rice-disease-resnet18

May 7, 2024

[30]:
```python
import os
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, models, transforms
from torch.utils.data import DataLoader
```

[31]:
```python
# Define data directory paths
data_dir = "rice_diseases_last_filter"
train_dir = os.path.join(data_dir, "train")
val_dir = os.path.join(data_dir, "val")
```

[32]:
```python
import numpy as np
import random
from sklearn.manifold import TSNE
from sklearn.decomposition import PCA
import seaborn as sns
```

[33]:
```python
# Data Visualization

# 1. Sample Image Exploration
def visualize_sample_images(dataset, num_samples=5):
    classes = dataset.classes
    class_to_idx = dataset.class_to_idx
    idx_to_class = {v: k for k, v in class_to_idx.items()}

    sample_images = []
    sample_labels = []
    for _ in range(num_samples):
        class_idx = random.randint(0, len(classes)-1)
        class_name = idx_to_class[class_idx]
        img_path = random.choice(dataset.samples)[0]
        sample_images.append(img_path)
        sample_labels.append(class_name)

    # Plot sample images
    plt.figure(figsize=(15, 5))
```

```
    for i in range(num_samples):
        plt.subplot(1, num_samples, i+1)
        img = plt.imread(sample_images[i])
        plt.imshow(img)
        plt.title(sample_labels[i])
        plt.axis('off')
    plt.show()

# 2. Class Distribution
def plot_class_distribution(dataset):
    class_counts = np.zeros(len(dataset.classes))
    for _, label in dataset.samples:
        class_counts[label] += 1

    plt.figure(figsize=(10, 5))
    sns.barplot(x=dataset.classes, y=class_counts)
    plt.title('Class Distribution')
    plt.xlabel('Class')
    plt.ylabel('Frequency')
    plt.xticks(rotation=45)
    plt.show()
```

[34]:
```
# 3. Image Augmentation (visualize transformed images)
def visualize_augmented_images(dataset, num_samples=3):
    transform = data_transforms['train']
    sample_images = []
    for _ in range(num_samples):
        img_path = random.choice(dataset.samples)[0]
        img = plt.imread(img_path)
        transformed_img = transform(img)
        sample_images.append(transformed_img)

    # Plot augmented images
    plt.figure(figsize=(15, 5))
    for i in range(num_samples):
        plt.subplot(1, num_samples, i+1)
        plt.imshow(sample_images[i].permute(1, 2, 0))
        plt.axis('off')
    plt.show()

# 4. Feature Visualization
def visualize_features(model, dataloaders):
    features = []
    labels = []
    model.eval()
    with torch.no_grad():
        for inputs, labels_batch in dataloaders['val']:
```

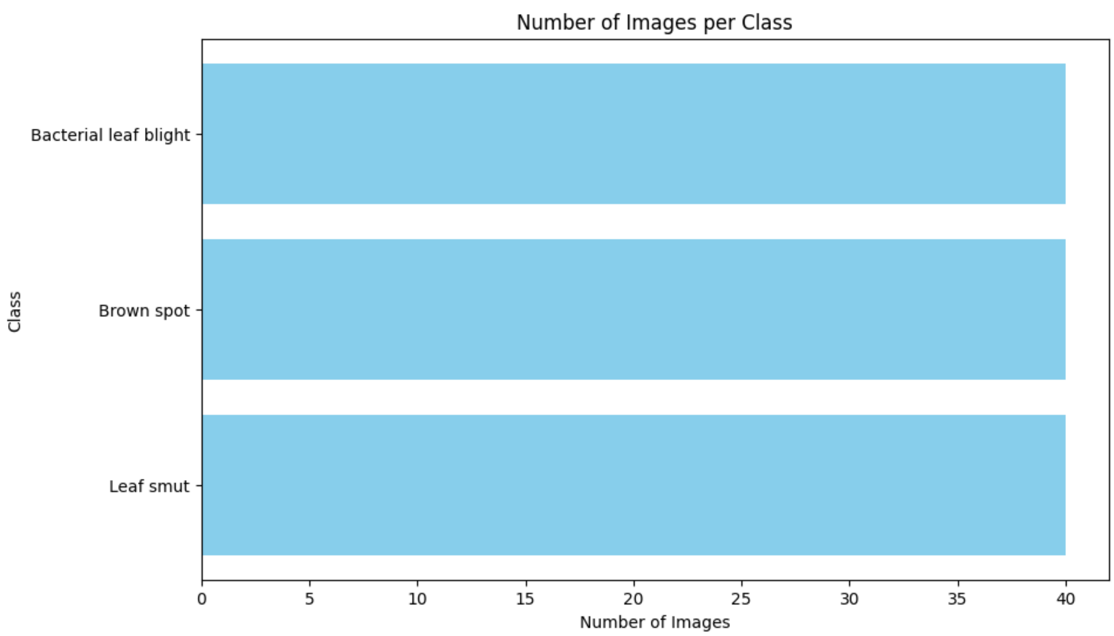
```
            inputs = inputs.to(device)
            labels.extend(labels_batch.numpy())
            outputs = model(inputs)
            features.extend(outputs.cpu().numpy())

    # Reduce dimensionality for visualization
    tsne = TSNE(n_components=2, random_state=42)
    pca = PCA(n_components=2, random_state=42)
    reduced_features_tsne = tsne.fit_transform(features)
    reduced_features_pca = pca.fit_transform(features)

    # Plot features using t-SNE and PCA
    plt.figure(figsize=(12, 5))
    plt.subplot(1, 2, 1)
    sns.scatterplot(x=reduced_features_tsne[:, 0], y=reduced_features_tsne[:,␣
↪1], hue=labels)
    plt.title('t-SNE Visualization')
    plt.subplot(1, 2, 2)
    sns.scatterplot(x=reduced_features_pca[:, 0], y=reduced_features_pca[:, 1],␣
↪hue=labels)
    plt.title('PCA Visualization')
    plt.show()
```

[35]:
```
import os
import matplotlib.pyplot as plt

def count_photos_in_folders(data_dir):
    folder_counts = {}
    folders = os.listdir(data_dir)
    for folder_name in folders:
        folder_path = os.path.join(data_dir, folder_name)
        if os.path.isdir(folder_path):
            num_photos = sum([len(files) for _, _, files in os.
↪walk(folder_path)])
            folder_counts[folder_name] = num_photos
    return folder_counts

def print_photo_counts(data_dir):
    print("Photo counts for", os.path.basename(data_dir))
    folder_counts = count_photos_in_folders(data_dir)
    for folder, count in folder_counts.items():
        print(f"Folder '{folder}': {count} photos")
    return folder_counts

def plot_image_counts(class_counts):
    classes = list(class_counts.keys())
    counts = list(class_counts.values())
```

