Report of Project-2

陈浩贤 18307110276

Abstract

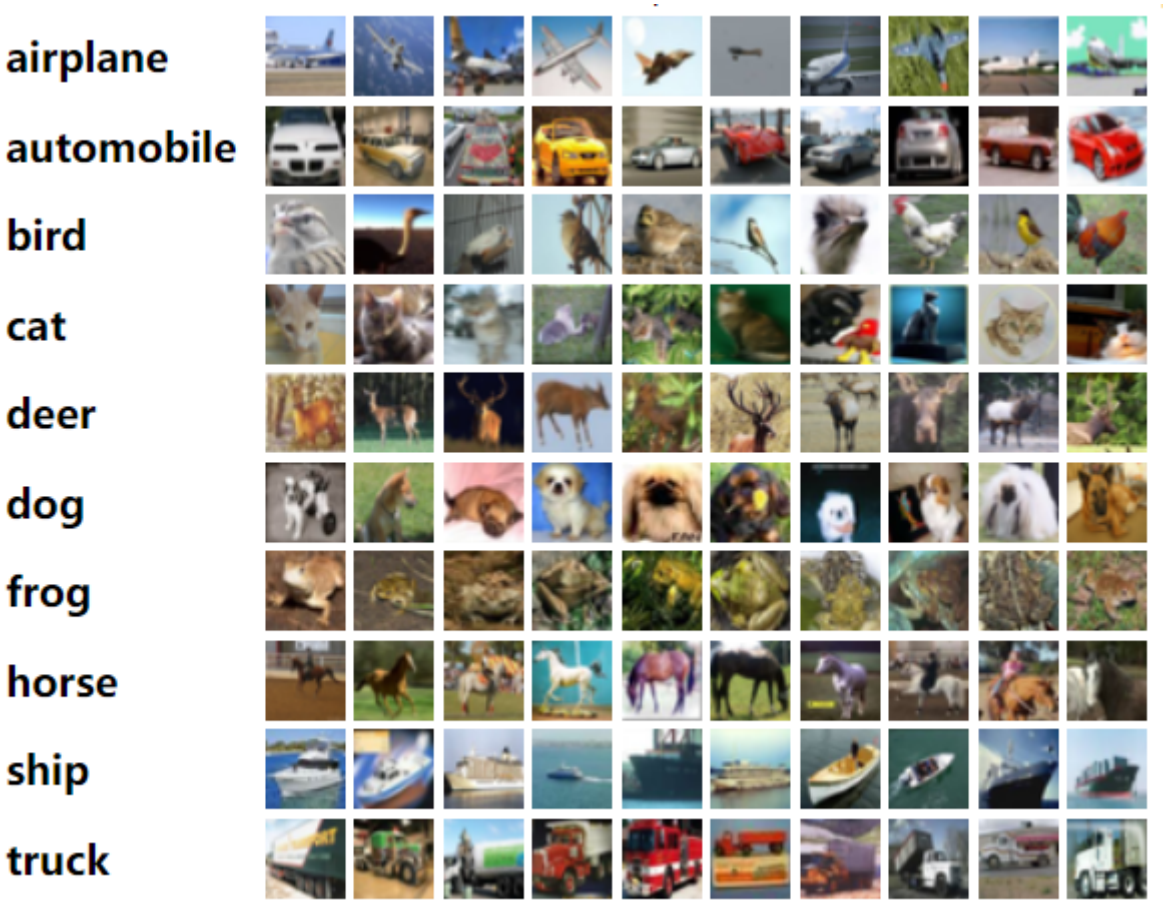
- 探究了不同的网络结构、激活函数、损失函数、优化器在 CIFAR-10 图像分类任务上的效果，并最终使用 WideResNet 的模型达到了 97.33% 的准确率
- 探究了 Batch Normalization 对神经网络训练起到的作用
- 利用 DessiLBI 训练神经网络，并实现了 Adam 与 DessiLBI 的结合
- Code Repository: https://github.com/chx7514/nndl-pj2

1 Train a Network on CIFAR-10

1.1 CIFAR-10

CIFAR-10 数据集由 10 个类的 60000 个 32x32 彩色图像组成，每个类有 6000 个图像，分为 50000 个训练图像和 10000 个测试图像。

以下为数据集的 10 个类，以及来自每个类的 10 个随机图像（图源CIFAR-10官网 CIFAR-10 and CIFAR-100 datasets (toronto.edu)）



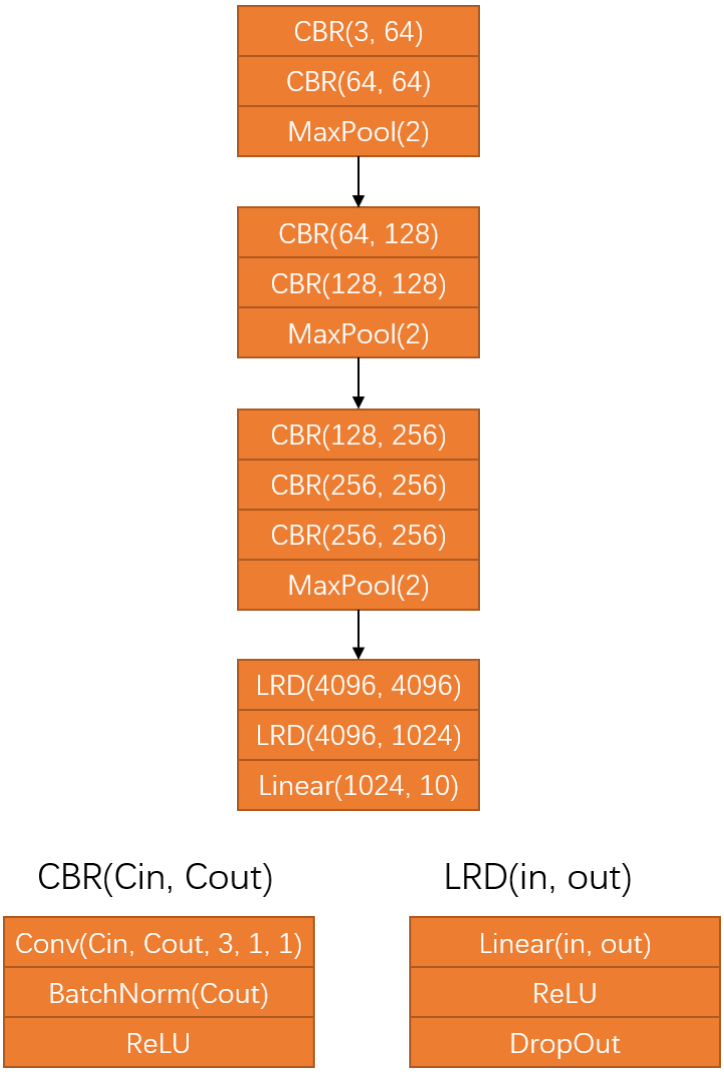
1.2 Network Structure

使用 Pytorch 搭建了以下不同结构的网络，并测试它们的效果。

1.2.1 ConvNet

卷积神经网络架构，包括了全连接层、卷积层、池化层、激活函数。卷积层均使用 3×3 的卷积核，每个卷积模块都使用 CBR(Conv-BatchNorm-ReLU) 结构，每个模块内将图像的通道数倍增，同时最后连接步长为 2 的最大值池化层。线性分类模块使用 LRD(Linear-ReLU-Dropout) 结构，最终转化为对应的 10 个分类结果。

