

神经网络 Project 2

18300290007 加兴华

0. 说明信息

- 1.本项目用ResNet9完成了CIFAR-10分类任务，测试集准确率95.4%，同时在程序速度和参数体积上表现优秀
 - 2.本项目在VGG网络上对Batchnorm的效果及原理进行了分析
 - 3.本项目使对深度学习过参数化的问题进行了探索，并验证了DessiLBI在解决该问题上的效果
 - 4.本项目在实现代码的每个任务文件中都带有实验指南，记录了如何运行实验
 - 5.项目代码Repo: [DATA130011.01-records/Project 2](#)

1. CIFAR-10分类网络

1.1 数据集

CIFAR-10数据集如下，数据集中包含了图像(32x32x3)和对应的类别标签(0~9)，分别指代十种不同的对象。



1.2 网络结构

我首先复现了一些图像分类任务中的经典网络模型(ResNet18、DLA34)，然后设计了自己的模型(ResNet9、simp_ResNet9)，并控制变量初步训练网络来分析个网络的性能指标。

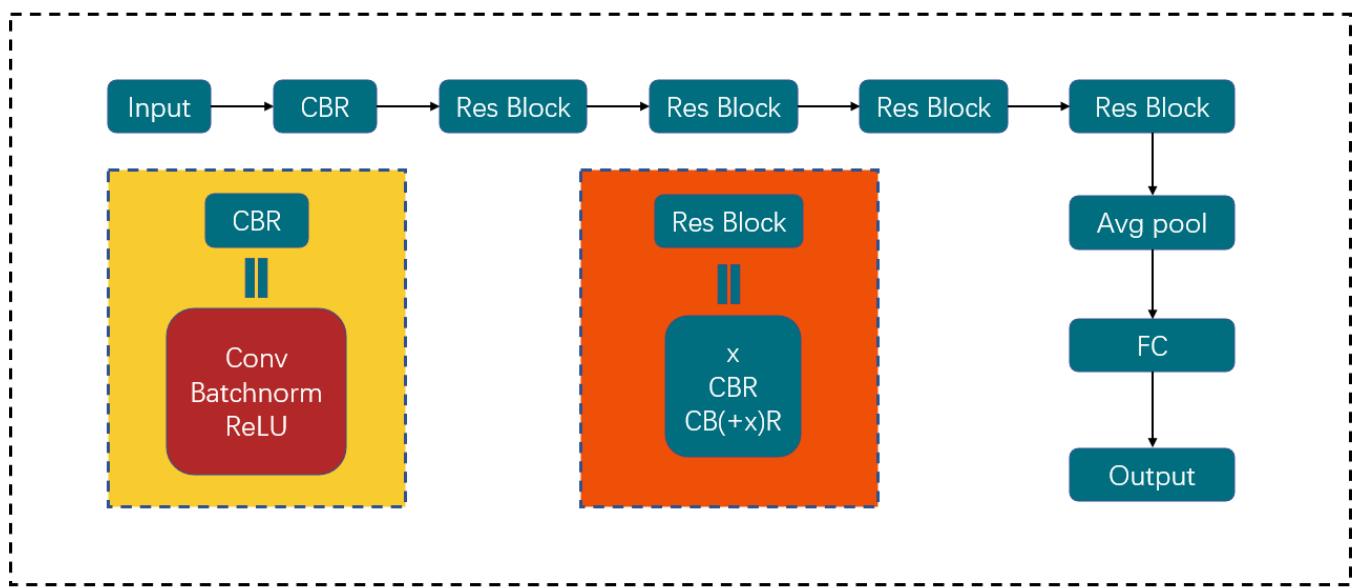
baseline: ResNet18

ResNet是何凯明等人在2015年提出的模型，主要用于改善深度学习中遇到的网络退化问题。何凯明一共设计了5种标准的ResNet(如下图)，在本项目中，我复现了其中的ResNet18网络模型。

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112			$7 \times 7, 64, \text{stride } 2$		
				$3 \times 3 \text{ max pool, stride } 2$		
conv2_x	56×56	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$
conv3_x	28×28	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 8$
conv4_x	14×14	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 36$
conv5_x	7×7	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$
	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

ResNet By 何凯明

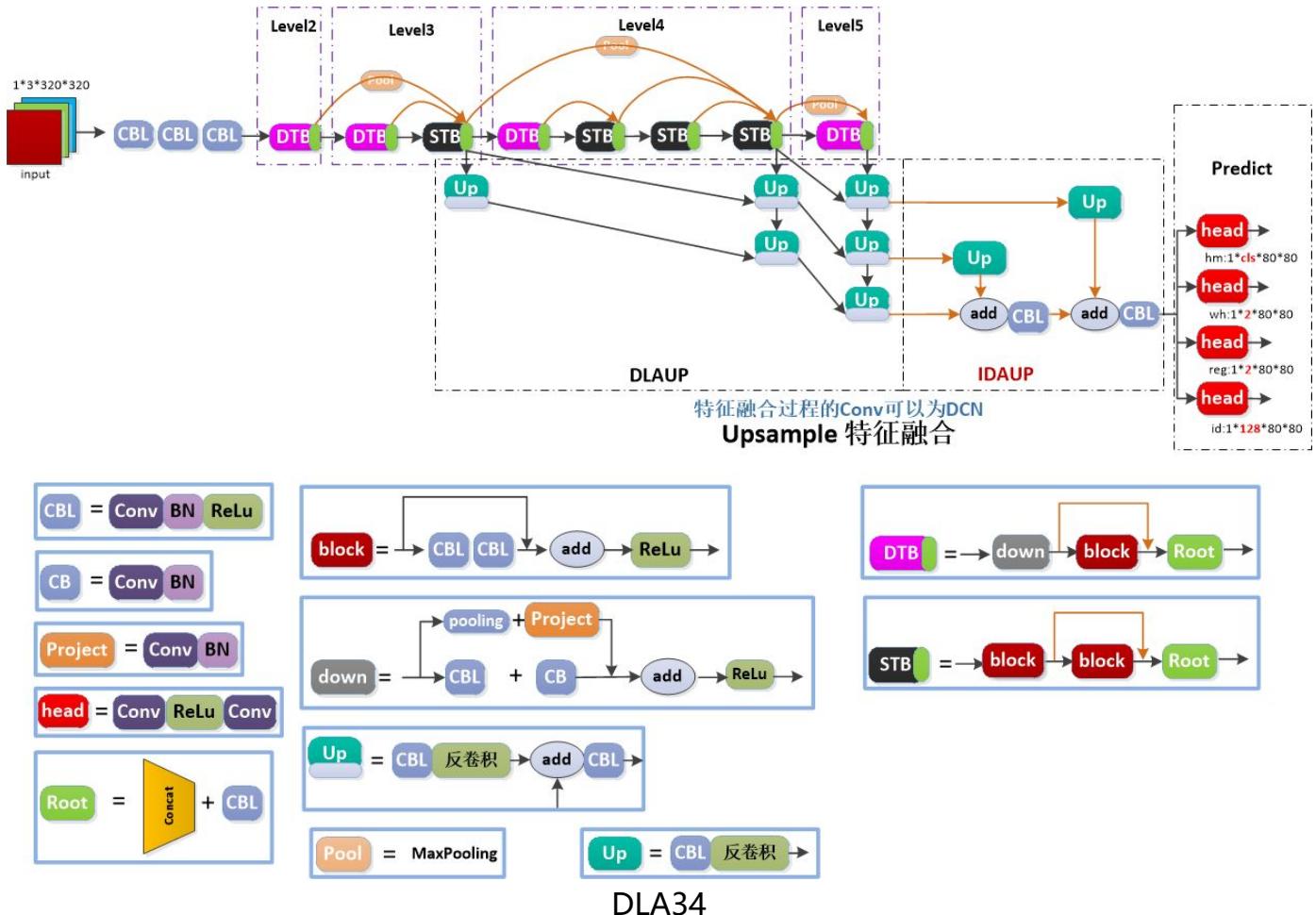
如下图所示，我复现的ResNet18与何凯明表格中的大体上一致，唯有网络的第一部分我用标准的CBR块(3×3)取代了何凯明标准中的卷积核(7×7)+最大池化(3×3)。



