天津大学

模式识别与深度学习课程

作业 4、卷积神经网络实验报告

学　　院　智能与计算学部

专　　业　计算机科学与技术

学　　号　3019244266

姓　　名　李润泽

1．实验目标

实验的总目标是帮助我们熟练掌握卷积神经网络原理，基于深度学习主流框架 PyTorch，完成数据集读取、模型结构设计、模型训练、模型测试评估的代码编程。其中有两个目标分别是：

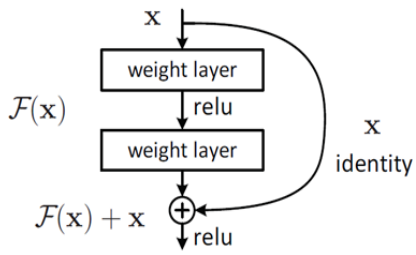
第一部分，熟练掌握 ResNet-18 的模型结构，了解残差网络的基本原理，并补全实验代码，从而完成 CIFAR-10 数据集的分类任务刘，具体地，需要实现包括 CIFAR-10 的数据预处理、ResNet-18 网络搭建、优化器设置、损失函数设置、模型训练以及模型测试的代码。

第二部分，在理解并掌握卷积神经网络原理的基础上，调节不同的参数，汇报不同参数对实验结果的影响，并尝试对结果进行分析，从而判断哪些参数对实验结果的影响比较大。此外，如果允许的话，可以分析使用 CPU 训练和 GPU 训练的实验差异。

2. 实验分析

2.1 算法实现

深度残差网络（ResNet）是 CNN 图像历史上一个里程碑事件，其作者 Kaiming He 因此荣获 CVPR2016 最佳论文奖。深度网络的退化问题表明深度网络并不容易训练，但考虑这样一个事实：现在你有一个浅层网络，你想通过向上堆积新层来建立深层网络，一个极端情况是这些增加的层什么也不学习，仅仅复制浅层网络的特征，即恒等映射(Identity Mapping)。在这种情况下，深层网络应该至少和浅层网络性能一样，也不应该出现退化现象。基于该假设，Kaiming He 提出了残差学习来解决退化问题。对于一个网络，将输入为 x 之后学习到的特征记为 F(x)，然后再加上一条分支，直接跳到输出记为 shortcut，则最终输出为 H(x) = F(x) + x，该结构记为 BasicBlock，如下图所示。

