Project-2 of Neural Network and Deep Learning

21307140069 田沐钊

1 CIFAR-10 分类网络

1.1 网络结构简述

该部分的大多数网络结构均基于ResNet进行设计，其由何凯明等人在2015年提出。其被提出的原因是，在进行深层网络学习的过程中，总有两个避不开的问题，即梯度消失/爆炸和网络退化。

以三层的神经网络为例，每一层的线性层和非线性层可以表示为：

$$
\begin{aligned}
Z^{[L]} &= W^{[L]} * A^{[L-1]} + b^{[L]} \\
A^{[L]} &= g^{[L]}(Z^{[L]})
\end{aligned}
$$

假设现在要计算第一层$W^{[1]}$的梯度，那么根据链式法则，有：

$$
\begin{aligned}
\frac{\partial LOSS}{\partial W^{[1]}} &= \frac{\partial LOSS}{\partial A^{[3]}} \frac{\partial A^{[3]}}{\partial Z^{[3]}} \frac{\partial Z^{[3]}}{\partial A^{[2]}} \frac{\partial A^{[2]}}{\partial Z^{[2]}} \frac{\partial Z^{[2]}}{\partial A^{[1]}} \frac{\partial A^{[1]}}{\partial Z^{[1]}} \frac{\partial Z^{[1]}}{\partial W^{[1]}} \\
&= \frac{\partial LOSS}{\partial A^{[3]}} {g^{[3]}}' W^{[3]} {g^{[2]}}' W^{[2]} {g^{[1]}}' \frac{\partial Z^{[1]}}{\partial W^{[1]}}
\end{aligned}
$$

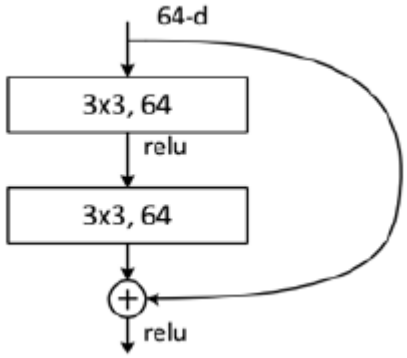
如果在神经网络中，多层都满足${g^{[L]}}' W^{[L]} > 1$，则越往下的网络层的梯度越大，这就造成了**梯度爆炸**的问题。反之，若多层都满足 ${g^{[L]}}' W^{[L]} < 1$，则越往下的网络层梯度越小，引起**梯度消失**的问题。而在深度学习网络中，为了让模型学到更多非线性的特征，在激活层往往使用例如**sigmoid**这样的激活函数。对sigmoid来说，其导数的取值范围在 $(0, \frac{1}{4}]$，在层数堆叠的情况下，更容易出现梯度消失的问题。

面对梯度消失/爆炸的情况，可以通过Normalization等方式解决，使得模型最终能够收敛。

但是，在模型能够收敛的情况下，网络越深，模型的准确率越低，同时，模型的准确率先达到饱和，此后迅速下降。这个情况我们称之为**网络退化（Degradation）**。这个现象也不是因为overfitting所引起的，因为在训练集上，深层网络的表现依然更差。其原因一般认为是因为结构过于复杂的模型虽然表达能力更强，但是参数学习难度更高，故更难进行稳定地训练。

最暴力的构造恒等映射的方法，就是在相应网络部分的尾端增加一层学习层$W^{[IM]}$，来满足输出和输入逼近。但是本来深网络要学的参数就很庞大了，再构造新的参数层，又增加了模型的复杂度。

为了解决这种问题，我们引入了残差块的结构：



如图所示，这个残差模块包含了神经网络中的两层，其中， X 表示输入，$\mathcal{F}(X)$ 表示过这两层之后的结果，$\mathcal{H}(X)$ 表示恒等映射，则在这样的构造方式下，恒等映射可以写成：

$$\mathcal{H}(X) = \mathcal{F}(X) + X$$

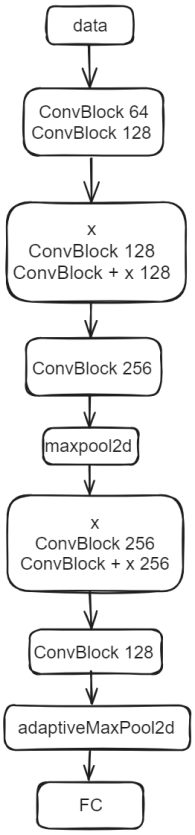
$\mathcal{F}(X)$就被称之为**残差函数（residule function）。**在网络深层的时候，在优化目标的约束下，模型通过学习使得 $\mathcal{F}(X)$ 逼近0 **(residule learning)**，让深层函数在学到东西的情况下，又不会发生网络退化的问题。

