哈尔滨工业大学

**<<模式识别与深度学习>>**

**实验二 实验报告**

**(2020春季学期)**

|  |  |
| --- | --- |
| **成员1：** | **1170300513 陈鋆** |
| **成员2：** |  |
| **成员3：** |  |

-

# 一、实验环境

* 操作系统：win10操作系统
* IDE：Pyrcharm Professional
* Python 运行环境：Anaconda 创建的虚拟环境 deeplearning
* 环境中必要的 Python 库：torch1.3.0、TensorboardX2.0.0、tensorboard2.0.0
* CIFAR-10 数据集存放目录如图1-1所示。

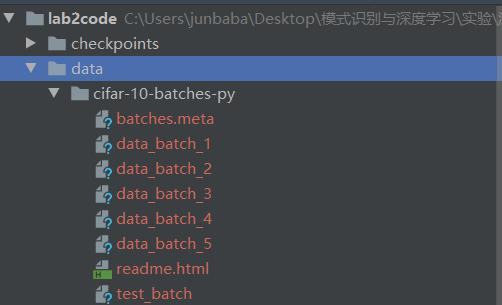


图 1-1 CIFAR-10存放目录

# 二、实验内容

* 基于PyTorch实现AlexNet结构。
* 在Cifar-10数据集上进行验证。
* 使用tensorboard进行训练数据可视化。

2.1 AlexNet

2012 年，Alex等人提出的 AlexNet网络在ImageNet大赛上以远超第二名的成绩夺冠，卷积神经网络乃至深度学习重新引起了广泛的关注。

AlexNet 是在LeNet的基础上加深了网络的结构[1]，学习更丰富更高维的图像特征。AlexNet 的特点：更深的网络结构；使用层叠的卷积层,即卷积层+卷积层+池化层来提取图像的特征；使用Dropout抑制过拟合；使用Relu替换之前的sigmoid的作为激活函数。



图2 - 1 AlexNet网络结构图

2.2 CIFAR-10

CIFAR-10数据集由10个类的60000个32x32彩色图像组成[2]，每个类有6000个图像。数据集共有5万张训练图片和1万张测试图片。

数据集分为五个训练批和一个测试批，每个测试批有10000个图像。测试批包含从每个类中随机选择的1000个图像。训练批包含随机顺序的剩余图像，但 某些训练批可能包含一个类的图像多于另一个类的图像。其中，训练批次包含每个类的5000幅图像。

图2-2中展示了数据集中的类，以及来自每个类的10个随机图像：

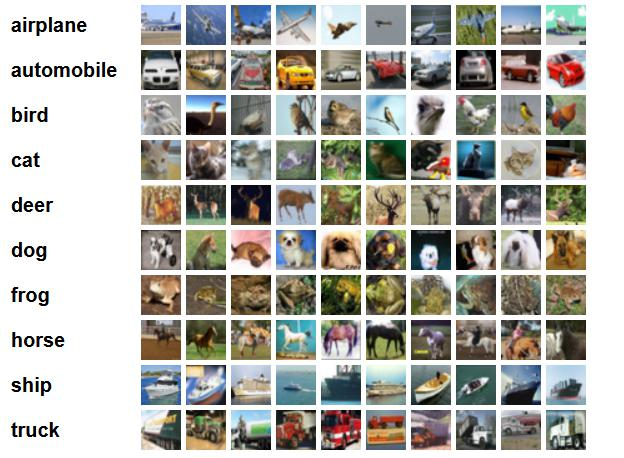


图2 - 2 CIFAR-10类别图

这些类是完全互斥的。如：汽车和卡车之间没有重叠。“汽车”包括轿车，越野车，等等。“卡车”只包括大卡车，并不包括皮卡。

CIFAR-10官网上下载的python版本解压后共有八个包（如图1-1所示）。

2.3 tensorboard

tensorboard是 tensorflow内置的一个可视化工具，它通过将tensorflow程序输出的日志文件的信息可视化使得tensorflow程序的理解、调试和优化更加简单高效。tensorboard的可视化依赖于tensorflow程序运行输出的日志文件，因而tensorboard和tensorflow程序在不同的进程中运行。

而在本次实验中，我们通过TensorboardX来代替tensorflow输出日志文件， 从而实现Pytorch的可视化。

# 三、实验代码实现

本次实验代码主要由四个结构构成：用于训练的train.py、用于测试的test.py、用于构造AlexNet 模型的Alexnet.py以及用于构造自己的CIFAR-10数据集的MyCIFAR10.py。程序执行的主体思路是：首先由train.py对构造的AlexNet模型在训练集上进行训练并保存相关的checkpoint，再由test.py读取checkpoint并在测试集上测试手写数字识别的准确率。

