CNN for OxFlowers17 实验报告

本实验尝试了多种超参数配置下的 CNN 在 OxFlowers17[1]数据集上的效果，并探究了多种相关技巧对效果的影响。

github 地址：https://github.com/Erutan-pku/DNN_TA_PKU/tree/master/17flowers_CNN

- 数据集描述

该数据集为一个图像分类任务的数据集。数据集包括 1360 张照片，分别属于 17 类花，每个类别各有 80 张照片。

该数据集提供了三种数据集分割，均为 50%-25%-25%的比例均匀分割出训练集、验证集和测试集。我们这里的实验主要使用第一种数据集分割。

该数据集的图片是 RGB24 位色，图片大小不一，大小范围分布在：(499~1057)×(499~1093)之间。所以在使用神经网络时，需要统一图片大小。

- 实验设置

本实验使用 Keras 编写神经网络代码，使用 tensorflow 作为 backend。如无特殊说明，图片大小调整为 128*128，batch_size = 5，每轮迭代随机 shuffle 训练数据顺序。损失函数为 categorical_crossentropy，优化方法为：adadelta，最后一层为带 softmax 激活函数输出 17 类概率的全连接层，其它地方的激活函数均为 ReLu。

- 实验 part-1 探究简单的 MLP 与 CNN 的效果

设置如下：

MLP_nl_m：n 隐层 MLP，隐层节点数 m

CNN_ss: 使用 32 个 3*3 的 CNN filter。（因为训练 4 轮，训练集准确率就到 100%了，而且该设置跑得实在是慢，所以该设置只训练了 10 轮）

CNN_s_1l_32: 使用 32 个 3*3 的 CNN filter，然后过一个 2*2 的 max-pooling filter。（训练 6 轮时训练集准确率到 100%）

[1] http://www.robots.ox.ac.uk/%7Evgg/data/flowers/17/

CNN_s_2l_32: CNN_s_1l_32 的设置叠了两遍。（训练 7 轮时训练集准确率到 100%）

CNN_1l_32_1l_128：在 CNN_s_1l_32 的输出层前面加一个 128 节点的隐层。（训练 11 轮时训练集准确率到 100%）

CNN_1l_32_2l_128：在 CNN_s_1l_32 的输出层前面加两个 128 节点的隐层。（训练 22 轮时训练集准确率到 100%）

CNN_2l_32_2l_128：在 CNN_s_2l_32 的输出层前面加两个 128 节点的隐层。（训练 14 轮时训练集准确率到 100%）

CNN_ss_2l_32_2l_128：CNN_ss 叠起来两遍之后再在输出层前面加两个 128 节点的隐层。（训练 7 轮时训练集准确率到 100%）

实验结果如表 1 所示，训练 30 轮迭代，acc_tst 和 acc_val 为验证集上效果最好的一轮迭代时测试集和验证集上的准确率，n_iter 为该轮的轮数。best_trn 为所有轮中，训练集效果最好的一轮的准确率，可以显示模型的学习能力。（主要考虑到神经网络的表达能力都是非常高的）

	acc_tst	acc_val	n_iter	best_trn
softmax_regression	44.71%	42.62%	30	94.12%
MLP_1l_128	42.35%	38.82%	26	82.79%
MLP_2l_128	35.59%	31.47%	26	60.74%
MLP_3l_128	39.71%	33.82%	29	78.53%
MLP_1l_256	45.29%	38.53%	30	98.38%
MLP_2l_256	46.18%	44.41%	22	99.10%
CNN_ss	58.53%	55.29%	7	100.00%
CNN_s_1l_32	61.47%	58.24%	11	100.00%
CNN_s_2l_32	61.18%	58.82%	7	100.00%
CNN_1l_32_1l_128	60.29%	57.06%	15	100.00%
CNN_1l_32_2l_128	57.35%	53.82%	27	100.00%
CNN_ss_2l_32_2l_128	60.00%	56.18%	7	100.00%
CNN_2l_32_2l_128	**64.71%**	**62.06%**	29	100.00%
CNN_2l_32_2l_256	64.12%	60.00%	20	100.00%

