计算机视觉技术基础
课程实践
实践项目一、 Resnet-18 实现图像分类

一、实验目的

1. 了解图像分类任务
2. 了解深度学习技术在图像分类领域的应用
3. 学习神经网络的设计与实现

二、实验内容

基于 CIFAR100 数据集，使用 Resnet 模型进行图像分类任务。

三、实验步骤

1.Resnet 介绍

随着我们设计越来越深的网络，深刻理解“新添加的层如何提升神经网络的性能”变得至关重要。Resnet 是一个经典的神经网络，它引入了残差的概念。何恺明等人提出了残差网络（ResNet）[He et al., 2016a]。它在 2015 年的 ImageNet 图像识别挑战赛夺魁，并深刻影响了后来的深度神经网络的设计。残差网络的核心思想是：每个附加层都应该更容易地包含原始函数作为其元素之一。于是，残差块（residual blocks）便诞生了，这个设计对如何建立深层神经网络产生了深远的影响。凭借它，ResNet 赢得了 2015 年 ImageNet 大规模视觉识别挑战赛。

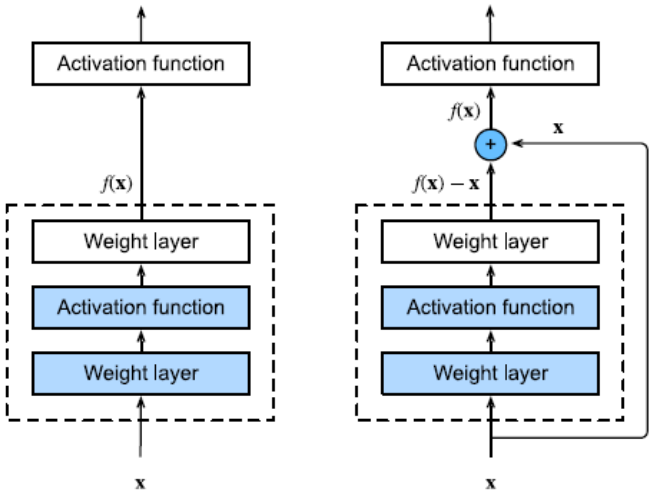

图 1 一个正常块（左图）和一个残差块（右图）

ResNet 沿用了 VGG 完整的 3x3 卷积层设计。残差块里首先有 2 个有相同输出通道数的 3x3 卷积层。每个卷积层后接一个批量归一化层和 ReLU 激活函数。然后我们通过跨层数据通路，跳过这 2 个卷积运算，将输入直接加在最后的

ReLU 激活函数前。这样的设计要求 2 个卷积层的输出与输入形状一样，从而可以相加。如果想改变通道数，就需要引入一个额外的 1x1 卷积层来将输入变换成需要的形状后再做相加运算。