3.1 Alexnet.py

用于构造AlexNet模型的 Alexnet.py，其中Alexnet参照论文完成后做出了如下改变：

1. 在Conv2d层中就卷积核大小、步长、填充长度进行了改变。
2. 在池化层中改变了卷积核大小，避免后续图片过小。
3. 删除了avgpool。

|  |
| --- |
| 1. **def** \_\_init\_\_(self, num\_classes=10): 2. super(Alexnet, self).\_\_init\_\_() 3. # 卷积层 4. self.features = nn.Sequential( 5. nn.Conv2d(3, 64, kernel\_size=3, stride=1, padding=1),  # 缩小卷积核，步长、填充 6. nn.ReLU(inplace=True),  # inplace=True，覆盖操作，节省空间 7. nn.MaxPool2d(kernel\_size=2),  # 32 -> 16 8. nn.Conv2d(64, 192, kernel\_size=3, stride=1, padding=1), 9. nn.ReLU(inplace=True), 10. nn.MaxPool2d(kernel\_size=2),  # 16 -> 8 11. nn.Conv2d(192, 384, kernel\_size=3, stride=1, padding=1), 12. nn.ReLU(inplace=True), 13. nn.Conv2d(384, 256, kernel\_size=3, stride=1, padding=1), 14. nn.ReLU(inplace=True), 15. nn.Conv2d(256, 256, kernel\_size=3, stride=1, padding=1), 16. nn.ReLU(inplace=True), 17. nn.MaxPool2d(kernel\_size=2),  # 8 -> 4 18. ) 19. # self.avgpool = nn.AdaptiveAvgPool2d((4, 4))  # 8 -> 4 20. # 全连接层 21. self.classifier = nn.Sequential( 22. nn.Dropout(), 23. nn.Linear(256 \* 4 \* 4, 4096), 24. nn.ReLU(inplace=True), 25. nn.Dropout(), 26. nn.Linear(4096, 4096), 27. nn.ReLU(inplace=True), 28. nn.Linear(4096, num\_classes), 29. ) 31. **def** forward(self, x): 32. out = self.features(x) 33. out = self.avgpool(out) 34. out = torch.flatten(out, 1)  # 进入全连接层之前展开为一维 35. out = self.classifier(out) 36. **return** out |

代码块3 - 1

3.2 MyCIFAR10.py

torch.utils.data.DataLoader不仅生成迭代数据非常方便，而且它也是经过优化的，效率十分之高（肯定比我们自己写一个要高多了），因此最好不要舍弃。因此，我的想法是根据CIFAR-10数据集构造一个Dataset的子类，使之能够作为torch.utils.data.DataLoader的参数，从而使数据集能被我们用于生成迭代数据进行训练与测试。

要构造Dataset的子类，就必须要实现两个方法：

1. \_getitem\_(self, index)：根据 index 来返回数据集中标号为 index 的元素及其标签。
2. *\_len\_*(self)：返回数据集的长度。

最后，代码时在init初始化之时读取数据集，由于图像要作为数据要送入网络中，因此最后需要将其累加值从 numpy数组转为Tensor。

|  |
| --- |
| 1. **class** MyCIFAR10(Dataset): 2. """ 3. 根据CIFAR-10定义的个人数据集类 4. 继承自Dataset类，因此能够被torch.utils.data.DataLoader使用，从而更高效地在训练和测试中迭代 5. """ 7. **def** \_\_init\_\_(self, root, train=True, transform=None, target\_transform=None): 8. super(MyCIFAR10, self).\_\_init\_\_() 9. self.transform = transform 10. self.target\_transform = target\_transform 11. self.imgs = None 12. self.labels = [] 14. # 根据CIFAR-10官网上下载的数据，训练集分为5个batch文件，每个里有10000张32\*32的图片；测试集只有1个batch文件，里面有10000张32\*32的图片 15. train\_lists = ['data\_batch\_1', 16. 'data\_batch\_2', 17. 'data\_batch\_3', 18. 'data\_batch\_4', 19. 'data\_batch\_5'] 20. test\_lists = ['test\_batch'] 22. # 根据train是否为True来选择测试集或训练集 23. **if** train: 24. lists = train\_lists 25. **else**: 26. lists = test\_lists 28. # 读取数据集，构造类中的图像集和标签 29. **for** list **in** lists: 30. filename = os.path.join(root, list) 31. with open(filename, 'rb') as f:  # 这里需要'rb' + 'latin1'才能读取 32. datadict = pickle.load(f, encoding='latin1') 33. X = datadict['data'].reshape(-1, 3, 32, 32) 34. Y = datadict['labels'] 35. **if** self.imgs **is** None: 36. self.imgs = np.vstack(X).reshape(-1, 3, 32, 32) 37. **else**: 38. self.imgs = np.vstack((self.imgs, X)).reshape(-1, 3, 32, 32) 39. self.labels = self.labels + Y 40. self.imgs = torch.from\_numpy(self.imgs).type(torch.FloatTensor) 42. # 继承的Dataset类需要实现两个方法之一：\_\_getitem\_\_(self, index) 43. **def** \_\_getitem\_\_(self, index): 44. img, label = self.imgs[index], self.labels[index] 46. **if** self.transform **is** **not** None: 47. img = self.transform(img) 49. **if** self.target\_transform **is** **not** None: 50. label = self.target\_transform(label) 52. **return** img, label 54. # 继承的Dataset类需要实现两个方法之一：\_\_len\_\_(self) 55. **def** \_\_len\_\_(self): 56. **return** len(self.imgs) |