具体结构如下：

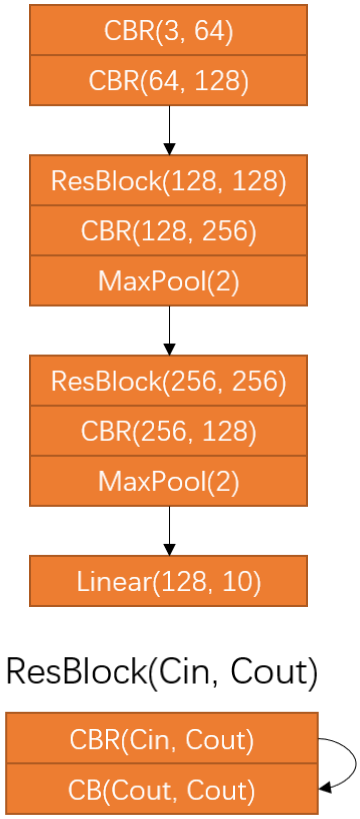


1.2.2 ResNet9

ResNet 引入了残差连接，使得网络的准确率更高，同时更容易训练，收敛得更快。自实现的 ResNet9 与传统的 ResNet 结构不同：

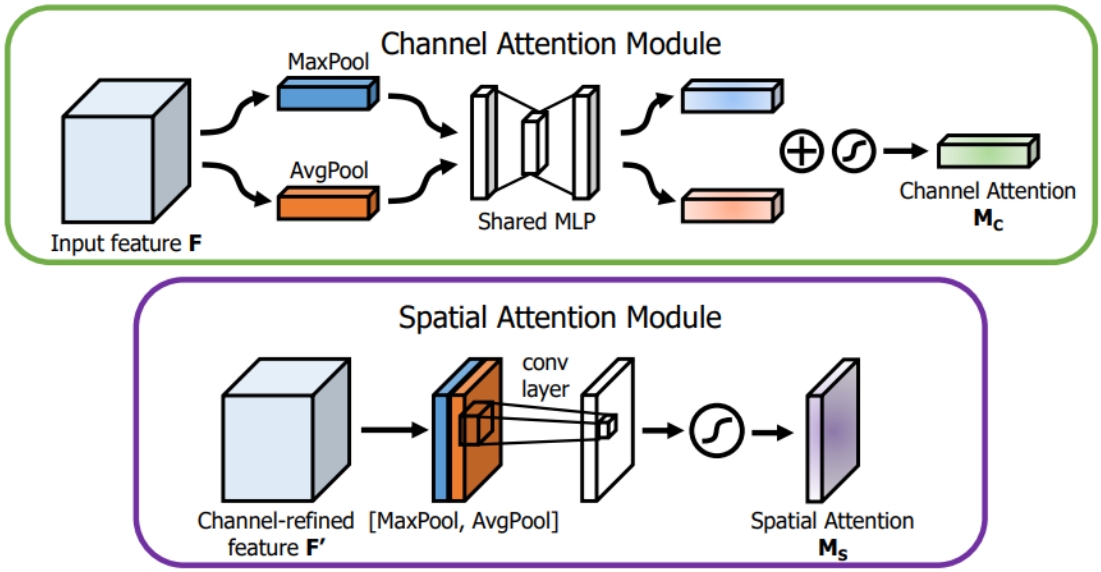
- 为了减少训练时间，减少了层数，只使用了两个 ResBlock，共有 9 层
- 由于 CIFAR-10 图片大小为 32×32，所以将第一层卷积核的大小从 7×7 改为 3×3

具体结构如下：

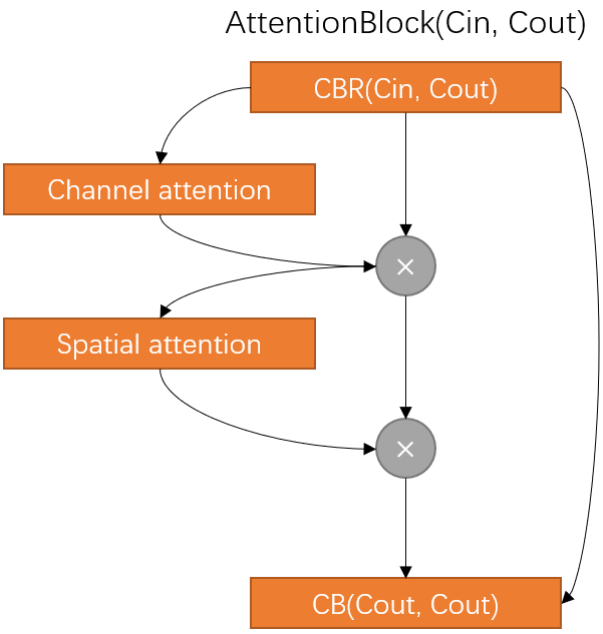


1.2.3 Attention + ResNet9

文章 《CBAM: Convolutional Block Attention Module》提出了轻量的注意力模块：空间注意力和通道注意力，如下图所示。通道注意力模块对图像分别做最大值和平均值池化，再经过共享的卷积层；空间注意力模块对图像同一位置的不同通道做最大值和平均值池化，将两者合并再经过卷积层。



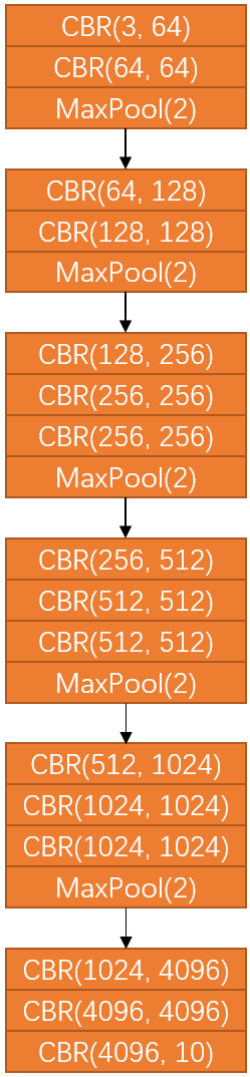
参考论文的思想，在以上的 ResNet9 中引入了空间注意力和通道注意力模块，将 ResBlock 换成以下的 AttentionBlock，其中 × 表示点乘



1.2.4 FullConvNet

全卷积神经网络在目标检测、图像分割任务中取得了较好的效果，如 FCN 等一系列工作。原始的卷积神经网络中包含卷积层和线性层，其中卷积层起到提取特征的作用，线性层起到分类的作用；而在全卷积神经网络中，使用卷积层替代线性层，让卷积层起到分类的作用。参考 FCN 构建自己的全卷积神经网络，主要的思路是使用 5 倍下采样将 32×32 的图片提取为 1×1 的特征。

具体的网络结构如下：



1.2.5 ResFullConvNet

尝试将 FCN 中的卷积层替换成 ResBlock，可以得到 ResFullConvNet，结构与以上构建的 FCN 相似。

1.2.6 WideResNet

文章《Wide Residual Networks》基于 ResNet 提出了 WideResNet。ResNet 的网络结构较深，每层的卷积核和通道数较小，是高瘦型的网络；而 WideResNet 则对 ResNet 的这种架构进行改进：