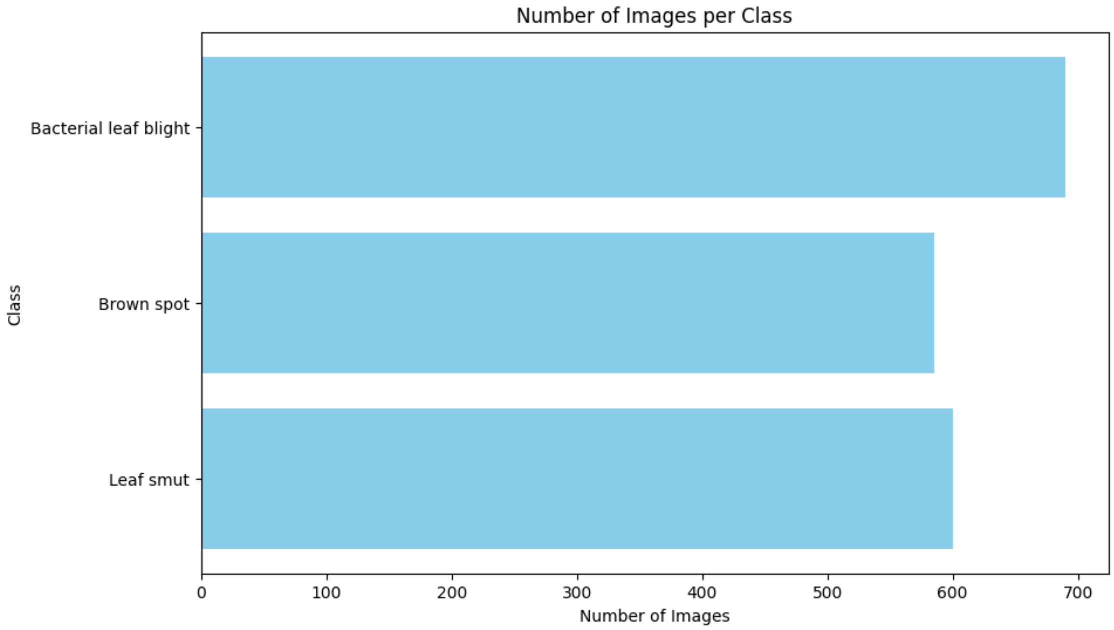
```
    plt.figure(figsize=(10, 6))
    plt.barh(classes, counts, color='skyblue')
    plt.xlabel('Number of Images')
    plt.ylabel('Class')
    plt.title('Number of Images per Class')
    plt.gca().invert_yaxis()  # Invert y-axis to have class names displayed␣
↪from top to bottom
    plt.show()

def count_original_photos(data_dir):
    folder_counts = {}
    diseases = os.listdir(data_dir)
    for disease in diseases:
        disease_path = os.path.join(data_dir, disease)
        if os.path.isdir(disease_path):
            num_photos = sum([len(files) for _, _, files in os.
↪walk(disease_path)])
            folder_counts[disease] = num_photos
    return folder_counts

def count_augmented_photos(train_dir, val_dir):
    train_counts = count_photos_in_folders(train_dir)
    val_counts = count_photos_in_folders(val_dir)
    augmented_counts = {}
    for folder in train_counts:
        augmented_counts[folder] = train_counts[folder] + val_counts.
↪get(folder, 0)
    for folder in val_counts:
        if folder not in augmented_counts:
            augmented_counts[folder] = val_counts[folder]
    return augmented_counts

data_dir = "rice_diseases_last_filter"
train_dir = os.path.join(data_dir, "train")
val_dir = os.path.join(data_dir, "val")

# Count original images per disease
original_counts = count_original_photos(os.path.join("rice_leaf_diseases"))
print("Original Images (Before Augmentation):")
print(original_counts)
plot_image_counts(original_counts)

print("\nOriginal Images (After Augmentation):")
# Count augmented images per disease
augmented_counts = count_augmented_photos(train_dir, val_dir)
print(augmented_counts)
```

```python
plot_image_counts(augmented_counts)

print("\nNumber of Photos in Train and Val:")
# Count photos in the "train" directory
train_counts = print_photo_counts(train_dir)
# Count photos in the "val" directory
val_counts = print_photo_counts(val_dir)

print("\nTotal Number of Photos After Augmentation:")
total_augmented_count = {folder: train_counts[folder] + val_counts.get(folder,
    0) for folder in train_counts}
print(total_augmented_count)
```

```
Original Images (Before Augmentation):
{'Bacterial leaf blight': 40, 'Brown spot': 40, 'Leaf smut': 40}
```




```
Original Images (After Augmentation):
{'Bacterial leaf blight': 690, 'Brown spot': 585, 'Leaf smut': 600}
```

