利用自主构建的神经网络框架分类 CIFAR-10 数据集的实践

摘要与说明

此篇研究报告是基于为期约三周的神经网络学习写就。按照老师的要求，在整个实践过程中,我使用 PyCharm 编辑器,在只导入 numpy 库的情况下自主构建神经网络框架，并尽可能准确地完成对 CIFAR-10 数据集中测试集的分类。

实践过程中，我主要参考了 cs231n 作业中自带的部分代码，以及知乎用户 WILL（华中科技大学）的学习笔记，任何参考了他们代码的地方我都会在报告中指出。

在此篇研究报告中，我会依次介绍：我所构建的神经网络库的基本结构、利用库中保存的网络结构（包括全连接网络及卷积神经网络（附带 BN 层））分类 CIFAR-10 数据集的详细过程，以及利用库中函数自由设计神经网络结构并完成训练及分类的过程。

自主构建的神经网络库(NeuralNetwork)介绍

1.1 NeuralNetwork 子目录简介

库名称	库内主要文件	文件简介
NeuralNetwork	datasets	文件夹，用于储存 CIFAR-10 数据集
	data_utils.py	Python 文件，用于数据的读取与预处理
	Layers.py	Python 文件，用于各个网络层前向传播与后向传播的实现
	LossFunctions.py	Python 文件，包括了针对分类问题的损失函数
	NetworkStructure.py	Python 文件，包括了若干已经编写好的表现较为良好的神经网络结构
	fast_layers.py im2col.py im2col_cython.c im2col_cython.pyx setup.py	这些文件是从 cs231n assignment2 文件夹中复制而来，主要用途是提高卷积运算中前向传播及反向传播的速度

1.2 部分关键 python 文件的详细介绍

文件名称	函数名称	主要作用
LossFunctions.py	softmax_loss()	依据 softmax 分类器计算损失函数并求梯度
	svm_loss()	依据 svm 计算损失函数并求梯度

文件名称	函数名称	主要作用
Layers.py	affine_forward() affine_backward()	分别用于全连接层的线性映射的前向传播与反向传播
	relu_forward() relu_backward()	分别用于全连接层 ReLU 激活函数的前向传播与反向传播
	affine_relu_forward() affine_relu_backward()	结合了全连接层的线性映射以及 ReLU 激活函数，形成一个常见的全连接层
	conv_forward_naive() conv_backward_naive() max_pool_forward_naive() max_pool_backward_naive()	利用 for 循环的形式完成卷积层的映射，实现前向与反向传播。受限于计算速度，这四个函数实际并未使用
	conv_relu_pool_forward() conv_relu_pool_backward()	整合了卷积、激活、池化运算，形成了完整的卷积层，这里使用了 cs231n 提供的快速计算函数
	batchnorm_forward() batchnorm_backward() batchnorm_backward_alt()	实现了 BN 层的前向传播与反向传播
	affine_bn_relu_forward() affine_bn_relu_backward()	结合了 BN 层的全连接层传播
	spatial_batchnorm_forward() spatial_batchnorm_forward()	结合了 BN 层的卷积层传播

注：Layers.py 与 LossFunction.py 的代码主要参照了知乎 WILL 的学习笔记。尽管参考了他的代码，但其中每一行代码都在领会了意图的情况下写入，只有部分代码（如 BN 层的反向传播代码）尚未理解原理，做了机械的搬运工作。另外，由于采取循环写成的卷积层传播在实际训练过程中耗费了太多时间，因此利用了 cs231n 课程中给出的 fast_layer.py 的算法，这一部分我直接使用了，没有查看源码。

