人工智能基础

第二次大作业

——基于深度学习的图像分类

班级：自 71

姓名：屈晨迪

学号：2017010928

名字：QuChendi **队名：**Sirius
Public：分数 0.90200 排名 57
Private：分数 0.88942 排名 76[1]

[1] 此处填写的是自己的分数，相关情况已与助教@Bohdan 说明，谢谢助教们谅解！

1 问题描述

本次作业需要利用深度学习的方法对 10 类图片进行分类，图片类别及示例如图 1 所示。提供的数据包含 30000 张带类别标签的图片组成的训练集，和 5000 张无类别的测试集，需要用训练好的模型对测试集图片进行分类，并将结果生成 csv 文件上传提交。选用 python 编写网络架构，深度学习框架在 pytorch/tensorflow/caffe 中任选其一。

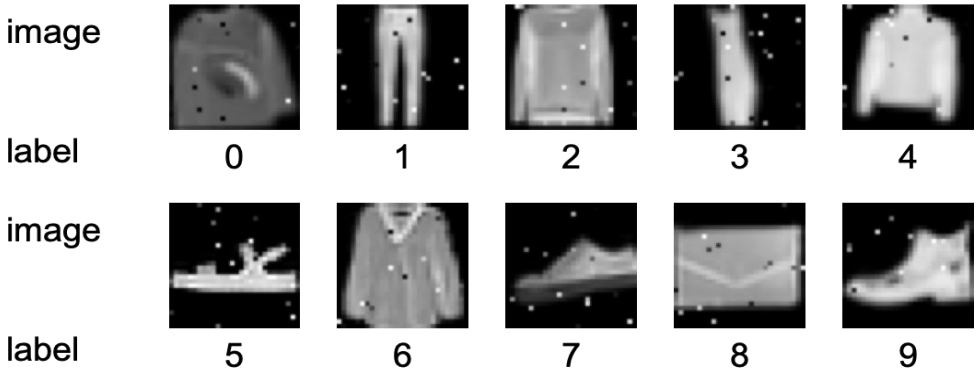


图 1 图片类别及示例

2 前期工作准备

（1）配置编译环境

在 anaconda3 上配置 python3.6 的新环境，安装 pytorch 框架，使用 jupyter notebook 编辑、运行代码。

（2）加载数据

使用 numpy.load 读取.npy 文件，pandas.read_csv 读取.csv 文件；创建 ImgDataset 类，用于载入和读取图片和类别标签；创建 DataLoader 迭代器，一个 batch 的大小设为 32，用于分批读出数据。

测试网络的过程中，为了更方便地判断、比较网络性能，从 30000 张训练图片中随机分出 5000 张作为验证集，每个 echo 结束后做一次测试和验证，从验证集的损失和精确度中能基本预测测试集的准确率。

（3）GPU 加速

本次作业涉及较大型的网络，网络训练中采用了 gpu 加速，获得更快的训练速度。

3 网络模型介绍及设计

在完成本次作业的过程中，笔者前后测试、改进过多种网络结构，现分别介绍如下。

3.1 简单的卷积网络

对于不甚复杂的图像分类问题，简单的卷积网络有时也能呈现较好的效果。以课程助教给的源代码为例，其网络架构如图 2 所示。

假设输入图片大小为 28*28，可以看到该网络首先对图片做了一次卷积操作，该操作包

含 10 个卷积核，每个卷积核的大小为 5*5，且未做padding补零，图片变为 24*24*10 大小，进行一次步长为 2 的池化，图片大小变为 12*12*10，之后采用 relu 函数做激活；接着再做一次包含 20 个卷积核、每个核大小为 5*5 的卷积操作，图片变为 8*8*20，再进行一次步长为 2 的池化，图片变为 4*4*20，包含 320 个值，再用 relu 函数激活一次；之后对图像做线性变换、激活、线性变换三步，获得输出。

```
def __init__(self):
    super(Model,self).__init__()
    self.conv1 = nn.Conv2d(1, 10, kernel_size=5)
    self.conv2 = nn.Conv2d(10, 20, kernel_size=5)
    self.fc1 = nn.Linear(320, 50)
    self.fc2 = nn.Linear(50, 10)

def forward(self, x):
    x = F.relu(F.max_pool2d(self.conv1(x), 2))
    x = F.relu(F.max_pool2d(self.conv2(x), 2))
    x = x.view(-1, 320)
    x = F.relu(self.fc1(x))
    #x = F.dropout(x, training=self.training)
    x = self.fc2(x)
    return x
```