表 1 简单设置下 CNN 与 MLP 的效果对比

从表 1 中可以看出，对于图片信息使用多层感知机的效果往往还不如直接使用 softmax 回归的效果。随着感知机的层数提升，带来的反而是该模型学习能力的下降，模型更难在训练集上拟合。使用卷积层后可以让模型更容易在训练集上拟合的的同时也能够提升模型在测试集上的准确率，在一定程度上体现

了卷积层更适合图片类数据的结构。在卷积层后使用 Pooling 层可以在加快模型学习的速度的同时小幅度提升模型效果（对比 CNN_2l_32_2l_128 Vs CNN_ss_2l_32_2l_128， CNN_s_1l_32 Vs CNN_ss）。在卷积层后加全连接层对效果的影响比较玄学，可能会对效果有所提升，也可能会让效果有所下降。

- 实验 part-2 探究一些 CNN 设置对分类准确率的影响

设置如下：

CNN_128(32,32)(128,128)：就是上一部分 CNN_2l_32_2l_128 的模型。图片大小被调整为 128*128 的，32 个 3*3 的 CNN filter， 2*2 的 max-pooling filter，然后再过 32 个 3*3 的 CNN filter， 2*2 的 max-pooling filter，再过 2 个 128 个节点的隐层，最后 softmax regression 输出。

_dp : 在基础设置上每个 pooling 层后面加入一个 25%的 dropout，每个隐层后面加入一个 50%的 dropout。

_bn: 在基础设置上每个 pooling 层后面加入一个 Batch Normalization,

由于加入 dropout 之后，模型拟合速度更慢，故这里每个用正则化方法的设置迭代训练为 80 轮，其余仍为 30 轮。实验结果如表 2 所示。

	acc_tst	acc_val	n_iter
CNN_128 (32,32)(128,128)	64.71%	62.06%	29
CNN_128 (32,64)(128,128)	61.76%	61.18%	17
CNN_128 (32,64,128)(128,128)	64.71%	**62.94%**	19
CNN_128 (16,32,64)(128,128)	64.41%	57.94%	20
CNN_256 (16,32,64)(128,128)	63.53%	57.94%	15
CNN_256 (32,64,128)(128,128)	63.24%	60.88%	19
CNN_128 (32,32)(128,128)_dp	65.59%	61.76%	65
CNN_128 (16,32,64)(128,128)_dp	**66.18%**	60.59%	75
CNN_256 (16,32,64)(128,128)_dp	62.65%	59.12%	69
CNN_128 (32,32)(128,128)_bn	59.12%	56.18%	19
CNN_128 (16,32,64)(128,128)_bn	62.65%	56.18%	25

表 2 探究一些 CNN 设置对分类准确率的影响

从上表中可以看出，使用 dropout 对效果略有提升，调整各种参数对效果提升得并不明显，反而对于一些比较没道理的参数配置（不在上表中）会让效果比较明显得下降。在这里的实验中，并没有看到 Batch Normalization 对效果的正面影响。

- 实验 part-3 实现一些有名字的 CNN 的效果

考虑到神经网络调参太过玄学，所以我尝试在这个数据集上实现一些有名字的 CNN 模型，以观察、比较效果。这部分我们对三种数据分割都做了实验。

这次是训练 60 轮，每个分割算的是该分割内验证集准确率最高的一轮迭代时的测试集准确率与其对应的验证集准确率。而 Average 取得是平均验证集准确率最高一轮迭代（先平均，再选取轮次）时的测试集准确率与其对应的验证集准确率。具体设置与结果如下：

- part-3.1 AlexNet

AlexNet：完全参照 NIPS12 AlexNet[i]的模型实现，包括使用的 local response normalization 的参数以及将输入图像大小调整为 227*227*3。不过由于这个数据集较小，batch size 减小到了 5。各层超参数：（96-256-384-384-256）-（4096-4096）

AlexNet_short：考虑到本任务训练数据量较小，适当缩减网络深度，减少网络参数与学习难度。删除原来 AlexNet 的第三个卷积层。各层超参数：（96-256-384-256）-（4096-4096）

AlexNet_light：考虑到本任务为 17 分类而不是 1000 分类，在 AlexNet_short 的基础上适当减少靠近输出的层的参数个数，各层超参数：（96-256-256-128）-（1024-1024）

