哈尔滨工业大学

<<模式识别与深度学习>>

实验 3 基于 Pytorch 的经典卷积网络实现

(2020 春季学期)

学院：计算机科学与技术

学号：1170300909

姓名：武磊

指导老师：左旺孟

日期：2020.4.26

一、实验目的

1. 基于 Pytorch 实现 VGG/Resnet/SENet 等网络结构
   1) VGG 要求部分参照论文，可以动态调整结构
   2) ResNet 要求基于残差块，参照论文，可动态调整
   3) 在上述 VGGnet/ ResNet 的基础上添加 SEBlock
2. 基于 VGG 选择不同优化器，学习率进行对比实验
3. 基于 VGG 进行 Data Augmentation 对比分析（可以基于上述实验的最佳模型分析翻转，旋转，移位操作）

二、实验环境配置

1. 硬件：
   CPU ： Inter Core i7-7500U 2.70GHz
   GPU ： NVIDIA GeForce 940MX
2. 软件：
   操作系统:WIN10
   软件：
   Python 3.6 (基于 Anaconda)
   Pytorch 1.2.0
   CUDA 10.1 V10.1.105
   cudnn v7.6.3 for CUDA 10.1
3. 开发 IDE
   Jupyter NoteBook
   Pycharm

三、实验过程

1. 数据导入

首先在 Cifar 数据官网下载 cifar10 数据集解压，关于数据文件的说明可以参考官网：https://www.cs.toronto.edu/~kriz/cifar.html

数据处理和数据读取包 MyDataset：

1) Class Cifar10：继承 torch.utils.data.Dataset 类
   __init__(): 读取 batch 文件，加载训练集和测试集数据
   __getitem__(): 根据 index 获取对应标号的数据和相应标签
   __len__(): 获取数据大小
2) Class Totensor():将数据转换成 torch.tensor.().float()
3) 也可以直接使用 kaggle 的 kernel 进行网络训练，获得超强算力。

2. 选择代价函数，超参数，优化器

1) 模型的超参数
   模型主要的超参数主要有以下因素：
   1. 学习率
   2. Batch size
   3. 网络结构，主要要有网络层数，网络结构设计等等

2) 代价函数选择

分类问题，采用交叉熵作为代价函数。

3) 优化器选择

主要选用 Aadm 和 SGDM 作为优化器。在实验过程中做对比实验。

3. 设计网络模型

本次实验中我们主要实现 VGGNet16，Resnet34，并在其基础上添加 SEBlock 结构。网络的结构参考相应论文。

1) VGGnet

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

上图中是 VGGnet 六种不同的结构。

- 对比 Alexnet，VGGnet 的卷积层参数设置很简单，统一采用 3*3 的卷积核，stride=1，padding=1；在下采样的过程中使用 2*2 的 kernel，stride=2，2 倍的下采样。
- VGGnet 的网络更深，采用多层垛堞，小卷积核的模式，能够在增加网络层数的同时，控制参数的增长。

2) Resnet

残差网络的提出主要是在神经网络到后期，为了解决网络深度大幅增加出现训练难度增加，网络效果反常变差的现象。即是要**解决深度网络出现的“退化”现象**。

作者在论文中提出了残差块（Residual Block）来解决退化的问题。

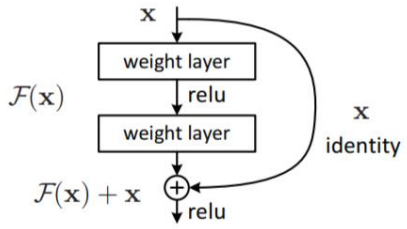


Figure 2. Residual learning: a building block.

- 对比 VGG，Resnet 的层数更深，但是作者认为残差块的设计能够缓解深层网络难以训练的问题。
- Resnet 实际上就是将 vggnet 中 conv 层的垛堞换成了残差块的垛堞，同时减少了全连接的层数。
- Resnet 的下采样过程没有采用一般的池化，而是选择在卷积的过程中增大 stride 实现。
- Resnet 通过 shortcut 有效的利用了之前卷积提取的 feature；同时在残差块的输出通过 global Avgpooling，输出（512*1*1）的特征向量；只采用一层全连接，就输出结果。同时上述三种实现细节都大幅减少了参数，同时提升的网络性能。

  实验发现：
  - ✓ 如果采用自适应池化按照（32-16-8-4-2）的层次下采样，网络的性能瓶颈为 test_acc=76%，同时会出现过拟合现象。
  - ✓ 如果最后残差输出没有经过全局池化，网络性能瓶颈 test_acc=83%
  - ✓ 使用小 batch32 训练能够达到 90%性能

