

手写数字识别实验报告


2020211543 经硕 201 杨喜凯

目录

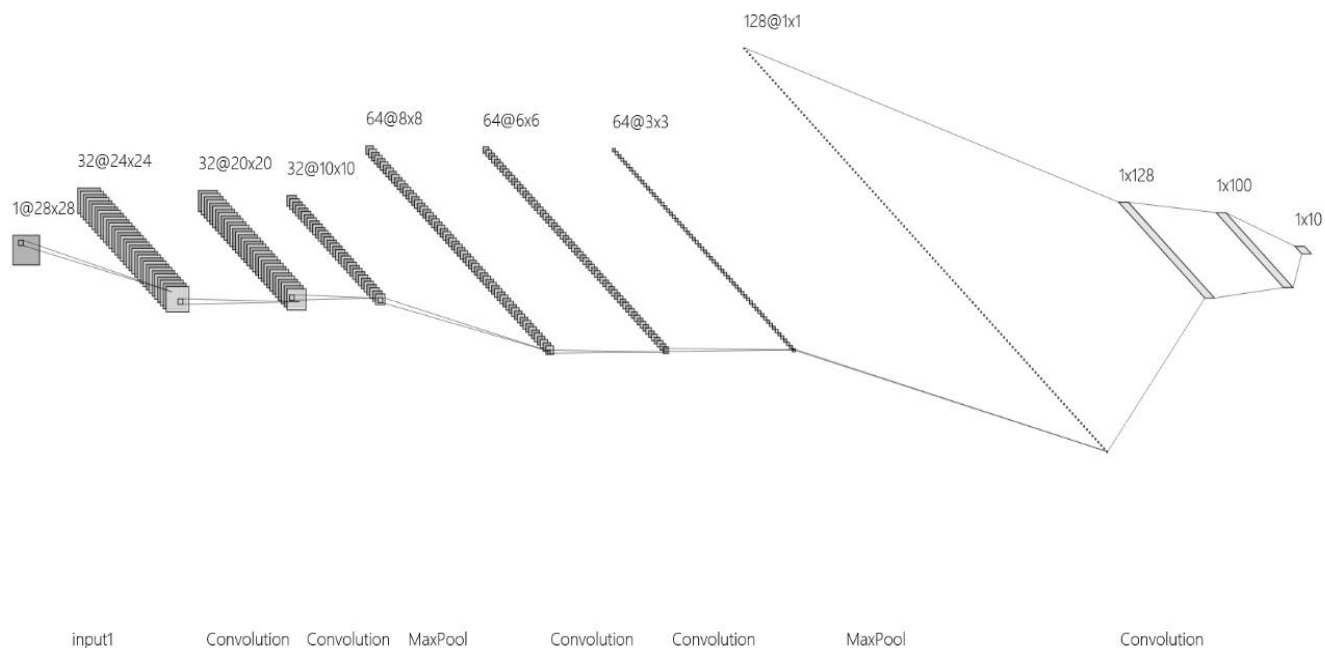
(一) 模型结构及流程分析	1
(二) 模型选择	3
(三) 超参数调整	4
(四) 问题思考	6
(五) 心得体会	8

(一) 模型结构及流程分析

本实验最终的结果如下所示，在测试集上的 accuracy 为 99.057%，共提交三次，leaderboard 排名为 918 名。

918	yxk2020211543		0.99057	3	1d
-----	---------------	---	---------	---	----

本实验最终采取的模型为卷积神经网络，卷积神经网络是一类包含卷积计算且具有深度结构的前馈神经网络，其在图像处理上有着独特的优势。本实验最终采取的卷积神经网络结构图如下所示：



由上图可见我们的卷积神经网络结构，但实际上我们还采用了 BatchNorm 层以及 Dropout 层，在上图中并未绘制出；我们利用 torchsummary 中的 summary 方法打印出模型的实际结构如下图所示：