通过这样的构造方式，让 $\mathcal{F}(X)$ 嵌套在了 $\mathcal{H}(X)$ 中，这样跃层构造的方式也被称为**残差连接(residule connection)/ 跳跃连接(skip connection)/短路(shortcuts)。**模型并不是严格的跨越2层，可以根据需要跨越3，4层进行连接。同时，该等式是在假设输入输出同维，即 $\mathcal{F}(X)$ 和X 同维的情况，不同维时，只需要在 X 前面增加一个转换矩阵 W_s 即可

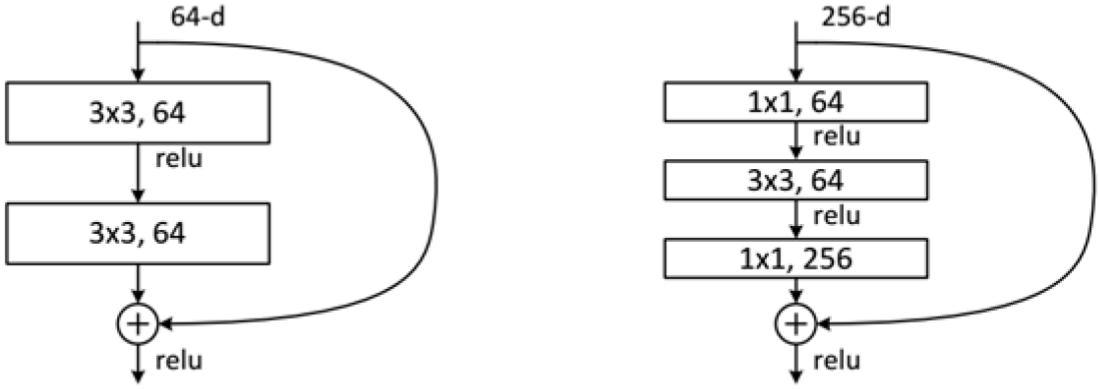
layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$
	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

原论文实现的ResNet中，最小的模型为Res18，该部分首先对该模型进行了复现，并将网络的第一部分由**卷积核(7x7)+最大池化(3x3)**修改为了**卷积核+批归一化+激活函数。**

此外，为了进一步减小模型参数量，我们构建了层数更少的 Res9 用于训练测试，其网络结构如下：



此外，ResNet实际上有两种Res Block可供使用，如下图所示。其中，左图的Basic Block 一般用于34层以下的较小模型，而对于较深的模型，一般会使用右图的 Bottleneck Block 来平衡模型的参数量使其不会过高，从而使模型的性能整体得到提升。



