```
import zipfile
import os

zip_path = '/content/dataset.zip'
extract_path = '/content/dataset_FINIAL'

# Unzip everything inside /content/dataset
with zipfile.ZipFile(zip_path, 'r') as zip_ref:
    zip_ref.extractall(extract_path)

print("Unzipped successfully to:", extract_path)
```

```
Unzipped successfully to: /content/dataset_FINIAL
```

```
import os

root_dir = '/content/dataset_FINIAL'

# Recursively walk through the folder and print structure
for root, dirs, files in os.walk(root_dir):
    level = root.replace(root_dir, '').count(os.sep)
    indent = '  ' * level
    print(f"{indent} {os.path.basename(root)}/")
    sub_indent = '  ' * (level + 1)
    for f in files:
        print(f"{sub_indent} {f}")
```

```
 dataset_FINIAL/
   dataset/
     unhealthy/
       DJI_0441.JPG
       DJI_0508.JPG
       DJI_0529.JPG
       DJI_0485.JPG
       DJI_0482.JPG
       DJI_0476.JPG
       DJI_0437.JPG
       DJI_0501.JPG
       DJI_0510.JPG
       DJI_0390.JPG
       DJI_0431.JPG
       DJI_0486.JPG
       DJI_0544.JPG
       DJI_0447.JPG
       DJI_0453.JPG
       DJI_0478.JPG
       DJI_0523.JPG
       DJI_0519.JPG
       DJI_0527.JPG
       DJI_0539.JPG
       DJI_0515.JPG
       DJI_0393.JPG
       DJI_0530.JPG
       DJI_0443.JPG
       DJI_0531.JPG
       DJI_0537.JPG
       DJI_0389.JPG
       DJI_0422.JPG
       DJI_0399.JPG
       DJI_0534.JPG
       DJI_0432.JPG
       DJI_0507.JPG
       DJI_0540.JPG
       DJI_0543.JPG
       DJI_0472.JPG
       DJI_0395.JPG
       DJI_0487.JPG
       DJI_0514.JPG
       DJI_0442.JPG
       DJI_0446.JPG
       DJI_0502.JPG
       DJI_0521.JPG
       DJI_0444.JPG
       DJI_0474.JPG
       DJI_0496.JPG
       DJI_0397.JPG
       DJI_0542.JPG
```

```
            DJI_0452.JPG
            DJI_0470.JPG
            DJI_0433.JPG
            DJI_0480.JPG
            DJI_0520.JPG
            DJI_0445.JPG
            DJI_0499.JPG
```

```
import os
import pandas as pd
import numpy as np
import cv2
from sklearn.preprocessing import MinMaxScaler
import torch
from torch.utils.data import Dataset, DataLoader, random_split
from torchvision import transforms
from PIL import Image
import matplotlib.pyplot as plt

# === CONFIG ===
SPECTRAL_XLSX = '/content/finaldata_shuffled (1).xlsx'
IMAGE_DIR = '/content/dataset_FINIAL/dataset'
IMAGE_SIZE = (224, 224)

# === STEP 1: Load and Normalize Spectral Data ===
spectral_df = pd.read_excel(SPECTRAL_XLSX)
spectral_df['Image_number'] = spectral_df['Image_number'].astype(int)

spectral_features = spectral_df.columns[2:]
scaler = MinMaxScaler()
spectral_df[spectral_features] = scaler.fit_transform(spectral_df[spectral_features])

spectral_labels = np.where(spectral_df['Label'].str.lower().str.startswith('healthy'), 0, 1)
spectral_df_clean = pd.DataFrame({
    'image_number': spectral_df['Image_number'],
    'spectral_features': list(spectral_df[spectral_features].values),
    'label_encoded': spectral_labels
})

# === STEP 2: Build Image DataFrame ===
image_paths, labels = [], []
for label in ['healthy', 'unhealthy']:
    folder = os.path.join(IMAGE_DIR, label)
    for fname in os.listdir(folder):
        if fname.lower().endswith(('.jpg', '.jpeg', '.png')):
            image_paths.append(os.path.join(folder, fname))
            labels.append(label)

image_df = pd.DataFrame({
    'image_path': image_paths,
    'label': labels,
    'image_number': [int(os.path.splitext(os.path.basename(p))[0].split('_')[-1]) for p in image_paths]
})

# === STEP 3: Merge Cleanly ===
merged_df = pd.merge(image_df, spectral_df_clean, on='image_number', how='inner').dropna()
merged_df['label_encoded'] = merged_df['label_encoded'].astype(int)

# === STEP 4: Define Transforms ===
train_transform = transforms.Compose([
    transforms.ToPILImage(),
    transforms.Resize(IMAGE_SIZE),
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.RandomVerticalFlip(p=0.5),
    transforms.RandomRotation(degrees=15),
    transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406],
                         [0.229, 0.224, 0.225])
])

val_transform = transforms.Compose([
    transforms.ToPILImage(),
    transforms.Resize(IMAGE_SIZE),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406],
                         [0.229, 0.224, 0.225])
])

# === STEP 5: Define Custom Dataset ===
class SmartGardenDataset(Dataset):
    def __init__(self, dataframe, image_size=(224, 224), transform=None):
        self.data = dataframe.reset_index(drop=True)
        self.image_size = image_size
        self.transform = transform

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

    def __getitem__(self, idx):
        row = self.data.loc[idx]
        image = cv2.imread(row['image_path'])
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        image = cv2.resize(image, self.image_size)

        if self.transform:
            image = self.transform(image)
        else:
            image = image / 255.0
            image = torch.tensor(image, dtype=torch.float32).permute(2, 0, 1)

        spectrum = torch.tensor(row['spectral_features'], dtype=torch.float32)
        label = torch.tensor(row['label_encoded'], dtype=torch.long)
        return image, spectrum, label

# === STEP 6: Create Dataset & Apply 60/40 Split ===
full_dataset = SmartGardenDataset(merged_df, image_size=IMAGE_SIZE)
train_size = int(0.6 * len(full_dataset))
val_size = len(full_dataset) - train_size
train_dataset, val_dataset = random_split(full_dataset, [train_size, val_size])

# Apply transforms
train_dataset.dataset.transform = train_transform
val_dataset.dataset.transform = val_transform

# === STEP 7: Create DataLoaders ===
train_loader = DataLoader(train_dataset, batch_size=16, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=16)

# === ✅ Final Check ===
print("✅ Dataset Ready!")
print("Total samples:", len(full_dataset))
print("Train:", len(train_dataset))
print("Val:", len(val_dataset))
print("Label Distribution:\n", merged_df['label_encoded'].value_counts().rename({0: 'Healthy', 1: 'Unhealthy'}))

# === (Optional) Visualize One Augmented Sample ===
sample_img, _, _ = train_dataset[0]
plt.imshow(sample_img.permute(1, 2, 0))
plt.title("Augmented Training Sample")
plt.axis('off')
plt.show()
```

```
✅ Dataset Ready!
Total samples: 221
Train: 132
Val: 89
Label Distribution:
 label_encoded
Unhealthy    118
Healthy      103
Name: count, dtype: int64
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers). G
```

