《人工智能基础》

实验报告

项目名称：人脸识别

学生姓名：唐麒

学　　号：17301138

1. 实验数据集

1.1. 数据预处理

原始数据集包含了三个类别的人脸图像(*pengyuyan*、*zhangziyi*、*jiangwen*)，且不同类别下人脸图片数量均在 100 张左右。

原始数据集部分图片

为了使模型能识别本人和 5 位电影明星照片，需要在原始数据集中添加新的类别，通过爬虫定向爬取明星照片获得另外两位电影明星的人脸数据，而本人的照片则通过 OpenCV 调用电脑摄像头进行连续拍摄。由于爬虫爬取的图片良莠不齐，新的类别中图片数量远大于原始数据中样本均在 100 左右的分布情况，且大小差异较大，故通过 OpenCV 的人脸分类器*haarcascade_frontalface_default.xml* 对获得的图像进行筛选，将包含人脸且所占像素大于 50×50 的图像保留(去除较小的人脸或图像中的其他人脸)，并将识别到的部分进行裁剪保留。最后，通过人工筛选，保留新增类别图像在 100 张左右。

添加数据后的数据集部分图片

1.2. 数据加载

在对数据进行预处理之后，将不同类别的图片数据放置在不同的文件夹下，以此表示该类别的标签。通过遍历图像数据文件夹，统计所有的类别和每种类别下图片的数量以及路径，生成用以加载数据的 trainer.list/test.list 和关于数据说明的 readme.json，从 readme.json 中，我们可以再观察一下新的数据集，如下图所示：

注：在生成数据列表的过程中，每十张图片中选出一张作为测试数据，其余的作为训练数据。

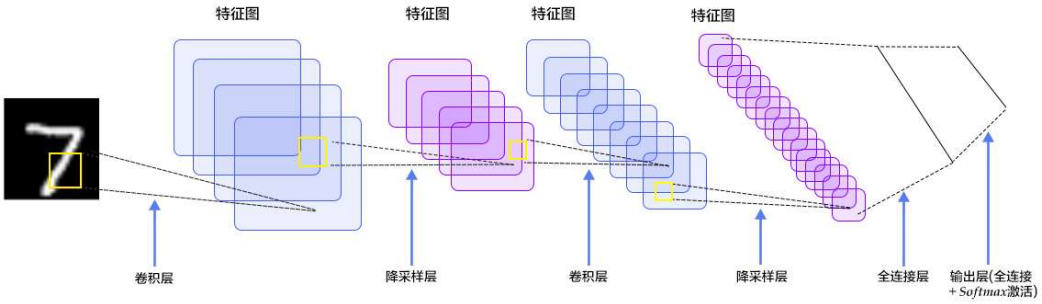
通过自定义 *train_reader* 和 *test_reader* 将数据加载到模型中。通过传入生成的 *list* 路径，逐行读取图片的存储路径和标签。利用 *Paddle* 提供的函数读入图片，并对图片的大小进行处理，变换为 $3 \times 100 \times 100$ 的形式，保留彩色图像的特征，并将图像存储的矩阵进行归一化（0~1 之间）。

通过 *Paddle* 提供的 batch 和 shuffle 进行批量读入和打乱。

2. 网络结构

1） Lenet-5

在多层感知器模型中，将图像展开成一维向量输入到网络中，忽略了图像的位置和结构信息，而卷积神经网络能够更好的利用图像的结构信息。$LeNet-5$ 是一个较简单的卷积神经网络。下图显示了其结构：输入的二维图像，先经过两次卷积层到池化层，再经过全连接层，最后使用 $Softmax$ 分类作为输出层。



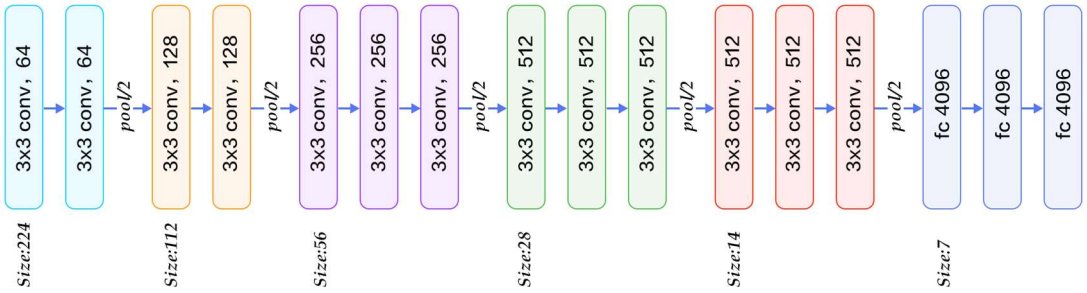
LeNet-5 模型

本项目中使用的 $LeNet-5$ 模型包含一个输入层、两个卷积层、两个池化（降采样）层、和一个输出层（全连接层），其中：

- 卷积层 1:卷积核大小为 5×5，一共有 20 个卷积核；
- 池化层 1:池化大小为 2×2，步长为 2；
- 卷积层 2:卷积核大小为 5×5，一共有 50 个卷积核；
- 池化层 2:池化大小为 2×2，步长为 2；
- 输出层：激活函数为 $Softmax$ 的输出层，大小为 10。