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 32, 24, 24]	832
BatchNorm2d-2	[-1, 32, 24, 24]	64
Conv2d-3	[-1, 32, 20, 20]	25,632
BatchNorm2d-4	[-1, 32, 20, 20]	64
Dropout2d-5	[-1, 32, 10, 10]	0
Conv2d-6	[-1, 64, 8, 8]	18,496
BatchNorm2d-7	[-1, 64, 8, 8]	128
Conv2d-8	[-1, 64, 6, 6]	36,928
BatchNorm2d-9	[-1, 64, 6, 6]	128
Dropout2d-10	[-1, 64, 3, 3]	0
Conv2d-11	[-1, 128, 1, 1]	73,856
BatchNorm2d-12	[-1, 128, 1, 1]	256
Conv2d-13	[-1, 128, 1, 1]	16,512
BatchNorm2d-14	[-1, 128, 1, 1]	256
Linear-15	[-1, 100]	12,900
Linear-16	[-1, 10]	1,010

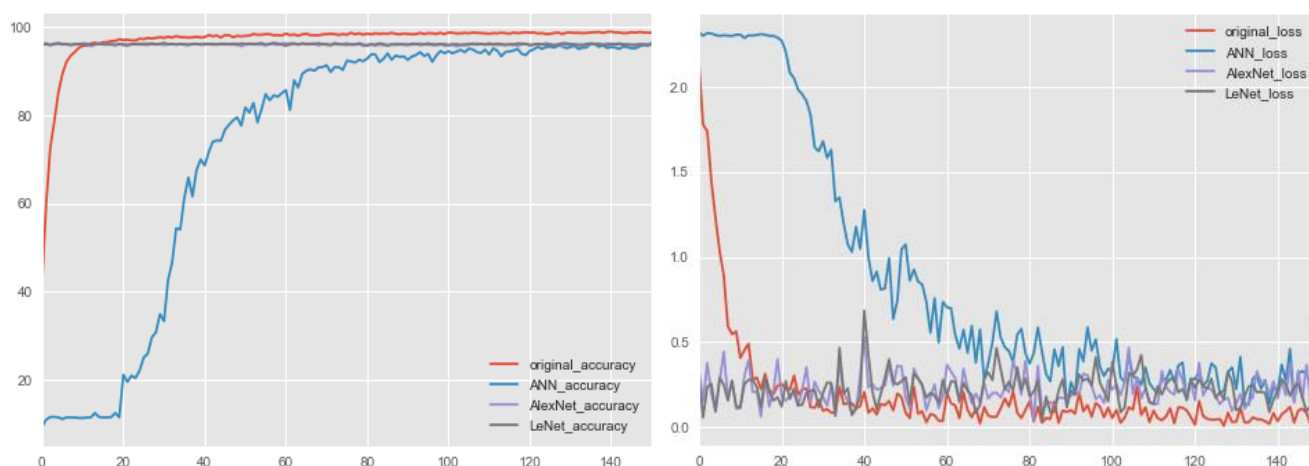
从上图我们也可以分析 CNN 模型的处理流程, 我们的 input 是一张灰度图像($1 \times 28 \times 28$), 首先经历一层 $\text{kernel_size}=5$ 的卷积层之后变为 ($32 \times 24 \times 24$), 接下来经历第二层 $\text{kernel_size}=5$ 的卷积层之后变为 ($32 \times 20 \times 20$), 接下来我们对其进行 $\text{kernel_size}=2$ 的 MaxPool2d 之后维度变为 ($32 \times 10 \times 10$); 接下来我们进行第三层和第四层 $\text{kernel_size}=3$ 的卷积层后维度变为 ($64 \times 6 \times 6$), 再进行一层 MaxPool2d 后维度变为 ($64 \times 3 \times 3$), 在最后一层卷积层中, 我们 $\text{kernel_size}=3$, 将数据维度变为 ($128 \times 1 \times 1$); 随后我们将数据放入两层全连接层并最终得到相应的输出。