ResNet18

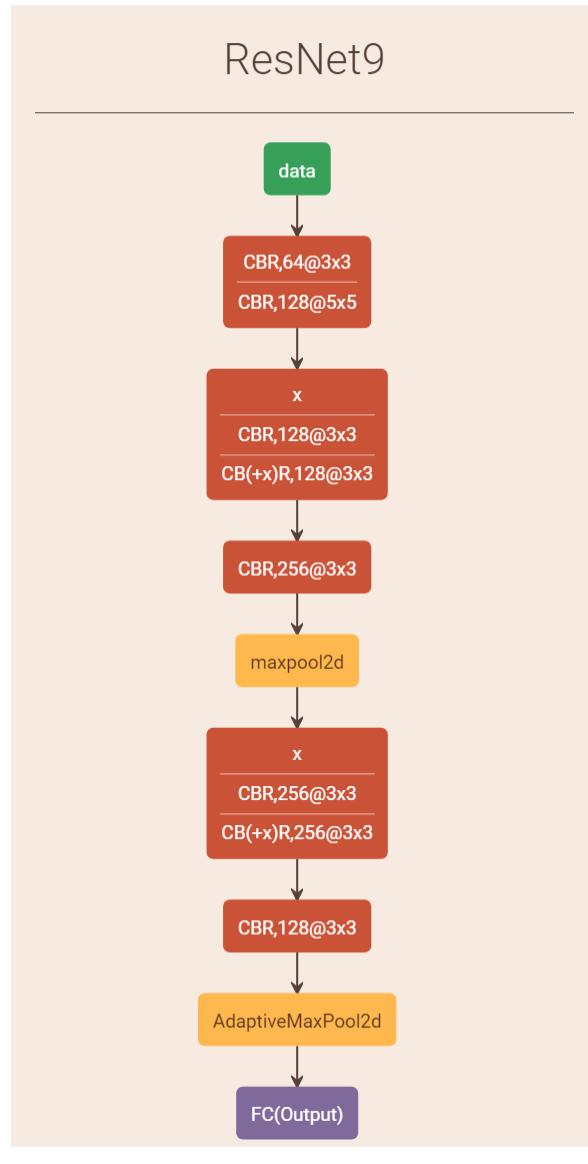
复现: DLA34

DLA全称Deep Layer Aggregation，旨在通过更深入的聚合来增强标准体系结构，以便更好地跨层融合信息，其网络结构如下。在本项目中，我对DLA34模型进行了复现，作为本项目中最为复杂的模型以便后续实验。



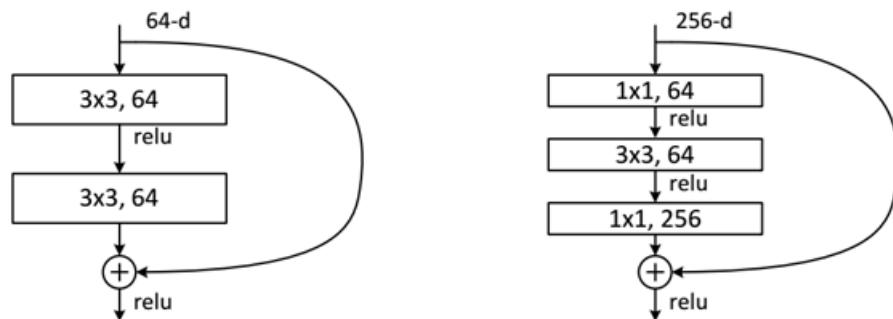
设计1: ResNet9

另一方面，虽然何凯明标准中最少层数的模型是ResNet18，但对于更少层数的ResNet仍然值得研究。我通过适当的删减，构建出了层数更少的ResNet9，其结构如下所示。



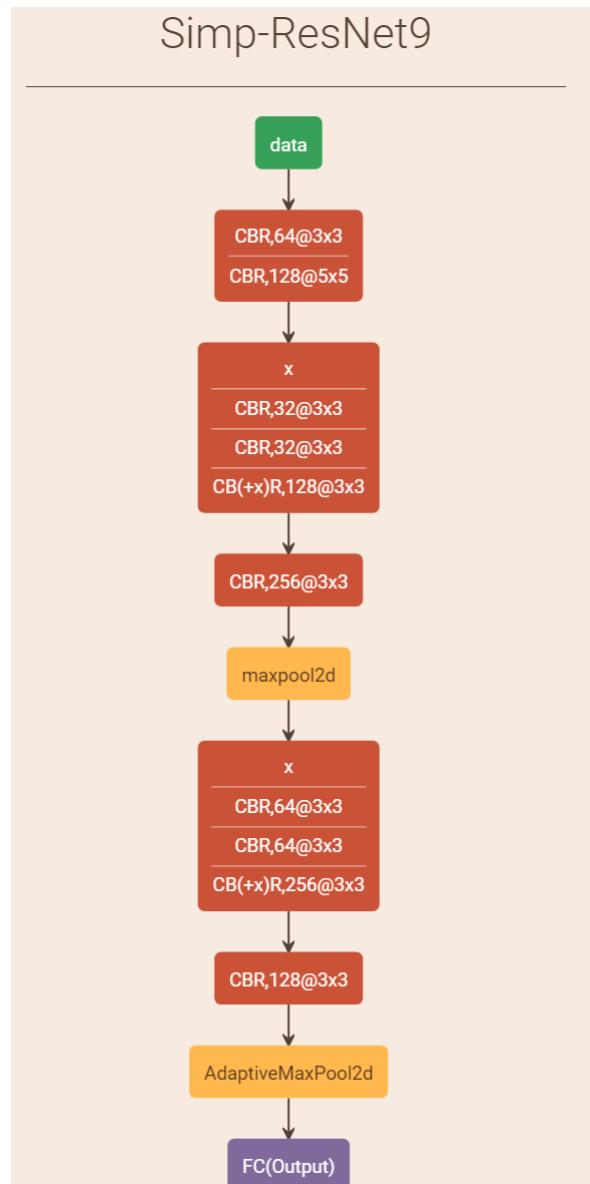
设计2: simp-ResNet9

此外，ResNet使用两种Res Block（如下所示），左图对应34层以下的情况，而右图Bottleneck Residual Block对应的是50层以上的情况，用于减少模型的参数量，属于简化版的残差块。何凯明通过实验发现虽然使用简易的残差块会使模型性能下降，但节省下来的参数用来增加层数反而会使模型整体性能提升。



ResNet中的残差单元

在本项目中，模型的训练速度和参数体积同样是很重要的评价指标，因此我也将这种简化残差块应用到先前的ResNet9模型中，其结构如下所示。



实验对比

超参数设置如下：

超参数				
weight decay	momentum	epoch num	batch size	lr
5e-4	0.9	20	128	1e-2

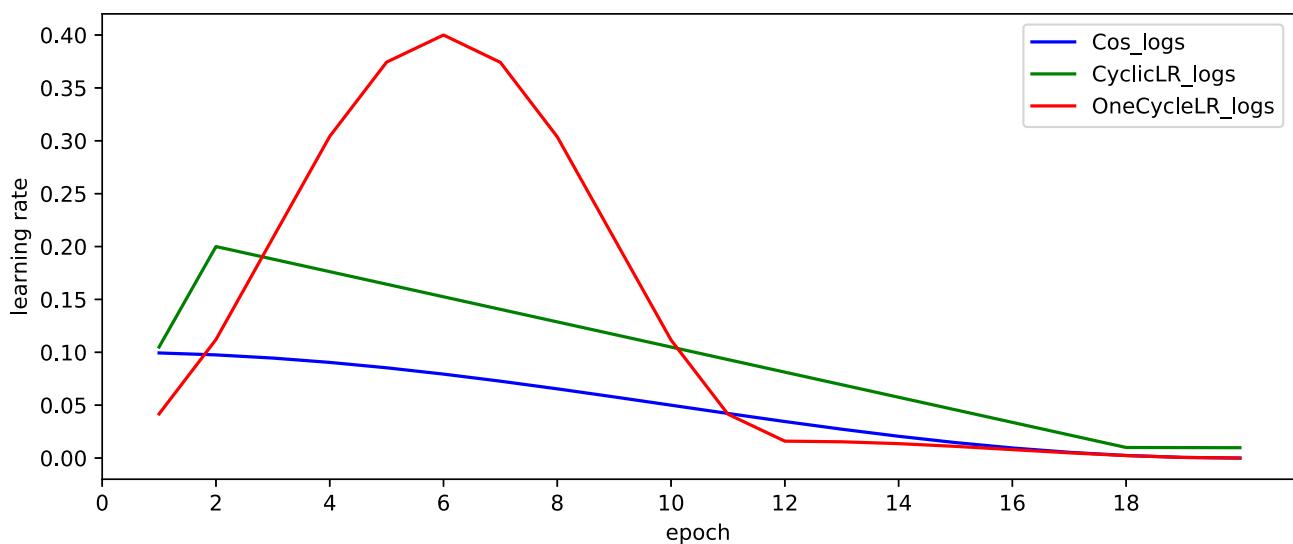
并使用SDG优化器和交叉熵损失函数，实验结果如下：

Model	Test Accuracy(%)	Time Cost(s)	参数体积(MB)
ResNet18	87.84	1518.023	42.7
DLA34	88.01	2926.004	69.3
ResNet9	86.98	373.718	8.7
simp_ResNet9	85.06	370.209	3.41

