:start_w+w]

def random_brightness(img):
    factor = random.uniform(0.7, 1.3)
    img = img.astype(np.float32) * factor
    img = np.clip(img, 0, 255).astype(np.uint8)
    return img

augmentations = [random_flip, random_rotate, random_zoom, random_brightness]

# Apply augmentations
classes = sorted(os.listdir(train_dir))
for cls in classes:
    cls_path = os.path.join(train_dir, cls)
    save_cls_path = os.path.join(aug_train_dir, cls)
    os.makedirs(save_cls_path, exist_ok=True)

    for file in os.listdir(cls_path):
        if not file.lower().endswith((".jpg", ".jpeg", ".png")):
            continue
        img_path = os.path.join(cls_path, file)
        img = cv2.imread(img_path)

        # Save original
        cv2.imwrite(os.path.join(save_cls_path, file), img)

        # 3 random augmentations
        for k in range(3):
            aug_img = img.copy()
            aug_funcs = random.sample(augmentations, 2)
            for func in aug_funcs:
                aug_img = func(aug_img)
            filename, ext = os.path.splitext(file)
            aug_name = f"{filename}_aug{k+1}{ext}"
            cv2.imwrite(os.path.join(save_cls_path, aug_name), aug_img)

# Print image count per class
print("\n Image count per class after augmentation:")
for cls in classes:
    cls_path = os.path.join(aug_train_dir, cls)
    count = len([f for f in os.listdir(cls_path) if f.lower().endswith((".jpg", ".jpeg", ".png"))])
    print(f"{cls}: {count} images")

print(f"\n All train images and augmented images saved in: {aug_train_dir}")
```

```
 Image count per class after augmentation:
Aphids: 1280 images
Army worm: 1280 images
Bacterial Blight: 1280 images
Healthy: 1280 images
Powdery Mildew: 1280 images
Target spot: 1280 images

 All train images and augmented images saved in: /kaggle/working/cotton_train_aug
```

In [ ]:

```
import os, time, psutil
import torch
from torch import nn, optim
from torch.utils.data import DataLoader
from torchvision import datasets, transforms, models
import numpy as np
from sklearn.metrics import classification_report, confusion_matrix, cohen_kappa_score, brier_score_loss
from sklearn.metrics import roc_auc_score, average_precision_score, roc_curve, precision_recall_curve
from scipy.stats import ttest_ind
import matplotlib.pyplot as plt
```

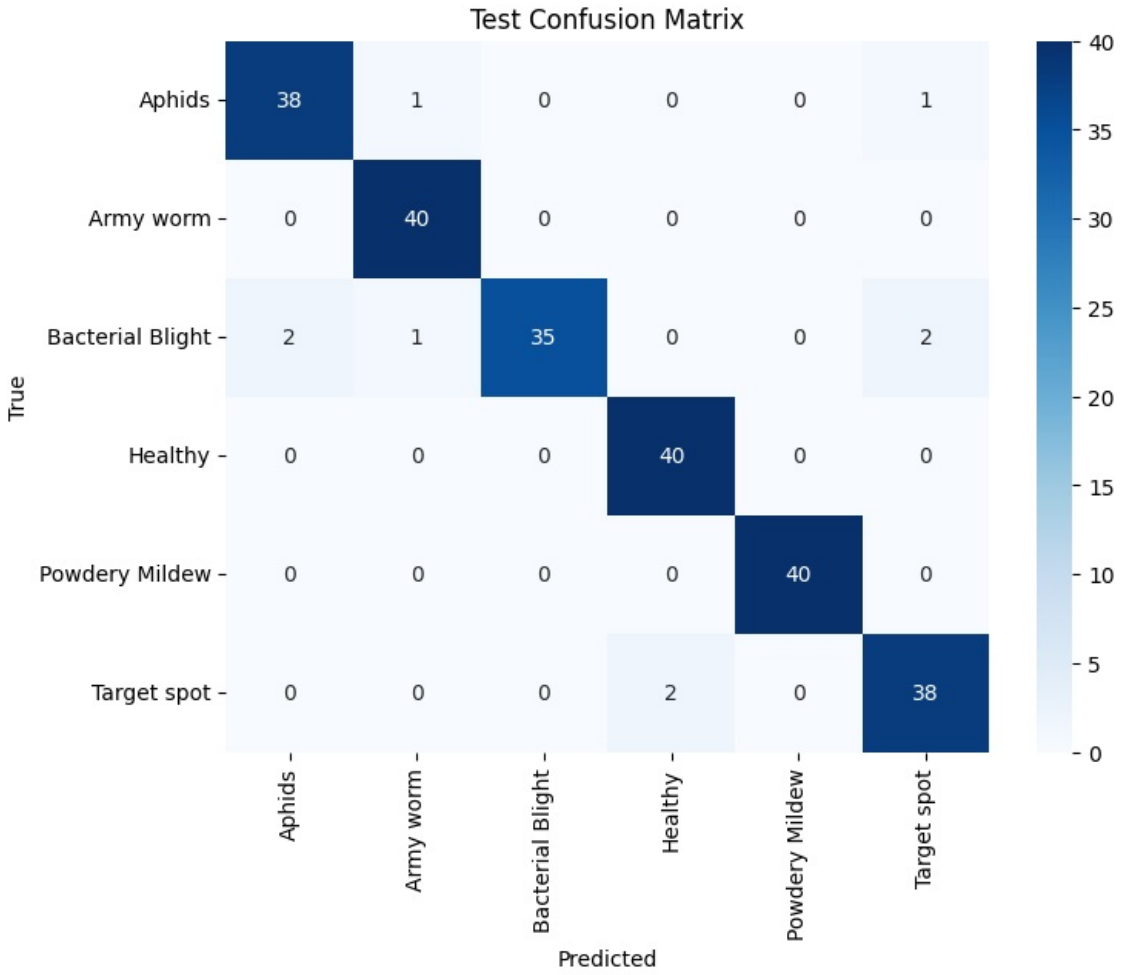
```python
import seaborn as sns
import pandas as pd

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# ----------------------
# 1️⃣ Dataset
# ----------------------
transform = transforms.Compose([
    transforms.Resize((224,224)),
    transforms.ToTensor(),
    transforms.Normalize([0.485,0.456,0.406],[0.229,0.224,0.225])
])

train_dataset = datasets.ImageFolder('/kaggle/working/cotton_train_aug', transform=transform)
val_dataset   = datasets.ImageFolder('/kaggle/working/cotton_split/val', transform=transform)
test_dataset  = datasets.ImageFolder('/kaggle/working/cotton_split/test', transform=transform)

train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
val_loader   = DataLoader(val_dataset, batch_size=32, shuffle=False)
test_loader  = DataLoader(test_dataset, batch_size=32, shuffle=False)
class_names = train_dataset.classes
num_classes = len(class_names)

# ----------------------
# 2️⃣ Multi-Head Self-Attention
# ----------------------
class MultiHeadSelfAttention(nn.Module):
    def __init__(self, in_channels, num_heads=4, dropout=0.3):
        super().__init__()
        assert in_channels % num_heads == 0, f"in_channels ({in_channels}) must be divisible by num_heads ({num_
        self.mha = nn.MultiheadAttention(embed_dim=in_channels, num_heads=num_heads, dropout=dropout, batch_firs
        self.dropout = nn.Dropout(dropout)
        self.norm = nn.LayerNorm(in_channels)

    def forward(self, x):
        b, c, h, w = x.size()
        x_flat = x.view(b, c, h*w).permute(0,2,1)  # [B, HW, C]
        attn_out, _ = self.mha(x_flat, x_flat, x_flat)
        attn_out = self.dropout(attn_out)
        attn_out = self.norm(attn_out + x_flat)
        out = attn_out.permute(0,2,1).view(b, c, h, w)
        return out

# ----------------------
# 3️⃣ ShuffleNetV2 + MHSA Model
# ----------------------
class ShuffleNetV2_MHSA(nn.Module):
    def __init__(self, num_classes, num_heads=4, dropout=0.3, freeze_backbone=True):
        super().__init__()
        self.backbone = models.shufflenet_v2_x1_0(weights=models.ShuffleNet_V2_X1_0_Weights.DEFAULT)
        if freeze_backbone:
            for param in self.backbone.parameters():
                param.requires_grad = False
        self.backbone.fc = nn.Identity()  # remove original fc
        self.mhsa = MultiHeadSelfAttention(in_channels=1024, num_heads=num_heads, dropout=dropout)
        self.classifier = nn.Sequential(
            nn.AdaptiveAvgPool2d(1),
            nn.Flatten(),
            nn.Dropout(dropout),
            nn.Linear(1024, num_classes)
        )

    def forward(self, x):
        features = self.backbone(x)
        features = features.unsqueeze(-1).unsqueeze(-1)  # [B, C, 1, 1]
        attn_features = self.mhsa(features)
        out = self.classifier(attn_features)
        return out

# ----------------------
# 4️⃣ Instantiate model, criterion, optimizer
# ----------------------
model = ShuffleNetV2_MHSA(num_classes=num_classes, num_heads=4, dropout=0.3).to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=1e-4, weight_decay=1e-4)
scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer, 'min', patience=5)

# ----------------------
# 5️⃣ Training loop
# ----------------------
num_epochs = 50
best_val_loss = float('inf')
```

```
patience_counter = 0
train_start_time = time.time()

for epoch in range(num_epochs):
    model.train()
    running_loss, correct, total = 0, 0, 0
    for inputs, labels in train_loader:
        inputs, labels = inputs.to(device), labels.to(device)
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
        _, preds = torch.max(outputs, 1)
        total += labels.size(0)
        correct += (preds == labels).sum().item()

    train_loss = running_loss / len(train_loader)
    train_acc = correct / total

    # Validation
    model.eval()
    val_loss, val_correct, val_total = 0, 0, 0
    with torch.no_grad():
        for inputs, labels in val_loader:
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            val_loss += loss.item()
            _, preds = torch.max(outputs, 1)
            val_total += labels.size(0)
            val_correct += (preds == labels).sum().item()

    val_loss /= len(val_loader)
    val_acc = val_correct / val_total
    scheduler.step(val_loss)

    print(f"Epoch {epoch+1}/{num_epochs} | Train Loss: {train_loss:.4f}, Acc: {train_acc:.4f} | "
          f"Val Loss: {val_loss:.4f}, Acc: {val_acc:.4f}")

    if val_loss < best_val_loss:
        best_val_loss = val_loss
        torch.save(model.state_dict(), 'best_shufflenetv2_mhsa.pth')
        patience_counter = 0
    else:
        patience_counter += 1
        if patience_counter >= 5:
            print("Early stopping triggered")
            break

train_time = time.time() - train_start_time
print(f"\n Training complete. Total time: {train_time:.2f}s")

# ----------------------
# 6️ Load best model
# ----------------------
model.load_state_dict(torch.load('best_shufflenetv2_mhsa.pth'))
model.eval()

# ----------------------
# 7️ Prediction helper
# ----------------------
def get_preds(loader):
    labels_list, preds_list, probs_list = [], [], []
    with torch.no_grad():
        for inputs, labels in loader:
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = model(inputs)
            probs = torch.softmax(outputs, dim=1)
            _, preds = torch.max(outputs, 1)
            labels_list.extend(labels.cpu().numpy())
            preds_list.extend(preds.cpu().numpy())
            probs_list.extend(probs.cpu().numpy())
    return np.array(labels_list), np.array(preds_list), np.array(probs_list)

train_labels, train_preds, train_probs = get_preds(train_loader)
val_labels, val_preds, val_probs       = get_preds(val_loader)
test_labels, test_preds, test_probs    = get_preds(test_loader)

# ----------------------
# 8️ Classification Reports
# ----------------------
```

```python
print("\n Train Classification Report:")
print(classification_report(train_labels, train_preds, target_names=class_names, digits=4))
print("\n Validation Classification Report:")
print(classification_report(val_labels, val_preds, target_names=class_names, digits=4))
print("\n Test Classification Report:")
print(classification_report(test_labels, test_preds, target_names=class_names, digits=4))

# ----------------------
# 9️ Confusion Matrix
# ----------------------
cm = confusion_matrix(test_labels, test_preds)
plt.figure(figsize=(8,6))
sns.heatmap(cm, annot=True, fmt='d', xticklabels=class_names, yticklabels=class_names, cmap='Blues')
plt.xlabel("Predicted"); plt.ylabel("True"); plt.title("Test Confusion Matrix")
plt.show()

# ----------------------
#  ROC & PR AUC
# ----------------------
test_labels_onehot = np.eye(num_classes)[test_labels]
roc_auc_list = [roc_auc_score(test_labels_onehot[:,i], test_probs[:,i]) for i in range(num_classes)]
pr_auc_list  = [average_precision_score(test_labels_onehot[:,i], test_probs[:,i]) for i in range(num_classes)]
auc_df = pd.DataFrame({"Class": class_names, "ROC AUC": [f"{x:.4f}" for x in roc_auc_list], "PR AUC": [f"{x:.4f}
print("\n Class-wise ROC AUC & PR AUC (Test):")
print(auc_df)

# ----------------------
# 1️1️ Inference time & memory
# ----------------------
start_inf = time.time(); _ = get_preds(test_loader)
inf_time = time.time()-start_inf
inf_time_per_sample = inf_time / len(test_dataset)
mem_usage = torch.cuda.memory_allocated()/1024**2 if torch.cuda.is_available() else 0
ram_usage = psutil.virtual_memory().used/1024**2
print(f"Inference time: {inf_time:.2f}s | Per sample: {inf_time_per_sample:.4f}s | GPU Mem: {mem_usage:.2f}MB |

# ----------------------
# 1️2️ Kappa, Brier, PPV, NPV, Accuracy CI, T-test
# ----------------------
kappa = cohen_kappa_score(test_labels, test_preds)
brier = brier_score_loss(test_labels_onehot.flatten(), test_probs.flatten())
n_samples = len(test_labels)
boot_acc = [np.mean(test_labels[np.random.choice(n_samples, n_samples)] == test_preds[np.random.choice(n_samples
ci_low, ci_high = np.percentile(boot_acc, [2.5,97.5])
ttest_res = ttest_ind(test_probs[:,0], test_probs[:,1])
cm = confusion_matrix(test_labels, test_preds)
TP = np.diag(cm); FP = cm.sum(axis=0)-TP; FN = cm.sum(axis=1)-TP; TN = cm.sum()-(TP+FP+FN)
PPV = TP/(TP+FP+1e-8); NPV=TN/(TN+FN+1e-8)
print(f"\nCohen's Kappa: {kappa:.4f}")
print(f"Brier Score: {brier:.4f}")
print(f"Accuracy 95% CI: [{ci_low:.4f},{ci_high:.4f}]")
print(f"T-test class0 vs class1: stat={ttest_res.statistic:.4f}, p={ttest_res.pvalue:.4f}")
print(f"Mean PPV: {np.mean(PPV):.4f} | Mean NPV: {np.mean(NPV):.4f}")
```

```
Downloading: "https://download.pytorch.org/models/shufflenetv2_x1-5666bf0f80.pth" to /root/.cache/torch/hub/chec
kpoints/shufflenetv2_x1-5666bf0f80.pth
100%|██████████| 8.79M/8.79M [00:00<00:00, 107MB/s]
```

```
Epoch 1/50 | Train Loss: 0.5897, Acc: 0.7964 | Val Loss: 0.2117, Acc: 0.9500
Epoch 2/50 | Train Loss: 0.2204, Acc: 0.9272 | Val Loss: 0.1365, Acc: 0.9625
Epoch 3/50 | Train Loss: 0.1583, Acc: 0.9496 | Val Loss: 0.1138, Acc: 0.9625
Epoch 4/50 | Train Loss: 0.1348, Acc: 0.9542 | Val Loss: 0.1316, Acc: 0.9625
Epoch 5/50 | Train Loss: 0.0952, Acc: 0.9690 | Val Loss: 0.1097, Acc: 0.9708
Epoch 6/50 | Train Loss: 0.0918, Acc: 0.9703 | Val Loss: 0.1261, Acc: 0.9667
Epoch 7/50 | Train Loss: 0.0842, Acc: 0.9724 | Val Loss: 0.1457, Acc: 0.9667
Epoch 8/50 | Train Loss: 0.0801, Acc: 0.9740 | Val Loss: 0.1484, Acc: 0.9583
Epoch 9/50 | Train Loss: 0.0669, Acc: 0.9770 | Val Loss: 0.1378, Acc: 0.9625
Epoch 10/50 | Train Loss: 0.0634, Acc: 0.9789 | Val Loss: 0.1492, Acc: 0.9667
Early stopping triggered

 Training complete. Total time: 136.17s

 Train Classification Report:
                  precision    recall  f1-score   support

          Aphids     0.9922    0.9961    0.9942      1280
       Army worm     0.9984    0.9953    0.9969      1280
Bacterial Blight     0.9968    0.9781    0.9874      1280
         Healthy     0.9930    0.9992    0.9961      1280
  Powdery Mildew     0.9969    0.9969    0.9969      1280
     Target spot     0.9815    0.9930    0.9872      1280

        accuracy                         0.9931      7680
       macro avg     0.9931    0.9931    0.9931      7680

    weighted avg     0.9931    0.9931    0.9931      7680


 Validation Classification Report:
                  precision    recall  f1-score   support

          Aphids     0.9500    0.9500    0.9500        40
       Army worm     0.9524    1.0000    0.9756        40
Bacterial Blight     1.0000    0.9250    0.9610        40
         Healthy     1.0000    0.9750    0.9873        40
  Powdery Mildew     1.0000    1.0000    1.0000        40
     Target spot     0.9286    0.9750    0.9512        40

        accuracy                         0.9708       240
       macro avg     0.9718    0.9708    0.9709       240
    weighted avg     0.9718    0.9708    0.9709       240


 Test Classification Report:
                  precision    recall  f1-score   support

          Aphids     0.9500    0.9500    0.9500        40
       Army worm     0.9524    1.0000    0.9756        40
Bacterial Blight     1.0000    0.8750    0.9333        40
         Healthy     0.9524    1.0000    0.9756        40
  Powdery Mildew     1.0000    1.0000    1.0000        40
     Target spot     0.9268    0.9500    0.9383        40

        accuracy                         0.9625       240
       macro avg     0.9636    0.9625    0.9621       240
    weighted avg     0.9636    0.9625    0.9621       240
```




```
 Class-wise ROC AUC & PR AUC (Test):
              Class ROC AUC  PR AUC
0            Aphids  0.9948  0.9839
1         Army worm  0.9994  0.9969
2  Bacterial Blight  0.9981  0.9915
3           Healthy  1.0000  1.0000
4    Powdery Mildew  1.0000  1.0000
5       Target spot  0.9946  0.9853
Inference time: 0.40s | Per sample: 0.0017s | GPU Mem: 94.55MB | RAM: 2136.94MB

Cohen's Kappa: 0.9550
Brier Score: 0.0097
Accuracy 95% CI: [0.1208,0.2167]
T-test class0 vs class1: stat=-0.2407, p=0.8099
Mean PPV: 0.9636 | Mean NPV: 0.9926
```

In [ ]:

```
import os, time, psutil
import torch
from torch import nn, optim
from torch.utils.data import DataLoader
```

```python
from torchvision import datasets, transforms, models
import numpy as np
from sklearn.metrics import classification_report, confusion_matrix, cohen_kappa_score, brier_score_loss
from sklearn.metrics import roc_auc_score, average_precision_score
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# ----------------------
# 1️⃣ Dataset
# ----------------------
transform = transforms.Compose([
    transforms.Resize((224,224)),
    transforms.ToTensor(),
    transforms.Normalize([0.485,0.456,0.406],[0.229,0.224,0.225])
])

train_dataset = datasets.ImageFolder('/kaggle/working/cotton_train_aug', transform=transform)
val_dataset   = datasets.ImageFolder('/kaggle/working/cotton_split/val', transform=transform)
test_dataset  = datasets.ImageFolder('/kaggle/working/cotton_split/test', transform=transform)

train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
val_loader   = DataLoader(val_dataset, batch_size=32, shuffle=False)
test_loader  = DataLoader(test_dataset, batch_size=32, shuffle=False)

class_names = train_dataset.classes
num_classes = len(class_names)

# ----------------------
# 2️⃣ Multi-Head Self-Attention
# ----------------------
class MultiHeadSelfAttention(nn.Module):
    def __init__(self, in_channels, num_heads=4, dropout=0.3):
        super().__init__()
        assert in_channels % num_heads == 0, f"in_channels {in_channels} not divisible by num_heads {num_heads}"
        self.mha = nn.MultiheadAttention(embed_dim=in_channels, num_heads=num_heads, dropout=dropout, batch_firs
        self.dropout = nn.Dropout(dropout)
        self.norm = nn.LayerNorm(in_channels)

    def forward(self, x):
        b, c, h, w = x.size()
        x_flat = x.view(b, c, h*w).permute(0,2,1)   # [B, HW, C]
        attn_out, _ = self.mha(x_flat, x_flat, x_flat)
        attn_out = self.dropout(attn_out)
        attn_out = self.norm(attn_out + x_flat)
        out = attn_out.permute(0,2,1).view(b, c, h, w)
        return out

# ----------------------
# 3️⃣ SqueezeNet + MHSA
# ----------------------
class SqueezeNet_MHSA(nn.Module):
    def __init__(self, num_classes, num_heads=4, dropout=0.3, freeze_backbone=True):
        super().__init__()
        self.backbone = models.squeezenet1_1(weights=models.SqueezeNet1_1_Weights.DEFAULT)
        if freeze_backbone:
            for p in self.backbone.parameters():
                p.requires_grad = False
        self.features = self.backbone.features
        last_channels = 512
        self.mhsa = MultiHeadSelfAttention(in_channels=last_channels, num_heads=num_heads, dropout=dropout)
        self.classifier = nn.Sequential(
            nn.AdaptiveAvgPool2d(1),
            nn.Flatten(),
            nn.Dropout(dropout),
            nn.Linear(last_channels, num_classes)
        )

    def forward(self, x):
        features = self.features(x)
        attn_features = self.mhsa(features)
        out = self.classifier(attn_features)
        return out

# ----------------------
# 4️⃣ Instantiate model, criterion, optimizer
# ----------------------
model = SqueezeNet_MHSA(num_classes=num_classes, num_heads=4, dropout=0.3).to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=1e-4, weight_decay=1e-4)
scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer, 'min', patience=5)
```