其中我们穿插了很多 Batch Normalization 层。我们的神经网络本质上是要学习训练数据的分布; 因此如果我们每一个 batch 输入的数据都具有不同的分布, 显然会给网络的训练带来困难。另一方面, 数据经过若干层网络后, 其数据分布也在发生着变化, 因此便产生了 internal covariate shift 问题, 会给后面的网络学习带来困难。因此我们采用了 Batch Normalization 层来帮助我们解决这个问题, 与此同时其可以帮助我们的神经网络提高训练速度。

神经网络面临的问题主要是训练速度慢以及过拟合, 对于前者来说, 我采用了使用 GPU 训练神经网络从而使得训练速度得以提升; 而对于后者而言, 我主要采用了 Dropout 层。Dropout 层是指在深度学习网络的训练过程中, 对于神经网络单元, 按照一定的概率将其暂时从网络中丢弃, 因此对于每一个 mini-batch, 我们都在训练不同的网络, 从而可以起到减少过拟合的效果。

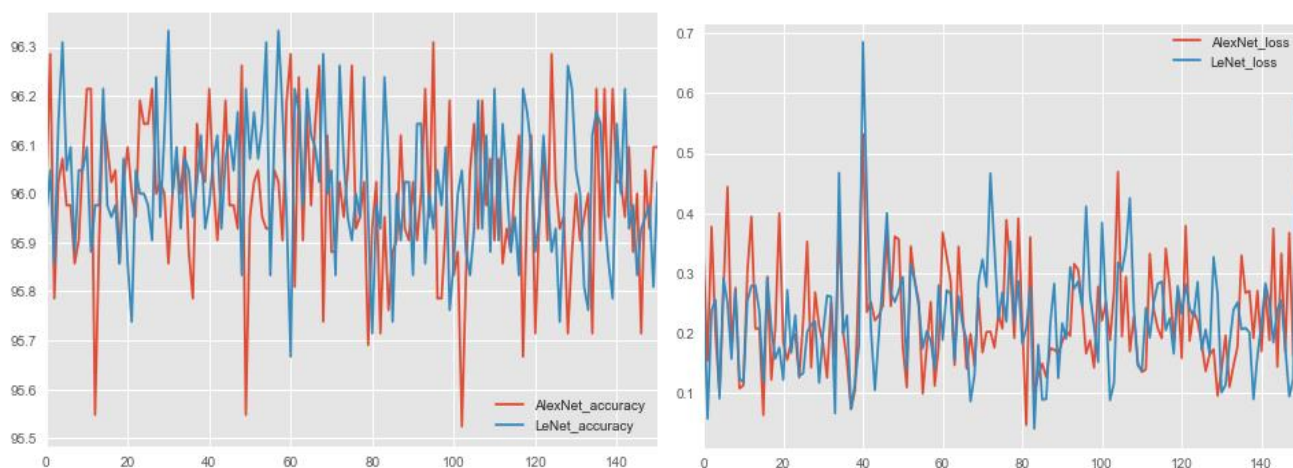
(二) 模型选择

接下来我们对模型进行比较, 除去我们最终采用的模型之外我们还选取了 LeNet, AlexNet 以及 ANN 进行比较; 对于每一个模型, 我们将学习率设置为 0.01, epochs 设为 20, 记录其在训练集上的 loss 以及在验证集上的 accuracy。该四个模型对应的 loss 曲线以及 accuracy 曲线如下所示(其中 original 代表我们所选取的模型, 横轴仅代表取样次数而非 epoch 数)。



由上图可见，我们所选取的模型在训练一定量的 epoch 之后不论是 accuracy 还是 loss 都优于其他模型；除此之外，我们发现 ANN/LeNet/AlexNet 最终的 accuracy 和 loss 都收敛至大致相近的值。略差于我们所选取的模型。有趣的是，LeNet 和 AlexNet 的表现一开始就较好，但随着 epoch 数目的提升，并没有明显的提升，而是呈现一种震荡的态势。

接下来我们仅绘制 LeNet 以及 AlexNet 相对应的图线。由下图可以看到，LeNet 和 AlexNet 的 accuracy 以及 loss 都呈现一种震荡的态势，但未随着 epoch 的提升而又显著提升。