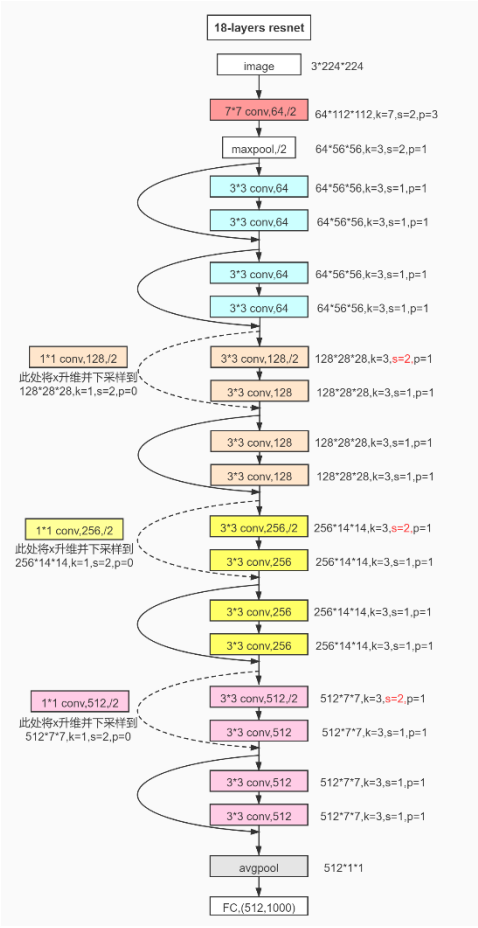


ResNet 的具体结构如下表所示：

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
		3×3 max pool, stride 2				
conv2_x	56×56	[3×3, 64; 3×3, 64] ×2	[3×3, 64; 3×3, 64] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3
conv3_x	28×28	[3×3, 128; 3×3, 128] ×2	[3×3, 128; 3×3, 128] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×8
conv4_x	14×14	[3×3, 256; 3×3, 256] ×2	[3×3, 256; 3×3, 256] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×23	[1×1, 256; 3×3, 256; 1×1, 1024] ×36
conv5_x	7×7	[3×3, 512; 3×3, 512] ×2	[3×3, 512; 3×3, 512] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Table 1. Architectures for ImageNet. Building blocks are shown in brackets (see also Fig. 5), with the numbers of blocks stacked. Down-sampling is performed by conv3_1, conv4_1, and conv5_1 with a stride of 2.

其中，对于 ResNet-18 网络，该网络由 5 个部分组成，中间每层都有 2 个残差块，而每个残差块都有两个卷积层，ResNet-18 的结构如下图所示：



其中，蓝色、橙色、黄色和粉色在论文中分别为 conv2, conv3, conv4, conv5。从 conv3 开始，第一个残差块的第一个卷积层的 stride 设定为 2，这是每层图片尺寸变化的原因。

首先实现残差块，残差块的实现代码如下：

```
# Residual block
# 补充代码时需注意：卷积核设置，bn 写法，shortcut 写法
class ResBlk(nn.Module):
    def __init__(self, ch_in, ch_out, stride=1):
        super(ResBlk, self).__init__()
        self.conv1 = nn.Conv2d(ch_in, ch_out, kernel_size=3,
stride=stride, padding=1)
        self.bn1 = nn.BatchNorm2d(ch_out)
        self.conv2 = nn.Conv2d(ch_out, ch_out, kernel_size=3, stride=1,
padding=1)
        self.bn2 = nn.BatchNorm2d(ch_out)
        if ch_out == ch_in:
            self.extra = nn.Sequential()
        else:
            self.extra = nn.Sequential(
                # 1*1 卷积和 bn
                nn.Conv2d(ch_in, ch_out, kernel_size=1, stride=stride),
                nn.BatchNorm2d(ch_out)
            )
    def forward(self, x):
        out = F.relu(self.bn1(self.conv1(x)))
        out = self.bn2(self.conv2(out))
        # print('shape of out     : {}'.format(out.shape))
        # print('shape of extra(x): {}'.format(self.extra(x).shape))
        out = self.extra(x) + out
        out = F.relu(out)
            return out
```

然后是 ResNet-18 网络的具体实现，这里需要注意的是，由于每层都有 2 个残差块，故我在进行代码实现的时候将其细分，因此 ResNet18 中会出现 8 个 block，实现代码如下：

```
# ResNet18
# 补充代码时需注意：block 构建
class ResNet18(nn.Module):
    def __init__(self):
        super(ResNet18, self).__init__()
        self.conv1 = nn.Sequential(
```

```
            # 第一层卷积和bn
            nn.Conv2d(3, 64, kernel_size=7, stride=2, padding=3),
            nn.BatchNorm2d(64),
            nn.MaxPool2d(kernel_size=3, stride=2, padding=1)
        )
        # 4 blocks
        # TODO: 这里应该有4个大block，但我将其细分
        self.blk1 = ResBlk(64, 64, stride=1)
        self.blk2 = ResBlk(64, 64, stride=1)
        self.blk3 = ResBlk(64, 128, stride=2)
        self.blk4 = ResBlk(128, 128, stride=1)
        self.blk5 = ResBlk(128, 256, stride=2)
        self.blk6 = ResBlk(256, 256, stride=1)
        self.blk7 = ResBlk(256, 512, stride=2)
        self.blk8 = ResBlk(512, 512, stride=1)
        # 全连接
        self.outlayer = nn.Linear(512*1*1, 10)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = self.blk1(x)
        x = self.blk2(x)
        x = self.blk3(x)
        x = self.blk4(x)
        x = self.blk5(x)
        x = self.blk6(x)
        x = self.blk7(x)
        x = self.blk8(x)
        x = F.adaptive_avg_pool2d(x, [1, 1])
        x = x.view(x.size(0), -1)
        x = self.outlayer(x)
            return x
```