在这里，我们尝试将ResNet的残差块替换成简化后的 Bottleneck Block，使其参数量进一步减少，我们将得到的模型称为 simp_Res9。

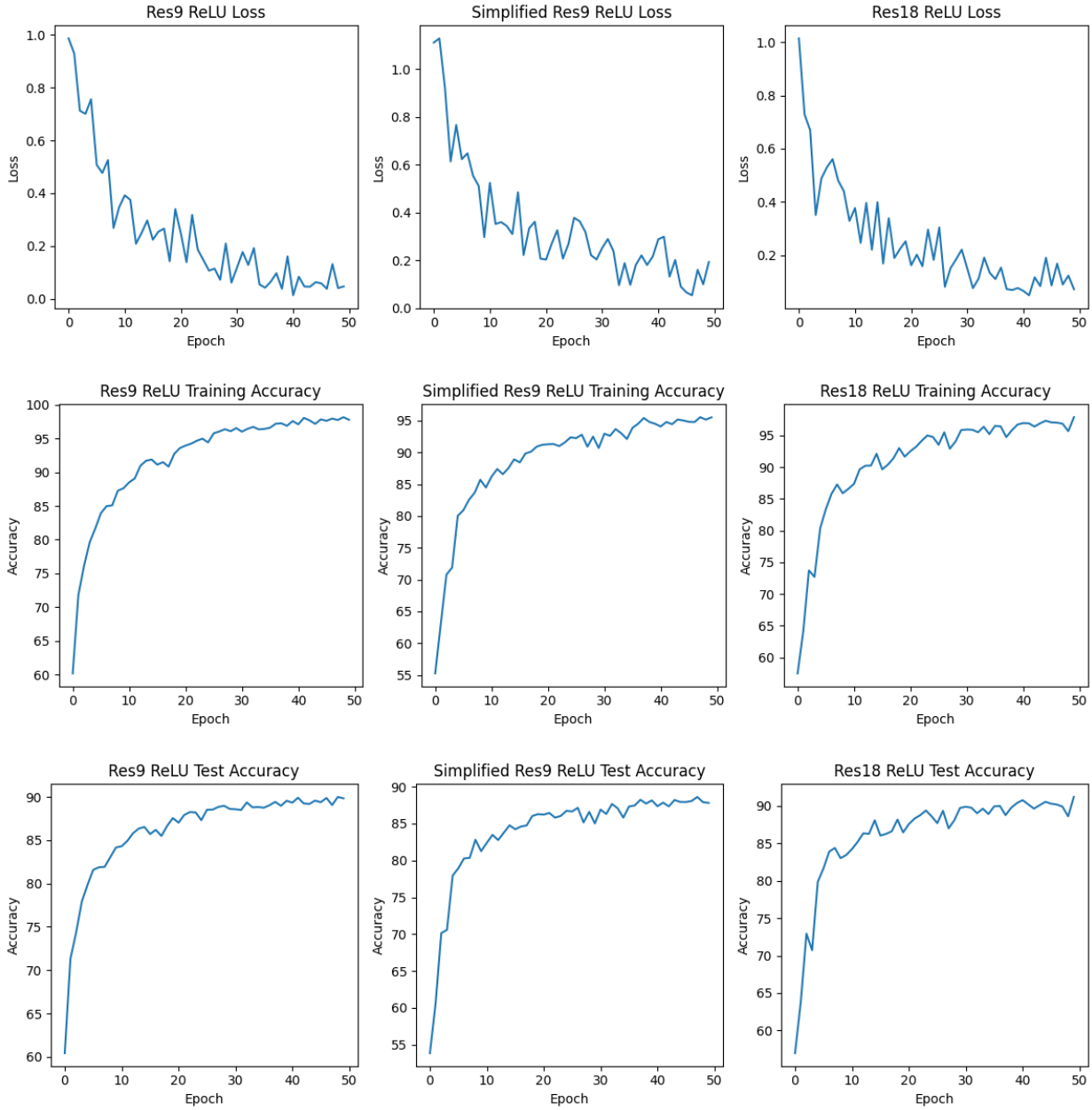
1.2 不同模型结构（神经元个数）的性能比较

在这里对Res18、Res9、simp_Res9三种不同结构的模型分别进行训练，对其模型性能进行比较如下。

训练的超参数设置为

optimizer	weight decay	momentum	epoch	lr	loss
SGD	5e-4	0.9	50	1e-2	cross entropy

训练中的损失函数曲线、训练集准确率曲线、测试集准确率曲线比较如下：



最终的训练结果统计如下：

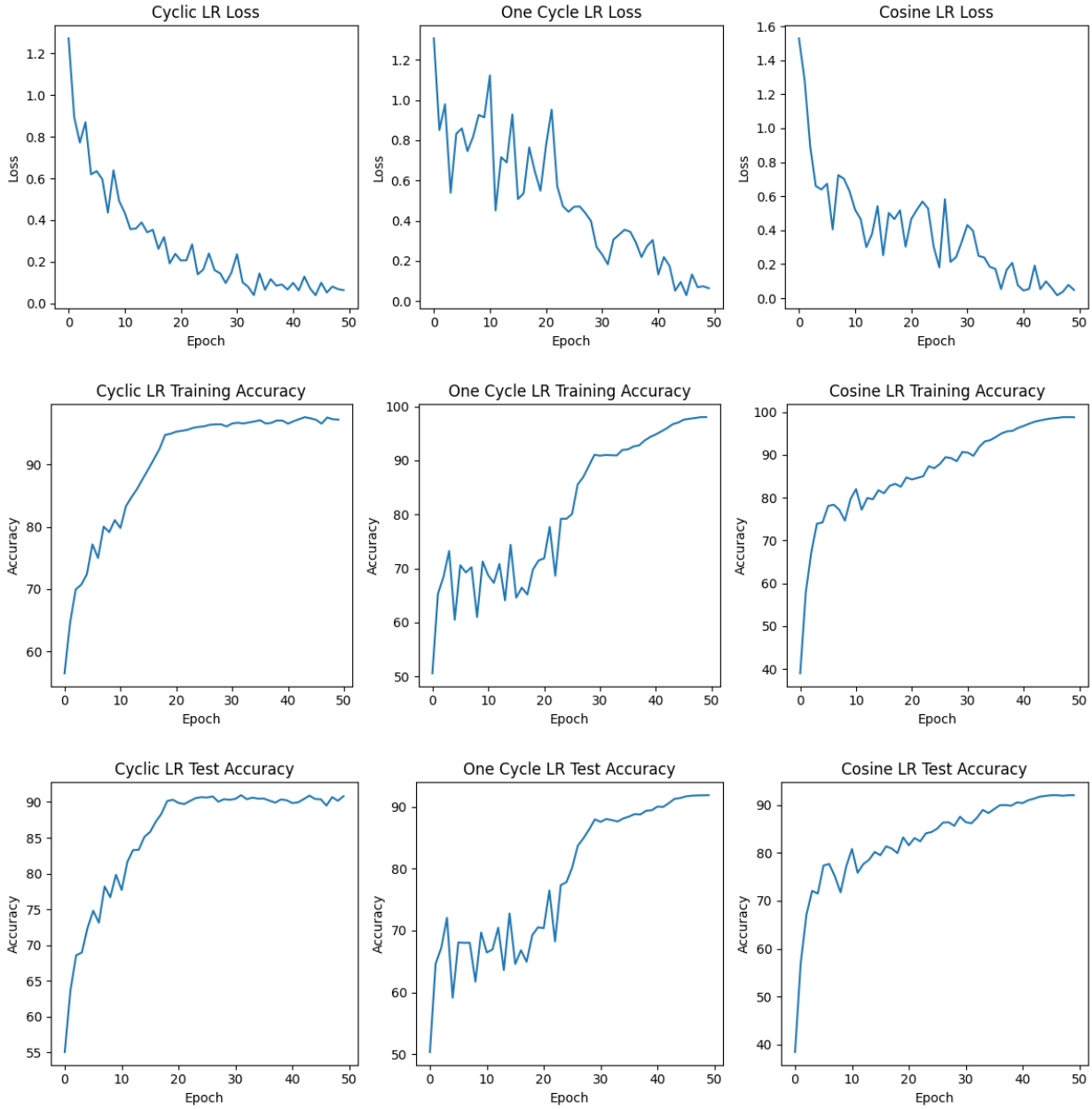
model	test accuracy（%）	time cost（s）	模型体积（MB）
Res18	91.210	1037.82	42.7
Res_9	89.830	748.32	8.7
simp_Res9	87.800	730.96	3.41

可以看到，相比之下小模型的loss曲线在前期下降更为迅速，并且在预测准确率上也并没有显著过多弱于较大的模型。从结果上来看，参数量较大的模型确实能够在设定的迭代次数下获得更高的准确率，但收益较低，与结构更简单的模型差距不大，而后者还在训练耗时和模型体积上具有明显的优势。

1.3 不同学习率策略的训练效果比较

下面我们尝试不同的学习率策略，探究不同学习率策略对模型训练效果的影响。这里我们选用simp_Res9 作为研究对象，其他超参数不变，分别对其使用CosineAnnealingLR、CyclicLR、OneCycleLR三种学习率策略进行训练。