2） VGG

该模型相比以往模型进一步加宽和加深了网络结构，它的核心是五组卷积操作，每两组之间做 $Max-Pooling$ 空间降维。同一组内采用多次连续的 3×3 卷积，卷积核的数目由较浅组的 64 增多到最深组的 512，同一组内的卷积核数目是一样的。卷积之后接两层全连接层，之后是分类层。由于每组内卷积层的不同，有 11、13、16、19 层这几种模型，下图展示一个 16 层的网络结构。



VGG 模型

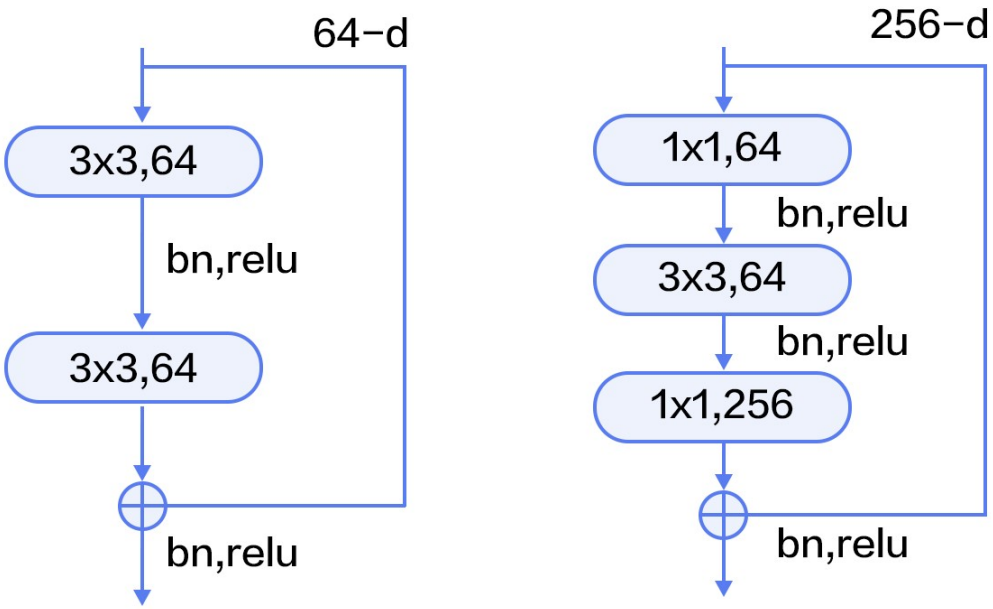
本项目定义一组卷积网络，卷积核大小为 3×3，池化窗口大小为2×2，窗口滑动大小为 2。 共进行五组卷积操作，第一、二组采用两次连续的卷积操作，第三、四、五组采用三次连续的卷积操作。最后接两层 512 维的全连接层和一个激活函数为 $Softmax$ 的输出层，大小为 10。

3） ResNet

VGG 网络试着探寻了一下深度学习中网络的深度与提高分类准确率的持续关系。一般印象当中，深度学习愈是深（复杂，参数多）愈是有着更强的表达能力。凭着这一基本准则 CNN 分类网络自 $Alexnet$ 的 7 层发展到了 VGG 的 16 乃至 19 层，后来更有了 $Googlenet$ 的 22 层。可当 CNN 网络达到一定深度后再一味地增加层数并不能带来进一步地分类性能提高，反而会招致网络收敛变得更慢，准确率也变得更差。