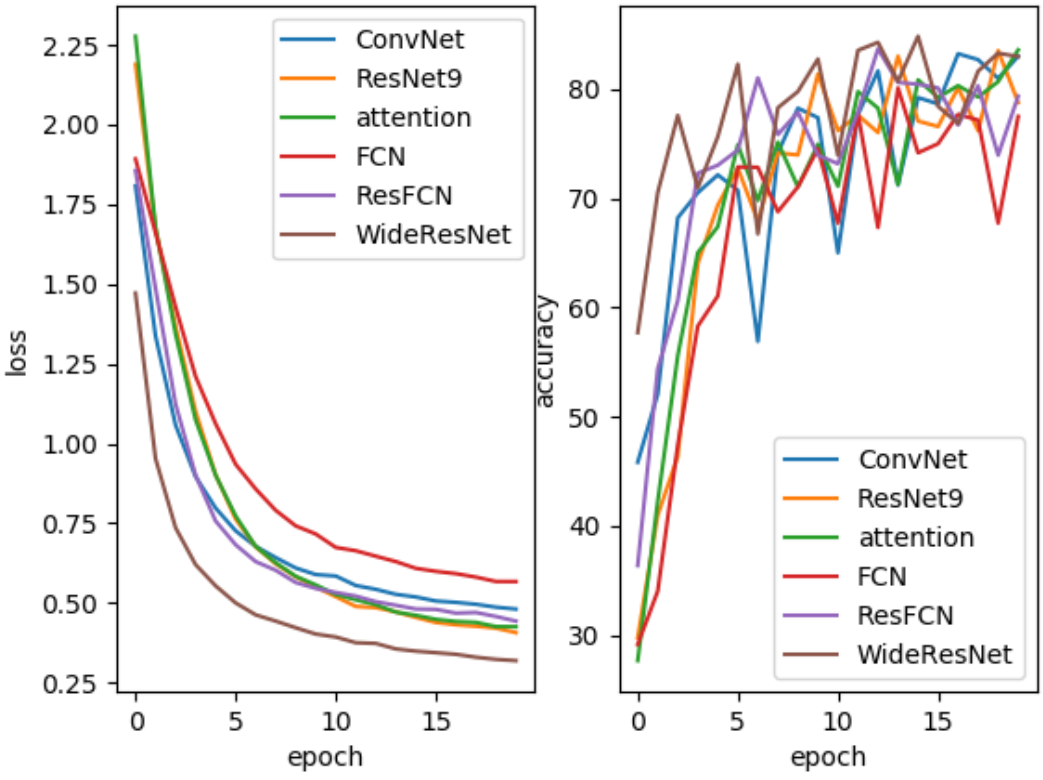
- 使用更大的卷积核和更多的通道数
- 使用更少的 ResBlock，增加每个 Block 中的卷积层
- 宽度的增大容易导致过拟合，使用 Dropout 进行正则化
- 将 Conv-BN-ReLU 的顺序调整为 BN-ReLU-Conv

测试以上网络的效果。为了进行对比，固定以下超参数和设置进行实验。

超参数/设置	值
learning rate	0.1
epoch	20
batch size	128
optimizer	SGD with momentum(0.9)
weight decay	5e-4
data augmentation	RandomCrop、RandomHorizontalFlip
activation function	ReLU
loss function	CrossEntropyLoss

结果如下：

模型	准确率	训练时间 (s)
ConvNet	83.210%	300.17
ResNet9	83.500%	329.59
Attention + ResNet9	83.560%	386.26
FullConvNet	80.090%	1654.24
ResFullConvNet	83.660%	1385.45
WideResNet	84.810%	1238.38



分析：

- 从准确率来看，WideResNet 在 20 个 epoch 的训练中效果最好，而 ResNet9\AttentionResNet\ResFullConvNet有着相近的效果
- ConvNet 和 ResNet9 的对比、FullConvNet 和 ResFullConvNet 的对比，都说明了残差连接确实起到很好的作用
- 前三个网络的参数量较少，训练时间也较短；后三个网络参数量更多，训练时间长，而效果也相对较好，说明了在结构相近的情况下，模型参数量增加，在能保证完成训练的条件下，往往效果更好
- 从训练曲线来看，WideResNet 的效果明显地好，最终的模型也将以 WideResNet 为基础进行训练

1.3 Loss Function

选择以下五种损失函数进行测试

1.3.1 CrossEntropyLoss

$$p_i = \frac{\exp(x_i)}{\sum \exp(x_i)}$$
$$l(x, y) = -\log(p_i),\ where\ y_i = 1$$

1.3.2 MSELoss

$$p_i = \frac{\exp(x_i)}{\sum \exp(x_i)}$$
$$l(x, y) = \|x - y\|_2$$

1.3.3 NLLLoss

实际上与 CrossEntropyLoss 一样，只不过在 Pytorch 中的实现不同

1.3.4 KLDivLoss

$$p_i = \frac{\exp(x_i)}{\sum \exp(x_i)}$$
$$l(x, y) = y_i(\log y_i - p_i),\ where\ y_i = 1$$

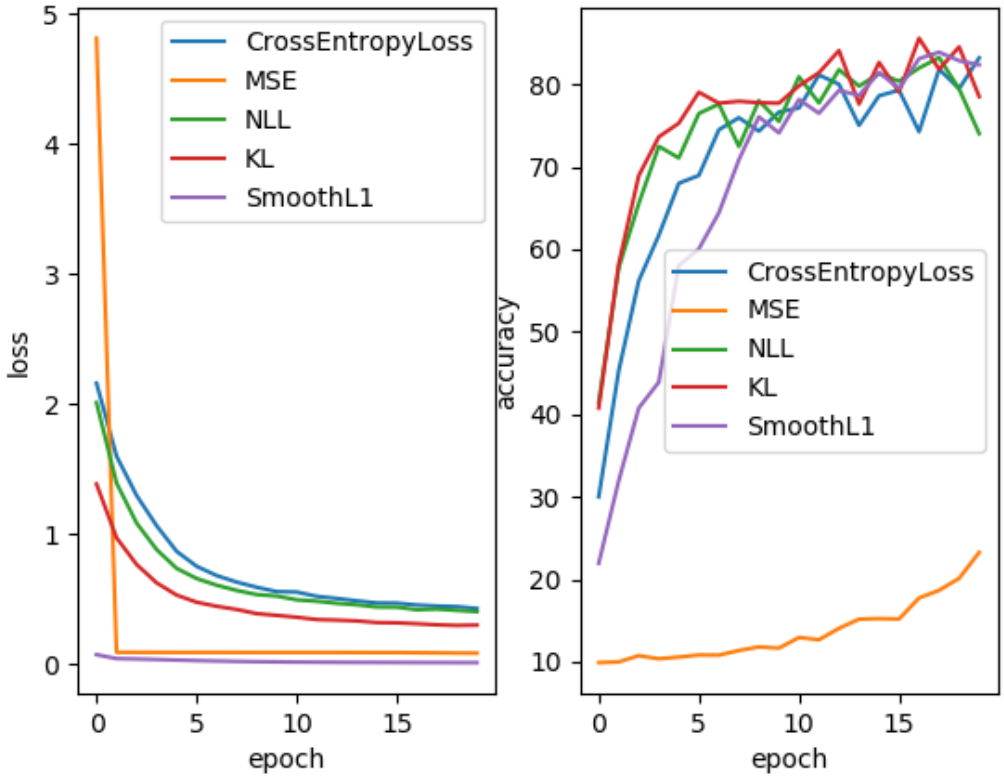
1.3.5 SmoothL1Loss

$$l(x, y) = \frac{1}{n}\sum_{i=1}^{n}\begin{cases}\frac{(x_i - y_i)^2}{2\beta}, & \text{if } |x - y| < \beta \\ |x_i - y_i| - \frac{1}{2}\beta, & o.w.\end{cases}$$

