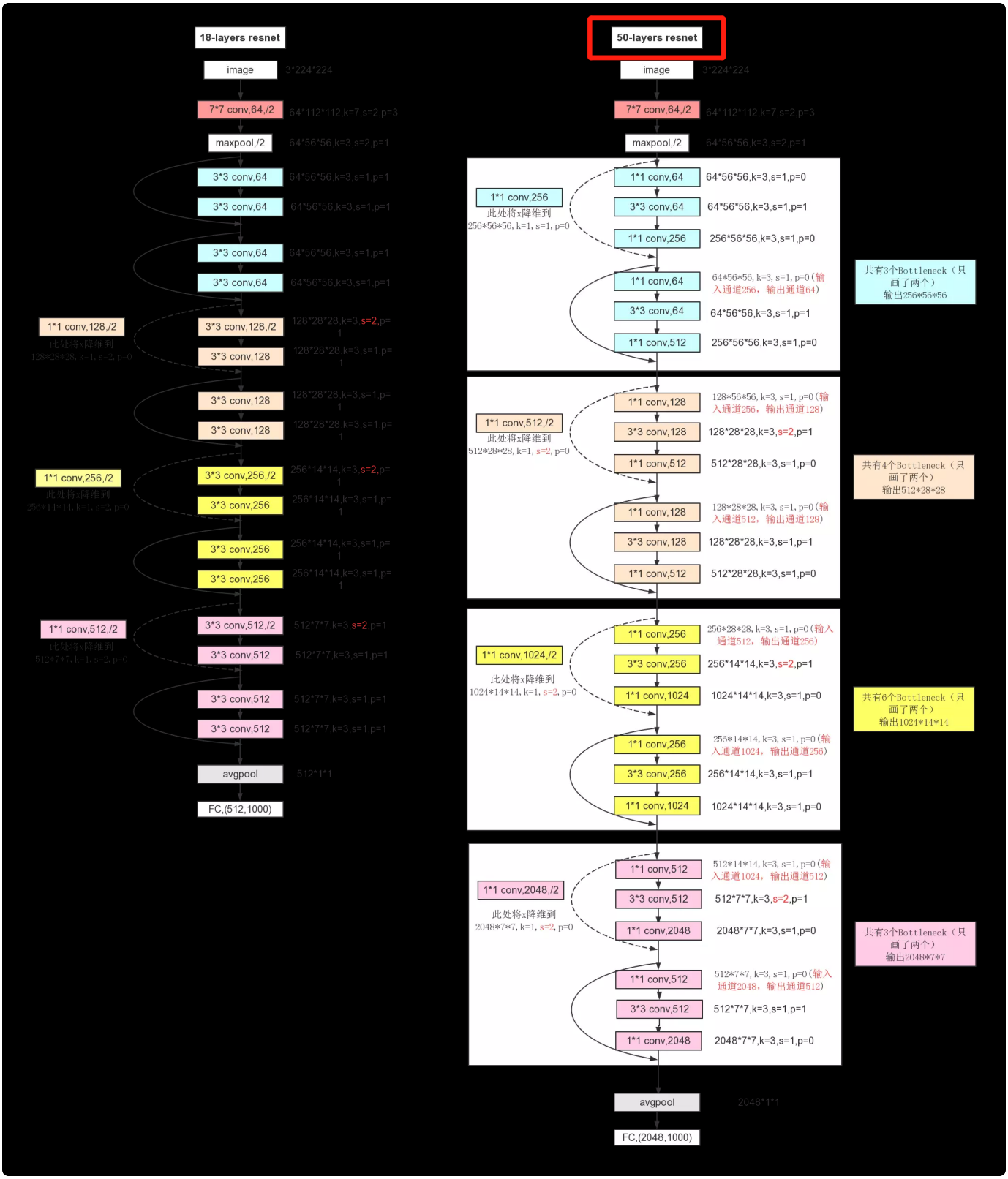
PJ1 第三部分 预训练模型

实验原理

模型选择

在本实验中，我们使用PyTorch深度学习框架中的torchvision.models库导入预训练的ResNet50模型。ResNet50是ResNet的一种变体，是2015年ILSVRC挑战赛 (ImageNet Large Scale Visual Recognition Challenge) 的冠军，具有50层深度，在ImageNet数据集上表现优秀，被广泛应用于计算机视觉任务。ResNet50采用残差连接来解决深度卷积网络中出现的问题，能够有效地训练深层网络，并取得了较好的性能。



上面这个网络图展示的是标准ResNet50的forward_propagate过程。首先输入的图片是3×224×224，也就是3个通道，图片尺寸为224×224；进入第一个卷积层，卷积核大小为7*7，卷积核个数为64，步长为2，padding为3；所以输出应该是（224−7+2×3）/2+1=112.5，向下取整得到112，所以输出是 64 ×112 ×112；maxpool层会改变维度，但是不会影响个数。

到此即将进入第一个实线方框中，第一个实线方框中本来应该有3个Bottleneck，作者只画了两个。每一个Bottleneck里面包含两种Block，一种是Conv Block，一种是Identity Block。Conv Block是第一个实线方框中虚线连接的三层：可以看到，总体的思路是先通过1×1的卷积对特征图像进行降维，做一次3×3的卷积操作，最后再通过1×1卷积恢复维度，后面跟着BN和ReLU层；虚线处用256个1×1的卷积网络，将maxpool的输出降维到255×56×56。Identity Block是实线连接所示，不经过卷积网络降维，直接将输入加到最后的1×1卷积输出上。