代码块3-2

3.3 train.py

训练部分的代码和实验1训练部分的代码基本一致，并且在代码中注释详尽，这里就不赘述了。这里主要展示一下train.py中相对之前的代码改进的部分。

首先，如代码块3-3所示，定义一个解析命令行参数的函数，使得我们能够通过命令行输入一些训练时需要的关键常量，便于我们进行调参。

其次，如代码块3-4所示，构造训练集是用上了自己定义MyCIFRA10类，并将其作为Dataloader中dataset参数，而使数据集能被我们用于生成迭代数据进行训练与测试。

最后，如代码块3-5所示。在每一个epoch中计算一次平均loss，并进行可视化绘画；最后添加了异常，使得按下Ctrl+C打断训练后能保存模型。

|  |
| --- |
| 1. **def** get\_args(): 2. """ 3. 解析命令行参数 4. 返回参数列表 5. """ 6. parser = OptionParser() 7. parser.add\_option('-e', '--epochs', dest='epochs', default=20, type='int', 8. help='number of epochs') 9. parser.add\_option('-b', '--batch\_size', dest='batchsize', default=50, 10. type='int', help='batch size') 11. parser.add\_option('-l', '--lr', dest='lr', default=3e-4, 12. type='float', help='learning rate') 13. (options, args) = parser.parse\_args() 14. **return** options |

代码块3-3

|  |
| --- |
| 1. # 构造训练集 2. cifar10 = MyCIFAR10.MyCIFAR10('./data/cifar-10-batches-py', train=True) 3. train\_loader = torch.utils.data.DataLoader(dataset=cifar10, batch\_size=batch\_size, shuffle=True) |

代码块3-4

|  |
| --- |
| 1. # 每个epoch计算一次平均Loss 2. **print**('Epoch [{}/{}], Loss: {:.4f}'.format(epoch + 1, epochs, trainloss / len(train\_loader))) 3. # write to tensorboard 4. writer.add\_scalar('scalar/TrainLoss', trainloss/len(train\_loader), epoch, walltime=epoch) 5. writer.close() 6. # ctrl + C 可停止训练并保存 7. **except** KeyboardInterrupt: 8. **print**("Save.....") 9. torch.save(model.state\_dict(), os.path.join('./checkpoints', 'Interrupt.ckpt')) 10. exit(0) |

代码块3-5

3.4 test.py

测试部分的代码和实验1测试部分的代码基本一致，并且在代码中注释详尽，这里也不赘述了。

# 四、实验结果与分析

4.1 lr=1e-3

首先，我将学习率lr定为1e-3，训练 20 个epoch 后，发现效果并不好，正确率仅仅只有70％。于是我增大迭代次数epochs，观察可视化绘制的loss函数，如图4-1所示。

可以看到，随着epochs的增大，loss先是下降，而后迅速上升。这可能是由于学习率过大，导致代价函数在优化的过程中越过了最低点并逐渐上升，如图4-2所示。为了进一步验证这个猜想，下面我将调低学习率再进行训练。

|  |  |
| --- | --- |
|  |  |
| 图4-1 lr=1e-3时训练的loss曲线 | 图4-2 学习率过高时的情况 |

4.2 lr=1e-4

将学习率lr定为1e-3，再次进行训练，并绘制了loss函数如图4-3所示。过对比可以发现，lr=1e-4时确实随着训练次数增加，Loss曲线逐渐减少并最终趋于平缓。这也证实了前面学习率 过大导致跳过了低点的猜想。