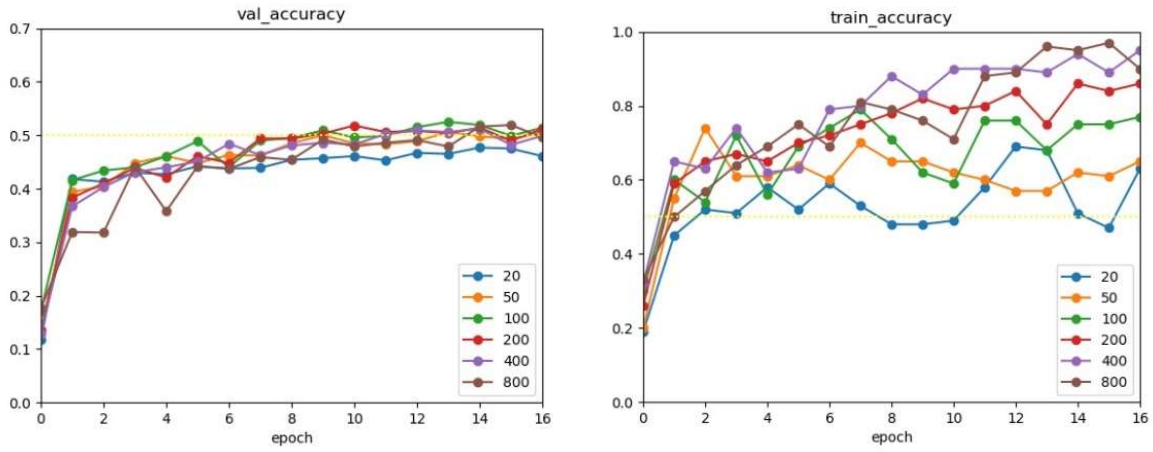
1.3 以多层全连接网络为例介绍网络结构 (NetworkStructure.py)

网络类型（类名）	函数名称	实现流程
FullyConnectedNet	__init__()	1. 初始化网络参数 包括：是否使用 BN 层、dropout、网络层数以及每一层的参数 W，b，gamma，beta 等
	loss()	2. 输入训练数据进行前向传播 在多层全连接网络中，利用 for 循环遍历每一层，将每一层的输出交给下一层，并缓存当前层的相关参数供反向传播调用 3. 在最后一层中输出 scores，如果处于训练状态，则将 scores 和标签集 Y 一起传给 softmax_loss()函数计算损失并返回分数梯度 dscores；如果处于预测状态则直接返回得分 scores 4. 训练状态中，将返回的 scores 经由反向传播函数重复类似前向传播的循环过程，并得到 dw, dx, db 等相关参数的梯度（注意：这里的每一次循环都要正则化） 5. 反向传播完成后，封装所有参数梯度，与损失值 loss 一起返回
	train()	6. train()函数将输入的训练集根据 batch_size 划成更小的数据集，并将小数据集放进 loss()函数中求出损失与梯度 7. 用给出的梯度更新参数（结合正则化） 8. 不断循环步骤 6、7，直到趋向于收敛 9. 返回 loss_history, train_acc_history 用于进一步分析
	predict()	10. 利用已经建立好的模型进行预测，同样调用 loss()函数，但不输入 y 值，这样 loss 函数将会给出 scores 矩阵，将矩阵每一样本得分最高的类作为标签并与真实标签对照得到验证集准确率，预测完成。