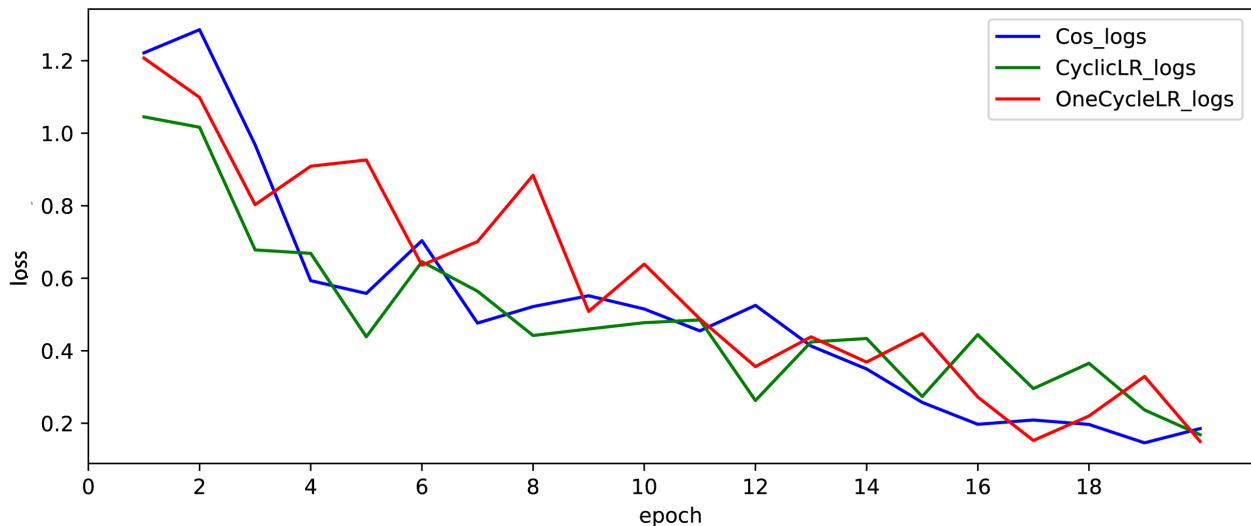
实验发现，在训练前期，模型的精确度与模型的复杂度正相关，但模型复杂度带来的收益并不大，相反，结构更简单的模型在精度方面表现不差的同时，在训练时长和参数体积方面还极具优势。因此，接下来的优化都将围绕RestNet9和simp_ResNet9展开。

1.3 学习率策略

由于已经研究表明SGD优化器对CIFAR-10数据集表现最好，所以在本项目中我将始终使用SGD优化器(momentum=0.9, weight decay=5e-4)，将研究点转移到学习率策略上。我选用了pytorch库中的CosineAnnealingLR、CyclicLR、OneCycleLR，如下图所示。



我对不同学习率策略的simp_ResNet9进行训练，并绘制了它们前20个epoch的交叉熵损失，如下所示：



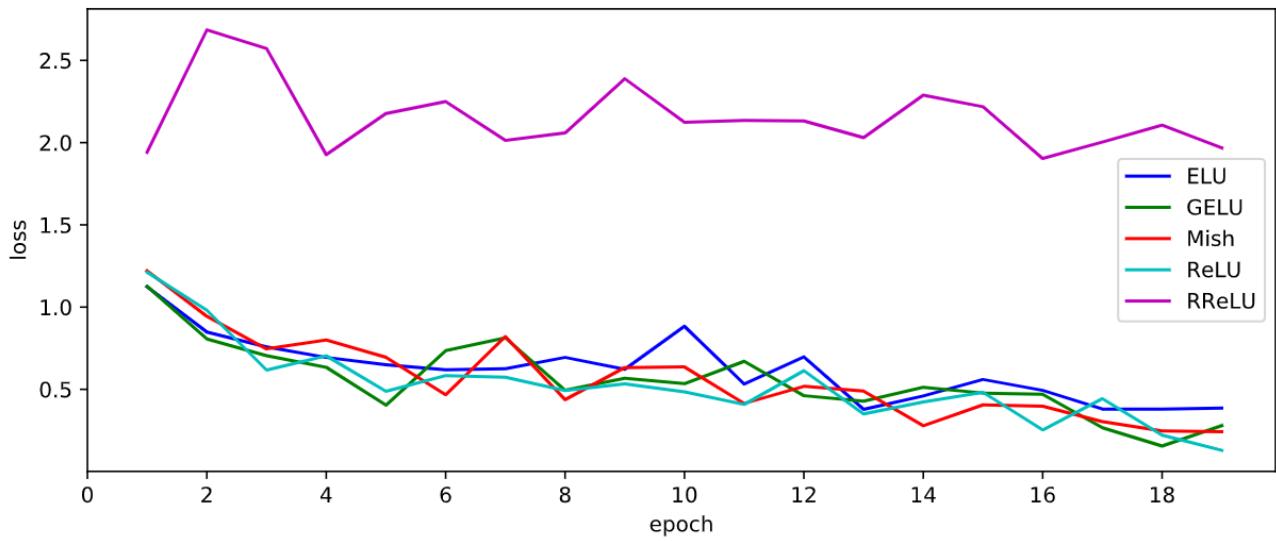
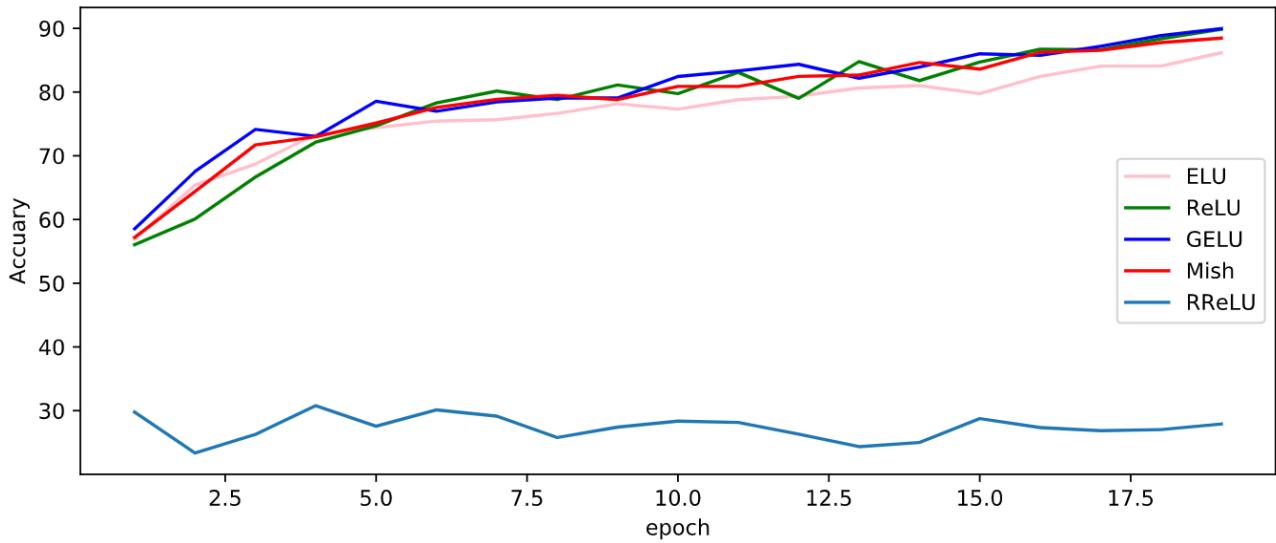
观察发现，在最后一轮中OneCycleLR损失最小，但是OneCycleLR和CyclicLR在下降过程中表现出很大的波动性，这不利于模型的收敛，而CosineAnnealingLR基本能够保证损失平稳下降，因此在训练模型到收敛时我将选用CosineAnnealingLR作为学习率策略。

如果只训练20轮，则三种学习率策略都让模型性能提升明显，结果如下：

Scheduler	Train Acc	Test Acc
CyclicLR	95.21	90.28
OneCycleLR	94.21	89.50
CosineAnnealingLR	94.15	90.24

1.4 激活函数

选用simp_ResNet9模型，我分别以pytorch库中的ELU, ReLU, GELU, Mish, RReLU作为激活函数，训练了20个epoch，结果如下：



观察结果，RReLU在损失和精度方面表现极差，几乎让模型无法训练；从损失函数上来看ELU也比其余击中稍差一些，因此在后续在将模型训练至收敛时会对GELU、ReLU、Mish都进行尝试。

1.5 损失函数

选用simp_ResNet9模型，我分别以pytorch库中的CrossEntropy、SmoothL1、BCEWithLogits作为损失函数训练了20个epoch，结果如下：

Loss Function	Train Acc(%)[20 epoch]
CrossEntropyLoss	90.63
SmoothL1Loss	61.59
BCEWithLogitsLoss	82.23

观察发现，使用CrossEntropy作为损失函数对于CIFAR-10分类任务的效果最好，因此在后续在将模型训练至收敛时将固定使用CrossEntropyLoss。

1.6 卷积核调整

在前面的工作中我已经对深层的卷积块进行了研究（Res Block->simp Res Block），因此本小节只对网络的第一卷积块进行调整实验。何凯明的研究和另一些研究都表明，相比于卷积核大小，网络的深度对于模型深度的影响力更大，于是我将第一卷积块分别设为【7x7】、【3x3，5x5】、【3x3，3x3，3x3】以及【3x3，3x3】的卷积组合进行实验，结果如下：

第一卷积块核结构	Test Acc(%)[20 epoch]	Test Acc(%)[200 epoch]
7	82.34	\
3,5	85.06	94.05
3,3	84.91	94.14
3,3,3	86.68	94.80

