rice-disease-alexnet

May 7, 2024

[2]:
```
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
```

[3]:
```
# Define device
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

# Define data directory paths
data_dir = "rice_diseases"
train_dir = os.path.join(data_dir, "train")
val_dir = os.path.join(data_dir, "val")
```

[4]:
```
# 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])
    ]),
}
```

[5]:
```
# Create datasets
image_datasets = {x: datasets.ImageFolder(os.path.join(data_dir, x),␣
 ↪data_transforms[x]) for x in ['train', 'val']}
```

[6]:
```
# Create data loaders
```

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

[7]:
```
# Define the model architecture (using AlexNet)
model = models.alexnet(pretrained=True)
num_ftrs = model.classifier[6].in_features
model.classifier[6] = nn.Linear(num_ftrs, len(image_datasets['train'].classes))
```

```
c:\Users\fr5009tu\AppData\Local\Programs\Python\Python311\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\fr5009tu\AppData\Local\Programs\Python\Python311\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=AlexNet_Weights.IMAGENET1K_V1`. You can also use
`weights=AlexNet_Weights.DEFAULT` to get the most up-to-date weights.
  warnings.warn(msg)
```

[8]:
```
# Move model to device
model = model.to(device)
```

[9]:
```
# Define loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)
```

[10]:
```
# 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}')
```

[11]:
```
# Train the model
train_model(model, criterion, optimizer, num_epochs=50)
```

```
Epoch 1/50
----------
train Loss: 0.7972 Acc: 0.6339
val Loss: 0.5686 Acc: 0.7540
Epoch 2/50
----------
train Loss: 0.5677 Acc: 0.7431
val Loss: 0.4750 Acc: 0.7908
Epoch 3/50
----------
train Loss: 0.5274 Acc: 0.7684
val Loss: 0.4156 Acc: 0.8483
Epoch 4/50
----------
train Loss: 0.4818 Acc: 0.7810
val Loss: 0.3906 Acc: 0.8437
Epoch 5/50
----------
train Loss: 0.4716 Acc: 0.7776
val Loss: 0.3646 Acc: 0.8598
```

```
Epoch 6/50
----------
train Loss: 0.4244 Acc: 0.8029
val Loss: 0.3618 Acc: 0.8092
Epoch 7/50
----------
train Loss: 0.4344 Acc: 0.8092
val Loss: 0.3471 Acc: 0.8322
Epoch 8/50
----------
train Loss: 0.4460 Acc: 0.7891
val Loss: 0.3439 Acc: 0.8621
Epoch 9/50
----------
train Loss: 0.4059 Acc: 0.8086
val Loss: 0.3889 Acc: 0.8299
Epoch 10/50
----------
train Loss: 0.4069 Acc: 0.8098
val Loss: 0.3477 Acc: 0.7977
Epoch 11/50
----------
train Loss: 0.4042 Acc: 0.8057
val Loss: 0.3169 Acc: 0.8552
Epoch 12/50
----------
train Loss: 0.4047 Acc: 0.8155
val Loss: 0.3530 Acc: 0.8437
Epoch 13/50
----------
train Loss: 0.4037 Acc: 0.8098
val Loss: 0.3054 Acc: 0.8598
Epoch 14/50
----------
train Loss: 0.3798 Acc: 0.8236
val Loss: 0.3029 Acc: 0.8736
Epoch 15/50
----------
train Loss: 0.4002 Acc: 0.8057
val Loss: 0.3068 Acc: 0.8690
Epoch 16/50
----------
train Loss: 0.3873 Acc: 0.8046
val Loss: 0.2830 Acc: 0.8621
Epoch 17/50
----------
train Loss: 0.3679 Acc: 0.8253