图 2 简单的卷积网络

仅采用上述网络模型，损失函数选择交叉熵，学习率设为 0.1，循环 30 个 echo，输出结果如图 3 所示，可以看到，训练集的损失逐步减小，准确率上升，而验证集的损失和准确率波动均较大，在 20 个 echo 左右逐渐趋于平稳，且从 20 个 echo 开始，观察到验证集的损失开始回升，有出现过拟合的趋势。最终验证集的准确率在 0.82 上下稳定。

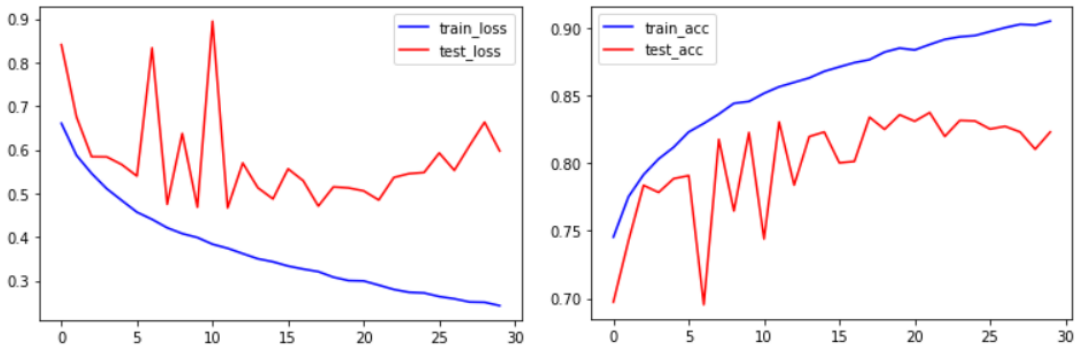


图 3 简单卷积网络的损失和准确率

在此种网络架构下，可以增多卷积操作，采用多步卷积后接一步池化的方法，经测试能够有效提升准确率。但网络结构整体而言较为简单，难以将准确率提到很高。

3.2 ResNet

笔者在选定 ResNet 网络做测试之前，对几种经典的神经网络如 AlexNet、VGG 等做了了解。在 VGG 网络中，为了达到更好的效果，显著增加了网络深度，但网络深度的增加带来了其他的问题，例如梯度爆炸或消散，这是指随着层数增多，反向传播的梯度会变得极不稳定，出现过大或过小的情况，导致网络性能不升反降，而 ResNet 正是针对此问题提出的一种解决方案。

ResNet 最核心的部分是引入了残差学习模块，如图 4 所示，其中 x 表示浅层网络的输出，$H(x) = F(x) + x$表示深层网络的输出，$F(x)$为中间两层的变换函数。残差学习模块在检测到浅层输出 x 已经较优，即再做改变 loss 均会增大，此时会将$F(x)$设为 0，相当于一个从浅层到深层输出的直接连接，即恒等映射，这样保证了深层网络的训练结果不会比浅层差；而在反向传播的过程中，残差模块会减小模块中的参数值从而使模块对损失变动更为敏感，

这样能有效避免梯度爆炸或消失的问题。

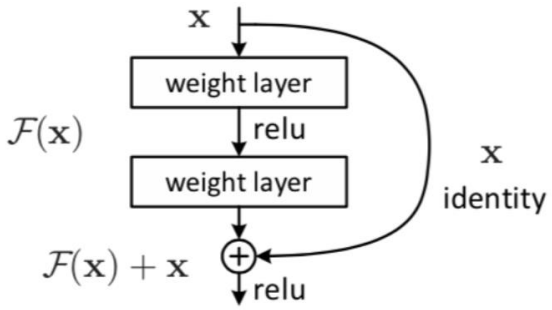


图 4 残差学习模块

本次作业中的 ResNet 模块参考了 pytorch 官网[2]上给出的源代码，ResNet 不同层数的网络配置如图 5，笔者主要测试了 18 层、34 层和 50 层的模型，较深层的网络规模过大，在本例中有大材小用之嫌。基础的 18 层和 34 层中，最重要的残差模块即Bottleneck由两个 3*3 的卷积核构成，可以看到，ResNet 网络首先对图像进行 7*7 的卷积和一步 3*3 的池化，再经过 4 个Bottleneck阶段，最后再经过一个 1*1 的卷积，得到输出。

