神经网络实验报告

学号：SA24219059　　姓名：周子谕

一、实验目的

学习神经网络图像分类方法。

二、实验内容

训练 CNN，来对 CIFAR-10 数据集进行图像分类。

三、实验平台

Pytorch, torchvision, matplotlib

四、代码（对核心代码进行详细说明。如调用库函数，详细分析参数及返回值）

```
'''Train CIFAR10 with PyTorch.'''
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
import torch.backends.cudnn as cudnn

import torchvision
import torchvision.transforms as
transforms

import os
import argparse

from models import *
from utils import progress_bar


parser =
argparse.ArgumentParser(description='PyT
orch CIFAR10 Training')
parser.add_argument('--lr', default=0.1,
type=float, help='learning rate')
parser.add_argument('--resume', '-r',
action='store_true',
                    help='resume from
checkpoint')
args = parser.parse_args()

device = 'cuda' if
torch.cuda.is_available() else 'cpu'
best_acc = 0  # best test accuracy
start_epoch = 0  # start from epoch 0 or
last checkpoint epoch

# Data
print('==> Preparing data..')
transform_train = transforms.Compose([
    transforms.RandomCrop(32, padding=4),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    transforms.Normalize((0.4914, 0.4822,
0.4465), (0.2023, 0.1994, 0.2010)),
])

transform_test = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.4914, 0.4822,
0.4465), (0.2023, 0.1994, 0.2010)),
])

trainset = torchvision.datasets.CIFAR10(
    root='./data', train=True,
download=True,
transform=transform_train)
trainloader =
torch.utils.data.DataLoader(
    trainset, batch_size=128,
shuffle=True, num_workers=2)

testset = torchvision.datasets.CIFAR10(
    root='./data', train=False,
download=True, transform=transform_test)
testloader =
torch.utils.data.DataLoader(
    testset, batch_size=100,
```

```
shuffle=False, num_workers=2)

classes = ('plane', 'car', 'bird',
'cat', 'deer',
          'dog', 'frog', 'horse', 'ship',
'truck')

# Model
print('==> Building model..')
net = ResNet18()
net = net.to(device)
if device == 'cuda':
    net = torch.nn.DataParallel(net)
    cudnn.benchmark = True

if args.resume:
    # Load checkpoint.
    print('==> Resuming from
checkpoint..')
    assert os.path.isdir('checkpoint'),
'Error: no checkpoint directory found!'
    checkpoint =
torch.load('./checkpoint/ckpt.pth')

net.load_state_dict(checkpoint['net'])
    best_acc = checkpoint['acc']
    start_epoch = checkpoint['epoch']

criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(),
lr=args.lr,
                      momentum=0.9,
weight_decay=5e-4)
scheduler =
torch.optim.lr_scheduler.CosineAnnealing
LR(optimizer, T_max=200)


# Training
def train(epoch):
    print('\nEpoch: %d' % epoch)
    net.train()
    train_loss = 0
    correct = 0
    total = 0
    for batch_idx, (inputs, targets) in
enumerate(trainloader):
        inputs, targets =
inputs.to(device), targets.to(device)
        optimizer.zero_grad()
        outputs = net(inputs)
        loss = criterion(outputs, targets)
        loss.backward()
        optimizer.step()

        train_loss += loss.item()
        _, predicted = outputs.max(1)
        total += targets.size(0)
        correct +=
predicted.eq(targets).sum().item()

        progress_bar(batch_idx,
len(trainloader), 'Loss: %.3f |
Acc: %.3f%% (%d/%d)'
                     %
(train_loss/(batch_idx+1),
100.*correct/total, correct, total))


def test(epoch):
    global best_acc
    net.eval()
    test_loss = 0
    correct = 0
    total = 0
    with torch.no_grad():
        for batch_idx, (inputs, targets)
in enumerate(testloader):
            inputs, targets =
inputs.to(device), targets.to(device)
            outputs = net(inputs)
            loss = criterion(outputs,
targets)

            test_loss += loss.item()
            _, predicted = outputs.max(1)
            total += targets.size(0)
            correct +=
```

```
predicted.eq(targets).sum().item()

            progress_bar(batch_idx,
len(testloader), 'Loss: %.3f |
Acc: %.3f%% (%d/%d)'
                         %
(test_loss/(batch_idx+1),
100.*correct/total, correct, total))

    # Save checkpoint.
    acc = 100.*correct/total
    if acc > best_acc:
        print('Saving..')
        state = {
            'net': net.state_dict(),
            'acc': acc,
            'epoch': epoch,
        }
        if not
os.path.isdir('checkpoint'):
            os.mkdir('checkpoint')
        torch.save(state,
'./checkpoint/ckpt.pth')
        best_acc = acc


for epoch in range(start_epoch,
start_epoch+200):
    train(epoch)
    test(epoch)
    scheduler.step()
```

五、实验结果与分析

1.前期准备

parser = argparse.ArgumentParser(description='PyTorch CIFAR10 Training')

parser.add_argument('--lr', default=0.1, type=float, help='learning rate') # 学习率默认 0.1

parser.add_argument('--resume', '-r', action='store_true', help='resume from checkpoint') # 断点续训

通过上述内容实现对学习率的预先设置。

device = 'cuda' if torch.cuda.is_available() else 'cpu' # 自动检测 GPU

cudnn.benchmark = True # 启用 CuDNN 自动优化器

配置硬件。

transform_train = transforms.Compose([

    transforms.RandomCrop(32, padding=4), # 随机裁剪

    transforms.RandomHorizontalFlip(), # 水平翻转

    transforms.ToTensor(),

    transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010)) # CIFAR10 标准归一化

])