```python
# ----------------------
# 5️⃣ Training loop
# ----------------------
num_epochs = 50
best_val_loss = float('inf')
patience_counter = 0

for epoch in range(num_epochs):
    model.train()
    running_loss, correct, total = 0, 0, 0
    for inputs, labels in train_loader:
        inputs, labels = inputs.to(device), labels.to(device)
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
        _, preds = torch.max(outputs, 1)
        total += labels.size(0)
        correct += (preds == labels).sum().item()

    train_loss = running_loss / len(train_loader)
    train_acc = correct / total

    # Validation
    model.eval()
    val_loss, val_correct, val_total = 0, 0, 0
    with torch.no_grad():
        for inputs, labels in val_loader:
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            val_loss += loss.item()
            _, preds = torch.max(outputs, 1)
            val_total += labels.size(0)
            val_correct += (preds == labels).sum().item()

    val_loss /= len(val_loader)
    val_acc = val_correct / val_total
    scheduler.step(val_loss)

    print(f"Epoch {epoch+1}/{num_epochs} | Train Loss: {train_loss:.4f}, Acc: {train_acc:.4f} | "
          f"Val Loss: {val_loss:.4f}, Acc: {val_acc:.4f}")

    if val_loss < best_val_loss:
        best_val_loss = val_loss
        torch.save(model.state_dict(), "best_squeezenet_mhsa.pth")
        patience_counter = 0
    else:
        patience_counter += 1
        if patience_counter >= 5:
            print("Early stopping!")
            break

# ----------------------
# 6️⃣ Load best model
# ----------------------
model.load_state_dict(torch.load("best_squeezenet_mhsa.pth"))
model.eval()

# ----------------------
# 7️⃣ Prediction helper
# ----------------------
def get_preds(loader):
    labels_list, preds_list, probs_list = [], [], []
    with torch.no_grad():
        for inputs, labels in loader:
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = model(inputs)
            probs = torch.softmax(outputs, dim=1)
            _, preds = torch.max(outputs, 1)
            labels_list.extend(labels.cpu().numpy())
            preds_list.extend(preds.cpu().numpy())
            probs_list.extend(probs.cpu().numpy())
    return np.array(labels_list), np.array(preds_list), np.array(probs_list)

train_labels, train_preds, train_probs = get_preds(train_loader)
val_labels, val_preds, val_probs       = get_preds(val_loader)
test_labels, test_preds, test_probs    = get_preds(test_loader)

# ----------------------
```

```python
# 8️⃣ Classification Reports
# ----------------------
print("\n Train Report:\n", classification_report(train_labels, train_preds, target_names=class_names, digits=4))
print("\n Validation Report:\n", classification_report(val_labels, val_preds, target_names=class_names, digits=4))
print("\n Test Report:\n", classification_report(test_labels, test_preds, target_names=class_names, digits=4))

# ----------------------
# 9️⃣ Confusion Matrix
# ----------------------
cm = confusion_matrix(test_labels, test_preds)
plt.figure(figsize=(8,6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=class_names, yticklabels=class_names)
plt.xlabel("Predicted"); plt.ylabel("True"); plt.title("Test Confusion Matrix")
plt.show()

# ----------------------
#  ROC & PR AUC
# ----------------------
test_labels_onehot = np.eye(num_classes)[test_labels]
roc_auc_list = [roc_auc_score(test_labels_onehot[:,i], test_probs[:,i]) for i in range(num_classes)]
pr_auc_list  = [average_precision_score(test_labels_onehot[:,i], test_probs[:,i]) for i in range(num_classes)]
print("\n ROC & PR AUC (per class):")
print(pd.DataFrame({"Class": class_names, "ROC AUC": roc_auc_list, "PR AUC": pr_auc_list}))

# ----------------------
# 1️⃣1️⃣ Extra Metrics with fixed bootstrap + PPV/NPV
# ----------------------
kappa = cohen_kappa_score(test_labels, test_preds)
brier = brier_score_loss(test_labels_onehot.flatten(), test_probs.flatten())

# Bootstrap Accuracy CI: sample the same indices for labels & predictions
n_samples = len(test_labels)
boot_acc = []
rng = np.random.default_rng()
for _ in range(1000):
    idx = rng.integers(0, n_samples, n_samples)
    boot_acc.append(np.mean(test_labels[idx] == test_preds[idx]))
ci_low, ci_high = np.percentile(boot_acc, [2.5, 97.5])

# Confusion matrix components
cm = confusion_matrix(test_labels, test_preds)
TP = np.diag(cm)
FP = cm.sum(axis=0) - TP
FN = cm.sum(axis=1) - TP
TN = cm.sum() - (TP + FP + FN)

PPV = TP / (TP + FP + 1e-8)  # Precision
NPV = TN / (TN + FN + 1e-8)

print(f"\nCohen's Kappa: {kappa:.4f}")
print(f"Brier Score: {brier:.4f}")
print(f"Accuracy 95% CI: [{ci_low:.4f}, {ci_high:.4f}]")
print(f"Mean PPV (Precision): {np.mean(PPV):.4f}")
print(f"Mean NPV: {np.mean(NPV):.4f}")
```

```
Downloading: "https://download.pytorch.org/models/squeezenet1_1-b8a52dc0.pth" to /root/.cache/torch/hub/checkpoints/squeezenet1_1-b8a52dc0.pth
100%|██████████| 4.73M/4.73M [00:00<00:00, 48.2MB/s]
```

```
Epoch 1/50 | Train Loss: 0.7848, Acc: 0.7559 | Val Loss: 0.2967, Acc: 0.9000
Epoch 2/50 | Train Loss: 0.2311, Acc: 0.9387 | Val Loss: 0.1578, Acc: 0.9542
Epoch 3/50 | Train Loss: 0.1054, Acc: 0.9770 | Val Loss: 0.1418, Acc: 0.9500
Epoch 4/50 | Train Loss: 0.0520, Acc: 0.9923 | Val Loss: 0.1353, Acc: 0.9542
Epoch 5/50 | Train Loss: 0.0269, Acc: 0.9986 | Val Loss: 0.1415, Acc: 0.9542
Epoch 6/50 | Train Loss: 0.0153, Acc: 0.9996 | Val Loss: 0.1393, Acc: 0.9625
Epoch 7/50 | Train Loss: 0.0092, Acc: 1.0000 | Val Loss: 0.1381, Acc: 0.9667
Epoch 8/50 | Train Loss: 0.0063, Acc: 1.0000 | Val Loss: 0.1468, Acc: 0.9583
Epoch 9/50 | Train Loss: 0.0046, Acc: 1.0000 | Val Loss: 0.1578, Acc: 0.9667
Early stopping!
```

Train Report:

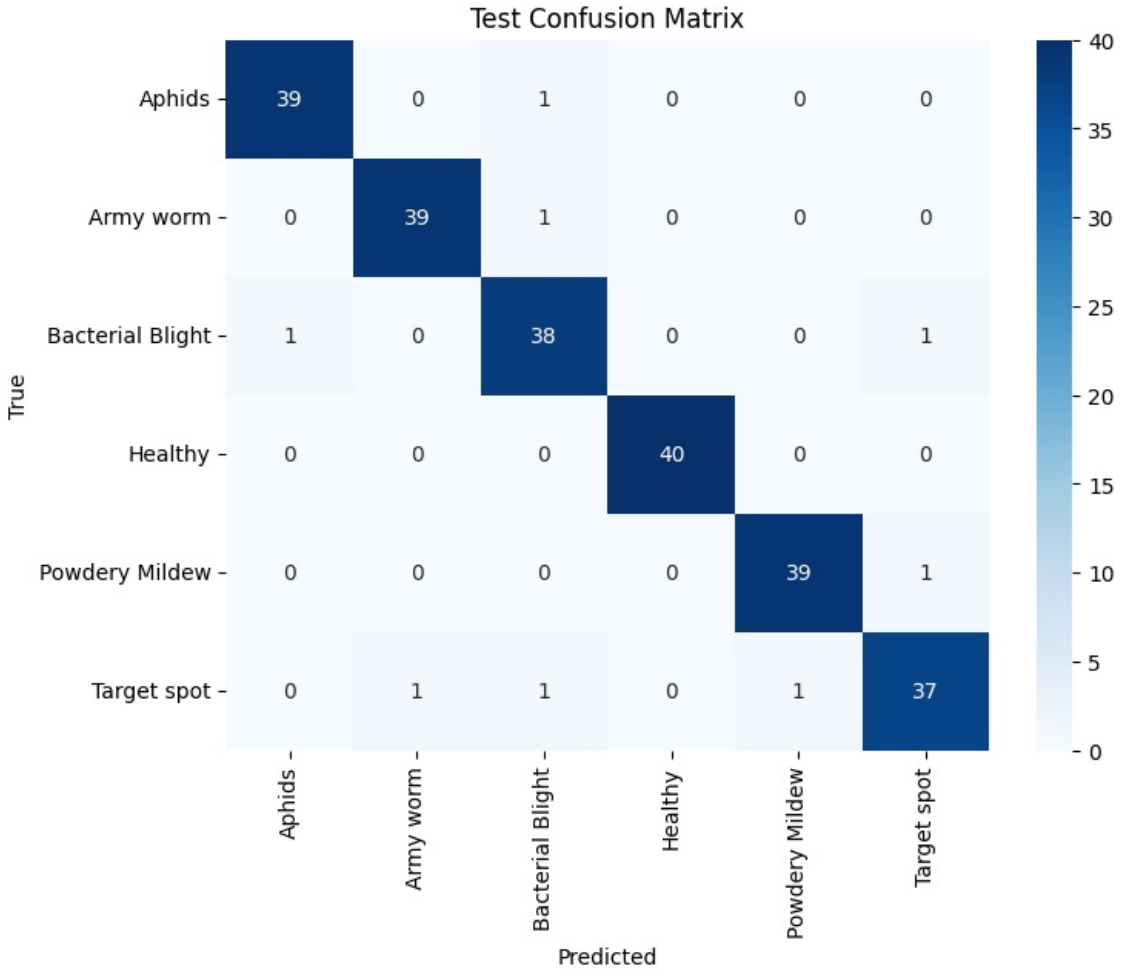
	precision	recall	f1-score	support
Aphids	0.9984	0.9984	0.9984	1280
Army worm	1.0000	0.9984	0.9992	1280
Bacterial Blight	0.9938	0.9984	0.9961	1280
Healthy	1.0000	1.0000	1.0000	1280
Powdery Mildew	1.0000	1.0000	1.0000	1280
Target spot	1.0000	0.9969	0.9984	1280
accuracy			0.9987	7680
macro avg	0.9987	0.9987	0.9987	7680
weighted avg	0.9987	0.9987	0.9987	7680

Validation Report:

	precision	recall	f1-score	support
Aphids	0.9722	0.8750	0.9211	40
Army worm	1.0000	1.0000	1.0000	40
Bacterial Blight	0.8636	0.9500	0.9048	40
Healthy	1.0000	1.0000	1.0000	40
Powdery Mildew	0.9524	1.0000	0.9756	40
Target spot	0.9474	0.9000	0.9231	40
accuracy			0.9542	240
macro avg	0.9559	0.9542	0.9541	240
weighted avg	0.9559	0.9542	0.9541	240

Test Report:

	precision	recall	f1-score	support
Aphids	0.9750	0.9750	0.9750	40
Army worm	0.9750	0.9750	0.9750	40
Bacterial Blight	0.9268	0.9500	0.9383	40
Healthy	1.0000	1.0000	1.0000	40
Powdery Mildew	0.9750	0.9750	0.9750	40
Target spot	0.9487	0.9250	0.9367	40
accuracy			0.9667	240
macro avg	0.9668	0.9667	0.9667	240
weighted avg	0.9668	0.9667	0.9667	240




```
 ROC & PR AUC (per class):
              Class   ROC AUC    PR AUC
0           Aphids  0.999375  0.997222
1        Army worm  0.999875  0.999390
2  Bacterial Blight  0.997625  0.988170
3           Healthy  1.000000  1.000000
4    Powdery Mildew  0.999500  0.997631
5       Target spot  0.997375  0.988708

Cohen's Kappa: 0.9600
Brier Score: 0.0079
Accuracy 95% CI: [0.9417, 0.9875]
Mean PPV (Precision): 0.9668
Mean NPV: 0.9933
```

In [ ]:

```
import os, time, psutil
import torch
```

```python
from torch import nn, optim
from torch.utils.data import DataLoader
from torchvision import datasets, transforms, models
import numpy as np
from sklearn.metrics import classification_report, confusion_matrix, cohen_kappa_score, brier_score_loss
from sklearn.metrics import roc_auc_score, average_precision_score
from scipy.stats import ttest_ind
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# ----------------------
# 1️ Dataset
# ----------------------
transform = transforms.Compose([
    transforms.Resize((224,224)),
    transforms.ToTensor(),
    transforms.Normalize([0.485,0.456,0.406],[0.229,0.224,0.225])
])

train_dataset = datasets.ImageFolder('/kaggle/working/cotton_train_aug', transform=transform)
val_dataset   = datasets.ImageFolder('/kaggle/working/cotton_split/val', transform=transform)
test_dataset  = datasets.ImageFolder('/kaggle/working/cotton_split/test', transform=transform)

train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
val_loader   = DataLoader(val_dataset, batch_size=32, shuffle=False)
test_loader  = DataLoader(test_dataset, batch_size=32, shuffle=False)

class_names = train_dataset.classes
num_classes = len(class_names)

# ----------------------
# 2️ Multi-Head Self-Attention
# ----------------------
class MultiHeadSelfAttention(nn.Module):
    def __init__(self, in_channels, num_heads=4, dropout=0.3):
        super().__init__()
        assert in_channels % num_heads == 0, f"in_channels {in_channels} not divisible by num_heads {num_heads}"
        self.mha = nn.MultiheadAttention(embed_dim=in_channels, num_heads=num_heads, dropout=dropout, batch_firs
        self.dropout = nn.Dropout(dropout)
        self.norm = nn.LayerNorm(in_channels)

    def forward(self, x):
        b, c, h, w = x.size()
        x_flat = x.view(b, c, h*w).permute(0,2,1)   # [B, HW, C]
        attn_out, _ = self.mha(x_flat, x_flat, x_flat)
        attn_out = self.dropout(attn_out)
        attn_out = self.norm(attn_out + x_flat)
        out = attn_out.permute(0,2,1).view(b, c, h, w)
        return out

# ----------------------
# 3️ VGG19 + MHSA
# ----------------------
class VGG19_MHSA(nn.Module):
    def __init__(self, num_classes, num_heads=4, dropout=0.3, freeze_backbone=True):
        super().__init__()
        self.backbone = models.vgg19(weights=models.VGG19_Weights.DEFAULT).features
        if freeze_backbone:
            for p in self.backbone.parameters():
                p.requires_grad = False
        last_channels = 512
        self.mhsa = MultiHeadSelfAttention(in_channels=last_channels, num_heads=num_heads, dropout=dropout)
        self.classifier = nn.Sequential(
            nn.AdaptiveAvgPool2d(1),
            nn.Flatten(),
            nn.Dropout(dropout),
            nn.Linear(last_channels, num_classes)
        )

    def forward(self, x):
        features = self.backbone(x)
        attn_features = self.mhsa(features)
        out = self.classifier(attn_features)
        return out

# ----------------------
# 4️ Model, criterion, optimizer
# ----------------------
model = VGG19_MHSA(num_classes=num_classes, num_heads=4, dropout=0.3).to(device)
criterion = nn.CrossEntropyLoss()
```

```python
optimizer = optim.Adam(model.parameters(), lr=1e-4, weight_decay=1e-4)
scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer, 'min', patience=5)

# ----------------------
# 5️⃣ Training loop
# ----------------------
num_epochs = 50
best_val_loss = float('inf')
patience_counter = 0

for epoch in range(num_epochs):
    model.train()
    running_loss, correct, total = 0, 0, 0
    for inputs, labels in train_loader:
        inputs, labels = inputs.to(device), labels.to(device)
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
        _, preds = torch.max(outputs, 1)
        total += labels.size(0)
        correct += (preds == labels).sum().item()

    train_loss = running_loss / len(train_loader)
    train_acc = correct / total

    # Validation
    model.eval()
    val_loss, val_correct, val_total = 0, 0, 0
    with torch.no_grad():
        for inputs, labels in val_loader:
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            val_loss += loss.item()
            _, preds = torch.max(outputs, 1)
            val_total += labels.size(0)
            val_correct += (preds == labels).sum().item()

    val_loss /= len(val_loader)
    val_acc = val_correct / val_total
    scheduler.step(val_loss)

    print(f"Epoch {epoch+1}/{num_epochs} | Train Loss: {train_loss:.4f}, Acc: {train_acc:.4f} | "
          f"Val Loss: {val_loss:.4f}, Acc: {val_acc:.4f}")

    if val_loss < best_val_loss:
        best_val_loss = val_loss
        torch.save(model.state_dict(), "best_vgg19_mhsa.pth")
        patience_counter = 0
    else:
        patience_counter += 1
        if patience_counter >= 5:
            print("Early stopping!")
            break

# ----------------------
# 6️⃣ Load best model
# ----------------------
model.load_state_dict(torch.load("best_vgg19_mhsa.pth"))
model.eval()

# ----------------------
# 7️⃣ Prediction helper
# ----------------------
def get_preds(loader):
    labels_list, preds_list, probs_list = [], [], []
    with torch.no_grad():
        for inputs, labels in loader:
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = model(inputs)
            probs = torch.softmax(outputs, dim=1)
            _, preds = torch.max(outputs, 1)
            labels_list.extend(labels.cpu().numpy())
            preds_list.extend(preds.cpu().numpy())
            probs_list.extend(probs.cpu().numpy())
    return np.array(labels_list), np.array(preds_list), np.array(probs_list)

train_labels, train_preds, train_probs = get_preds(train_loader)
val_labels, val_preds, val_probs       = get_preds(val_loader)
test_labels, test_preds, test_probs    = get_preds(test_loader)
```

```python
# ----------------------
# 8️⃣ Classification Reports
# ----------------------
print("\n Train Report:\n", classification_report(train_labels, train_preds, target_names=class_names, digits=4))
print("\n Validation Report:\n", classification_report(val_labels, val_preds, target_names=class_names, digits=4))
print("\n Test Report:\n", classification_report(test_labels, test_preds, target_names=class_names, digits=4))

# ----------------------
# 9️⃣ Confusion Matrix
# ----------------------
cm = confusion_matrix(test_labels, test_preds)
plt.figure(figsize=(8,6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=class_names, yticklabels=class_names)
plt.xlabel("Predicted"); plt.ylabel("True"); plt.title("Test Confusion Matrix")
plt.show()

# ----------------------
#  ROC & PR AUC
# ----------------------
test_labels_onehot = np.eye(num_classes)[test_labels]
roc_auc_list = [roc_auc_score(test_labels_onehot[:,i], test_probs[:,i]) for i in range(num_classes)]
pr_auc_list  = [average_precision_score(test_labels_onehot[:,i], test_probs[:,i]) for i in range(num_classes)]
print("\n ROC & PR AUC (per class):")
print(pd.DataFrame({"Class": class_names, "ROC AUC": roc_auc_list, "PR AUC": pr_auc_list}))

# ----------------------
# 1️⃣1️⃣ Extra Metrics with fixed bootstrap + PPV/NPV
# ----------------------
kappa = cohen_kappa_score(test_labels, test_preds)
brier = brier_score_loss(test_labels_onehot.flatten(), test_probs.flatten())

# Bootstrap Accuracy CI: sample the same indices for labels & predictions
n_samples = len(test_labels)
boot_acc = []
rng = np.random.default_rng()
for _ in range(1000):
    idx = rng.integers(0, n_samples, n_samples)
    boot_acc.append(np.mean(test_labels[idx] == test_preds[idx]))
ci_low, ci_high = np.percentile(boot_acc, [2.5, 97.5])

# Confusion matrix components
cm = confusion_matrix(test_labels, test_preds)
TP = np.diag(cm)
FP = cm.sum(axis=0) - TP
FN = cm.sum(axis=1) - TP
TN = cm.sum() - (TP + FP + FN)

PPV = TP / (TP + FP + 1e-8)  # Precision
NPV = TN / (TN + FN + 1e-8)

print(f"\nCohen's Kappa: {kappa:.4f}")
print(f"Brier Score: {brier:.4f}")
print(f"Accuracy 95% CI: [{ci_low:.4f}, {ci_high:.4f}]")
print(f"Mean PPV (Precision): {np.mean(PPV):.4f}")
print(f"Mean NPV: {np.mean(NPV):.4f}")
```

```
Downloading: "https://download.pytorch.org/models/vgg19-dcbb9e9d.pth" to /root/.cache/torch/hub/checkpoints/vgg19-dcbb9e9d.pth
100%|██████████| 548M/548M [00:02<00:00, 213MB/s]
```