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

图 5 ResNet 不同层数的网络配置

3.3 SE-ResNet

SE 模块作为一种网络单元可以嵌入多种网络结构中，将其加入 ResNet 的Bottleneck模块中，即获得 SE-ResNet 网络模型，核心结构如图 6 所示。可以看到 SE 模块包含了一层池化，两次 FC 和 ReLu 激活，以及一次 Sigmoid，其中的 FC 结构相当于一个 1*1 的卷积，SE 模块考虑了通道之间的相关性，可以有效提高网络性能，并且能够一定程度上减小了参数量和计算量。

[2] 见参考文献[2]

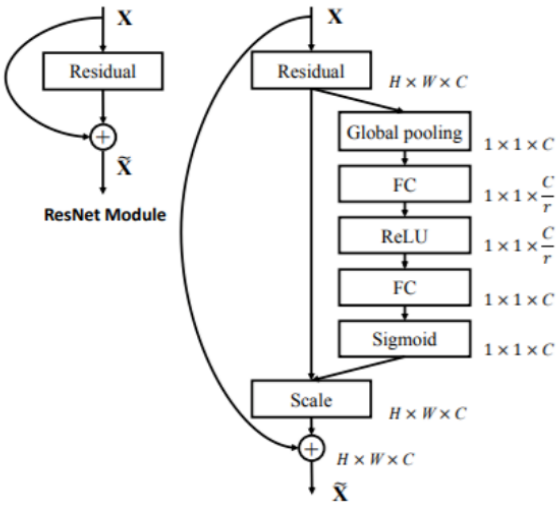


图 6 SE-ResNet 核心结构

4 模型优化和参数选择

4.1 优化方法

（1）针对 ResNet 的调整

由于 ResNet 在一开始是针对较大的 RGB 图片设计的网络模型，在输入网络前将 18*18 的图片调整到 224*224[3]的大小，再扩展到三个维度，效果较小图片有一定程度的提升，并且观察到将图片放大后，图像上的噪声点有所去除。

（2）数据增强

在使用性能较好的网络，如 ResNet 时，若不做任何预处理，会很容易出现过拟合的现象，即训练集的损失和准确率均在上升，但测试集的损失停止下降甚至回升，准确率也不再变化，针对此种现象可以有多种处理方法，其中一个即数据增强。

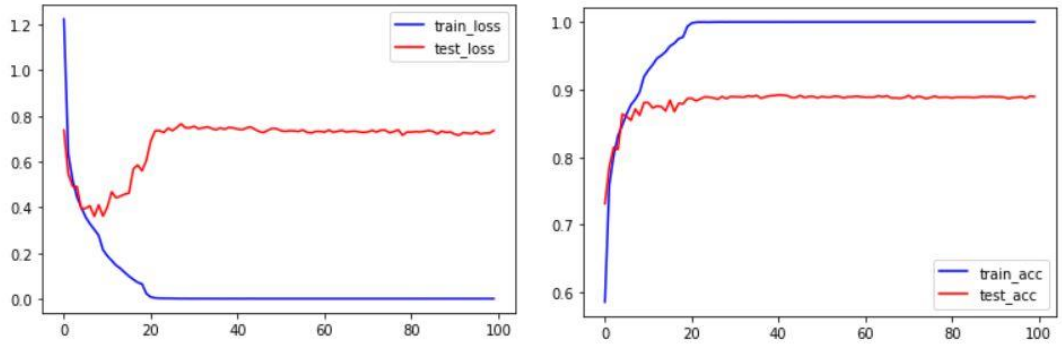
数据增强是指通过对图片做放大缩小、翻转、裁剪、归一化等操作，达到扩大数据集的目的，在有限的数据集下避免过拟合的问题。在 Pytorch 框架下可以使用 torchvision.transformas 包方便地实现数据预处理，本次作业中的数据预处理如图 7 所示，其中包括随机垂直或水平翻转、随机裁剪和归一化。需要注意的是，其中随机裁剪和归一化对训练集和测试集均要进行操作，但随机翻转只需对训练集做，这是笔者实验比较得到的结论，深层原因尚不可知。

```
im_aug = transforms.Compose([
    transforms.ToPILImage(),
    transforms.Resize(256),
    transforms.RandomHorizontalFlip(),
    transforms.RandomVerticalFlip(),
    transforms.RandomCrop(224),
    #transforms.ColorJitter(brightness=0.5, contrast=0.5, hue=0.5),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
 ])
data = im_aug(data)
```

