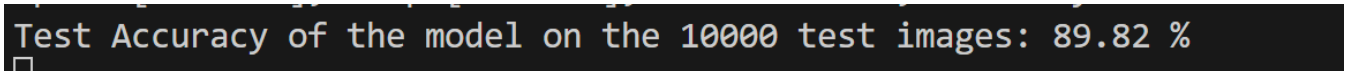
2400010833 闵诗珈

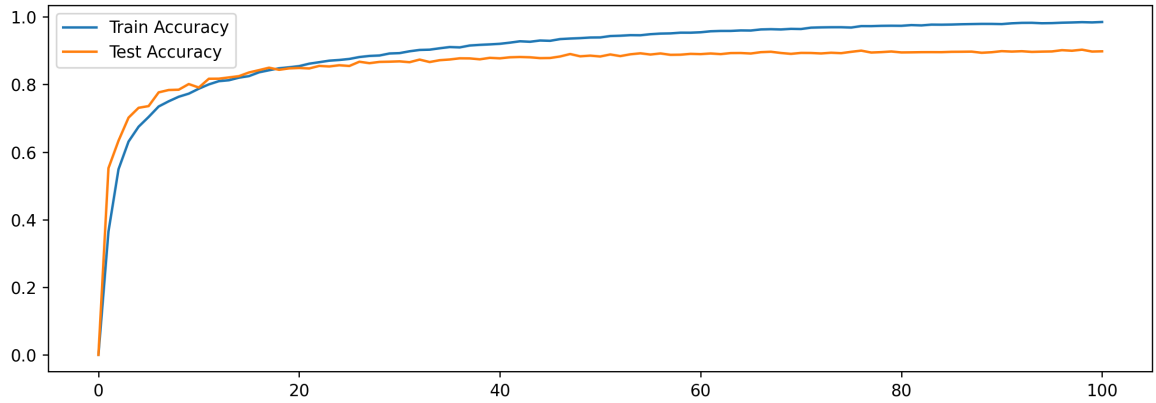
人工智能基础

作业三

1. 网络实现过程
   - 仅采用作业说明要求的一层`nn.Conv2d`、`nn.MaxPool2d`，并按经验添加`ReLU`函数时，测试集正确率较低（<55%）。
   - 于是我增加了卷积层、池化层数目，并且对图像进行了数据增强处理。此时计算成本增加，正确率上升（根据层数正确率处于<70%~74%）。
   - 考虑在卷积层后引入`BatchNorm2d`进行归一化后，略有改善但不明显，正确率约75%。但引入该函数明显增大了计算复杂度，CPU运算时间大大增长（甚至降到1小时10个epoch）。
   - `ReLU`换成`LeakyReLU`后正确率有小幅改善（最高到达78%）。但仍未到80%要求。
   - 查找CIFAR-10相关论文，发现ELU函数：arXiv:1511.07289v5 [cs.LG] 22 Feb 2016。其表达式为 $f(x) = \begin{cases} x, & x > 0 \\ \alpha(\exp(x) - 1), & x \leq 0 \end{cases}$，在ReLU的基础上有负值，能使激活均值更接近零，减少偏置偏移效应，让标准梯度更接近自然梯度，从而加快学习速度。此外，ELU对噪声更鲁棒。因此可以用ELU替代BatchNorm2d+ReLU，训练效果提高的前提下大大提高了训练速度，于是在计算成本可控的前提下再次增加卷积核数量和通道数，形成了最终的网络结构。
   - 最终网络结构有较好的特征提取能力，100个epoch后测试集正确率接近90%。
2. 网络结构
   - 查找学习了一些经典的CNN网络结构，在基础上自己改编的网络结构。
   - 卷积层：设计了多个卷积层，提取图像的特征；

     具体参数：通过跑10个epoch来调整（包括增删卷积层和调整参数等），最终确定了代码使用的网络结构。
   - 激活函数：使用 ELU 激活函数，相比于 ReLU 能缓解神经元死亡问题，且在负半轴有一定的输出，能加速收敛；
   - 池化层：在卷积后衔接MaxPool2d，减少参数数量和计算量，同时保留重要特征；
   - 全连接层：将卷积层输出展平并最终输出10个类别。
3. 数据增强技术
   - 使用了随机裁剪、水平翻转、色彩抖动等数据增强技术，增加了训练数据的多样性，提高模型繁华能力
   - 超参数的选择：归一化参数通过`mean.py`（见报告后附代码）计算得到，`ColorJitter`通过小规模训练得到合适参数。
4. 其他修改
   - `torch.optim.lr_scheduler.StepLR(optimizer, step_size=5, gamma=0.9)`在训练过程中调整学习率，有效减少了过拟合。
   - 使用8进程，对CPU压力较小同时加快了训练速度。
5. 最终模型性能评估：约15个epoch后测试集正确率达到80%。100个epoch后，训练集准确率在99%以上，测试集正确率为89.82%。由于epoch较多文本过长，此处作图显示训练过程中以及训练结束时的模型性能。