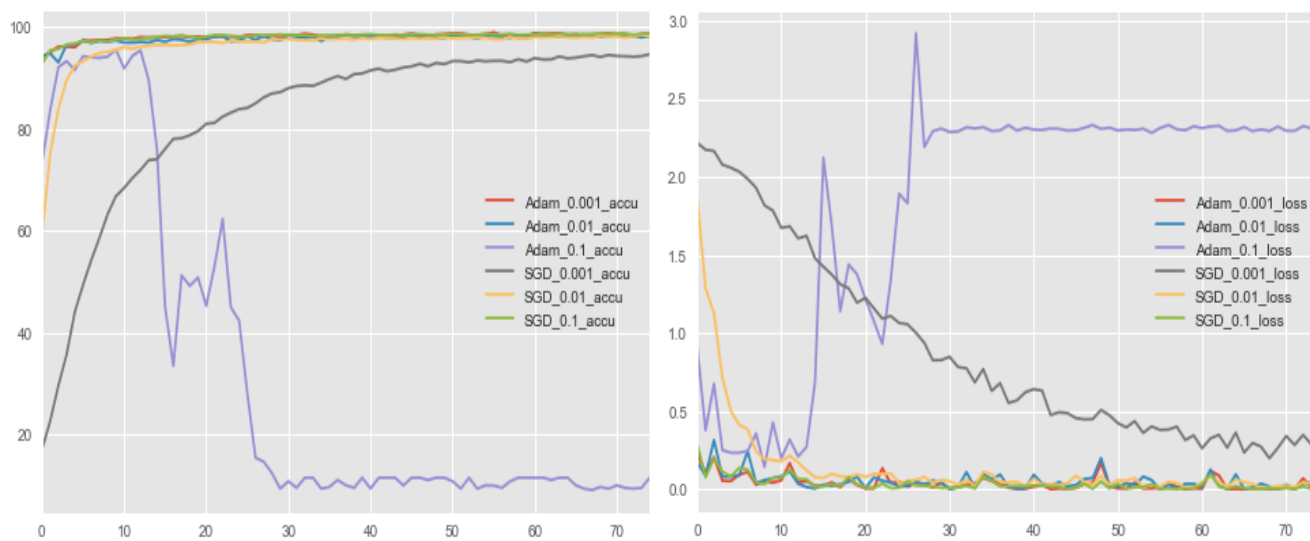


因此由上所述，我们认为我们最初的模型要优于其他的模型，因此最终我们仍然选用我们最初的模型。

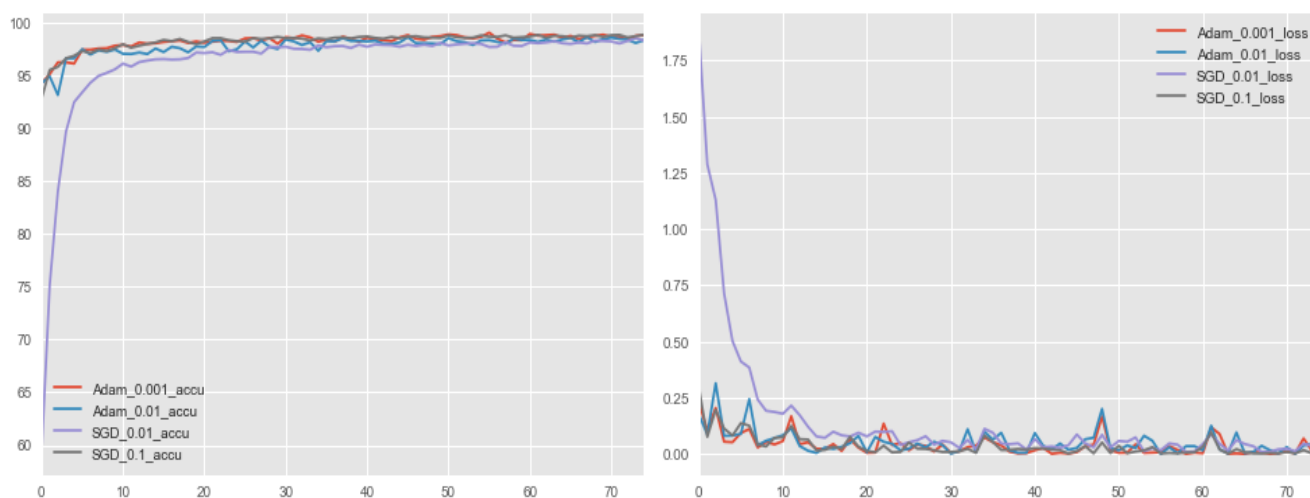
(三) 超参数调整

接下来我们调整优化器以及学习率，此处我们选取 learning

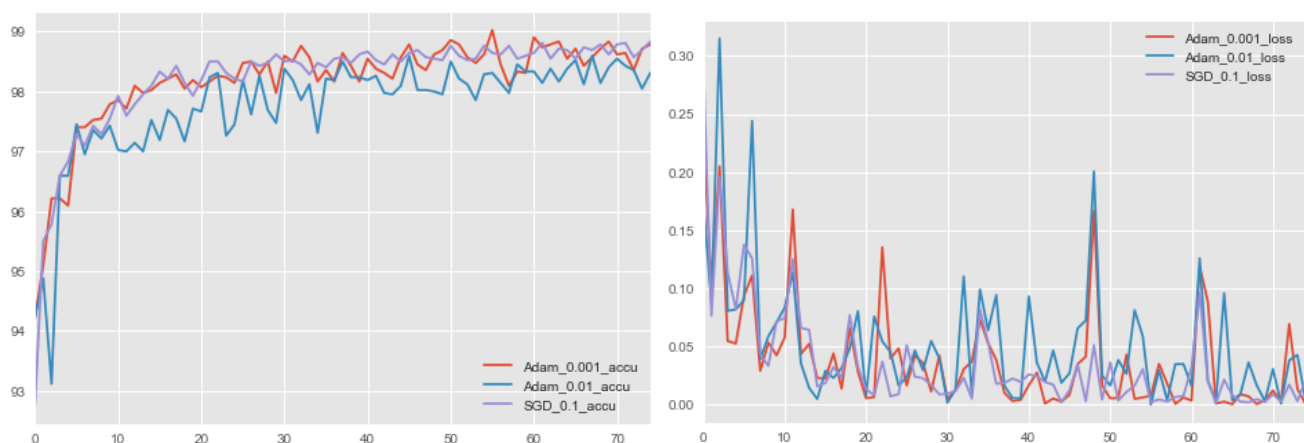
rate={0.001, 0.01, 0.1}, optimizer = {SGD, Adam} 共六种组合；将 epochs 设置为 10，记录其在训练集上的 loss 以及在验证集上的 accuracy 并绘制图像如下所示：



可以看到 {Adam, 0.1} 以及 {SGD, 0.01} 这两个组合显著劣于其他组合，并且此处出现了一个较为异常的现象，对于 {Adam, 0.1} 的组合，其出现了 loss 值下降后又显著上升的现象；我们认为可能是因为 Adam 的学习率过大而导致的。我们将这两种组合去掉后绘制新的图像如下所示：



从以上图像中我们可以看到四种组合最终的收敛效果类似，但 {SGD, 0.01} 要稍劣于其他三种组合。接下来我们仅对剩下的三种组合绘制图像如下所示：



从 accuracy 中我们可以看到当 optimizer=SGD, learning_rate=0.1 与 optimizer=Adam, learning_rate=0.001 时效果较好; 我们进一步将 num_epochs 增大发现当采用 SGD, 学习率为 0.1 时效果要略好于 Adam/learning_rate=0.001; 因此最终我们的 optimizer 选取 SGD, 学习率设置为 0.1。并且最终在测试集上 accuracy 达到 99%。