```
Epoch 1/50 | Train Loss: 0.6217, Acc: 0.8126 | Val Loss: 0.2864, Acc: 0.8958
Epoch 2/50 | Train Loss: 0.2010, Acc: 0.9408 | Val Loss: 0.1510, Acc: 0.9625
Epoch 3/50 | Train Loss: 0.1105, Acc: 0.9684 | Val Loss: 0.1367, Acc: 0.9417
Epoch 4/50 | Train Loss: 0.0644, Acc: 0.9827 | Val Loss: 0.1148, Acc: 0.9542
Epoch 5/50 | Train Loss: 0.0387, Acc: 0.9913 | Val Loss: 0.1263, Acc: 0.9458
Epoch 6/50 | Train Loss: 0.0245, Acc: 0.9967 | Val Loss: 0.1245, Acc: 0.9542
Epoch 7/50 | Train Loss: 0.0159, Acc: 0.9986 | Val Loss: 0.1204, Acc: 0.9542
Epoch 8/50 | Train Loss: 0.0109, Acc: 0.9995 | Val Loss: 0.1176, Acc: 0.9500
Epoch 9/50 | Train Loss: 0.0083, Acc: 0.9996 | Val Loss: 0.1260, Acc: 0.9542
Early stopping!
```

Train Report:

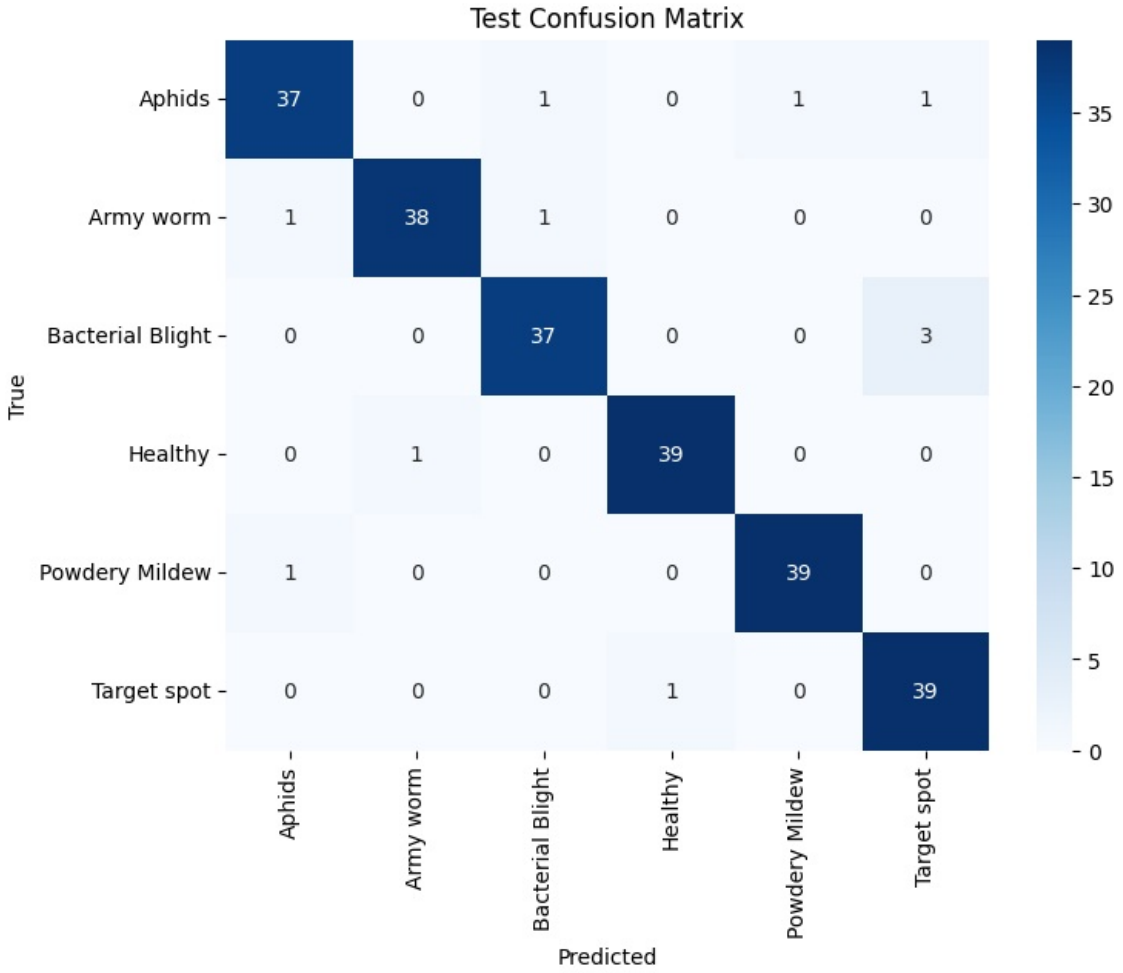
	precision	recall	f1-score	support
Aphids	0.9915	0.9977	0.9945	1280
Army worm	0.9984	0.9953	0.9969	1280
Bacterial Blight	0.9952	0.9812	0.9882	1280
Healthy	0.9969	0.9969	0.9969	1280
Powdery Mildew	0.9961	0.9969	0.9965	1280
Target spot	0.9822	0.9922	0.9872	1280
accuracy			0.9934	7680
macro avg	0.9934	0.9934	0.9934	7680
weighted avg	0.9934	0.9934	0.9934	7680

Validation Report:

	precision	recall	f1-score	support
Aphids	0.9500	0.9500	0.9500	40
Army worm	0.9737	0.9250	0.9487	40
Bacterial Blight	0.9512	0.9750	0.9630	40
Healthy	0.9737	0.9250	0.9487	40
Powdery Mildew	0.9756	1.0000	0.9877	40
Target spot	0.9048	0.9500	0.9268	40
accuracy			0.9542	240
macro avg	0.9548	0.9542	0.9541	240
weighted avg	0.9548	0.9542	0.9541	240

Test Report:

	precision	recall	f1-score	support
Aphids	0.9487	0.9250	0.9367	40
Army worm	0.9744	0.9500	0.9620	40
Bacterial Blight	0.9487	0.9250	0.9367	40
Healthy	0.9750	0.9750	0.9750	40
Powdery Mildew	0.9750	0.9750	0.9750	40
Target spot	0.9070	0.9750	0.9398	40
accuracy			0.9542	240
macro avg	0.9548	0.9542	0.9542	240
weighted avg	0.9548	0.9542	0.9542	240




```
 ROC & PR AUC (per class):
              Class   ROC AUC    PR AUC
0           Aphids  0.993375  0.977877
1        Army worm  0.999250  0.996470
2  Bacterial Blight  0.998500  0.992989
3           Healthy  0.999875  0.999390
4    Powdery Mildew  0.997500  0.990258
5       Target spot  0.998375  0.992584

Cohen's Kappa: 0.9450
Brier Score: 0.0103
Accuracy 95% CI: [0.9250, 0.9792]
Mean PPV (Precision): 0.9548
Mean NPV: 0.9909
```

In [ ]:

```
import os, time, psutil
import torch
```

```python
from torch import nn, optim
from torch.utils.data import DataLoader
from torchvision import datasets, transforms, models
import numpy as np
from sklearn.metrics import classification_report, confusion_matrix, cohen_kappa_score, brier_score_loss
from sklearn.metrics import roc_auc_score, average_precision_score
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# ----------------------
# 1️⃣ Dataset
# ----------------------
transform = transforms.Compose([
    transforms.Resize((224,224)),
    transforms.ToTensor(),
    transforms.Normalize([0.485,0.456,0.406],[0.229,0.224,0.225])
])

train_dataset = datasets.ImageFolder('/kaggle/working/cotton_train_aug', transform=transform)
val_dataset   = datasets.ImageFolder('/kaggle/working/cotton_split/val', transform=transform)
test_dataset  = datasets.ImageFolder('/kaggle/working/cotton_split/test', transform=transform)

train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
val_loader   = DataLoader(val_dataset, batch_size=32, shuffle=False)
test_loader  = DataLoader(test_dataset, batch_size=32, shuffle=False)

class_names = train_dataset.classes
num_classes = len(class_names)

# ----------------------
# 2️⃣ Multi-Head Self-Attention
# ----------------------
class MultiHeadSelfAttention(nn.Module):
    def __init__(self, in_channels, num_heads=4, dropout=0.3):
        super().__init__()
        assert in_channels % num_heads == 0, f"in_channels {in_channels} not divisible by num_heads {num_heads}"
        self.mha = nn.MultiheadAttention(embed_dim=in_channels, num_heads=num_heads, dropout=dropout, batch_firs
        self.dropout = nn.Dropout(dropout)
        self.norm = nn.LayerNorm(in_channels)

    def forward(self, x):
        b, c, h, w = x.size()
        x_flat = x.view(b, c, h*w).permute(0,2,1)   # [B, HW, C]
        attn_out, _ = self.mha(x_flat, x_flat, x_flat)
        attn_out = self.dropout(attn_out)
        attn_out = self.norm(attn_out + x_flat)
        out = attn_out.permute(0,2,1).view(b, c, h, w)
        return out

# ----------------------
# 3️⃣ ResNet101 + MHSA
# ----------------------
class ResNet101_MHSA(nn.Module):
    def __init__(self, num_classes, num_heads=4, dropout=0.3, freeze_backbone=True):
        super().__init__()
        self.backbone = models.resnet101(weights=models.ResNet101_Weights.DEFAULT)
        if freeze_backbone:
            for p in self.backbone.parameters():
                p.requires_grad = False
        # Remove fully connected layer
        self.backbone.fc = nn.Identity()
        last_channels = self.backbone.fc.in_features if hasattr(self.backbone.fc, 'in_features') else 2048
        self.mhsa = MultiHeadSelfAttention(in_channels=2048, num_heads=num_heads, dropout=dropout)
        self.classifier = nn.Sequential(
            nn.AdaptiveAvgPool2d(1),
            nn.Flatten(),
            nn.Dropout(dropout),
            nn.Linear(2048, num_classes)
        )

    def forward(self, x):
        features = self.backbone(x).unsqueeze(-1).unsqueeze(-1)  # [B, 2048, 1, 1] if needed
        attn_features = self.mhsa(features)
        out = self.classifier(attn_features)
        return out

# ----------------------
# 4️⃣ Instantiate model, criterion, optimizer
# ----------------------
model = ResNet101_MHSA(num_classes=num_classes, num_heads=4, dropout=0.3).to(device)
```

```python
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=1e-4, weight_decay=1e-4)
scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer, 'min', patience=5)

# ----------------------
# 5️⃣ Training loop
# ----------------------
num_epochs = 50
best_val_loss = float('inf')
patience_counter = 0

for epoch in range(num_epochs):
    model.train()
    running_loss, correct, total = 0, 0, 0
    for inputs, labels in train_loader:
        inputs, labels = inputs.to(device), labels.to(device)
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
        _, preds = torch.max(outputs, 1)
        total += labels.size(0)
        correct += (preds == labels).sum().item()

    train_loss = running_loss / len(train_loader)
    train_acc = correct / total

    # Validation
    model.eval()
    val_loss, val_correct, val_total = 0, 0, 0
    with torch.no_grad():
        for inputs, labels in val_loader:
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            val_loss += loss.item()
            _, preds = torch.max(outputs, 1)
            val_total += labels.size(0)
            val_correct += (preds == labels).sum().item()

    val_loss /= len(val_loader)
    val_acc = val_correct / val_total
    scheduler.step(val_loss)

    print(f"Epoch {epoch+1}/{num_epochs} | Train Loss: {train_loss:.4f}, Acc: {train_acc:.4f} | "
          f"Val Loss: {val_loss:.4f}, Acc: {val_acc:.4f}")

    if val_loss < best_val_loss:
        best_val_loss = val_loss
        torch.save(model.state_dict(), "best_resnet101_mhsa.pth")
        patience_counter = 0
    else:
        patience_counter += 1
        if patience_counter >= 5:
            print("Early stopping!")
            break

# ----------------------
# 6️⃣ Load best model
# ----------------------
model.load_state_dict(torch.load("best_resnet101_mhsa.pth"))
model.eval()

# ----------------------
# 7️⃣ Prediction helper
# ----------------------
def get_preds(loader):
    labels_list, preds_list, probs_list = [], [], []
    with torch.no_grad():
        for inputs, labels in loader:
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = model(inputs)
            probs = torch.softmax(outputs, dim=1)
            _, preds = torch.max(outputs, 1)
            labels_list.extend(labels.cpu().numpy())
            preds_list.extend(preds.cpu().numpy())
            probs_list.extend(probs.cpu().numpy())
    return np.array(labels_list), np.array(preds_list), np.array(probs_list)

train_labels, train_preds, train_probs = get_preds(train_loader)
val_labels, val_preds, val_probs       = get_preds(val_loader)
```

```python
test_labels, test_preds, test_probs    = get_preds(test_loader)

# ----------------------
# 8️⃣ Classification Reports
# ----------------------
print("\n Train Report:\n", classification_report(train_labels, train_preds, target_names=class_names, digits=4))
print("\n Validation Report:\n", classification_report(val_labels, val_preds, target_names=class_names, digits=4))
print("\n Test Report:\n", classification_report(test_labels, test_preds, target_names=class_names, digits=4))

# ----------------------
# 9️⃣ Confusion Matrix
# ----------------------
cm = confusion_matrix(test_labels, test_preds)
plt.figure(figsize=(8,6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=class_names, yticklabels=class_names)
plt.xlabel("Predicted"); plt.ylabel("True"); plt.title("Test Confusion Matrix")
plt.show()

# ----------------------
#  ROC & PR AUC
# ----------------------
test_labels_onehot = np.eye(num_classes)[test_labels]
roc_auc_list = [roc_auc_score(test_labels_onehot[:,i], test_probs[:,i]) for i in range(num_classes)]
pr_auc_list  = [average_precision_score(test_labels_onehot[:,i], test_probs[:,i]) for i in range(num_classes)]
print("\n ROC & PR AUC (per class):")
print(pd.DataFrame({"Class": class_names, "ROC AUC": roc_auc_list, "PR AUC": pr_auc_list}))

# ----------------------
# 1️⃣1️⃣ Extra Metrics with PPV/NPV
# ----------------------
kappa = cohen_kappa_score(test_labels, test_preds)
brier = brier_score_loss(test_labels_onehot.flatten(), test_probs.flatten())

# Bootstrap Accuracy CI
n_samples = len(test_labels)
boot_acc = []
rng = np.random.default_rng()
for _ in range(1000):
    idx = rng.integers(0, n_samples, n_samples)
    boot_acc.append(np.mean(test_labels[idx] == test_preds[idx]))
ci_low, ci_high = np.percentile(boot_acc, [2.5, 97.5])

# Confusion matrix components
TP = np.diag(cm)
FP = cm.sum(axis=0) - TP
FN = cm.sum(axis=1) - TP
TN = cm.sum() - (TP + FP + FN)

PPV = TP / (TP + FP + 1e-8)
NPV = TN / (TN + FN + 1e-8)

print(f"\nCohen's Kappa: {kappa:.4f}")
print(f"Brier Score: {brier:.4f}")
print(f"Accuracy 95% CI: [{ci_low:.4f}, {ci_high:.4f}]")
print(f"Mean PPV (Precision): {np.mean(PPV):.4f}")
print(f"Mean NPV: {np.mean(NPV):.4f}")
```

```
Downloading: "https://download.pytorch.org/models/resnet101-cd907fc2.pth" to /root/.cache/torch/hub/checkpoints/
resnet101-cd907fc2.pth
100%|██████████| 171M/171M [00:00<00:00, 229MB/s]
```

```
Epoch 1/50 | Train Loss: 0.4504, Acc: 0.8438 | Val Loss: 0.0975, Acc: 0.9708
Epoch 2/50 | Train Loss: 0.1449, Acc: 0.9510 | Val Loss: 0.1080, Acc: 0.9667
Epoch 3/50 | Train Loss: 0.1069, Acc: 0.9634 | Val Loss: 0.0688, Acc: 0.9792
Epoch 4/50 | Train Loss: 0.0850, Acc: 0.9716 | Val Loss: 0.0721, Acc: 0.9708
Epoch 5/50 | Train Loss: 0.0670, Acc: 0.9762 | Val Loss: 0.1074, Acc: 0.9667
Epoch 6/50 | Train Loss: 0.0744, Acc: 0.9717 | Val Loss: 0.0954, Acc: 0.9750
Epoch 7/50 | Train Loss: 0.0796, Acc: 0.9728 | Val Loss: 0.0670, Acc: 0.9833
Epoch 8/50 | Train Loss: 0.0569, Acc: 0.9796 | Val Loss: 0.0775, Acc: 0.9875
Epoch 9/50 | Train Loss: 0.0600, Acc: 0.9793 | Val Loss: 0.0586, Acc: 0.9792
Epoch 10/50 | Train Loss: 0.0479, Acc: 0.9849 | Val Loss: 0.1528, Acc: 0.9708
Epoch 11/50 | Train Loss: 0.0623, Acc: 0.9797 | Val Loss: 0.0750, Acc: 0.9750
Epoch 12/50 | Train Loss: 0.0423, Acc: 0.9875 | Val Loss: 0.1240, Acc: 0.9708
Epoch 13/50 | Train Loss: 0.0424, Acc: 0.9865 | Val Loss: 0.1482, Acc: 0.9750
Epoch 14/50 | Train Loss: 0.0421, Acc: 0.9862 | Val Loss: 0.0259, Acc: 0.9875
Epoch 15/50 | Train Loss: 0.0520, Acc: 0.9806 | Val Loss: 0.0718, Acc: 0.9833
Epoch 16/50 | Train Loss: 0.0423, Acc: 0.9861 | Val Loss: 0.0860, Acc: 0.9833
Epoch 17/50 | Train Loss: 0.0396, Acc: 0.9862 | Val Loss: 0.0676, Acc: 0.9792
Epoch 18/50 | Train Loss: 0.0378, Acc: 0.9879 | Val Loss: 0.0783, Acc: 0.9792
Epoch 19/50 | Train Loss: 0.0496, Acc: 0.9833 | Val Loss: 0.1002, Acc: 0.9750
Early stopping!
```

Train Report:

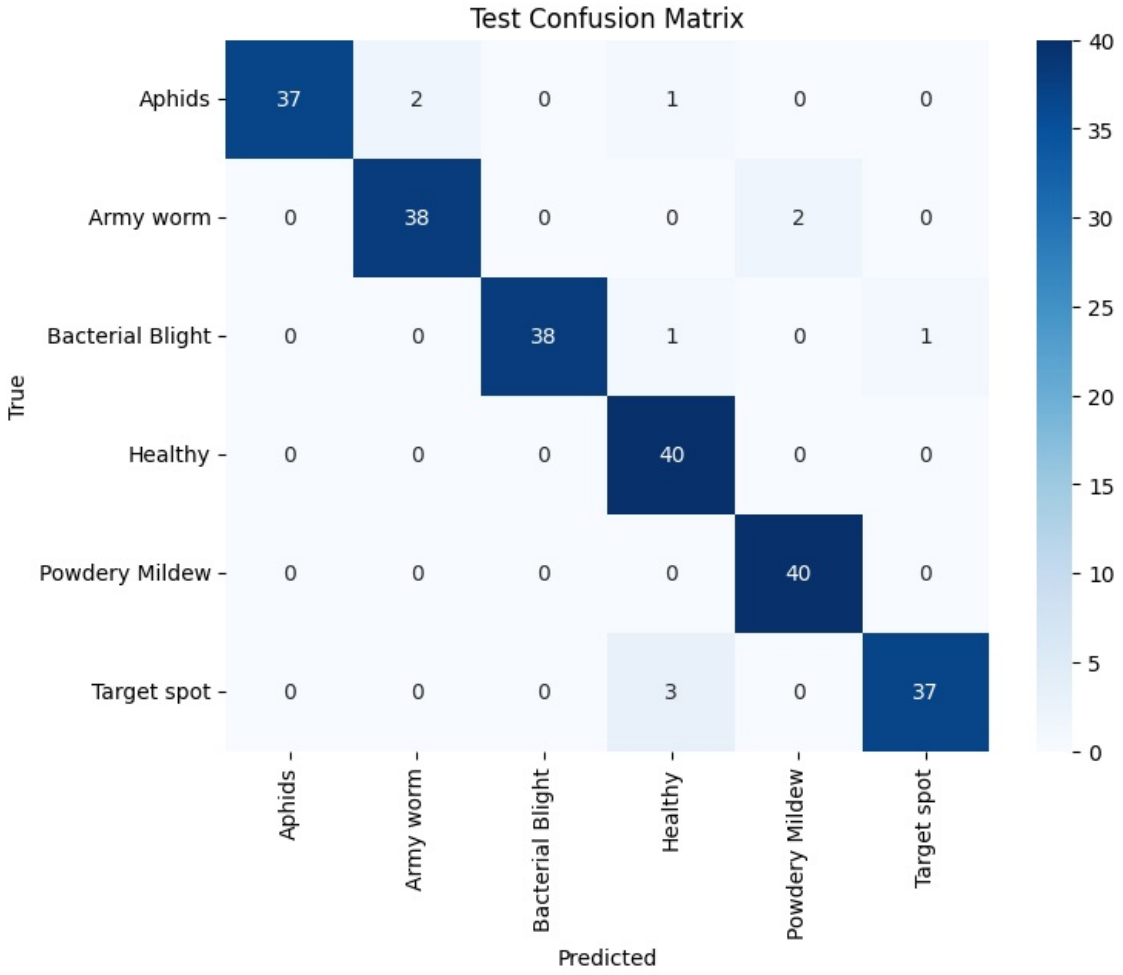
	precision	recall	f1-score	support
Aphids	0.9969	1.0000	0.9984	1280
Army worm	1.0000	0.9992	0.9996	1280
Bacterial Blight	0.9992	0.9977	0.9984	1280
Healthy	0.9977	1.0000	0.9988	1280
Powdery Mildew	1.0000	0.9992	0.9996	1280
Target spot	1.0000	0.9977	0.9988	1280
accuracy			0.9990	7680
macro avg	0.9990	0.9990	0.9990	7680
weighted avg	0.9990	0.9990	0.9990	7680

Validation Report:

	precision	recall	f1-score	support
Aphids	0.9750	0.9750	0.9750	40
Army worm	1.0000	0.9750	0.9873	40
Bacterial Blight	0.9524	1.0000	0.9756	40
Healthy	1.0000	1.0000	1.0000	40
Powdery Mildew	1.0000	1.0000	1.0000	40
Target spot	1.0000	0.9750	0.9873	40
accuracy			0.9875	240
macro avg	0.9879	0.9875	0.9875	240
weighted avg	0.9879	0.9875	0.9875	240

Test Report:

	precision	recall	f1-score	support
Aphids	1.0000	0.9250	0.9610	40
Army worm	0.9500	0.9500	0.9500	40
Bacterial Blight	1.0000	0.9500	0.9744	40
Healthy	0.8889	1.0000	0.9412	40
Powdery Mildew	0.9524	1.0000	0.9756	40
Target spot	0.9737	0.9250	0.9487	40
accuracy			0.9583	240
macro avg	0.9608	0.9583	0.9585	240
weighted avg	0.9608	0.9583	0.9585	240




```
 ROC & PR AUC (per class):
              Class   ROC AUC    PR AUC
0           Aphids  0.996625  0.988060
1         Army worm  0.999500  0.997590
2  Bacterial Blight  1.000000  1.000000
3           Healthy  0.999875  0.999390
4    Powdery Mildew  0.999875  0.999390
5       Target spot  0.995625  0.985056

Cohen's Kappa: 0.9500
Brier Score: 0.0099
Accuracy 95% CI: [0.9333, 0.9833]
Mean PPV (Precision): 0.9608
Mean NPV: 0.9917
```

In [ ]:

```
import os, time, psutil
import torch
from torch import nn, optim
from torch.utils.data import DataLoader
from torchvision import datasets, transforms, models
import numpy as np
from sklearn.metrics import classification_report, confusion_matrix, cohen_kappa_score, brier_score_loss
from sklearn.metrics import roc_auc_score, average_precision_score
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
```

```python
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# ----------------------
# 1️⃣ Dataset
# ----------------------
transform = transforms.Compose([
    transforms.Resize((224,224)),
    transforms.ToTensor(),
    transforms.Normalize([0.485,0.456,0.406],[0.229,0.224,0.225])
])

train_dataset = datasets.ImageFolder('/kaggle/working/cotton_train_aug', transform=transform)
val_dataset   = datasets.ImageFolder('/kaggle/working/cotton_split/val', transform=transform)
test_dataset  = datasets.ImageFolder('/kaggle/working/cotton_split/test', transform=transform)

train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
val_loader   = DataLoader(val_dataset, batch_size=32, shuffle=False)
test_loader  = DataLoader(test_dataset, batch_size=32, shuffle=False)

class_names = train_dataset.classes
num_classes = len(class_names)

# ----------------------
# 2️⃣ Multi-Head Self-Attention
# ----------------------
class MultiHeadSelfAttention(nn.Module):
    def __init__(self, in_channels, num_heads=4, dropout=0.3):
        super().__init__()
        assert in_channels % num_heads == 0, f"in_channels {in_channels} not divisible by num_heads {num_heads}"
        self.mha = nn.MultiheadAttention(embed_dim=in_channels, num_heads=num_heads, dropout=dropout, batch_firs
        self.dropout = nn.Dropout(dropout)
        self.norm = nn.LayerNorm(in_channels)

    def forward(self, x):
        b, c, h, w = x.size()
        x_flat = x.view(b, c, h*w).permute(0,2,1)   # [B, HW, C]
        attn_out, _ = self.mha(x_flat, x_flat, x_flat)
        attn_out = self.dropout(attn_out)
        attn_out = self.norm(attn_out + x_flat)
        out = attn_out.permute(0,2,1).view(b, c, h, w)
        return out

