model training code

April 12, 2024

[1]:
```
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
from torch.utils.data import DataLoader
from torchvision.datasets import CIFAR10
from torchsummary import summary
import matplotlib.pyplot as plt
```

[2]:
```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Using device: {device}")
```

```
Using device: cuda
```

[3]:
```
transform_train = transforms.Compose([
    transforms.RandomCrop(32, padding=4),
    transforms.RandomHorizontalFlip(),
    transforms.ColorJitter(brightness=0.3, contrast=0.3, saturation=0.3, hue=0.
 ↪1),
    transforms.RandomRotation(degrees=20),
    transforms.RandomAffine(degrees=0, translate=(0.1, 0.1)),
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),
])

transform_test = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),
])

trainset = CIFAR10(root='./data', train=True, download=True,␣
 ↪transform=transform_train)
trainloader = DataLoader(trainset, batch_size=128, shuffle=True, num_workers=2)

testset = CIFAR10(root='./data', train=False, download=True,␣
 ↪transform=transform_test)
```

```
testloader = DataLoader(testset, batch_size=100, shuffle=False, num_workers=2)
```

```
Files already downloaded and verified
Files already downloaded and verified
```

[4]:
```
class BasicBlock(nn.Module):
    expansion = 1

    def __init__(self, in_planes, planes, stride=1):
        super(BasicBlock, self).__init__()
        self.conv1 = nn.Conv2d(in_planes, planes, kernel_size=3, stride=stride,␣
↪padding=1, bias=False)
        self.bn1 = nn.BatchNorm2d(planes)
        self.conv2 = nn.Conv2d(planes, planes, kernel_size=3, stride=1,␣
↪padding=1, bias=False)
        self.bn2 = nn.BatchNorm2d(planes)
        self.shortcut = nn.Sequential()
        if stride != 1 or in_planes != self.expansion*planes:
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_planes, self.expansion*planes, kernel_size=1,␣
↪stride=stride, bias=False),
                nn.BatchNorm2d(self.expansion*planes)
            )

    def forward(self, x):
        out = torch.relu(self.bn1(self.conv1(x)))
        out = self.bn2(self.conv2(out))
        out += self.shortcut(x)
        out = torch.relu(out)
        return out
```

[5]:
```
class ModifiedResNet(nn.Module):
    def __init__(self, block, num_blocks, num_classes=10):
        super(ModifiedResNet, self).__init__()
        self.in_planes = 64  # Increased number of initial channels

        self.conv1 = nn.Conv2d(3, 64, kernel_size=3, stride=1, padding=1,␣
↪bias=False)  # Increased initial channels
        self.bn1 = nn.BatchNorm2d(64)
        self.layer1 = self._make_layer(block, 64, num_blocks[0], stride=1)
        self.layer2 = self._make_layer(block, 128, num_blocks[1], stride=2)  #␣
↪Increased channels
        self.layer3 = self._make_layer(block, 256, num_blocks[2], stride=2)  #␣
↪Increased channels
        self.linear = nn.Linear(256*block.expansion, num_classes)

    def _make_layer(self, block, planes, num_blocks, stride):
```

```
        strides = [stride] + [1]*(num_blocks-1)
        layers = []
        for stride in strides:
            layers.append(block(self.in_planes, planes, stride))
            self.in_planes = planes * block.expansion
        return nn.Sequential(*layers)

    def forward(self, x):
        out = torch.relu(self.bn1(self.conv1(x)))
        out = self.layer1(out)
        out = self.layer2(out)
        out = self.layer3(out)
        out = nn.functional.avg_pool2d(out, 8)
        out = out.view(out.size(0), -1)
        out = self.linear(out)
        return out
```

[6]:
```
def ResNetModel():
    return ModifiedResNet(ResidualBlock, [2,2,2])

def count_parameters(model):
    return sum(p.numel() for p in model.parameters() if p.requires_grad)
```