Augmented Training Sample

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

class MultimodalNet(nn.Module):
    def __init__(self, num_spectral_features=18, fusion_dim=256):
        super(MultimodalNet, self).__init__()

        # === Image Encoder (CNN - Pretrained ResNet18) ===
        self.cnn = models.resnet18(weights=models.ResNet18_Weights.DEFAULT)
        self.cnn.fc = nn.Identity()  # remove final FC layer → output: 512-dim features

        # === Spectral Encoder (Autoencoder: Encoder part only) ===
        self.spectral_encoder = nn.Sequential(
            nn.Linear(num_spectral_features, 64),
            nn.ReLU(),
            nn.BatchNorm1d(64),
            nn.Linear(64, 32),  # Bottleneck
            nn.ReLU(),
            nn.BatchNorm1d(32)
        )

        # === Fusion Classifier ===
        self.classifier = nn.Sequential(
            nn.Linear(512 + 32, fusion_dim),
            nn.ReLU(),
            nn.Dropout(0.3),
            nn.Linear(fusion_dim, 1),  # Binary classification
            nn.Sigmoid()
        )

    def forward(self, image, spectrum):
        img_feat = self.cnn(image)                          # [batch, 512]
        spec_feat = self.spectral_encoder(spectrum)         # [batch, 32]
        fused = torch.cat([img_feat, spec_feat], dim=1)     # [batch, 544]
        out = self.classifier(fused).squeeze(1)             # [batch]
        return out