AlexNet_lighter：各层超参数：（96-192-192-96）-（512-512）

AlexNet_lightest：各层超参数：（72-128-128-64）-（128-128）

AlexNet_light_l, AlexNet_lighter_l, AlexNet_lightest_l：对应模型第一个卷积层 filter 大小从 11*11 改为 7*7，步长不变。

AlexNet_lighter_lbn：将 AlexNet_lighter_l 中的 local response normalization 改成普通的 batch normalization。其它不变。

	Seg=1		Seg=2		Seg=3		Average	
	tst	val	tst	val	tst	val	tst	val
AlexNet	66.76	64.12	57.65	63.53	64.12	62.65	61.86	61.57
AlexNet_short	67.94	63.53	61.47	66.47	66.47	66.76	65.49	64.02
AlexNet_light	72.06	68.24	64.12	68.24	70.88	67.35	69.31	66.18
AlexNet_lighter	75.29	68.24	**70.88**	**72.06**	76.76	74.12	71.76	**70.88**
AlexNet_lightest	72.94	68.82	64.71	69.12	73.82	**75.00**	71.57	69.71
AlexNet_light_l	71.47	68.53	63.24	71.47	75.29	73.53	71.47	70.49
AlexNet_lighter_l	76.18	69.41	68.82	71.76	**77.94**	72.94	**74.61**	70.29
AlexNet_lightest_l	**76.76**	**70.29**	65.88	71.76	75.59	70.69	72.16	70.69

AlexNet_lighter_lbn	72.94	68.24	66.47	71.76	69.41	68.82	68.92	67.75

表 3 AlexNet 的效果

从表 3 中可以看出，AlexNet 不愧是有名字的网络，效果普遍不错。不过在减少层数与减少参数、特征数之后，效果会有比较显著的提升。这大概是和这个数据集规模比较小，分得类别比较少有关吧。最后一组实验表明，AlexNet 中的 local response normalization 比普通的 batch normalization 对模型效果的帮助明显要大一些。使用缩减参数后的 AlexNet 网络时，测试集准确率普遍可以达到 75%左右。