(四) 问题思考

关于如何确定训练的最佳次数, 一种做法是将 epochs 作为一项超参数来进行调整, 但这样存在的问题是计算耗时太高, 效率过低; 另一种方法是 early stopping, 在训练过程中如果在验证集上的 loss 开始上升或者 accuracy 开始下降, 此时我们应该停止训练过程以尽可能减少过拟合; 但实际训练过程中远比此复杂, 损失函数可能存在多个局部极值点, 也就意味着在训练过程中 loss 在上升一段时间后仍然有可能在未来下降, 因此常见的做法是记录到目前为止最好的验证集精度, 当连续 10 次(或者更多次)Epoch 没达到最佳精度时, 则可以认为精度不再提高了。而对于固定迭代次数, 我们认为其更适合在参数调整阶段设置一个较为固定的值, 从而便于我们比较不同参数的效果; 本次实验中我们使用的是固定迭代次数, 最终的 epochs=50, 不过我们会发现当 epochs=20 时几乎已经达到收敛, 再之后 loss 一直较为稳定, 但在验证集上的 accuracy 也并未恶化, 因此对于我们而言是可接受的。

对于实验参数的初始化, 如果将权重参数做零初始化, 那么各个节点的激活函数值是相同的, 梯度也是相同的, 更新后的权重仍然是相同的。那么不同的节点就无法学习到不同的特征, 这对于神经网络而言是无法接受的。而如果将权重做高斯分布初始化, 那么当神经网络层数增加时, 容易出现梯度消失或者梯度爆炸的现象, 那么神经网络收敛速度很慢。

Xavier 初始化的基本思想是, activations 的方差是逐层递减的, 这导致反向传播中的梯度

也逐层递减。要解决梯度消失，就要避免 **activations** 方差的衰减，即每一层输出的方差应该尽量相等。若对于每一层网络的输出都可以保持正态分布且方差相近，这样就可以避免输出趋向于 0，从而避免梯度消失情况，Xavier 初始化便是实现了这一功能。Xavier 初始化有一定限制，其推导过程假设激活函数在零点附近接近线性函数，且激活值关于 0 对称。因此对于 **tanh** 激活函数而言，初始化方法可以使用 Xavier 初始化。

但对于 **ReLU** 而言，其显然是非对称的。He 初始化的基本思想是，由于 **ReLU** 函数让一半的 **Z** 值变为零，实际上移除了大约一半的方差。所以我们需要加倍权重的方差来补偿这一点。补偿的方法便是对 Xavier 初始化进行调整，将权重的方差乘以 2。

随着网络深度加深，输入值的分布逐渐发生偏移，之所以训练收敛慢，一般是整体分布逐渐往非线性函数的取值区间的上下限两端靠近，导致反向传播时低层神经网络的梯度消失。因此除去上述初始化方法之外，我们还可以用 **Batch Normalization** 来解决这种偏移问题。

本实验中参数的初始化选取的是默认初始化方法；对于 **pytorch** 而言，不同的 **layer** 有不同的默认初始化方法，例如 **Conv2d** 中 **weights** 采用的是 **Kaiming_He_init_function**，我们认为已经较为合适。因此在本实验中我们对于各个层均采用了其默认初始化方法。

关于如何避免过拟合。过拟合本质上是一种数据问题，因此最直接的方式便是增加数据；而对于本实验而言，进行数据增强 (**data augmentation**) 便可以避免过拟合；对于手写数字而言，对其进行随机平移以及适度旋转，便可以在不改变真实 **label** 的前提下扩充数据的分布，从而避免陷入过拟合。

除此之外，更简单的网络结构也有利于减少过拟合；减少网络层次，神经元个数都有助于网络结构复杂度的下降；

除此之外还可以使用 **dropout** 技术，每次训练时随机“暂时”舍弃一部分节点，从而每个 **mini-batch** 训练的节点都有所不同。其核心思想便是通过引入噪音（某些节点的输出值设置为 0），从而增强模型的泛化能力。