在 Pytorch 中 β 默认为 1，故在实验中也选择 $\beta = 1$

训练所用的网络为 ResNet9，其他设置和前面一致，结果如下：

损失函数	准确率
CrossEntropyLoss	83.200%
MSELoss	23.260%
NLLLoss	83.240%
KLDivLoss	85.580%
SmoothL1Loss	83.890%



分析：

- 相对其他损失函数，MSE 收敛速度极慢
- 以上几种损失函数中，表现最佳的为 KL 散度损失

1.4 Activation

选择以下几种激活函数进行测试

1.4.1 ReLU

$$\mathrm{ReLU}(x) = \max(x, 0)$$

1.4.2 Sigmoid

$$\mathrm{Sigmoid}(x) = \frac{1}{1 + \exp(-x)}$$

1.4.3 Tanh

$$\tanh(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)}$$

1.4.4 ELU

$$\mathrm{ELU}(x) = \max(0, x) + \min(0, \alpha(\exp(x) - 1))$$

这里取 $\alpha = 1$

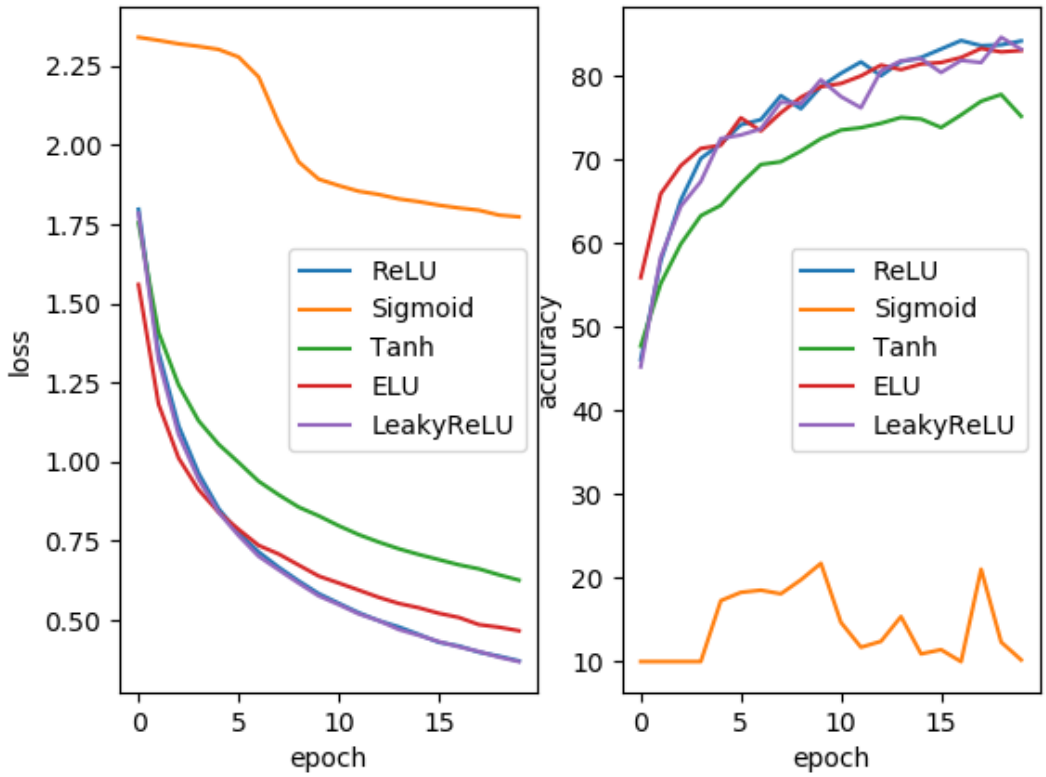
1.4.5 LeaklyReLU

$$\mathrm{LeaklyReLU}(x) = \begin{cases} x, & \text{if } x \geq 0 \\ ax, & o.w \end{cases}$$

这里取 $\alpha = 0.01$

训练所用的网络为 ConvNet，学习率调整为 $1e-3$，结果如下：

激活函数	准确率
ReLU	84.180%
Sigmoid	21.740%
Tanh	77.780%
ELU	83.280%
LeaklyReLU	84.620%



分析：

- 与之前几次实验不同，此处学习率选择为 $1e-3$，原因是过大的学习率会导致 ELU 损失变为 NaN
- 使用 Sigmoid 作为激活函数会导致不收敛

1.5 Optimizer

选择以下几种优化器进行测试

1.5.1 SGD(+Momentum)

略

1.5.2 Adam (Adaptive Moment Estimation)

$$
\begin{aligned}
g_t &= \nabla_\theta f_t(\theta_{t-1}) \\
m_t &= \beta_1 \cdot m_{t-1} + (1-\beta_1) \cdot g_t \\
v_t &= \beta_2 \cdot v_{t-1} + (1-\beta_2) \cdot g_t^2 \\
\hat{m}_t &= m_t / (1-\beta_1^t) \\
\hat{v}_t &= v_t / (1-\beta_2^t) \\
\theta_t &= \theta_{t-1} - \alpha \cdot \hat{m}_t / (\sqrt{\hat{v}_t + \epsilon})
\end{aligned}
$$

1.5.3 RMSProp (Root Mean Square prop)

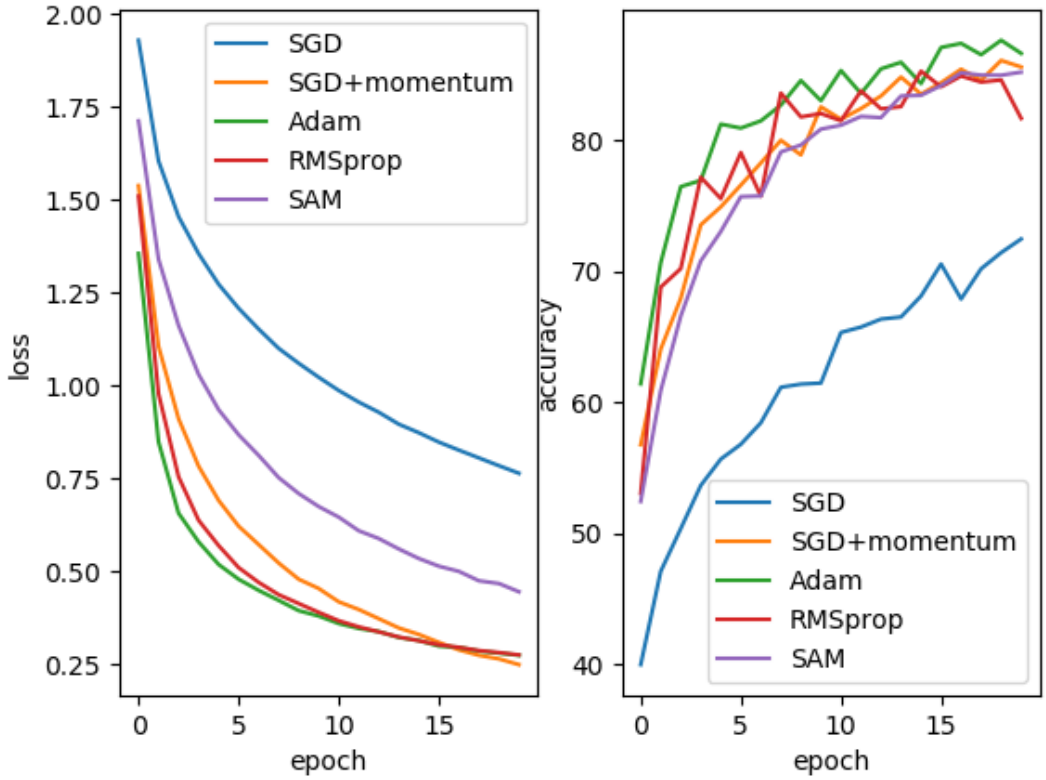
$$
\begin{aligned}
g_t &= \nabla_\theta f_t(\theta_{t-1}) \\
v_t &= \gamma \cdot v_{t-1} + (1-\gamma) \cdot g_t^2 \\
\theta_t &= \theta_{t-1} - \alpha \cdot g_t / (\sqrt{v_t + \epsilon})
\end{aligned}
$$