进行测试后得到准确率为 80％左右。

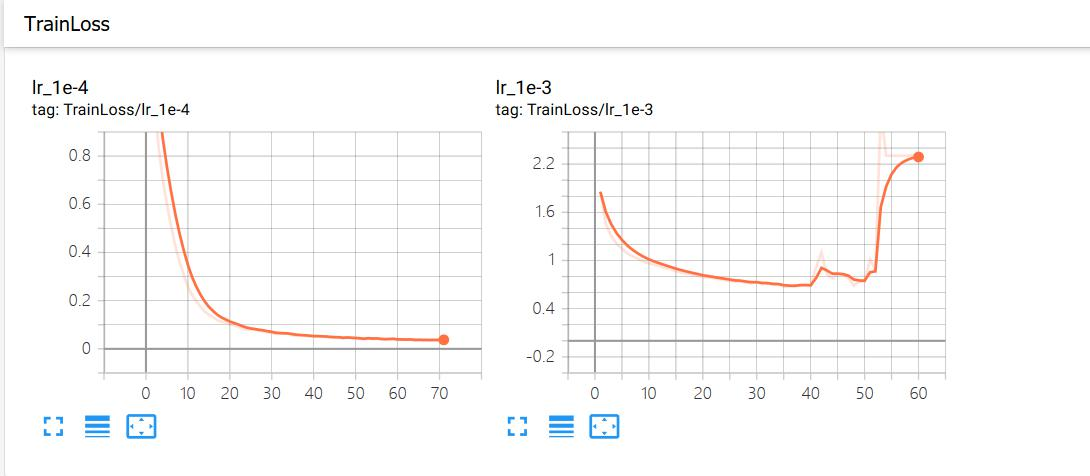


图4-3 lr=1e-3与lr=1e-4的Loss曲线对比图

4.3 推迟Pooling

前面的两个实验从调整学习率的角度来提高AlexNet的识别率。并且很显然的是，继续在学习率方面继续调整以降低损失，从而获得更高的准确率是对我们来说较为困难。我认为目前限制准确率主要因素是网络结构。

根据老师上课中提到的对 AlexNet 的改进思路有几个方向如：减小卷积核、推迟Pooling。我决定采用推迟Pooling的方式，观察其是否能继续提高性能。

在代码上的主要改变是调整了Alexnet.py中的网络结构，删除了第一层池化层，并恢复了卷积层与全连接层之间的avgpooling。另外，在原有的代码基础上，我还另外增加了绘制测试时的Loss曲线以及测试准确率曲线的内容。

实验的结果如图4-4与图4-5所示。图4-4为训练与测试的Loss曲线，其中蓝色曲线是训练损失，而红色线条是测试损失。可以看到，最终准确率趋于平稳，达到了86%左右，确实相较于之前有较大提升。另外，曲线的后半段，当测试损失上升时，测试的准确率反而上升。经过查阅网上资料，我认为可能是多元分类问题的损失函数与正确率不一致所导致的，模型可能学会了更多的分类信息，但却使得损失增大。

|  |  |
| --- | --- |
|  |  |
| 图4-4 训练与测试的Loss曲线 | 图4-5 测试的准确率曲线 |

4.4 添加BN层

在上述实验结果的基础上，继续探究网络结构的改进对识别准确率的影响。

Batch Normalization（BN）是由Google于2015年提出，这是一个深度神经网络训练的技巧。它是对于训练中某一个batch的数据（这个数据是可以输入也可以是网络中间的某一层输出）进行归一化。我在原有的网络结构上添加上BN层，观察其是否进一步提高性能。

在代码上的主要变化是继续调整了Alexnet.py中的网络结构，在每个卷积层后添加了BatchNorm2d层。

实验的结果如图4-6与4-7所示。图4-6为训练与测试的Loss曲线，其中蓝色曲线是训练损失，而红色线条是测试损失。可以看到，最终准确率趋于平稳，最高达到了89%左右，相较于之前有了进一步的提升。添加BN层确实对识别的准确率提升有所帮助。这可能是BN重新改变了优化问题，使得优化空间变得非常平滑[3]。

|  |  |
| --- | --- |
|  |  |
| 图4-6 训练与测试的Loss曲线 | 图4-7 测试的准确率曲线 |

# 五、心得体会

这次实验使我收获颇多，主要有两点收获：首先，因为要自己写一个dataset类，为了学习torch中对其的构建，我 不得不去尝试着查看源码了解其情况。这使我戒掉了对库的了解严重依赖于搜索引擎的坏习惯，养成了ctrl+B查看源码实现的好习惯；其次，这次实验也使我更了解了CNN、BN的一些性质，并且初步掌握了构造CNN 以及在训练中对模型进行调参的方法。

# 六、相关引用

1. Krizhevsky A, Sutskever I, Hinton G. ImageNet Classification with Deep Convolutional Neural Networks[J]. Advances in neural information processing systems, 2012, 25(2).
2. CIFAR-10 and CIFAR-100 datasets.[EB/OL].http://www.cs.toronto.edu/~kriz/cifar.html.
3. autocyz.为什么Batch Normalization那么有用？.[EB/OL].<https://zhuanlan.zhihu.com/p/52749286,2018.12.29.>