3) SEBlock

SEBlock 的想法来自于 Squeeze and Excitation Net。作者提出了一种叫做 SEBlock 的结构能够有效的动态考虑 channel-wise feature recalibration，能够有效的提升网络的表示能力。实际上就是在 channel 层面上做了特征的 attention 机制（不知道我的理解是否正确）。

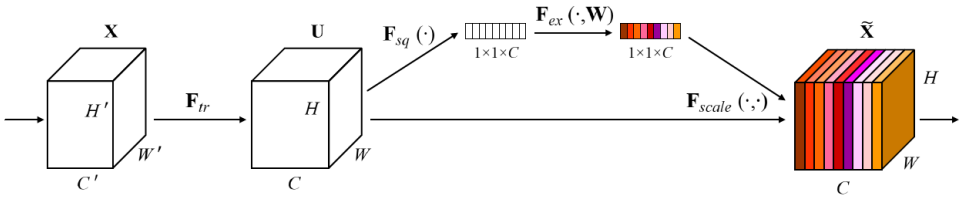


Fig. 1. A Squeeze-and-Excitation block.

在计算复杂度上面，根据论文提供的对比实验，添加 SEBlock 能够在增加微小的计算开销获得可观的性能增益。作者在 SE 过程中使用了 reduction ration（recommended r = 16）,有效的减少了计算和参数的复杂程度。

a) SEVGGnet

有两种实现方式：

- 对卷积层做扩展，每一层卷积之后都添加 SEBlock。
  毫无疑问，这样所带来的的高计算复杂性和参数量和性能增益应该有 tradeoff。
- 对原网络的每一层 layer 做扩展，一层 layer 之后添加一个 SEBlock。
  比方法一的做法能够用更少的参数和计算复杂度。

因为时间原因，本次实验中只训练了实现较为简单的第一种方式。

b) SEResnet

采用论文中的实现方式，对原实现做扩充：

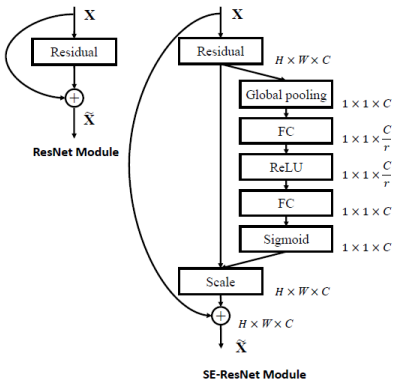


Fig. 3. The schema of the original Residual module (left) and the SE-ResNet module (right).

四、结果分析

1. 网络性能对比

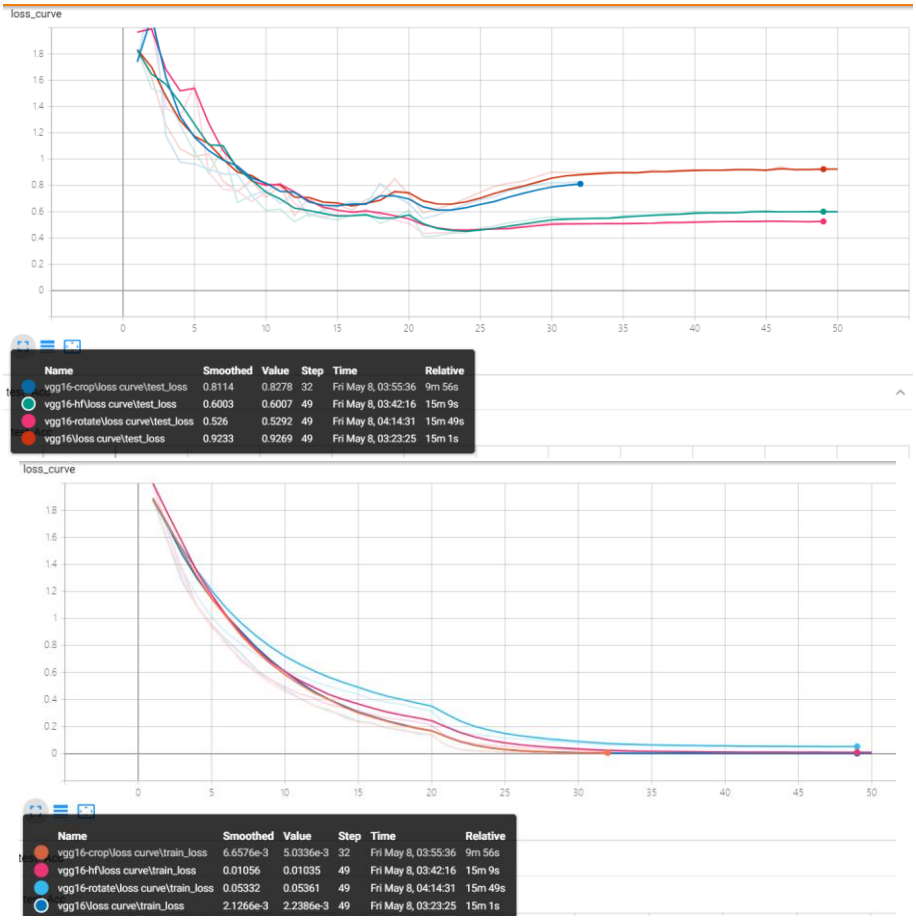
网络	Epoch	Batchsize	Lr	优化器	Accuracy
VGG16	50	256	0.005	Adam	87.50%
Resnet18	50	256	0.005	Adam	85.62%
SE-VGG16-1	50	256	0.005	Adam	87.42 %
SE-VGG16-2	50	256	0.005	Adam	87.04 %
SE-Resnet18	50	256	0.005	Adam	85.19%