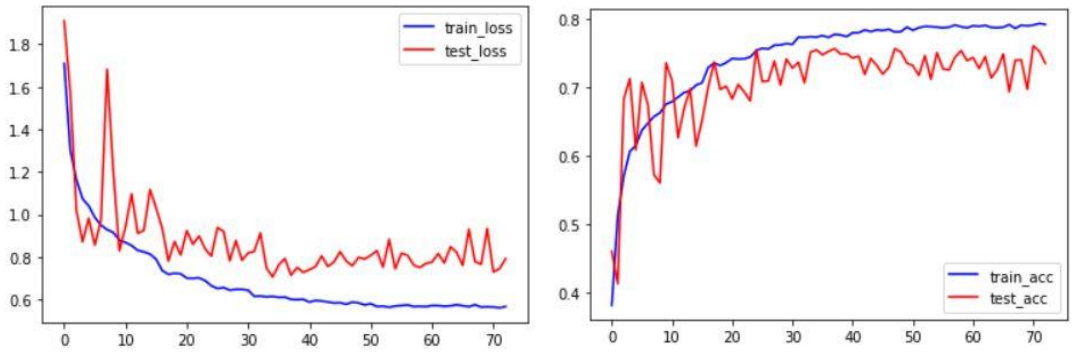
图 7 数据增强操作

[3] 也可以调整到 256*256，在随机裁剪中裁剪至 224*224

以 ResNet34 为例，在相同的学习率设置下，未做数据增强的结果如图 8(a)所示，可以看到其验证集的损失在 10 个 echo 之后停止下降开始上升，准确率不再变化，说明出现了过拟合，在加入了数据增强之后，如(b)中结果，验证集的损失下降速度变缓，并且未出现明显回升，精确率的上升也会放缓[4]。



(a)未做数据增强



(b)经过数据增强

图 8 对比数据增强性能

4.2 参数调整

（1）学习率

笔者在实验中发现，学习率的变动对准确率有较大的影响，学习率越小，梯度下降越慢，收敛越慢，但能有效降低损失，如果学习率持续较大，会在极值附近震荡，难以达到全局最优。较好的情况是，在训练初始时有较大的学习率，随着训练 echo 的增多，学习率逐步降低。采用动态调整学习率的方法，在 pytorch 的 optimizer.lr_scheduler 下进行设置，笔者共尝试了两种调整方法，一种是经过特定次数的 echo 学习率降低至原来的 m 倍（m<1），另一种是根据验证集的损失或准确率调整，当损失不再降低或准确率不再有提升至一定次数的 echo 时，学习率降低至原 m 倍。

最终采用方法二，初始学习率设为 0.1，当准确率在 4 个 echo 内没有上升时，降低至原来的 0.5 倍，得到了较好的效果。

（2）动量和权重衰减

在每次更新的时候，设定动量（momentum）可以对梯度方向与上一次方向相同的参数进行一定程度的加强，对不同的参数做减弱，经过测试和线上查询，选定 momentum=0.9；权重衰减（weight-decay）可以有效减轻过拟合现象，较小的 w 值可以通过抑制导数值，使

[4] 此情况下较明显的波动是学习率设置不当所致

拟合曲线不能很好的贴合所有点，从而防止过拟合，本作业中选定 weight-decay=0.00004.

（3）卷积核大小

使用 3 中介绍的简单卷积网络对卷积核大小进行测试，相同参数和 echo 下，5*5 的卷积核效果如图 3，3*3 的卷积核效果如图 9 所示，对比可以看出，5*5 的卷积核下验证机的损失和准确率波动较大，在接近 30 个 echo 处出现过拟合趋势，而使用 3*3 的卷积核时波动显著减小，在 30echo 内未出现过拟合趋势，收敛速度较慢。