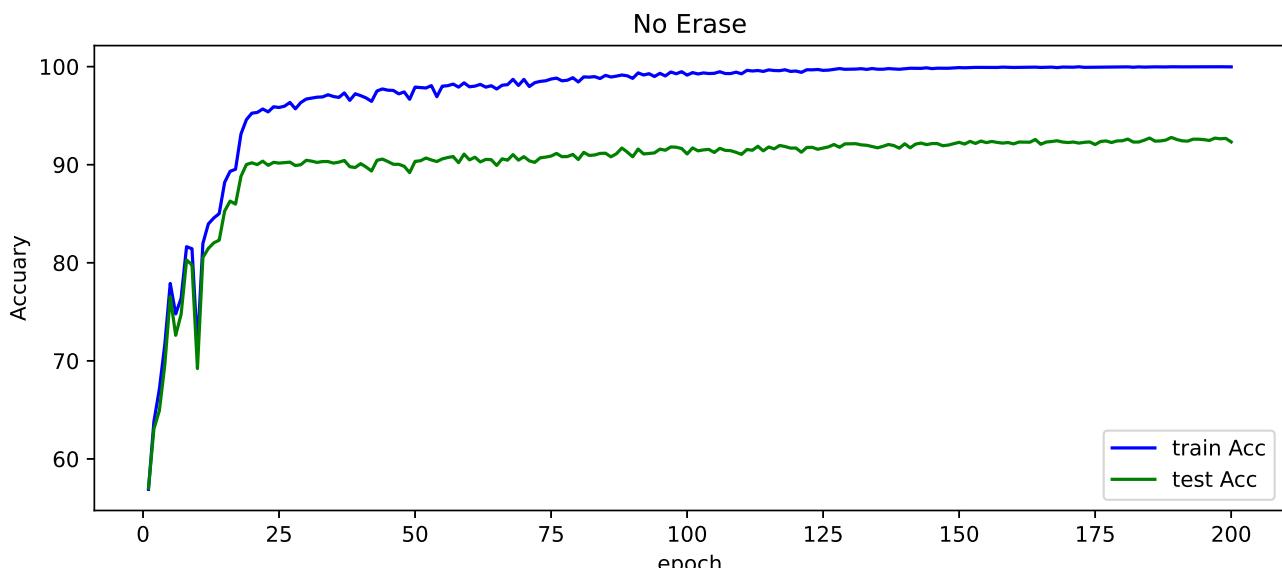
观察结果，在感受野相同的情况下，使用【7x7】、【3x3，5x5】、【3x3，3x3，3x3】的模型性能依次递增；而在第一卷积块包含两层卷积层的情况下，使用【3x3，5x5】的模型与使用【3x3，3x3】的模型没有显著的性能差异，这与何凯明等人的研究结果一致。

1.7 数据增强

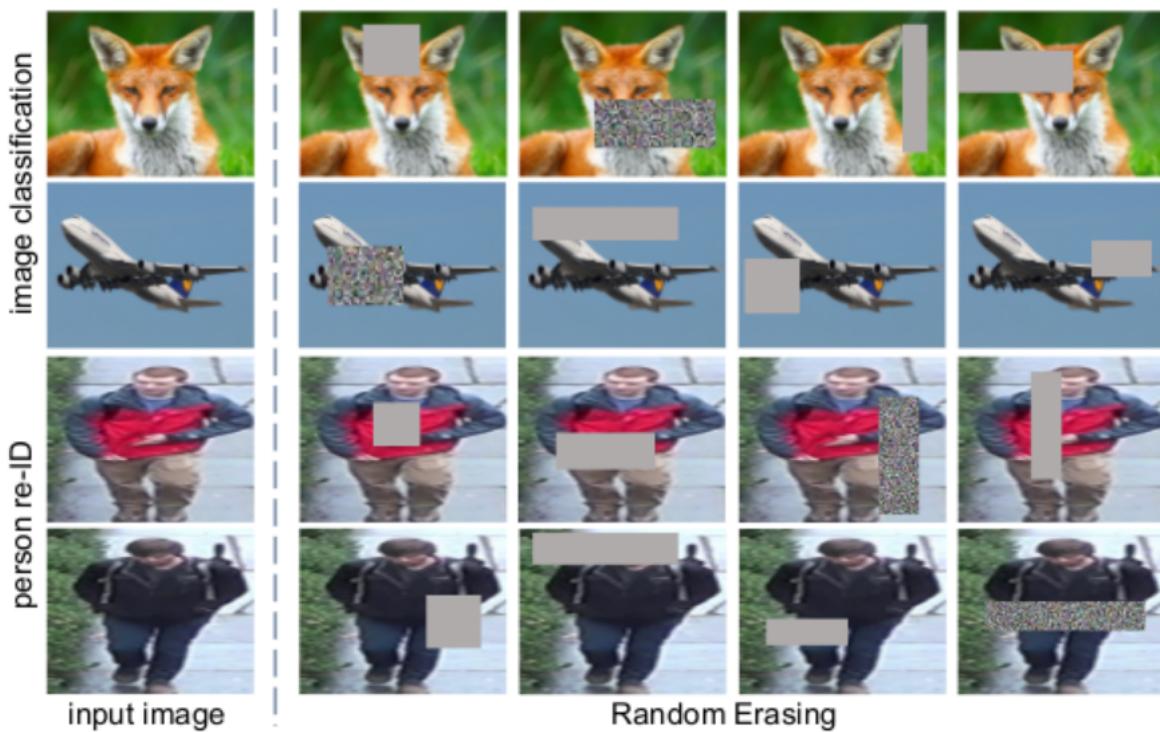
Baseline: RandomCrop、RandomHorizontalFlip、Normalize

在本项目中，默认对训练数据进行了基础的增强，包括随机截取图像、随机翻转和标准化。

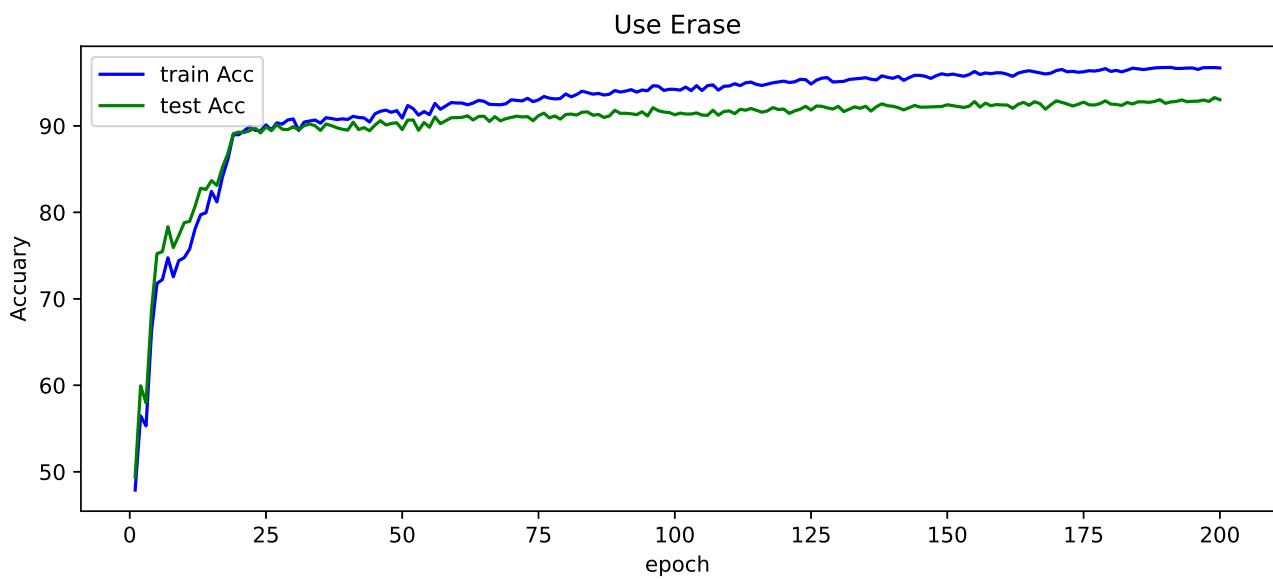
尽管如此，由于我自己设计的网络模型深度较浅，模型还是存在相当巨大的泛化误差，如下图，我将simp_ResNet9模型使用CosLR策略训练了200个epoch，观察发现在第75个epoch后训练精度已经趋于100%，同时测试精度也接近无法提升，这体现了模型的泛化性能较差。



Random Erasing



因此，我尝试提高模型的泛化性能，其中一个方向即为继续对训练数据进行增强。如上图所示，Random Erasing将图像中的一部分区域进行遮挡，以此防止模型对训练图像过度学习。在维持相同条件，即使用simp_ResNet9模型、CosLR策略训练200个epoch的情况下，我启用Random Erasing来增强数据，结果如下。



观察发现，在训练进入后期时泛化误差仍然存在，但相比于不使用Random Erasing，模型的泛化性能确实有所提升。

1.8 Maxdropout

注：下文将把ResNet9简称为rn9， simp_ResNet9简称为srn9

Dropout随机地去除训练阶段的神经元，Maxdropout则根据前一层的张量情况执行。它首先将张量 s 值归一化，然后将每一个大于给定阈值 p 的输出设置为0，因此这个值越高，它就越有可能被禁用。最初的工作表明，Maxdropout可以改进在CIFAR-10和CIFAR-100数据集上的ResNet18结果，而且它在WideResNet-28模型上的性能也优于Dropout。

