In [1]:
```
import kagglehub

# Download latest version
path = kagglehub.dataset_download("sohansakib75/cotton-4-class")

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

```
Path to dataset files: /kaggle/input/cotton-4-class
```

In [2]:
```
import os
import kagglehub

# Download dataset
path = kagglehub.dataset_download("sohansakib75/cotton-4-class")

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-4-class
Contents inside dataset folder:
 Cotton leaf/
```

In [3]:
```
import os
import glob

dataset_path = os.path.join(path, "Cotton leaf")

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-4-class/Cotton leaf
diseased cotton leaf: 346 images
diseased cotton plant: 921 images
fresh cotton leaf: 519 images
fresh cotton plant: 514 images
```

In [4]:
```
import os
import glob
import matplotlib.pyplot as plt
import random

dataset_path = "/kaggle/input/cotton-4-class/Cotton leaf"

# 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 [5]:

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

# Paths
input_dir = "/kaggle/input/cotton-4-class/Cotton leaf"
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])
```

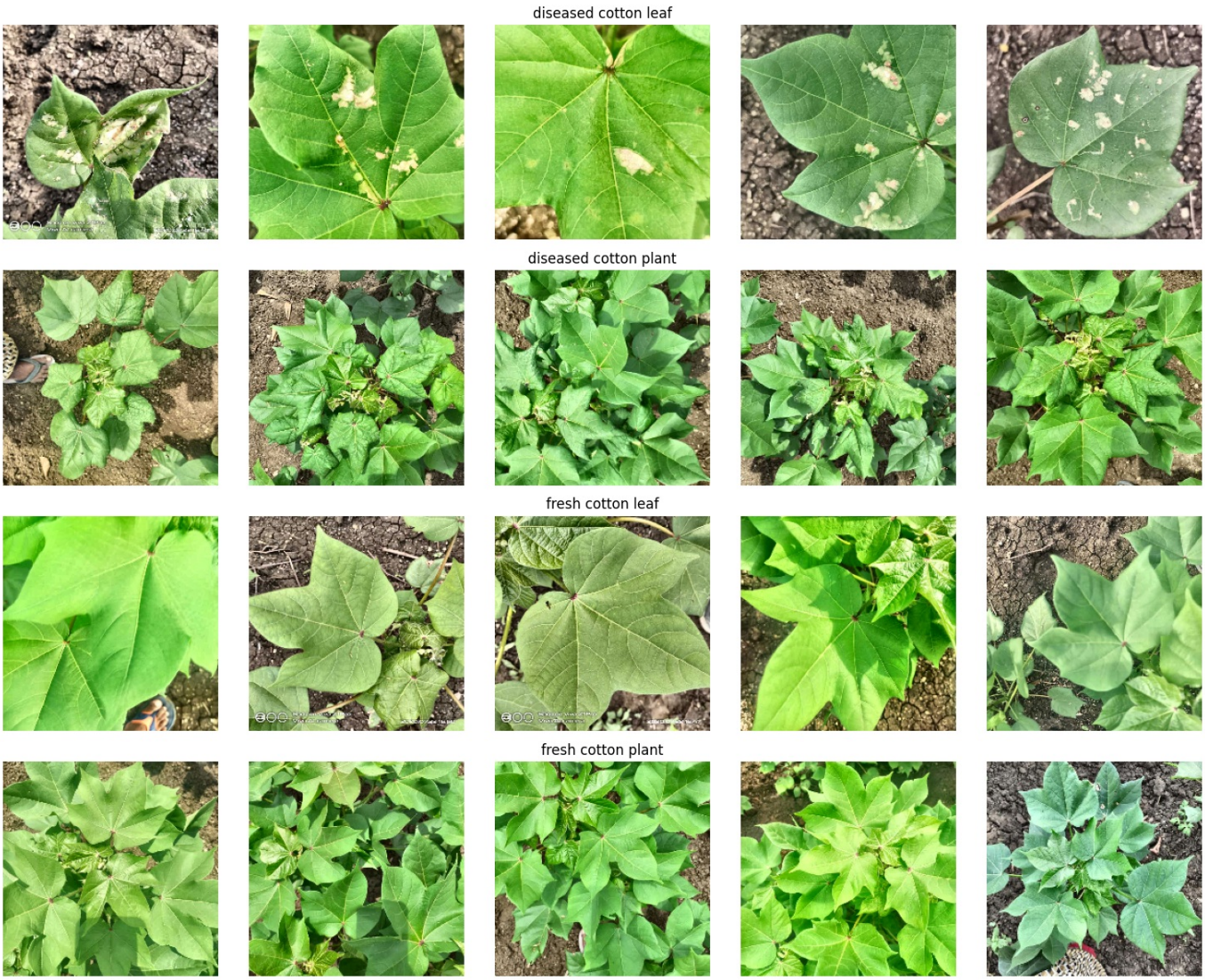
```
            img_clahe = cv2.cvtColor(yuv, cv2.COLOR_YUV2BGR)

            # 3️ Normalize to [0,1]
            img_final = img_clahe.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 [6]:

```
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 = 0.75
```

```
val_ratio = 0.1
test_ratio = 0.15

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 75-10-15 and saved in:", split_base)
```

```
Dataset split into 75-10-15 and saved in: /kaggle/working/cotton_split
```

In [7]:

```
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:
diseased cotton leaf: 259 images
diseased cotton plant: 690 images
fresh cotton leaf: 389 images
fresh cotton plant: 385 images

 Val Split:
diseased cotton leaf: 34 images
diseased cotton plant: 92 images
fresh cotton leaf: 51 images
fresh cotton plant: 51 images

 Test Split:
diseased cotton leaf: 53 images
diseased cotton plant: 139 images
fresh cotton leaf: 79 images
fresh cotton plant: 78 images
```

In [8]:

```
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:
diseased cotton leaf: 1036 images
diseased cotton plant: 2760 images
fresh cotton leaf: 1556 images
fresh cotton plant: 1540 images

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

In [9]:
```
!pip install timm
```

```
Requirement already satisfied: timm in /usr/local/lib/python3.11/dist-packages (1.0.19)
Requirement already satisfied: torch in /usr/local/lib/python3.11/dist-packages (from timm) (2.6.0+cu124)
Requirement already satisfied: torchvision in /usr/local/lib/python3.11/dist-packages (from timm) (0.21.0+cu124)
Requirement already satisfied: pyyaml in /usr/local/lib/python3.11/dist-packages (from timm) (6.0.3)
Requirement already satisfied: huggingface_hub in /usr/local/lib/python3.11/dist-packages (from timm) (1.0.0rc2)
Requirement already satisfied: safetensors in /usr/local/lib/python3.11/dist-packages (from timm) (0.5.3)
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-packages (from huggingface_hub->timm)
(3.19.1)
Requirement already satisfied: fsspec>=2023.5.0 in /usr/local/lib/python3.11/dist-packages (from huggingface_hub
->timm) (2025.9.0)
Requirement already satisfied: packaging>=20.9 in /usr/local/lib/python3.11/dist-packages (from huggingface_hub-
>timm) (25.0)
Requirement already satisfied: httpx<1,>=0.23.0 in /usr/local/lib/python3.11/dist-packages (from huggingface_hub
->timm) (0.28.1)
Requirement already satisfied: tqdm>=4.42.1 in /usr/local/lib/python3.11/dist-packages (from huggingface_hub->ti
mm) (4.67.1)
Requirement already satisfied: typer-slim in /usr/local/lib/python3.11/dist-packages (from huggingface_hub->timm
```

```
) (0.19.2)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.11/dist-packages (from huggi
ngface_hub->timm) (4.15.0)
Requirement already satisfied: hf-xet<2.0.0,>=1.1.3 in /usr/local/lib/python3.11/dist-packages (from huggingface
_hub->timm) (1.1.10)
Requirement already satisfied: networkx in /usr/local/lib/python3.11/dist-packages (from torch->timm) (3.5)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.11/dist-packages (from torch->timm) (3.1.6)
Collecting nvidia-cuda-nvrtc-cu12==12.4.127 (from torch->timm)
  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->timm)
  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->timm)
  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->timm)
  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->timm)
  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->timm)
  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->timm)
  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->timm)
  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->timm)
  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->timm) (0.6.2)
Requirement already satisfied: nvidia-nccl-cu12==2.21.5 in /usr/local/lib/python3.11/dist-packages (from torch->
timm) (2.21.5)
Requirement already satisfied: nvidia-nvtx-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (from torch
->timm) (12.4.127)
Collecting nvidia-nvjitlink-cu12==12.4.127 (from torch->timm)
  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->timm) (3.2.
0)
Requirement already satisfied: sympy==1.13.1 in /usr/local/lib/python3.11/dist-packages (from torch->timm) (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->timm) (1.3.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (from torchvision->timm) (1.26.4
)
Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in /usr/local/lib/python3.11/dist-packages (from torchvisio
n->timm) (11.3.0)
Requirement already satisfied: anyio in /usr/local/lib/python3.11/dist-packages (from httpx<1,>=0.23.0->huggingf
ace_hub->timm) (4.11.0)
Requirement already satisfied: certifi in /usr/local/lib/python3.11/dist-packages (from httpx<1,>=0.23.0->huggin
gface_hub->timm) (2025.8.3)
Requirement already satisfied: httpcore==1.* in /usr/local/lib/python3.11/dist-packages (from httpx<1,>=0.23.0->
huggingface_hub->timm) (1.0.9)
Requirement already satisfied: idna in /usr/local/lib/python3.11/dist-packages (from httpx<1,>=0.23.0->huggingfa
ce_hub->timm) (3.10)
Requirement already satisfied: h11>=0.16 in /usr/local/lib/python3.11/dist-packages (from httpcore==1.*->httpx<1
,>=0.23.0->huggingface_hub->timm) (0.16.0)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.11/dist-packages (from jinja2->torch->t
imm) (3.0.2)
Requirement already satisfied: mkl_fft in /usr/local/lib/python3.11/dist-packages (from numpy->torchvision->timm
) (1.3.8)
Requirement already satisfied: mkl_random in /usr/local/lib/python3.11/dist-packages (from numpy->torchvision->t
imm) (1.2.4)
Requirement already satisfied: mkl_umath in /usr/local/lib/python3.11/dist-packages (from numpy->torchvision->ti
mm) (0.1.1)
Requirement already satisfied: mkl in /usr/local/lib/python3.11/dist-packages (from numpy->torchvision->timm) (2
025.2.0)
Requirement already satisfied: tbb4py in /usr/local/lib/python3.11/dist-packages (from numpy->torchvision->timm)
(2022.2.0)
Requirement already satisfied: mkl-service in /usr/local/lib/python3.11/dist-packages (from numpy->torchvision->
timm) (2.4.1)
Requirement already satisfied: click>=8.0.0 in /usr/local/lib/python3.11/dist-packages (from typer-slim->hugging
face_hub->timm) (8.3.0)
Requirement already satisfied: sniffio>=1.1 in /usr/local/lib/python3.11/dist-packages (from anyio->httpx<1,>=0.
23.0->huggingface_hub->timm) (1.3.1)
Requirement already satisfied: intel-openmp<2026,>=2024 in /usr/local/lib/python3.11/dist-packages (from mkl->nu
mpy->torchvision->timm) (2024.2.0)
Requirement already satisfied: tbb==2022.* in /usr/local/lib/python3.11/dist-packages (from mkl->numpy->torchvis
ion->timm) (2022.2.0)
Requirement already satisfied: tcmlib==1.* in /usr/local/lib/python3.11/dist-packages (from tbb==2022.*->mkl->nu
mpy->torchvision->timm) (1.4.0)
Requirement already satisfied: intel-cmplr-lib-rt in /usr/local/lib/python3.11/dist-packages (from mkl_umath->nu
mpy->torchvision->timm) (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->torchvision->timm) (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 5.0 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 109.3 MB/s eta 0:00:0000:010:01
Downloading nvidia_cuda_nvrtc_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl (24.6 MB)
    24.6/24.6 MB 61.4 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 46.9 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 2.1 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 3.4 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 24.6 MB/s eta 0:00:0000:0100:01
Downloading nvidia_cusolver_cu12-11.6.1.9-py3-none-manylinux2014_x86_64.whl (127.9 MB)
    127.9/127.9 MB 12.4 MB/s eta 0:00:0000:0100:01
Downloading nvidia_cusparse_cu12-12.3.1.170-py3-none-manylinux2014_x86_64.whl (207.5 MB)
    207.5/207.5 MB 8.0 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 63.7 MB/s eta 0:00:00:00:0100:01
Installing collected packages: 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-cudnn-cu
12, nvidia-cusolver-cu12
  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
ERROR: pip's dependency resolver does not currently take into account all the packages that are installed. This
behaviour is the source of the following dependency conflicts.
libcugraph-cu12 25.6.0 requires libraft-cu12==25.6.*, but you have libraft-cu12 25.2.0 which is incompatible.
pylibcugraph-cu12 25.6.0 requires pylibraft-cu12==25.6.*, but you have pylibraft-cu12 25.2.0 which is incompatib
le.
pylibcugraph-cu12 25.6.0 requires rmm-cu12==25.6.*, but you have rmm-cu12 25.2.0 which is incompatible.
Successfully installed nvidia-cublas-cu12-12.4.5.8 nvidia-cuda-cupti-cu12-12.4.127 nvidia-cuda-nvrtc-cu12-12.4.1
27 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
```

In [10]:
```
import timm
```