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

model = MultimodalNet().to(device)
```

```
Downloading: "https://download.pytorch.org/models/resnet18-f37072fd.pth" to /root/.cache/torch/hub/checkpoints/resnet18-f37072fd.pth
100%|██████████| 44.7M/44.7M [00:00<00:00, 162MB/s]
```

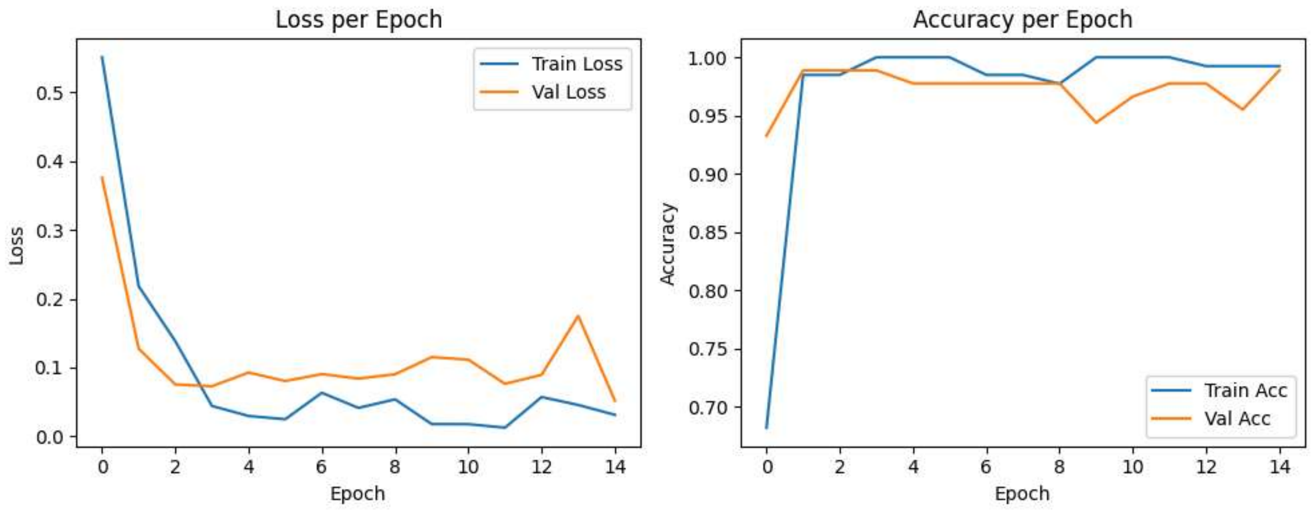
```
print(model)
```

```
          (0): BasicBlock(
            (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
            (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            (relu): ReLU(inplace=True)
            (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
            (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            (downsample): Sequential(
              (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), bias=False)
              (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            )
          )
          (1): BasicBlock(
            (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
            (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            (relu): ReLU(inplace=True)
            (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
            (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          )
        )
        (layer4): Sequential(
          (0): BasicBlock(
            (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
            (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            (relu): ReLU(inplace=True)
            (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
            (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            (downsample): Sequential(
              (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
              (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            )
          )
          (1): BasicBlock(
            (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
            (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            (relu): ReLU(inplace=True)
            (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
            (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          )
        )
        (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
        (fc): Identity()
      )
      (spectral_encoder): Sequential(
        (0): Linear(in_features=18, out_features=64, bias=True)
        (1): ReLU()
        (2): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (3): Linear(in_features=64, out_features=32, bias=True)
        (4): ReLU()
        (5): BatchNorm1d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
      (classifier): Sequential(
        (0): Linear(in_features=544, out_features=256, bias=True)
        (1): ReLU()
        (2): Dropout(p=0.3, inplace=False)
        (3): Linear(in_features=256, out_features=1, bias=True)
        (4): Sigmoid()
      )
    )
```