1.5.4 SAM (Sharpness-Aware Minimization)

$$
\begin{aligned}
\epsilon_t &= \rho \cdot \frac{\nabla_\theta f_t(\theta_{t-1})}{\|\nabla_\theta f_t(\theta_{t-1})\|_2} \\
\theta_t &= \theta_{t-1} - \alpha \cdot (\nabla_\theta f_t(\theta_{t-1} + \epsilon_t) + \lambda \cdot \theta_{t-1})
\end{aligned}
$$

训练所用的网络为 ResNet9，学习率为 $1e-3$，其他设置和前面一致，结果如下：

优化器	准确率	训练时间 (s)
SGD	72.440%	324.09
SGD+Momentum	86.060%	326.55
Adam	87.600%	330.00
RMSProp	85.240%	329.18
SAM	85.170%	555.70



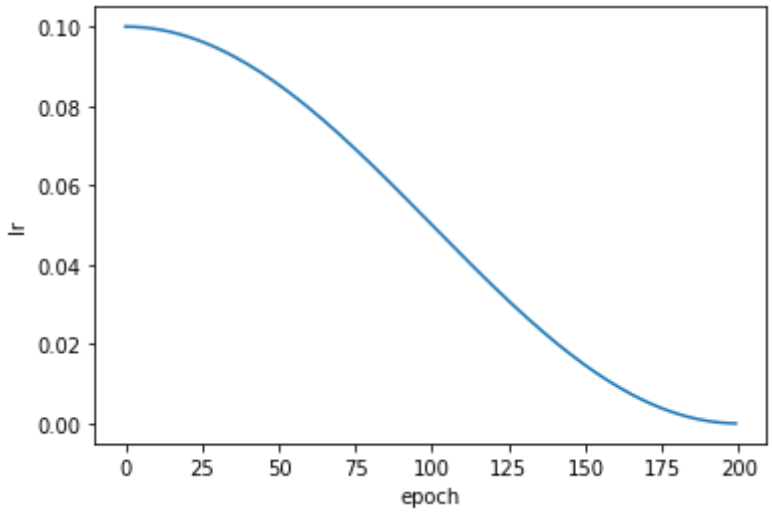
分析：

- 由于需要两步更新，所以 SAM 所需的时间要更多
- SGD 的收敛速度要远远慢于其他优化算法
- SAM 在损失大于剩余三个优化算法的同时，测试集上的准确率也能做到相当，说明 SAM 确实起到了降低泛化误差的作用

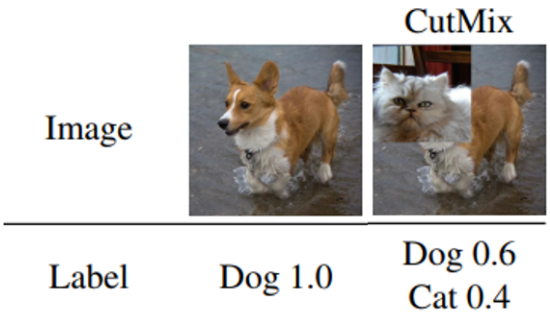
1.7 Final Model

基于以上的分析，最终选择 16 层的 WideResNet 作为模型，使用 SAM 优化器、KL 散度损失进行训练，并加入了 CutMix 的数据增强方法，使用余弦退火的学习率衰减策略，进行 200 个 epoch 的训练，最终得到测试集准确率为 97.330% 的模型。

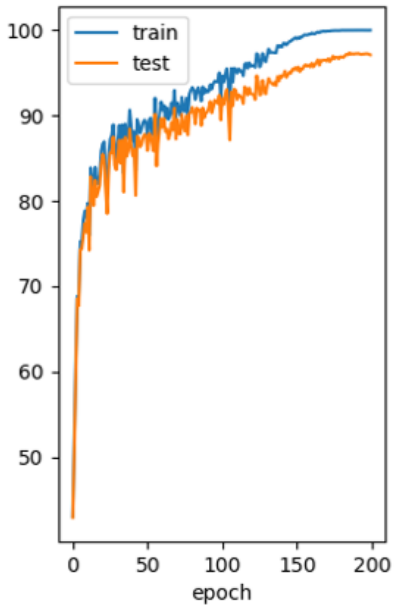
余弦退火：以余弦函数的形式做学习率衰减



CutMix：将两张样本图片按一定的比例裁剪在一起，生成新的样本，新样本的标签为原来样本标签按裁剪比例的凸组合 (下图来源于 CutMix 原论文)