因此为了进一步提升模型的泛化性能，我尝试将它也应用到我设计的模型中，实验结果如下：

Model	MaxDropout	Test Acc(%)[200 epoch]
rn9+CosLR+erase+ReLU	√	95.21
rn9+CosLR+erase+ReLU		95.21
srn9+CosLR+erase+GELU	√	94.03
srn9+CosLR+erase+GELU		94.05

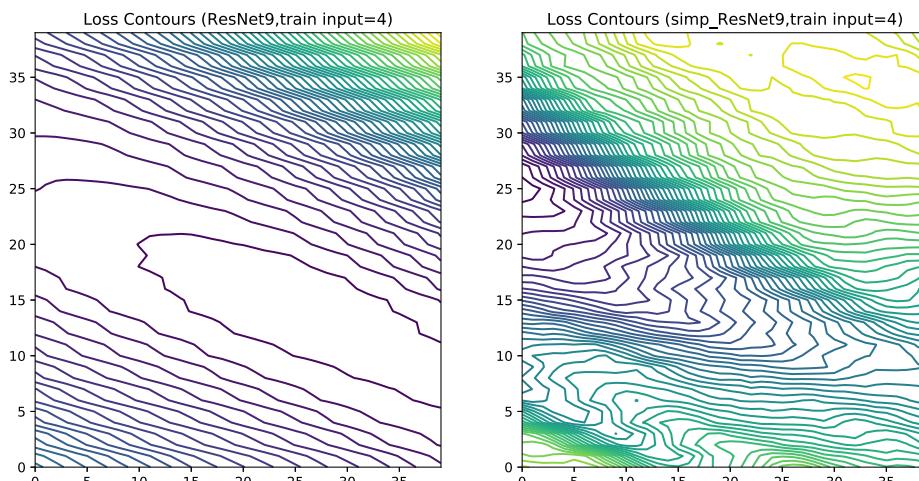
观察发现，Maxdropout的引入没有给模型带来提升，这一方面可能是模型本身已经具有相当的精度，边际效应不明显；另一方面则是泛化性能更可能受限于网络深度，maxdropout对此无能为力。

1.9 可视化

loss landscape

[Visualizing the Loss Landscape of Neural Nets](#)一文中作者对使用线性插值法 (Linear interpolation) 来可视化loss landscape的方法进行了抨击，指出两个不足：首先，1D 图很难显示非凸性；其次，该方法不考虑批量归一化或网络中的不变对称性。然后作者提出了更好的可视化方法--2D等高线图，并进行了相关实验，结果表明当一个模型具有越宽的损失等高线，那么它的泛化性就越好。

借助python中的loss_landscapes工具包，我对本项目中的rn9模型和srn9模型绘制了训练(样本数=4)的损失2d等高线，如下图所示：



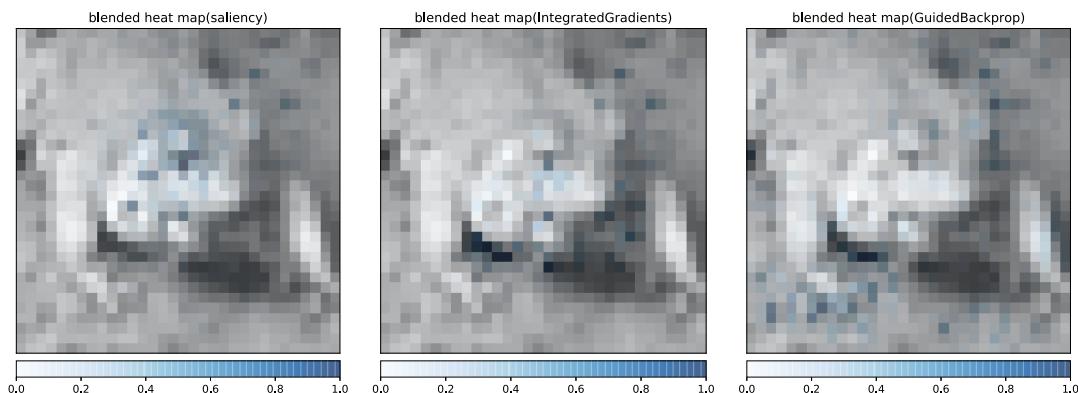
观察发现，两张图上的等高线存在着显著的疏密差异，srn9对应的图上明显的更加密集、曲线也更加不光滑，而rn9对应的图则要规则许多，使用上面提到的研究结论，可以对我实验中ResNet9的泛化性能高于srn9做出解释。

模型可解释性

借助python中的captum工具包，我使用了基于梯度的方法，这种方法在一定程度上可以对模型决策结果提供局部解释。随机从训练样本中抽取一张图片，通过模型的前线传播后它被预测为'frog'，与真实标签一致。



基于梯度的方法我一共使用了sallency、积分梯度算法（Integrated Gradients）和Guide Backpropagation三种，并以热力图的形式可视化，热力图中越深的色块对于模型分类结果的影响越大，可以观察到各种方法得到的热力图相比原图都有所出入，这部分不同反映了模型自身的误差，但除此以外基本把握住了青蛙的轮廓特征，这能够为模型的分类决策为'frog'做出解释。



Note：基于梯度的方法，就好像在问模型，对于这张图片，你为什么认为它是'frog'? 而模型给出的答案是对他图像注意程度的打分，各种打分方法都能大致描摹出青蛙的轮廓，说明模型确实是先把握了青蛙的轮廓特征再进行分类的。

1.10 总结

在我的实验中，简单的前期训练(20 epoch)最高可以达到90.26%的测试准确率，具体如下：

Best Model	Test Acc(%)	参数体积(MB)
srn9+CyclicLR	90.26	3.41

比较复杂且完整的训练(200 epoch)最高可以达到95.42%的测试准确率，且可以用准确率降低0.6%的代价，将参数体积降到2.98MB，具体如下：

Best Model	Test Acc(%)	参数体积(MB)
srn9_k333+CosLR+erase+GELU	94.80	2.98
rn9+CosLR+Maxdropout+erase+GELU	95.42	8.7

2. VGG与BatchNorm

实际应用时BN层有两个表现：1.能加速模型收敛；2.能降低模型对lr的敏感性。

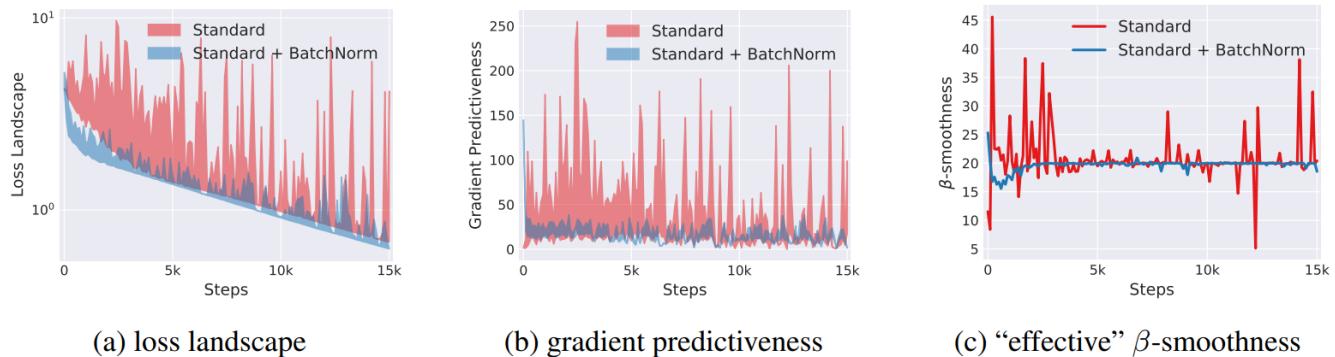
[How Does Batch Normalization Help Optimization](#)是NIPS2018的一篇文章，该文作者做出的解释是**BN层具有平滑效应**。在添加BN层后，模型具有系统具备L-Lipschitz稳定性，即满足下式：