[7]:
```
def train_model(model, criterion, optimizer, scheduler, num_epochs=10):
    train_losses = []
    test_losses = []
    train_accs = []
    test_accs = []

    model.to(device)

    for epoch in range(num_epochs):
        model.train()
        running_loss = 0.0
        correct = 0
        total = 0
        for i, data in enumerate(trainloader, 0):
            inputs, labels = data[0].to(device), data[1].to(device)
            optimizer.zero_grad()

            outputs = model(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            running_loss += loss.item()
            _, predicted = outputs.max(1)
```

```
            total += labels.size(0)
            correct += predicted.eq(labels).sum().item()

        train_loss = running_loss / len(trainloader)
        train_acc = correct / total
        train_losses.append(train_loss)
        train_accs.append(train_acc)

        model.eval()
        correct = 0
        total = 0
        test_loss = 0.0
        with torch.no_grad():
            for data in testloader:
                inputs, labels = data[0].to(device), data[1].to(device)
                outputs = model(inputs)
                loss = criterion(outputs, labels)
                test_loss += loss.item()
                _, predicted = outputs.max(1)
                total += labels.size(0)
                correct += predicted.eq(labels).sum().item()

        test_loss /= len(testloader)
        test_acc = correct / total
        test_losses.append(test_loss)
        test_accs.append(test_acc)

        print(f'Epoch [{epoch+1}/{num_epochs}], Train Loss: {train_loss:.4f},␣
↪Train Acc: {train_acc:.4f}, Test Loss: {test_loss:.4f}, Test Acc: {test_acc:.
↪4f}')
        scheduler.step(test_loss)

    plt.figure(figsize=(10, 5))
    plt.plot(train_losses, label='Train Loss')
    plt.plot(test_losses, label='Test Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.title('Training and Testing Loss Curves')
    plt.legend()
    plt.show()

    plt.figure(figsize=(10, 5))
    plt.plot(train_accs, label='Train Accuracy')
    plt.plot(test_accs, label='Test Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.title('Training and Testing Accuracy Curves')
```

```
    plt.legend()
    plt.show()

    print('Final Test Accuracy: {:.4f}'.format(test_acc))
```

[8]:
```
model = ModifiedResNet(BasicBlock, [2, 2, 2, 2])
model.to(device)
```

[8]:
```
ModifiedResNet(
  (conv1): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (layer1): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (shortcut): Sequential()
    )
    (1): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (shortcut): Sequential()
    )
  )
  (layer2): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
```

```
      (shortcut): Sequential(
        (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (shortcut): Sequential()
    )
  )
  (layer3): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (shortcut): Sequential(
        (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (shortcut): Sequential()
    )
  )
  (linear): Linear(in_features=256, out_features=10, bias=True)
```

```
)
```

[9]:
```
print(summary(model, (3, 32, 32)))
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 64, 32, 32]           1,728
       BatchNorm2d-2           [-1, 64, 32, 32]             128
            Conv2d-3           [-1, 64, 32, 32]          36,864
       BatchNorm2d-4           [-1, 64, 32, 32]             128
            Conv2d-5           [-1, 64, 32, 32]          36,864
       BatchNorm2d-6           [-1, 64, 32, 32]             128
        BasicBlock-7           [-1, 64, 32, 32]               0
            Conv2d-8           [-1, 64, 32, 32]          36,864
       BatchNorm2d-9           [-1, 64, 32, 32]             128
           Conv2d-10           [-1, 64, 32, 32]          36,864
      BatchNorm2d-11           [-1, 64, 32, 32]             128
       BasicBlock-12           [-1, 64, 32, 32]               0
           Conv2d-13          [-1, 128, 16, 16]          73,728
      BatchNorm2d-14          [-1, 128, 16, 16]             256
           Conv2d-15          [-1, 128, 16, 16]         147,456
      BatchNorm2d-16          [-1, 128, 16, 16]             256
           Conv2d-17          [-1, 128, 16, 16]           8,192
      BatchNorm2d-18          [-1, 128, 16, 16]             256
       BasicBlock-19          [-1, 128, 16, 16]               0
           Conv2d-20          [-1, 128, 16, 16]         147,456
      BatchNorm2d-21          [-1, 128, 16, 16]             256
           Conv2d-22          [-1, 128, 16, 16]         147,456
      BatchNorm2d-23          [-1, 128, 16, 16]             256
       BasicBlock-24          [-1, 128, 16, 16]               0
           Conv2d-25            [-1, 256, 8, 8]         294,912
      BatchNorm2d-26            [-1, 256, 8, 8]             512
           Conv2d-27            [-1, 256, 8, 8]         589,824
      BatchNorm2d-28            [-1, 256, 8, 8]             512
           Conv2d-29            [-1, 256, 8, 8]          32,768
      BatchNorm2d-30            [-1, 256, 8, 8]             512
       BasicBlock-31            [-1, 256, 8, 8]               0
           Conv2d-32            [-1, 256, 8, 8]         589,824
      BatchNorm2d-33            [-1, 256, 8, 8]             512
           Conv2d-34            [-1, 256, 8, 8]         589,824
      BatchNorm2d-35            [-1, 256, 8, 8]             512
       BasicBlock-36            [-1, 256, 8, 8]               0
           Linear-37                  [-1, 10]           2,570
================================================================
Total params: 2,777,674
Trainable params: 2,777,674
```

