```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torchvision
import torchvision.transforms as transforms
from torch.utils.data import DataLoader


class ResBlock(nn.Module):
    def __init__(self, n_chans):
        super().__init__()
        self.conv1 = nn.Conv2d(n_chans, n_chans, kernel_size=3, padding=1)
        self.conv2 = nn.Conv2d(n_chans, n_chans, kernel_size=3, padding=1)

    def forward(self, x):
        identity = x
        out = F.relu(self.conv1(x))
        out = self.conv2(out)
        out += identity
        return F.relu(out)


# Define NetResDeep
class NetResDeep(nn.Module):
    def __init__(self, n_chans1=32, n_blocks=10):
        super().__init__()
        self.n_chans1 = n_chans1
        self.conv1 = nn.Conv2d(3, n_chans1, kernel_size=3, padding=1)
        self.resblocks = nn.Sequential(
            *(n_blocks * [ResBlock(n_chans=n_chans1)])
        )
        self.fc1 = nn.Linear(8 * 8 * n_chans1, 32)
        self.fc2 = nn.Linear(32, 10)  # 10 classes for CIFAR-10

    def forward(self, x):
        out = F.max_pool2d(torch.relu(self.conv1(x)), 2)
        out = self.resblocks(out)
        out = F.max_pool2d(out, 2)
        out = out.view(-1, 8 * 8 * self.n_chans1)
        out = torch.relu(self.fc1(out))
        out = self.fc2(out)
        return out


device = torch.device("cuda" if torch.cuda.is_available() else "cpu")


transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))  # Normalize to [-1, 1]
])

train_dataset = torchvision.datasets.CIFAR10(
    root='./data', train=True, transform=transform, download=True
)
test_dataset = torchvision.datasets.CIFAR10(
    root='./data', train=False, transform=transform, download=True
)

train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=64, shuffle=False)
```

```
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to ./data/cifar-10-python.tar.gz
100%|██████████| 170M/170M [00:03<00:00, 47.9MB/s]
Extracting ./data/cifar-10-python.tar.gz to ./data
Files already downloaded and verified
```

```
# Initialize the model, loss function, and optimizer
model = NetResDeep(n_chans1=32, n_blocks=10).to(device)
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)

# Training loop
num_epochs = 200


for epoch in range(num_epochs):
    model.train()
    running_loss = 0.0
    correct = 0
    total = 0

    for images, labels in train_loader:
        images, labels = images.to(device), labels.to(device)

        # Forward pass
        outputs = model(images)
        loss = criterion(outputs, labels)

        # Backward pass
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        # Statistics
        running_loss += loss.item()
        _, predicted = outputs.max(1)
        total += labels.size(0)
        correct += predicted.eq(labels).sum().item()

    # Print epoch stats
    train_accuracy = 100.0 * correct / total
    print(f"Epoch [{epoch + 1}/{num_epochs}], Loss: {running_loss/len(train_loader):.4f}, Accuracy: {train_accuracy:.2f}%")
```

```
Epoch [1/200], Loss: 1.4736, Accuracy: 46.60%
Epoch [2/200], Loss: 1.0955, Accuracy: 61.08%
Epoch [3/200], Loss: 0.9493, Accuracy: 66.29%
Epoch [4/200], Loss: 0.8719, Accuracy: 69.29%
Epoch [5/200], Loss: 0.8082, Accuracy: 71.47%
Epoch [6/200], Loss: 0.7579, Accuracy: 73.18%
Epoch [7/200], Loss: 0.7250, Accuracy: 74.30%
Epoch [8/200], Loss: 0.6924, Accuracy: 75.56%
Epoch [9/200], Loss: 0.6685, Accuracy: 76.40%
Epoch [10/200], Loss: 0.6433, Accuracy: 77.37%
Epoch [11/200], Loss: 0.6230, Accuracy: 77.84%
Epoch [12/200], Loss: 0.6045, Accuracy: 78.64%
Epoch [13/200], Loss: 0.5843, Accuracy: 79.18%
Epoch [14/200], Loss: 0.5709, Accuracy: 79.74%
Epoch [15/200], Loss: 0.5588, Accuracy: 80.01%
Epoch [16/200], Loss: 0.5548, Accuracy: 80.06%
Epoch [17/200], Loss: 0.5400, Accuracy: 80.69%
Epoch [18/200], Loss: 0.5294, Accuracy: 81.09%
Epoch [19/200], Loss: 0.5208, Accuracy: 81.26%
Epoch [20/200], Loss: 0.5126, Accuracy: 81.53%
Epoch [21/200], Loss: 0.5071, Accuracy: 81.86%
Epoch [22/200], Loss: 0.4995, Accuracy: 82.11%
Epoch [23/200], Loss: 0.4925, Accuracy: 82.43%
Epoch [24/200], Loss: 0.4829, Accuracy: 82.58%
Epoch [25/200], Loss: 0.4840, Accuracy: 82.64%
Epoch [26/200], Loss: 0.4767, Accuracy: 82.73%
Epoch [27/200], Loss: 0.4742, Accuracy: 82.83%
Epoch [28/200], Loss: 0.4706, Accuracy: 83.00%
Epoch [29/200], Loss: 0.4597, Accuracy: 83.50%
Epoch [30/200], Loss: 0.4555, Accuracy: 83.44%
Epoch [31/200], Loss: 0.4547, Accuracy: 83.39%
Epoch [32/200], Loss: 0.4496, Accuracy: 83.72%
Epoch [33/200], Loss: 0.4450, Accuracy: 83.97%
Epoch [34/200], Loss: 0.4416, Accuracy: 84.01%
Epoch [35/200], Loss: 0.4372, Accuracy: 84.13%
Epoch [36/200], Loss: 0.4393, Accuracy: 84.13%
Epoch [37/200], Loss: 0.4336, Accuracy: 84.23%
Epoch [38/200], Loss: 0.4292, Accuracy: 84.45%
Epoch [39/200], Loss: 0.4271, Accuracy: 84.49%
Epoch [40/200], Loss: 0.4248, Accuracy: 84.53%
Epoch [41/200], Loss: 0.4150, Accuracy: 84.88%
Epoch [42/200], Loss: 0.4183, Accuracy: 84.90%
Epoch [43/200], Loss: 0.4123, Accuracy: 85.08%
Epoch [44/200], Loss: 0.4112, Accuracy: 85.21%
Epoch [45/200], Loss: 0.4082, Accuracy: 85.14%
Epoch [46/200], Loss: 0.4017, Accuracy: 85.53%
Epoch [47/200], Loss: 0.4060, Accuracy: 85.27%
Epoch [48/200], Loss: 0.4025, Accuracy: 85.52%
Epoch [49/200], Loss: 0.3980, Accuracy: 85.29%
Epoch [50/200], Loss: 0.3952, Accuracy: 85.61%
Epoch [51/200], Loss: 0.3948, Accuracy: 85.53%
Epoch [52/200], Loss: 0.4006, Accuracy: 85.22%
Epoch [53/200], Loss: 0.3857, Accuracy: 85.91%
Epoch [54/200], Loss: 0.3933, Accuracy: 85.85%
Epoch [55/200], Loss: 0.3916, Accuracy: 85.79%
Epoch [56/200], Loss: 0.3886, Accuracy: 85.73%
Epoch [57/200], Loss: 0.3841, Accuracy: 86.20%
Epoch [58/200], Loss: 0.3868, Accuracy: 85.82%
```