在实验期间，我分别使用 GPU 和 CPU 进行实验，具体代码如下：

```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
# device = torch.device("cpu")
    model = ResNet18().to(device)
```

对于训练过程，我对损失函数和优化器分别进行了设置，其中，损失函数默认为交叉熵损失函数（nn.CrossEntropyLoss），优化器默认为 Adam 算法，学习率为 0.001，最终得到损失变化，实现代码如下：

```
def train(total_epoch):
    # 补充损失函数设置和优化器设置
    loss_fun = nn.CrossEntropyLoss()
    # m = nn.LogSoftmax(dim=1)
    # loss_fun = nn.NLLLoss()
    optimizer = optim.Adam(model.parameters(), lr=1e-3)
    # optimizer = optim.Adam(model.parameters(), lr=1e-4)
    # optimizer = optim.Adam(model.parameters(), lr=1e-2)
    # optimizer = optim.RMSprop(model.parameters(), lr=1e-3)
    # optimizer = optim.SGD(model.parameters(), lr=1e-3)
    loss_list = []
    for epoch in range(total_epoch):
        model.train()
        running_loss = 0.0
        # 补充训练部分代码，注意课件里的7步
        for batchidx, (inputs, labels) in enumerate(cifar_train):
            # 获取当前batch的数据，并挂GPU
            inputs, labels = inputs.to(device), labels.to(device)
            # 清空梯度
            optimizer.zero_grad()
            # 输入数据进模型
            # with torch.no_grad():
            outputs = model(inputs)
            # 计算模型损失
            loss = loss_fun(outputs, labels)
            # 自动求梯度
            loss.backward()
            # 优化模型参数
            optimizer.step()
            # 观察损失变化，下面5行代码不用改
            running_loss += loss.item()
            loss_list.append(loss.item())
            if (batchidx + 1) % 100 == 0:
                print('epoch = %d , batch = %d , loss = %.6f' % (epoch +
1, batchidx + 1, running_loss / 100))
                train_losses.append(loss.item())
                train_counter.append((batchidx * batch_size) + ((epoch-
1) * len(cifar_train.dataset)))
                running_loss = 0.0
            test(epoch)
```

对于测试过程，我们需要计算预测是否正确，从而计算出 accuracy，实现代码如下：

```
def test(epoch):
    # 模型调整为eval 模式，即测试
    model.eval()
    # loss_fun = nn.CrossEntropyLoss()
    with torch.no_grad():
        # test_loss = 0
        total_correct = 0
        total_num = 0

        # 补充测试部分代码
        for inputs, labels in cifar_test:
            # 获取当前batch 的数据
            inputs, labels = inputs.to(device), labels.to(device)
            # 训练数据输入模型
            outputs = model(inputs)
            # 计算损失
            # test_loss = loss_fun(outputs, labels)
            # 计算预测是否正确
            pred = outputs.argmax(dim=1)
            total_correct += torch.eq(pred, labels).float().sum().item()
            total_num += inputs.size(0)

        # 观察测试集精度变化，下面3 行代码不用改
        acc = total_correct / total_num
        test_counter.append(epoch+1)
        test_accuracy.append(acc)
        print("Accuracy of the network on the 10000 test
images:%.5f %%" % (100 * acc))
            print("================================================")
```