```
/usr/local/lib/python3.11/dist-packages/pydantic/_internal/_generate_schema.py:2225: UnsupportedFieldAttributeWa
rning: The 'repr' attribute with value False was provided to the `Field()` function, which has no effect in the
context it was used. 'repr' is field-specific metadata, and can only be attached to a model field using `Annotat
ed` metadata or by assignment. This may have happened because an `Annotated` type alias using the `type` stateme
nt was used, or if the `Field()` function was attached to a single member of a union type.
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/pydantic/_internal/_generate_schema.py:2225: UnsupportedFieldAttributeWa
rning: The 'frozen' attribute with value True was provided to the `Field()` function, which has no effect in the
context it was used. 'frozen' is field-specific metadata, and can only be attached to a model field using `Annot
ated` metadata or by assignment. This may have happened because an `Annotated` type alias using the `type` state
ment was used, or if the `Field()` function was attached to a single member of a union type.
  warnings.warn(
```

In [11]:

```python
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, Subset
from torchvision import datasets, transforms
from timm import create_model
from torch.optim.lr_scheduler import ReduceLROnPlateau
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score
import numpy as np
import time
import gc

# ==================================
# CONFIGURATION
# ==================================
train_dir = "/kaggle/working/cotton_train_aug"
val_dir   = "/kaggle/working/cotton_split/val"
test_dir  = "/kaggle/working/cotton_split/test"

num_classes = 4
batch_size = 32
num_epochs = 35
patience = 5
num_folds = 5
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
class_names = ["diseased cotton leaf", "diseased cotton plant", "fresh cotton leaf", "fresh cotton plant"]

# ==================================
# DATASET & TRANSFORMS
# ==================================
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize([0.5]*3, [0.5]*3)
])

dataset = datasets.ImageFolder(root=train_dir, transform=transform)

# ==================================
# 5-FOLD CROSS VALIDATION
# ==================================
kf = KFold(n_splits=num_folds, shuffle=True, random_state=42)
fold_accuracies = []

for fold, (train_idx, val_idx) in enumerate(kf.split(dataset)):
    print(f"\n========== Fold {fold+1}/{num_folds} ==========")

    train_subset = Subset(dataset, train_idx)
    val_subset   = Subset(dataset, val_idx)

    train_loader = DataLoader(train_subset, batch_size=batch_size, shuffle=True, num_workers=2)
    val_loader   = DataLoader(val_subset, batch_size=batch_size, shuffle=False, num_workers=2)

    # ====================
    # Model Setup
    # ====================
    model = create_model("mobilevit_s", pretrained=True, num_classes=num_classes, drop_rate=0.2)
    model = model.to(device)

    criterion = nn.CrossEntropyLoss()
    optimizer = optim.AdamW(model.parameters(), lr=1e-4, weight_decay=1e-4)
    scheduler = ReduceLROnPlateau(optimizer, mode="min", patience=2, factor=0.5, verbose=False)

    best_val_loss = float("inf")
    patience_counter = 0

    # ====================
    # Training
    # ====================
```

```
    for epoch in range(num_epochs):
        start_time = time.time()
        model.train()
        running_loss, correct = 0, 0

        for imgs, labels in train_loader:
            imgs, labels = imgs.to(device), labels.to(device)
            optimizer.zero_grad()
            outputs = model(imgs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            running_loss += loss.item() * imgs.size(0)
            correct += (outputs.argmax(1) == labels).sum().item()

        train_loss = running_loss / len(train_loader.dataset)
        train_acc = correct / len(train_loader.dataset)

        # ---- Validation ----
        model.eval()
        val_loss, val_correct = 0, 0
        with torch.no_grad():
            for imgs, labels in val_loader:
                imgs, labels = imgs.to(device), labels.to(device)
                outputs = model(imgs)
                loss = criterion(outputs, labels)

                val_loss += loss.item() * imgs.size(0)
                val_correct += (outputs.argmax(1) == labels).sum().item()

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

        elapsed = time.time() - start_time
        print(f"Epoch [{epoch+1}/{num_epochs}] | Train Acc: {train_acc:.4f} | Val Acc: {val_acc:.4f} | Time: {el

        # Early Stopping
        if val_loss < best_val_loss:
            best_val_loss = val_loss
            patience_counter = 0
            torch.save(model.state_dict(), f"fold_{fold+1}_best.pth")
        else:
            patience_counter += 1
            if patience_counter >= patience:
                print("Early stopping triggered!")
                break

    # =====================
    # Evaluation on Validation Set
    # =====================
    model.load_state_dict(torch.load(f"fold_{fold+1}_best.pth"))
    model.eval()
    y_true, y_pred = [], []

    with torch.no_grad():
        for imgs, labels in val_loader:
            imgs, labels = imgs.to(device), labels.to(device)
            outputs = model(imgs)
            preds = outputs.argmax(1)
            y_true.extend(labels.cpu().numpy())
            y_pred.extend(preds.cpu().numpy())

    acc = accuracy_score(y_true, y_pred)
    fold_accuracies.append(acc)
    print(f"Fold {fold+1} Accuracy: {acc:.4f}")

    # =====================
    # MEMORY CLEANUP
    # =====================
    del model, optimizer, scheduler, train_loader, val_loader, train_subset, val_subset
    torch.cuda.empty_cache()
    gc.collect()

# ================================
# RESULTS SUMMARY
# ================================
mean_acc = np.mean(fold_accuracies)
std_acc = np.std(fold_accuracies)
print("\n====================== 5-FOLD CROSS VALIDATION RESULTS ======================")
for i, acc in enumerate(fold_accuracies):
    print(f"Fold {i+1}: {acc:.4f}")
```

```
print(f"\nMean Accuracy: {mean_acc:.4f}")
print(f"Standard Deviation: {std_acc:.4f}")
```

```
========== Fold 1/5 ==========
model.safetensors:   0%|          | 0.00/22.4M [00:00<?, ?B/s]
/usr/local/lib/python3.11/dist-packages/torch/optim/lr_scheduler.py:62: UserWarning: The verbose parameter is deprecated. Please use get_last_lr() to access the learning rate.
  warnings.warn(
Epoch [1/35] | Train Acc: 0.9048 | Val Acc: 0.9920 | Time: 35.12s
Epoch [2/35] | Train Acc: 0.9829 | Val Acc: 0.9935 | Time: 34.29s
Epoch [3/35] | Train Acc: 0.9904 | Val Acc: 0.9964 | Time: 34.14s
Epoch [4/35] | Train Acc: 0.9949 | Val Acc: 0.9956 | Time: 34.27s
Epoch [5/35] | Train Acc: 0.9955 | Val Acc: 0.9993 | Time: 34.24s
Epoch [6/35] | Train Acc: 0.9949 | Val Acc: 0.9971 | Time: 34.21s
Epoch [7/35] | Train Acc: 0.9953 | Val Acc: 0.9949 | Time: 34.42s
Epoch [8/35] | Train Acc: 0.9980 | Val Acc: 0.9971 | Time: 34.26s
Epoch [9/35] | Train Acc: 0.9985 | Val Acc: 0.9985 | Time: 34.25s
Epoch [10/35] | Train Acc: 0.9984 | Val Acc: 0.9942 | Time: 34.40s
Epoch [11/35] | Train Acc: 0.9985 | Val Acc: 0.9978 | Time: 34.13s
Epoch [12/35] | Train Acc: 0.9998 | Val Acc: 0.9978 | Time: 34.19s
Epoch [13/35] | Train Acc: 0.9996 | Val Acc: 0.9985 | Time: 34.32s
Epoch [14/35] | Train Acc: 0.9991 | Val Acc: 0.9964 | Time: 34.32s
Epoch [15/35] | Train Acc: 0.9993 | Val Acc: 0.9978 | Time: 34.11s
Epoch [16/35] | Train Acc: 0.9998 | Val Acc: 0.9978 | Time: 34.23s
Epoch [17/35] | Train Acc: 1.0000 | Val Acc: 0.9985 | Time: 34.43s
Epoch [18/35] | Train Acc: 0.9998 | Val Acc: 0.9993 | Time: 34.30s
Epoch [19/35] | Train Acc: 0.9998 | Val Acc: 0.9993 | Time: 34.18s
Epoch [20/35] | Train Acc: 0.9998 | Val Acc: 0.9993 | Time: 34.13s
Epoch [21/35] | Train Acc: 0.9996 | Val Acc: 0.9978 | Time: 34.36s
Epoch [22/35] | Train Acc: 0.9996 | Val Acc: 0.9978 | Time: 34.32s
Epoch [23/35] | Train Acc: 0.9995 | Val Acc: 0.9978 | Time: 34.38s
Epoch [24/35] | Train Acc: 0.9998 | Val Acc: 0.9978 | Time: 34.30s
Epoch [25/35] | Train Acc: 0.9998 | Val Acc: 0.9978 | Time: 34.20s
Early stopping triggered!
Fold 1 Accuracy: 0.9993