# ----------------------
# 3️⃣ MobileNetV2 + MHSA
# ----------------------
class MobileNetV2_MHSA(nn.Module):
    def __init__(self, num_classes, num_heads=4, dropout=0.3, freeze_backbone=True):
        super().__init__()
        self.backbone = models.mobilenet_v2(weights=models.MobileNet_V2_Weights.DEFAULT)
        if freeze_backbone:
            for p in self.backbone.parameters():
                p.requires_grad = False
        # Remove classifier, keep features
        self.features = self.backbone.features
        last_channels = 1280  # final channels of MobileNetV2
        self.mhsa = MultiHeadSelfAttention(in_channels=last_channels, num_heads=num_heads, dropout=dropout)
        self.classifier = nn.Sequential(
            nn.AdaptiveAvgPool2d(1),
            nn.Flatten(),
            nn.Dropout(dropout),
            nn.Linear(last_channels, num_classes)
        )

    def forward(self, x):
        features = self.features(x)        # [B, 1280, H, W]
        attn_features = self.mhsa(features)
        out = self.classifier(attn_features)
        return out

# ----------------------
# 4️⃣ Instantiate model, criterion, optimizer
# ----------------------
model = MobileNetV2_MHSA(num_classes=num_classes, num_heads=4, dropout=0.3).to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=1e-4, weight_decay=1e-4)
scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer, 'min', patience=5)

# ----------------------
# 5️⃣ Training loop
# ----------------------
num_epochs = 50
best_val_loss = float('inf')
patience_counter = 0
```

```python
for epoch in range(num_epochs):
    model.train()
    running_loss, correct, total = 0, 0, 0
    for inputs, labels in train_loader:
        inputs, labels = inputs.to(device), labels.to(device)
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
        _, preds = torch.max(outputs, 1)
        total += labels.size(0)
        correct += (preds == labels).sum().item()

    train_loss = running_loss / len(train_loader)
    train_acc = correct / total

    # Validation
    model.eval()
    val_loss, val_correct, val_total = 0, 0, 0
    with torch.no_grad():
        for inputs, labels in val_loader:
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            val_loss += loss.item()
            _, preds = torch.max(outputs, 1)
            val_total += labels.size(0)
            val_correct += (preds == labels).sum().item()

    val_loss /= len(val_loader)
    val_acc = val_correct / val_total
    scheduler.step(val_loss)

    print(f"Epoch {epoch+1}/{num_epochs} | Train Loss: {train_loss:.4f}, Acc: {train_acc:.4f} | "
          f"Val Loss: {val_loss:.4f}, Acc: {val_acc:.4f}")

    if val_loss < best_val_loss:
        best_val_loss = val_loss
        torch.save(model.state_dict(), "best_mobilenetv2_mhsa.pth")
        patience_counter = 0
    else:
        patience_counter += 1
        if patience_counter >= 5:
            print("Early stopping!")
            break

# ----------------------
# 6️⃣ Load best model
# ----------------------
model.load_state_dict(torch.load("best_mobilenetv2_mhsa.pth"))
model.eval()

# ----------------------
# 7️⃣ Prediction helper
# ----------------------
def get_preds(loader):
    labels_list, preds_list, probs_list = [], [], []
    with torch.no_grad():
        for inputs, labels in loader:
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = model(inputs)
            probs = torch.softmax(outputs, dim=1)
            _, preds = torch.max(outputs, 1)
            labels_list.extend(labels.cpu().numpy())
            preds_list.extend(preds.cpu().numpy())
            probs_list.extend(probs.cpu().numpy())
    return np.array(labels_list), np.array(preds_list), np.array(probs_list)

train_labels, train_preds, train_probs = get_preds(train_loader)
val_labels, val_preds, val_probs       = get_preds(val_loader)
test_labels, test_preds, test_probs    = get_preds(test_loader)

# ----------------------
# 8️⃣ Classification Reports
# ----------------------
print("\n Train Report:\n", classification_report(train_labels, train_preds, target_names=class_names, digits=4)
print("\n Validation Report:\n", classification_report(val_labels, val_preds, target_names=class_names, digits=4
print("\n Test Report:\n", classification_report(test_labels, test_preds, target_names=class_names, digits=4))

# ----------------------
```

```python
# 9️⃣ Confusion Matrix
# ----------------------
cm = confusion_matrix(test_labels, test_preds)
plt.figure(figsize=(8,6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=class_names, yticklabels=class_names)
plt.xlabel("Predicted"); plt.ylabel("True"); plt.title("Test Confusion Matrix")
plt.show()

# ----------------------
#  ROC & PR AUC
# ----------------------
test_labels_onehot = np.eye(num_classes)[test_labels]
roc_auc_list = [roc_auc_score(test_labels_onehot[:,i], test_probs[:,i]) for i in range(num_classes)]
pr_auc_list  = [average_precision_score(test_labels_onehot[:,i], test_probs[:,i]) for i in range(num_classes)]
print("\n ROC & PR AUC (per class):")
print(pd.DataFrame({"Class": class_names, "ROC AUC": roc_auc_list, "PR AUC": pr_auc_list}))

# ----------------------
# 1️⃣1️⃣ Extra Metrics with PPV/NPV
# ----------------------
kappa = cohen_kappa_score(test_labels, test_preds)
brier = brier_score_loss(test_labels_onehot.flatten(), test_probs.flatten())

# Bootstrap Accuracy CI
n_samples = len(test_labels)
boot_acc = []
rng = np.random.default_rng()
for _ in range(1000):
    idx = rng.integers(0, n_samples, n_samples)
    boot_acc.append(np.mean(test_labels[idx] == test_preds[idx]))
ci_low, ci_high = np.percentile(boot_acc, [2.5, 97.5])

# Confusion matrix components
TP = np.diag(cm)
FP = cm.sum(axis=0) - TP
FN = cm.sum(axis=1) - TP
TN = cm.sum() - (TP + FP + FN)

PPV = TP / (TP + FP + 1e-8)
NPV = TN / (TN + FN + 1e-8)

print(f"\nCohen's Kappa: {kappa:.4f}")
print(f"Brier Score: {brier:.4f}")
print(f"Accuracy 95% CI: [{ci_low:.4f}, {ci_high:.4f}]")
print(f"Mean PPV (Precision): {np.mean(PPV):.4f}")
print(f"Mean NPV: {np.mean(NPV):.4f}")
```

```
Downloading: "https://download.pytorch.org/models/mobilenet_v2-7ebf99e0.pth" to /root/.cache/torch/hub/checkpoints/mobilenet_v2-7ebf99e0.pth
100%|██████████| 13.6M/13.6M [00:00<00:00, 147MB/s]
```

```
Epoch 1/50 | Train Loss: 0.4002, Acc: 0.8757 | Val Loss: 0.1670, Acc: 0.9417
Epoch 2/50 | Train Loss: 0.1173, Acc: 0.9645 | Val Loss: 0.1031, Acc: 0.9625
Epoch 3/50 | Train Loss: 0.0779, Acc: 0.9745 | Val Loss: 0.0809, Acc: 0.9667
Epoch 4/50 | Train Loss: 0.0492, Acc: 0.9845 | Val Loss: 0.0998, Acc: 0.9583
Epoch 5/50 | Train Loss: 0.0400, Acc: 0.9887 | Val Loss: 0.1372, Acc: 0.9500
Epoch 6/50 | Train Loss: 0.0313, Acc: 0.9901 | Val Loss: 0.1098, Acc: 0.9708
Epoch 7/50 | Train Loss: 0.0319, Acc: 0.9883 | Val Loss: 0.1081, Acc: 0.9750
Epoch 8/50 | Train Loss: 0.0363, Acc: 0.9878 | Val Loss: 0.1265, Acc: 0.9875
Early stopping!
```

Train Report:

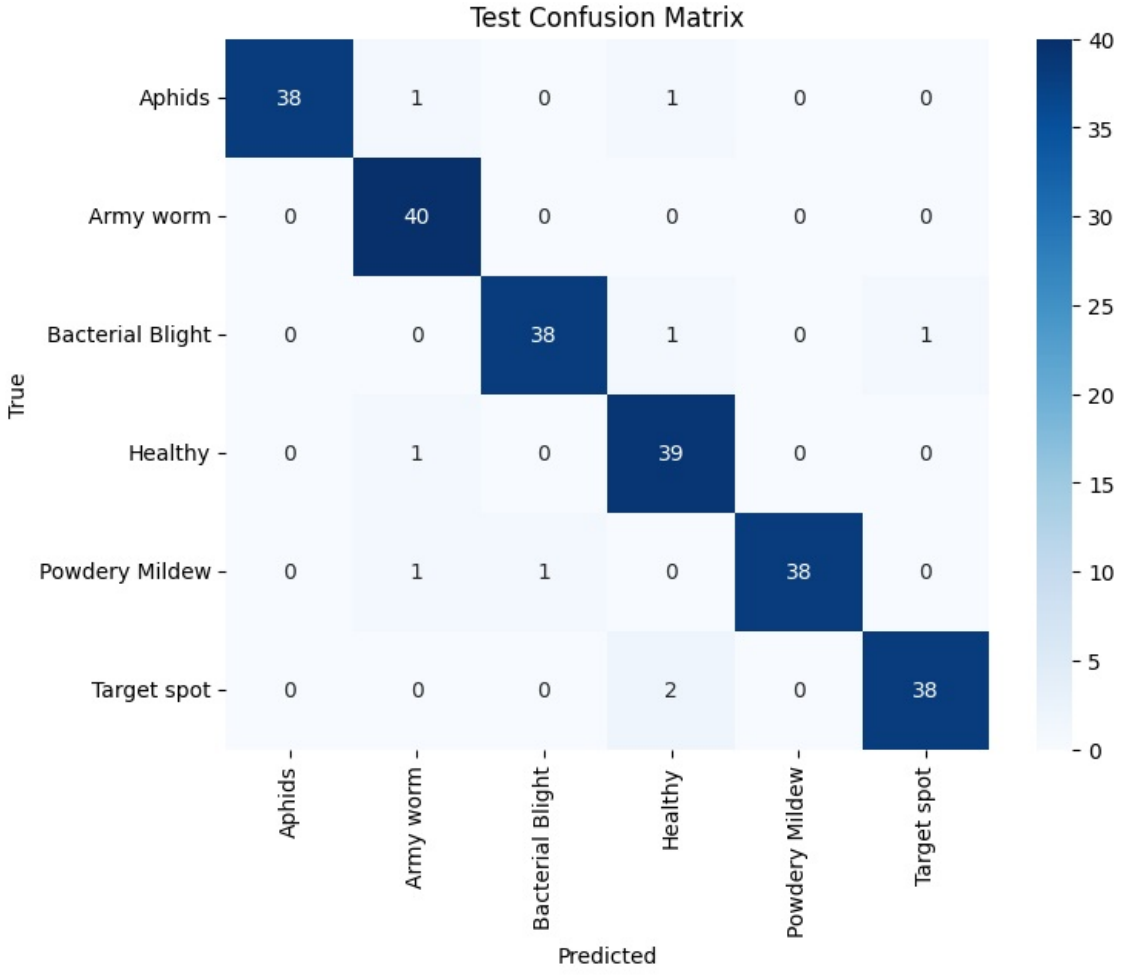
	precision	recall	f1-score	support
Aphids	0.9961	0.9984	0.9973	1280
Army worm	1.0000	0.9977	0.9988	1280
Bacterial Blight	0.9976	0.9875	0.9925	1280
Healthy	0.9969	1.0000	0.9984	1280
Powdery Mildew	0.9992	0.9969	0.9980	1280
Target spot	0.9884	0.9977	0.9930	1280
accuracy			0.9964	7680
macro avg	0.9964	0.9964	0.9964	7680
weighted avg	0.9964	0.9964	0.9964	7680

Validation Report:

	precision	recall	f1-score	support
Aphids	0.9512	0.9750	0.9630	40
Army worm	0.9744	0.9500	0.9620	40
Bacterial Blight	1.0000	0.9500	0.9744	40
Healthy	0.9302	1.0000	0.9639	40
Powdery Mildew	1.0000	1.0000	1.0000	40
Target spot	0.9487	0.9250	0.9367	40
accuracy			0.9667	240
macro avg	0.9674	0.9667	0.9667	240
weighted avg	0.9674	0.9667	0.9667	240

Test Report:

	precision	recall	f1-score	support
Aphids	1.0000	0.9500	0.9744	40
Army worm	0.9302	1.0000	0.9639	40
Bacterial Blight	0.9744	0.9500	0.9620	40
Healthy	0.9070	0.9750	0.9398	40
Powdery Mildew	1.0000	0.9500	0.9744	40
Target spot	0.9744	0.9500	0.9620	40
accuracy			0.9625	240
macro avg	0.9643	0.9625	0.9627	240
weighted avg	0.9643	0.9625	0.9627	240




```
 ROC & PR AUC (per class):
              Class   ROC AUC    PR AUC
0           Aphids  0.998250  0.993011
1        Army worm  0.999750  0.998810
2  Bacterial Blight  0.999500  0.997466
3           Healthy  0.998375  0.993606
4    Powdery Mildew  0.999375  0.997102
5       Target spot  0.994000  0.984904

Cohen's Kappa: 0.9550
Brier Score: 0.0098
Accuracy 95% CI: [0.9375, 0.9833]
Mean PPV (Precision): 0.9643
Mean NPV: 0.9925
```

In [ ]:
```
import os, time, psutil
```

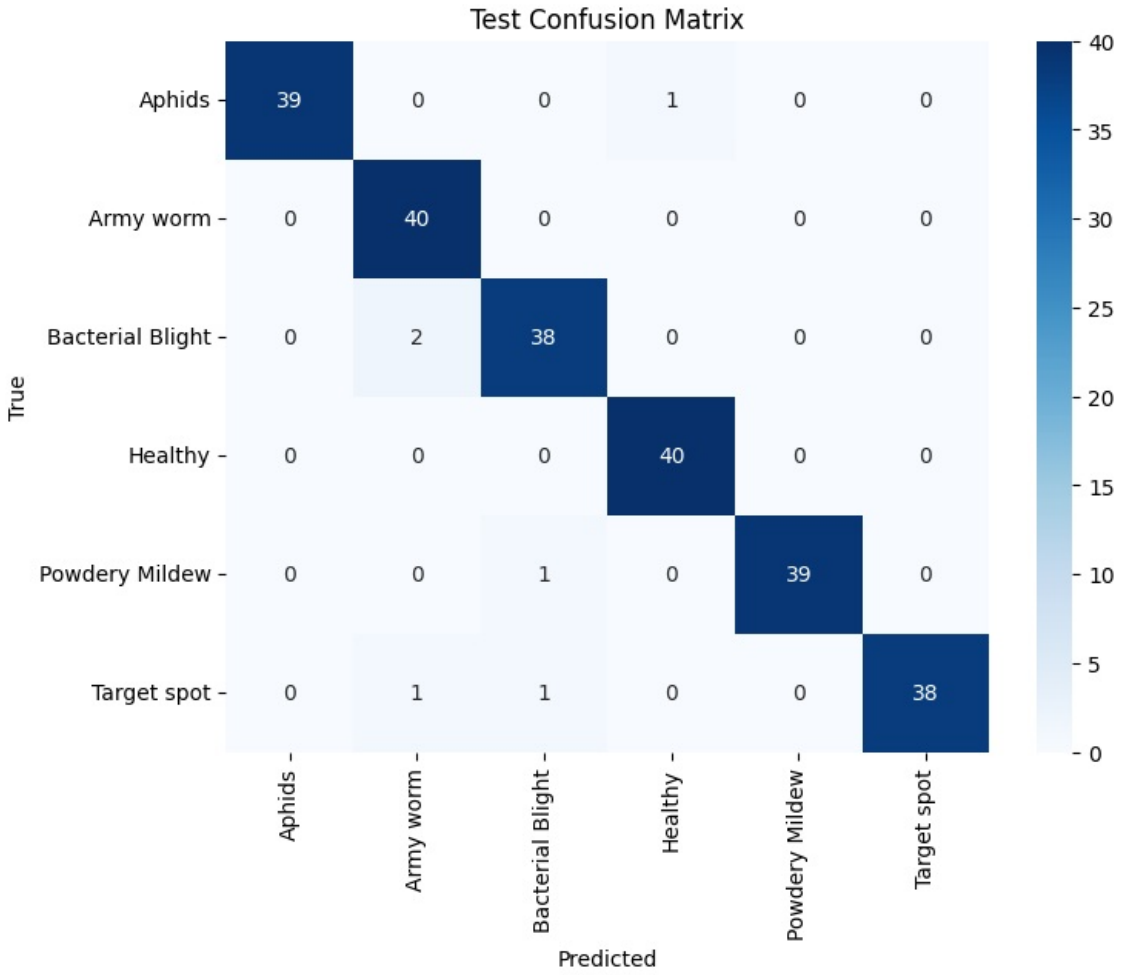
```python
import torch
from torch import nn, optim
from torch.utils.data import DataLoader
from torchvision import datasets, transforms, models
import numpy as np
from sklearn.metrics import classification_report, confusion_matrix, cohen_kappa_score, brier_score_loss
from sklearn.metrics import roc_auc_score, average_precision_score
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# ----------------------
# 1️⃣ Dataset
# ----------------------
transform = transforms.Compose([
    transforms.Resize((224,224)),
    transforms.ToTensor(),
    transforms.Normalize([0.485,0.456,0.406],[0.229,0.224,0.225])
])

train_dataset = datasets.ImageFolder('/kaggle/working/cotton_train_aug', transform=transform)
val_dataset   = datasets.ImageFolder('/kaggle/working/cotton_split/val', transform=transform)
test_dataset  = datasets.ImageFolder('/kaggle/working/cotton_split/test', transform=transform)

train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
val_loader   = DataLoader(val_dataset, batch_size=32, shuffle=False)
test_loader  = DataLoader(test_dataset, batch_size=32, shuffle=False)

class_names = train_dataset.classes
num_classes = len(class_names)

# ----------------------
# 2️⃣ Multi-Head Self-Attention
# ----------------------
class MultiHeadSelfAttention(nn.Module):
    def __init__(self, in_channels, num_heads=4, dropout=0.3):
        super().__init__()
        assert in_channels % num_heads == 0, f"in_channels {in_channels} not divisible by num_heads {num_heads}"
        self.mha = nn.MultiheadAttention(embed_dim=in_channels, num_heads=num_heads, dropout=dropout, batch_firs
        self.dropout = nn.Dropout(dropout)
        self.norm = nn.LayerNorm(in_channels)

    def forward(self, x):
        b, c, h, w = x.size()
        x_flat = x.view(b, c, h*w).permute(0,2,1)   # [B, HW, C]
        attn_out, _ = self.mha(x_flat, x_flat, x_flat)
        attn_out = self.dropout(attn_out)
        attn_out = self.norm(attn_out + x_flat)
        out = attn_out.permute(0,2,1).view(b, c, h, w)
        return out

# ----------------------
# 3️⃣ EfficientNet-B0 + MHSA
# ----------------------
class EfficientNetB0_MHSA(nn.Module):
    def __init__(self, num_classes, num_heads=4, dropout=0.3, freeze_backbone=True):
        super().__init__()
        self.backbone = models.efficientnet_b0(weights=models.EfficientNet_B0_Weights.DEFAULT)
        if freeze_backbone:
            for p in self.backbone.parameters():
                p.requires_grad = False
        # Keep features only
        self.features = self.backbone.features
        last_channels = 1280   # final channels of EfficientNet-B0
        self.mhsa = MultiHeadSelfAttention(in_channels=last_channels, num_heads=num_heads, dropout=dropout)
        self.classifier = nn.Sequential(
            nn.AdaptiveAvgPool2d(1),
            nn.Flatten(),
            nn.Dropout(dropout),
            nn.Linear(last_channels, num_classes)
        )

    def forward(self, x):
        features = self.features(x)
        attn_features = self.mhsa(features)
        out = self.classifier(attn_features)
        return out

# ----------------------
# 4️⃣ Instantiate model, criterion, optimizer
# ----------------------
```

```python
model = EfficientNetB0_MHSA(num_classes=num_classes, num_heads=4, dropout=0.3).to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=1e-4, weight_decay=1e-4)
scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer, 'min', patience=5)

# ----------------------
# 5️⃣ Training loop
# ----------------------
num_epochs = 50
best_val_loss = float('inf')
patience_counter = 0

for epoch in range(num_epochs):
    model.train()
    running_loss, correct, total = 0, 0, 0
    for inputs, labels in train_loader:
        inputs, labels = inputs.to(device), labels.to(device)
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
        _, preds = torch.max(outputs, 1)
        total += labels.size(0)
        correct += (preds == labels).sum().item()

    train_loss = running_loss / len(train_loader)
    train_acc = correct / total

    # Validation
    model.eval()
    val_loss, val_correct, val_total = 0, 0, 0
    with torch.no_grad():
        for inputs, labels in val_loader:
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            val_loss += loss.item()
            _, preds = torch.max(outputs, 1)
            val_total += labels.size(0)
            val_correct += (preds == labels).sum().item()

    val_loss /= len(val_loader)
    val_acc = val_correct / val_total
    scheduler.step(val_loss)

    print(f"Epoch {epoch+1}/{num_epochs} | Train Loss: {train_loss:.4f}, Acc: {train_acc:.4f} | "
          f"Val Loss: {val_loss:.4f}, Acc: {val_acc:.4f}")

    if val_loss < best_val_loss:
        best_val_loss = val_loss
        torch.save(model.state_dict(), "best_efficientnetb0_mhsa.pth")
        patience_counter = 0
    else:
        patience_counter += 1
        if patience_counter >= 5:
            print("Early stopping!")
            break

# ----------------------
# 6️⃣ Load best model
# ----------------------
model.load_state_dict(torch.load("best_efficientnetb0_mhsa.pth"))
model.eval()

# ----------------------
# 7️⃣ Prediction helper
# ----------------------
def get_preds(loader):
    labels_list, preds_list, probs_list = [], [], []
    with torch.no_grad():
        for inputs, labels in loader:
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = model(inputs)
            probs = torch.softmax(outputs, dim=1)
            _, preds = torch.max(outputs, 1)
            labels_list.extend(labels.cpu().numpy())
            preds_list.extend(preds.cpu().numpy())
            probs_list.extend(probs.cpu().numpy())
    return np.array(labels_list), np.array(preds_list), np.array(probs_list)

train_labels, train_preds, train_probs = get_preds(train_loader)
```

```python
val_labels, val_preds, val_probs       = get_preds(val_loader)
test_labels, test_preds, test_probs    = get_preds(test_loader)

# ----------------------
# 8️⃣ Classification Reports
# ----------------------
print("\n Train Report:\n", classification_report(train_labels, train_preds, target_names=class_names, digits=4))
print("\n Validation Report:\n", classification_report(val_labels, val_preds, target_names=class_names, digits=4))
print("\n Test Report:\n", classification_report(test_labels, test_preds, target_names=class_names, digits=4))