利用库内模型对 CIFAR-10 进行训练与分类

2.1 多层全连接网络模型（此模型以 WILL 的笔记为原型，删去了其中的 solver 类，将其简化、修改并入了网络结构之中）

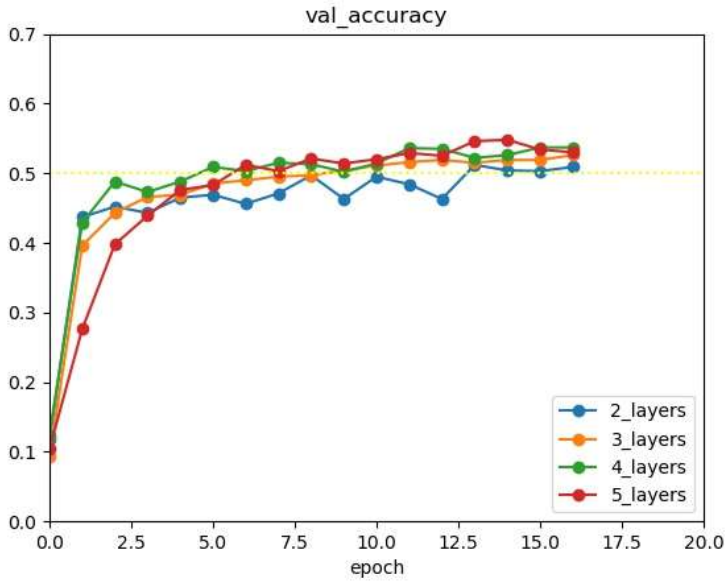
2.1.1 隐藏节点数量对准确率的影响



图表 1：两层全连接网络中隐藏层节点数量对准确率的影响

根据图表，一个两层的全连接网络中，如果节点数过少（少于 50），会出现欠拟合现象（20 节点的验证集准确率在 45%左右）、如果节点数过多，则训练集准确率会逐渐逼近 100%，但验证集准确率并未提高，即产生过拟合现象。

2.1.2 隐藏层数量对准确率的影响



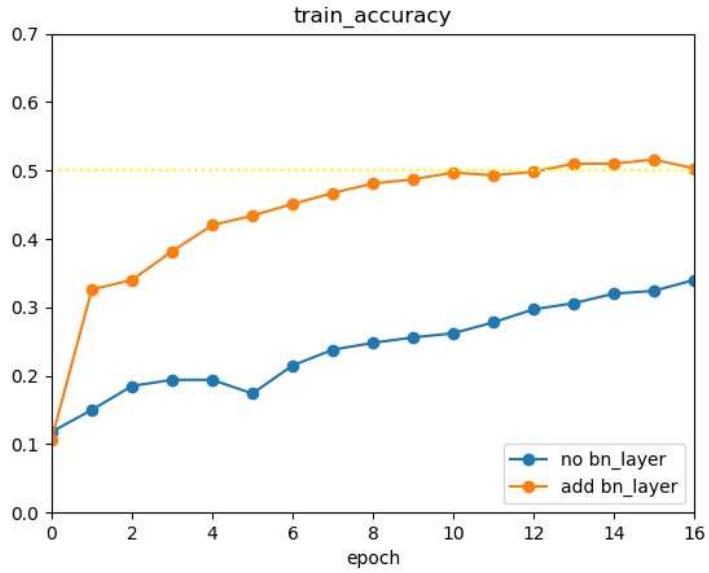
图表 2：全连接网络层数对准确度的影响

从图表中可以看出，多层的全连接网络相较于两层的网络取得了一定的进步，准确率的提高并不明显。尤其是在 3 层网络之后，继续增加网络层数对准确率的提高越来越有限。

注：图表中的多层网络准确率的上升趋势理论上应更平缓一点，由于网络训练速度比较慢，因此我提高了多层时网络的学习率使得多层网络也可以较快地收敛，这一部分主要代码如下：

```
for i in range(4):
    print('here is in No.%d iterations' %(i))
    clf = FullyConnectedNet(multiLayers['%d_layers' %(i+2)],
weight_scale=1e-2)
    stats = clf.train(x_train, y_train, x_val, y_val, num_iters=8000,
batch_size=100,
                      learning_rate=1e-3 if i<2 else 7e-3,
verbose=True)
    val_acc_data.append(stats['val_acc_history'])
```