2.2 参数调节说明

在上文中，我设定以下参数及其变量：

迭代次数 epoch_num 设定为 10；

batch size 默认为 56，变量分别为 28、14；

损失函数默认为交叉熵损失函数（nn.CrossEntropyLoss），变量为负对数似然损失函数（nn.NLLLoss）；

优化器默认为 Adam 函数（optim.Adam），变量分别为随机梯度下降法（optim.SGD）与 RMSprop 优化算法（optim.RMSprop）；

其中，Adam 优化算法的学习率默认为 0.001，变量分别为 0.01 与 0.0001；

此外，我分别使用 GPU 和 CPU 进行实验，但由于使用 CPU 进行实验的运行时间过长，因此我在 CPU 的实验中仅进行了 2 次迭代。

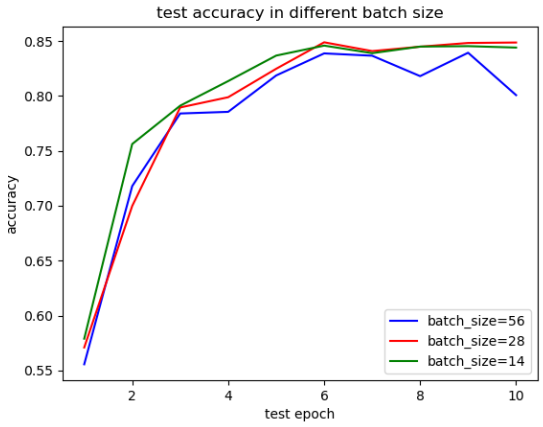
实验结果如下表所示（所有结果以及图表见“lab3 数据”文件夹）：

lab	f1	f2	f3	f4	f5	f6	f7	f8	f9
batch size	56						28	14	56
loss function	CrossEntropyLoss					NLLLoss	CrossEntropyLoss		
optimizer	Adam	SGD	RMSprop	Adam					
learning rate	0.001	0.0001	0.01	0.001					
CPU/GPU	GPU								CPU
epoch	acc								
1	55.57%	64.22%	42.97%	37.72%	50.39%	52.74%	57.10%	57.90%	57.32%
2	71.79%	70.73%	65.28%	50.36%	64.71%	70.97%	69.98%	75.62%	69.69%
3	78.40%	80.41%	70.72%	51.01%	75.37%	75.74%	78.95%	79.11%	
4	78.55%	76.69%	75.30%	56.15%	79.70%	77.05%	79.88%	81.35%	
5	81.86%	80.22%	80.06%	58.42%	81.66%	80.06%	82.47%	83.67%	
6	83.87%	81.06%	78.06%	49.96%	78.01%	82.88%	84.88%	84.58%	
7	83.66%	82.52%	80.46%	57.92%	82.22%	83.09%	84.08%	83.89%	
8	81.80%	82.66%	80.20%	67.25%	81.45%	83.94%	84.48%	84.49%	
9	83.93%	83.22%	81.11%	69.11%	83.45%	83.79%	84.82%	84.53%	
10	80.07%	73.24%	82.13%	65.95%	83.35%	84.28%	84.86%	84.40%	

我将对上表的不同参数分别进行分析：

2.2.1 不同 Batch Size 分析

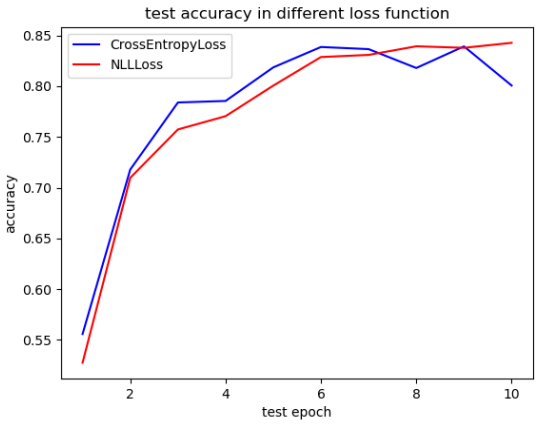
lab	f1	f7	f8
batch size	56	28	14
loss function	CrossEntropyLoss		
optimizer	Adam		
learning rate	1.00E-03		
CPU/GPU	GPU		
epoch	test accuracy		
1	55.57%	57.10%	57.90%
2	71.79%	69.98%	75.62%
3	78.40%	78.95%	79.11%
4	78.55%	79.88%	81.35%
5	81.86%	82.47%	83.67%
6	83.87%	84.88%	84.58%
7	83.66%	84.08%	83.89%
8	81.80%	84.48%	84.49%
9	83.93%	84.82%	84.53%
10	80.07%	84.86%	84.40%