```
from torch.optim import Adam
import torch.nn.functional as F
from sklearn.metrics import precision_score, recall_score, f1_score, confusion_matrix, matthews_corrcoef, balanced_accuracy_score

# === CONFIG ===
num_epochs = 15
learning_rate = 1e-4

# === LOSS & OPTIMIZER ===
criterion = nn.BCELoss()
optimizer = Adam(model.parameters(), lr=learning_rate)

# === TRACKING HISTORY ===
train_losses, val_losses = [], []
train_accuracies, val_accuracies = [], []

# === TRAINING LOOP ===
for epoch in range(num_epochs):
    model.train()
    train_loss = 0
    correct = 0
    total = 0

    for images, spectra, labels in train_loader:
        images = images.to(device)
        spectra = spectra.to(device)
        labels = labels.float().to(device)

        # Forward
        outputs = model(images, spectra)
        loss = criterion(outputs, labels)

        # Backward
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        train_loss += loss.item() * labels.size(0)

        preds = (outputs > 0.5).long()
        correct += (preds == labels.long()).sum().item()
        total += labels.size(0)

    avg_train_loss = train_loss / total
    train_acc = correct / total

    train_losses.append(avg_train_loss)
    train_accuracies.append(train_acc)

    # === VALIDATION ===
    model.eval()
    val_loss = 0
    val_correct = 0
    val_total = 0
    all_labels = []
    all_preds = []

    with torch.no_grad():
        for images, spectra, labels in val_loader:
            images = images.to(device)
            spectra = spectra.to(device)
            labels = labels.float().to(device)

            outputs = model(images, spectra)
            loss = criterion(outputs, labels)

            val_loss += loss.item() * labels.size(0)
            preds = (outputs > 0.5).long()

            val_correct += (preds == labels.long()).sum().item()
            val_total += labels.size(0)

            all_labels.extend(labels.cpu().numpy())
            all_preds.extend(preds.cpu().numpy())

    avg_val_loss = val_loss / val_total
    val_acc = val_correct / val_total

    val_losses.append(avg_val_loss)
    val_accuracies.append(val_acc)

    # === METRICS ===
    precision = precision_score(all_labels, all_preds, zero_division=0)
    recall = recall_score(all_labels, all_preds, zero_division=0)
    f1 = f1_score(all_labels, all_preds, zero_division=0)
    mcc = matthews_corrcoef(all_labels, all_preds)
    balanced_acc = balanced_accuracy_score(all_labels, all_preds)
    cm = confusion_matrix(all_labels, all_preds)

    # === PRINT ===
    print(f"\n Epoch [{epoch+1}/{num_epochs}]")
    print(f"    Train Loss: {avg_train_loss:.4f} | Train Acc: {train_acc:.4f}")
    print(f"    Val   Loss: {avg_val_loss:.4f} | Val   Acc: {val_acc:.4f}")
    print(f"    Precision: {precision:.4f} | Recall: {recall:.4f} | F1 Score: {f1:.4f}")
    print(f"    Balanced Acc: {balanced_acc:.4f} | MCC: {mcc:.4f}")
    print(f"    Confusion Matrix:\n{cm}")
    print("-" * 60)
```

```
    Train Loss: 0.0178 | Train Acc: 1.0000
    Val   Loss: 0.1151 | Val   Acc: 0.9438
    Precision: 1.0000 | Recall: 0.9000 | F1 Score: 0.9474
    Balanced Acc: 0.9500 | MCC: 0.8932
    Confusion Matrix:
[[39  0]
 [ 5 45]]
------------------------------------------------------------

 Epoch [11/15]
    Train Loss: 0.0177 | Train Acc: 1.0000
    Val   Loss: 0.1114 | Val   Acc: 0.9663
    Precision: 1.0000 | Recall: 0.9400 | F1 Score: 0.9691
    Balanced Acc: 0.9700 | MCC: 0.9343
    Confusion Matrix:
[[39  0]
 [ 3 47]]
------------------------------------------------------------

 Epoch [12/15]
    Train Loss: 0.0126 | Train Acc: 1.0000
    Val   Loss: 0.0763 | Val   Acc: 0.9775
    Precision: 1.0000 | Recall: 0.9600 | F1 Score: 0.9796
    Balanced Acc: 0.9800 | MCC: 0.9556
    Confusion Matrix:
[[39  0]
 [ 2 48]]
------------------------------------------------------------