AlexNet_lighter_l 在 seg-1 分割下的训练收敛曲线如图 1 所示。

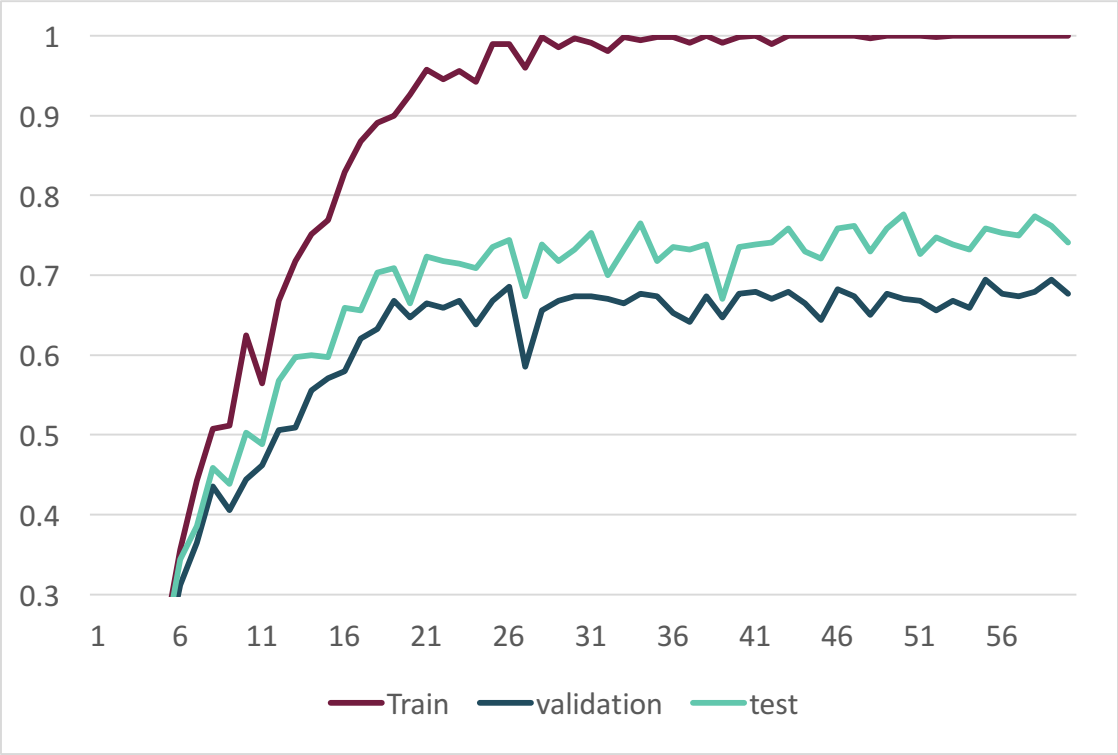

图 1 AlexNet_lighter_l 在 Seg-1 下的训练收敛曲线

（chosen：test=76.18% validation = 69.14%）

- part-3.2 VGG

VGG16：参照 VGG16[ii]的模型实现。各层超参数：（64-64-pooling-128-128-pooling-256-256-256-pooling-512-512-512-pooling-512-512-512-pooling）-（4096-4096）。然而 VGG16 模型迭代训练 40 轮 loss 不降，故放弃。也许是参数太多，网络太深的原因。

VGG13：参照 13 层的 VGG 实现，各层超参数：（64-64-pooling-128-128-pooling-256-256-pooling-512-512-pooling-512-512-pooling）-（4096-4096）。跑得时候出了点意外，第三组分割上 VGG13 也和 VGG16 上一样没有收敛。

VGG13_light：各层超参数：（64-64-pooling-128-128-pooling-256-256-pooling-256-256-pooling-384-384-pooling）-（1024-1024）

VGG13_lighter：各层超参数：（64-64-pooling-96-96-pooling-128-128-pooling-128-128-pooling-64-64-pooling）-（512-512）

VGG13_lightest：各层超参数：（48-48-pooling-64-64-pooling-96-96-pooling-96-96-pooling-48-48-pooling）-（256-256）

VGG13_light_lrn：在 VGG13_light 上加入 local response normalization 与 dropout。

	Seg=1		Seg=2		Seg=3		Average	
	tst	val	tst	val	tst	val	tst	val
VGG16	----	----	----	----	----	----	----	----
VGG13	**57.06**	46.47	**50.00**	**50.59**	----	----	----	----
VGG13_light	53.53	**48.24**	47.06	48.53	**56.47**	**51.47**	**52.16**	**47.84**
VGG13_lighter	48.53	42.94	44.71	46.47	44.71	48.53	44.80	45.10
VGG13_lightest	48.82	42.94	42.06	45.29	45.88	47.94	45.29	45.00
VGG13_light_lrn	----	----	----	----	----	----	----	----

表 4 VGG 的效果

从表 4 中可以看到，VGG 的效果普遍不太好，而且像上面一样的删参数会导致效果更差。怀疑是因为 VGG 网络层数较多，并且因为没有使用正则化的方法，导致训练集上收敛速度过快，过拟合导致的。然而有些设置下不知为何地不收敛，换了其它的优化方法也没有变化。由于 VGG 跑起来实在是太慢了，这里并没有做太多的实验。

- part-3.3 ResNet

ResNet-18	ResNet-34	ResNet-50	ResNet-101
64：k(7*7)，s(2*2)			
MaxPooling：k(3*3)，s(2*2)			
[64, 64]*2	[64, 64]*3	[64, 64, 256]*3	[64, 64, 256]*3
[128, 128]*2	[128, 128]*4	[128, 128, 512]*4	[128, 128, 512]*4
[256, 256]*2	[256, 256]*6	[256, 256, 1024]*6	[256, 256, 1024]*23
[512, 512]*2	[512, 512]*3	[512, 512, 2048]*3	[512, 512, 2048]*3
AveragePooling：k(7*7)			
softmax classification			

表 5 ResNet 的结构设置

ResNet-18、ResNet-34、ResNet-50、ResNet-101：参考原来的论文[iii]中的方法实现的，使用了 batch normalization。网络详细结构见表 5，其中，两层的模块 kernel size 为(3*3),(3*3)，三层的模块 kernel size 为(1*1),(3*3),(1*1)。考虑到 ResNet 有些比较深，所以这里的训练集迭代了 120 轮。