2.1.3 BN 层对神经网络的影响

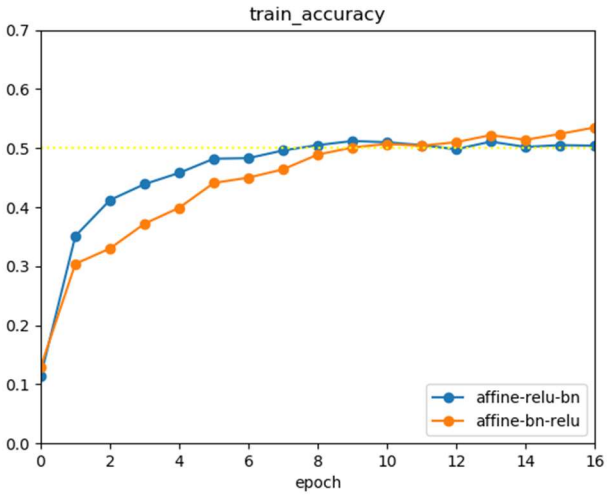


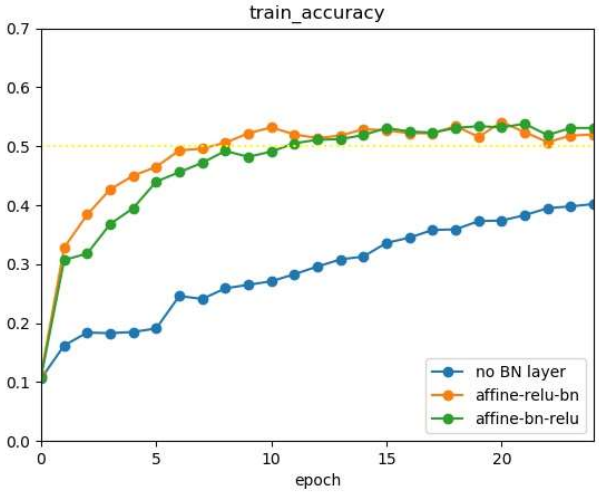
图表 3：BN 层对网络训练的影响

当全连接网络为 5 层时，在相同的学习率（1e-3）情况下，添加了 bn 层的网络收敛速度有非常明显的加快。与此同时，我也测试了相同学习率下双层网络 bn 层的效果，结果显示在浅层网络中 bn 层对结果的提升并不明显。BN 层的加入使得深度网络的收敛速度明显加快，这为模型的训练缩短了很多时间（尤其是在不使用任何库的情况下，训练一次节省了非常多的时间）。

2.1.4 BN 层位置是否会对训练效果造成影响？

根据吴恩达深度学习视频，科研人员对 BN 层放在激活函数前面还是后面曾有过争论。而 cs231n 中给出的函数是默认将 BN 层放在激活函数前面的。为了核实这个问题，我做了两组对照试验





根据结果，无论 BN 层放在哪里，实际的收敛速度都远高于不使用 BN 层的速度。当 BN 层放在激活函数后面，数据前期的拟合速度似乎稍快一点，可以率先达到比较高的水平。但随着训练的进行，BN 层放在激活函数前面的表现要略好于放在后面。
实际上在实际测试过程中，完全可以使用 affine-relu-bn 的模式使模型较快达到一个理想水平。确定理想模型后再使用 affine-bn-relu 让模型稳定的到达最佳状态。

2.2 利用卷积神经网络进行训练

我先完全按照 LeNet 网络结构构造模型，发现无论是训练集还是验证集，准确率都很低（验证集约 45%），甚至不如之前的全连接网络。

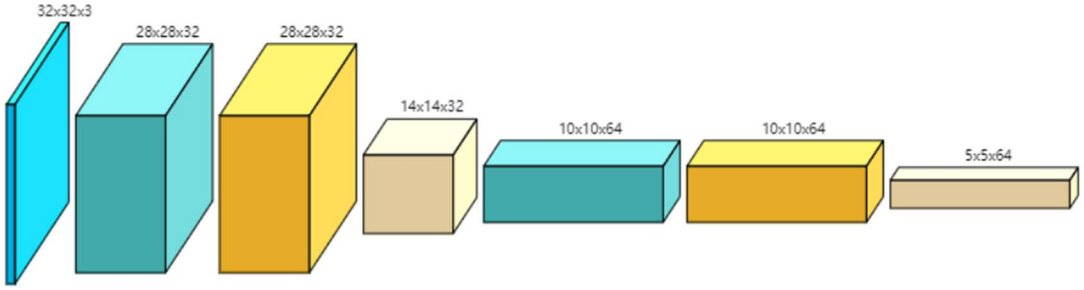
紧接着我增加了卷积网络的通道数量，将 LeNet 中两个卷积层的通道(6, 16)调整为(16, 32)，但在训练过程中准确率反而越来越低，甚至小于 10%。经过一些摸索，我通过调整 W 参数的初始权重解决了此问题，模型准确率有所提高，可以超越之前的全连接网络，达到 65%左右。