 Epoch [13/15]
    Train Loss: 0.0571 | Train Acc: 0.9924
    Val   Loss: 0.0894 | Val   Acc: 0.9775
    Precision: 1.0000 | Recall: 0.9600 | F1 Score: 0.9796
    Balanced Acc: 0.9800 | MCC: 0.9556
    Confusion Matrix:
[[39  0]
 [ 2 48]]
------------------------------------------------------------

 Epoch [14/15]
    Train Loss: 0.0456 | Train Acc: 0.9924
    Val   Loss: 0.1750 | Val   Acc: 0.9551
    Precision: 1.0000 | Recall: 0.9200 | F1 Score: 0.9583
    Balanced Acc: 0.9600 | MCC: 0.9135
    Confusion Matrix:
[[39  0]
 [ 4 46]]
------------------------------------------------------------

 Epoch [15/15]
    Train Loss: 0.0313 | Train Acc: 0.9924
    Val   Loss: 0.0519 | Val   Acc: 0.9888
    Precision: 1.0000 | Recall: 0.9800 | F1 Score: 0.9899
    Balanced Acc: 0.9900 | MCC: 0.9775
    Confusion Matrix:
[[39  0]
 [ 1 49]]
------------------------------------------------------------
```

```
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 4))

# === LOSS ===
plt.subplot(1, 2, 1)
plt.plot(train_losses, label='Train Loss')
plt.plot(val_losses, label='Val Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Loss per Epoch')
plt.legend()

# === ACCURACY ===
plt.subplot(1, 2, 2)
plt.plot(train_accuracies, label='Train Acc')
plt.plot(val_accuracies, label='Val Acc')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('Accuracy per Epoch')
plt.legend()

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



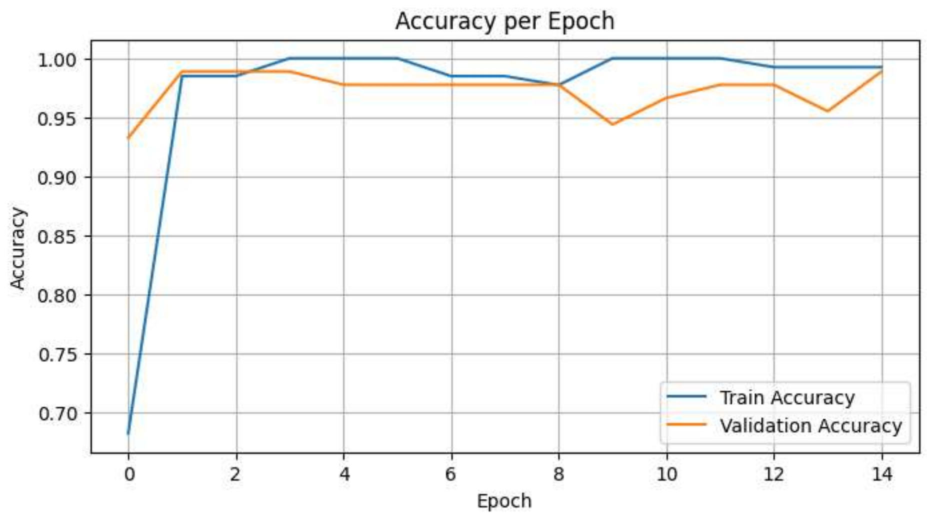
```
from sklearn.metrics import precision_score, recall_score, f1_score, confusion_matrix

# === VALIDATION ===
model.eval()
val_loss = 0
val_correct = 0
val_total = 0
all_labels = []
all_preds = []

with torch.no_grad():
    for images, spectra, labels in val_loader:
        images = images.to(device)
        spectra = spectra.to(device)
        labels = labels.float().to(device)

        outputs = model(images, spectra)
        loss = criterion(outputs, labels)

        val_loss += loss.item() * labels.size(0)
        preds = (outputs > 0.5).long()

        val_correct += (preds == labels.long()).sum().item()
        val_total += labels.size(0)

        all_labels.extend(labels.cpu().numpy())
        all_preds.extend(preds.cpu().numpy())

# === Calculate Metrics ===
val_acc = val_correct / val_total
avg_val_loss = val_loss / val_total

precision = precision_score(all_labels, all_preds, zero_division=0)
recall = recall_score(all_labels, all_preds, zero_division=0)
f1 = f1_score(all_labels, all_preds, zero_division=0)
cm = confusion_matrix(all_labels, all_preds)

# === PRINT EVERYTHING CLEANLY ===
print(f"\n Epoch [{epoch+1}/{num_epochs}]")
print(f"    Train Loss: {avg_train_loss:.4f} | Train Acc: {train_acc:.4f}")
print(f"    Val   Loss: {avg_val_loss:.4f} | Val   Acc: {val_acc:.4f}")
print(f"    Precision: {precision:.4f} | Recall: {recall:.4f} | F1 Score: {f1:.4f}")
print("    Confusion Matrix:")
print(cm)
print("-" * 60)
```