最终训练使用CPU: Intel i7 1360P，100个epoch耗时5h。

可以发现模型仍然有过拟合现象，未来改进可以考虑交叉验证（比如K折交叉验证）或者换用其他动态调整学习率的函数等方式。

代码

`cifar10_cnn.py`

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

# 定义超参数
batch_size = 128
learning_rate = 0.001
num_epochs = 100

# 数据增强的数据预处理方式
transform_train = transforms.Compose([
    transforms.RandomCrop(32, padding=4),
    transforms.RandomHorizontalFlip(),
    transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.1),
    transforms.ToTensor(),
    transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2470, 0.2435, 0.2616))  # 由mean.py计
算得到合适参数
])
transform_test = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2470, 0.2435, 0.2616))
])


class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.features = nn.Sequential(
            # 1
```

```
            nn.Conv2d(3, 32, kernel_size=3, stride=1, padding=1),
            nn.ELU(True),
            nn.Conv2d(32, 32, kernel_size=3, stride=1, padding=1),
            nn.ELU(True),
            nn.MaxPool2d(2, 2),
            # 2
            nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1),
            nn.ELU(True),
            nn.Conv2d(64, 64, kernel_size=3, stride=1, padding=1),
            nn.ELU(True),
            nn.MaxPool2d(2, 2),
            # 3
            nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1),
            nn.ELU(True),
            nn.Conv2d(128,128, kernel_size=3, stride=1, padding=1),
            nn.MaxPool2d(2, 2),
            # 4
            nn.Conv2d(128, 256, kernel_size=3, stride=1, padding=1),
            nn.ELU(True),
            nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=1),
            nn.ELU(True),
            nn.MaxPool2d(2, 2)

        )

        self.classify = nn.Sequential(
            nn.Linear(1024,200),
            nn.ReLU(True),
            nn.Dropout(0.5),
            nn.Linear(200,10)
        )

    def forward(self, x):
        x=self.features(x)
        x=x.view(x.size(0),-1)
        x=self.classify(x)
        return x


def train_model():
    # 实例化模型
    model = Net()
    device = torch.device('cpu')
    model = model.to(device)

    # 定义损失函数和优化器
    criterion = nn.CrossEntropyLoss()
    optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate, weight_decay=0.0001)
    scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=5, gamma=0.9)

    train_acc=[0]
    test_acc=[0]

    # 训练模型
    for epoch in range(num_epochs):
        # 训练模式
```

```
        model.train()
        epoch_train_acc=0
        num_batches=0
        for i, (images, labels) in enumerate(train_loader):
            images = images.to(device)
            labels = labels.to(device)

            # 前向传播
            outputs = model(images)
            loss = criterion(outputs, labels)

            # 反向传播
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            accuracy = (outputs.argmax(1) == labels).float().mean()
            epoch_train_acc+=accuracy.item()
            num_batches+=1
            # train_acc.append(accuracy.item())

            # 打印训练信息
            if (i + 1) % 100 == 0:
                print('Epoch [{}/{}], Step [{}/{}], Loss: {:.4f}, Accuracy:
{:.2f}%'.format(
                    epoch + 1, num_epochs, i + 1, len(train_loader), loss.item(),
accuracy.item() * 100))
        epoch_avg_train_acc=epoch_train_acc/num_batches
        train_acc.append(epoch_avg_train_acc)

        scheduler.step()

        # 测试模式
        model.eval()
        with torch.no_grad():
            correct = 0
            total = 0
            for images, labels in test_loader:
                images = images.to(device)
                labels = labels.to(device)

                outputs = model(images)
                _, predicted = torch.max(outputs.data, 1)
                total += labels.size(0)
                correct += (predicted == labels).sum().item()

            print('Test Accuracy of the model on the 10000 test images: {} %'.format(100 *
correct / total))
            test_acc.append(correct/total)

        # 绘图
    p.figure(figsize=(12,4))
    p.plot(train_acc, label='Train Accuracy')
    p.plot(test_acc, label='Test Accuracy')
    p.legend()
    p.show()
```

```
if __name__ == '__main__':
    # 定义数据集
    train_dataset = torchvision.datasets.CIFAR10(root='./data', train=True, download=True,
transform=transform_train)
    test_dataset = torchvision.datasets.CIFAR10(root='./data', train=False, download=True,
transform=transform_test)

    # 定义数据加载器
    train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True,
num_workers=8)
    test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False,
num_workers=8)

    train_model()
```

mean.py

```
import torch
import torchvision
import torchvision.transforms as transforms

# 仅将图像转换为张量，不进行其他处理
transform = transforms.Compose([transforms.ToTensor()])

# 加载训练集
trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                        download=True, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=len(trainset),
shuffle=False)

# 计算均值和标准差
data = next(iter(trainloader))[0]
mean = data.mean(dim=(0, 2, 3))
std = data.std(dim=(0, 2, 3))

print(f"Mean: {mean}")
print(f"Std: {std}")
```