为了进一步提高训练准确性，我继续提高卷积层的通道数量，相应的也增加了全连接网络中每个隐层的神经元数量，这一次，经过 80 分钟的训练，模型测试集准确率可以达到接近 70%

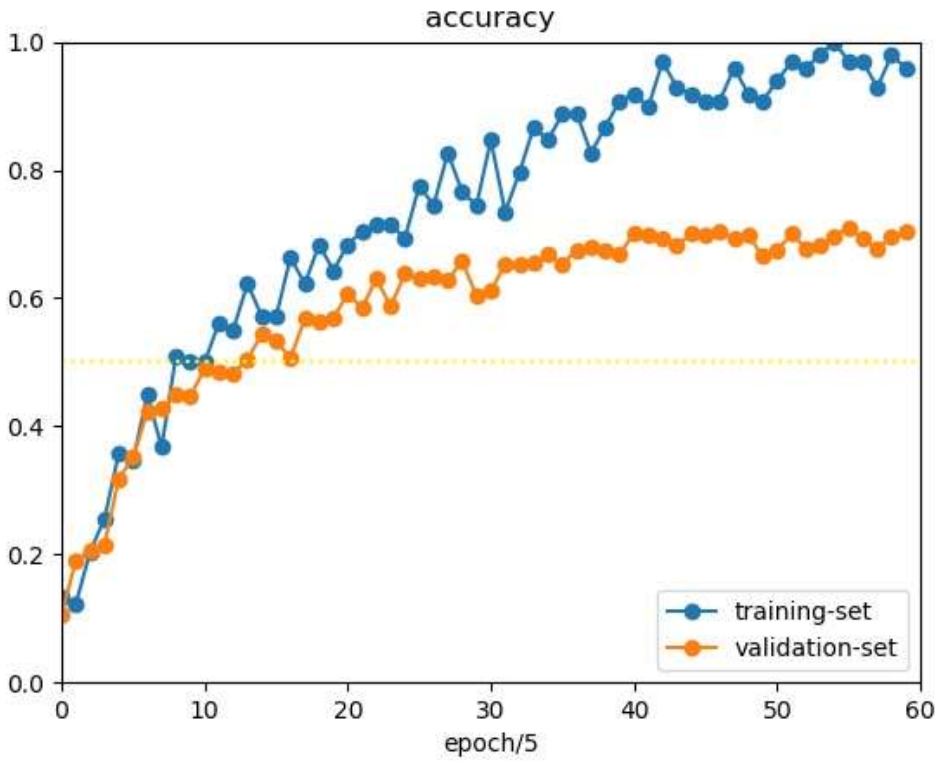
给出模型相关参数：

LeNet-5 结构			
input_size	iteration_number	batch_size	learning_rate / reg
32 * 32 * 3	10000	98	1e-2 / 1e-3
Conv_layer1	filter_size (f)	Channels	params
	5 * 5	32	pad: 0, stride: 1
Pool_layer1	Pool_size	stride	
	2 * 2	2	
Conv_layer2	filter_size (f)	Channels	params
	5 * 5	64	pad: 0, stride: 1

Pool_layer2	pool_size	stride	
	2 * 2	2	
FC_layer3	hidden_neurals	400	
FC_layer4	hidden_neurals	80	
FC_layer5	hidden_neurals	10	