# ----------------------
# 9️⃣ Confusion Matrix
# ----------------------
cm = confusion_matrix(test_labels, test_preds)
plt.figure(figsize=(8,6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=class_names, yticklabels=class_names)
plt.xlabel("Predicted"); plt.ylabel("True"); plt.title("Test Confusion Matrix")
plt.show()

# ----------------------
#  ROC & PR AUC
# ----------------------
test_labels_onehot = np.eye(num_classes)[test_labels]
roc_auc_list = [roc_auc_score(test_labels_onehot[:,i], test_probs[:,i]) for i in range(num_classes)]
pr_auc_list  = [average_precision_score(test_labels_onehot[:,i], test_probs[:,i]) for i in range(num_classes)]
print("\n ROC & PR AUC (per class):")
print(pd.DataFrame({"Class": class_names, "ROC AUC": roc_auc_list, "PR AUC": pr_auc_list}))

# ----------------------
# 1️⃣1️⃣ Extra Metrics with PPV/NPV
# ----------------------
kappa = cohen_kappa_score(test_labels, test_preds)
brier = brier_score_loss(test_labels_onehot.flatten(), test_probs.flatten())

# Bootstrap Accuracy CI
n_samples = len(test_labels)
boot_acc = []
rng = np.random.default_rng()
for _ in range(1000):
    idx = rng.integers(0, n_samples, n_samples)
    boot_acc.append(np.mean(test_labels[idx] == test_preds[idx]))
ci_low, ci_high = np.percentile(boot_acc, [2.5, 97.5])

# Confusion matrix components
TP = np.diag(cm)
FP = cm.sum(axis=0) - TP
FN = cm.sum(axis=1) - TP
TN = cm.sum() - (TP + FP + FN)

PPV = TP / (TP + FP + 1e-8)
NPV = TN / (TN + FN + 1e-8)

print(f"\nCohen's Kappa: {kappa:.4f}")
print(f"Brier Score: {brier:.4f}")
print(f"Accuracy 95% CI: [{ci_low:.4f}, {ci_high:.4f}]")
print(f"Mean PPV (Precision): {np.mean(PPV):.4f}")
print(f"Mean NPV: {np.mean(NPV):.4f}")
```

```
Downloading: "https://download.pytorch.org/models/efficientnet_b0_rwightman-7f5810bc.pth" to /root/.cache/torch/hub/checkpoints/efficientnet_b0_rwightman-7f5810bc.pth
100%|██████████| 20.5M/20.5M [00:00<00:00, 167MB/s]
```

```
Epoch 1/50 | Train Loss: 0.4245, Acc: 0.8660 | Val Loss: 0.1130, Acc: 0.9708
Epoch 2/50 | Train Loss: 0.1365, Acc: 0.9549 | Val Loss: 0.0865, Acc: 0.9875
Epoch 3/50 | Train Loss: 0.0734, Acc: 0.9767 | Val Loss: 0.0550, Acc: 0.9875
Epoch 4/50 | Train Loss: 0.0559, Acc: 0.9832 | Val Loss: 0.0951, Acc: 0.9833
Epoch 5/50 | Train Loss: 0.0387, Acc: 0.9883 | Val Loss: 0.0772, Acc: 0.9792
Epoch 6/50 | Train Loss: 0.0321, Acc: 0.9897 | Val Loss: 0.0358, Acc: 0.9958
Epoch 7/50 | Train Loss: 0.0343, Acc: 0.9887 | Val Loss: 0.0805, Acc: 0.9833
Epoch 8/50 | Train Loss: 0.0273, Acc: 0.9917 | Val Loss: 0.0664, Acc: 0.9792
Epoch 9/50 | Train Loss: 0.0240, Acc: 0.9917 | Val Loss: 0.0174, Acc: 0.9958
Epoch 10/50 | Train Loss: 0.0285, Acc: 0.9909 | Val Loss: 0.0997, Acc: 0.9792
Epoch 11/50 | Train Loss: 0.0224, Acc: 0.9932 | Val Loss: 0.0908, Acc: 0.9667
Epoch 12/50 | Train Loss: 0.0179, Acc: 0.9952 | Val Loss: 0.0596, Acc: 0.9875
Epoch 13/50 | Train Loss: 0.0196, Acc: 0.9940 | Val Loss: 0.0716, Acc: 0.9792
Epoch 14/50 | Train Loss: 0.0218, Acc: 0.9926 | Val Loss: 0.1059, Acc: 0.9792
Early stopping!

 Train Report:
                   precision    recall  f1-score   support

          Aphids     1.0000    0.9984    0.9992      1280
       Army worm     1.0000    1.0000    1.0000      1280
Bacterial Blight     0.9977    1.0000    0.9988      1280

         Healthy     1.0000    1.0000    1.0000      1280
  Powdery Mildew     0.9984    1.0000    0.9992      1280
     Target spot     1.0000    0.9977    0.9988      1280

        accuracy                         0.9993      7680
       macro avg     0.9994    0.9993    0.9993      7680
    weighted avg     0.9994    0.9993    0.9993      7680


 Validation Report:
                   precision    recall  f1-score   support

          Aphids     1.0000    1.0000    1.0000        40
       Army worm     0.9756    1.0000    0.9877        40
Bacterial Blight     1.0000    0.9750    0.9873        40
         Healthy     1.0000    1.0000    1.0000        40
  Powdery Mildew     1.0000    1.0000    1.0000        40
     Target spot     1.0000    1.0000    1.0000        40

        accuracy                         0.9958       240
       macro avg     0.9959    0.9958    0.9958       240
    weighted avg     0.9959    0.9958    0.9958       240


 Test Report:
                   precision    recall  f1-score   support

          Aphids     1.0000    0.9750    0.9873        40
       Army worm     0.9302    1.0000    0.9639        40
Bacterial Blight     0.9500    0.9500    0.9500        40
         Healthy     0.9756    1.0000    0.9877        40
  Powdery Mildew     1.0000    0.9750    0.9873        40
     Target spot     1.0000    0.9500    0.9744        40

        accuracy                         0.9750       240
       macro avg     0.9760    0.9750    0.9751       240
    weighted avg     0.9760    0.9750    0.9751       240
```




```
 ROC & PR AUC (per class):
              Class   ROC AUC    PR AUC
0            Aphids  0.999375  0.997222
1         Army worm  0.999750  0.998765
2  Bacterial Blight  0.997500  0.988376
3           Healthy  1.000000  1.000000
4    Powdery Mildew  1.000000  1.000000
5       Target spot  0.998000  0.991604

Cohen's Kappa: 0.9700
Brier Score: 0.0082
Accuracy 95% CI: [0.9542, 0.9918]
Mean PPV (Precision): 0.9760
Mean NPV: 0.9950
```

In [ ]:

```
# ----------------------
# IMPORTS
# ----------------------
import os, time, psutil
import torch
from torch import nn, optim
from torch.utils.data import DataLoader
```

```python
from torchvision import datasets, transforms, models
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import classification_report, confusion_matrix, cohen_kappa_score, brier_score_loss
from sklearn.metrics import roc_auc_score, average_precision_score
from scipy.stats import ttest_ind

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# ----------------------
# DATASET
# ----------------------
train_transform_aug = transforms.Compose([
    transforms.Resize((224,224)),
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(15),
    transforms.ToTensor(),
    transforms.Normalize([0.485,0.456,0.406],[0.229,0.224,0.225])
])

eval_transform = transforms.Compose([
    transforms.Resize((224,224)),
    transforms.ToTensor(),
    transforms.Normalize([0.485,0.456,0.406],[0.229,0.224,0.225])
])

train_dataset = datasets.ImageFolder('/kaggle/working/cotton_train_aug', transform=train_transform_aug)
train_eval_dataset = datasets.ImageFolder('/kaggle/working/cotton_train_aug', transform=eval_transform)
val_dataset   = datasets.ImageFolder('/kaggle/working/cotton_split/val', transform=eval_transform)
test_dataset  = datasets.ImageFolder('/kaggle/working/cotton_split/test', transform=eval_transform)

train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
train_eval_loader = DataLoader(train_eval_dataset, batch_size=32, shuffle=False)
val_loader   = DataLoader(val_dataset, batch_size=32, shuffle=False)
test_loader  = DataLoader(test_dataset, batch_size=32, shuffle=False)

class_names = train_dataset.classes
num_classes = len(class_names)

# ----------------------
# MULTI-HEAD SELF-ATTENTION
# ----------------------
class MultiHeadSelfAttention(nn.Module):
    def __init__(self, in_channels, num_heads=4, dropout=0.3):
        super().__init__()
        assert in_channels % num_heads == 0
        self.mha = nn.MultiheadAttention(embed_dim=in_channels, num_heads=num_heads, dropout=dropout, batch_firs
        self.dropout = nn.Dropout(dropout)
        self.norm = nn.LayerNorm(in_channels)

    def forward(self, x):
        b, c, h, w = x.size()
        x_flat = x.view(b, c, h*w).permute(0,2,1)
        attn_out, _ = self.mha(x_flat, x_flat, x_flat)
        attn_out = self.dropout(attn_out)
        attn_out = self.norm(attn_out + x_flat)
        out = attn_out.permute(0,2,1).view(b, c, h, w)
        return out

# ----------------------
# MODEL WRAPPERS
# ----------------------
class SqueezeNet_MHSA(nn.Module):
    def __init__(self, num_classes, num_heads=4, dropout=0.3, freeze_backbone=True):
        super().__init__()
        self.backbone = models.squeezenet1_1(weights=models.SqueezeNet1_1_Weights.DEFAULT)
        if freeze_backbone:
            for p in self.backbone.parameters(): p.requires_grad = False
        self.features = self.backbone.features
        last_channels = 512
        self.mhsa = MultiHeadSelfAttention(last_channels, num_heads=num_heads, dropout=dropout)
        self.classifier = nn.Sequential(
            nn.AdaptiveAvgPool2d(1),
            nn.Flatten(),
            nn.Dropout(dropout),
            nn.Linear(last_channels, num_classes)
        )
    def forward(self, x):
        f = self.features(x)
        f = self.mhsa(f)
        return self.classifier(f)
```

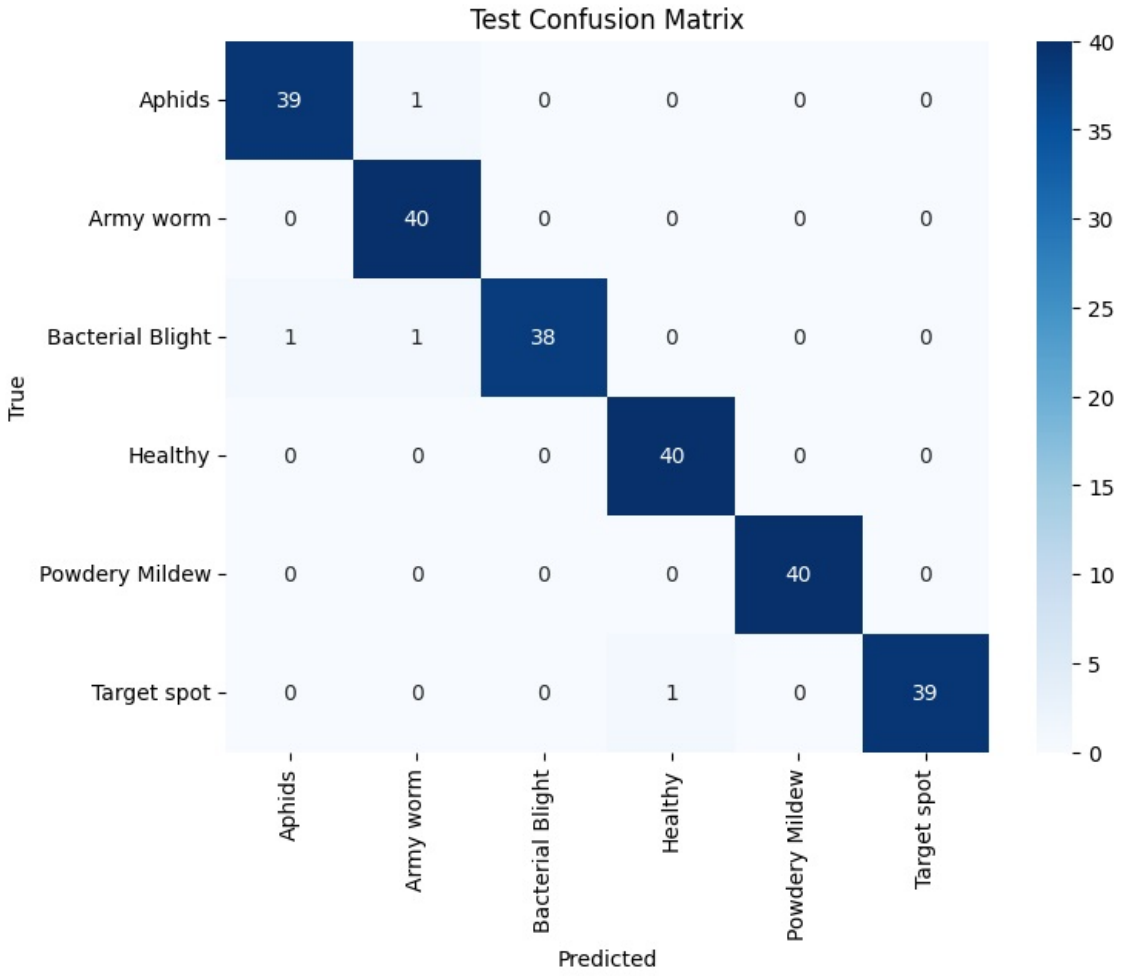
```python
class ShuffleNetV2_MHSA(nn.Module):
    def __init__(self, num_classes, num_heads=4, dropout=0.3, freeze_backbone=True):
        super().__init__()
        self.backbone = models.shufflenet_v2_x1_0(weights=models.ShuffleNet_V2_X1_0_Weights.DEFAULT)
        if freeze_backbone:
            for p in self.backbone.parameters(): p.requires_grad = False
        self.backbone.fc = nn.Identity()
        self.mhsa = MultiHeadSelfAttention(1024, num_heads=num_heads, dropout=dropout)
        self.classifier = nn.Sequential(
            nn.AdaptiveAvgPool2d(1),
            nn.Flatten(),
            nn.Dropout(dropout),
            nn.Linear(1024, num_classes)
        )
    def forward(self, x):
        f = self.backbone(x).unsqueeze(-1).unsqueeze(-1)
        f = self.mhsa(f)
        return self.classifier(f)

class EfficientNetB0_MHSA(nn.Module):
    def __init__(self, num_classes, num_heads=4, dropout=0.3, freeze_backbone=True):
        super().__init__()
        self.backbone = models.efficientnet_b0(weights=models.EfficientNet_B0_Weights.DEFAULT)
        if freeze_backbone:
            for p in self.backbone.parameters(): p.requires_grad = False
        self.backbone.classifier = nn.Identity()
        self.mhsa = MultiHeadSelfAttention(1280, num_heads=num_heads, dropout=dropout)
        self.classifier = nn.Sequential(
            nn.AdaptiveAvgPool2d(1),
            nn.Flatten(),
            nn.Dropout(dropout),
            nn.Linear(1280, num_classes)
        )
    def forward(self, x):
        f = self.backbone.features(x)
        f = self.mhsa(f)
        return self.classifier(f)

# ----------------------
# INSTANTIATE MODELS
# ----------------------
squeezenet_model = SqueezeNet_MHSA(num_classes=num_classes).to(device)
shufflenet_model = ShuffleNetV2_MHSA(num_classes=num_classes).to(device)
efficientnet_model = EfficientNetB0_MHSA(num_classes=num_classes).to(device)

models_list = [squeezenet_model, shufflenet_model, efficientnet_model]

criterion = nn.CrossEntropyLoss()
optimizer_list = [optim.Adam(m.parameters(), lr=1e-4, weight_decay=1e-4) for m in models_list]
scheduler_list = [optim.lr_scheduler.ReduceLROnPlateau(opt, 'min', patience=5) for opt in optimizer_list]

# ----------------------
# TRAINING LOOP
# ----------------------
num_epochs = 50
for idx, model in enumerate(models_list):
    best_val_loss = float('inf'); patience_counter = 0
    optimizer = optimizer_list[idx]; scheduler = scheduler_list[idx]
    for epoch in range(num_epochs):
        model.train()
        running_loss, correct, total = 0,0,0
        for inputs, labels in train_loader:
            inputs, labels = inputs.to(device), labels.to(device)
            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
            running_loss += loss.item()
            _, preds = torch.max(outputs,1)
            correct += (preds==labels).sum().item()
            total += labels.size(0)
        train_loss = running_loss / len(train_loader)
        train_acc = correct / total

        # Validation
        model.eval()
        val_loss, val_correct, val_total = 0,0,0
        with torch.no_grad():
            for inputs, labels in val_loader:
                inputs, labels = inputs.to(device), labels.to(device)
                outputs = model(inputs)
```

```
                loss = criterion(outputs, labels)
                val_loss += loss.item()
                _, preds = torch.max(outputs,1)
                val_correct += (preds==labels).sum().item()
                val_total += labels.size(0)
        val_loss /= len(val_loader); val_acc=val_correct/val_total
        scheduler.step(val_loss)

        print(f"[Model {idx+1}] Epoch {epoch+1}/{num_epochs} | Train Loss: {train_loss:.4f}, Acc: {train_acc:.4f

        if val_loss<best_val_loss:
            best_val_loss=val_loss
            torch.save(model.state_dict(), f"best_model_{idx}.pth")
            patience_counter=0
        else:
            patience_counter+=1
            if patience_counter>=5:
                print("Early stopping!")
                break

# ----------------------
# ENSEMBLE PREDICTION
# ----------------------
def get_preds_ensemble(loader, models_list):
    probs_list=[]
    with torch.no_grad():
        for inputs, _ in loader:
            inputs = inputs.to(device)
            outputs_sum = 0
            for model in models_list:
                model.eval()
                outputs_sum += torch.softmax(model(inputs),1)
            probs_avg = outputs_sum / len(models_list)
            probs_list.append(probs_avg.cpu().numpy())
    probs_all = np.vstack(probs_list)
    preds_all = np.argmax(probs_all, axis=1)
    labels_all = np.array([y for _, y in loader.dataset])
    return labels_all, preds_all, probs_all

train_labels, train_preds, train_probs = get_preds_ensemble(train_eval_loader, models_list)
val_labels, val_preds, val_probs       = get_preds_ensemble(val_loader, models_list)
test_labels, test_preds, test_probs    = get_preds_ensemble(test_loader, models_list)

# ----------------------
# METRICS & EVALUATION
# ----------------------
print("\n Train Classification Report:")
print(classification_report(train_labels, train_preds, target_names=class_names, digits=4))
print("\n Validation Classification Report:")
print(classification_report(val_labels, val_preds, target_names=class_names, digits=4))
print("\n Test Classification Report:")
print(classification_report(test_labels, test_preds, target_names=class_names, digits=4))

# Confusion Matrix
cm = confusion_matrix(test_labels, test_preds)
plt.figure(figsize=(8,6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=class_names, yticklabels=class_names)
plt.xlabel("Predicted"); plt.ylabel("True"); plt.title("Test Confusion Matrix")
plt.show()

# ROC & PR AUC
test_labels_onehot = np.eye(num_classes)[test_labels]
roc_auc_list = [roc_auc_score(test_labels_onehot[:,i], test_probs[:,i]) for i in range(num_classes)]
pr_auc_list  = [average_precision_score(test_labels_onehot[:,i], test_probs[:,i]) for i in range(num_classes)]
auc_df = pd.DataFrame({"Class": class_names, "ROC AUC":[f"{x:.4f}" for x in roc_auc_list], "PR AUC":[f"{x:.4f}"
print("\n Class-wise ROC & PR AUC:")
print(auc_df)

# Bootstrap CI (corrected)
n_samples = len(test_labels)
boot_acc = []
for _ in range(1000):
    idx = np.random.choice(n_samples, n_samples, replace=True)
    boot_acc.append(np.mean(test_labels[idx]==test_preds[idx]))
ci_low, ci_high = np.percentile(boot_acc, [2.5,97.5])
print(f"\nAccuracy 95% CI: [{ci_low:.4f}, {ci_high:.4f}]")

# Extra Metrics
kappa = cohen_kappa_score(test_labels, test_preds)
brier = brier_score_loss(test_labels_onehot.flatten(), test_probs.flatten())
TP = np.diag(cm); FP = cm.sum(axis=0)-TP; FN = cm.sum(axis=1)-TP; TN = cm.sum()-(TP+FP+FN)
PPV = TP/(TP+FP+1e-8); NPV = TN/(TN+FN+1e-8)
ttest_res = ttest_ind(test_probs[:,0], test_probs[:,1])
```

```python
print(f"Cohen's Kappa: {kappa:.4f}")
print(f"Brier Score: {brier:.4f}")
print(f"Mean PPV: {np.mean(PPV):.4f} | Mean NPV: {np.mean(NPV):.4f}")
print(f"T-test class0 vs class1: stat={ttest_res.statistic:.4f}, p={ttest_res.pvalue:.4f}")

# ----------------------
# SAVE ENSEMBLE FOR DEPLOYMENT
# ----------------------
torch.save({
    'squeezenet': squeezenet_model.state_dict(),
    'shufflenet': shufflenet_model.state_dict(),
    'efficientnet': efficientnet_model.state_dict(),
    'class_names': class_names
}, "BEST.pth")
print("\n Ensemble saved as 'BEST.pth' for deployment.")
```