$$|f(x_1) - f(x_2)| \leq L \|x_1 - x_2\|, \quad \forall x_1 \text{ and } x_2$$

它导致系统输出的变化波动会被限制在一个常系数范围内，从而使得梯度更加值得依赖和具有预测性，因为深度学习的优化方法正是基于梯度下降。

因此，在使用BN层后我们可以放心地设置一个较大的学习步数且能保证梯度方向维持不错的精确。

作者通过下图所示的三个实验来支撑刚才所提的结论，在本项目中我将对它们进行复现来进一步理解文章。

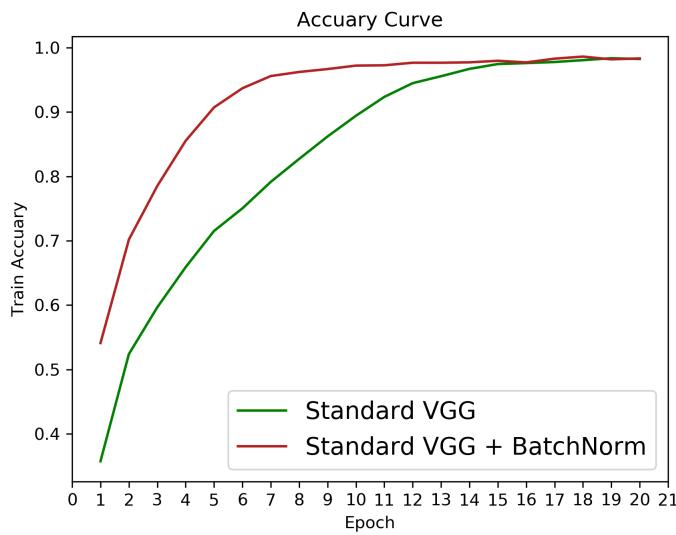


2.1 BatchNorm对VGG影响

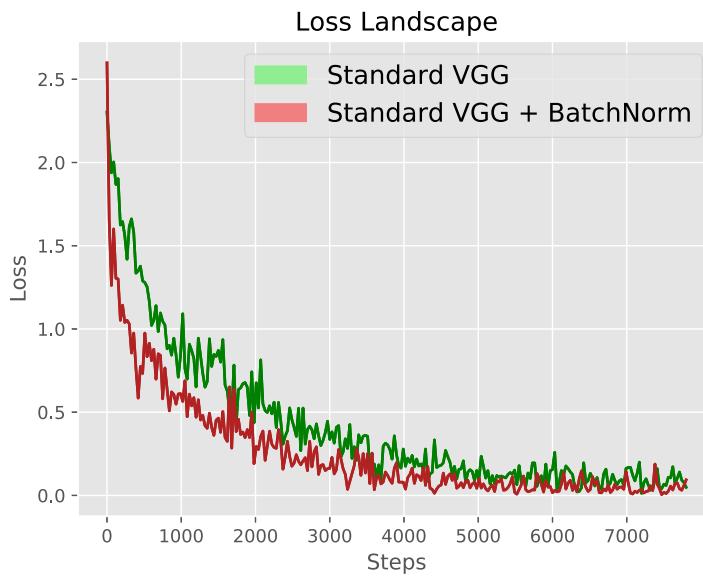
在正式复现三个实验前，我先在标准VGG网络上对使用BatchNorm与否做了对比研究。给定下表超参数：

超参数				
optim	loss function	epoch num	batch size	lr
Adam	CrossEntropy	20	128	1e-4

训练集上的精度曲线如下图所示：



模型在训练每次迭代的损失如下图所示：



观察发现，使用BN让模型在训练集上精度提升更快、损失下降更快，这说明BN确实能加速模型收敛；

另一方面，模型的测试集精确度和速度对比如下：

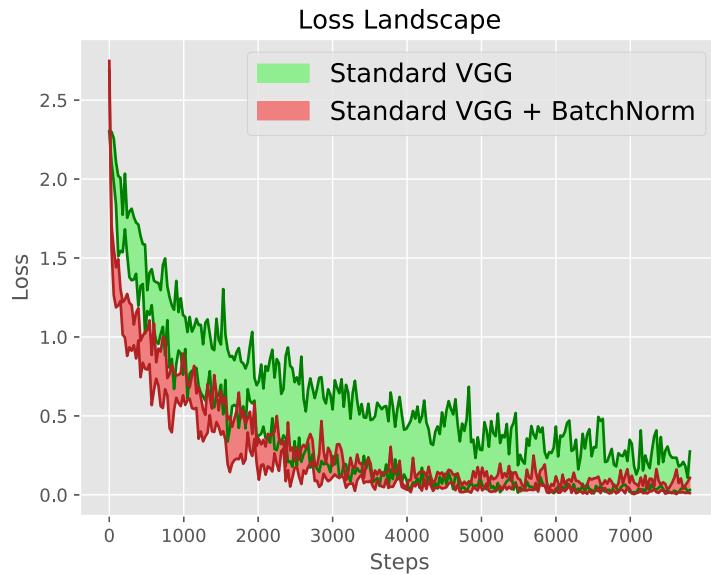
Model	Test Acc(%)[20 epoch]	Speed(s/epoch)
VGG	76.92	9.87
VGG+BatchNorm	81.92	11.08

相同轮数下带BN层的模型泛化性能更好，这也可以用敏感性降低来解释，而训练计算量的提升正是为此必须承担的代价。

2.2 Loss Landscape

实验一为Loss Landscape，对模型每步的损失分布进行可视化对比，参数设置与结果如下：

超参数				
optim	loss function	epoch num	batch size	lr
Adam	CrossEntropy	20	128	2e-3, 1e-3, 5e-4, 1e-4



观察发现：

1. 使用BN的模型在各个迭代的损失分布均值总是小于不使用BN的模型
2. 使用BN的模型在各个迭代的损失分布波动也总是小于不使用BN的模型

从这两点可以看出，在使用BN后模型的训练损失稳定性也得到了增强，是符合L-Lipschitz性质的。

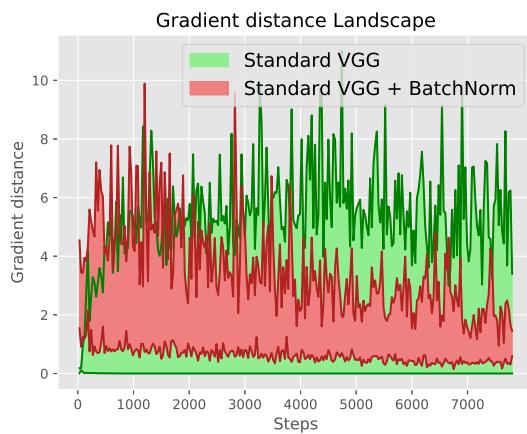
2.3 Gradient Prediction(Bounds 1.1)

实验二为VGG模型使用BN与否情况的梯度预测水平对比，在我复现的实验中，我使用迭代前后的梯度张量的二范数来反映梯度预测的误差：

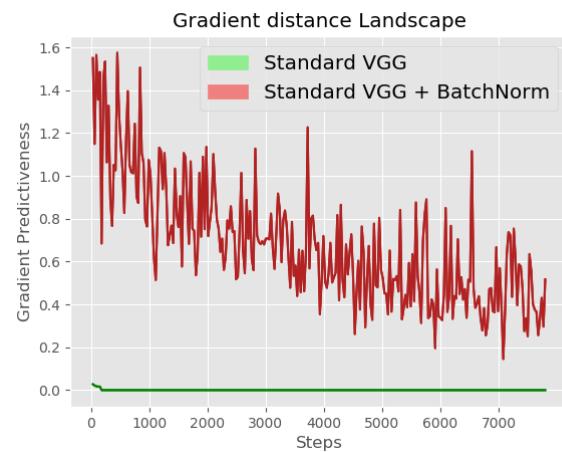
$$Err_{k+1} = \|\mathbf{grad}_{k+1} - \mathbf{grad}_k\|_2, \quad k = 1, 2, \dots$$

然后，类似于2.2节，我也选取了一组学习率以landscape的形式可视化，参数设置与结果如下：

超参数				
optim	loss function	epoch num	batch size	lr
Adam	CrossEntropy	20	128	2e-3, 1e-3, 5e-4



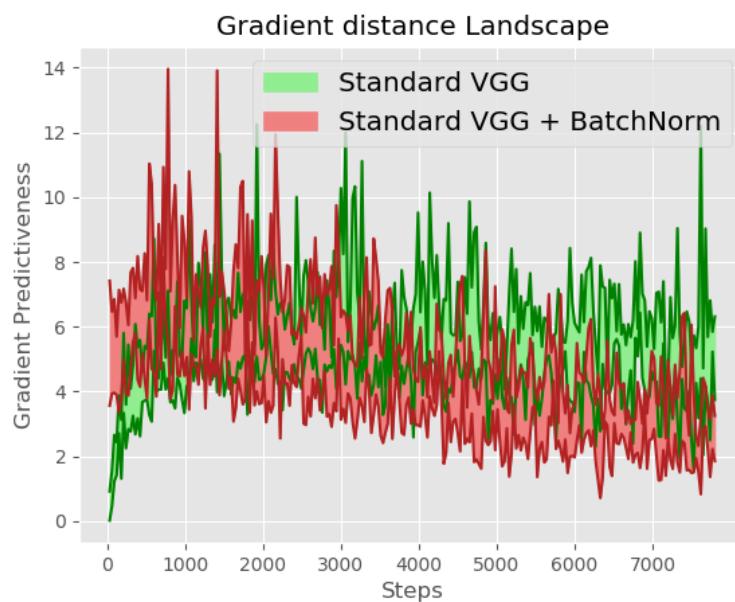
(a) learning rates=[$2\text{e-}3$, $1\text{e-}3$, $5\text{e-}4$];



(b) learning rate= $2\text{e-}3$;

观察发现图(a)与本节总述中参考文章的实验结果并不一致，于是我将学习率组中的每个水平单独可视化，发现学习率在大于 $2\text{e-}3$ 时，会出现图(b)中的现象，即不使用BN的模型梯度预测误差恒等于0。然而，这并不意味此时模型的梯度预测误差真的为0，相反，可能是因为学习率过大，导致短短几次迭代后就出现了梯度爆炸或梯度消失的现象，从而迭代前后的梯度恒等于0或者无穷，这种百分百准确的预测也并非我们希望看到的。