图表 4：神经网络中卷积层部分



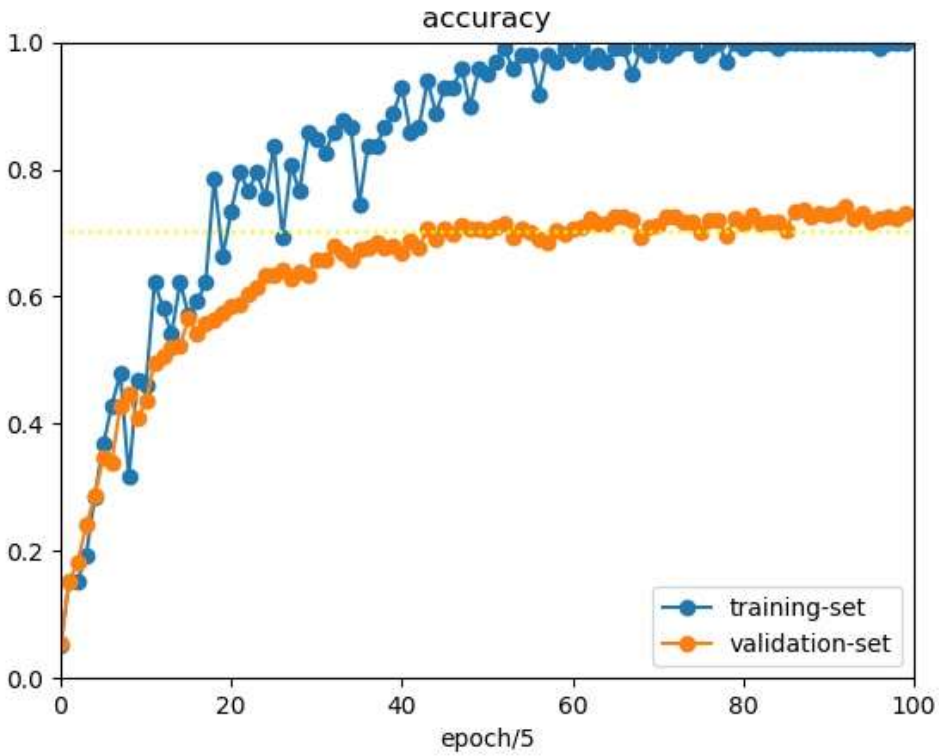
图表 5：训练 80 分钟(6000 iterations)的准确率

上述模型经过 80 分钟的训练，训练集准确率接近 100%，验证集准确率达到 70%，测试集准确率：68.6%

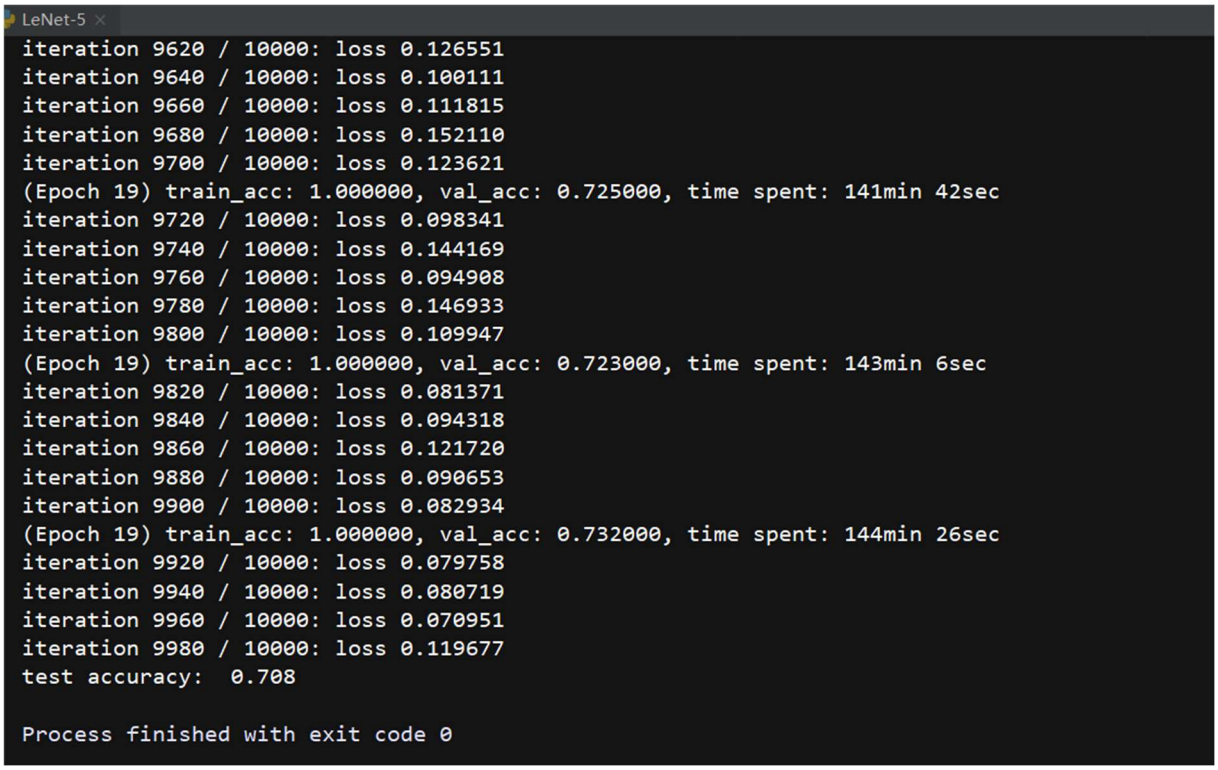
```
LeNet-5
iteration 5560 / 6000: loss 0.530480
iteration 5580 / 6000: loss 0.571186
iteration 5600 / 6000: loss 0.427547
(Epoch 11) train_acc: 0.969388, val_acc: 0.693000, time spent: 77min 40sec
iteration 5620 / 6000: loss 0.436221
iteration 5640 / 6000: loss 0.451420
iteration 5660 / 6000: loss 0.432829
iteration 5680 / 6000: loss 0.492811
iteration 5700 / 6000: loss 0.571696
(Epoch 11) train_acc: 0.928571, val_acc: 0.677000, time spent: 79min 2sec
iteration 5720 / 6000: loss 0.332055
iteration 5740 / 6000: loss 0.449045
iteration 5760 / 6000: loss 0.346143
iteration 5780 / 6000: loss 0.623192
iteration 5800 / 6000: loss 0.498056
(Epoch 11) train_acc: 0.979592, val_acc: 0.695000, time spent: 80min 25sec
iteration 5820 / 6000: loss 0.435762
iteration 5840 / 6000: loss 0.441986
iteration 5860 / 6000: loss 0.423687
iteration 5880 / 6000: loss 0.403469