========== Fold 2/5 ==========
/usr/local/lib/python3.11/dist-packages/torch/optim/lr_scheduler.py:62: UserWarning: The verbose parameter is deprecated. Please use get_last_lr() to access the learning rate.
  warnings.warn(
Epoch [1/35] | Train Acc: 0.8776 | Val Acc: 0.9826 | Time: 34.30s
Epoch [2/35] | Train Acc: 0.9793 | Val Acc: 0.9869 | Time: 34.20s
Epoch [3/35] | Train Acc: 0.9880 | Val Acc: 0.9920 | Time: 34.21s
Epoch [4/35] | Train Acc: 0.9906 | Val Acc: 0.9993 | Time: 34.25s
Epoch [5/35] | Train Acc: 0.9933 | Val Acc: 0.9978 | Time: 34.21s
Epoch [6/35] | Train Acc: 0.9978 | Val Acc: 0.9964 | Time: 34.26s
Epoch [7/35] | Train Acc: 0.9964 | Val Acc: 0.9993 | Time: 34.36s
Epoch [8/35] | Train Acc: 0.9940 | Val Acc: 0.9985 | Time: 34.25s
Epoch [9/35] | Train Acc: 0.9978 | Val Acc: 0.9993 | Time: 34.28s
Epoch [10/35] | Train Acc: 0.9973 | Val Acc: 0.9985 | Time: 34.38s
Epoch [11/35] | Train Acc: 0.9989 | Val Acc: 0.9993 | Time: 34.14s
Epoch [12/35] | Train Acc: 0.9995 | Val Acc: 0.9978 | Time: 34.19s
Epoch [13/35] | Train Acc: 0.9987 | Val Acc: 0.9949 | Time: 34.19s
Epoch [14/35] | Train Acc: 0.9973 | Val Acc: 0.9985 | Time: 34.11s
Epoch [15/35] | Train Acc: 0.9993 | Val Acc: 0.9956 | Time: 34.16s
Epoch [16/35] | Train Acc: 0.9996 | Val Acc: 0.9993 | Time: 34.23s
Early stopping triggered!
Fold 2 Accuracy: 0.9993

========== Fold 3/5 ==========
/usr/local/lib/python3.11/dist-packages/torch/optim/lr_scheduler.py:62: UserWarning: The verbose parameter is deprecated. Please use get_last_lr() to access the learning rate.
  warnings.warn(
Epoch [1/35] | Train Acc: 0.8894 | Val Acc: 0.9877 | Time: 34.25s
Epoch [2/35] | Train Acc: 0.9804 | Val Acc: 0.9927 | Time: 34.25s
Epoch [3/35] | Train Acc: 0.9917 | Val Acc: 0.9964 | Time: 34.18s
Epoch [4/35] | Train Acc: 0.9944 | Val Acc: 0.9942 | Time: 34.15s
Epoch [5/35] | Train Acc: 0.9949 | Val Acc: 0.9964 | Time: 34.33s
Epoch [6/35] | Train Acc: 0.9955 | Val Acc: 0.9942 | Time: 34.19s
Epoch [7/35] | Train Acc: 0.9966 | Val Acc: 0.9877 | Time: 34.31s
Epoch [8/35] | Train Acc: 0.9975 | Val Acc: 0.9927 | Time: 34.30s
Epoch [9/35] | Train Acc: 0.9987 | Val Acc: 0.9971 | Time: 34.25s
Epoch [10/35] | Train Acc: 0.9985 | Val Acc: 0.9978 | Time: 34.14s
Epoch [11/35] | Train Acc: 0.9995 | Val Acc: 0.9978 | Time: 34.14s
Epoch [12/35] | Train Acc: 0.9980 | Val Acc: 0.9985 | Time: 34.29s
Epoch [13/35] | Train Acc: 0.9996 | Val Acc: 0.9971 | Time: 34.20s
Epoch [14/35] | Train Acc: 0.9998 | Val Acc: 0.9978 | Time: 34.18s
Epoch [15/35] | Train Acc: 0.9995 | Val Acc: 0.9978 | Time: 34.33s
Early stopping triggered!
Fold 3 Accuracy: 0.9978

========== Fold 4/5 ==========
```

```
/usr/local/lib/python3.11/dist-packages/torch/optim/lr_scheduler.py:62: UserWarning: The verbose parameter is de
precated. Please use get_last_lr() to access the learning rate.
  warnings.warn(
Epoch [1/35] | Train Acc: 0.8932 | Val Acc: 0.9862 | Time: 34.30s
Epoch [2/35] | Train Acc: 0.9839 | Val Acc: 0.9927 | Time: 34.32s
Epoch [3/35] | Train Acc: 0.9900 | Val Acc: 0.9935 | Time: 34.16s
Epoch [4/35] | Train Acc: 0.9895 | Val Acc: 0.9964 | Time: 34.19s
Epoch [5/35] | Train Acc: 0.9962 | Val Acc: 0.9978 | Time: 34.23s
Epoch [6/35] | Train Acc: 0.9956 | Val Acc: 0.9964 | Time: 34.22s
Epoch [7/35] | Train Acc: 0.9975 | Val Acc: 0.9942 | Time: 34.26s
Epoch [8/35] | Train Acc: 0.9982 | Val Acc: 0.9949 | Time: 34.17s
Epoch [9/35] | Train Acc: 0.9967 | Val Acc: 0.9978 | Time: 34.37s
Epoch [10/35] | Train Acc: 0.9985 | Val Acc: 0.9978 | Time: 34.27s
Epoch [11/35] | Train Acc: 0.9995 | Val Acc: 0.9985 | Time: 34.23s
Epoch [12/35] | Train Acc: 0.9987 | Val Acc: 0.9993 | Time: 34.30s
Epoch [13/35] | Train Acc: 0.9998 | Val Acc: 0.9985 | Time: 34.12s
Epoch [14/35] | Train Acc: 0.9998 | Val Acc: 0.9985 | Time: 34.25s
Epoch [15/35] | Train Acc: 0.9991 | Val Acc: 0.9978 | Time: 34.33s
Epoch [16/35] | Train Acc: 0.9987 | Val Acc: 0.9978 | Time: 34.24s
Epoch [17/35] | Train Acc: 0.9996 | Val Acc: 0.9985 | Time: 34.17s
Early stopping triggered!
Fold 4 Accuracy: 0.9993

========== Fold 5/5 ==========
/usr/local/lib/python3.11/dist-packages/torch/optim/lr_scheduler.py:62: UserWarning: The verbose parameter is de
precated. Please use get_last_lr() to access the learning rate.
  warnings.warn(
Epoch [1/35] | Train Acc: 0.8886 | Val Acc: 0.9797 | Time: 34.26s
Epoch [2/35] | Train Acc: 0.9799 | Val Acc: 0.9913 | Time: 34.30s
Epoch [3/35] | Train Acc: 0.9927 | Val Acc: 0.9927 | Time: 34.25s
Epoch [4/35] | Train Acc: 0.9933 | Val Acc: 0.9891 | Time: 34.40s
Epoch [5/35] | Train Acc: 0.9915 | Val Acc: 0.9985 | Time: 34.23s
Epoch [6/35] | Train Acc: 0.9940 | Val Acc: 0.9978 | Time: 34.34s
Epoch [7/35] | Train Acc: 0.9967 | Val Acc: 0.9964 | Time: 34.18s
Epoch [8/35] | Train Acc: 0.9973 | Val Acc: 0.9964 | Time: 34.19s
Epoch [9/35] | Train Acc: 0.9984 | Val Acc: 0.9985 | Time: 34.14s
Epoch [10/35] | Train Acc: 0.9980 | Val Acc: 0.9985 | Time: 34.37s
Epoch [11/35] | Train Acc: 0.9991 | Val Acc: 0.9985 | Time: 34.26s
Epoch [12/35] | Train Acc: 0.9993 | Val Acc: 0.9993 | Time: 34.20s
Epoch [13/35] | Train Acc: 0.9996 | Val Acc: 0.9985 | Time: 34.13s
Epoch [14/35] | Train Acc: 0.9995 | Val Acc: 0.9985 | Time: 34.33s
Epoch [15/35] | Train Acc: 0.9996 | Val Acc: 0.9985 | Time: 34.23s
Epoch [16/35] | Train Acc: 0.9995 | Val Acc: 0.9985 | Time: 34.39s
Epoch [17/35] | Train Acc: 0.9996 | Val Acc: 0.9985 | Time: 34.38s
Early stopping triggered!
Fold 5 Accuracy: 0.9993

====================== 5-FOLD CROSS VALIDATION RESULTS ======================
Fold 1: 0.9993
Fold 2: 0.9993
Fold 3: 0.9978
Fold 4: 0.9993
Fold 5: 0.9993

Mean Accuracy: 0.9990
Standard Deviation: 0.0006
```

In [12]:

```python
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, Subset
from torchvision import datasets, transforms
from timm import create_model
from torch.optim.lr_scheduler import ReduceLROnPlateau
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score
import numpy as np
import time
import gc