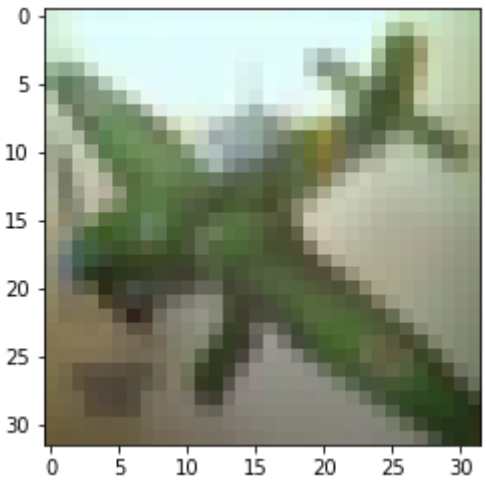
训练的准确率曲线如下：



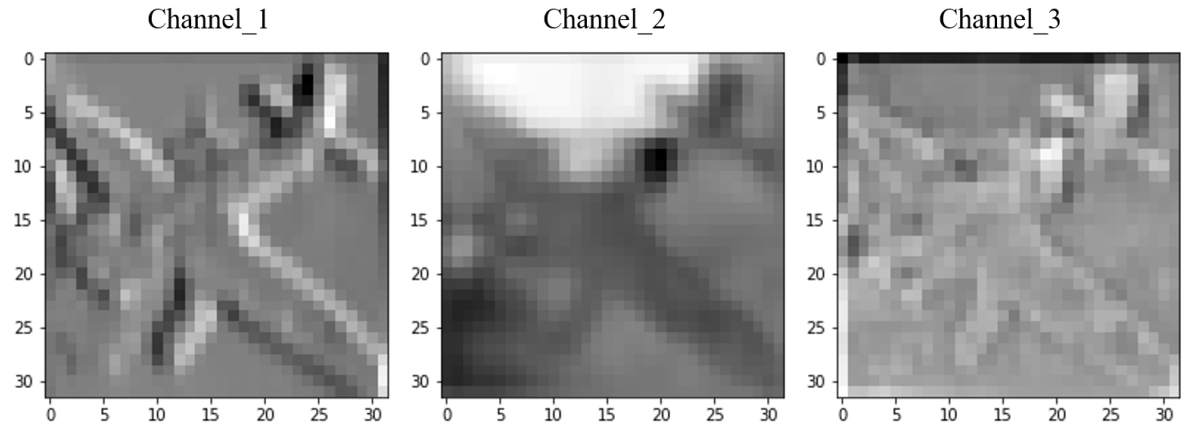
1.8 Visualization and Interpretation

对以上得到的最佳模型进行可视化和解释

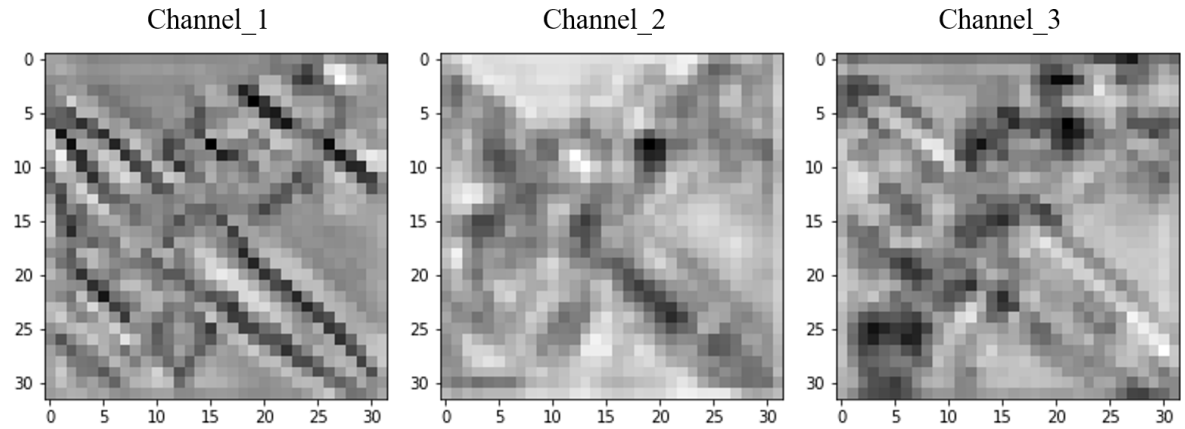
选取以以下图片进行可视化，标签为飞机



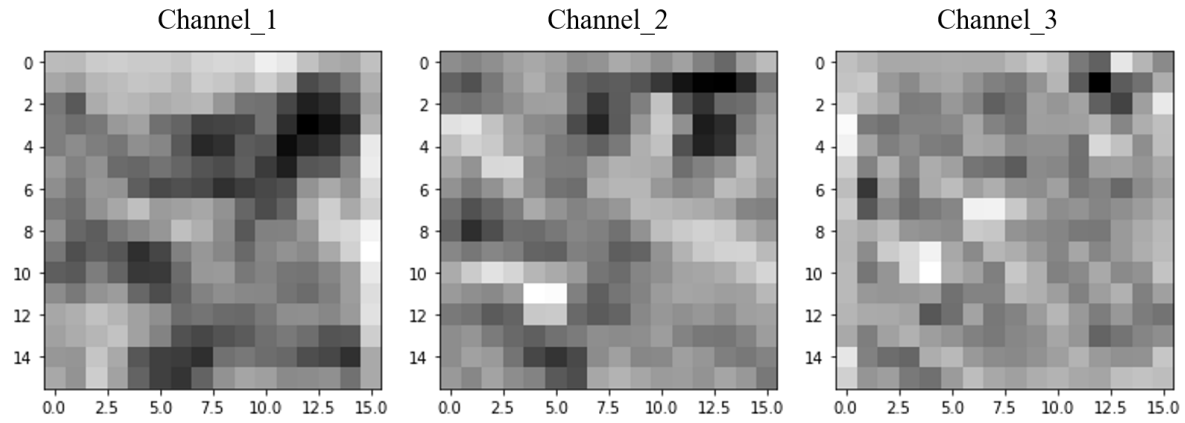
绘制经过第一层卷积后的前三个通道的图像：



绘制经过下一个 Block 后前三个通道的图像：

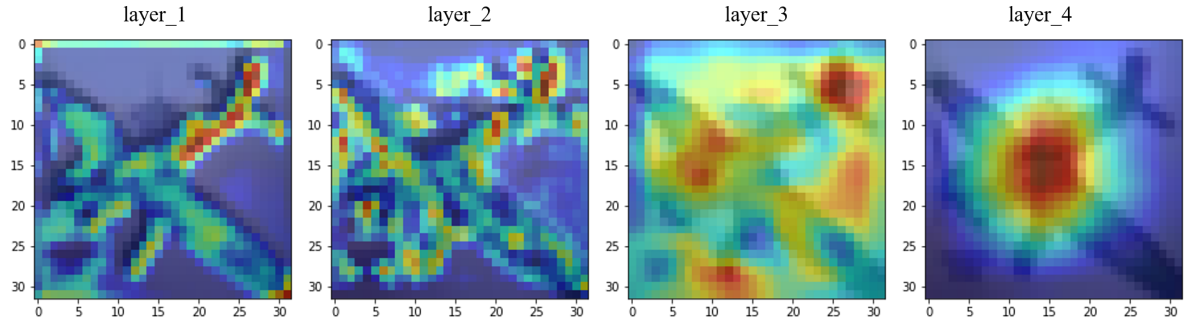


绘制经过第二个 Block 后前三个通道的图像：



从以上可视化中，可以看出，第一层卷积层的作用主要在于提取物体的形状信息，能将物体主要在的区域识别出来；第二层在第一层的基础上进一步提取图像的信息，但是已经较为抽象，只能看出大概的形状；更深层的图像提取到的信息更为抽象，难以解释。

接下来使用《Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization》中提出的 Grad-CAM 方法进行进一步解释，根据每层的梯度信息给每个神经元赋予重要值