注释：

- 上述训练过程均使用 MultiStepLR(optimizer,[20,30,40],0.1)用于训练过程中动态更新学习率。
- 上述训练均未使用数据增广技术

2. 数据增广对 VGG 训练的增益

网络	Epoch	Batchsize	Lr	优化器	Accuracy
VGG16	50	256	0.005	Adam	87.50%
Add HF	50	256	0.005	Adam	89.53%
Add Rotate	50	256	0.005	Adam	88.80 %
Add RandomCorp	50	256	0.005	Adam	87.53%



实验发现数据增广主要帮助数据在 test_loss 上获得更好的下降，即提高了模型的鲁棒性。

实验发现并不是所有的数据增广方式都能够有效，至少从 test_acc 反映情况确实如此。但是这也是有 test_acc 的样本决定的。对于实际情况而言，除非限定输入图片的形式，否则多样的数据增广方式如翻转，裁剪，色彩变换，旋转是能够显著提升模型的鲁棒性和泛化能力的。

选择的数据增广方式以所需要的模型决定：是希望得到一个对于具有某个规定的样本表现优秀的模型还是得到一个对大多数样本（随机分布）都还不错的模型。

3. 不同学习率和优化器对比

受限于实验训练时间的限制，只能做较为粗糙的简单对比。

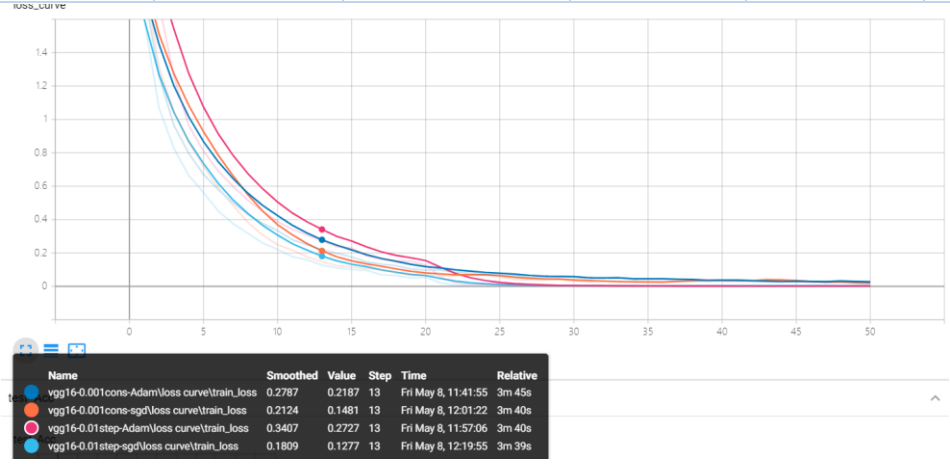
我们发现在任意情况下 Adam 都能获得一个较为鲁邦的收敛效果，在前期确实有加速收敛的作用。但是后期由于梯度过小导致学习缓慢甚至无法完全收敛。

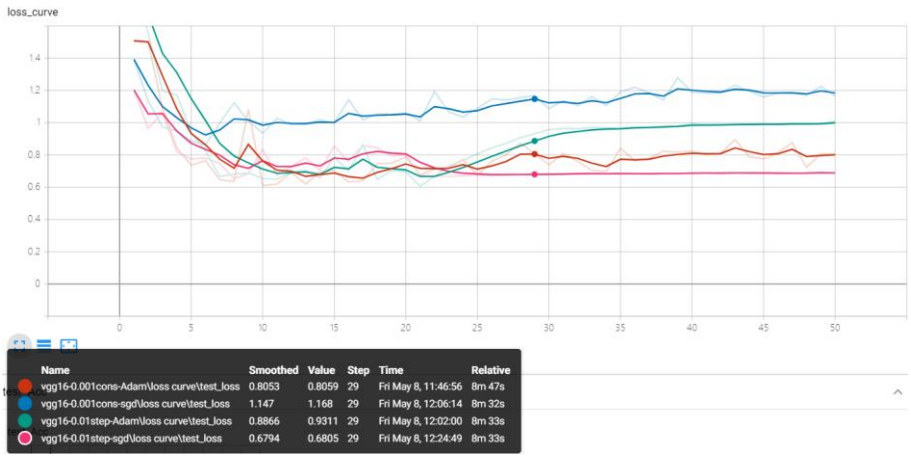
SGDM 受到学习率的影响很大，而且 epoch 之间可能出现较大波动，从下图中可以看出，如果适当的动态调整学习率，能够获得更好的训练效果。