接下来，为了验证我的猜想，我将学习率组水平降低并重复了实验二，结果如下：



(c) learning rates=[$1\text{e-}3$, $7\text{e-}4$, $5\text{e-}4$]

可以看到，在减小了学习率水平后，无BN的模型没有再出现梯度预测误差始终为0的情况，我的猜想得到了验证。

此外，图(c)的表现与参考文章中的表现也大体一致，但也有所不同，不同在于：图(c)中的前半段带BN模型的梯度预测误差比起无BN模型还要更高，直到后半段才明显低于无BN模型，这说明了BN对于模型的影响力是呈现阶段性的，随着训练步数的增加，其影响力也越来越大，也即是说，梯度预测的稳定性越来越高。

思考：在训练之初，梯度预测的稳定性差可能反而有利于模型收敛，因为模型的实际优化场景总是非凸的，迭代间梯度的差异有助于模型跳出局部最优的吸引力，帮助模型更快地找到全局最优点。

2.4 Effective β -Smoothness(Bounds 1.2)

实验三为VGG模型使用BN与否情况的Effective β -Smoothness对比，我先从理论上进行了梳理。

在数学上一个函数是 β -Smooth的当且仅当其梯度函数是 β -Lipschitz的，也即：

$$||\nabla f(x_1) - \nabla f(x_2)|| \leq \beta ||x_1 - x_2||$$

神经网络可以将输入转化成输出，因此也可以被看作是一种特殊的函数 $f(x, w)$ ，每次迭代都是一次求解关于 w 的优化问题，则有：

$$w_{i+1} = w_i - lr_i * \nabla f_{x_i}(w_i)$$

代入到 β -Lipschitz公式中：

$$||\nabla f_{x_i}(w_i) - \nabla f_{x_i}(w_{i+1})|| \leq lr_i * \beta ||\nabla f_{x_i}(w_i)||$$

从而第*i+1*次迭代的 β 满足：

$$\beta_{i+1} = \max \frac{||\nabla f_{x_i}(w_i) - \nabla f_{x_i}(w_{i+1})||}{lr_i ||\nabla f_{x_i}(w_i)||}$$

在使用mini-batch的情况下，可以假设有以下近似：

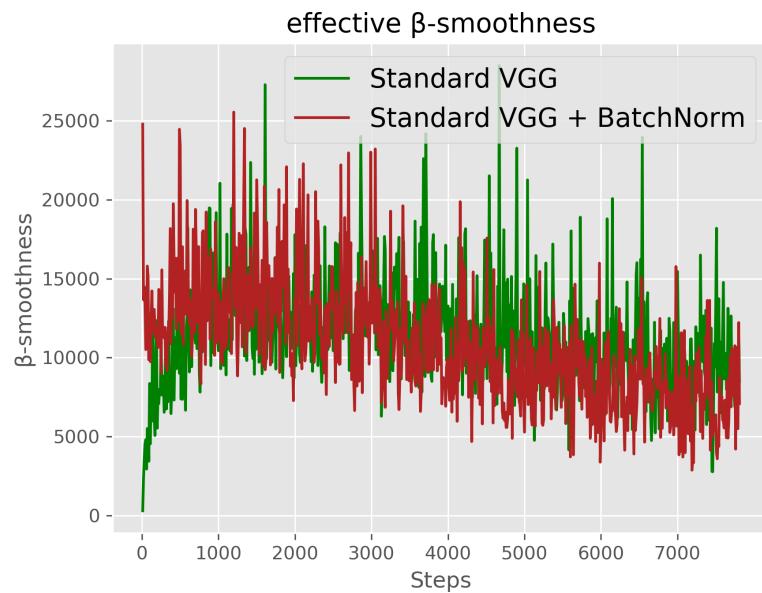
$$\nabla f_{x_i}(w_{i+1}) \approx \nabla f_{x_{i+1}}(w_{i+1})$$

为了方便程序迭代计算与可视化美观，可以稍作放缩：

$$\beta_{i+1} = \max \frac{||\nabla f_{x_i}(w_i) - \nabla f_{x_{i+1}}(w_{i+1})||}{lr_i} * M, \quad M = \max_i ||\nabla f_{x_i}(w_i)||$$

至此理论梳理完毕，而当 β_{i+1} 越小，说明模型越平滑，接下来我对带BN与无BN模型进行训练，训练参数以及 β 随迭代数变化的可视化结果如下：

超参数				
optim	loss function	epoch num	batch size	lr
Adam	CrossEntropy	20	128	2e-3, 1e-3, 5e-4



观察发现，大部分情况下带BN模型的 β 都小于无BN模型，这足以说明BN的使用可以提高模型的平滑性；至于我的结果没有参考文章中的效果那么明显，很可能是由于我选择的学习率组更小的缘故，这时学习率对标准的VGG来说也已经比较合适，因此在 β 上就没有表现出夸张的波动性。

3. DessiLBI(Bonus 2)

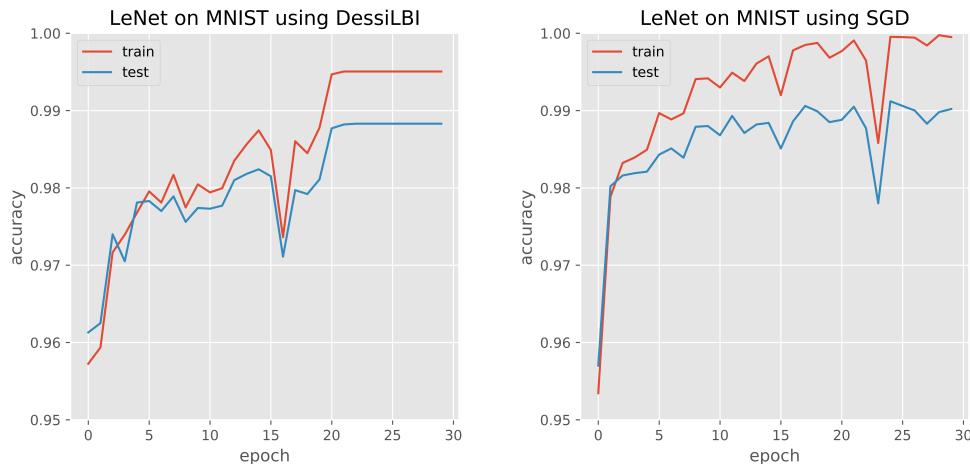
在深度学习中，过参数化是一个非常常见的问题。正常情况下，我们会使用反向选择方法来获得一个紧凑的模型，此时过参数化往往不可避免，从而给深度学习的应用带来困难；DessiLBI是一种通过训练正向选择、得到带有稀疏结构的模型的优化方法。以下我将通过几个实验来认识 DessiLBI 在这一方面的改善。

3.1 LeNet on MNIST using DessiLBI & SGD

在 [DessiLBI: Exploring Structural Sparsity of Deep Networks via Differential Inclusion Paths](#) 一文中，作者给出了 Example 代码以及相应的超参数，在此基础上，我首先对 MNIST 数据集使用 LeNet 进行分类，实验对比以 DessiLBI 为优化器和以 pytorch 库中的标准 SGD 为优化器，超参数如下：

超参数					
optim	kappa	mu	epoch num	batch size	lr
DessiLBI & SGD	1	20	30	128	$0.1^{(1+epoch/20)}$

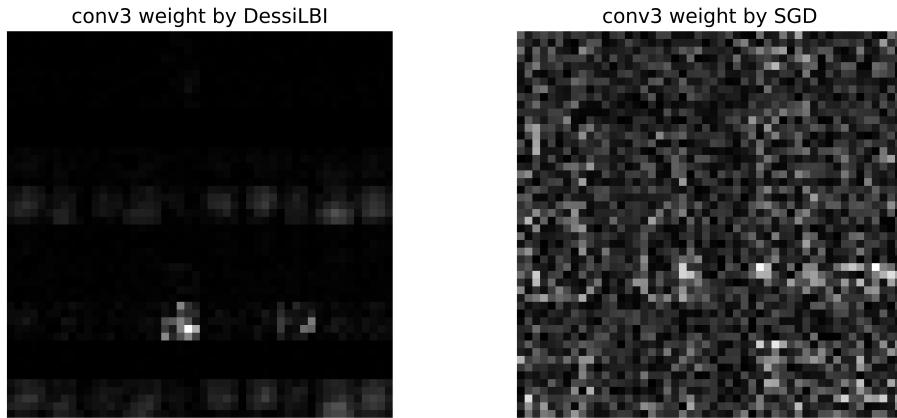
训练结果如下：



可以看到，在模型精度保持差不多的情况下，使用DessiLBI的实验组训练精度与测试精度的差距更小，这说明DessiLBI具有更好的泛化性能；

此外，使用DessiLBI的实验组测试精度实际上略低于使用SGD的实验组，分析后我认为这是超参数的选取导致的：在训练超过20轮后，使用DessiLBI的实验组两种精度同时不再上升，是因为学习率骤减为0.01，对于此时的LeNet来说学习率过小了。