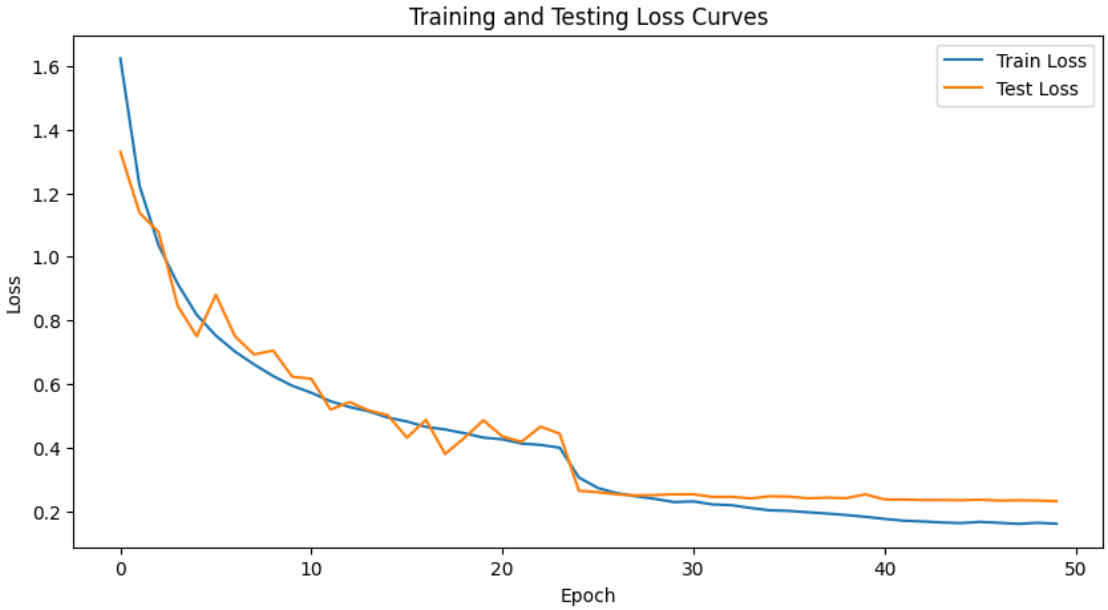
```
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 10.50
Params size (MB): 10.60
Estimated Total Size (MB): 21.11
----------------------------------------------------------------
None
```

