normal-learning

December 20, 2023

0.1 Normal Learning

[13]:
```python
import torch
from torchvision import datasets, transforms
from torch.utils.data import DataLoader
import torch.nn as nn
import torch.optim as optim
import time
from torchvision.models import resnet50, ResNet50_Weights
```

[9]:
```python
def train_model(global_model_name, train_dataset, output_model_name):
    # Load the model architecture
    model = resnet50(weights=None)
    model.eval()

    # Replace the final fully connected layer for transfer learning with the␣
↪same num_classes
    num_ftrs = model.fc.in_features
    num_classes = 6
    model.fc = nn.Linear(num_ftrs, num_classes)

    # Load the trained weights from the saved .pth file
    model.load_state_dict(torch.load(global_model_name))
    model.eval()

    # Define the loss function and optimizer
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.SGD(model.fc.parameters(), lr=0.001, momentum=0.9)

    # Move the model to GPU if available
    device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
    model = model.to(device)

    train_dataloader = torch.utils.data.DataLoader(train_dataset,␣
↪batch_size=32, shuffle=True)
    # Train the model
    start_time = time.time()  # Record the end time of the epoch
    num_epochs = 20  # You can change this
```

```
    for epoch in range(num_epochs):
        model.train()
        running_loss = 0.0

        for inputs, labels in train_dataloader:
            inputs = inputs.to(device)
            labels = labels.to(device)

            optimizer.zero_grad()
            outputs = model(inputs)

            loss = criterion(outputs, labels)

            loss.backward()
            optimizer.step()

            running_loss += loss.item() * inputs.size(0)

        epoch_loss = running_loss / len(train_dataset)
        print(f'Epoch {epoch+1}/{num_epochs} | Loss: {epoch_loss:.4f}')
        if epoch_loss <= 0.01:
            print(" Early Stopping ")
            break
    end_time = time.time()  # Record the end time of the epoch
    print(" Training Time ", end_time - start_time)

    # Save the trained model
    torch.save(model.state_dict(), output_model_name)
    return model
```

[10]:
```
def test_model(data_dir, model_name):
    import torch
    from torchvision import datasets, transforms
    from torch.utils.data import DataLoader
    import torch.nn as nn

    # Define paths to your test dataset folder
    test_data_dir = data_dir  # Update with your test dataset path

    # Define transformations for testing (similar to training)
    test_transforms = transforms.Compose([
        transforms.Resize(64),
        transforms.ToTensor()
    ])

    # Load the test dataset using ImageFolder with the defined transformations
```

```python
    test_dataset = datasets.ImageFolder(root=test_data_dir, transform=test_transforms)

    # Define the test dataloader
    test_dataloader = DataLoader(test_dataset, batch_size=32, shuffle=False, num_workers=4)

    # Load the model architecture
    model = resnet50(weights=None)
    model.eval()

    # Replace the final fully connected layer for transfer learning with the same num_classes
    num_ftrs = model.fc.in_features
    num_classes = 6
    model.fc = nn.Linear(num_ftrs, num_classes)

    # Load the trained weights from the saved .pth file
    model.load_state_dict(torch.load(model_name))
    model.eval()

    # Move the model to GPU if available
    device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
    model = model.to(device)

    # Evaluate the model on the test dataset for both top-1 and top-3 accuracy
    correct_top1 = 0
    correct_top3 = 0

    total = 0
    with torch.no_grad():
        for images, labels in test_dataloader:
            images = images.to(device)
            labels = labels.to(device)

            outputs = model(images)
            _, preds = torch.topk(outputs, 3, dim=1)  # Get top-3 predictions
            total += labels.size(0)
            for i in range(labels.size(0)):
                if labels[i] == preds[i, 0]:  # Check top-1 accuracy
                    correct_top1 += 1
                if labels[i] in preds[i]:  # Check top-3 accuracy
                    correct_top3 += 1

    top1_accuracy = 100 * correct_top1 / total
    top3_accuracy = 100 * correct_top3 / total
    print(f'Top-1 Accuracy on the {data_dir} dataset: {top1_accuracy:.2f}%')
```

```
    print(f'Top-3 Accuracy on the {data_dir} dataset: {top3_accuracy:.2f}%')
```

[11]:
```
train_transforms = transforms.Compose([
        transforms.Resize(64),
        transforms.ToTensor()
    ])

dataset = datasets.ImageFolder(root="../Dataset/train-full",␣
 ↪transform=train_transforms)

train_model("trained_global_model.pth", dataset, "train_normal_model.pth")
test_model('../Dataset/validation_data', "train_normal_model.pth")
```

```
Epoch 1/20 | Loss: 0.3271
Epoch 2/20 | Loss: 0.1286
Epoch 3/20 | Loss: 0.0985
Epoch 4/20 | Loss: 0.0834
Epoch 5/20 | Loss: 0.0734
Epoch 6/20 | Loss: 0.0659
Epoch 7/20 | Loss: 0.0604
Epoch 8/20 | Loss: 0.0600
Epoch 9/20 | Loss: 0.0542
Epoch 10/20 | Loss: 0.0510
Epoch 11/20 | Loss: 0.0488
Epoch 12/20 | Loss: 0.0486
Epoch 13/20 | Loss: 0.0460
Epoch 14/20 | Loss: 0.0435
Epoch 15/20 | Loss: 0.0423
Epoch 16/20 | Loss: 0.0404
Epoch 17/20 | Loss: 0.0404
Epoch 18/20 | Loss: 0.0396
Epoch 19/20 | Loss: 0.0390
Epoch 20/20 | Loss: 0.0366
 Training Time  857.2332816123962
Top-1 Accuracy on the ../Dataset/validation_data dataset: 99.68%
Top-3 Accuracy on the ../Dataset/validation_data dataset: 100.00%
```

[12]:
```
test_model('../Dataset/test_data', "train_normal_model.pth")
```

```
Top-1 Accuracy on the ../Dataset/test_data dataset: 99.63%
Top-3 Accuracy on the ../Dataset/test_data dataset: 100.00%
```

[ ]:
```

```