对比图表如上所示，可以看出，batch_size 对测试准确率影响不大，但更小的 batch_size 可以略微提高测试的准确率。

2.2.2 不同损失函数分析

lab	f1	f6
batch size	56	
loss function	CrossEntropyLoss	NLLLoss
optimizer	Adam	
learning rate	1.00E-03	
CPU/GPU	GPU	
epoch	test accuracy	
1	55.57%	52.74%
2	71.79%	70.97%
3	78.40%	75.74%
4	78.55%	77.05%
5	81.86%	80.06%
6	83.87%	82.88%
7	83.66%	83.09%
8	81.80%	83.94%

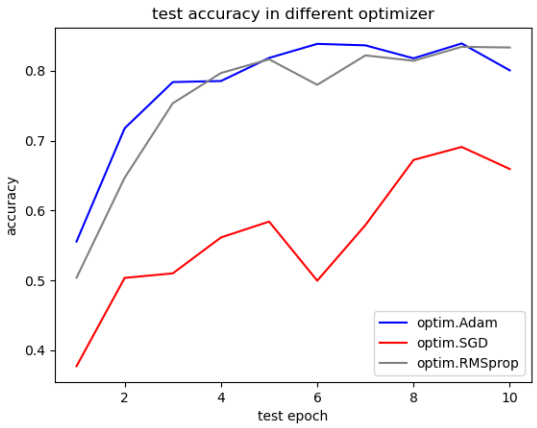


9	83.93%	83.79%
10	80.07%	84.28%

对比图表如上所示，可以看出，损失函数对测试准确率影响不大。

2.2.3 不同优化器分析

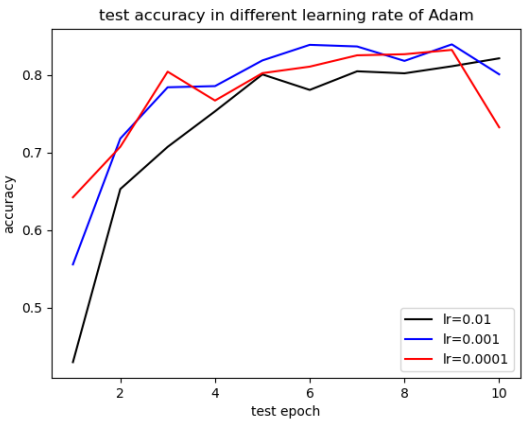
lab	f1	f4	f5
batch size	56		
loss function	CrossEntropyLoss		
optimizer	Adam	SGD	RMSprop
learning rate	1.00E-03		
CPU/GPU	GPU		
epoch	test accuracy		
1	55.57%	37.72%	50.39%
2	71.79%	50.36%	64.71%
3	78.40%	51.01%	75.37%
4	78.55%	56.15%	79.70%
5	81.86%	58.42%	81.66%
6	83.87%	49.96%	78.01%
7	83.66%	57.92%	82.22%
8	81.80%	67.25%	81.45%
9	83.93%	69.11%	83.45%
10	80.07%	65.95%	83.35%



对比图表如上所示，可以看出，随机梯度下降法（SGD）对测试准确率影响较大，准确率最低，而 Adam 算法与 RMSprop 算法的测试准确率大致相同。

2.2.4 不同 Adam 优化函数的学习率分析

lab	f1	f2	f3
batch size	56		
loss function	CrossEntropyLoss		
optimizer	Adam		
learning rate	1.00E-03	1.00E-04	1.00E-02
CPU/GPU	GPU		
epoch	test accuracy		
1	55.57%	64.22%	42.97%
2	71.79%	70.73%	65.28%
3	78.40%	80.41%	70.72%
4	78.55%	76.69%	75.30%