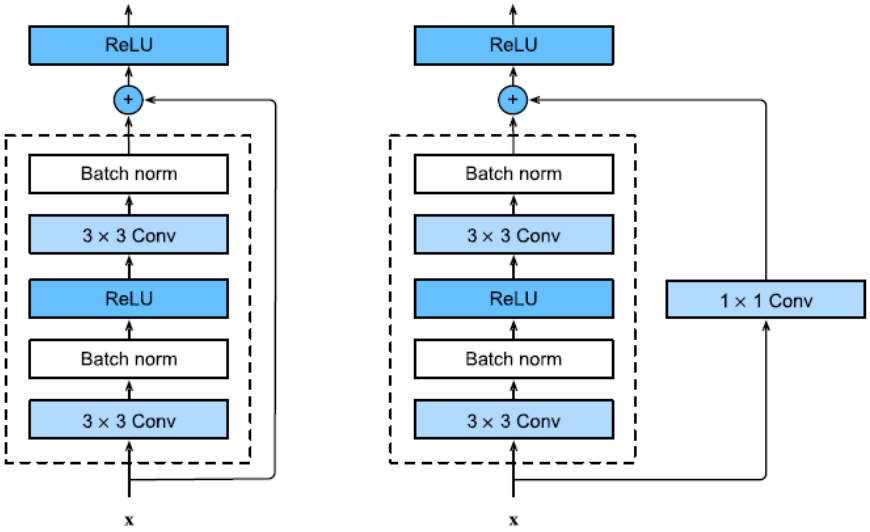

图 2 包含以及不包含 1x1 卷积层的残差块

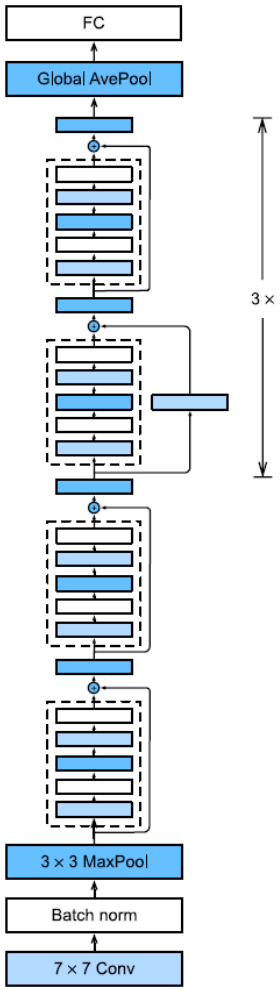

图 3 ResNet-18 架构

2. Resnet 实现

（1）导入工具包

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

（2）定义 ResNet 模型

```
class BasicBlock(torch.nn.Module):
    """残差块"""

    def __init__(self, inplanes, planes, stride=1):
        """初始化"""

        super(BasicBlock, self).__init__()

        self.conv1 = torch.nn.Conv2d(in_channels=inplanes, out_channels=planes, kernel_size=(3, 3),
                                     stride=(stride, stride), padding=1)  # 卷积层1
        self.bn1 = torch.nn.BatchNorm2d(planes)  # 标准化层1


        self.conv2 = torch.nn.Conv2d(in_channels=planes, out_channels=planes, kernel_size=(3, 3), padding=1)  # 卷积层2
        self.bn2 = torch.nn.BatchNorm2d(planes)  # 标准化层2

        # 如果步长不为1，用1*1的卷积实现下采样
        if stride != 1:
            self.downsample = torch.nn.Sequential(
                # 下采样
                torch.nn.Conv2d(in_channels=inplanes, out_channels=planes, kernel_size=(1, 1), stride=(stride, stride)))
        else:
            self.downsample = lambda x: x  # 返回x

    def forward(self, input):
        """前向传播"""
        out = self.conv1(input)
        out = self.bn1(out)
        out = F.relu(out)

        out = self.conv2(out)
        out = self.bn2(out)

        identity = self.downsample(input)
        output = torch.add(out, identity)
        output = F.relu(output)

        return output
```

```
ResNet_18 = torch.nn.Sequential(
    # 初始层
    torch.nn.Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1)),  # 卷积
    torch.nn.BatchNorm2d(64),
    torch.nn.ReLU(),
    torch.nn.MaxPool2d((2, 2)),  # 池化

    # 8个block(每个为两层)
    BasicBlock(64, 64, stride=1),
    BasicBlock(64, 64, stride=1),
    BasicBlock(64, 128, stride=2),
    BasicBlock(128, 128, stride=1),
    BasicBlock(128, 256, stride=2),
    BasicBlock(256, 256, stride=1),
    BasicBlock(256, 512, stride=2),
    BasicBlock(512, 512, stride=1),
    torch.nn.AvgPool2d(2),  # 池化

    torch.nn.Flatten(),  # 平铺层

    # 全连接层
    torch.nn.Linear(512, 100)  # 100类
```