从结果可以看出，对于这张图片而言，飞机的尾翼对网络的分类判断起了比较重要的作用。

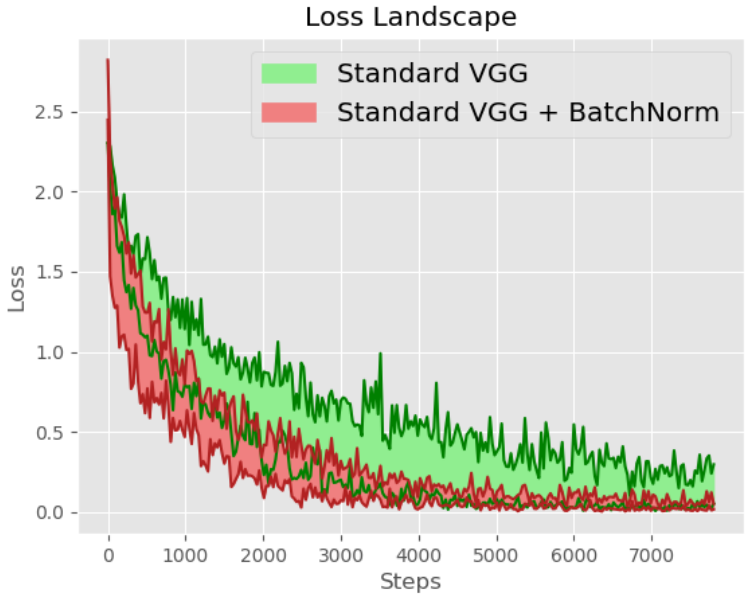
2 Batch Normalization

在原始的 VGG-A 模型中，在每个卷积层和激活函数增加一层 Batch Normalization 层，即可以得到 VGG-A-BatchNorm 模型

接下来从以下几个方面探究 Batch Normalization 如何对神经网络的训练起作用

2.1 Loss Landscape or Variation of the Value of the Loss

使用不同的学习率 $[1e-3, 8e-4, 6e-4]$ 训练 20 个 epoch，记录每次更新参数后的损失，结果如下：



实验结果分析：

- 与不使用 BN 相比，使用 BN 后损失下降得更快，最终模型的准确率也更高
- 使用 BN 后，学习率的改变对损失的影响变小，损失沿负梯度方向的方差减小

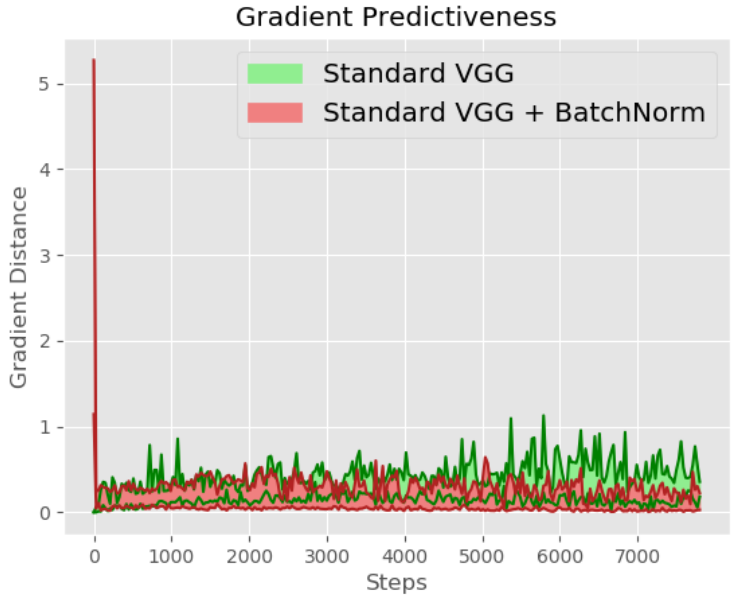
2.2 Gradient Predictiveness or the Change of the Loss Gradient

按照原论文的定义，令 l 表示损失，$W_k^{(t)}$ 表示在第 t 步迭代后第 k 层的参数

$$G_{t,i} = \nabla_{W_i(t)} l(W_1^{(t)}, \ldots, W_n^{(t)}; x^{(t)}, y^{(t)})$$
$$G'_{t,i} = \nabla_{W_i(t)} l(W_1^{(t+1)}, \ldots, W_{i-1}^{(t+1)}, W_i^{(t)}, \ldots, W_n^{(t)}; x^{(t)}, y^{(t)})$$

定义 internal covariate shift (ICS) 为 $\|G_{t,i} - G'_{t,i}\|$，也即为所谓的 Gradient Predictiveness. 注意更新的参数只包括第 i 层前的参数。此处定义的 ICS 反映了沿着负梯度方向的梯度变化情况，而只改变前面层的网络参数，能反映出层间的影响。

使用学习率 $[1e-3, 2e-3, 1e-4, 5e-4]$ 训练 20 个 epoch，记录第二个卷积层的 ICS，结果如下：



实验结果分析：

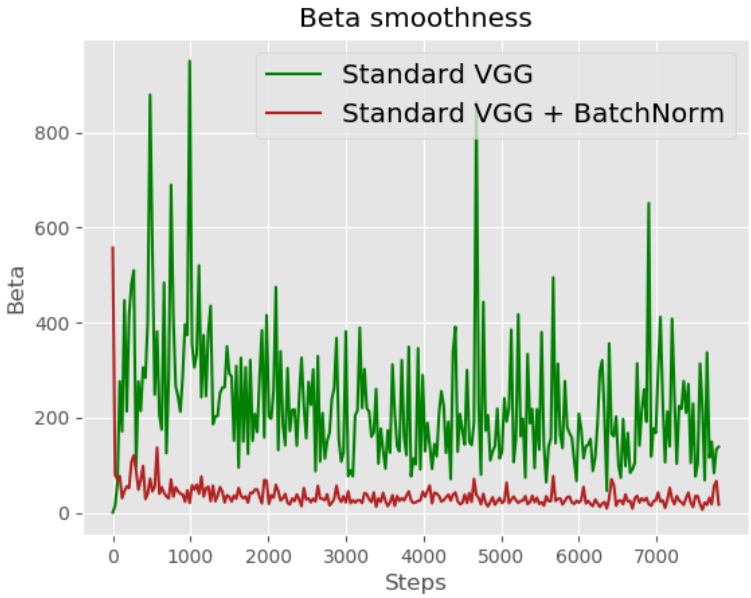
- 与不使用 BN 相比，使用 BN 后，ICS 有了比较明显的减小，同时它随着学习率的变化也在减小，这一点在训练后期尤为显著
- 这个现象体现了使用 BN 后，参数更新前后的梯度发生的改变变小，这体现了优化函数更加光滑，一阶近似更加准确，使用一阶优化算法在其上的收敛会加快

2.3 "Effective" β-Smoothness

"Effective" β-Smoothness 在此处指沿着负梯度方向的最大梯度变化，沿用 2.2 中的符号，即

$$\min_{lr} \frac{1}{lr} \|G'_{t,i} - G_{t,i}\| : W_k^{(t+1)} = W_k^{(t)} - lr \cdot G_{t,k}$$

其中 lr 不超过 0.4，因为通常过大的学习率会导致效果变差。为了简化，学习率只在 $[0.01, 0.05, 0.1, 0.2, 0.4]$ 中搜索。使用 0.01 的学习率训练 20 个 epoch，结果如下：



实验结果分析：

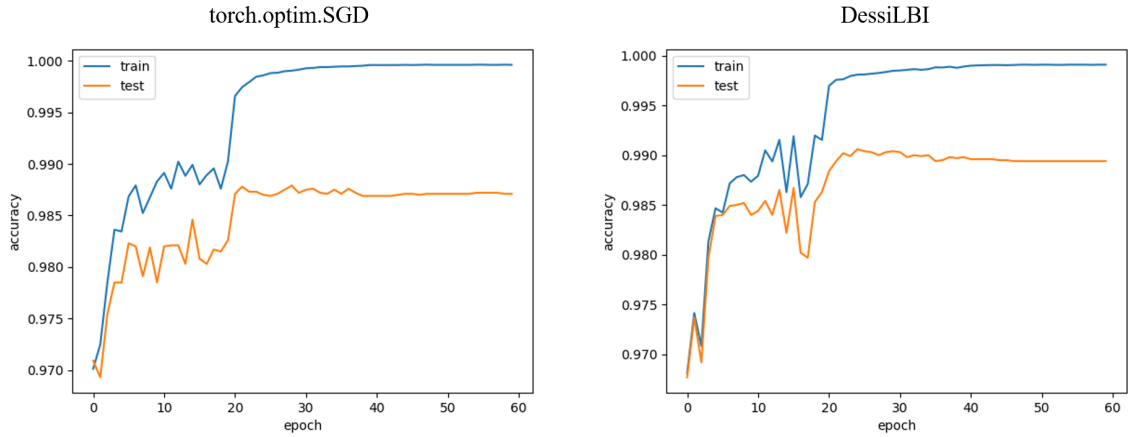
- β-Smoothness 是指 $\|\nabla f(x) - \nabla f(y)\| \leq \beta \|x - y\|_2$，所以以上定义的"Effective" β-Smoothness 反映了函数导数的 Lipschit 连续性
- 从以上的结果可以看出，加入 BN 后，"Effective" β-Smoothness 明显减小，更进一步说明了优化函数的光滑性，对优化起到很大的改进作用