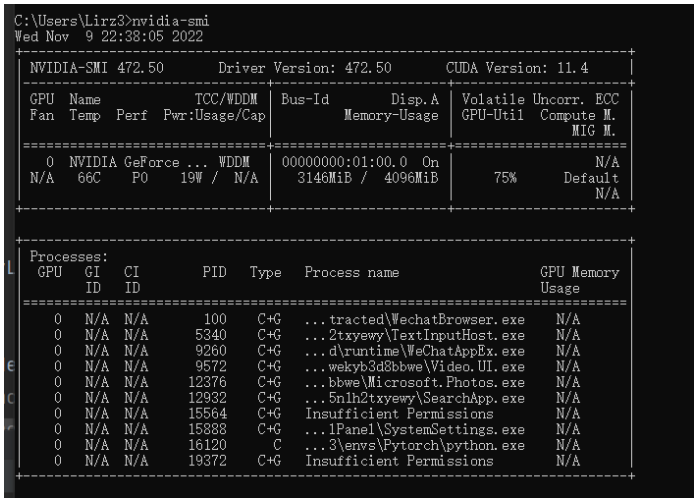
5	81.86%	80.22%	80.06%
6	83.87%	81.06%	78.06%
7	83.66%	82.52%	80.46%
8	81.80%	82.66%	80.20%
9	83.93%	83.22%	81.11%
10	80.07%	73.24%	82.13%

对比图表如上所示，可以看出，测试准确率随着学习率的上升而降低。

2.2.5 CPU/GPU 分析

lab	**f1**	**f9**
batch size	56	
loss function	CrossEntropyLoss	
optimizer	Adam	
learning rate	1.00E-03	
CPU/GPU	GPU	CPU
epoch	**test accuracy**	
1	55.57%	57.32%
2	71.79%	69.69%
3	78.40%	跑的时间太长，一晚上没跑出来
4	78.55%	
5	81.86%	
6	83.87%	
7	83.66%	
8	81.80%	
9	83.93%	
10	80.07%	

使用 GPU 进行实验可以在命令行输入 nvidia-smi 进行查看，如下图所示：



从测试过程中可以看出，CPU 在测试过程中的测试速度过慢，而 CUDA 加速能力较强，大约 30min 即可得出结果。

2.3 结果分析

从上述的表格和图像可以得出以下结论，batch_size 对测试准确率影响不大，但更小的 batch_size 可以略微提高测试的准确率；损失函数对测试准确率影响不大；随机梯度下降法（SGD）对测试准确率影响较大，准确率最低，而 Adam 算法与 RMSprop 算法的测试准确率大致相同；测试准确率随着学习率的上升而降低；而 CPU 在测试过程中的测试速度过慢，而 CUDA 加速能力较强，大约 30min 即可得出结果。

最终我们可以选取出最佳的参数配置：

```
batch size = 56;
loss_fun = nn.CrossEntropyLoss();
optimizer = optim.Adam(model.parameters(), lr=1e-3);
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu");
```

2.4 遇见的问题

在实验过程中，我进行环境的配置时出现了问题，我进行实验的时候发现实验始终无法出现结果，我认为可能是没有调用 GPU 所致，因此我使用 print(torch.cuda_is_available())进行测试，输出了 False，因此我明白该实验可以运行但因为使用的是 CPU 故运行极慢，在重新进行环境配置后得以快速运行。

3. 总结

实验在 PyCharm 中使用了 Pytorch 库运行，ResNet-18 模型应用到交叉熵损失函数、Adam 优化函数以及卷积神经网络等函数。在实验的过程中，我深入学习了 ResNet-18 模型并对 CIFAR-10 数据集成功进行了分类任务。实验的测试正确率可以得到 80%，但我发现随着迭代次数的增加，测试准确率反而降低，因此我明白测试的迭代次数过多并不一定能提高测试准确率。

通过这次实验，我对深度学习以及卷积神经网络有了更多的理解。在我对 CNN 模型对 MNIST 数据集进行分类任务的代码进行了理解分析之后，成功依此

完成 CIFAR-10 分类任务，从而提升了我的代码能力。针对老师在实验指导书上的各种问题我也进行了分析，最终顺利完成了本次实验，对于我在学习模式识别与深度学习的过程中会有更大的帮助。