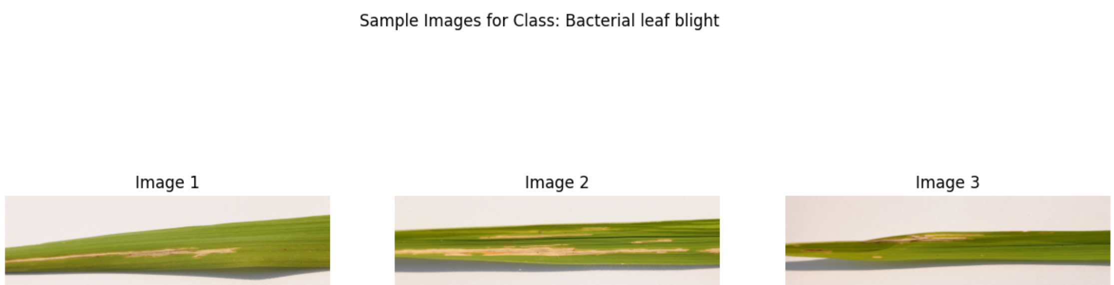


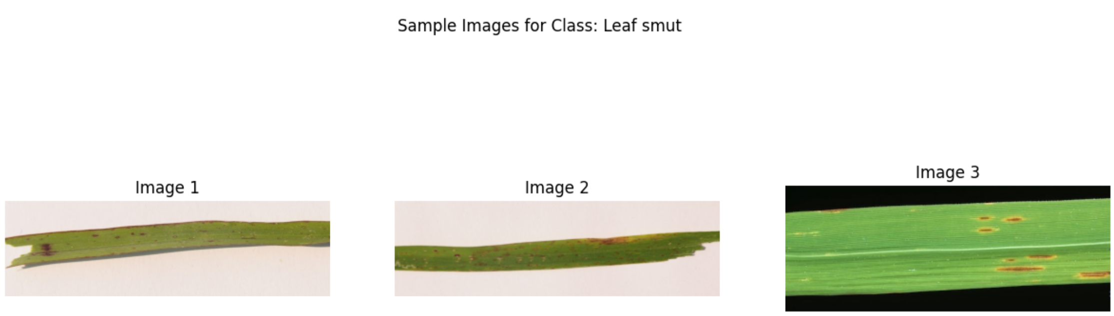

```
Number of Photos in Train and Val:
Photo counts for train
Folder 'Bacterial leaf blight': 552 photos
Folder 'Brown spot': 468 photos
Folder 'Leaf smut': 480 photos
Photo counts for val
Folder 'Bacterial leaf blight': 138 photos
Folder 'Brown spot': 117 photos
Folder 'Leaf smut': 120 photos

Total Number of Photos After Augmentation:
{'Bacterial leaf blight': 690, 'Brown spot': 585, 'Leaf smut': 600}
```

[36]:

```
def visualize_sample_images(data_dir, num_samples_per_class=3):
    class_folders = os.listdir(data_dir)
    for class_folder in class_folders:
        class_path = os.path.join(data_dir, class_folder)
        if os.path.isdir(class_path):
            image_files = os.listdir(class_path)
            sample_images = random.sample(image_files,␣
↪min(num_samples_per_class, len(image_files)))

            # Plot sample images for the current class
            plt.figure(figsize=(15, 5))
            plt.suptitle(f'Sample Images for Class: {class_folder}')
```

```
            for i, image_file in enumerate(sample_images, 1):
                img_path = os.path.join(class_path, image_file)
                img = plt.imread(img_path)
                plt.subplot(1, num_samples_per_class, i)
                plt.imshow(img)
                plt.title(f'Image {i}')
                plt.axis('off')
            plt.show()

# Directory containing the images
data_dir = "rice_leaf_diseases"

# Visualize 3 sample images for each class
visualize_sample_images(data_dir, num_samples_per_class=3)
```





[59]:
```
import os
import matplotlib.pyplot as plt
from PIL import Image

# Define the paths to the directories
rice_leaf_diseases_dir = "rice_leaf_diseases"
last_filter_train_dir = os.path.join("rice_diseases_last_filter", "train")

# Define the class to display
class_name_to_display = "Bacterial leaf blight"

# Display the first image from the folder in rice_leaf_diseases_dir
print("First image from the folder in rice_leaf_diseases:")
class_dir = os.path.join(rice_leaf_diseases_dir, class_name_to_display)
image_files = os.listdir(class_dir)
if image_files:
    first_image_path = os.path.join(class_dir, image_files[0])
    print(f"Class: {class_name_to_display}")
    image = Image.open(first_image_path)
    plt.imshow(image)
    plt.title(class_name_to_display)
    plt.axis('off')
    plt.show()

# Display only 6 images from the folder in last_filter_train_dir for "Bacterial␣
↪leaf blight" class
print("\nImages from the folder in last_filter_train directory:")
class_dir = os.path.join(last_filter_train_dir, class_name_to_display)
image_files = os.listdir(class_dir)
num_images_to_display = min(6, len(image_files))  # Limit to 6 images or less␣
↪if there are fewer images
for i in range(num_images_to_display):
    image_path = os.path.join(class_dir, image_files[i])
    print(f"Image {i+1}: {image_path}")
    # Display the image
    image = Image.open(image_path)
    plt.imshow(image)
    plt.title(f"{class_name_to_display} - Image {i+1}")
    plt.axis('off')
    plt.show()

import os
import torch
from torchvision import transforms
from PIL import Image
```

```python
import matplotlib.pyplot as plt

# Data Preprocessing

# Define data transforms
data_transforms = {
    'train': transforms.Compose([
        transforms.RandomResizedCrop(224),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ]),
    'val': transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ]),
}

# Define the path to the rice leaf diseases directory
rice_leaf_diseases_dir = "rice_leaf_diseases"

# Get the list of classes (folder names)
classes = os.listdir(rice_leaf_diseases_dir)

# Exclude bacterial_leaf_blight class
classes.remove("Bacterial leaf blight")

# Determine the number of classes
num_classes = len(classes)

## Display augmented images for each class
num_augmentations = 6 # Display 20 augmented images for each class
plt.figure(figsize=(15, 5*num_classes))
for i, class_name in enumerate(classes, start=1):
    # Get the path to the first image in the class folder
    class_dir = os.path.join(rice_leaf_diseases_dir, class_name)
    image_path = os.path.join(class_dir, os.listdir(class_dir)[0])

    # Load the example image
    image = Image.open(image_path)

    # Display the original image
    plt.subplot(num_classes, num_augmentations + 1, (i-1)*(num_augmentations + 1) + 1)
    plt.title(f'Original - {class_name}')
```

```
    plt.imshow(image)
    plt.axis('off')

    # Apply data augmentation transformations
    for j in range(num_augmentations):
        augmented_image = data_transforms['train'](image)

        # Display the augmented image
        plt.subplot(num_classes, num_augmentations + 1,␣
 ↪(i-1)*(num_augmentations + 1) + j + 2)
        plt.title(f'Augmented {j+1}')
        plt.imshow(augmented_image.permute(1, 2, 0))  # Convert from tensor to␣
 ↪image format
        plt.axis('off')

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

```
First image from the folder in rice_leaf_diseases:
Class: Bacterial leaf blight
```