# ==================================
# CONFIGURATION
# ==================================
train_dir = "/kaggle/working/cotton_train_aug"
val_dir   = "/kaggle/working/cotton_split/val"
test_dir  = "/kaggle/working/cotton_split/test"

num_classes = 4
batch_size = 32
num_epochs = 35
patience = 5
num_folds = 5
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

```python
class_names = ["diseased cotton leaf", "diseased cotton plant", "fresh cotton leaf", "fresh cotton plant"]

# ================================
# DATASET & TRANSFORMS
# ================================
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize([0.5]*3, [0.5]*3)
])

dataset = datasets.ImageFolder(root=train_dir, transform=transform)

# ================================
# 5-FOLD CROSS VALIDATION
# ================================
kf = KFold(n_splits=num_folds, shuffle=True, random_state=42)
fold_accuracies = []

for fold, (train_idx, val_idx) in enumerate(kf.split(dataset)):
    print(f"\n========== Fold {fold+1}/{num_folds} ==========")

    train_subset = Subset(dataset, train_idx)
    val_subset   = Subset(dataset, val_idx)

    train_loader = DataLoader(train_subset, batch_size=batch_size, shuffle=True, num_workers=2)
    val_loader   = DataLoader(val_subset, batch_size=batch_size, shuffle=False, num_workers=2)

    # ====================
    # Model Setup
    # ====================
    model = create_model("tiny_vit_5m_224", pretrained=True, num_classes=num_classes)
    model = model.to(device)

    criterion = nn.CrossEntropyLoss()
    optimizer = optim.AdamW(model.parameters(), lr=1e-4, weight_decay=1e-4)
    scheduler = ReduceLROnPlateau(optimizer, mode="min", patience=2, factor=0.5, verbose=False)

    best_val_loss = float("inf")
    patience_counter = 0

    # ====================
    # Training
    # ====================
    for epoch in range(num_epochs):
        start_time = time.time()
        model.train()
        running_loss, correct = 0, 0

        for imgs, labels in train_loader:
            imgs, labels = imgs.to(device), labels.to(device)
            optimizer.zero_grad()
            outputs = model(imgs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            running_loss += loss.item() * imgs.size(0)
            correct += (outputs.argmax(1) == labels).sum().item()

        train_loss = running_loss / len(train_loader.dataset)
        train_acc = correct / len(train_loader.dataset)

        # ---- Validation ----
        model.eval()
        val_loss, val_correct = 0, 0
        with torch.no_grad():
            for imgs, labels in val_loader:
                imgs, labels = imgs.to(device), labels.to(device)
                outputs = model(imgs)
                loss = criterion(outputs, labels)

                val_loss += loss.item() * imgs.size(0)
                val_correct += (outputs.argmax(1) == labels).sum().item()

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

        elapsed = time.time() - start_time
        print(f"Epoch [{epoch+1}/{num_epochs}] | Train Acc: {train_acc:.4f} | Val Acc: {val_acc:.4f} | Time: {el

        # Early Stopping
```

```
        if val_loss < best_val_loss:
            best_val_loss = val_loss
            patience_counter = 0
            torch.save(model.state_dict(), f"fold_{fold+1}_best.pth")
        else:
            patience_counter += 1
            if patience_counter >= patience:
                print("Early stopping triggered!")
                break

    # =====================
    # Evaluation on Validation Set
    # =====================
    model.load_state_dict(torch.load(f"fold_{fold+1}_best.pth"))
    model.eval()
    y_true, y_pred = [], []

    with torch.no_grad():
        for imgs, labels in val_loader:
            imgs, labels = imgs.to(device), labels.to(device)
            outputs = model(imgs)
            preds = outputs.argmax(1)
            y_true.extend(labels.cpu().numpy())
            y_pred.extend(preds.cpu().numpy())

    acc = accuracy_score(y_true, y_pred)
    fold_accuracies.append(acc)
    print(f"Fold {fold+1} Accuracy: {acc:.4f}")

    # =====================
    # MEMORY CLEANUP
    # =====================
    del model, optimizer, scheduler, train_loader, val_loader, train_subset, val_subset
    torch.cuda.empty_cache()
    gc.collect()

# ==================================
# RESULTS SUMMARY
# ==================================
mean_acc = np.mean(fold_accuracies)
std_acc = np.std(fold_accuracies)
print("\n====================== 5-FOLD CROSS VALIDATION RESULTS ======================")
for i, acc in enumerate(fold_accuracies):
    print(f"Fold {i+1}: {acc:.4f}")
print(f"\nMean Accuracy: {mean_acc:.4f}")
print(f"Standard Deviation: {std_acc:.4f}")
```

```
========== Fold 1/5 ==========
model.safetensors:   0%|          | 0.00/48.4M [00:00<?, ?B/s]
/usr/local/lib/python3.11/dist-packages/torch/optim/lr_scheduler.py:62: UserWarning: The verbose parameter is deprecated. Please use get_last_lr() to access the learning rate.
  warnings.warn(
Epoch [1/35] | Train Acc: 0.9144 | Val Acc: 0.9898 | Time: 27.11s
Epoch [2/35] | Train Acc: 0.9898 | Val Acc: 0.9978 | Time: 26.65s
Epoch [3/35] | Train Acc: 0.9969 | Val Acc: 0.9978 | Time: 26.80s
Epoch [4/35] | Train Acc: 0.9953 | Val Acc: 0.9978 | Time: 26.72s
Epoch [5/35] | Train Acc: 0.9978 | Val Acc: 0.9978 | Time: 26.76s
Epoch [6/35] | Train Acc: 0.9967 | Val Acc: 0.9985 | Time: 26.63s
Epoch [7/35] | Train Acc: 0.9989 | Val Acc: 0.9971 | Time: 26.72s
Epoch [8/35] | Train Acc: 0.9973 | Val Acc: 0.9993 | Time: 26.83s
Epoch [9/35] | Train Acc: 0.9991 | Val Acc: 0.9978 | Time: 26.73s
Epoch [10/35] | Train Acc: 0.9995 | Val Acc: 0.9978 | Time: 26.71s
Epoch [11/35] | Train Acc: 0.9989 | Val Acc: 0.9978 | Time: 26.77s