```
[Model 1] Epoch 1/50 | Train Loss: 0.8633, Acc: 0.7195 | Val Loss: 0.4677, Acc: 0.8583
[Model 1] Epoch 2/50 | Train Loss: 0.3243, Acc: 0.8997 | Val Loss: 0.3150, Acc: 0.8833
[Model 1] Epoch 3/50 | Train Loss: 0.2015, Acc: 0.9424 | Val Loss: 0.2363, Acc: 0.9125
[Model 1] Epoch 4/50 | Train Loss: 0.1430, Acc: 0.9561 | Val Loss: 0.1824, Acc: 0.9375
[Model 1] Epoch 5/50 | Train Loss: 0.0990, Acc: 0.9727 | Val Loss: 0.1842, Acc: 0.9333
[Model 1] Epoch 6/50 | Train Loss: 0.0857, Acc: 0.9754 | Val Loss: 0.1685, Acc: 0.9458
[Model 1] Epoch 7/50 | Train Loss: 0.0620, Acc: 0.9835 | Val Loss: 0.1882, Acc: 0.9375
[Model 1] Epoch 8/50 | Train Loss: 0.0509, Acc: 0.9858 | Val Loss: 0.1897, Acc: 0.9500
[Model 1] Epoch 9/50 | Train Loss: 0.0445, Acc: 0.9889 | Val Loss: 0.1645, Acc: 0.9583
[Model 1] Epoch 10/50 | Train Loss: 0.0350, Acc: 0.9908 | Val Loss: 0.1864, Acc: 0.9708
[Model 1] Epoch 11/50 | Train Loss: 0.0342, Acc: 0.9902 | Val Loss: 0.1793, Acc: 0.9667
[Model 1] Epoch 12/50 | Train Loss: 0.0259, Acc: 0.9949 | Val Loss: 0.1709, Acc: 0.9583
[Model 1] Epoch 13/50 | Train Loss: 0.0247, Acc: 0.9941 | Val Loss: 0.1891, Acc: 0.9708
[Model 1] Epoch 14/50 | Train Loss: 0.0235, Acc: 0.9932 | Val Loss: 0.1559, Acc: 0.9625
[Model 1] Epoch 15/50 | Train Loss: 0.0226, Acc: 0.9949 | Val Loss: 0.1840, Acc: 0.9667
[Model 1] Epoch 16/50 | Train Loss: 0.0207, Acc: 0.9953 | Val Loss: 0.1665, Acc: 0.9625
[Model 1] Epoch 17/50 | Train Loss: 0.0153, Acc: 0.9960 | Val Loss: 0.1731, Acc: 0.9750
[Model 1] Epoch 18/50 | Train Loss: 0.0167, Acc: 0.9948 | Val Loss: 0.1922, Acc: 0.9708
[Model 1] Epoch 19/50 | Train Loss: 0.0145, Acc: 0.9965 | Val Loss: 0.1948, Acc: 0.9542
Early stopping!
[Model 2] Epoch 1/50 | Train Loss: 0.6246, Acc: 0.7802 | Val Loss: 0.2455, Acc: 0.9375
[Model 2] Epoch 2/50 | Train Loss: 0.2647, Acc: 0.9163 | Val Loss: 0.1596, Acc: 0.9542
[Model 2] Epoch 3/50 | Train Loss: 0.2111, Acc: 0.9316 | Val Loss: 0.1609, Acc: 0.9667
[Model 2] Epoch 4/50 | Train Loss: 0.1891, Acc: 0.9385 | Val Loss: 0.1456, Acc: 0.9625
[Model 2] Epoch 5/50 | Train Loss: 0.1546, Acc: 0.9461 | Val Loss: 0.1513, Acc: 0.9708
[Model 2] Epoch 6/50 | Train Loss: 0.1371, Acc: 0.9516 | Val Loss: 0.1295, Acc: 0.9583
[Model 2] Epoch 7/50 | Train Loss: 0.1309, Acc: 0.9561 | Val Loss: 0.1709, Acc: 0.9542
[Model 2] Epoch 8/50 | Train Loss: 0.1346, Acc: 0.9578 | Val Loss: 0.2130, Acc: 0.9458
[Model 2] Epoch 9/50 | Train Loss: 0.1231, Acc: 0.9579 | Val Loss: 0.1624, Acc: 0.9625
[Model 2] Epoch 10/50 | Train Loss: 0.1222, Acc: 0.9576 | Val Loss: 0.1696, Acc: 0.9625
[Model 2] Epoch 11/50 | Train Loss: 0.1094, Acc: 0.9628 | Val Loss: 0.1891, Acc: 0.9500
Early stopping!
[Model 3] Epoch 1/50 | Train Loss: 0.4513, Acc: 0.8526 | Val Loss: 0.1698, Acc: 0.9417
[Model 3] Epoch 2/50 | Train Loss: 0.1682, Acc: 0.9462 | Val Loss: 0.1379, Acc: 0.9625
[Model 3] Epoch 3/50 | Train Loss: 0.1107, Acc: 0.9612 | Val Loss: 0.0919, Acc: 0.9708
[Model 3] Epoch 4/50 | Train Loss: 0.0884, Acc: 0.9685 | Val Loss: 0.0652, Acc: 0.9792
[Model 3] Epoch 5/50 | Train Loss: 0.0710, Acc: 0.9740 | Val Loss: 0.0479, Acc: 0.9833
[Model 3] Epoch 6/50 | Train Loss: 0.0621, Acc: 0.9805 | Val Loss: 0.0909, Acc: 0.9792
[Model 3] Epoch 7/50 | Train Loss: 0.0644, Acc: 0.9777 | Val Loss: 0.0592, Acc: 0.9792
[Model 3] Epoch 8/50 | Train Loss: 0.0505, Acc: 0.9837 | Val Loss: 0.0651, Acc: 0.9792
[Model 3] Epoch 9/50 | Train Loss: 0.0453, Acc: 0.9848 | Val Loss: 0.0697, Acc: 0.9667
[Model 3] Epoch 10/50 | Train Loss: 0.0549, Acc: 0.9801 | Val Loss: 0.0474, Acc: 0.9833
[Model 3] Epoch 11/50 | Train Loss: 0.0477, Acc: 0.9842 | Val Loss: 0.0610, Acc: 0.9875
[Model 3] Epoch 12/50 | Train Loss: 0.0329, Acc: 0.9883 | Val Loss: 0.0433, Acc: 0.9792
[Model 3] Epoch 13/50 | Train Loss: 0.0321, Acc: 0.9896 | Val Loss: 0.0515, Acc: 0.9833
[Model 3] Epoch 14/50 | Train Loss: 0.0334, Acc: 0.9891 | Val Loss: 0.0482, Acc: 0.9792
[Model 3] Epoch 15/50 | Train Loss: 0.0365, Acc: 0.9871 | Val Loss: 0.0729, Acc: 0.9625
[Model 3] Epoch 16/50 | Train Loss: 0.0321, Acc: 0.9891 | Val Loss: 0.0456, Acc: 0.9750
[Model 3] Epoch 17/50 | Train Loss: 0.0331, Acc: 0.9884 | Val Loss: 0.0753, Acc: 0.9833
Early stopping!

 Train Classification Report:
                  precision    recall  f1-score   support

          Aphids     1.0000    1.0000    1.0000      1280
       Army worm     1.0000    1.0000    1.0000      1280
Bacterial Blight     0.9977    0.9992    0.9984      1280
         Healthy     1.0000    1.0000    1.0000      1280
  Powdery Mildew     0.9992    1.0000    0.9996      1280
     Target spot     1.0000    0.9977    0.9988      1280

        accuracy                         0.9995      7680
       macro avg     0.9995    0.9995    0.9995      7680
    weighted avg     0.9995    0.9995    0.9995      7680


 Validation Classification Report:
```

```
                  precision    recall  f1-score   support

          Aphids     1.0000    0.9500    0.9744        40
       Army worm     1.0000    1.0000    1.0000        40
Bacterial Blight     0.9750    0.9750    0.9750        40
         Healthy     1.0000    1.0000    1.0000        40
  Powdery Mildew     0.9756    1.0000    0.9877        40
     Target spot     0.9512    0.9750    0.9630        40

        accuracy                         0.9833       240
       macro avg     0.9836    0.9833    0.9833       240
    weighted avg     0.9836    0.9833    0.9833       240


 Test Classification Report:
                  precision    recall  f1-score   support

          Aphids     0.9750    0.9750    0.9750        40
       Army worm     0.9524    1.0000    0.9756        40
Bacterial Blight     1.0000    0.9500    0.9744        40
         Healthy     0.9756    1.0000    0.9877        40
  Powdery Mildew     1.0000    1.0000    1.0000        40
     Target spot     1.0000    0.9750    0.9873        40

        accuracy                         0.9833       240
       macro avg     0.9838    0.9833    0.9833       240
    weighted avg     0.9838    0.9833    0.9833       240
```




```
 Class-wise ROC & PR AUC:
              Class ROC AUC  PR AUC
0           Aphids  0.9999  0.9994
1        Army worm  1.0000  1.0000
2  Bacterial Blight  1.0000  1.0000
3          Healthy  0.9998  0.9988
4   Powdery Mildew  1.0000  1.0000
5      Target spot  0.9975  0.9917

Accuracy 95% CI: [0.9667, 0.9958]
Cohen's Kappa: 0.9800
Brier Score: 0.0051
Mean PPV: 0.9838 | Mean NPV: 0.9967
T-test class0 vs class1: stat=-0.2739, p=0.7843

 Ensemble saved as 'BEST.pth' for deployment.
```

In [ ]:
```
!pip install grad-cam
```

```
Collecting grad-cam
  Downloading grad-cam-1.5.5.tar.gz (7.8 MB)
     7.8/7.8 MB 58.8 MB/s eta 0:00:0000:0100:01
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Preparing metadata (pyproject.toml) ... done
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (from grad-cam) (1.26.4)
Requirement already satisfied: Pillow in /usr/local/lib/python3.11/dist-packages (from grad-cam) (11.2.1)
Requirement already satisfied: torch>=1.7.1 in /usr/local/lib/python3.11/dist-packages (from grad-cam) (2.6.0+cu
124)
Requirement already satisfied: torchvision>=0.8.2 in /usr/local/lib/python3.11/dist-packages (from grad-cam) (0.
21.0+cu124)
Collecting ttach (from grad-cam)
  Downloading ttach-0.0.3-py3-none-any.whl.metadata (5.2 kB)
Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-packages (from grad-cam) (4.67.1)
Requirement already satisfied: opencv-python in /usr/local/lib/python3.11/dist-packages (from grad-cam) (4.11.0.
86)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.11/dist-packages (from grad-cam) (3.7.2)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.11/dist-packages (from grad-cam) (1.2.2)
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-packages (from torch>=1.7.1->grad-cam)
(3.18.0)
Requirement already satisfied: typing-extensions>=4.10.0 in /usr/local/lib/python3.11/dist-packages (from torch>
=1.7.1->grad-cam) (4.14.0)
Requirement already satisfied: networkx in /usr/local/lib/python3.11/dist-packages (from torch>=1.7.1->grad-cam)
(3.5)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.11/dist-packages (from torch>=1.7.1->grad-cam) (
3.1.6)
Requirement already satisfied: fsspec in /usr/local/lib/python3.11/dist-packages (from torch>=1.7.1->grad-cam) (
2025.5.1)
Collecting nvidia-cuda-nvrtc-cu12==12.4.127 (from torch>=1.7.1->grad-cam)
  Downloading nvidia_cuda_nvrtc_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl.metadata (1.5 kB)
Collecting nvidia-cuda-runtime-cu12==12.4.127 (from torch>=1.7.1->grad-cam)
  Downloading nvidia_cuda_runtime_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl.metadata (1.5 kB)
Collecting nvidia-cuda-cupti-cu12==12.4.127 (from torch>=1.7.1->grad-cam)
  Downloading nvidia_cuda_cupti_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl.metadata (1.6 kB)
Collecting nvidia-cudnn-cu12==9.1.0.70 (from torch>=1.7.1->grad-cam)
  Downloading nvidia_cudnn_cu12-9.1.0.70-py3-none-manylinux2014_x86_64.whl.metadata (1.6 kB)
Collecting nvidia-cublas-cu12==12.4.5.8 (from torch>=1.7.1->grad-cam)
  Downloading nvidia_cublas_cu12-12.4.5.8-py3-none-manylinux2014_x86_64.whl.metadata (1.5 kB)
Collecting nvidia-cufft-cu12==11.2.1.3 (from torch>=1.7.1->grad-cam)
  Downloading nvidia_cufft_cu12-11.2.1.3-py3-none-manylinux2014_x86_64.whl.metadata (1.5 kB)
Collecting nvidia-curand-cu12==10.3.5.147 (from torch>=1.7.1->grad-cam)
  Downloading nvidia_curand_cu12-10.3.5.147-py3-none-manylinux2014_x86_64.whl.metadata (1.5 kB)
Collecting nvidia-cusolver-cu12==11.6.1.9 (from torch>=1.7.1->grad-cam)
  Downloading nvidia_cusolver_cu12-11.6.1.9-py3-none-manylinux2014_x86_64.whl.metadata (1.6 kB)
Collecting nvidia-cusparse-cu12==12.3.1.170 (from torch>=1.7.1->grad-cam)
  Downloading nvidia_cusparse_cu12-12.3.1.170-py3-none-manylinux2014_x86_64.whl.metadata (1.6 kB)
Requirement already satisfied: nvidia-cusparselt-cu12==0.6.2 in /usr/local/lib/python3.11/dist-packages (from to
rch>=1.7.1->grad-cam) (0.6.2)
Requirement already satisfied: nvidia-nccl-cu12==2.21.5 in /usr/local/lib/python3.11/dist-packages (from torch>=
1.7.1->grad-cam) (2.21.5)
Requirement already satisfied: nvidia-nvtx-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (from torch
>=1.7.1->grad-cam) (12.4.127)
Collecting nvidia-nvjitlink-cu12==12.4.127 (from torch>=1.7.1->grad-cam)
  Downloading nvidia_nvjitlink_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl.metadata (1.5 kB)
Requirement already satisfied: triton==3.2.0 in /usr/local/lib/python3.11/dist-packages (from torch>=1.7.1->grad
-cam) (3.2.0)
Requirement already satisfied: sympy==1.13.1 in /usr/local/lib/python3.11/dist-packages (from torch>=1.7.1->grad
-cam) (1.13.1)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.11/dist-packages (from sympy==1.13.1
->torch>=1.7.1->grad-cam) (1.3.0)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib->gra
d-cam) (1.3.2)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-packages (from matplotlib->grad-ca
m) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib->gr
ad-cam) (4.58.4)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib->gr
ad-cam) (1.4.8)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib->grad
-cam) (25.0)
Requirement already satisfied: pyparsing<3.1,>=2.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib
->grad-cam) (3.0.9)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.11/dist-packages (from matplotlib-
>grad-cam) (2.9.0.post0)
Requirement already satisfied: mkl_fft in /usr/local/lib/python3.11/dist-packages (from numpy->grad-cam) (1.3.8)
Requirement already satisfied: mkl_random in /usr/local/lib/python3.11/dist-packages (from numpy->grad-cam) (1.2
.4)
Requirement already satisfied: mkl_umath in /usr/local/lib/python3.11/dist-packages (from numpy->grad-cam) (0.1.
1)
Requirement already satisfied: mkl in /usr/local/lib/python3.11/dist-packages (from numpy->grad-cam) (2025.2.0)
```

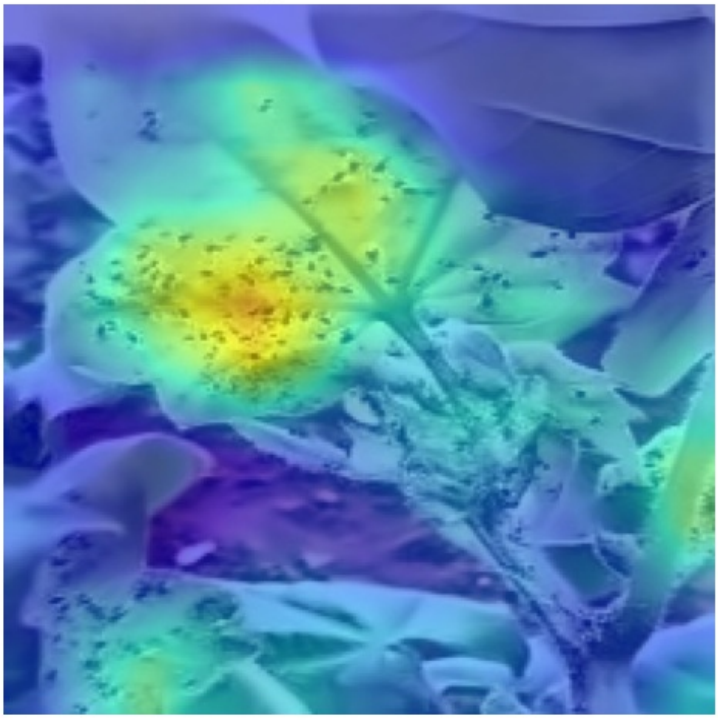
```
Requirement already satisfied: tbb4py in /usr/local/lib/python3.11/dist-packages (from numpy->grad-cam) (2022.2.
0)
Requirement already satisfied: mkl-service in /usr/local/lib/python3.11/dist-packages (from numpy->grad-cam) (2.
4.1)
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.11/dist-packages (from scikit-learn->grad-
cam) (1.15.3)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.11/dist-packages (from scikit-learn->grad
-cam) (1.5.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.11/dist-packages (from scikit-lear
n->grad-cam) (3.6.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.7->m
atplotlib->grad-cam) (1.17.0)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.11/dist-packages (from jinja2->torch>=1
.7.1->grad-cam) (3.0.2)
Requirement already satisfied: intel-openmp<2026,>=2024 in /usr/local/lib/python3.11/dist-packages (from mkl->nu
mpy->grad-cam) (2024.2.0)
Requirement already satisfied: tbb==2022.* in /usr/local/lib/python3.11/dist-packages (from mkl->numpy->grad-cam
) (2022.2.0)
Requirement already satisfied: tcmlib==1.* in /usr/local/lib/python3.11/dist-packages (from tbb==2022.*->mkl->nu
mpy->grad-cam) (1.4.0)
Requirement already satisfied: intel-cmplr-lib-rt in /usr/local/lib/python3.11/dist-packages (from mkl_umath->nu
mpy->grad-cam) (2024.2.0)
Requirement already satisfied: intel-cmplr-lib-ur==2024.2.0 in /usr/local/lib/python3.11/dist-packages (from int
el-openmp<2026,>=2024->mkl->numpy->grad-cam) (2024.2.0)
Downloading nvidia_cublas_cu12-12.4.5.8-py3-none-manylinux2014_x86_64.whl (363.4 MB)
    363.4/363.4 MB 4.7 MB/s eta 0:00:00:00:0100:01
Downloading nvidia_cuda_cupti_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl (13.8 MB)
    13.8/13.8 MB 93.1 MB/s eta 0:00:00:00:0100:01
Downloading nvidia_cuda_nvrtc_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl (24.6 MB)
    24.6/24.6 MB 75.1 MB/s eta 0:00:00:00:0100:01
Downloading nvidia_cuda_runtime_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl (883 kB)
    883.7/883.7 kB 44.2 MB/s eta 0:00:00
Downloading nvidia_cudnn_cu12-9.1.0.70-py3-none-manylinux2014_x86_64.whl (664.8 MB)
    664.8/664.8 MB 2.5 MB/s eta 0:00:00:00:0100:01
Downloading nvidia_cufft_cu12-11.2.1.3-py3-none-manylinux2014_x86_64.whl (211.5 MB)
    211.5/211.5 MB 8.0 MB/s eta 0:00:00:00:0100:01
Downloading nvidia_curand_cu12-10.3.5.147-py3-none-manylinux2014_x86_64.whl (56.3 MB)
    56.3/56.3 MB 30.8 MB/s eta 0:00:00:00:0100:01
Downloading nvidia_cusolver_cu12-11.6.1.9-py3-none-manylinux2014_x86_64.whl (127.9 MB)
    127.9/127.9 MB 8.7 MB/s eta 0:00:00:00:0100:01
Downloading nvidia_cusparse_cu12-12.3.1.170-py3-none-manylinux2014_x86_64.whl (207.5 MB)
    207.5/207.5 MB 8.3 MB/s eta 0:00:00:00:0100:01
Downloading nvidia_nvjitlink_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl (21.1 MB)
    21.1/21.1 MB 85.7 MB/s eta 0:00:00:00:0100:01
Downloading ttach-0.0.3-py3-none-any.whl (9.8 kB)
Building wheels for collected packages: grad-cam
  Building wheel for grad-cam (pyproject.toml) ... done
  Created wheel for grad-cam: filename=grad_cam-1.5.5-py3-none-any.whl size=44284 sha256=56552a8d487a81e29f97f46
9d180ec067c7d67d2392ba3e9216d59a6ec346b54
  Stored in directory: /root/.cache/pip/wheels/bc/52/78/893c3b94279ef238f43a9e89608af648de401b96415bebbd1f
Successfully built grad-cam
Installing collected packages: ttach, nvidia-nvjitlink-cu12, nvidia-curand-cu12, nvidia-cufft-cu12, nvidia-cuda-
runtime-cu12, nvidia-cuda-nvrtc-cu12, nvidia-cuda-cupti-cu12, nvidia-cublas-cu12, nvidia-cusparse-cu12, nvidia-c
udnn-cu12, nvidia-cusolver-cu12, grad-cam
  Attempting uninstall: nvidia-nvjitlink-cu12
    Found existing installation: nvidia-nvjitlink-cu12 12.5.82
    Uninstalling nvidia-nvjitlink-cu12-12.5.82:
      Successfully uninstalled nvidia-nvjitlink-cu12-12.5.82
  Attempting uninstall: nvidia-curand-cu12
    Found existing installation: nvidia-curand-cu12 10.3.6.82
    Uninstalling nvidia-curand-cu12-10.3.6.82:
      Successfully uninstalled nvidia-curand-cu12-10.3.6.82
  Attempting uninstall: nvidia-cufft-cu12
    Found existing installation: nvidia-cufft-cu12 11.2.3.61
    Uninstalling nvidia-cufft-cu12-11.2.3.61:
      Successfully uninstalled nvidia-cufft-cu12-11.2.3.61
  Attempting uninstall: nvidia-cuda-runtime-cu12
    Found existing installation: nvidia-cuda-runtime-cu12 12.5.82
    Uninstalling nvidia-cuda-runtime-cu12-12.5.82:
      Successfully uninstalled nvidia-cuda-runtime-cu12-12.5.82
  Attempting uninstall: nvidia-cuda-nvrtc-cu12
    Found existing installation: nvidia-cuda-nvrtc-cu12 12.5.82
    Uninstalling nvidia-cuda-nvrtc-cu12-12.5.82:
      Successfully uninstalled nvidia-cuda-nvrtc-cu12-12.5.82
  Attempting uninstall: nvidia-cuda-cupti-cu12
    Found existing installation: nvidia-cuda-cupti-cu12 12.5.82
    Uninstalling nvidia-cuda-cupti-cu12-12.5.82:
      Successfully uninstalled nvidia-cuda-cupti-cu12-12.5.82
  Attempting uninstall: nvidia-cublas-cu12
    Found existing installation: nvidia-cublas-cu12 12.5.3.2
    Uninstalling nvidia-cublas-cu12-12.5.3.2:
      Successfully uninstalled nvidia-cublas-cu12-12.5.3.2
  Attempting uninstall: nvidia-cusparse-cu12
```