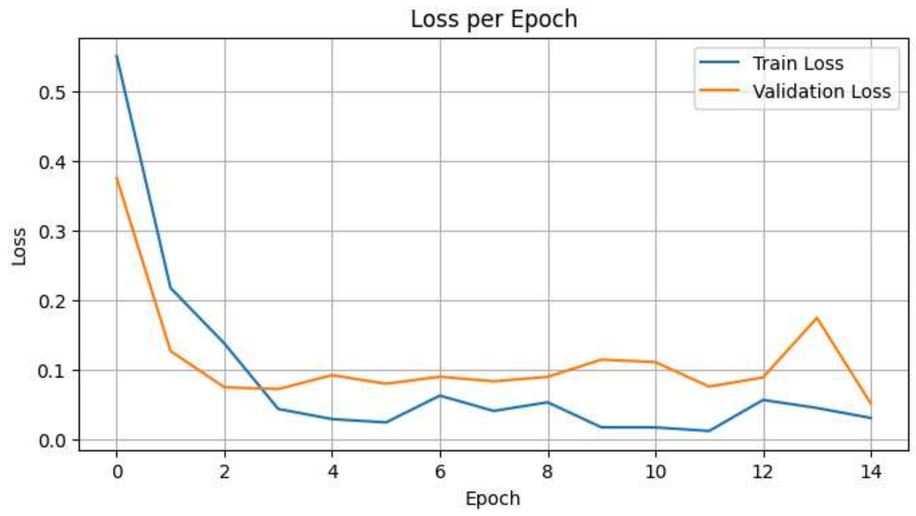
```
 Epoch [15/15]
    Train Loss: 0.0313 | Train Acc: 0.9924
    Val   Loss: 0.0519 | Val   Acc: 0.9888
    Precision: 1.0000 | Recall: 0.9800 | F1 Score: 0.9899
    Confusion Matrix:
[[39  0]
 [ 1 49]]
------------------------------------------------------------
```

```
torch.save(model.state_dict(), 'smart_gardening_model.pt')
print("✅ Model saved as smart_gardening_model.pt")
```

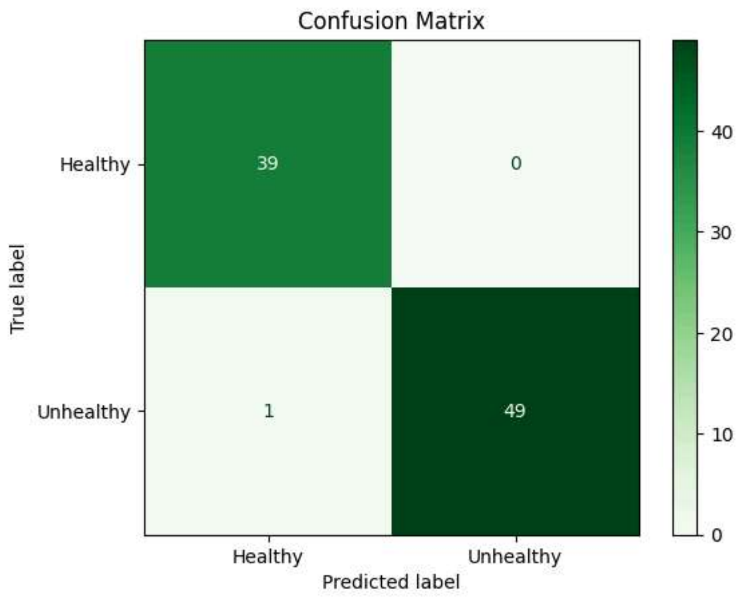
```
✅ Model saved as smart_gardening_model.pt
```

```
import matplotlib.pyplot as plt

# === Accuracy Plot ===
plt.figure(figsize=(8, 4))
plt.plot(train_accuracies, label='Train Accuracy')
plt.plot(val_accuracies, label='Validation Accuracy')
plt.title("Accuracy per Epoch")
plt.xlabel("Epoch")
plt.ylabel("Accuracy")
plt.legend()
plt.grid(True)
plt.show()

# === Loss Plot ===
plt.figure(figsize=(8, 4))
plt.plot(train_losses, label='Train Loss')
plt.plot(val_losses, label='Validation Loss')
plt.title("Loss per Epoch")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend()
plt.grid(True)
plt.show()
```