Epoch [12/35] | Train Acc: 0.9985 | Val Acc: 0.9978 | Time: 26.86s
Epoch [13/35] | Train Acc: 0.9996 | Val Acc: 0.9964 | Time: 26.83s
Epoch [14/35] | Train Acc: 0.9998 | Val Acc: 0.9949 | Time: 26.72s
Epoch [15/35] | Train Acc: 0.9995 | Val Acc: 1.0000 | Time: 26.91s
Epoch [16/35] | Train Acc: 1.0000 | Val Acc: 0.9985 | Time: 26.85s
Epoch [17/35] | Train Acc: 0.9996 | Val Acc: 0.9993 | Time: 26.73s
Epoch [18/35] | Train Acc: 1.0000 | Val Acc: 0.9978 | Time: 26.79s
Epoch [19/35] | Train Acc: 0.9998 | Val Acc: 0.9993 | Time: 26.83s
Epoch [20/35] | Train Acc: 0.9998 | Val Acc: 0.9993 | Time: 26.82s
Early stopping triggered!
Fold 1 Accuracy: 1.0000

========== Fold 2/5 ==========
/usr/local/lib/python3.11/dist-packages/torch/optim/lr_scheduler.py:62: UserWarning: The verbose parameter is deprecated. Please use get_last_lr() to access the learning rate.
  warnings.warn(
```

```
Epoch [1/35] | Train Acc: 0.9068 | Val Acc: 0.9927 | Time: 27.04s
Epoch [2/35] | Train Acc: 0.9891 | Val Acc: 0.9964 | Time: 26.85s
Epoch [3/35] | Train Acc: 0.9964 | Val Acc: 0.9927 | Time: 26.76s
Epoch [4/35] | Train Acc: 0.9978 | Val Acc: 0.9978 | Time: 26.91s

Epoch [5/35] | Train Acc: 0.9958 | Val Acc: 0.9964 | Time: 26.81s
Epoch [6/35] | Train Acc: 0.9964 | Val Acc: 0.9993 | Time: 26.81s
Epoch [7/35] | Train Acc: 0.9989 | Val Acc: 0.9985 | Time: 26.85s
Epoch [8/35] | Train Acc: 0.9998 | Val Acc: 0.9993 | Time: 26.84s
Epoch [9/35] | Train Acc: 1.0000 | Val Acc: 0.9978 | Time: 26.84s
Epoch [10/35] | Train Acc: 0.9996 | Val Acc: 0.9985 | Time: 26.84s
Epoch [11/35] | Train Acc: 0.9980 | Val Acc: 0.9949 | Time: 26.85s
Epoch [12/35] | Train Acc: 0.9984 | Val Acc: 0.9985 | Time: 26.79s
Epoch [13/35] | Train Acc: 0.9985 | Val Acc: 0.9985 | Time: 27.04s
Epoch [14/35] | Train Acc: 0.9995 | Val Acc: 0.9956 | Time: 26.97s
Epoch [15/35] | Train Acc: 0.9996 | Val Acc: 1.0000 | Time: 26.87s
Epoch [16/35] | Train Acc: 1.0000 | Val Acc: 0.9993 | Time: 26.85s
Epoch [17/35] | Train Acc: 0.9993 | Val Acc: 0.9985 | Time: 26.90s
Epoch [18/35] | Train Acc: 0.9998 | Val Acc: 0.9993 | Time: 26.85s
Epoch [19/35] | Train Acc: 1.0000 | Val Acc: 0.9993 | Time: 26.89s
Epoch [20/35] | Train Acc: 1.0000 | Val Acc: 0.9993 | Time: 26.83s
Early stopping triggered!
Fold 2 Accuracy: 1.0000

========== Fold 3/5 ==========
/usr/local/lib/python3.11/dist-packages/torch/optim/lr_scheduler.py:62: UserWarning: The verbose parameter is de
precated. Please use get_last_lr() to access the learning rate.
  warnings.warn(
Epoch [1/35] | Train Acc: 0.8899 | Val Acc: 0.9840 | Time: 26.75s
Epoch [2/35] | Train Acc: 0.9898 | Val Acc: 0.9891 | Time: 26.75s
Epoch [3/35] | Train Acc: 0.9960 | Val Acc: 0.9913 | Time: 26.92s
Epoch [4/35] | Train Acc: 0.9951 | Val Acc: 0.9949 | Time: 26.93s
Epoch [5/35] | Train Acc: 0.9975 | Val Acc: 0.9942 | Time: 26.75s
Epoch [6/35] | Train Acc: 0.9975 | Val Acc: 0.9971 | Time: 26.94s
Epoch [7/35] | Train Acc: 0.9982 | Val Acc: 0.9971 | Time: 26.95s
Epoch [8/35] | Train Acc: 0.9995 | Val Acc: 0.9978 | Time: 26.76s
Epoch [9/35] | Train Acc: 0.9982 | Val Acc: 0.9964 | Time: 26.84s
Epoch [10/35] | Train Acc: 0.9995 | Val Acc: 0.9956 | Time: 26.77s
Epoch [11/35] | Train Acc: 0.9998 | Val Acc: 0.9985 | Time: 26.86s
Epoch [12/35] | Train Acc: 1.0000 | Val Acc: 0.9978 | Time: 26.84s
Epoch [13/35] | Train Acc: 0.9995 | Val Acc: 0.9971 | Time: 26.85s
Early stopping triggered!
Fold 3 Accuracy: 0.9978

========== Fold 4/5 ==========
/usr/local/lib/python3.11/dist-packages/torch/optim/lr_scheduler.py:62: UserWarning: The verbose parameter is de
precated. Please use get_last_lr() to access the learning rate.
  warnings.warn(
Epoch [1/35] | Train Acc: 0.9120 | Val Acc: 0.9906 | Time: 26.86s
Epoch [2/35] | Train Acc: 0.9898 | Val Acc: 0.9978 | Time: 26.70s
Epoch [3/35] | Train Acc: 0.9962 | Val Acc: 0.9956 | Time: 26.82s
Epoch [4/35] | Train Acc: 0.9984 | Val Acc: 0.9956 | Time: 26.87s
Epoch [5/35] | Train Acc: 0.9980 | Val Acc: 0.9956 | Time: 26.83s
Epoch [6/35] | Train Acc: 0.9960 | Val Acc: 0.9971 | Time: 26.93s
Epoch [7/35] | Train Acc: 0.9980 | Val Acc: 0.9971 | Time: 26.90s
Epoch [8/35] | Train Acc: 0.9993 | Val Acc: 0.9964 | Time: 26.82s
Epoch [9/35] | Train Acc: 0.9980 | Val Acc: 0.9978 | Time: 26.84s
Epoch [10/35] | Train Acc: 0.9980 | Val Acc: 0.9964 | Time: 26.93s
Epoch [11/35] | Train Acc: 0.9993 | Val Acc: 0.9978 | Time: 26.86s
Epoch [12/35] | Train Acc: 0.9995 | Val Acc: 0.9971 | Time: 26.90s
Early stopping triggered!
Fold 4 Accuracy: 0.9971

========== Fold 5/5 ==========
/usr/local/lib/python3.11/dist-packages/torch/optim/lr_scheduler.py:62: UserWarning: The verbose parameter is de
precated. Please use get_last_lr() to access the learning rate.
  warnings.warn(
```

```
Epoch [1/35] | Train Acc: 0.8954 | Val Acc: 0.9913 | Time: 26.75s
Epoch [2/35] | Train Acc: 0.9908 | Val Acc: 0.9964 | Time: 26.82s
Epoch [3/35] | Train Acc: 0.9955 | Val Acc: 0.9724 | Time: 26.84s
Epoch [4/35] | Train Acc: 0.9958 | Val Acc: 0.9782 | Time: 26.93s
Epoch [5/35] | Train Acc: 0.9975 | Val Acc: 0.9956 | Time: 27.07s
Epoch [6/35] | Train Acc: 0.9975 | Val Acc: 0.9978 | Time: 26.81s
Epoch [7/35] | Train Acc: 0.9998 | Val Acc: 0.9985 | Time: 26.86s
Epoch [8/35] | Train Acc: 0.9989 | Val Acc: 0.9935 | Time: 26.89s
Epoch [9/35] | Train Acc: 0.9993 | Val Acc: 0.9971 | Time: 26.87s
Epoch [10/35] | Train Acc: 0.9951 | Val Acc: 0.9906 | Time: 26.78s
Epoch [11/35] | Train Acc: 0.9989 | Val Acc: 0.9978 | Time: 26.98s
Epoch [12/35] | Train Acc: 0.9995 | Val Acc: 0.9978 | Time: 26.75s
Early stopping triggered!
Fold 5 Accuracy: 0.9985

====================== 5-FOLD CROSS VALIDATION RESULTS ======================
Fold 1: 1.0000
Fold 2: 1.0000
Fold 3: 0.9978
Fold 4: 0.9971
Fold 5: 0.9985

Mean Accuracy: 0.9987
Standard Deviation: 0.0012
```