```
Images from the folder in last_filter_train directory:
Image 1: rice_diseases_last_filter\train\Bacterial leaf blight\Bacterial leaf
blight__0_1005.jpeg
```

Bacterial leaf blight - Image 1

Image 2: rice_diseases_last_filter\train\Bacterial leaf blight\Bacterial leaf blight__0_1045.jpeg

Bacterial leaf blight - Image 2

Image 3: rice_diseases_last_filter\train\Bacterial leaf blight\Bacterial leaf blight__0_1048.jpeg

Bacterial leaf blight - Image 3

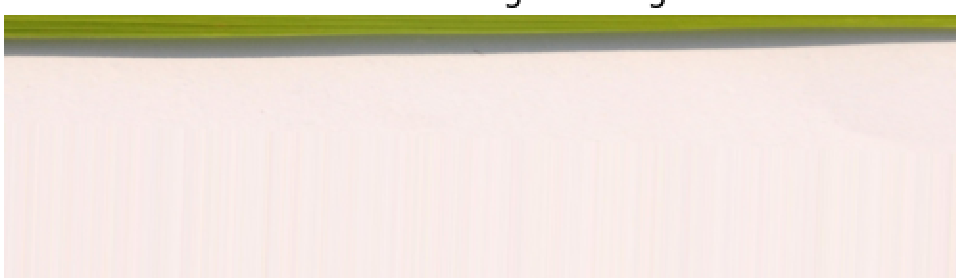


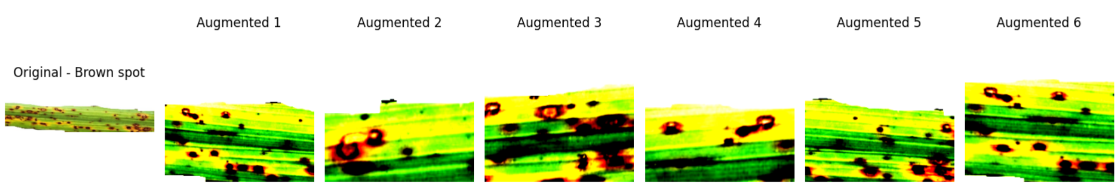
Image 4: rice_diseases_last_filter\train\Bacterial leaf blight\Bacterial leaf blight__0_1050.jpeg

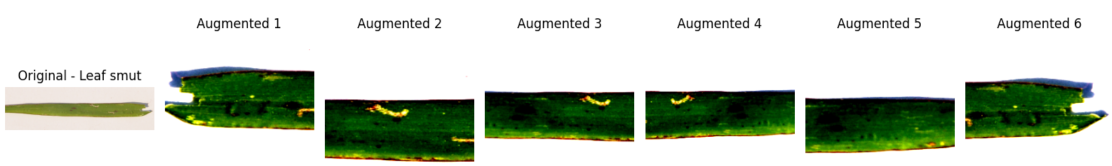
Image 5: rice_diseases_last_filter\train\Bacterial leaf blight\Bacterial leaf blight__0_1058.jpeg

Image 6: rice_diseases_last_filter\train\Bacterial leaf blight\Bacterial leaf blight__0_1070.jpeg


```
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
```

```
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
```





[ ]:
```python
# Data Preprocessing

# 1. Image Resizing (already implemented in data transforms)
# 2. Normalization (already implemented in data transforms)
# 3. Data Augmentation (already implemented in data transforms)

# Define data transforms
data_transforms = {
    'train': transforms.Compose([
```

```
        transforms.RandomResizedCrop(224),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ]),
    'val': transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ]),
}
```

[ ]:
```
# Create datasets
image_datasets = {x: datasets.ImageFolder(os.path.join(data_dir, x),␣
 ↪data_transforms[x]) for x in ['train', 'val']}

# Create data loaders
dataloaders = {x: DataLoader(image_datasets[x], batch_size=32, shuffle=True,␣
 ↪num_workers=4) for x in ['train', 'val']}
```

[ ]:
```
#Define the model architecture
model = models.resnet18(pretrained=True)
num_ftrs = model.fc.in_features
model.fc = nn.Linear(num_ftrs, len(image_datasets['train'].classes))
```

```
c:\Users\LENOVO\AppData\Local\Programs\Python\Python312\Lib\site-
packages\torchvision\models\_utils.py:208: UserWarning: The parameter
'pretrained' is deprecated since 0.13 and may be removed in the future, please
use 'weights' instead.
  warnings.warn(
c:\Users\LENOVO\AppData\Local\Programs\Python\Python312\Lib\site-
packages\torchvision\models\_utils.py:223: UserWarning: Arguments other than a
weight enum or `None` for 'weights' are deprecated since 0.13 and may be removed
in the future. The current behavior is equivalent to passing
`weights=ResNet18_Weights.IMAGENET1K_V1`. You can also use
`weights=ResNet18_Weights.DEFAULT` to get the most up-to-date weights.
  warnings.warn(msg)
```

[ ]:
```
# Define loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)
```

[ ]:
```
#Define device
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
```

[ ]:

```python
# Train the model
def train_model(model, criterion, optimizer, num_epochs=50):
    for epoch in range(num_epochs):
        print(f'Epoch {epoch+1}/{num_epochs}')
        print('-' * 10)

        # Each epoch has a training and validation phase
        for phase in ['train', 'val']:
            if phase == 'train':
                model.train()  # Set model to training mode
            else:
                model.eval()   # Set model to evaluate mode

            running_loss = 0.0
            running_corrects = 0

            # Iterate over data.
            for inputs, labels in dataloaders[phase]:
                inputs = inputs.to(device)
                labels = labels.to(device)

                # Zero the parameter gradients
                optimizer.zero_grad()

                # Forward pass
                with torch.set_grad_enabled(phase == 'train'):
                    outputs = model(inputs)
                    _, preds = torch.max(outputs, 1)
                    loss = criterion(outputs, labels)

                    # Backward + optimize only if in training phase
                    if phase == 'train':
                        loss.backward()
                        optimizer.step()

                # Statistics
                running_loss += loss.item() * inputs.size(0)
                running_corrects += torch.sum(preds == labels.data)

            epoch_loss = running_loss / len(image_datasets[phase])
            epoch_acc = running_corrects.double() / len(image_datasets[phase])

            print(f'{phase} Loss: {epoch_loss:.4f} Acc: {epoch_acc:.4f}')
```

[ ]:

```python
#Define device
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
# Train the model
```