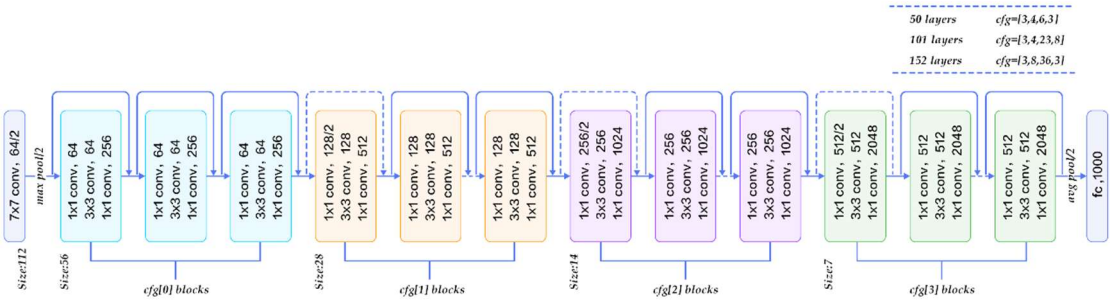
$ResNet$ 通过使用多个有参层来学习输入输出之间的残差表示，而非像一般 CNN 网络（如 $Alexnet/VGG$ 等）那样使用有参层来直接尝试学习输入、输出之间的映射。$ResNet$ 有关实验表明使用一般意义上的有参层来直接学习残差比直接学习输入、输出间映射要容易得多（收敛速度更快），也有效得多（可通过使用更多的层来达到更高的分类精度）。

残差模块如下图所示，左边是基本模块连接方式，由两个输出通道数相同的 3×3 卷积组成。右边是瓶颈模块(Bottleneck)连接方式，之所以称为瓶颈，是因为上面的 1×1 卷积用来降维(图示例即 256->64)，下面的 1x1 卷积用来升维(图示例即 64->256)，这样中间 3×3 卷积的输入和输出通道数都较小(图示例即 64->64)。



残差模块

下图展示了 50、101、152 层网络连接示意图，使用的是瓶颈模块。这三个模型的区别在于每组中残差模块的重复次数不同(见图右上角)。ResNet 训练收敛较快，成功的训练了上百乃至近千层的卷积神经网络。

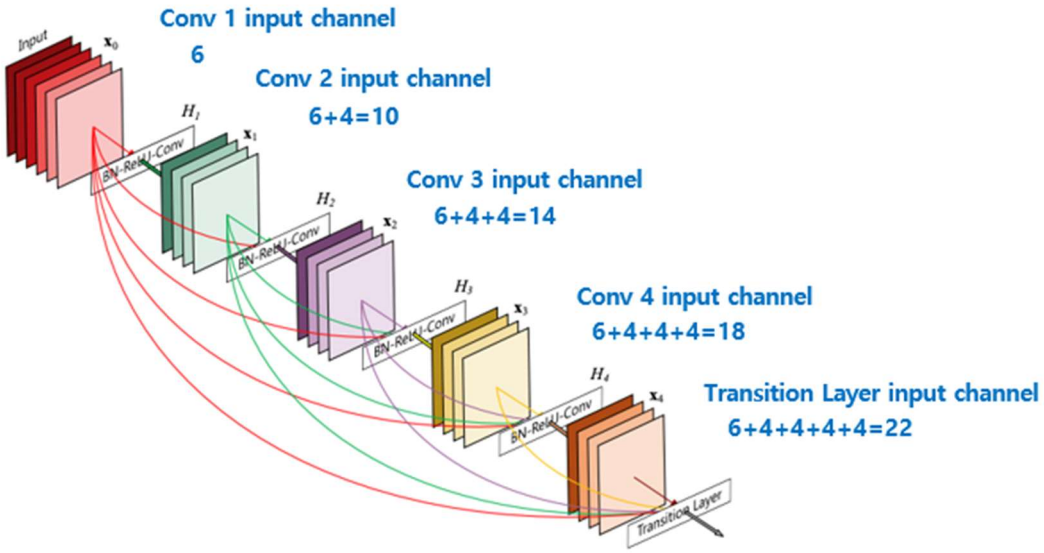


ResNet 模型

本项目的网络模型的连接为 1 个卷积层，卷积核大小为 3×3，一共有 16 个卷积核；1 个池化层，一个池化层，池化大小为 3×3，步长为 2；定义一组残差模块，由若干个基础残差模块堆积而成，每个基础残差模块，由三组大小分别为 1×1 ，3×3，1×1 的卷积组成的路径和一条"直连"路径组成，每一组卷积和的个数与传入本组残差模块的参数有关，分别为参数的 1 倍，1 倍和 4 倍。本项目共设置 3 组残差模块，传入残差模块卷积核的个数参数分别为 16, 32, 32。然后对网络做均值池化，最后是一个激活函数为 $Softmax$ 的输出层，大小为 10。

4) *DenseNet*

ResNet 模型的出现是 *CNN* 史上的一个里程碑事件，*ResNet* 可以训练出更深的 CNN 模型，从而实现更高的准确度。*ResNet* 模型的核心是通过建立前面层与后面层之间的“短路连接”（shortcuts，skip connection），这有助于训练过程中梯度的反向传播，从而能训练出更深的 *CNN* 网络。*DenseNet* 模型的基本思路与 *ResNet* 一致，但是它建立的是前面所有层与后面层的密集连接（dense connection），它的名称也是由此而来。*DenseNet* 的另一大特色是通过特征在 channel 上的连接来实现特征重用（feature reuse）。这些特点让 *DenseNet* 在参数和计算成本更少的情形下实现比 *ResNet* 更优的性能。