In [13]:

```python
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, Subset
from torchvision import datasets, transforms
from timm import create_model
from torch.optim.lr_scheduler import ReduceLROnPlateau
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score
import numpy as np
import time
import gc

# ==================================
# CONFIGURATION
# ==================================
train_dir = "/kaggle/working/cotton_train_aug"
val_dir   = "/kaggle/working/cotton_split/val"
test_dir  = "/kaggle/working/cotton_split/test"

num_classes = 4
batch_size = 32
num_epochs = 35
patience = 5
num_folds = 5
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
class_names = ["diseased cotton leaf", "diseased cotton plant", "fresh cotton leaf", "fresh cotton plant"]

# ==================================
# DATASET & TRANSFORMS
# ==================================
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize([0.5]*3, [0.5]*3)
])

dataset = datasets.ImageFolder(root=train_dir, transform=transform)

# ==================================
# 5-FOLD CROSS VALIDATION
# ==================================
kf = KFold(n_splits=num_folds, shuffle=True, random_state=42)
fold_accuracies = []

for fold, (train_idx, val_idx) in enumerate(kf.split(dataset)):
    print(f"\n========== Fold {fold+1}/{num_folds} ==========")

    train_subset = Subset(dataset, train_idx)
    val_subset   = Subset(dataset, val_idx)

    train_loader = DataLoader(train_subset, batch_size=batch_size, shuffle=True, num_workers=2)
    val_loader   = DataLoader(val_subset, batch_size=batch_size, shuffle=False, num_workers=2)

    # ====================
    # Model Setup
    # ====================
    model = create_model("levit_128", pretrained=True, num_classes=num_classes)  # smaller LeViT
```

```
    model = model.to(device)

    criterion = nn.CrossEntropyLoss()
    optimizer = optim.AdamW(model.parameters(), lr=1e-4, weight_decay=1e-4)
    scheduler = ReduceLROnPlateau(optimizer, mode="min", patience=2, factor=0.5, verbose=False)

    best_val_loss = float("inf")
    patience_counter = 0

    # =====================
    # Training
    # =====================
    for epoch in range(num_epochs):
        start_time = time.time()
        model.train()
        running_loss, correct = 0, 0

        for imgs, labels in train_loader:
            imgs, labels = imgs.to(device), labels.to(device)
            optimizer.zero_grad()
            outputs = model(imgs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            running_loss += loss.item() * imgs.size(0)
            correct += (outputs.argmax(1) == labels).sum().item()

        train_loss = running_loss / len(train_loader.dataset)
        train_acc = correct / len(train_loader.dataset)

        # ---- Validation ----
        model.eval()
        val_loss, val_correct = 0, 0
        with torch.no_grad():
            for imgs, labels in val_loader:
                imgs, labels = imgs.to(device), labels.to(device)
                outputs = model(imgs)
                loss = criterion(outputs, labels)

                val_loss += loss.item() * imgs.size(0)
                val_correct += (outputs.argmax(1) == labels).sum().item()

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

        elapsed = time.time() - start_time
        print(f"Epoch [{epoch+1}/{num_epochs}] | Train Acc: {train_acc:.4f} | Val Acc: {val_acc:.4f} | Time: {el

        # Early Stopping
        if val_loss < best_val_loss:
            best_val_loss = val_loss
            patience_counter = 0
            torch.save(model.state_dict(), f"fold_{fold+1}_best.pth")
        else:
            patience_counter += 1
            if patience_counter >= patience:
                print("Early stopping triggered!")
                break

    # =====================
    # Evaluation on Validation Set
    # =====================
    model.load_state_dict(torch.load(f"fold_{fold+1}_best.pth"))
    model.eval()
    y_true, y_pred = [], []

    with torch.no_grad():
        for imgs, labels in val_loader:
            imgs, labels = imgs.to(device), labels.to(device)
            outputs = model(imgs)
            preds = outputs.argmax(1)
            y_true.extend(labels.cpu().numpy())
            y_pred.extend(preds.cpu().numpy())

    acc = accuracy_score(y_true, y_pred)
    fold_accuracies.append(acc)
    print(f"Fold {fold+1} Accuracy: {acc:.4f}")

    # =====================
    # MEMORY CLEANUP
    # =====================
```

```python
    del model, optimizer, scheduler, train_loader, val_loader, train_subset, val_subset
    torch.cuda.empty_cache()
    gc.collect()

# ==================================
# RESULTS SUMMARY
# ==================================
mean_acc = np.mean(fold_accuracies)
std_acc = np.std(fold_accuracies)
print("\n====================== 5-FOLD CROSS VALIDATION RESULTS ======================")
for i, acc in enumerate(fold_accuracies):
    print(f"Fold {i+1}: {acc:.4f}")
print(f"\nMean Accuracy: {mean_acc:.4f}")
print(f"Standard Deviation: {std_acc:.4f}")
```

```
========== Fold 1/5 ==========
model.safetensors:   0%|          | 0.00/37.1M [00:00<?, ?B/s]
/usr/local/lib/python3.11/dist-packages/torch/optim/lr_scheduler.py:62: UserWarning: The verbose parameter is deprecated. Please use get_last_lr() to access the learning rate.
  warnings.warn(
Epoch [1/35] | Train Acc: 0.7684 | Val Acc: 0.8767 | Time: 14.25s
Epoch [2/35] | Train Acc: 0.9207 | Val Acc: 0.9384 | Time: 14.20s
Epoch [3/35] | Train Acc: 0.9478 | Val Acc: 0.9565 | Time: 14.38s
Epoch [4/35] | Train Acc: 0.9619 | Val Acc: 0.9434 | Time: 14.01s
Epoch [5/35] | Train Acc: 0.9746 | Val Acc: 0.9463 | Time: 13.89s
Epoch [6/35] | Train Acc: 0.9777 | Val Acc: 0.9594 | Time: 13.54s
Epoch [7/35] | Train Acc: 0.9797 | Val Acc: 0.9601 | Time: 14.03s
Epoch [8/35] | Train Acc: 0.9844 | Val Acc: 0.9608 | Time: 13.71s
Early stopping triggered!
Fold 1 Accuracy: 0.9565

========== Fold 2/5 ==========
/usr/local/lib/python3.11/dist-packages/torch/optim/lr_scheduler.py:62: UserWarning: The verbose parameter is deprecated. Please use get_last_lr() to access the learning rate.
  warnings.warn(
Epoch [1/35] | Train Acc: 0.7731 | Val Acc: 0.9333 | Time: 13.80s
Epoch [2/35] | Train Acc: 0.9469 | Val Acc: 0.9746 | Time: 14.15s
Epoch [3/35] | Train Acc: 0.9755 | Val Acc: 0.9840 | Time: 13.64s
Epoch [4/35] | Train Acc: 0.9842 | Val Acc: 0.9855 | Time: 13.82s
Epoch [5/35] | Train Acc: 0.9931 | Val Acc: 0.9891 | Time: 14.36s
Epoch [6/35] | Train Acc: 0.9960 | Val Acc: 0.9869 | Time: 14.00s
Epoch [7/35] | Train Acc: 0.9942 | Val Acc: 0.9869 | Time: 14.15s
Epoch [8/35] | Train Acc: 0.9964 | Val Acc: 0.9869 | Time: 13.95s
Epoch [9/35] | Train Acc: 0.9971 | Val Acc: 0.9877 | Time: 14.22s
Epoch [10/35] | Train Acc: 0.9973 | Val Acc: 0.9869 | Time: 14.11s
Early stopping triggered!
Fold 2 Accuracy: 0.9891

========== Fold 3/5 ==========
/usr/local/lib/python3.11/dist-packages/torch/optim/lr_scheduler.py:62: UserWarning: The verbose parameter is deprecated. Please use get_last_lr() to access the learning rate.
  warnings.warn(
Epoch [1/35] | Train Acc: 0.7711 | Val Acc: 0.9267 | Time: 13.74s
Epoch [2/35] | Train Acc: 0.9438 | Val Acc: 0.9557 | Time: 14.05s
Epoch [3/35] | Train Acc: 0.9672 | Val Acc: 0.9739 | Time: 14.04s
Epoch [4/35] | Train Acc: 0.9802 | Val Acc: 0.9855 | Time: 13.76s
Epoch [5/35] | Train Acc: 0.9884 | Val Acc: 0.9826 | Time: 13.96s
Epoch [6/35] | Train Acc: 0.9944 | Val Acc: 0.9855 | Time: 14.12s
Epoch [7/35] | Train Acc: 0.9947 | Val Acc: 0.9826 | Time: 14.01s
Epoch [8/35] | Train Acc: 0.9958 | Val Acc: 0.9884 | Time: 14.07s
Epoch [9/35] | Train Acc: 0.9982 | Val Acc: 0.9898 | Time: 13.63s
Early stopping triggered!
Fold 3 Accuracy: 0.9855

========== Fold 4/5 ==========
/usr/local/lib/python3.11/dist-packages/torch/optim/lr_scheduler.py:62: UserWarning: The verbose parameter is deprecated. Please use get_last_lr() to access the learning rate.
  warnings.warn(
Epoch [1/35] | Train Acc: 0.7807 | Val Acc: 0.9151 | Time: 13.95s
Epoch [2/35] | Train Acc: 0.9354 | Val Acc: 0.9724 | Time: 13.92s
Epoch [3/35] | Train Acc: 0.9625 | Val Acc: 0.9688 | Time: 14.08s
Epoch [4/35] | Train Acc: 0.9764 | Val Acc: 0.9790 | Time: 14.15s
Epoch [5/35] | Train Acc: 0.9833 | Val Acc: 0.9833 | Time: 14.25s
Epoch [6/35] | Train Acc: 0.9820 | Val Acc: 0.9804 | Time: 14.30s
Epoch [7/35] | Train Acc: 0.9873 | Val Acc: 0.9840 | Time: 14.01s
Epoch [8/35] | Train Acc: 0.9886 | Val Acc: 0.9731 | Time: 13.65s
Epoch [9/35] | Train Acc: 0.9908 | Val Acc: 0.9848 | Time: 13.61s
Epoch [10/35] | Train Acc: 0.9933 | Val Acc: 0.9855 | Time: 14.00s
Early stopping triggered!
Fold 4 Accuracy: 0.9833

========== Fold 5/5 ==========
```

```
/usr/local/lib/python3.11/dist-packages/torch/optim/lr_scheduler.py:62: UserWarning: The verbose parameter is de
precated. Please use get_last_lr() to access the learning rate.
  warnings.warn(
Epoch [1/35] | Train Acc: 0.7691 | Val Acc: 0.9180 | Time: 13.87s
Epoch [2/35] | Train Acc: 0.9483 | Val Acc: 0.9485 | Time: 13.58s
Epoch [3/35] | Train Acc: 0.9723 | Val Acc: 0.9565 | Time: 14.10s
Epoch [4/35] | Train Acc: 0.9875 | Val Acc: 0.9739 | Time: 13.81s
Epoch [5/35] | Train Acc: 0.9877 | Val Acc: 0.9768 | Time: 13.67s
Epoch [6/35] | Train Acc: 0.9726 | Val Acc: 0.9485 | Time: 14.22s
Epoch [7/35] | Train Acc: 0.9657 | Val Acc: 0.9731 | Time: 14.27s
Epoch [8/35] | Train Acc: 0.9726 | Val Acc: 0.9731 | Time: 13.90s
Epoch [9/35] | Train Acc: 0.9819 | Val Acc: 0.9710 | Time: 13.92s
Epoch [10/35] | Train Acc: 0.9828 | Val Acc: 0.9695 | Time: 13.83s
Epoch [11/35] | Train Acc: 0.9918 | Val Acc: 0.9652 | Time: 13.86s
Epoch [12/35] | Train Acc: 0.9893 | Val Acc: 0.9695 | Time: 13.83s
Early stopping triggered!
Fold 5 Accuracy: 0.9731

====================== 5-FOLD CROSS VALIDATION RESULTS ======================
Fold 1: 0.9565
Fold 2: 0.9891
Fold 3: 0.9855
Fold 4: 0.9833
Fold 5: 0.9731

Mean Accuracy: 0.9775
Standard Deviation: 0.0118
```