```python
train_model(model, criterion, optimizer, num_epochs=50)
```

```
Epoch 1/50
----------
train Loss: 0.8224 Acc: 0.6160
val Loss: 0.5420 Acc: 0.7760
Epoch 2/50
----------
train Loss: 0.4892 Acc: 0.8020
val Loss: 0.3861 Acc: 0.8560
Epoch 3/50
----------
train Loss: 0.4770 Acc: 0.7933
val Loss: 0.3611 Acc: 0.8640
Epoch 4/50
----------
train Loss: 0.4391 Acc: 0.8040
val Loss: 0.2814 Acc: 0.8987
Epoch 5/50
----------
train Loss: 0.4026 Acc: 0.8127
val Loss: 0.3072 Acc: 0.8533
Epoch 6/50
----------
train Loss: 0.3576 Acc: 0.8400
val Loss: 0.2201 Acc: 0.9013
Epoch 7/50
----------
train Loss: 0.3403 Acc: 0.8580
val Loss: 0.1956 Acc: 0.9360
Epoch 8/50
----------
train Loss: 0.3737 Acc: 0.8380
val Loss: 0.1891 Acc: 0.9253
Epoch 9/50
----------
train Loss: 0.3198 Acc: 0.8600
val Loss: 0.2057 Acc: 0.9147
Epoch 10/50
----------
train Loss: 0.3341 Acc: 0.8480
val Loss: 0.1669 Acc: 0.9360
Epoch 11/50
----------
train Loss: 0.2972 Acc: 0.8753
val Loss: 0.1739 Acc: 0.9387
Epoch 12/50
```

```
----------
train Loss: 0.2951 Acc: 0.8740
val Loss: 0.1902 Acc: 0.9253
Epoch 13/50
----------
train Loss: 0.3049 Acc: 0.8687
val Loss: 0.2012 Acc: 0.9147
Epoch 14/50
----------
train Loss: 0.2869 Acc: 0.8707
val Loss: 0.1504 Acc: 0.9413
Epoch 15/50
----------
train Loss: 0.2909 Acc: 0.8707
val Loss: 0.1459 Acc: 0.9440
Epoch 16/50
----------
train Loss: 0.2783 Acc: 0.8727
val Loss: 0.1594 Acc: 0.9493
Epoch 17/50
----------
train Loss: 0.2594 Acc: 0.8847
val Loss: 0.1642 Acc: 0.9413
Epoch 18/50
----------
train Loss: 0.2880 Acc: 0.8740
val Loss: 0.1501 Acc: 0.9387
Epoch 19/50
----------
train Loss: 0.2702 Acc: 0.8747
val Loss: 0.1595 Acc: 0.9387
Epoch 20/50
----------
train Loss: 0.2655 Acc: 0.8793
val Loss: 0.1518 Acc: 0.9387
Epoch 21/50
----------
train Loss: 0.2678 Acc: 0.8853
val Loss: 0.1376 Acc: 0.9547
Epoch 22/50
----------
train Loss: 0.2543 Acc: 0.8920
val Loss: 0.1369 Acc: 0.9387
Epoch 23/50
----------
train Loss: 0.2490 Acc: 0.8907
val Loss: 0.1802 Acc: 0.9173
Epoch 24/50
```

```
----------
train Loss: 0.2490 Acc: 0.8893
val Loss: 0.1598 Acc: 0.9360
Epoch 25/50
----------
train Loss: 0.2564 Acc: 0.8900
val Loss: 0.1498 Acc: 0.9493
Epoch 26/50
----------
train Loss: 0.2692 Acc: 0.8793
val Loss: 0.1324 Acc: 0.9493
Epoch 27/50
----------
train Loss: 0.2504 Acc: 0.8860
val Loss: 0.1718 Acc: 0.9360
Epoch 28/50
----------
train Loss: 0.2406 Acc: 0.8920
val Loss: 0.1271 Acc: 0.9440
Epoch 29/50
----------
train Loss: 0.2351 Acc: 0.9040
val Loss: 0.1411 Acc: 0.9413
Epoch 30/50
----------
train Loss: 0.2320 Acc: 0.8913
val Loss: 0.1475 Acc: 0.9467
Epoch 31/50
----------
train Loss: 0.2175 Acc: 0.9007
val Loss: 0.1513 Acc: 0.9440
Epoch 32/50
----------
train Loss: 0.2438 Acc: 0.8900
val Loss: 0.1912 Acc: 0.9253
Epoch 33/50
----------
train Loss: 0.2126 Acc: 0.9040
val Loss: 0.2224 Acc: 0.8853
Epoch 34/50
----------
train Loss: 0.2309 Acc: 0.8973
val Loss: 0.1732 Acc: 0.9280
Epoch 35/50
----------
train Loss: 0.2093 Acc: 0.9013
val Loss: 0.1177 Acc: 0.9520
Epoch 36/50
```

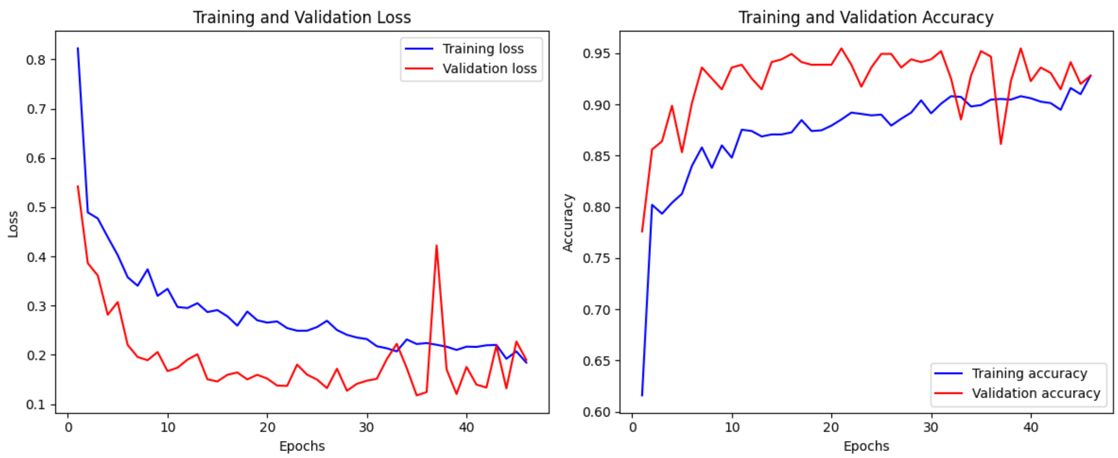
```
----------
train Loss: 0.2133 Acc: 0.9080
val Loss: 0.1243 Acc: 0.9467
Epoch 37/50
----------
train Loss: 0.2070 Acc: 0.9073
val Loss: 0.4222 Acc: 0.8613
Epoch 38/50
----------
train Loss: 0.2313 Acc: 0.8980
val Loss: 0.1705 Acc: 0.9227
Epoch 39/50
----------
train Loss: 0.2220 Acc: 0.8993
val Loss: 0.1204 Acc: 0.9547
Epoch 40/50
----------
train Loss: 0.2239 Acc: 0.9047
val Loss: 0.1754 Acc: 0.9227
Epoch 41/50
----------
train Loss: 0.2204 Acc: 0.9053
val Loss: 0.1395 Acc: 0.9360
Epoch 42/50
----------
train Loss: 0.2165 Acc: 0.9047
val Loss: 0.1336 Acc: 0.9307
Epoch 43/50
----------
train Loss: 0.2100 Acc: 0.9080
val Loss: 0.2188 Acc: 0.9147
Epoch 44/50
----------
train Loss: 0.2166 Acc: 0.9060
val Loss: 0.1318 Acc: 0.9413
Epoch 45/50
----------
train Loss: 0.2161 Acc: 0.9027
val Loss: 0.2271 Acc: 0.9200
Epoch 46/50
----------
train Loss: 0.2194 Acc: 0.9013
val Loss: 0.1898 Acc: 0.9280
Epoch 47/50
----------
train Loss: 0.2201 Acc: 0.8947
val Loss: 0.1574 Acc: 0.9253
Epoch 48/50
```