如果用公式表示的话，传统的网络在 l 层的输出为：

$$x_l = H_l(x_{l-1})$$

而对于 *ResNet*，增加了来自上一层输入的 *identity* 函数：

$$x_l = H_l(x_{l-1}) + X_{l-1}$$

在 *DenseNet* 中，会连接前面所有层作为输入：

$$x_l = H_l(x_0, x_1, \cdots, x_{l-1})$$

其中，上面的$H(\cdot)$代表是非线性转化函数（non-liear transformation），它是一个组合操作，其可能包括一系列的 BN(Batch Normalization)，*ReLU*，*Pooling* 及 *Conv* 操作。注意这里 l 层与 $l-1$ 层之间可能实际上包含多个卷积层。

CNN 网络一般要经过 *Pooling* 或者 $stride > 1$ 的 *Conv* 来降低特征图的大小，而 *DenseNet* 的密集连接方式需要特征图大小保持一致。为了解决这个问题，*DenseNet* 网络中使用 *DenseBlock* + *Transition* 的结构，其中 *DenseBlock* 是包含很多层的模块，每个层的特征图大小相同，层与层之间采用密集连接方式。而 *Transition* 模块是连接两个相邻的 *DenseBlock*，并且通过 *Pooling* 使特征图大小降低。

在 *DenseBlock* 中，各个层的特征图大小一致，可以在 *channel* 维度上连接。*DenseBlock* 中的非线性组合函数 $H(\cdot)$ 采用的是 $BN + ReLU + 3 \times 3\ Conv$ 的结构。另与 *ResNet* 不同，所有 *DenseBlock* 中各个层卷积之后均输出 k 个特征图，即得到的特征图的 *channel* 数为 k ，或者说采用 k 个卷积核。k 在

$DenseNet$ 称为 $growth\,rate$，这是一个超参数。一般情况下使用较小的 k，就可以得到较佳的性能。假定输入层的特征图的 $channel$ 数为 k_0 ，那么 l 层输入的 $channel$ 数为 $k_0 + k(l-1)$ ，因此随着层数增加，尽管 k 设定得较小，DenseBlock 的输入会非常多，不过这是由于特征重用所造成的，每个层仅有 k 个特征是自己独有的。

由于后面层的输入会非常大，$DenseBlock$ 内部可以采用 $bottleneck$ 层来减少计算量，主要是原有的结构中增加 $1\times1\,Conv$，即 $BN+ReLU+1\times1\,Conv+BN+ReLU+3\times3\,Conv$，称为 $DenseNet-B$ 结构。其中 $1\times1\,Conv$ 得到 $4k$ 个特征图它起到的作用是降低特征数量，从而提升计算效率。

对于 $Transition$ 层，它主要是连接两个相邻的 $DenseBlock$，并且降低特征图大小。$Transition$ 层包括一个1×1 的卷积和 2×2 的 $AvgPooling$，结构为 $BN+ReLU+1\times1\,Conv+2\times2\,AvgPooling$。另外，$Transition$ 层可以起到压缩模型的作用。假定 $Transition$ 的上接 $DenseBlock$ 得到的特征图 $channels$ 数为 m ，$Transition$ 层可以产生 $\lfloor\theta m\rfloor$ 个特征（通过卷积层），其中 $\theta\in(0,1]$ 是压缩系数（compression rate）。当 $\theta=1$ 时，特征个数经过 Transition 层没有变化，即无压缩，而当压缩系数小于 1 时，这种结构称为 $DenseNet-C$，文中使用 $\theta=0.5$ 。对于使用 $bottleneck$ 层的 $DenseBlock$ 结构和压缩系数小于 1 的 $Transition$ 组合结构称为 $DenseNet-BC$。

本项目分别尝试了 $DenseNet121$ 、$DenseNet169$、$DenseNet201$、$DenseNet161$。定义 $DenseBlock$，$DenseBlock$ 内部采用 $bottleneck$ 层，$bottleneck$ 层中 $BN+ReLU+1\times1\,Conv+BN+ReLU+3\times3\,Conv$ ， 而 $Transition$ 层 $BN+ReLU+1\times1\,Conv+Pooling$，对于不同的 $DenseNet$，k 分别取以下值：

$DenseNet121$	$k=32$
$DenseNet169$	$k=32$
$DenseNet201$	$k=32$
$DenseNet161$	$k=48$