iteration 5900 / 6000: loss 0.405949
(Epoch 11) train_acc: 0.959184, val_acc: 0.703000, time spent: 81min 48sec
iteration 5920 / 6000: loss 0.529665
iteration 5940 / 6000: loss 0.420735
iteration 5960 / 6000: loss 0.422182
iteration 5980 / 6000: loss 0.445467
test accuracy:  0.686

Process finished with exit code 0
```

图表 6：训练 80 分钟 (6000 iterations)的准确率



图表 7：训练 144 分钟（10000 iterations）的准确率



图表 8：训练 144 分钟（10000 iterations）的测试集准确率达到 70.8%

由于时间关系，我目前没能进行更多的参数优化以及更长时间的测试，准确率停留在 144 分钟的训练水平：训练集接近 100%，验证集 73%左右，测试集 70.8%。

参考资料：

[1] 知乎用户 WILL 的学习笔记 CS231n assignment https://zhuanlan.zhihu.com/p/28483726

[2] CS231n 官方提供的代码 http://cs231n.github.io/assignments2018/assignment2/

[3] Lécun Y, Bottou L, Bengio Y, et al. Gradient-based learning applied to document recognition[J]. Proceedings of the IEEE, 1998, 86(11):2278-2324.

缺陷与不足

1. 由于导入了与加速运算相关的库，使得 NeuralNetwork 库可能无法直接引用（需要进入 NeuralNetwork 先进行 setup 配置才可以使用 fast_layers 部分）。
2. 目前我还没能实现库与测试代码的分离，即：我的训练文件必须在 NeuralNetwork 文件夹内部才能运行，如果移出这个文件夹则无法运行
3. 卷积层中我没有加入 BN 层，这或许延长了拟合时间，降低了拟合准确率
4. 对模型参数的优化不充分，模型训练时间不够，准确率仍有上升空间