经过后面的Block，经过平均池化和全连接，用softmax实现回归。

残差学习

避免梯度消失

在梯度更新的步骤，我们一般使用反向传播来计算损失函数对于权重的梯度，并加以更新。然而在深层的神经网络中，由于各种原因梯度在反向传播的过程中，信号会逐渐消失，这会导致顶层权重更新速度快，而底层权重几乎不更新的情况。这是灾难性的，因为底层所提取到的特征往往更为重要，例如边缘、纹理等等更为普遍的性质，从而使得训练效果不佳。除此之外，在一般的神经网络中，如果某一层停止学习，则信号就会在此消失，而无法传到更底层，从而造成“梯度消失”现象。

然而，在shortcut connection的存在下，梯度回传时，即使几层还没有开始学习（残差函数的梯度接近零），由于恒等函数的导数恒为1，整个函数的梯度依然接近于1，根据链式法则，先前的梯度依然可以反向传播，网络也可以开始取得进展。

残差函数训练更容易

ResNet基于这样一种假设：最优函数与线性函数有一定的相似性。初始化常规神经网络时，其权重参数接近零，因此网络仅输出其输入的副本。换言之，它首先对恒等函数建模。所以如果目标函数和恒等函数相当接近（通常是这种情况），那么训练速度会大大加快。这也是残差学习相对更加容易的原因。

Code

pre-train

Plain Text | 复制代码

```
resnet50 = models.resnet50(pretrained=True)

resnet50.fc = nn.Linear(2048, 12)

resnet50.conv1 = nn.Conv2d(1, 64,kernel_size=5, stride=2, padding=3, bias=False)

resnet50.train()
```

初始化一个预训练好的resnet50模型。要想直接利用该模型，首先要做的就是对于网络的神经元属性做出修改（也即所谓的模型微调）：

首先修改输出层，nn.Linear(2048, 12) 定义了一个新的全连接层，输入维度为 2048，输出维度为12，其中2048是ResNet50模型最后一个卷积层的特征维度，12是当前分类任务的类别数。

接下来改变网络中的所有卷积层：nn.Conv2d(1, 64, kernel_size=5, stride=2, padding=3, bias=False) 直接应用了上个CNN实验中定义了的卷积层，将自定义的卷积层赋值给ResNet50模型的conv1属性，从而替换掉原有的卷积层。

train

Plain Text | 复制代码

```
def train(model,train_loader, optimizer, epoch):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        output = model(data)
        loss = loss_func(output,target)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        if batch_idx % 10 == 0:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
            epoch, batch_idx * len(data), len(train_loader.dataset),
            100. * batch_idx / len(train_loader), loss.item()))
```

train函数用来训练，将模型调整至训练模式，每个batch进行一次反向传播。

test

Plain Text | 复制代码

```
def test(model, test_loader):
    model.eval()
    test_loss = 0
    correct = 0
    with torch.no_grad():
        for data, target in test_loader:
            output = model(data)
            test_loss += loss_func(output, target).item() # sum up batch loss
            pred = output.argmax(dim=1, keepdim=True) # get the index of the max log-probability
            correct += pred.eq(target.view_as(pred)).sum().item()
            test_loss /= len(test_loader.dataset)

    print('\nTest set: Average loss: {:.4f}, Accuracy: {}/{} ({:.0f}%)\n'.format(
    test_loss, correct, len(test_loader.dataset),
    100. * correct / len(test_loader.dataset)))
```

test函数用来测试模型精度，将模型调整到评估模式，累积每个batch内的实际输出与预期结果之差。

main

Plain Text | 复制代码

```
def main():
    transform=transforms.Compose([transforms.ToTensor(), transforms.Normalize((0.1307,), (0.3081,))])
    dataset1 = datasets.MNIST('../data', train=True, download=True, transform=transform)
    dataset2 = datasets.MNIST('../data', train=False, transform=transform)

    train_loader = torch.utils.data.DataLoader(dataset1,batch_size=64)
    test_loader = torch.utils.data.DataLoader(dataset2,batch_size=64)

    model = resnet50
    optimizer = optim.SGD(model.parameters(), lr=0.1)

    for epoch in range(2):
        train(model, train_loader, optimizer, epoch)
        test(model,test_loader)
        torch.save(model, "mnist_cnn_" + str(epoch) + ".pt")
```

main函数读入mnist数据集并划分为训练集和测试集，选取resnet50作为模型，SGD作为optimizer。需要注意的是在读入图片数据时使用了transform进行预处理：MNIST数据集的每个像素值

都在0-255之间，而ToTensor()方法将每个像素值除以255.0，使得每个像素值都在0到1之间。Normalize()方法用于将数据中心化并调整其标准差。其中，参数(0.1307,)和(0.3081,)分别表示每个像素的平均值和标准差。

根据实际训练情况，有时一轮训练下来精确度达不到96%，因此我们训练两轮，测试时load第二轮的模型。

实验结果

```
PS C:\Users\16367\Desktop\pj1.3> & C:/Users/16367/AppData/Local/Programs/Python/Python311/python.exe c:/Users/16367/Desktop/pj1.3/pretrain-test.py
Test set: Average loss: 0.000000, Accuracy: 9883/10000 (98.830000%)
```

测试精度达到了98.83%

实验思考

1. 在预训练模型上进行微调，可以防止过拟合，也使训练过程大大加快。
2. 残差神经网络的提出其实是源于传统CNN产生的问题，窃以为最难想到的点就是恒等网络。

Reference

1. https://blog.csdn.net/Cheungleilei/article/details/103610799
2. https://zhuanlan.zhihu.com/p/463935188