```
----------
train Loss: 0.1920 Acc: 0.9160
val Loss: 0.1612 Acc: 0.9467
Epoch 49/50
----------
train Loss: 0.2070 Acc: 0.9100
val Loss: 0.2483 Acc: 0.9227
Epoch 50/50
----------
train Loss: 0.1842 Acc: 0.9280
val Loss: 0.1191 Acc: 0.9387
```

[ ]:
```
import pickle
# Save the entire model object
with open('rice_disease_model_final.pkl', 'wb') as f:
    pickle.dump(model, f)
```

[ ]:
```
import matplotlib.pyplot as plt

# Training and validation loss and accuracy values
train_loss = [0.8224, 0.4892, 0.4770, 0.4391, 0.4026, 0.3576, 0.3403, 0.3737, 0.
 ↪3198, 0.3341,
              0.2972, 0.2951, 0.3049, 0.2869, 0.2909, 0.2783, 0.2594, 0.2880, 0.
 ↪2702, 0.2655,
              0.2678, 0.2543, 0.2490, 0.2490, 0.2564, 0.2692, 0.2504, 0.2406, 0.
 ↪2351, 0.2320,
              0.2175, 0.2133, 0.2070, 0.2313, 0.2220, 0.2239, 0.2204, 0.2165, 0.
 ↪2100, 0.2166,
              0.2161, 0.2194, 0.2201, 0.1920, 0.2070, 0.1842]

val_loss = [0.5420, 0.3861, 0.3611, 0.2814, 0.3072, 0.2201, 0.1956, 0.1891, 0.
 ↪2057, 0.1669,
            0.1739, 0.1902, 0.2012, 0.1504, 0.1459, 0.1594, 0.1642, 0.1501, 0.
 ↪1595, 0.1518,
            0.1376, 0.1369, 0.1802, 0.1598, 0.1498, 0.1324, 0.1718, 0.1271, 0.
 ↪1411, 0.1475,
            0.1513, 0.1912, 0.2224, 0.1732, 0.1177, 0.1243, 0.4222, 0.1705, 0.
 ↪1204, 0.1754,
            0.1395, 0.1336, 0.2188, 0.1318, 0.2271, 0.1898, 0.1574, 0.1612, 0.
 ↪2483, 0.1191]

train_acc = [0.6160, 0.8020, 0.7933, 0.8040, 0.8127, 0.8400, 0.8580, 0.8380, 0.
 ↪8600, 0.8480,
             0.8753, 0.8740, 0.8687, 0.8707, 0.8707, 0.8727, 0.8847, 0.8740, 0.
 ↪8747, 0.8793,
```

```
            0.8853, 0.8920, 0.8907, 0.8893, 0.8900, 0.8793, 0.8860, 0.8920, 0.
↪9040, 0.8913,
            0.9007, 0.9080, 0.9073, 0.8980, 0.8993, 0.9047, 0.9053, 0.9047, 0.
↪9080, 0.9060,
            0.9027, 0.9013, 0.8947, 0.9160, 0.9100, 0.9280, 0.9160, 0.9100, 0.
↪9280]

val_acc = [0.7760, 0.8560, 0.8640, 0.8987, 0.8533, 0.9013, 0.9360, 0.9253, 0.
↪9147, 0.9360,
          0.9387, 0.9253, 0.9147, 0.9413, 0.9440, 0.9493, 0.9413, 0.9387, 0.
↪9387, 0.9387,
          0.9547, 0.9387, 0.9173, 0.9360, 0.9493, 0.9493, 0.9360, 0.9440, 0.
↪9413, 0.9440,
          0.9520, 0.9253, 0.8853, 0.9280, 0.9520, 0.9467, 0.8613, 0.9227, 0.
↪9547, 0.9227,
          0.9360, 0.9307, 0.9147, 0.9413, 0.9200, 0.9280, 0.9253, 0.9467, 0.
↪9227, 0.9387]

epochs = range(1, len(train_loss) + 1)

# Plotting training and validation loss
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(epochs, train_loss[:len(epochs)], 'b', label='Training loss')
plt.plot(epochs, val_loss[:len(epochs)], 'r', label='Validation loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

# Plotting training and validation accuracy
plt.subplot(1, 2, 2)
plt.plot(epochs, train_acc[:len(epochs)], 'b', label='Training accuracy')
plt.plot(epochs, val_acc[:len(epochs)], 'r', label='Validation accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

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



[ ]:

[ ]:
```
# Define data transforms
data_transforms = {
    'train': transforms.Compose([
        transforms.RandomResizedCrop(224),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ]),
    'val': transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ]),
}
```

[ ]:
```
# Create datasets
image_datasets = {x: datasets.ImageFolder(os.path.join(data_dir, x),␣
 ↪data_transforms[x]) for x in ['train', 'val']}

# Create data loaders
dataloaders = {x: DataLoader(image_datasets[x], batch_size=32, shuffle=True,␣
 ↪num_workers=4) for x in ['train', 'val']}
```