val Loss: 0.2825 Acc: 0.8782
```

```
Epoch 18/50
----------
train Loss: 0.3500 Acc: 0.8264
val Loss: 0.2929 Acc: 0.8690
Epoch 19/50
----------
train Loss: 0.3649 Acc: 0.8328
val Loss: 0.2852 Acc: 0.8736
Epoch 20/50
----------
train Loss: 0.3376 Acc: 0.8443
val Loss: 0.2849 Acc: 0.8736
Epoch 21/50
----------
train Loss: 0.3566 Acc: 0.8368
val Loss: 0.2666 Acc: 0.8874
Epoch 22/50
----------
train Loss: 0.3544 Acc: 0.8287
val Loss: 0.2709 Acc: 0.8920
Epoch 23/50
----------
train Loss: 0.3390 Acc: 0.8276
val Loss: 0.2678 Acc: 0.8966
Epoch 24/50
----------
train Loss: 0.3436 Acc: 0.8322
val Loss: 0.2827 Acc: 0.8713
Epoch 25/50
----------
train Loss: 0.3637 Acc: 0.8195
val Loss: 0.2717 Acc: 0.8782
Epoch 26/50
----------
train Loss: 0.3232 Acc: 0.8414
val Loss: 0.2885 Acc: 0.8644
Epoch 27/50
----------
train Loss: 0.3280 Acc: 0.8506
val Loss: 0.4008 Acc: 0.8437
Epoch 28/50
----------
train Loss: 0.3407 Acc: 0.8454
val Loss: 0.2636 Acc: 0.8851
Epoch 29/50
----------
train Loss: 0.3454 Acc: 0.8402
val Loss: 0.2656 Acc: 0.8782
```

```
Epoch 30/50
----------
train Loss: 0.3300 Acc: 0.8460
val Loss: 0.2707 Acc: 0.8897
Epoch 31/50
----------
train Loss: 0.3004 Acc: 0.8534
val Loss: 0.2713 Acc: 0.8828
Epoch 32/50
----------
train Loss: 0.3133 Acc: 0.8466
val Loss: 0.2648 Acc: 0.8920
Epoch 33/50
----------
train Loss: 0.3029 Acc: 0.8489
val Loss: 0.2625 Acc: 0.8897
Epoch 34/50
----------
train Loss: 0.3128 Acc: 0.8483
val Loss: 0.2752 Acc: 0.8851
Epoch 35/50
----------
train Loss: 0.3340 Acc: 0.8316
val Loss: 0.2643 Acc: 0.8943
Epoch 36/50
----------
train Loss: 0.3187 Acc: 0.8466
val Loss: 0.2543 Acc: 0.8920
Epoch 37/50
----------
train Loss: 0.3023 Acc: 0.8621
val Loss: 0.2650 Acc: 0.8874
Epoch 38/50
----------
train Loss: 0.3045 Acc: 0.8517
val Loss: 0.2626 Acc: 0.8897
Epoch 39/50
----------
train Loss: 0.3216 Acc: 0.8489
val Loss: 0.2530 Acc: 0.8966
Epoch 40/50
----------
train Loss: 0.2963 Acc: 0.8603
val Loss: 0.2518 Acc: 0.8920
Epoch 41/50
----------
train Loss: 0.2984 Acc: 0.8598
val Loss: 0.2683 Acc: 0.8782
```

```
Epoch 42/50
----------
train Loss: 0.3049 Acc: 0.8523
val Loss: 0.2561 Acc: 0.8874
Epoch 43/50
----------
train Loss: 0.3206 Acc: 0.8460
val Loss: 0.2477 Acc: 0.8874
Epoch 44/50
----------
train Loss: 0.2997 Acc: 0.8598
val Loss: 0.2578 Acc: 0.8736
Epoch 45/50
----------
train Loss: 0.3198 Acc: 0.8483
val Loss: 0.2484 Acc: 0.8874
Epoch 46/50
----------
train Loss: 0.3061 Acc: 0.8575
val Loss: 0.2462 Acc: 0.8805
Epoch 47/50
----------
train Loss: 0.2960 Acc: 0.8598
val Loss: 0.2380 Acc: 0.8966
Epoch 48/50
----------
train Loss: 0.2933 Acc: 0.8580
val Loss: 0.2403 Acc: 0.8943
Epoch 49/50
----------
train Loss: 0.3016 Acc: 0.8592
val Loss: 0.2413 Acc: 0.8943
Epoch 50/50
----------
train Loss: 0.2906 Acc: 0.8672
val Loss: 0.2372 Acc: 0.8897
```

[16]:
```python
import pickle

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

[12]:
```python
torch.save(model.state_dict(), "rice_disease_model.pth")
```