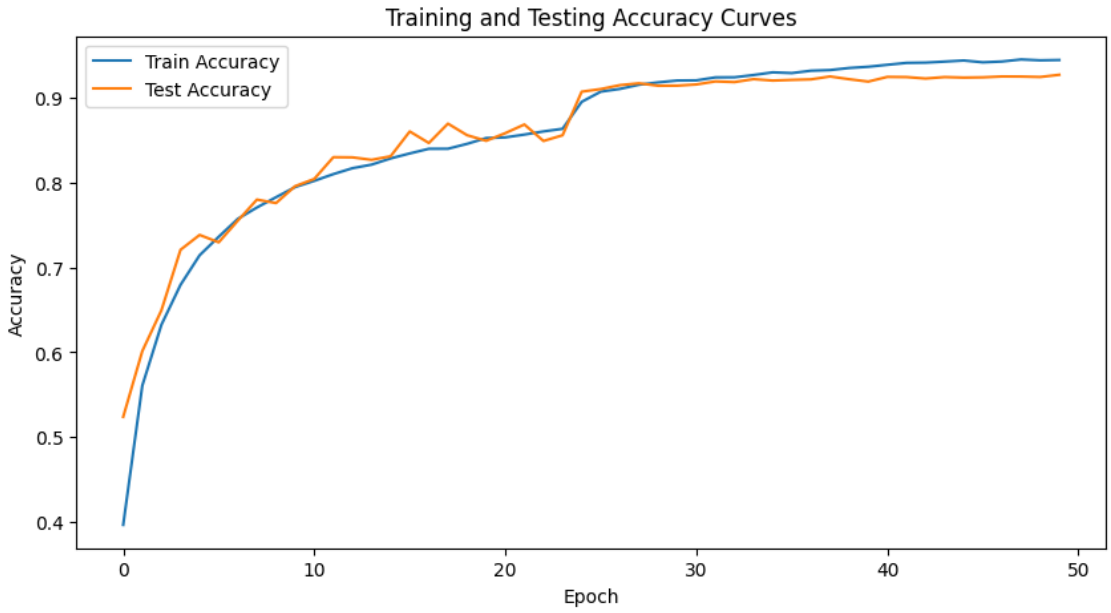
[10]:
```
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001, weight_decay=1e-4)
scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer, mode='min',␣
 ↪factor=0.1, patience=5, verbose=True)

train_model(model, criterion, optimizer, scheduler, num_epochs=50)

torch.save(model.state_dict(), 'modified_resnet_cifar10_model_50_epochs.pth')
print('Model saved successfully!')
```

```
/home/nat/.local/lib/python3.10/site-packages/torch/optim/lr_scheduler.py:28:
UserWarning: The verbose parameter is deprecated. Please use get_last_lr() to
access the learning rate.
  warnings.warn("The verbose parameter is deprecated. Please use get_last_lr() "
Epoch [1/50], Train Loss: 1.6236, Train Acc: 0.3969, Test Loss: 1.3303, Test
Acc: 0.5241
Epoch [2/50], Train Loss: 1.2241, Train Acc: 0.5609, Test Loss: 1.1386, Test
Acc: 0.6015
Epoch [3/50], Train Loss: 1.0363, Train Acc: 0.6325, Test Loss: 1.0775, Test
Acc: 0.6495
Epoch [4/50], Train Loss: 0.9151, Train Acc: 0.6794, Test Loss: 0.8460, Test
Acc: 0.7207
Epoch [5/50], Train Loss: 0.8173, Train Acc: 0.7143, Test Loss: 0.7502, Test
Acc: 0.7382
Epoch [6/50], Train Loss: 0.7521, Train Acc: 0.7361, Test Loss: 0.8804, Test
Acc: 0.7295
Epoch [7/50], Train Loss: 0.7022, Train Acc: 0.7571, Test Loss: 0.7500, Test
Acc: 0.7550
Epoch [8/50], Train Loss: 0.6616, Train Acc: 0.7705, Test Loss: 0.6930, Test
Acc: 0.7798
Epoch [9/50], Train Loss: 0.6251, Train Acc: 0.7826, Test Loss: 0.7053, Test
Acc: 0.7757
Epoch [10/50], Train Loss: 0.5946, Train Acc: 0.7945, Test Loss: 0.6230, Test
Acc: 0.7956
Epoch [11/50], Train Loss: 0.5725, Train Acc: 0.8018, Test Loss: 0.6164, Test
Acc: 0.8041
Epoch [12/50], Train Loss: 0.5459, Train Acc: 0.8098, Test Loss: 0.5196, Test
Acc: 0.8297
```

```
Epoch [13/50], Train Loss: 0.5278, Train Acc: 0.8167, Test Loss: 0.5433, Test
Acc: 0.8295
Epoch [14/50], Train Loss: 0.5142, Train Acc: 0.8209, Test Loss: 0.5171, Test
Acc: 0.8266
Epoch [15/50], Train Loss: 0.4949, Train Acc: 0.8283, Test Loss: 0.5021, Test
Acc: 0.8308
Epoch [16/50], Train Loss: 0.4823, Train Acc: 0.8341, Test Loss: 0.4316, Test
Acc: 0.8600
Epoch [17/50], Train Loss: 0.4652, Train Acc: 0.8396, Test Loss: 0.4874, Test
Acc: 0.8464
Epoch [18/50], Train Loss: 0.4572, Train Acc: 0.8396, Test Loss: 0.3801, Test
Acc: 0.8692
Epoch [19/50], Train Loss: 0.4456, Train Acc: 0.8454, Test Loss: 0.4302, Test
Acc: 0.8556
Epoch [20/50], Train Loss: 0.4317, Train Acc: 0.8522, Test Loss: 0.4859, Test
Acc: 0.8490
Epoch [21/50], Train Loss: 0.4263, Train Acc: 0.8530, Test Loss: 0.4352, Test
Acc: 0.8580
Epoch [22/50], Train Loss: 0.4131, Train Acc: 0.8563, Test Loss: 0.4186, Test
Acc: 0.8682
Epoch [23/50], Train Loss: 0.4087, Train Acc: 0.8600, Test Loss: 0.4664, Test
Acc: 0.8488
Epoch [24/50], Train Loss: 0.4000, Train Acc: 0.8631, Test Loss: 0.4439, Test
Acc: 0.8555
Epoch [25/50], Train Loss: 0.3064, Train Acc: 0.8949, Test Loss: 0.2646, Test
Acc: 0.9068
Epoch [26/50], Train Loss: 0.2732, Train Acc: 0.9069, Test Loss: 0.2601, Test
Acc: 0.9099
Epoch [27/50], Train Loss: 0.2568, Train Acc: 0.9101, Test Loss: 0.2533, Test
Acc: 0.9145
Epoch [28/50], Train Loss: 0.2473, Train Acc: 0.9150, Test Loss: 0.2498, Test
Acc: 0.9167
Epoch [29/50], Train Loss: 0.2390, Train Acc: 0.9178, Test Loss: 0.2512, Test