	Seg=1		Seg=2		Seg=3		Average	
	tst	val	tst	val	tst	val	tst	val
ResNet-18	**78.24**	73.24	72.94	**75.59**	72.65	74.71	75.78	73.14
ResNet-34	77.94	**75.00**	**74.71**	73.53	**78.82**	**79.41**	**78.43**	**74.31**
ResNet-50	74.71	73.82	63.24	74.12	73.24	75.29	72.55	72.16
ResNet-101	76.76	70.88	66.18	73.53	69.71	72.06	71.27	71.27

表 6 ResNet 的效果

从上表中可以看到，ResNet 的效果普遍比较好，在 34 层时效果最好，在层数更多时准确率略有下降，也许是因为这个数据集类别数比较少，所以不需要太复杂的特征吧。从 50 层的开始，就使用 1*1 的卷积核在不怎么增加参数的情况下尽量增加特征数了。

ResNet-34 在 seg-1 分割下的训练收敛曲线如图 2 所示。

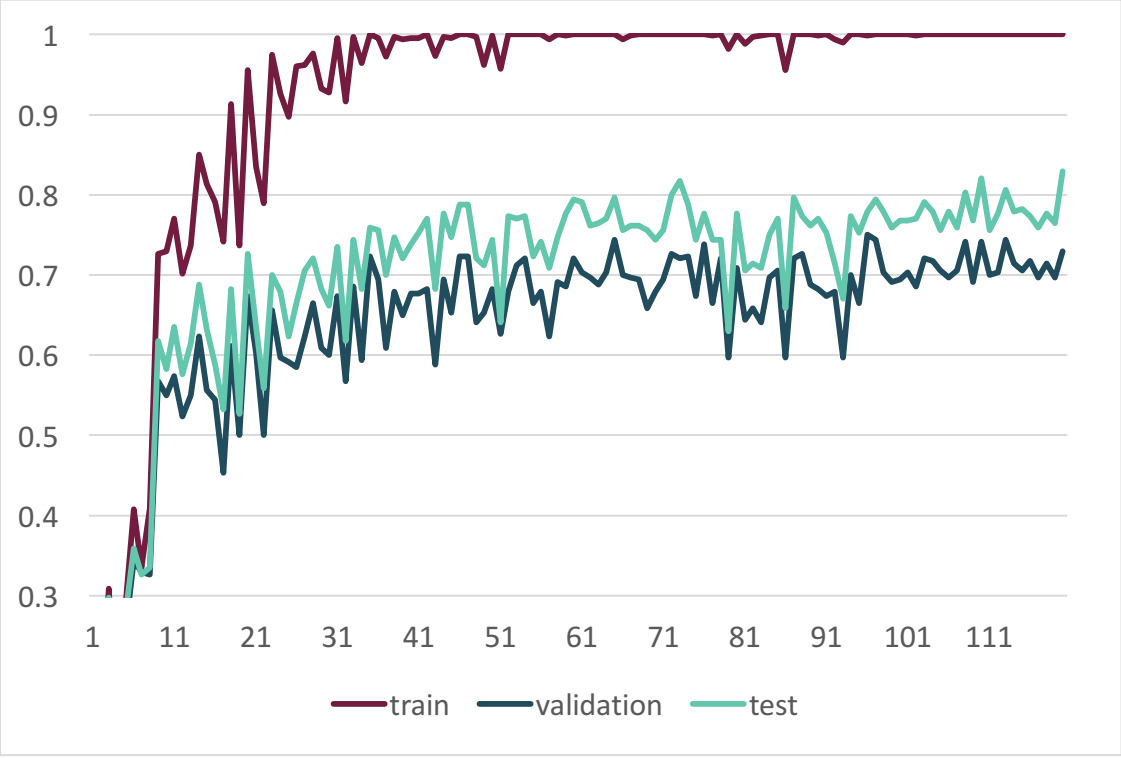

图 2 ResNet-34 在 Seg-1 下的训练收敛曲线

（chosen：test=77.94% validation = 75.00%）

- part-3.4 GoogleNet

GoogleNet V1_s：参考这篇文章[iv]实现的 GoogleNet V1，不过三个输出的地方，只取最后一个计算 loss。具体的网络结构太复杂了，这里就不列了，详见那篇文章。这里迭代了 120 轮。