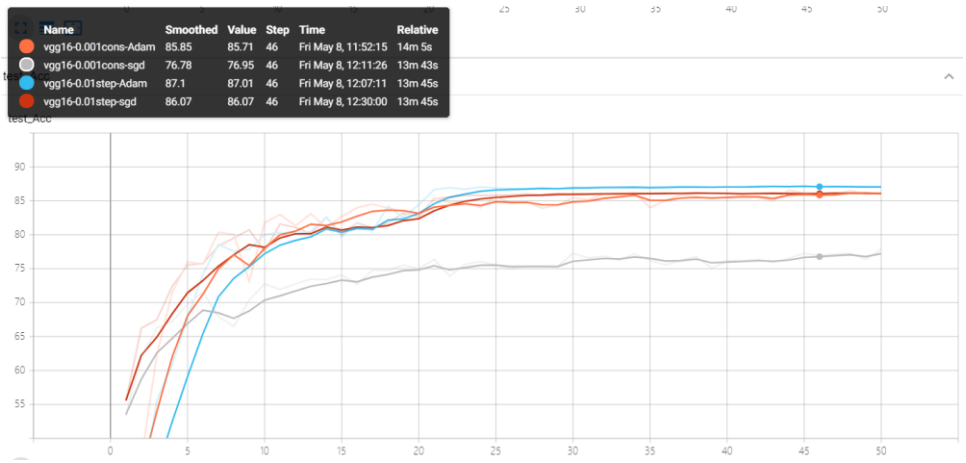
- 实际上在多次实验中，我们发现如果采用小 batch（32,64）multistep 学习率的手段，能够获得更好的效果。但是如果训练样本过大，小 batch 花费时间较长，这里统一采用 bs=256。

- 实验中分别采用了 32（88.0%）,128（86.0%）,256（84.3%）等不同的 batchsize 大小，发现小 batch 的效果更好。同时实验中也发现了实验二中的跃变现象，但是这次实验中我们并没有在一次实验中变换 batch 的大小。
  所以合理的解释是：实验 2 中对于 batch 对性能的提升也是由于改变 batch 进而影响学习率，达到更好的收敛？
  但是 trainloss 一直在下降，只有 testloss 出现的过拟合被减小学习率抑制。

网络	Epoch	Batchsize	Lr	优化器	Accuracy
VGG16	50	256	0.001const	Adam	86.05%
VGG16	50	256	0.01step	Adam	87.06%
VGG16	50	256	0.001const	SGDM	77.89 %
VGG16	50	256	0.01step	SGDM	86.21%







4. 半精度训练加速
   为了加快神经网络训练速度，Nvidia 基于 Pytorch 开发了混合精度计算的 API，能够快捷实现混合精度计算，提升计算的速度，隐形增加显存。

```
1. from apex import amp
2. model, optimizer = amp.initialize(model, optimizer, opt_level="O1")
3. with amp.scale_loss(loss, optimizer) as scaled_loss:
4.     scaled_loss.backward()
```

五、总结

1. 多看论文，理论指导实验。
2. 在上述实验中我们主要实现了四种网络结构，并且做了相应的调整，但是还有两方面的问题（没有添加数据增广）,网络实现是没有问题的，可能是训练的问题：
   a) 添加了 SEBLock 之后，原有网络的效果没有提升，反而有轻微的下降。
   b) Resnet 的优势没有发挥出来，尝试 resnet34，效果没有 resnet18 好，这是不符合实验预期的，需要在后续阶段改进。
3. 添加了数据增广之后，resnet 和 seresnet 都有很好的提升，猜想深度网络需要用更多的数据，才能达到更好的效果
4. 比较各个网络训练的情况可以发现，如果 train_loss 下降到 0，这是不足提供新的动力帮助网络在测试集上获得更好的性能的。数据增广的方式则是通过随机产生新数据防止网络在训练集上过拟合，获得 testloss 的持续下降。

六、参考

【1】 Very Deep Convolutional Networks for Large-Scale Image Recognition
【2】 Deep Residual Learning for Image Recognition
【3】 Squeeze-and-Excitation Networks
【4】 Use Apex accelerate Pytorchhttps://zhuanlan.zhihu.com/p/79887894