In [14]:

```python
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, Subset
from torchvision import datasets, transforms
from timm import create_model
from torch.optim.lr_scheduler import ReduceLROnPlateau
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score
import numpy as np
import time
import gc

# ==================================
# CONFIGURATION
# ==================================
train_dir = "/kaggle/working/cotton_train_aug"
val_dir   = "/kaggle/working/cotton_split/val"
test_dir  = "/kaggle/working/cotton_split/test"

num_classes = 4
batch_size = 32
num_epochs = 35
patience = 5
num_folds = 5
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
class_names = ["diseased cotton leaf", "diseased cotton plant", "fresh cotton leaf", "fresh cotton plant"]

# ==================================
# DATASET & TRANSFORMS
# ==================================
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize([0.5]*3, [0.5]*3)
])

dataset = datasets.ImageFolder(root=train_dir, transform=transform)

# ==================================
# 5-FOLD CROSS VALIDATION
# ==================================
kf = KFold(n_splits=num_folds, shuffle=True, random_state=42)
fold_accuracies = []

for fold, (train_idx, val_idx) in enumerate(kf.split(dataset)):
    print(f"\n========== Fold {fold+1}/{num_folds} ==========")

    train_subset = Subset(dataset, train_idx)
    val_subset   = Subset(dataset, val_idx)

    train_loader = DataLoader(train_subset, batch_size=batch_size, shuffle=True, num_workers=2)
    val_loader   = DataLoader(val_subset, batch_size=batch_size, shuffle=False, num_workers=2)
```

```python
    # ====================
    # Model Setup
    # ====================
    model = create_model("deit_tiny_patch16_224", pretrained=True, num_classes=num_classes)
    model = model.to(device)

    criterion = nn.CrossEntropyLoss()
    optimizer = optim.AdamW(model.parameters(), lr=1e-4, weight_decay=1e-4)
    scheduler = ReduceLROnPlateau(optimizer, mode="min", patience=2, factor=0.5, verbose=False)

    best_val_loss = float("inf")
    patience_counter = 0

    # ====================
    # Training
    # ====================
    for epoch in range(num_epochs):
        start_time = time.time()
        model.train()
        running_loss, correct = 0, 0

        for imgs, labels in train_loader:
            imgs, labels = imgs.to(device), labels.to(device)
            optimizer.zero_grad()
            outputs = model(imgs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            running_loss += loss.item() * imgs.size(0)
            correct += (outputs.argmax(1) == labels).sum().item()

        train_loss = running_loss / len(train_loader.dataset)
        train_acc = correct / len(train_loader.dataset)

        # ---- Validation ----
        model.eval()
        val_loss, val_correct = 0, 0
        with torch.no_grad():
            for imgs, labels in val_loader:
                imgs, labels = imgs.to(device), labels.to(device)
                outputs = model(imgs)
                loss = criterion(outputs, labels)

                val_loss += loss.item() * imgs.size(0)
                val_correct += (outputs.argmax(1) == labels).sum().item()

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

        elapsed = time.time() - start_time
        print(f"Epoch [{epoch+1}/{num_epochs}] | Train Acc: {train_acc:.4f} | Val Acc: {val_acc:.4f} | Time: {el

        # Early Stopping
        if val_loss < best_val_loss:
            best_val_loss = val_loss
            patience_counter = 0
            torch.save(model.state_dict(), f"fold_{fold+1}_best.pth")
        else:
            patience_counter += 1
            if patience_counter >= patience:
                print("Early stopping triggered!")
                break

    # ====================
    # Evaluation on Validation Set
    # ====================
    model.load_state_dict(torch.load(f"fold_{fold+1}_best.pth"))
    model.eval()
    y_true, y_pred = [], []

    with torch.no_grad():
        for imgs, labels in val_loader:
            imgs, labels = imgs.to(device), labels.to(device)
            outputs = model(imgs)
            preds = outputs.argmax(1)
            y_true.extend(labels.cpu().numpy())
            y_pred.extend(preds.cpu().numpy())

    acc = accuracy_score(y_true, y_pred)
    fold_accuracies.append(acc)
    print(f"Fold {fold+1} Accuracy: {acc:.4f}")
```

```
    # ====================
    # MEMORY CLEANUP
    # ====================
    del model, optimizer, scheduler, train_loader, val_loader, train_subset, val_subset
    torch.cuda.empty_cache()
    gc.collect()

# ================================
# RESULTS SUMMARY
# ================================
mean_acc = np.mean(fold_accuracies)
std_acc = np.std(fold_accuracies)
print("\n====================== 5-FOLD CROSS VALIDATION RESULTS ======================")
for i, acc in enumerate(fold_accuracies):
    print(f"Fold {i+1}: {acc:.4f}")
print(f"\nMean Accuracy: {mean_acc:.4f}")
print(f"Standard Deviation: {std_acc:.4f}")
```

```
========== Fold 1/5 ==========
model.safetensors:   0%|          | 0.00/22.9M [00:00<?, ?B/s]
/usr/local/lib/python3.11/dist-packages/torch/optim/lr_scheduler.py:62: UserWarning: The verbose parameter is de
precated. Please use get_last_lr() to access the learning rate.
  warnings.warn(
Epoch [1/35] | Train Acc: 0.9356 | Val Acc: 0.9761 | Time: 17.10s
Epoch [2/35] | Train Acc: 0.9906 | Val Acc: 0.9891 | Time: 17.08s
Epoch [3/35] | Train Acc: 0.9964 | Val Acc: 0.9434 | Time: 17.12s
Epoch [4/35] | Train Acc: 0.9868 | Val Acc: 0.9659 | Time: 17.15s
Epoch [5/35] | Train Acc: 0.9976 | Val Acc: 0.9913 | Time: 17.16s
Epoch [6/35] | Train Acc: 0.9995 | Val Acc: 0.9920 | Time: 17.06s
Epoch [7/35] | Train Acc: 1.0000 | Val Acc: 0.9906 | Time: 17.31s
Epoch [8/35] | Train Acc: 0.9893 | Val Acc: 0.9877 | Time: 17.36s
Epoch [9/35] | Train Acc: 0.9993 | Val Acc: 0.9927 | Time: 17.05s
Epoch [10/35] | Train Acc: 1.0000 | Val Acc: 0.9935 | Time: 17.27s
Epoch [11/35] | Train Acc: 1.0000 | Val Acc: 0.9935 | Time: 17.26s
Early stopping triggered!
Fold 1 Accuracy: 0.9920

========== Fold 2/5 ==========
/usr/local/lib/python3.11/dist-packages/torch/optim/lr_scheduler.py:62: UserWarning: The verbose parameter is de
precated. Please use get_last_lr() to access the learning rate.
  warnings.warn(
Epoch [1/35] | Train Acc: 0.9391 | Val Acc: 0.9797 | Time: 17.28s
Epoch [2/35] | Train Acc: 0.9864 | Val Acc: 0.9906 | Time: 17.25s
Epoch [3/35] | Train Acc: 0.9929 | Val Acc: 0.9906 | Time: 17.12s
Epoch [4/35] | Train Acc: 0.9955 | Val Acc: 0.9630 | Time: 17.25s
Epoch [5/35] | Train Acc: 0.9964 | Val Acc: 0.9869 | Time: 17.31s
Epoch [6/35] | Train Acc: 0.9991 | Val Acc: 0.9942 | Time: 17.38s
Epoch [7/35] | Train Acc: 1.0000 | Val Acc: 0.9942 | Time: 17.31s
Epoch [8/35] | Train Acc: 1.0000 | Val Acc: 0.9942 | Time: 17.27s
Epoch [9/35] | Train Acc: 1.0000 | Val Acc: 0.9942 | Time: 17.11s
Epoch [10/35] | Train Acc: 1.0000 | Val Acc: 0.9942 | Time: 17.33s
Epoch [11/35] | Train Acc: 1.0000 | Val Acc: 0.9949 | Time: 17.14s
Epoch [12/35] | Train Acc: 1.0000 | Val Acc: 0.9949 | Time: 17.17s
Epoch [13/35] | Train Acc: 1.0000 | Val Acc: 0.9949 | Time: 17.09s
Early stopping triggered!
Fold 2 Accuracy: 0.9942

========== Fold 3/5 ==========
/usr/local/lib/python3.11/dist-packages/torch/optim/lr_scheduler.py:62: UserWarning: The verbose parameter is de
precated. Please use get_last_lr() to access the learning rate.
  warnings.warn(
Epoch [1/35] | Train Acc: 0.9284 | Val Acc: 0.9717 | Time: 17.43s
Epoch [2/35] | Train Acc: 0.9857 | Val Acc: 0.9833 | Time: 17.18s
Epoch [3/35] | Train Acc: 0.9964 | Val Acc: 0.9790 | Time: 17.25s
Epoch [4/35] | Train Acc: 0.9924 | Val Acc: 0.9833 | Time: 17.19s
Epoch [5/35] | Train Acc: 0.9938 | Val Acc: 0.9884 | Time: 17.37s
Epoch [6/35] | Train Acc: 0.9949 | Val Acc: 0.9949 | Time: 17.18s
Epoch [7/35] | Train Acc: 0.9956 | Val Acc: 0.9920 | Time: 17.33s
Epoch [8/35] | Train Acc: 0.9995 | Val Acc: 0.9913 | Time: 17.39s
Epoch [9/35] | Train Acc: 1.0000 | Val Acc: 0.9891 | Time: 17.41s
Epoch [10/35] | Train Acc: 1.0000 | Val Acc: 0.9898 | Time: 17.36s
Epoch [11/35] | Train Acc: 1.0000 | Val Acc: 0.9906 | Time: 17.21s
Early stopping triggered!
Fold 3 Accuracy: 0.9949

========== Fold 4/5 ==========
/usr/local/lib/python3.11/dist-packages/torch/optim/lr_scheduler.py:62: UserWarning: The verbose parameter is de
precated. Please use get_last_lr() to access the learning rate.
  warnings.warn(
```

```
Epoch [1/35] | Train Acc: 0.9280 | Val Acc: 0.9855 | Time: 17.13s
Epoch [2/35] | Train Acc: 0.9909 | Val Acc: 0.9623 | Time: 17.20s
Epoch [3/35] | Train Acc: 0.9873 | Val Acc: 0.9855 | Time: 17.32s
Epoch [4/35] | Train Acc: 0.9913 | Val Acc: 0.9862 | Time: 17.12s
Epoch [5/35] | Train Acc: 0.9976 | Val Acc: 0.9819 | Time: 17.26s
Epoch [6/35] | Train Acc: 0.9991 | Val Acc: 0.9906 | Time: 17.18s
Epoch [7/35] | Train Acc: 0.9975 | Val Acc: 0.9920 | Time: 17.33s
Epoch [8/35] | Train Acc: 0.9859 | Val Acc: 0.9848 | Time: 17.25s
Epoch [9/35] | Train Acc: 0.9971 | Val Acc: 0.9978 | Time: 17.40s
Epoch [10/35] | Train Acc: 0.9993 | Val Acc: 0.9906 | Time: 17.30s
Epoch [11/35] | Train Acc: 0.9958 | Val Acc: 0.9956 | Time: 17.17s
Epoch [12/35] | Train Acc: 0.9998 | Val Acc: 0.9956 | Time: 17.28s
Epoch [13/35] | Train Acc: 1.0000 | Val Acc: 0.9971 | Time: 17.29s
Epoch [14/35] | Train Acc: 1.0000 | Val Acc: 0.9971 | Time: 17.27s
Early stopping triggered!
Fold 4 Accuracy: 0.9978

========== Fold 5/5 ==========
/usr/local/lib/python3.11/dist-packages/torch/optim/lr_scheduler.py:62: UserWarning: The verbose parameter is de
precated. Please use get_last_lr() to access the learning rate.
  warnings.warn(
Epoch [1/35] | Train Acc: 0.9343 | Val Acc: 0.9819 | Time: 17.20s
Epoch [2/35] | Train Acc: 0.9906 | Val Acc: 0.9840 | Time: 17.31s
Epoch [3/35] | Train Acc: 0.9942 | Val Acc: 0.9906 | Time: 17.24s
Epoch [4/35] | Train Acc: 0.9918 | Val Acc: 0.9848 | Time: 17.17s
Epoch [5/35] | Train Acc: 0.9955 | Val Acc: 0.9790 | Time: 17.31s
Epoch [6/35] | Train Acc: 0.9995 | Val Acc: 0.9942 | Time: 17.18s
Epoch [7/35] | Train Acc: 0.9946 | Val Acc: 0.9826 | Time: 17.18s
Epoch [8/35] | Train Acc: 0.9980 | Val Acc: 0.9949 | Time: 17.13s
Epoch [9/35] | Train Acc: 0.9998 | Val Acc: 0.9949 | Time: 17.31s
Epoch [10/35] | Train Acc: 1.0000 | Val Acc: 0.9949 | Time: 17.15s
Epoch [11/35] | Train Acc: 1.0000 | Val Acc: 0.9949 | Time: 17.22s
Early stopping triggered!
Fold 5 Accuracy: 0.9942

====================== 5-FOLD CROSS VALIDATION RESULTS ======================
Fold 1: 0.9920
Fold 2: 0.9942
Fold 3: 0.9949
Fold 4: 0.9978
Fold 5: 0.9942

Mean Accuracy: 0.9946
Standard Deviation: 0.0019
```