3. 损失函数

交叉熵（cross entropy） 是分类问题中使用最为广泛的损失函数，其公式为

$$L = -[y log \hat{y} + (1 - y) \log(1 - \hat{y})]$$

PaddlePaddle 提供了可用于计算硬标签或软标签的交叉熵的函数：

1）硬标签交叉熵算法： $label[i_1, i_2, \dots, i_k]$表示每个样本的硬标签值:

$$output[i_1, i_2, \dots, i_k] = -log(input[i_1, i_2, \dots, i_k, j]),$$

$$label[i_1, i_2, \dots, i_k] = j, j! = ignore_index$$

2）软标签交叉熵算法：$label[i_1, i_2, \dots, i_k, j]$表明每个样本对应类别$j$的软标签值:

$$output[i_1, i_2, \dots, i_k] = -\sum_j label[i_1, i_2, \dots, i_k, j] \times log(input[i_1, i_2, \dots, i_k, j])$$

损失函数定义了拟合结果和真实结果之间的差异，作为优化的目标直接关系模型训练的好坏，输出层收到反馈，通过反向传播将反馈传向隐含层，并更新连接的权值。

4. 优化器

神经网络最终是一个最优化问题，在经过前向计算和反向传播后，Optimizer 使用反向传播梯度，优化神经网络中的参数。

1） SGD/SGDOptimizer

SGD 是实现随机梯度下降的一个 Optimizer 子类，是梯度下降大类中的一种方法。当需要训练大量样本的时候，往往选择 SGD 来使损失函数更快的收敛。

$$param\_out = param - learning\_rate \times grad$$

2） Momentum/MomentumOptimizer

Momentum 优化器在 SGD 基础上引入动量，减少了随机梯度下降过程中存在的噪声问题。

该优化器含有牛顿动量标志，公式更新如下：

$$velocity = \mu \times velocity + gradient$$
$$if(use\_nesterov):$$
$$\quad param = param - (gradient + \mu \times velocity) \times learning\_rate$$
$$else:$$
$$\quad param = param - learning\_rate \times velocity$$

3） Adam/AdamOptimizer

Adam 的优化器是一种自适应调整学习率的方法， 适用于大多非凸优化、大数据集和高维空间的场景。在实际应用中，Adam 是最为常用的一种优化方法。

Adam 更新如下：

$$t = t + 1$$
$$moment\_out = \beta_1 \times moment + (1 - \beta_1) \times grad$$
$$inf\_norm\_out = max(\beta_2 \times inf\_norm + \epsilon, |grad|)$$
$$learning\_rate = \frac{learning\_rate}{1 - \beta_1^t}$$
$$param\_out = param - learning\_rate \times \frac{moment\_out}{inf\_norm\_out}$$

4） Adamax/AdamaxOptimizer

Adamax 是 Adam 算法的一个变体，对学习率的上限提供了一个更简单的范围，使学习率的边界范围更简单。

Adamax 是基于无穷大范数的 Adam 算法的一个变种。Adamax 更新规则:

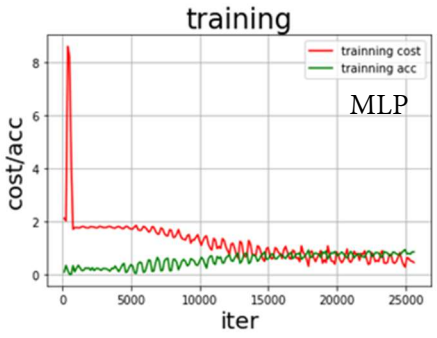
$$t = t + 1$$
$$moment\_out = \beta_1 \times moment + (1 - \beta_1) \times grad$$
$$inf\_norm\_out = max(\beta_2 \times inf\_norm + \epsilon, |grad|)$$
$$learning\_rate = \frac{learning\_rate}{1 - \beta_1^t}$$

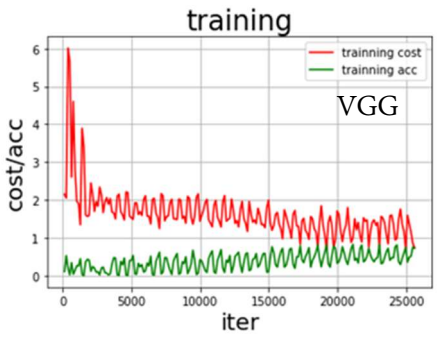
$$param\_out = param - learning\_rate \times \frac{moment\_out}{inf\_norm\_out}$$