```
    Found existing installation: nvidia-cusparse-cu12 12.5.1.3
    Uninstalling nvidia-cusparse-cu12-12.5.1.3:
      Successfully uninstalled nvidia-cusparse-cu12-12.5.1.3
  Attempting uninstall: nvidia-cudnn-cu12
    Found existing installation: nvidia-cudnn-cu12 9.3.0.75
    Uninstalling nvidia-cudnn-cu12-9.3.0.75:
      Successfully uninstalled nvidia-cudnn-cu12-9.3.0.75
  Attempting uninstall: nvidia-cusolver-cu12
    Found existing installation: nvidia-cusolver-cu12 11.6.3.83
    Uninstalling nvidia-cusolver-cu12-11.6.3.83:
      Successfully uninstalled nvidia-cusolver-cu12-11.6.3.83
Successfully installed grad-cam-1.5.5 nvidia-cublas-cu12-12.4.5.8 nvidia-cuda-cupti-cu12-12.4.127 nvidia-cuda-nvrtc-cu12-12.4.127 nvidia-cuda-runtime-cu12-12.4.127 nvidia-cudnn-cu12-9.1.0.70 nvidia-cufft-cu12-11.2.1.3 nvidia-curand-cu12-10.3.5.147 nvidia-cusolver-cu12-11.6.1.9 nvidia-cusparse-cu12-12.3.1.170 nvidia-nvjitlink-cu12-12.4.127 ttach-0.0.3
```

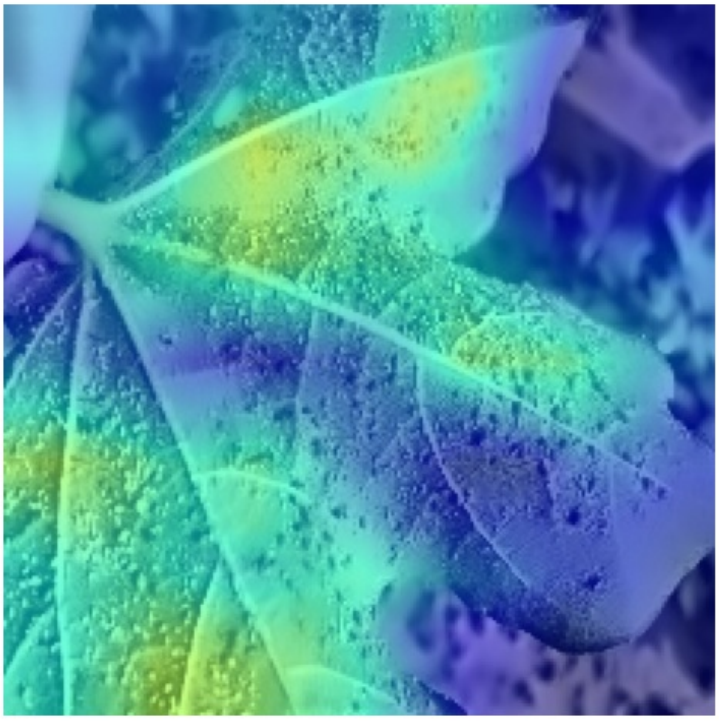
In [ ]:
```python
from pytorch_grad_cam import GradCAM
from pytorch_grad_cam.utils.model_targets import ClassifierOutputTarget
from pytorch_grad_cam.utils.image import show_cam_on_image
```

In [ ]:
```python
from pytorch_grad_cam import GradCAM
from pytorch_grad_cam.utils.model_targets import ClassifierOutputTarget
from pytorch_grad_cam.utils.image import show_cam_on_image

# ----------------------
# GRAD-CAM VISUALIZATION
# ----------------------
def plot_gradcam(models_list, loader, class_names, num_images=5):
    device = next(models_list[0].parameters()).device


    for model in models_list:
        model.eval()
        if isinstance(model, SqueezeNet_MHSA):
            for p in model.features[-1].parameters(): p.requires_grad = True
        elif isinstance(model, ShuffleNetV2_MHSA):
            for p in model.backbone.conv5.parameters(): p.requires_grad = True
        elif isinstance(model, EfficientNetB0_MHSA):
            for p in model.backbone.features[-1].parameters(): p.requires_grad = True

    for class_idx, class_name in enumerate(class_names):

        class_images = [(img, lbl) for img, lbl in loader.dataset if lbl==class_idx][:num_images]

        for i, (img_tensor, lbl) in enumerate(class_images):
            input_tensor = img_tensor.unsqueeze(0).to(device)
            input_tensor.requires_grad = True

            combined_map = 0
            for model in models_list:

                if isinstance(model, SqueezeNet_MHSA):
                    target_layer = model.features[-1]
                elif isinstance(model, ShuffleNetV2_MHSA):
                    target_layer = model.backbone.conv5
                elif isinstance(model, EfficientNetB0_MHSA):
                    target_layer = model.backbone.features[-1]

                cam = GradCAM(model=model, target_layers=[target_layer])
                targets = [ClassifierOutputTarget(lbl)]
                grayscale_cam = cam(input_tensor, targets=targets)[0]
                combined_map += grayscale_cam

            combined_map /= len(models_list)


            img_np = img_tensor.permute(1,2,0).cpu().numpy()
            img_np = (img_np - img_np.min()) / (img_np.max() - img_np.min() + 1e-8)

            # Overlay CAM
            visualization = show_cam_on_image(img_np, combined_map, use_rgb=True)

            plt.figure(figsize=(6,6))
            plt.imshow(visualization)
            plt.title(f"{class_name} Image {i+1} Grad-CAM Ensemble")
            plt.axis('off')
            plt.show()