3 DessiLBI

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。以下通过几个实验来说明 DessiLBI 的有效性。

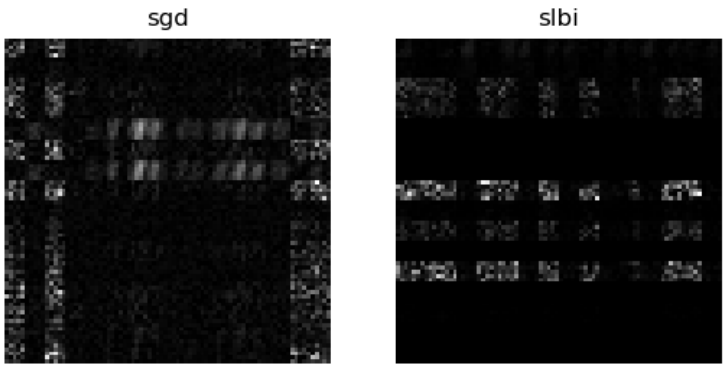
3.1 DessiLBI on MNIST

使用 Lenet 网络，分别利用带动量的 SGD 和 DessiLBI 对 MNIST 数据集进行训练，训练曲线如下：



从曲线中可以看出，相对于 SGD，DessiLBI 的训练集准确率和测试集准确率之间的差距更小，同时最终的测试集准确率也较高，说明 DessiLBI 确实起到了正则化的效果，使得模型的过参数化得以改进。

再查看两个模型相同层的部分权重 (conv3)，结果如下：



发现在准确率相近的情况下，DessiLBI 确实实现了更为稀疏的结构。

对模型进行剪枝，再观察剪枝后的测试准确率，结果如下：

只剪去一层 (conv3)：

Percentage	Accuracy
0	0.9894
20	0.9894
40	0.9894
60	0.9888
80	0.9872

剪去两层 (conv3, fc1)：

Percentage	Accuracy
0	0.9894
20	0.9894
40	0.9894
60	0.9888
80	0.9872

以上结果说明本身的稀疏性已经较高，并且经过剪枝模型的复杂度进一步降低，而准确率并没有发生很大的变化，说明本身模型确实是过参数化的，存在更为稀疏的网络结构。

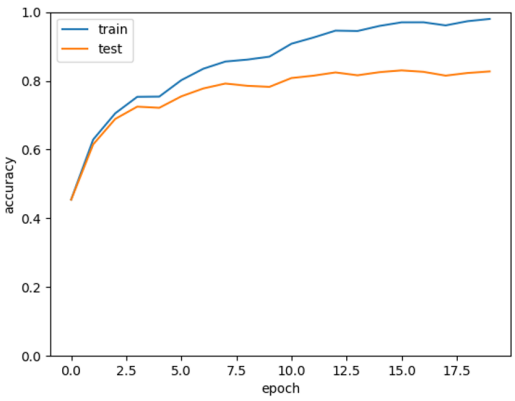
3.2 DessiLBI with Adam

将 Adam 的更新方式用于更新 DessiLbI 的 W 上，只需修改 slbi_opt.py 中的 step 函数，主要修改如下：

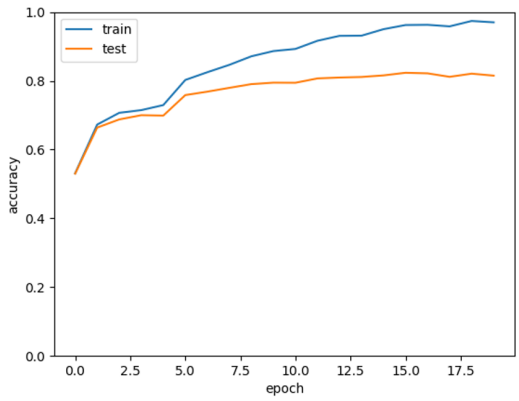
```
if 'step' not in param_state:
    param_state['step'] = 0
    param_state['exp_avg'] = torch.zeros_like(p)
    param_state['exp_avg_sq'] = torch.zeros_like(p)

param_state['step'] += 1

# 获取step等
step = param_state['step']
exp_avg = param_state['exp_avg']
exp_avg_sq = param_state['exp_avg_sq']

# bias-corrected estimate
bias_corr1 = 1 - beta_1 ** step
bias_corr2 = 1 - beta_2 ** step

exp_avg.mul_(beta_1).add_(d_p, alpha=1-beta_1)
exp_avg_sq.mul_(beta_2).addcmul_(d_p, d_p, value=1-beta_2)

step_size = lr_kappa / bias_corr1
p.data.addcdiv_(exp_avg, (exp_avg_sq.sqrt() /
math.sqrt(bias_corr2)).add_(epsilon), value=-step_size)
```

使用 VGG 网络，分别利用 torch.optim.Adam 和 DessiLBI 对 CIFAR10 数据集进行训练，训练曲线如下：

Torch.optim.Adam



DessiLBI



从曲线可以得出和以上相同的结论，即DessiLBI 的训练集准确率和测试集准确率之间的差距更小，同时最终的测试集准确率相近，说明 DessiLBI 确实起到了正则化的效果，使得模型的过参数化得以改进。

对模型进行剪枝，再观察剪枝后的测试准确率，结果如下：

只剪去一层 (conv3)：

Percentage	Accuracy
0	0.8318
5	0.7934
10	0.7757
15	0.7364
20	0.7117

发现此时模型本身的稀疏性并不高，而且剪枝比例也不能过大；随着剪枝比例增大，模型的准确率有了明显的下降。推测原因是 CIFAR10 数据集更复杂，而使用 VGG 提取出来的特征都比较重要。

4. References

[1] https://www.cs.toronto.edu/~kriz/cifar.html

[2] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).

[3] Woo, S., Park, J., Lee, J. Y., & Kweon, I. S. (2018). Cbam: Convolutional block attention module. In Proceedings of the European conference on computer vision (ECCV) (pp. 3-19).

[4] Zagoruyko, S., & Komodakis, N. (2016). Wide residual networks. arXiv preprint arXiv:1605.07146.

[5] Selvaraju, R. R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., & Batra, D. (2017). Grad-cam: Visual explanations from deep networks via gradient-based localization. In Proceedings of the IEEE international conference on computer vision (pp. 618-626).

[6] Santurkar, S., Tsipras, D., Ilyas, A., & Madry, A. (2018). How does batch normalization help optimization?. Advances in neural information processing systems, 31.

[7] Fu, Y., Liu, C., Li, D., Sun, X., Zeng, J., & Yao, Y. (2020, November). Dessilbi: Exploring structural sparsity of deep networks via differential inclusion paths. In International Conference on Machine Learning (pp. 3315-3326). PMLR.