在训练结束后，我还分别将两组实验组LeNet网络中的第三层卷积层提取出来进行可视化，结果如下：



观察发现，DensiLBI实验组图片中大部分区域为黑色的，意味着权重为0，SGD实验组图片中则大部分不是黑色的，意味着权重非0，但是两个模型的测试精度却基本相同，这证实了：1. 过参数化问题确实存在；2. DensiLBI确实能让模型更加稀疏。

接着我对DensiLBI实验组的第三卷积层做剪枝，另外又对第三卷积层和第一全连接层做剪枝，精度变化如下：

剪枝层\剪枝程度	0%	5%	10%	20%
conv3	0.9830	0.9830	0.9830	0.9830
conv3+fc1	0.9830	0.9830	0.9830	0.9830

剪枝层\剪枝程度	40%	60%	80%
conv3		0.9830	0.9826	0.9668
conv3+fc1		0.9830	0.9826	0.9668

可以看到，经过剪枝模型的精度变化依旧不大，这可能是因为模型本身已经非常稀疏了，被剪枝的有相当一部分原先权重已经为0，直到剪枝80%时比较关键的权重才被剪枝，大体可以理解为原模型在该层过参数化程度为80%。

3.2 Update by Adam

```

        exp_avg, exp_avg_sq = state['exp_avg'], state['exp_avg_sq']
        beta1, beta2 = group['betas']

        state['step'] += 1

        if group['weight_decay'] != 0:
            grad.add_(group['weight_decay'], p.data)

        # Decay the first and second moment running average coefficient
        exp_avg.mul_(beta1).add_(1 - beta1, grad)
        exp_avg_sq.mul_(beta2).addcmul_(1 - beta2, grad, grad)
        denom = exp_avg_sq.sqrt().add_(group['eps'])

        bias_correction1 = 1 - beta1 ** state['step']
        bias_correction2 = 1 - beta2 ** state['step']
        step_size = group['lr'] * math.sqrt(bias_correction2) / bias_correction1

        p.data.addcdiv_(-step_size, exp_avg, denom)

```

在完成了MNIST上述试验后，我对DessiLBI中W的更新进行了修改，让它的算法从SGD变成Adam，参考了PJ指导中的Adam代码后，我在slbi_opt.py文件中的step函数内加入以下代码并调整：

```

# 字典初始化
if 'step' not in param_state:
    param_state['step'] = 0
    param_state['exp_avg'] = torch.zeros_like(p)
    param_state['exp_avg_sq'] = torch.zeros_like(p)

param_state['step'] += 1

#参数step, avg, avg_sq
step = param_state['step']
exp_avg = param_state['exp_avg']
exp_avg_sq = param_state['exp_avg_sq']

#使用指数加权平均计算一阶和二阶梯度量，不断地更新avg和avg_sq
exp_avg.mul_(beta1).add_(d_p, alpha=1 - beta1)
exp_avg_sq.mul_(beta2).addcmul_(d_p, d_p, value=1 - beta2)

#以step为指数计算bias_corr
bias_correction1 = 1 - beta1 ** step
bias_correction2 = 1 - beta2 ** step
step_size = lr_kappa / bias_correction1
denom = (exp_avg_sq.sqrt() / math.sqrt(bias_correction2)).add_(eps)

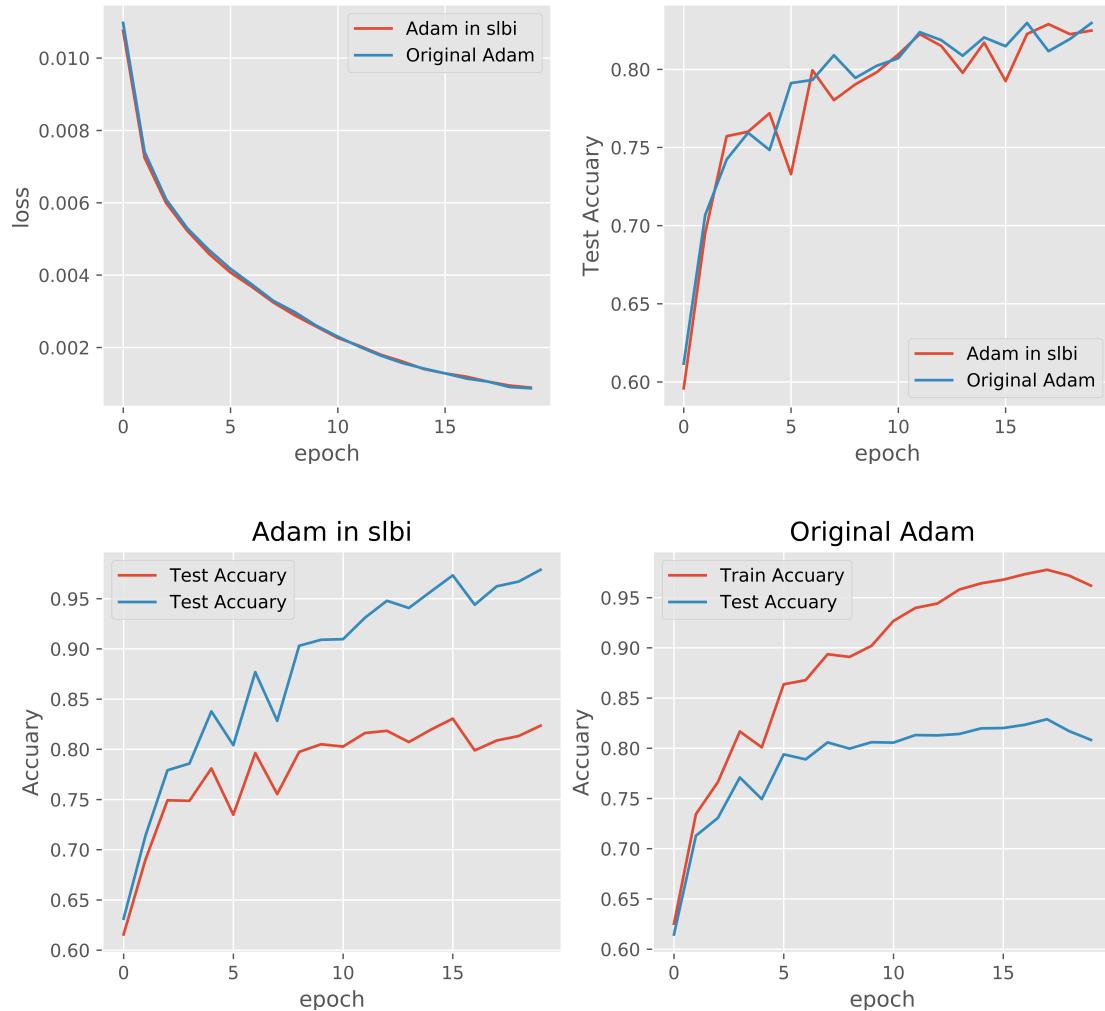
p.data.addcdiv_(exp_avg, denom, value=-step_size)

```

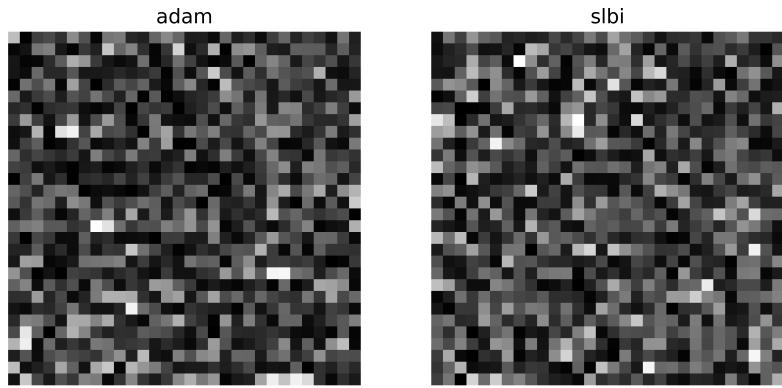
在修改后选取CIFAR-10数据集，对ResNet网络进行训练，分别以Adam版的DessiLBI和原版的Adam为优化器，超参数和训练中的损失、精度曲线对比如下：

超参数

optim	kappa	mu	epoch num	batch size	lr
DessiLBI_Adam & Adam	1	20	30	128	3e-4



观察发现，两实验组在测试精度和损失上的表现总体上没有区别，这有可能是数据集样本更加复杂，模型设计比较简单而导致没有出现过参数化现象，于是我类似3.1节中的方法，进一步对ResNet的第三卷积块的第一层进行权重可视化，如下所示：



可以看到，两实验组图片中的权重也确实没有显著差异，说明原ResNet在CIFAR-10上没有过参数化。

Reference

- [1]. [Deep Residual Learning for Image Recognition](#)
- [2]. [Visualizing the Loss Landscape of Neural Nets](#)
- [3]. [How Does Batch Normalization Help Optimization](#)
- [4]. [DessiLBI: Exploring Structural Sparsity of Deep Networks via Differential Inclusion Paths](#)
- [5]. [Mish: A Self Regularized Non-Monotonic Activation Function](#)
- [6]. [Deep Layer Aggregation](#)