Acc: 0.9138
Epoch [30/50], Train Loss: 0.2290, Train Acc: 0.9199, Test Loss: 0.2533, Test
Acc: 0.9139
Epoch [31/50], Train Loss: 0.2311, Train Acc: 0.9201, Test Loss: 0.2535, Test
Acc: 0.9153
Epoch [32/50], Train Loss: 0.2217, Train Acc: 0.9236, Test Loss: 0.2454, Test
Acc: 0.9189
Epoch [33/50], Train Loss: 0.2191, Train Acc: 0.9238, Test Loss: 0.2454, Test
Acc: 0.9181
Epoch [34/50], Train Loss: 0.2105, Train Acc: 0.9264, Test Loss: 0.2405, Test
Acc: 0.9216
Epoch [35/50], Train Loss: 0.2030, Train Acc: 0.9296, Test Loss: 0.2473, Test
Acc: 0.9199
Epoch [36/50], Train Loss: 0.2012, Train Acc: 0.9287, Test Loss: 0.2462, Test
Acc: 0.9207
```

```
Epoch [37/50], Train Loss: 0.1970, Train Acc: 0.9315, Test Loss: 0.2410, Test
Acc: 0.9213
Epoch [38/50], Train Loss: 0.1929, Train Acc: 0.9322, Test Loss: 0.2428, Test
Acc: 0.9247
Epoch [39/50], Train Loss: 0.1882, Train Acc: 0.9347, Test Loss: 0.2414, Test
Acc: 0.9215
Epoch [40/50], Train Loss: 0.1827, Train Acc: 0.9361, Test Loss: 0.2532, Test
Acc: 0.9187
Epoch [41/50], Train Loss: 0.1760, Train Acc: 0.9385, Test Loss: 0.2372, Test
Acc: 0.9242
Epoch [42/50], Train Loss: 0.1705, Train Acc: 0.9407, Test Loss: 0.2371, Test
Acc: 0.9240
Epoch [43/50], Train Loss: 0.1681, Train Acc: 0.9410, Test Loss: 0.2355, Test
Acc: 0.9223
Epoch [44/50], Train Loss: 0.1648, Train Acc: 0.9422, Test Loss: 0.2356, Test
Acc: 0.9240
Epoch [45/50], Train Loss: 0.1629, Train Acc: 0.9435, Test Loss: 0.2349, Test
Acc: 0.9234
Epoch [46/50], Train Loss: 0.1667, Train Acc: 0.9414, Test Loss: 0.2366, Test
Acc: 0.9237
Epoch [47/50], Train Loss: 0.1637, Train Acc: 0.9423, Test Loss: 0.2337, Test
Acc: 0.9247
Epoch [48/50], Train Loss: 0.1605, Train Acc: 0.9447, Test Loss: 0.2347, Test
Acc: 0.9246
Epoch [49/50], Train Loss: 0.1638, Train Acc: 0.9437, Test Loss: 0.2338, Test
Acc: 0.9241
Epoch [50/50], Train Loss: 0.1611, Train Acc: 0.9441, Test Loss: 0.2318, Test
Acc: 0.9267
```






```
Final Test Accuracy: 0.9267
Model saved successfully!
```

The model is trained using the Cross Entropy Loss function and optimized with the Adam optimizer, with a learning rate of 0.001 and weight decay set to 1e-4. Additionally, a learning rate scheduler, ReduceLROnPlateau, is employed to adjust the learning rate based on the validation loss.

During training, the model undergoes 50 epochs, during which the training and testing loss and accuracy are evaluated and displayed at each epoch. Both training and testing accuracies demonstrate a steady increase over the epochs, indicating effective learning. The loss and accuracy curves are plotted, depicting the gradual decrease in both training and testing loss alongside the increase in accuracy, affirming the model's learning progress.

[11]:
```python
def evaluate_model(model, dataloader):
    model.eval()
    correct = 0
    total = 0
    with torch.no_grad():
        for data in dataloader:
            inputs, labels = data[0].to(device), data[1].to(device)
            outputs = model(inputs)
            _, predicted = torch.max(outputs, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
    accuracy = correct / total
    print(f'Accuracy on the dataset: {100 * accuracy:.2f}%')
```

```
evaluate_model(model, testloader)
```

```
Accuracy on the dataset: 92.67%
```