(3) 获得数据集

```
def get_data():
    """获取数据"""

    # 获取测试集
    train = torchvision.datasets.CIFAR100(root="./data", train=True, download=True,
                                          transform=torchvision.transforms.Compose([
                                              torchvision.transforms.ToTensor(),  # 转换成张量
                                              torchvision.transforms.Normalize((0.1307,), (0.3081,))  # 标准化
                                          ]))
    train_loader = DataLoader(train, batch_size=batch_size)  # 分割测试集

    # 获取测试集
    test = torchvision.datasets.CIFAR100(root="./data", train=False, download=True,
                                         transform=torchvision.transforms.Compose([
                                             torchvision.transforms.ToTensor(),  # 转换成张量
                                             torchvision.transforms.Normalize((0.1307,), (0.3081,))  # 标准化
                                         ]))
    test_loader = DataLoader(test, batch_size=batch_size)  # 分割训练

    # 返回分割好的训练集和测试集
    return train_loader, test_loader
```

（4）定义训练过程（基于训练集）

```
def train(model, epoch, train_loader):
    """训练"""
    # 训练模式
    model.train()
    # 迭代
    for step, (x, y) in enumerate(train_loader):
        # 加速
        if use_cuda:
            model = model.cuda()
            x, y = x.cuda(), y.cuda()
        # 梯度清零
        optimizer.zero_grad()
        output = model(x)
        # 计算损失
        loss = F.cross_entropy(output, y)
        # 反向传播
        loss.backward()
        # 更新梯度
        optimizer.step()
        # 打印损失
        if step % 10 == 0:
            print('Epoch: {}, Step {}, Loss: {}'.format(epoch, step, loss))
```

（5）定义测试验证（基于验证集）

```
def test(model, test_loader):
    """测试"""
    # 测试模式
    model.eval()
    # 存放正确个数
    correct = 0
    with torch.no_grad():
        for x, y in test_loader:
            # 加速
            if use_cuda:
                model = model.cuda()
                x, y = x.cuda(), y.cuda()
            # 获取结果
            output = model(x)
            # 预测结果
            pred = output.argmax(dim=1, keepdim=True)
            # 计算准确个数
            correct += pred.eq(y.view_as(pred)).sum().item()
    # 计算准确率
    accuracy = correct / len(test_loader.dataset) * 100
    # 输出准确
    print("Test Accuracy: {}%".format(accuracy))
```

（6）定义一些超参数

```
# 定义超参数
batch_size = 1024  # 一次训练的样本数目
learning_rate = 0.0001  # 学习率
iteration_num = 100  # 迭代次数
network = ResNet_18
optimizer = torch.optim.Adam(network.parameters(), lr=learning_rate)  # 优化器
# GPU 加速
use_cuda = torch.cuda.is_available()
if use_cuda:
    network.cuda()
print("是否使用 GPU 加速:", use_cuda)
```

（7）开始训练

```
def main():
    # 获取数据
    train_loader, test_loader = get_data()

    # 迭代
    for epoch in range(iteration_num):
        print("\n================ epoch: {} ================".format(epoch))
        train(network, epoch, train_loader)
        test(network, test_loader)

if __name__ == "__main__":
    main()
```

四、实验作业提交

1. 根据以上过程使用华为云计算平台实现 Resnet 并进行训练，并打印 train loss 和 test loss 随 Epoch 的变化。
2. 回答下列问题

（1）解释什么是 batch size

（2）什么是一个 epoch

（3）什么是过拟合，如何克服过拟合现象？

（4）你认为 Resnet 引入的残差块为什么有效？

（5）有哪些常用的激活函数？分析这些激活函数的特点。

（6）常用的优化器 optimizer 有哪些？这些优化器各有什么特点？

3. 尝试对以上模型进行修改，以提高基于 CIFAR-100 数据集的分类精度。可以从一下角度进行：数据预处理（数据增强等）、改变模型结构、改变训练策略等。提供你的思路和源代码，用记录训练结果以证明你的方法有效。