In [15]:

```python
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, Subset
from torchvision import datasets, transforms
from timm import create_model
from torch.optim.lr_scheduler import ReduceLROnPlateau
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score
import numpy as np
import time
import gc

# ==================================
# CONFIGURATION
# ==================================
train_dir = "/kaggle/working/cotton_train_aug"
val_dir   = "/kaggle/working/cotton_split/val"
test_dir  = "/kaggle/working/cotton_split/test"

num_classes = 4
batch_size = 32
num_epochs = 35
patience = 5
num_folds = 5
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
class_names = ["diseased cotton leaf", "diseased cotton plant", "fresh cotton leaf", "fresh cotton plant"]

# ==================================
# DATASET & TRANSFORMS
# ==================================
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize([0.5]*3, [0.5]*3)
])

dataset = datasets.ImageFolder(root=train_dir, transform=transform)
```

```python
# ==================================
# 5-FOLD CROSS VALIDATION
# ==================================
kf = KFold(n_splits=num_folds, shuffle=True, random_state=42)
fold_accuracies = []

for fold, (train_idx, val_idx) in enumerate(kf.split(dataset)):
    print(f"\n========== Fold {fold+1}/{num_folds} ==========")

    train_subset = Subset(dataset, train_idx)
    val_subset   = Subset(dataset, val_idx)

    train_loader = DataLoader(train_subset, batch_size=batch_size, shuffle=True, num_workers=2)
    val_loader   = DataLoader(val_subset, batch_size=batch_size, shuffle=False, num_workers=2)

    # ====================
    # Model Setup
    # ====================
    model = create_model("swin_tiny_patch4_window7_224", pretrained=True, num_classes=num_classes)
    model = model.to(device)

    criterion = nn.CrossEntropyLoss()
    optimizer = optim.AdamW(model.parameters(), lr=1e-4, weight_decay=1e-4)
    scheduler = ReduceLROnPlateau(optimizer, mode="min", patience=2, factor=0.5, verbose=False)

    best_val_loss = float("inf")
    patience_counter = 0

    # ====================
    # Training
    # ====================
    for epoch in range(num_epochs):
        start_time = time.time()
        model.train()
        running_loss, correct = 0, 0

        for imgs, labels in train_loader:
            imgs, labels = imgs.to(device), labels.to(device)
            optimizer.zero_grad()
            outputs = model(imgs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            running_loss += loss.item() * imgs.size(0)
            correct += (outputs.argmax(1) == labels).sum().item()

        train_loss = running_loss / len(train_loader.dataset)
        train_acc = correct / len(train_loader.dataset)

        # ---- Validation ----
        model.eval()
        val_loss, val_correct = 0, 0
        with torch.no_grad():
            for imgs, labels in val_loader:
                imgs, labels = imgs.to(device), labels.to(device)
                outputs = model(imgs)
                loss = criterion(outputs, labels)

                val_loss += loss.item() * imgs.size(0)
                val_correct += (outputs.argmax(1) == labels).sum().item()

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

        elapsed = time.time() - start_time
        print(f"Epoch [{epoch+1}/{num_epochs}] | Train Acc: {train_acc:.4f} | Val Acc: {val_acc:.4f} | Time: {el

        # Early Stopping
        if val_loss < best_val_loss:
            best_val_loss = val_loss
            patience_counter = 0
            torch.save(model.state_dict(), f"fold_{fold+1}_best.pth")
        else:
            patience_counter += 1
            if patience_counter >= patience:
                print("Early stopping triggered!")
                break

    # ====================
    # Evaluation on Validation Set
```

```
    # ====================
    model.load_state_dict(torch.load(f"fold_{fold+1}_best.pth"))
    model.eval()
    y_true, y_pred = [], []

    with torch.no_grad():
        for imgs, labels in val_loader:
            imgs, labels = imgs.to(device), labels.to(device)
            outputs = model(imgs)
            preds = outputs.argmax(1)
            y_true.extend(labels.cpu().numpy())
            y_pred.extend(preds.cpu().numpy())

    acc = accuracy_score(y_true, y_pred)
    fold_accuracies.append(acc)
    print(f"Fold {fold+1} Accuracy: {acc:.4f}")

    # ====================
    # MEMORY CLEANUP
    # ====================
    del model, optimizer, scheduler, train_loader, val_loader, train_subset, val_subset
    torch.cuda.empty_cache()
    gc.collect()

# ==================================
# RESULTS SUMMARY
# ==================================
mean_acc = np.mean(fold_accuracies)
std_acc = np.std(fold_accuracies)
print("\n====================== 5-FOLD CROSS VALIDATION RESULTS ======================")
for i, acc in enumerate(fold_accuracies):
    print(f"Fold {i+1}: {acc:.4f}")
print(f"\nMean Accuracy: {mean_acc:.4f}")
print(f"Standard Deviation: {std_acc:.4f}")
```

```
========== Fold 1/5 ==========
model.safetensors:   0%|          | 0.00/114M [00:00<?, ?B/s]
/usr/local/lib/python3.11/dist-packages/torch/optim/lr_scheduler.py:62: UserWarning: The verbose parameter is deprecated. Please use get_last_lr() to access the learning rate.
  warnings.warn(
Epoch [1/35] | Train Acc: 0.9407 | Val Acc: 0.9840 | Time: 48.34s
Epoch [2/35] | Train Acc: 0.9891 | Val Acc: 0.9920 | Time: 48.38s
Epoch [3/35] | Train Acc: 0.9975 | Val Acc: 0.9971 | Time: 48.19s
Epoch [4/35] | Train Acc: 0.9929 | Val Acc: 0.9811 | Time: 48.35s
Epoch [5/35] | Train Acc: 0.9918 | Val Acc: 0.9949 | Time: 48.20s
Epoch [6/35] | Train Acc: 0.9996 | Val Acc: 0.9942 | Time: 48.15s
Epoch [7/35] | Train Acc: 1.0000 | Val Acc: 0.9964 | Time: 48.17s
Epoch [8/35] | Train Acc: 0.9995 | Val Acc: 0.9964 | Time: 48.21s
Early stopping triggered!
Fold 1 Accuracy: 0.9971

========== Fold 2/5 ==========
/usr/local/lib/python3.11/dist-packages/torch/optim/lr_scheduler.py:62: UserWarning: The verbose parameter is deprecated. Please use get_last_lr() to access the learning rate.
  warnings.warn(
Epoch [1/35] | Train Acc: 0.9231 | Val Acc: 0.9927 | Time: 48.31s
Epoch [2/35] | Train Acc: 0.9891 | Val Acc: 0.9964 | Time: 48.18s
Epoch [3/35] | Train Acc: 0.9931 | Val Acc: 0.9920 | Time: 48.16s
Epoch [4/35] | Train Acc: 0.9891 | Val Acc: 0.9884 | Time: 48.16s
Epoch [5/35] | Train Acc: 0.9967 | Val Acc: 0.9971 | Time: 48.10s
Epoch [6/35] | Train Acc: 0.9989 | Val Acc: 0.9993 | Time: 48.17s
Epoch [7/35] | Train Acc: 0.9996 | Val Acc: 0.9985 | Time: 48.26s
Epoch [8/35] | Train Acc: 0.9900 | Val Acc: 0.9848 | Time: 48.08s
Epoch [9/35] | Train Acc: 0.9980 | Val Acc: 0.9985 | Time: 48.14s
Epoch [10/35] | Train Acc: 0.9996 | Val Acc: 0.9985 | Time: 48.11s
Epoch [11/35] | Train Acc: 0.9971 | Val Acc: 0.9891 | Time: 48.14s
Epoch [12/35] | Train Acc: 0.9993 | Val Acc: 0.9964 | Time: 48.16s
Epoch [13/35] | Train Acc: 1.0000 | Val Acc: 0.9978 | Time: 48.09s
Epoch [14/35] | Train Acc: 1.0000 | Val Acc: 0.9985 | Time: 48.07s
Early stopping triggered!
Fold 2 Accuracy: 0.9985

========== Fold 3/5 ==========
/usr/local/lib/python3.11/dist-packages/torch/optim/lr_scheduler.py:62: UserWarning: The verbose parameter is deprecated. Please use get_last_lr() to access the learning rate.
  warnings.warn(
```

```
Epoch [1/35] | Train Acc: 0.9295 | Val Acc: 0.9819 | Time: 48.39s
Epoch [2/35] | Train Acc: 0.9897 | Val Acc: 0.9920 | Time: 48.14s
Epoch [3/35] | Train Acc: 0.9895 | Val Acc: 0.9913 | Time: 48.26s
Epoch [4/35] | Train Acc: 0.9964 | Val Acc: 0.9811 | Time: 48.18s
Epoch [5/35] | Train Acc: 0.9960 | Val Acc: 0.9906 | Time: 48.19s
Epoch [6/35] | Train Acc: 0.9991 | Val Acc: 0.9942 | Time: 48.24s
Epoch [7/35] | Train Acc: 1.0000 | Val Acc: 0.9942 | Time: 48.28s
Epoch [8/35] | Train Acc: 0.9998 | Val Acc: 0.9935 | Time: 48.28s
Epoch [9/35] | Train Acc: 0.9956 | Val Acc: 0.9884 | Time: 48.14s
Epoch [10/35] | Train Acc: 0.9996 | Val Acc: 0.9949 | Time: 48.31s
Epoch [11/35] | Train Acc: 0.9998 | Val Acc: 0.9956 | Time: 48.20s
Epoch [12/35] | Train Acc: 0.9998 | Val Acc: 0.9942 | Time: 48.23s
Epoch [13/35] | Train Acc: 1.0000 | Val Acc: 0.9949 | Time: 48.29s
Epoch [14/35] | Train Acc: 1.0000 | Val Acc: 0.9949 | Time: 48.21s
Epoch [15/35] | Train Acc: 1.0000 | Val Acc: 0.9956 | Time: 48.21s
Epoch [16/35] | Train Acc: 0.9998 | Val Acc: 0.9964 | Time: 48.27s
Early stopping triggered!
Fold 3 Accuracy: 0.9956

========== Fold 4/5 ==========
/usr/local/lib/python3.11/dist-packages/torch/optim/lr_scheduler.py:62: UserWarning: The verbose parameter is de
precated. Please use get_last_lr() to access the learning rate.
  warnings.warn(
Epoch [1/35] | Train Acc: 0.9207 | Val Acc: 0.9884 | Time: 48.26s
Epoch [2/35] | Train Acc: 0.9884 | Val Acc: 0.9949 | Time: 48.35s
Epoch [3/35] | Train Acc: 0.9917 | Val Acc: 0.9906 | Time: 48.34s
Epoch [4/35] | Train Acc: 0.9931 | Val Acc: 0.9913 | Time: 48.23s
Epoch [5/35] | Train Acc: 0.9973 | Val Acc: 0.9978 | Time: 48.32s
Epoch [6/35] | Train Acc: 0.9967 | Val Acc: 0.9978 | Time: 48.18s
Epoch [7/35] | Train Acc: 0.9975 | Val Acc: 0.9949 | Time: 48.21s
Epoch [8/35] | Train Acc: 0.9971 | Val Acc: 0.9985 | Time: 48.30s
Epoch [9/35] | Train Acc: 0.9996 | Val Acc: 0.9993 | Time: 48.29s
Epoch [10/35] | Train Acc: 1.0000 | Val Acc: 0.9985 | Time: 48.13s
Epoch [11/35] | Train Acc: 1.0000 | Val Acc: 0.9993 | Time: 48.30s
Epoch [12/35] | Train Acc: 1.0000 | Val Acc: 0.9993 | Time: 48.32s
Epoch [13/35] | Train Acc: 1.0000 | Val Acc: 0.9993 | Time: 48.23s
Epoch [14/35] | Train Acc: 1.0000 | Val Acc: 0.9993 | Time: 48.44s
Early stopping triggered!
Fold 4 Accuracy: 0.9993

========== Fold 5/5 ==========
/usr/local/lib/python3.11/dist-packages/torch/optim/lr_scheduler.py:62: UserWarning: The verbose parameter is de
precated. Please use get_last_lr() to access the learning rate.
  warnings.warn(
Epoch [1/35] | Train Acc: 0.9236 | Val Acc: 0.9913 | Time: 48.20s
Epoch [2/35] | Train Acc: 0.9859 | Val Acc: 0.9898 | Time: 48.17s
Epoch [3/35] | Train Acc: 0.9859 | Val Acc: 0.9927 | Time: 48.12s
Epoch [4/35] | Train Acc: 0.9966 | Val Acc: 0.9964 | Time: 48.17s
Epoch [5/35] | Train Acc: 0.9998 | Val Acc: 0.9964 | Time: 48.13s
Epoch [6/35] | Train Acc: 0.9969 | Val Acc: 0.9753 | Time: 48.33s
Epoch [7/35] | Train Acc: 0.9906 | Val Acc: 0.9949 | Time: 48.09s
Epoch [8/35] | Train Acc: 0.9940 | Val Acc: 0.9935 | Time: 48.17s
Epoch [9/35] | Train Acc: 0.9987 | Val Acc: 0.9971 | Time: 48.21s
Epoch [10/35] | Train Acc: 0.9980 | Val Acc: 0.9971 | Time: 48.22s
Epoch [11/35] | Train Acc: 0.9996 | Val Acc: 0.9978 | Time: 48.23s
Epoch [12/35] | Train Acc: 1.0000 | Val Acc: 0.9985 | Time: 48.27s
Epoch [13/35] | Train Acc: 1.0000 | Val Acc: 0.9985 | Time: 48.21s
Epoch [14/35] | Train Acc: 1.0000 | Val Acc: 0.9985 | Time: 48.20s
Epoch [15/35] | Train Acc: 1.0000 | Val Acc: 0.9978 | Time: 48.14s
Epoch [16/35] | Train Acc: 1.0000 | Val Acc: 0.9985 | Time: 48.18s
Epoch [17/35] | Train Acc: 1.0000 | Val Acc: 0.9985 | Time: 48.24s
Epoch [18/35] | Train Acc: 1.0000 | Val Acc: 0.9978 | Time: 48.21s
Epoch [19/35] | Train Acc: 1.0000 | Val Acc: 0.9985 | Time: 48.28s
Early stopping triggered!
Fold 5 Accuracy: 0.9985

====================== 5-FOLD CROSS VALIDATION RESULTS ======================
Fold 1: 0.9971
Fold 2: 0.9985
Fold 3: 0.9956
Fold 4: 0.9993
Fold 5: 0.9985

Mean Accuracy: 0.9978
Standard Deviation: 0.0013
```