传统统计学习中常用的 **L1/L2** 正则化同样也适用于神经网络，通过加入惩罚项来限制参数的大小，从而起到简化模型结构的作用。前面提及到的 **early stopping** 通过提前结束训练同样可以避免过拟合。

关于卷积神经网络相对于全连接神经网络的优点，卷积神经网络可以利用卷积核对局部的特

征进行抽取，一般来说，卷积核的大小会小于图像的大小(否则等价于全连接神经网络)，因此卷积提取出的特征会更更多地关注局部，这与人类观察图像时的特点是一致的，只需要先观察局部的特征，而后在更高层将局部的特征综合起来得到其分类。而对于全连接神经网络而言，其实际上是不关心像素点彼此之间的位置关系的。

因此在抽取局部特征时，不同位置实际上是共享同一个卷积核，这就带来了参数共享。参数共享最大的作用便在于很大程度上减少了参数的数量，降低了计算的复杂度，使得神经网络可以处理更高维的数据。而如果使用全连接神经网络处理图像则需要估计大量的参数，在参数估计和内存消耗上都效率较低。

卷积神经网络还可以使用池化层，可以在减少参数数量的同时仍然保持图像的特征，也即可以用较少的特征来代表一个区域的特征。而对于一般的全连接神经网络而言，我们无法缩减参数量，因为每个 input 之间并不存在位置上的相关关系。

除此之外，一般我们都不会只用一个卷积核对输入图像进行处理，因为其提取的特征过于单一化。通常我们需要多个核来处理图像，也就是提取图像的不同特征，这有些类似随机森林的思想，也有些类似人类从多个角度来看待事物。

(五) 心得体会

这次作业我最深刻的体会便是感受了深度学习的神奇之处;之前总是听有人戏称深度学习为“炼丹”，当时并没有很深的体会，不过这次作业让我感受到了“炼丹”的感觉。经常会发现对于同样的模型，只需要换一下优化器或者学习率，其表现就可以有很大的不同，也许这在某种程度上说明了损失函数的复杂性；在与其他同学交流的过程中得知有些同学的模型未经过 data augmentation 就可以在几个 epoch 内到达 99%以上的 accuracy，但我自己尝试了很多不同的模型结构也调参了很久却最高只能到达 99%左右，这令我感到很困惑，也许是中间某个环节处理得不够恰当。

另外一个发现便是模型的复杂性与最终的 accuracy 之间可能并不是简单的正比关系或是二次函数关系；我在最开始时训练了一个非常简单的 CNN 模型(20 epochs)，当时在验证集上的 accuracy 便可以达到 98.5%，后来尝试了很多更为复杂的模型，发现在验证集上的表现大多也都没有显著提升。即使最终我选用的模型效果较 98.5%有了 0.5%的提升，但我仍然对于模型结构的选择感到困惑。因此如何挑选合适的模型结构，对于我而言还是一个类似“黑盒”的事情，希望今后能有更多的机会去学习去尝试。

除此之外，在浏览一些效果比较好的 notebooks 时我发现大多数参赛者的模型可能各不相同，但大家都有着一个共同点那便是使用了 data augmentation，在本次实验中我并没有使用 data augmentation，只是阅读了一下相关的代码(Keras 中有现成的 API 可以便捷的去做图像的数据 augmentation，但在 pytorch 中需要自己写)。而且我们可以发现使用 data augmentation 可以让模型的 accuracy 有大幅度的提升，这也向我们展现了增加数据量/减少过拟合对于提升模型表现的重要性。

这次大作业中我第一次接触了深度学习一个相对来说比较全面的流程，也借此熟悉了 kaggle 平台的使用。之前我对于深度学习的理解仅限于理论层面，最多也就是用 keras 写一个简单的 ANN；不过这次作业我并没有用 keras，也是想借这次机会学习一下 pytorch。整个过程收获了很多，不仅仅体验了深度学习的整个流程，而且学会了 pytorch 的基本使用，受益颇丰。感谢这次大作业的机会，感谢马老师和助教学长学姐的悉心教导！