plot_gradcam(models_list, test_loader, class_names, num_images=5)
```

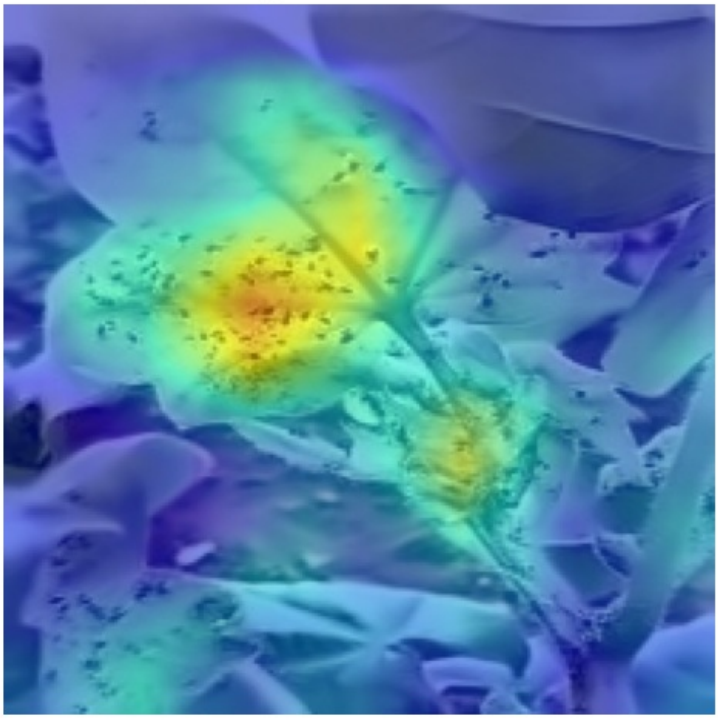
Aphids Image 1 Grad-CAM Ensemble



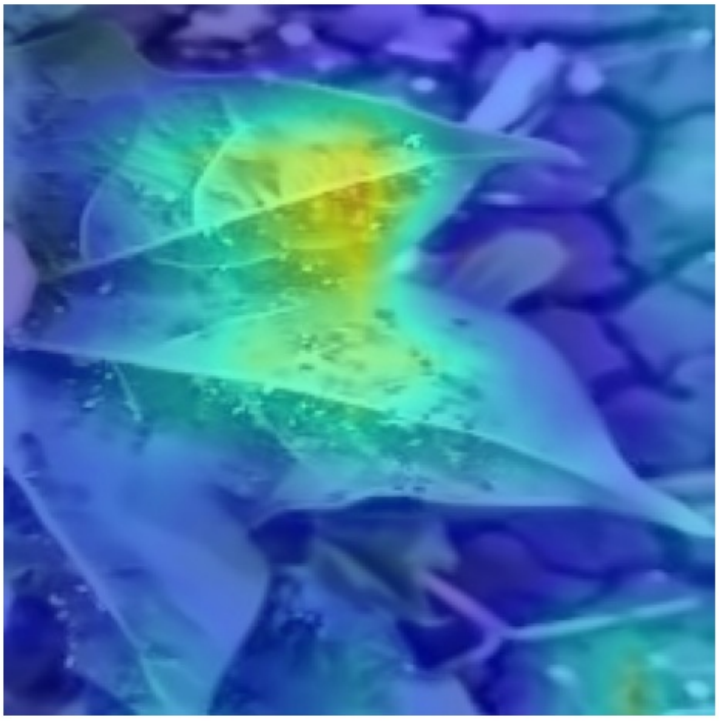
Aphids Image 2 Grad-CAM Ensemble



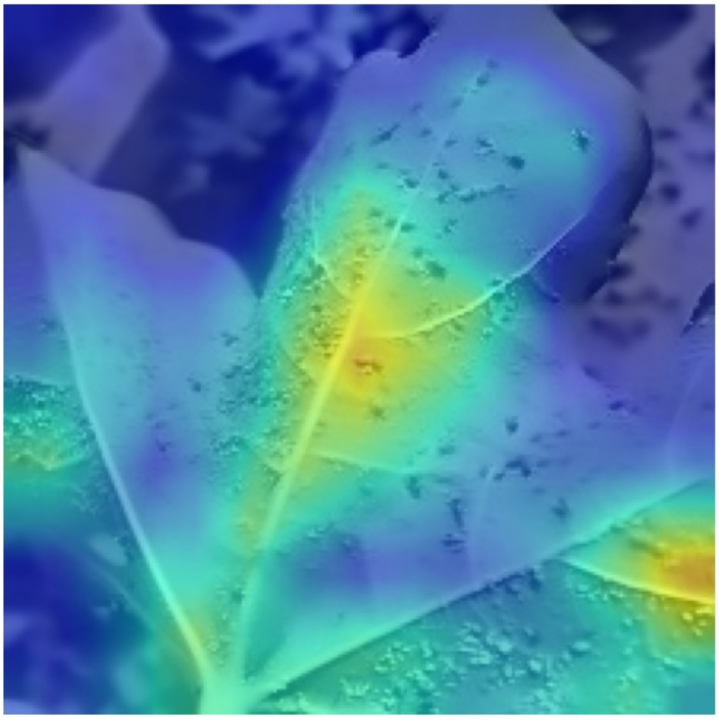
Aphids Image 3 Grad-CAM Ensemble



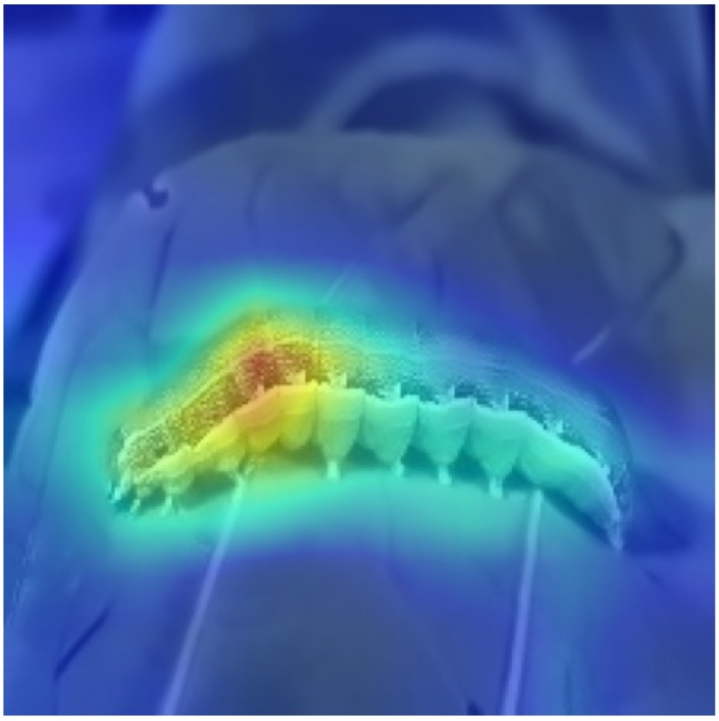
Aphids Image 4 Grad-CAM Ensemble



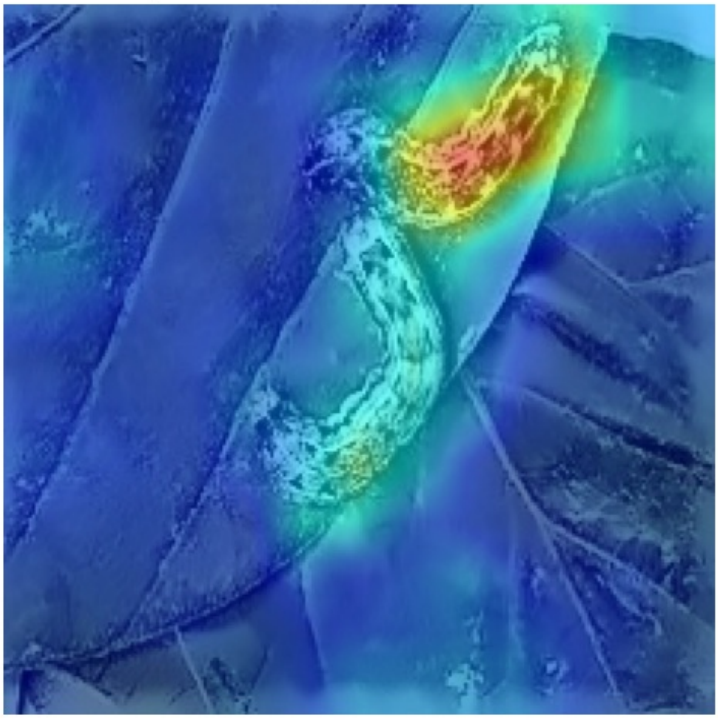
Aphids Image 5 Grad-CAM Ensemble



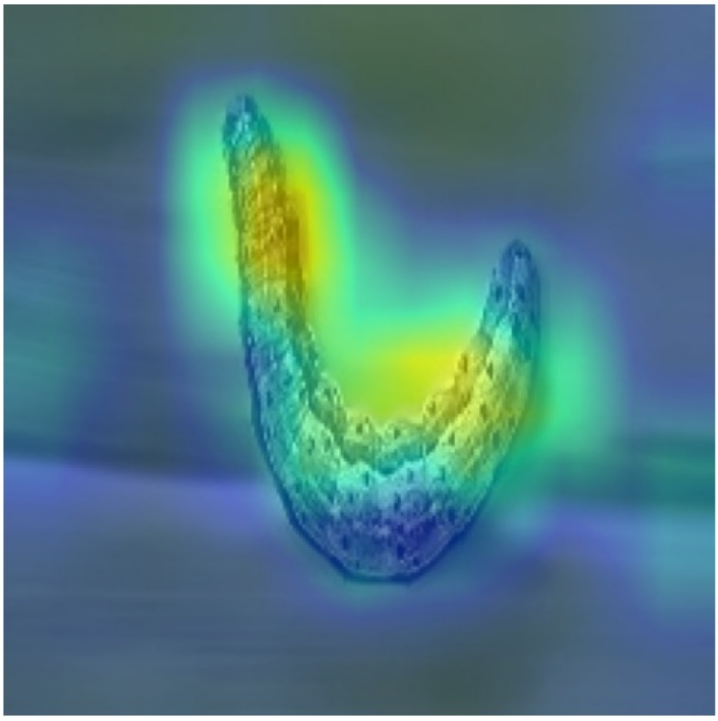
Army worm Image 1 Grad-CAM Ensemble



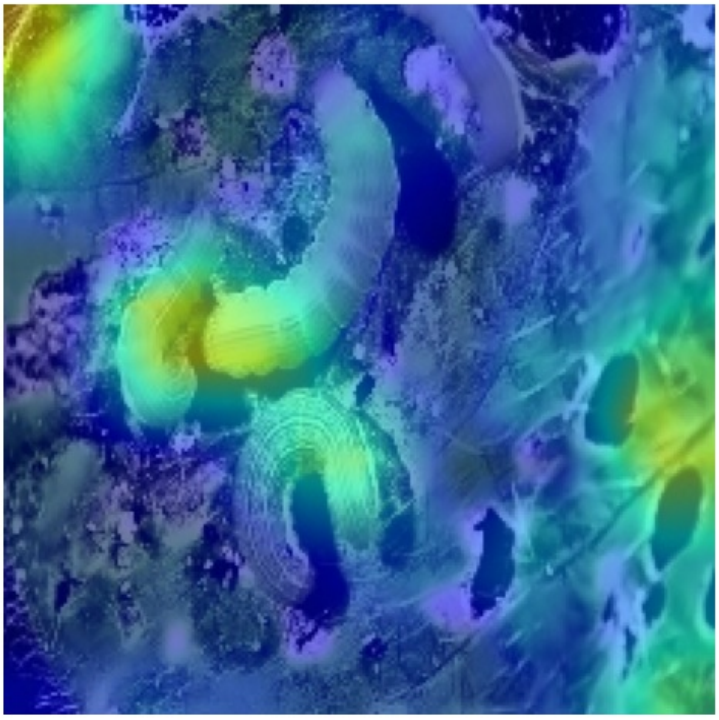
Army worm Image 2 Grad-CAM Ensemble



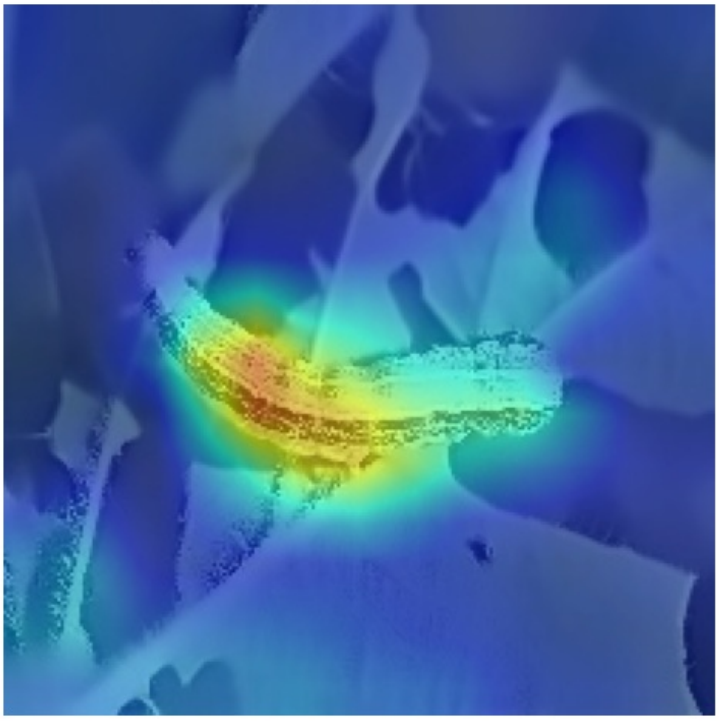
Army worm Image 3 Grad-CAM Ensemble



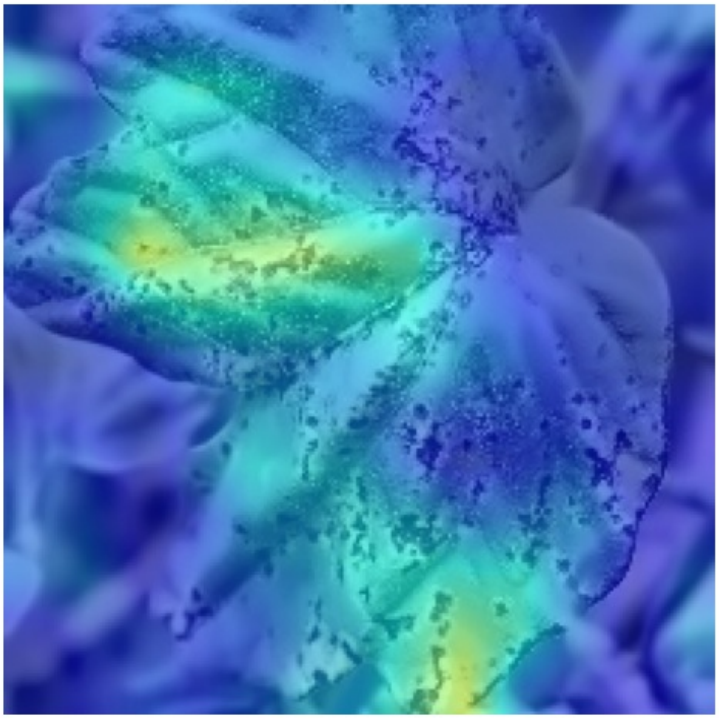
Army worm Image 4 Grad-CAM Ensemble



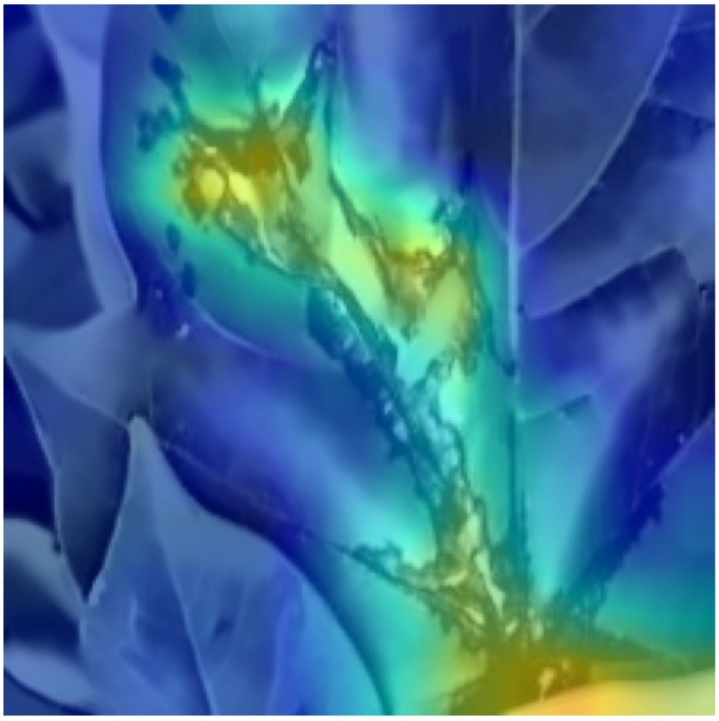
Army worm Image 5 Grad-CAM Ensemble



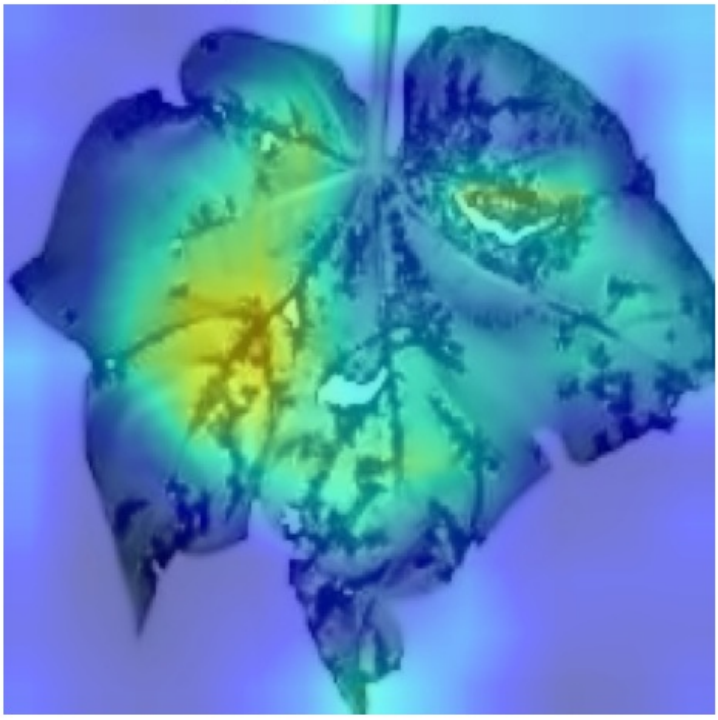
Bacterial Blight Image 1 Grad-CAM Ensemble



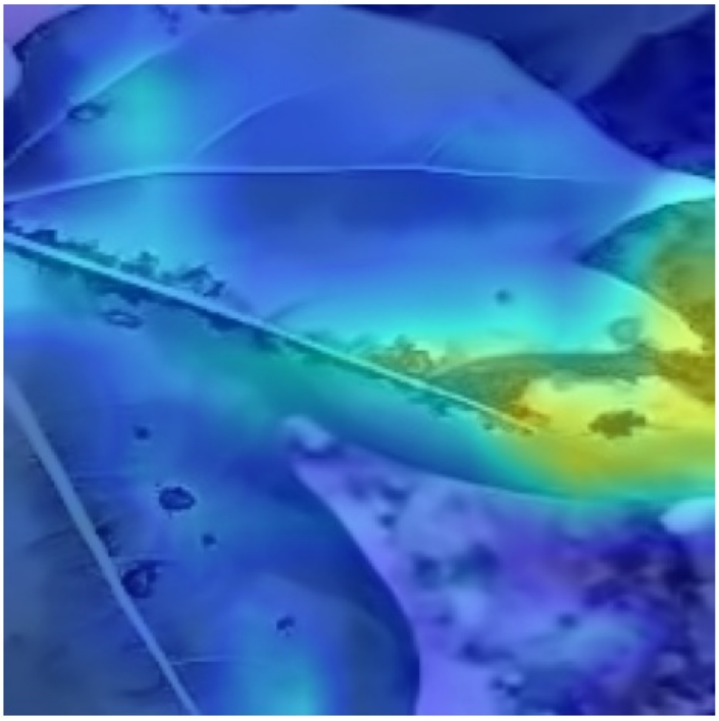
Bacterial Blight Image 2 Grad-CAM Ensemble



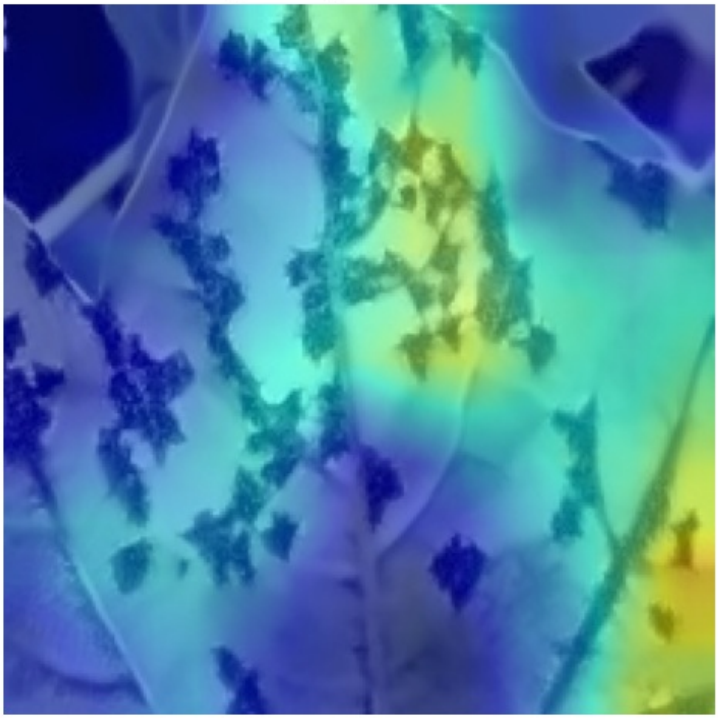
Bacterial Blight Image 3 Grad-CAM Ensemble



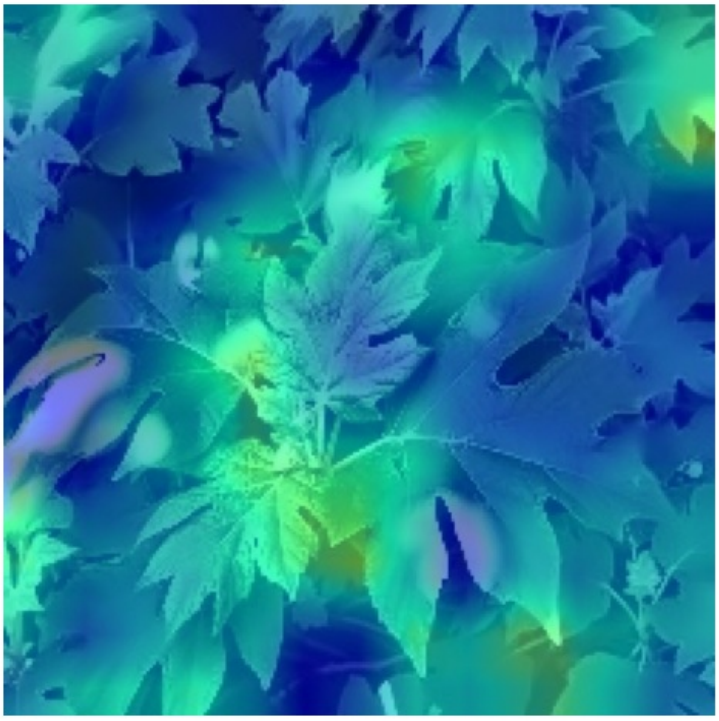
Bacterial Blight Image 4 Grad-CAM Ensemble



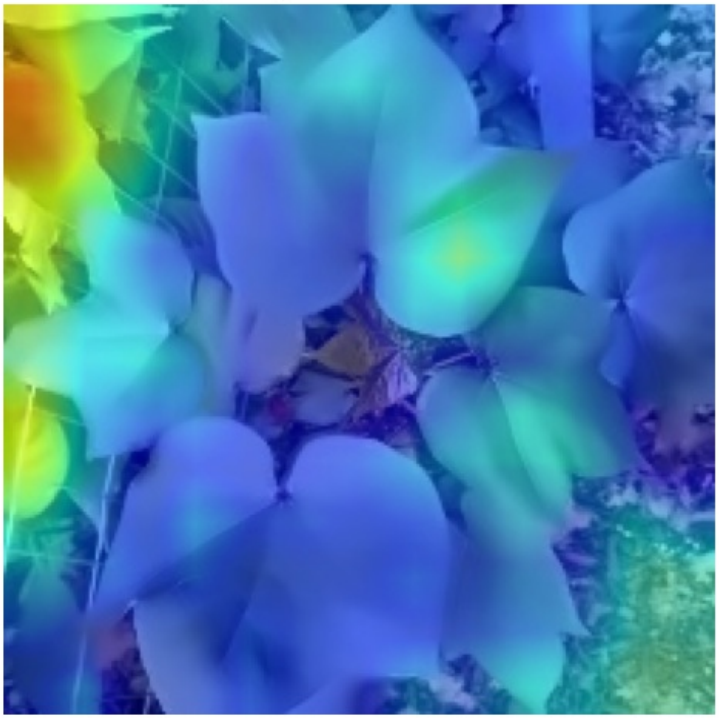
Bacterial Blight Image 5 Grad-CAM Ensemble



Healthy Image 1 Grad-CAM Ensemble



Healthy Image 2 Grad-CAM Ensemble

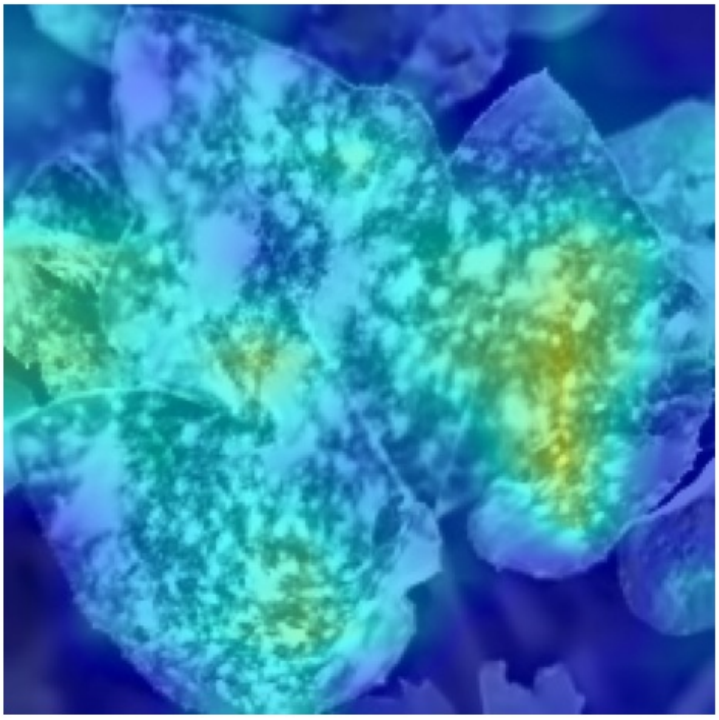


Healthy Image 3 Grad-CAM Ensemble

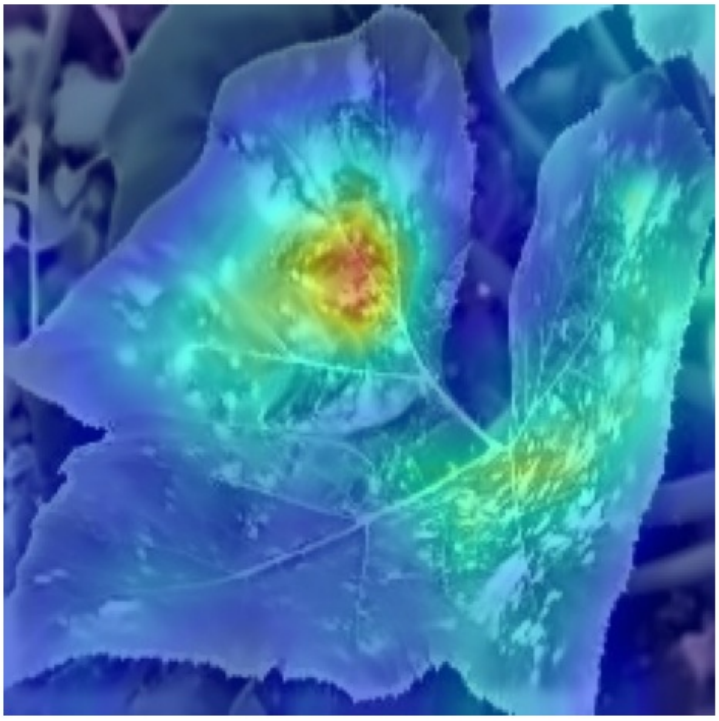
Healthy Image 4 Grad-CAM Ensemble

Healthy Image 5 Grad-CAM Ensemble

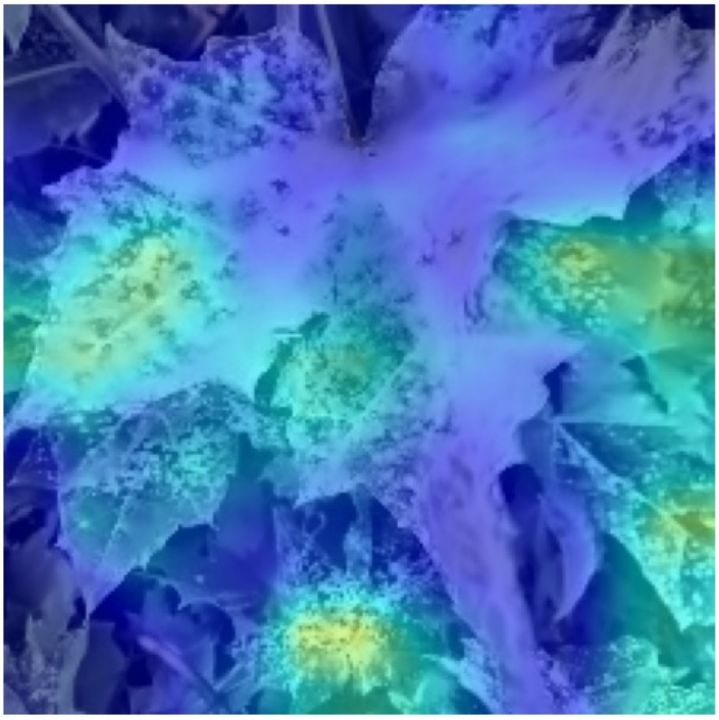
Powdery Mildew Image 1 Grad-CAM Ensemble



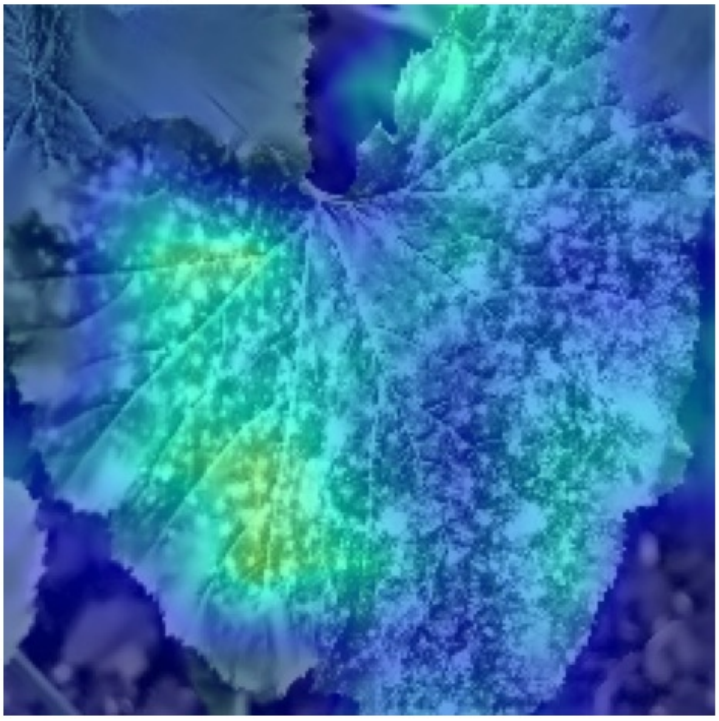
Powdery Mildew Image 2 Grad-CAM Ensemble



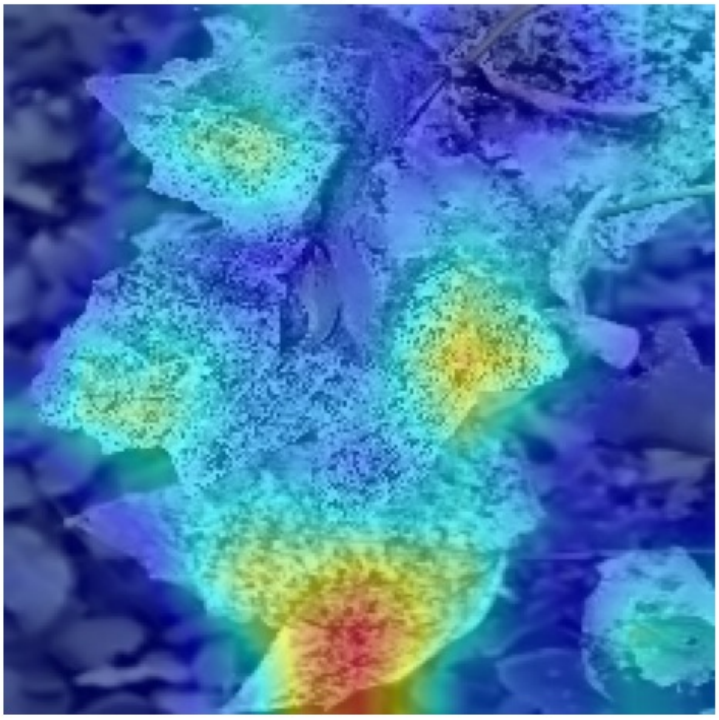
Powdery Mildew Image 3 Grad-CAM Ensemble



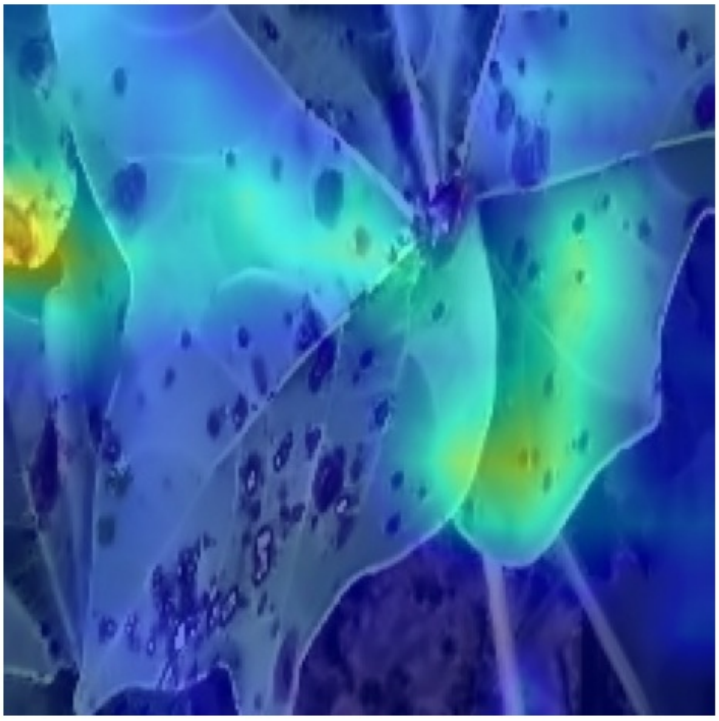
Powdery Mildew Image 4 Grad-CAM Ensemble



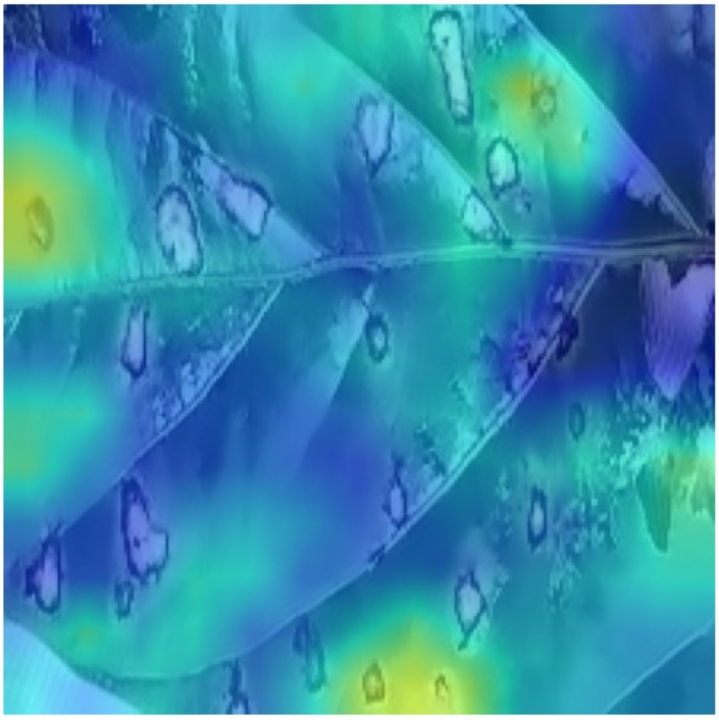
Powdery Mildew Image 5 Grad-CAM Ensemble



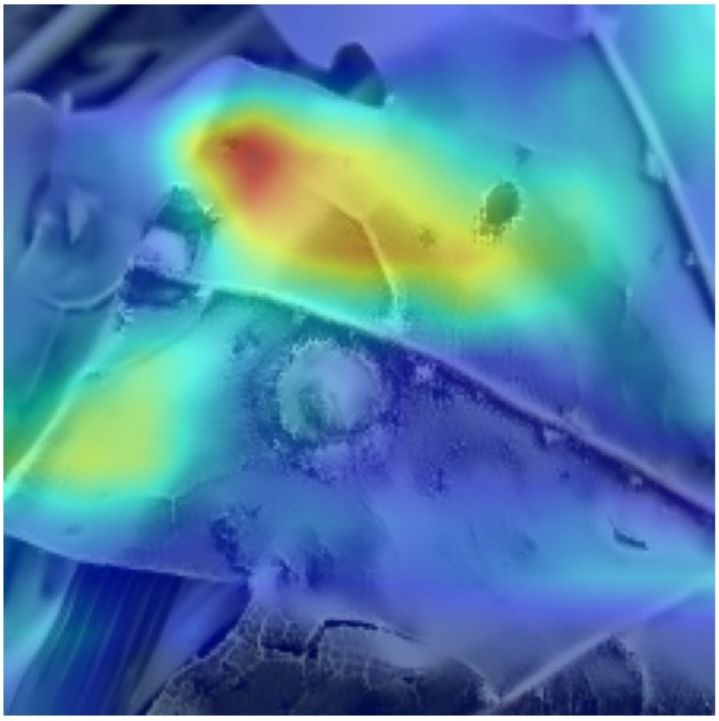
Target spot Image 1 Grad-CAM Ensemble



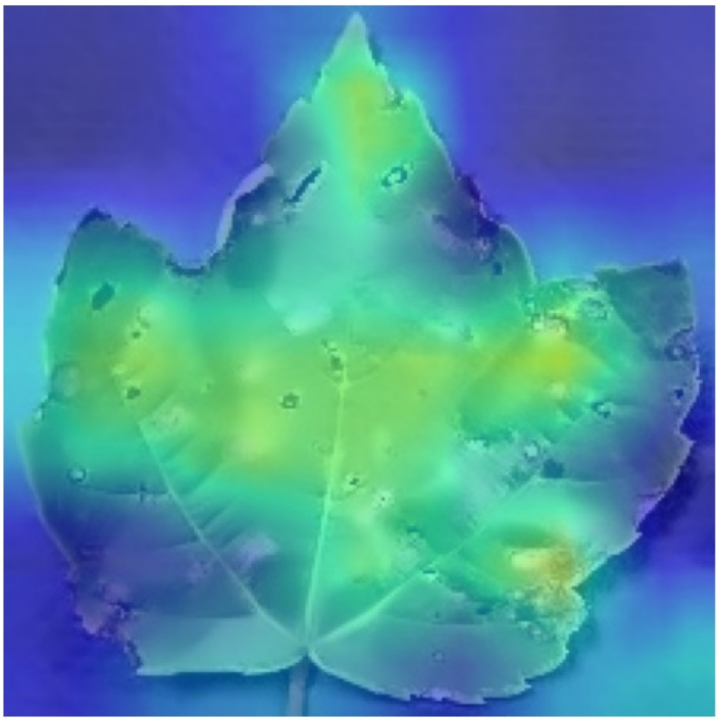
Target spot Image 2 Grad-CAM Ensemble



Target spot Image 3 Grad-CAM Ensemble



Target spot Image 4 Grad-CAM Ensemble



Target spot Image 5 Grad-CAM Ensemble