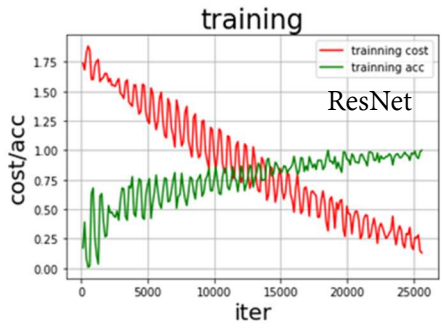
5. 实验结果

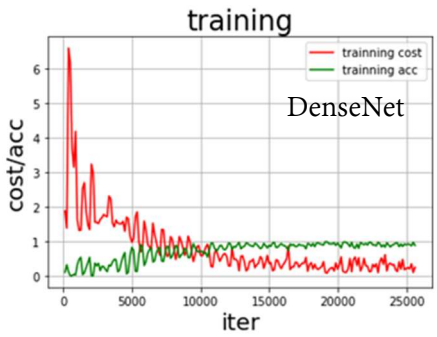
- 模型对比

	模型	损失函数	优化器	学习率	训练回合	准确率
1	LeNet-5	交叉熵损失函数	Adam 算法	0.001	40	0.90909
2	VGG	交叉熵损失函数	Adam 算法	0.001	40	0.53030
3	ResNet	交叉熵损失函数	Adam 算法	0.001	40	1.00000
4	DenseNet	交叉熵损失函数	Adam 算法	0.001	40	0.98485





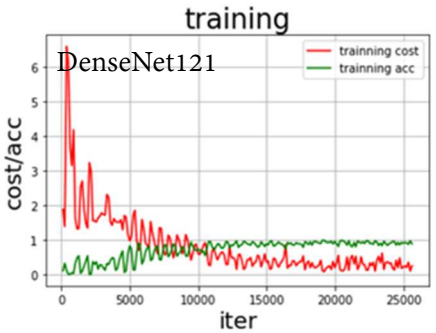


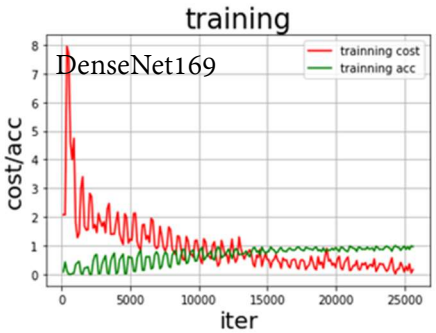


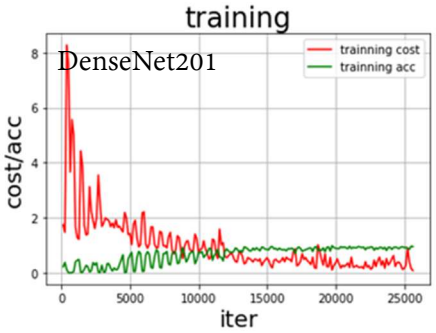
分析：在模型的选择上，按照卷积神经网络的发展，依次对 LeNet-5、VGG 、ResNet 和 DenseNet 进行了训练，可以明显地看到，网络层数的增加，并不一定能保证模型准确度的持续提高。尽管 Lenet-5 层数较少，但仍能取得较好的训练效果，而 VGG 由于层数更深， 网络收敛变得更慢，准确率也变得更差，此外，与数据集的体量也有一定的关系。而 ResNet 通过使用多个有参层来学习输入输出之间的残差表示，解决了 VGG 层数过高带来的问题，使得收敛速度更快，分类精度更高。 DenseNet 则如前面分析的那样，与 ResNet 原理基本一致，但通过增加连接对模型进行了改进，可以看到相对于 Lenet-5 和 VGG 都有了改善。

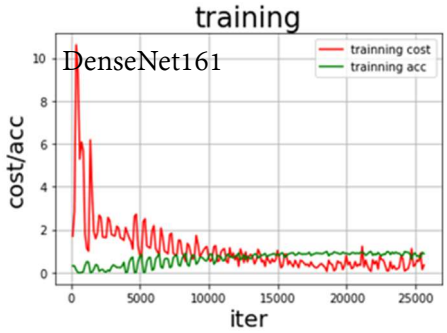
- DenseNet 层数对比

	DenseNet	损失函数	优化器	学习率	训练回合	准确率
1	DenseNet121	交叉熵损失函数	Adam 算法	0.001	40	0.98485
2	DenseNet169	交叉熵损失函数	Adam 算法	0.001	40	0.96970
3	DenseNet201	交叉熵损失函数	Adam 算法	0.001	40	0.93939
4	DenseNet161	交叉熵损失函数	Adam 算法	0.001	40	0.97990