GoogleNet V1：和 GoogleNet V1_s 相同，不过三个输出的地方都计算 loss，三处比例为：0.5:0.3:0.2，深度越大的地方比例越大。发现这个设置训练速度有点慢，这里迭代了 180 轮。

GoogleNet V1_ave：输出的是 3 个地方输出向量的均值。这里迭代了 180 轮。其实这个相当于 1：1：1 地分配 loss，不过最后测试时使用的不再只是最深层的那个输出而是三个输出的均值。

GoogleNet V3：参考这篇文章实现[v]。参数细节参考 arXiv 上提交的 model.txt。这里迭代了 180 轮。两处输出的 loss 权重比为 1.0:0.4，深度越大的地方比例越大。

	Seg=1		Seg=2		Seg=3		Average	
	tst	val	tst	val	tst	val	tst	val
GoogleNet V1_s	77.06	73.53	67.06	71.76	73.82	73.24	72.16	71.27
GoogleNet V1	**79.41**	**75.88**	**75.00**	**76.76**	**79.41**	**78.82**	**78.14**	**75.59**
GoogleNet V1_ave	74.71	73.82	72.35	74.71	73.24	74.12	72.33	72.65
GoogleNet V3	70.29	67.94	64.41	65.29	66.18	70.29	67.45	67.06

表 7 GoogleNet 的效果

图 3 GoogleNet V1 在 Seg-1 下的训练收敛曲线

（chosen：test=79.41% validation = 75.88%）

表 7 中，GoogleNet V1_ave 的效果下降比较明显，看起来即使迭代了 180 轮，训练集数据效果也没有收敛到 100%。所以还是 GoogleNet V1 的损失分配比率比较合理。GoogleNet V3 的效果较差，感觉上可能是因为这个网络层数过多，结构过于复杂，这个数据集规模过小有关。总体而言，使用 GoogleNet 的效果都还是比较好的。毕竟 GoogleNet 用了很多方法减少模型参数与计算复杂度。

GoogleNet V1 在 seg-1 分割下的训练收敛曲线如图 3 所示。

- 总结

这里，我们测试了一些简单的 MLP 与 CNN 设置以及一些有名字的 CNN 设置在 OxFlowers17 的效果。结果汇总见表 8。总体而言，MLP 与 softmax regression 效果差不多。 CNN 的效果普遍比 MLP 要好。有名字的比较深的 CNN 的效果普遍比自己随便调的 CNN 的效果要好。因为这个数据集比较小，像 VGG 这种比较 heavy 的 CNN 模型的效果相对而言要差一些，而像一些比较新的 CNN 模型如 GoogleNet 与 ResNet 在加深的同时尽量减少参数和增加模型的易学习度，可以在这个数据集上取得相对更好的效果。

	Seg=1		Seg=2		Seg=3		Average	
	tst	val	tst	val	tst	val	tst	val
softmax_regression	44.71	42.62	----	----	----	----	----	----
CNN_128 (32,32)(128,128)	64.71	62.06	----	----	----	----	----	----
CNN_128 (16,32,64)(128,128)dp	66.18	60.59	----	----	----	----	----	----
AlexNet_lighter_l	76.18	69.41	68.82	71.76	77.94	72.94	74.61	70.29
ResNet-34	77.94	75.00	74.71	73.53	78.82	**79.41**	**78.43**	74.31
GoogleNet V1	**79.41**	**75.88**	**75.00**	**76.76**	**79.41**	78.82	78.14	**75.59**

表 8 各种模型效果汇总

---

[i] Krizhevsky, Alex, I. Sutskever, and G. E. Hinton. "ImageNet classification with deep convolutional neural networks." *International Conference on Neural Information Processing Systems* Curran Associates Inc. 2012:1097-1105.

[ii] Simonyan K, Zisserman A. Very Deep Convolutional Networks for Large-Scale Image Recognition[J]. Computer Science, 2014.

[iii] He, Kaiming, et al. "Deep Residual Learning for Image Recognition." (2015):770-778.

[iv] Szegedy, C, et al. "Going deeper with convolutions." (2015):1-9.

[v] Szegedy, Christian, et al. "Rethinking the Inception Architecture for Computer Vision." *Computer Science* (2015):2818-2826.