[ ]:
```
#Define the model architecture
model = models.resnet18(pretrained=True)
num_ftrs = model.fc.in_features
model.fc = nn.Linear(num_ftrs, len(image_datasets['train'].classes))
```

```
c:\Users\LENOVO\AppData\Local\Programs\Python\Python312\Lib\site-
packages\torchvision\models\_utils.py:208: UserWarning: The parameter
'pretrained' is deprecated since 0.13 and may be removed in the future, please
use 'weights' instead.
  warnings.warn(
c:\Users\LENOVO\AppData\Local\Programs\Python\Python312\Lib\site-
packages\torchvision\models\_utils.py:223: UserWarning: Arguments other than a
weight enum or `None` for 'weights' are deprecated since 0.13 and may be removed
in the future. The current behavior is equivalent to passing
`weights=ResNet18_Weights.IMAGENET1K_V1`. You can also use
`weights=ResNet18_Weights.DEFAULT` to get the most up-to-date weights.
  warnings.warn(msg)
```

[ ]:
```
# Define loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)
```

[ ]:
```
#Define device
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
```

[ ]:
```
# Train the model
def train_model(model, criterion, optimizer, num_epochs=50):
    for epoch in range(num_epochs):
        print(f'Epoch {epoch+1}/{num_epochs}')
        print('-' * 10)

        # Each epoch has a training and validation phase
        for phase in ['train', 'val']:
            if phase == 'train':
                model.train()  # Set model to training mode
            else:
                model.eval()   # Set model to evaluate mode

            running_loss = 0.0
            running_corrects = 0

            # Iterate over data.
            for inputs, labels in dataloaders[phase]:
                inputs = inputs.to(device)
                labels = labels.to(device)

                # Zero the parameter gradients
                optimizer.zero_grad()

                # Forward pass
                with torch.set_grad_enabled(phase == 'train'):
                    outputs = model(inputs)
```

```
                    _, preds = torch.max(outputs, 1)
                    loss = criterion(outputs, labels)

                    # Backward + optimize only if in training phase
                    if phase == 'train':
                        loss.backward()
                        optimizer.step()

                # Statistics
                running_loss += loss.item() * inputs.size(0)
                running_corrects += torch.sum(preds == labels.data)

            epoch_loss = running_loss / len(image_datasets[phase])
            epoch_acc = running_corrects.double() / len(image_datasets[phase])

            print(f'{phase} Loss: {epoch_loss:.4f} Acc: {epoch_acc:.4f}')
```

[ ]:
```
#Define device
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
# Train the model
train_model(model, criterion, optimizer, num_epochs=50)
```

```
Epoch 1/50
----------
train Loss: 0.8224 Acc: 0.6160
val Loss: 0.5420 Acc: 0.7760
Epoch 2/50
----------
train Loss: 0.4892 Acc: 0.8020
val Loss: 0.3861 Acc: 0.8560
Epoch 3/50
----------
train Loss: 0.4770 Acc: 0.7933
val Loss: 0.3611 Acc: 0.8640
Epoch 4/50
----------
train Loss: 0.4391 Acc: 0.8040
val Loss: 0.2814 Acc: 0.8987
Epoch 5/50
----------
train Loss: 0.4026 Acc: 0.8127
val Loss: 0.3072 Acc: 0.8533
Epoch 6/50
----------
train Loss: 0.3576 Acc: 0.8400
val Loss: 0.2201 Acc: 0.9013
Epoch 7/50
```

```
----------
train Loss: 0.3403 Acc: 0.8580
val Loss: 0.1956 Acc: 0.9360
Epoch 8/50
----------
train Loss: 0.3737 Acc: 0.8380
val Loss: 0.1891 Acc: 0.9253
Epoch 9/50
----------
train Loss: 0.3198 Acc: 0.8600
val Loss: 0.2057 Acc: 0.9147
Epoch 10/50
----------
train Loss: 0.3341 Acc: 0.8480
val Loss: 0.1669 Acc: 0.9360
Epoch 11/50
----------
train Loss: 0.2972 Acc: 0.8753
val Loss: 0.1739 Acc: 0.9387
Epoch 12/50
----------
train Loss: 0.2951 Acc: 0.8740
val Loss: 0.1902 Acc: 0.9253
Epoch 13/50
----------
train Loss: 0.3049 Acc: 0.8687
val Loss: 0.2012 Acc: 0.9147
Epoch 14/50
----------
train Loss: 0.2869 Acc: 0.8707
val Loss: 0.1504 Acc: 0.9413
Epoch 15/50
----------
train Loss: 0.2909 Acc: 0.8707
val Loss: 0.1459 Acc: 0.9440
Epoch 16/50
----------
train Loss: 0.2783 Acc: 0.8727
val Loss: 0.1594 Acc: 0.9493
Epoch 17/50
----------
train Loss: 0.2594 Acc: 0.8847
val Loss: 0.1642 Acc: 0.9413
Epoch 18/50
----------
train Loss: 0.2880 Acc: 0.8740
val Loss: 0.1501 Acc: 0.9387
Epoch 19/50
```

```
----------
train Loss: 0.2702 Acc: 0.8747
val Loss: 0.1595 Acc: 0.9387
Epoch 20/50
----------
train Loss: 0.2655 Acc: 0.8793
val Loss: 0.1518 Acc: 0.9387
Epoch 21/50
----------
train Loss: 0.2678 Acc: 0.8853
val Loss: 0.1376 Acc: 0.9547
Epoch 22/50
----------
train Loss: 0.2543 Acc: 0.8920
val Loss: 0.1369 Acc: 0.9387
Epoch 23/50
----------
train Loss: 0.2490 Acc: 0.8907
val Loss: 0.1802 Acc: 0.9173
Epoch 24/50
----------
train Loss: 0.2490 Acc: 0.8893
val Loss: 0.1598 Acc: 0.9360
Epoch 25/50
----------
train Loss: 0.2564 Acc: 0.8900
val Loss: 0.1498 Acc: 0.9493
Epoch 26/50
----------
train Loss: 0.2692 Acc: 0.8793
val Loss: 0.1324 Acc: 0.9493
Epoch 27/50
----------
train Loss: 0.2504 Acc: 0.8860
val Loss: 0.1718 Acc: 0.9360
Epoch 28/50
----------
train Loss: 0.2406 Acc: 0.8920
val Loss: 0.1271 Acc: 0.9440
Epoch 29/50
----------
train Loss: 0.2351 Acc: 0.9040
val Loss: 0.1411 Acc: 0.9413
Epoch 30/50
----------
train Loss: 0.2320 Acc: 0.8913
val Loss: 0.1475 Acc: 0.9467
Epoch 31/50
```

```
----------
train Loss: 0.2175 Acc: 0.9007
val Loss: 0.1513 Acc: 0.9440
Epoch 32/50
----------
train Loss: 0.2438 Acc: 0.8900
val Loss: 0.1912 Acc: 0.9253
Epoch 33/50
----------
train Loss: 0.2126 Acc: 0.9040
val Loss: 0.2224 Acc: 0.8853
Epoch 34/50
----------
train Loss: 0.2309 Acc: 0.8973
val Loss: 0.1732 Acc: 0.9280
Epoch 35/50
----------
train Loss: 0.2093 Acc: 0.9013
val Loss: 0.1177 Acc: 0.9520
Epoch 36/50
----------
train Loss: 0.2133 Acc: 0.9080
val Loss: 0.1243 Acc: 0.9467
Epoch 37/50
----------
train Loss: 0.2070 Acc: 0.9073
val Loss: 0.4222 Acc: 0.8613
Epoch 38/50
----------
train Loss: 0.2313 Acc: 0.8980
val Loss: 0.1705 Acc: 0.9227
Epoch 39/50
----------
train Loss: 0.2220 Acc: 0.8993
val Loss: 0.1204 Acc: 0.9547
Epoch 40/50
----------
train Loss: 0.2239 Acc: 0.9047
val Loss: 0.1754 Acc: 0.9227
Epoch 41/50
----------
train Loss: 0.2204 Acc: 0.9053
val Loss: 0.1395 Acc: 0.9360
Epoch 42/50
----------
train Loss: 0.2165 Acc: 0.9047
val Loss: 0.1336 Acc: 0.9307
Epoch 43/50
```

```
----------
train Loss: 0.2100 Acc: 0.9080
val Loss: 0.2188 Acc: 0.9147
Epoch 44/50
----------
train Loss: 0.2166 Acc: 0.9060
val Loss: 0.1318 Acc: 0.9413
Epoch 45/50
----------
train Loss: 0.2161 Acc: 0.9027
val Loss: 0.2271 Acc: 0.9200
Epoch 46/50
----------
train Loss: 0.2194 Acc: 0.9013
val Loss: 0.1898 Acc: 0.9280
Epoch 47/50
----------
train Loss: 0.2201 Acc: 0.8947
val Loss: 0.1574 Acc: 0.9253
Epoch 48/50
----------
train Loss: 0.1920 Acc: 0.9160
val Loss: 0.1612 Acc: 0.9467
Epoch 49/50
----------
train Loss: 0.2070 Acc: 0.9100
val Loss: 0.2483 Acc: 0.9227
Epoch 50/50
----------
train Loss: 0.1842 Acc: 0.9280
val Loss: 0.1191 Acc: 0.9387
```