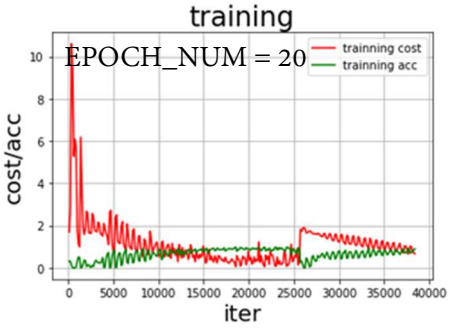


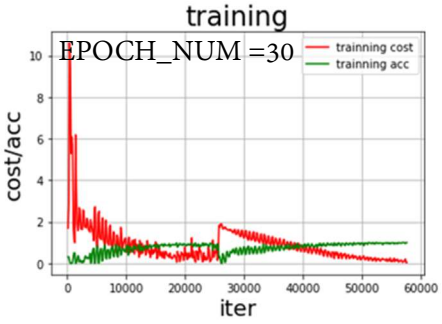


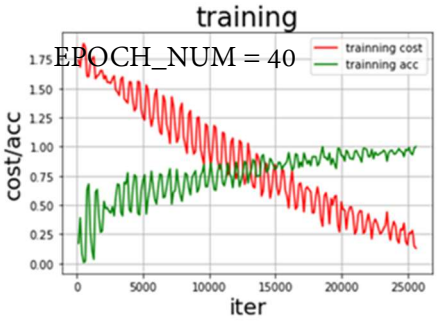
分析：根据目前的实验结果来看，随着 DenseNet 层数的增多，模型的准确率略有下降，主要考虑数据集体量较小的原因，后期可以通过在数据预处理的过程中给每个类别添加图像，增大训练数据或调整学习率、dropout 概率等参数来进行改进和测试。

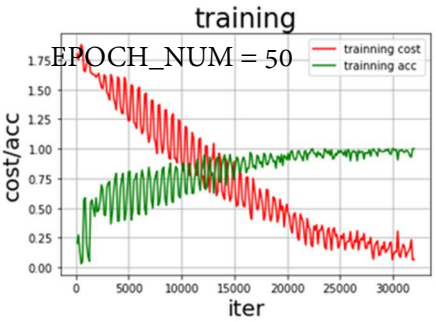
- ResNet 训练回合

	模型	损失函数	优化器	学习率	训练回合	准确率
1	ResNet	交叉熵损失函数	Adam 算法	0.001	20	0.86364
2	ResNet	交叉熵损失函数	Adam 算法	0.001	30	1.00000
3	ResNet	交叉熵损失函数	Adam 算法	0.001	40	1.00000
4	ResNet	交叉熵损失函数	Adam 算法	0.001	50	1.00000





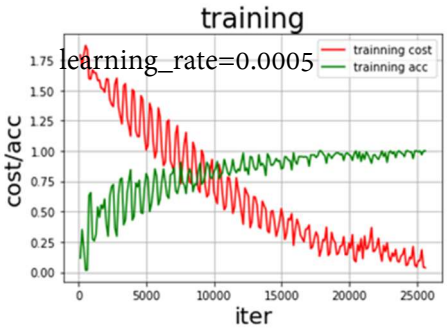


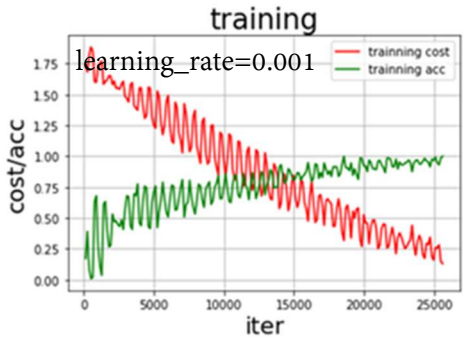


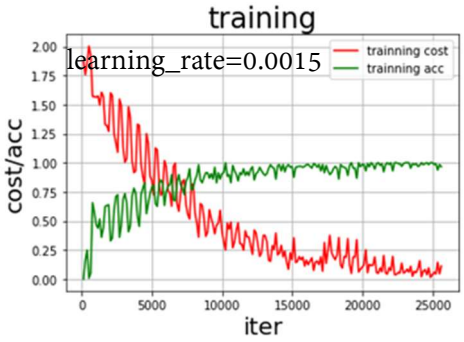
分析：在简单地调整训练回合后，可以看到 ResNet 模型在训练回合为 30 时，准确率可以上升到 1，但在调试过程（或上图）中也发现，当训练回合取不同的值（≥ 30）时，准确率会随着训练回合的增大而产生震荡，难以收敛，因此，并不意味着训练回合越大越好。