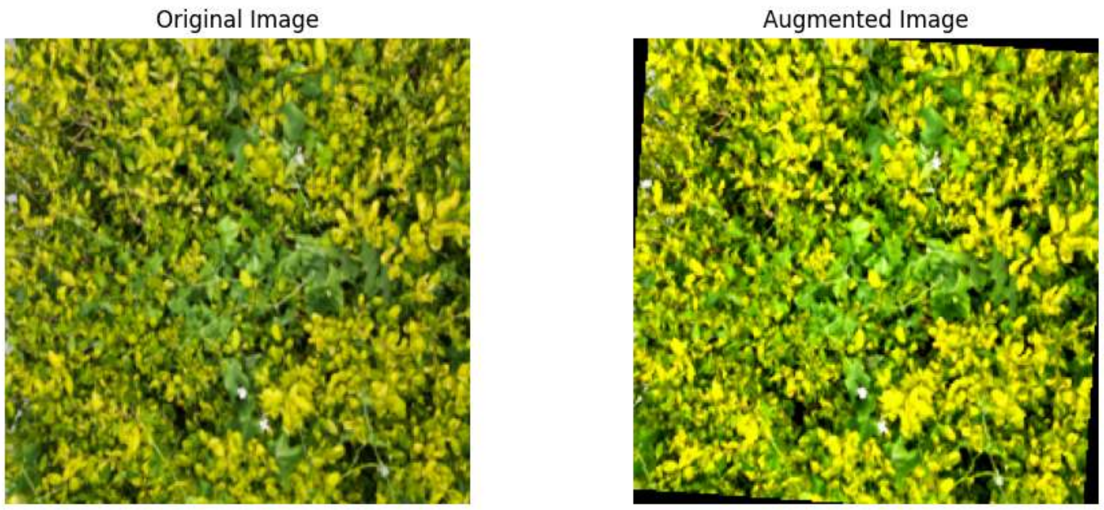




```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

cm = confusion_matrix(all_labels, all_preds)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=["Healthy", "Unhealthy"])
disp.plot(cmap='Greens')
plt.title("Confusion Matrix")
plt.grid(False)
plt.show()
```



```
# Sample index from training set
img_path = merged_df.iloc[0]['image_path']
img_orig = cv2.imread(img_path)
img_orig = cv2.cvtColor(img_orig, cv2.COLOR_BGR2RGB)
img_resized = cv2.resize(img_orig, (224, 224))

# Augmented version
augmented_img_tensor = train_transform(img_resized)
augmented_img = augmented_img_tensor.permute(1, 2, 0).numpy()
augmented_img = (augmented_img * [0.229, 0.224, 0.225]) + [0.485, 0.456, 0.406]  # de-normalize
augmented_img = np.clip(augmented_img, 0, 1)

# Show side by side
plt.figure(figsize=(10, 4))
plt.subplot(1, 2, 1)
plt.imshow(img_resized)
plt.title("Original Image")
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(augmented_img)
plt.title("Augmented Image")
plt.axis('off')
plt.tight_layout()
plt.show()
```



```
import matplotlib.pyplot as plt
import numpy as np
```