通过 transforms 进行数据增强。

net = ResNet18() # 默认使用 ResNet-18

net = net.to(device)

if device == 'cuda':

    net = torch.nn.DataParallel(net) # 多 GPU 支持

启用 ResNet-18，启用数据并行（多 GPU 训练）

if args.resume:

    checkpoint = torch.load('./checkpoint/ckpt.pth')

    net.load_state_dict(checkpoint['net']) # 加载模型参数

    best_acc = checkpoint['acc'] # 历史最佳准确率

    start_epoch = checkpoint['epoch'] # 恢复训练轮次

断点恢复机制。

criterion = nn.CrossEntropyLoss() # 交叉熵损失

```
optimizer = optim.SGD(net.parameters(), lr=args.lr,
                      momentum=0.9, weight_decay=5e-4)   # 带动量与 L2 正则
scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=200)   # 余弦退火学习率
SGD + 动量 + 权重衰减，200 个 epoch 的余弦退火策略
for epoch in range(start_epoch, start_epoch+200):
    train(epoch)
    test(epoch)
    scheduler.step()   # 更新学习率
```

进行训练循环。

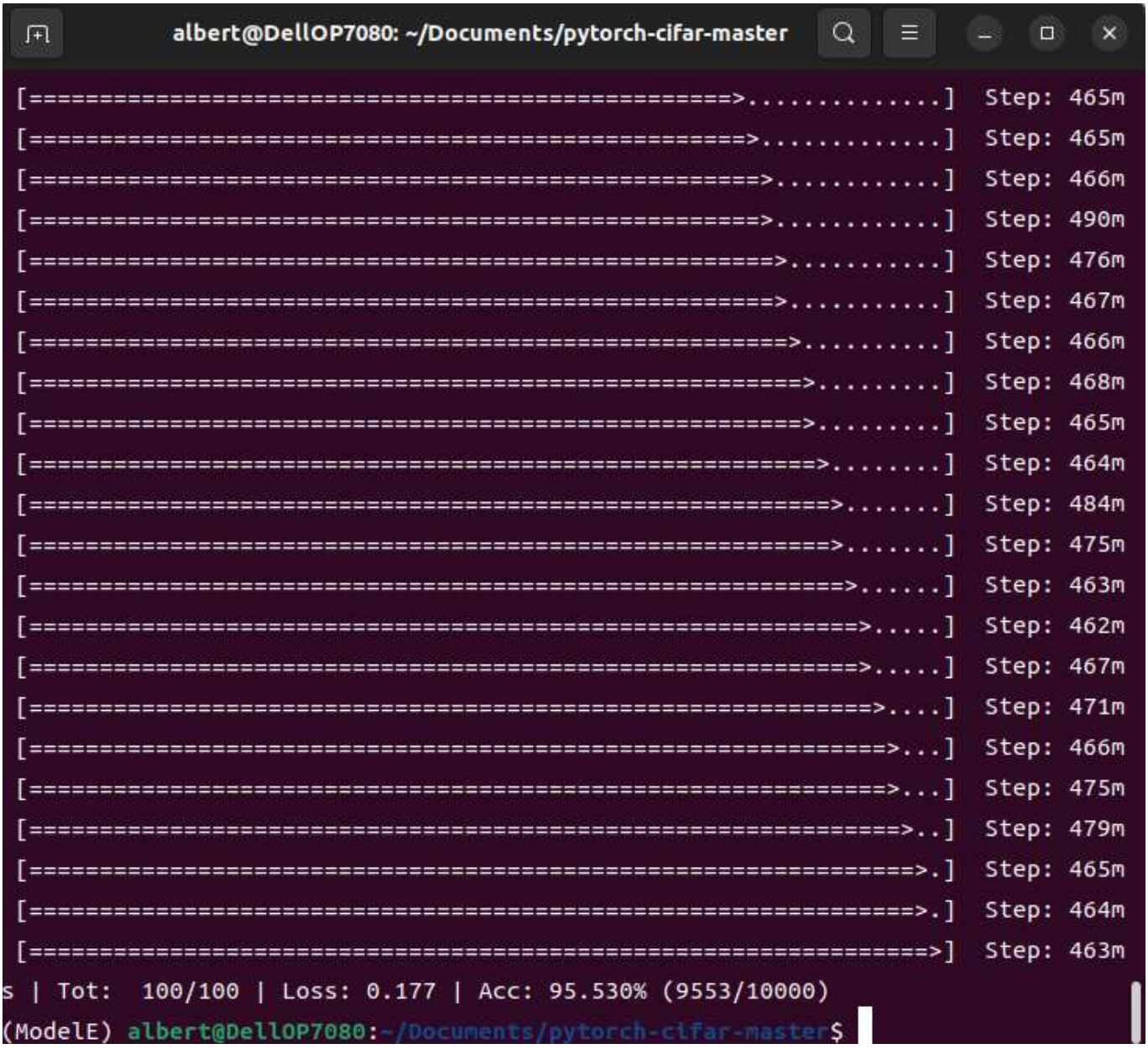

图 5-1 训练结果

六、总结

使用 CIFAR-10 数据集进行训练，直接采用 ResNet-18 网络结构，训练轮次 200 轮，最终正确率达到 95.53%。