[ ]:
```
import pickle
# Save the entire model object
with open('rice_disease_model_final.pkl', 'wb') as f:
    pickle.dump(model, f)
```

[ ]:
```
import matplotlib.pyplot as plt

# Training and validation loss and accuracy values
train_loss = [0.8224, 0.4892, 0.4770, 0.4391, 0.4026, 0.3576, 0.3403, 0.3737, 0.
↪3198, 0.3341,
              0.2972, 0.2951, 0.3049, 0.2869, 0.2909, 0.2783, 0.2594, 0.2880, 0.
↪2702, 0.2655,
              0.2678, 0.2543, 0.2490, 0.2490, 0.2564, 0.2692, 0.2504, 0.2406, 0.
↪2351, 0.2320,
```

```
                0.2175, 0.2133, 0.2070, 0.2313, 0.2220, 0.2239, 0.2204, 0.2165, 0.
↪2100, 0.2166,
                0.2161, 0.2194, 0.2201, 0.1920, 0.2070, 0.1842]

val_loss = [0.5420, 0.3861, 0.3611, 0.2814, 0.3072, 0.2201, 0.1956, 0.1891, 0.
↪2057, 0.1669,
            0.1739, 0.1902, 0.2012, 0.1504, 0.1459, 0.1594, 0.1642, 0.1501, 0.
↪1595, 0.1518,
            0.1376, 0.1369, 0.1802, 0.1598, 0.1498, 0.1324, 0.1718, 0.1271, 0.
↪1411, 0.1475,
            0.1513, 0.1912, 0.2224, 0.1732, 0.1177, 0.1243, 0.4222, 0.1705, 0.
↪1204, 0.1754,
            0.1395, 0.1336, 0.2188, 0.1318, 0.2271, 0.1898, 0.1574, 0.1612, 0.
↪2483, 0.1191]

train_acc = [0.6160, 0.8020, 0.7933, 0.8040, 0.8127, 0.8400, 0.8580, 0.8380, 0.
↪8600, 0.8480,
             0.8753, 0.8740, 0.8687, 0.8707, 0.8707, 0.8727, 0.8847, 0.8740, 0.
↪8747, 0.8793,
             0.8853, 0.8920, 0.8907, 0.8893, 0.8900, 0.8793, 0.8860, 0.8920, 0.
↪9040, 0.8913,
             0.9007, 0.9080, 0.9073, 0.8980, 0.8993, 0.9047, 0.9053, 0.9047, 0.
↪9080, 0.9060,
             0.9027, 0.9013, 0.8947, 0.9160, 0.9100, 0.9280, 0.9160, 0.9100, 0.
↪9280]

val_acc = [0.7760, 0.8560, 0.8640, 0.8987, 0.8533, 0.9013, 0.9360, 0.9253, 0.
↪9147, 0.9360,
           0.9387, 0.9253, 0.9147, 0.9413, 0.9440, 0.9493, 0.9413, 0.9387, 0.
↪9387, 0.9387,
           0.9547, 0.9387, 0.9173, 0.9360, 0.9493, 0.9493, 0.9360, 0.9440, 0.
↪9413, 0.9440,
           0.9520, 0.9253, 0.8853, 0.9280, 0.9520, 0.9467, 0.8613, 0.9227, 0.
↪9547, 0.9227,
           0.9360, 0.9307, 0.9147, 0.9413, 0.9200, 0.9280, 0.9253, 0.9467, 0.
↪9227, 0.9387]

epochs = range(1, len(train_loss) + 1)

# Plotting training and validation loss
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(epochs, train_loss[:len(epochs)], 'b', label='Training loss')
plt.plot(epochs, val_loss[:len(epochs)], 'r', label='Validation loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
```

```
plt.ylabel('Loss')
plt.legend()

# Plotting training and validation accuracy
plt.subplot(1, 2, 2)
plt.plot(epochs, train_acc[:len(epochs)], 'b', label='Training accuracy')
plt.plot(epochs, val_acc[:len(epochs)], 'r', label='Validation accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

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