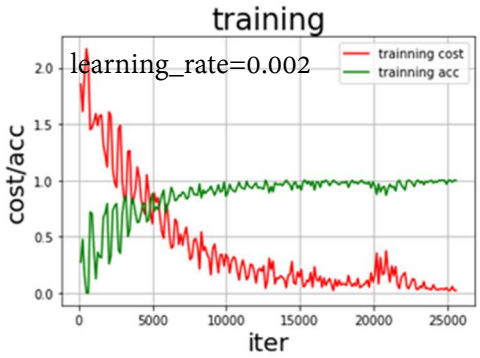
- ResNet 学习率

	模型	损失函数	优化器	学习率	训练回合	准确率
1	ResNet	交叉熵损失函数	Adam 算法	0.0005	40	1.00000
2	ResNet	交叉熵损失函数	Adam 算法	0.001	40	1.00000
3	ResNet	交叉熵损失函数	Adam 算法	0.0015	40	0.984375
4	ResNet	交叉熵损失函数	Adam 算法	0.002	40	1.00000





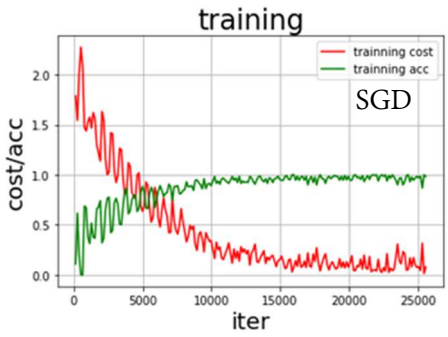


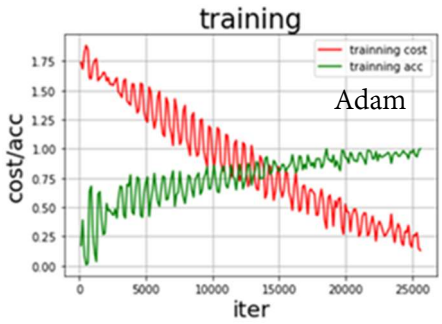


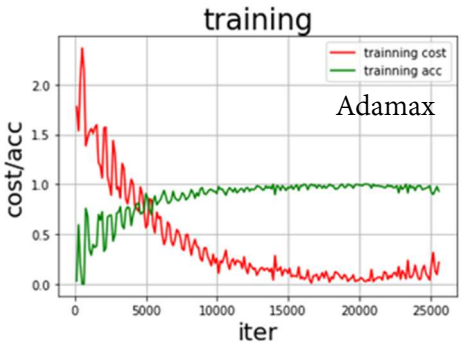
分析：在不改变优化器的前提下，对学习进行细微调整，可以看到 ResNet 模型在 0.0005~0.002 的范围内，准确率随着学习率的增大偶有下降（不排除训练时的偶然情况）。在这里，我们只探讨学习率对模型训练的影响（这一部分的比较产生的机理暂不做深入谈谈），学习率直接影响模型的收敛状态，学习率选取较小时，学习速度慢，一般在一定轮数过后使用，易使模型过拟合且收敛速度慢，而学习率选取较大时，学习速度快，一般在刚开始训练时使用，但以发生损失值爆炸和振荡的现象。

- ResNet 与优化器

	模型	损失函数	优化器	学习率	训练回合	准确率
1	ResNet	交叉熵损失函数	SGD 算法	0.001	40	1.00000
2	ResNet	交叉熵损失函数	Momentum 算法	0.001	40	1.00000
3	ResNet	交叉熵损失函数	Adam 算法	0.001	40	1.00000
4	ResNet	交叉熵损失函数	Adamax 算法	0.001	40	1.00000







分析：随机梯度下降(SGD)是最经典的方法，其他的优化器，都是在这个基础上修改完善得来的，Momentum 优化器在 SGD 基础上引入动量，减少了随机梯度下降过程中存在的噪声问题，而 Adam 综合了 Momentum 的更新方向策略和 RMProp 的计算衰减系数策略，是实际应用中最为常用的一种优化方法。通过对比结果，可以发现，上述方法在其他条件相同的情况下，模型的准确度基本没有变化，但是例如 Adamax 在训练过程中可能会有震荡。

6. 总结与收获

本次实验在上一次实验的基础上，再次学习了卷积神经网络的发展历程，并在 ResNet 的基础上进一步探讨了 DenseNet 的基本原理和简单实现，在通过对 ResNet 和 DenseNet 的简单对比，有了初步的理解。

PaddlePaddle 提供的深度学习框架和各种接口，对初学者来说很容易上手，可以根据文档给出的案例在自己的项目中进行编码、调参。本次项目给我带来的收获一方面是通过编码实现神经网络的配置，对神经网络的模型有了更深的印象和理解，另一方面是对于几种经典的卷积神经网络的原理和其优劣有了更进一步的理解。

同时，本次实验中也存在着一些不足，例如 DenseNet 在层数增加时准确率下降的原因，和在 ResNet 中调整学习率和优化器对训练结果影响的机理，在本次实验中没有得到探讨，希望在有时间的条件下后续的工作可以展开。