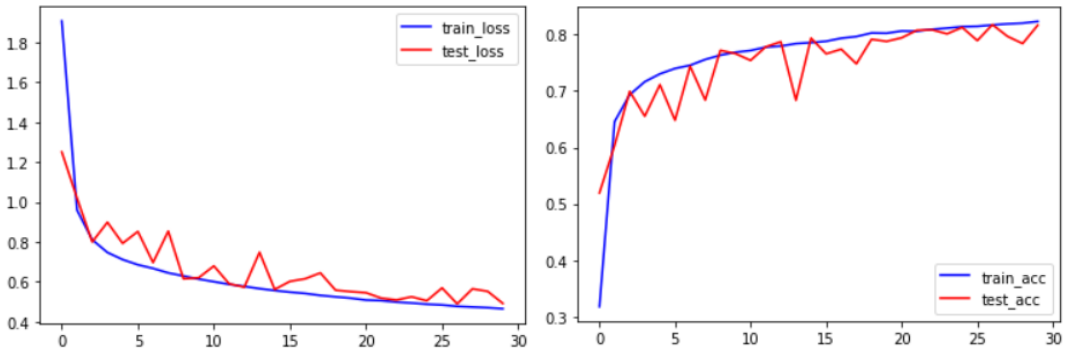


图 9 简单卷积网络中使用 3*3 卷积核

观察经典的大型网络，如 ResNet，可以看到其中多使用 3*3 的卷积核，在 VGG 网络中，使用 3 个 3*3 的卷积核与一个 7*7 的卷积核能收获同样大小的感受野，笔者在线上搜索时，看到数个实验表明小卷积核往往有更好的效果，但也不绝对，还是应根据网络实际情况而定。

5 总结与心得体会

本次大作业最终使用的网络是 SE-ResNet34，循环 80 个 echo，测试集准确率为 0.902，实验结果如图 10 所示。

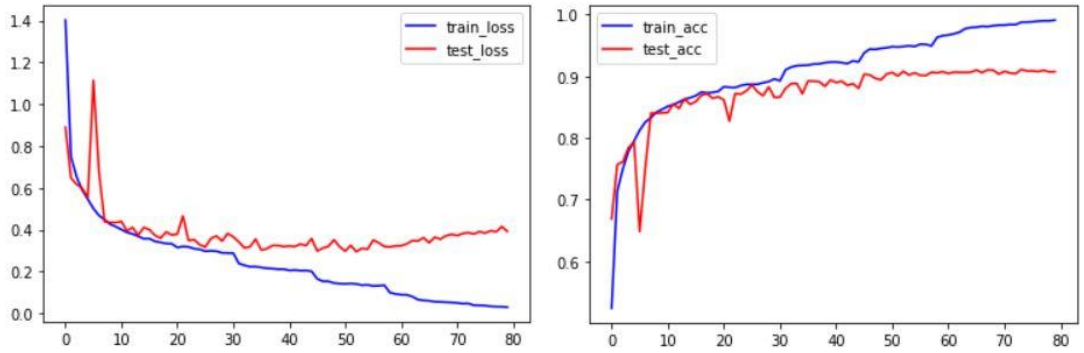


图 10 最终结果

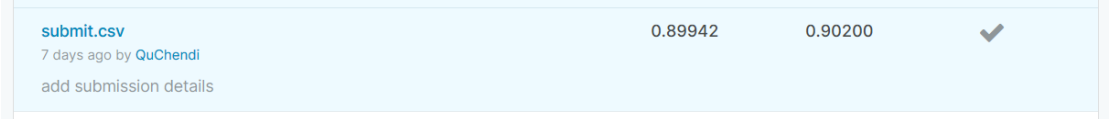


图 11 Kaggle 得分

本次大作业是笔者真正意义上第一次接触 python 编程和神经网络，从 anaconda 的安装、环境配置，到学习使用 pytorch 框架、编写数据集的 DataLoader，从最基础的网络结构开始读懂代码，到使用不同的经典网络架构做测试、学习参数调整的经验，甚至服务器、GPU 的连接和配置，每一步回想起来都异常艰难，但不失为一次收获颇丰的体验，最终虽然在 kaggle 榜上没有排到很靠前的位置，但也达到了期望。刷榜的过程也让我体会到了沟通交流的重要，对自己了解不深的问题，在与同学的交流中能获得更多的经验方法，效率更高。最

后，谢谢老师和助教对我本次大作业的帮助！

6 参考文献

• ResNet：

[1] https://blog.csdn.net/weixin_43624538/article/details/85049699

[2] https://github.com/pytorch/vision/blob/master/torchvision/models/resnet.py

[3] https://www.cnblogs.com/wzyuan/p/9880342.html

[4] https://arxiv.org/pdf/1512.03385.pdf

• SE-ResNet

[5] https://blog.csdn.net/xzy528521717/article/details/86582889

• 数据增强

[6] https://blog.csdn.net/u010801994/article/details/81914716

[7]https://blog.csdn.net/lwplwf/article/details/85776309

• 其他

[8]https://blog.csdn.net/rocling/article/details/92828799