下面是三种学习率策略的训练效果的比较：



下面是模型的训练结果比较：

scheduler	train accuracy（%）	test accuracy（%）
CyclicLR	97.212	90.810
OneCycleLR	98.014	91.880
CosineAnnealingLR	98.820	92.080

基于训练数据，对三种学习率策略进行分析比较如下：

1. **CosineAnnealingLR**:
    - 特点:这种策略会将学习率随着训练epoch的增加而逐渐降低,遵循余弦函数的曲线。
    - 优点:学习率变化平滑,能够帮助模型在训练后期更好地收敛。对于较复杂的模型训练非常有帮助。
    - 缺点:由于学习率单调下降,无法在训练过程中重新提高学习率,可能会限制模型进一步提升性能。
2. **CyclicLR**:
    - 特点:这种策略会让学习率在预设的上下界之间周期性地变化,形成一个"波浪"型曲线。
    - 优点:通过周期性地提高和降低学习率,可以帮助模型跳出局部最优,探索更好的解。对于一些复杂的优化问题很有帮助。并且loss的下降更为迅速。
3. **OneCycleLR**:
    - 特点:这种策略会先快速提高学习率到一个很高的峰值,然后再快速降低至很低的水平。全过程只有一个循环。
    - 优点:充分利用了前期高学习率有助于快速逃离鞍点,后期低学习率有助于精细调整的特点。训练效率和最终效果都可能优于其他策略。
    - 缺点:训练不稳定，loss曲线和准确率曲线均发生较大波动，难以平稳收敛。

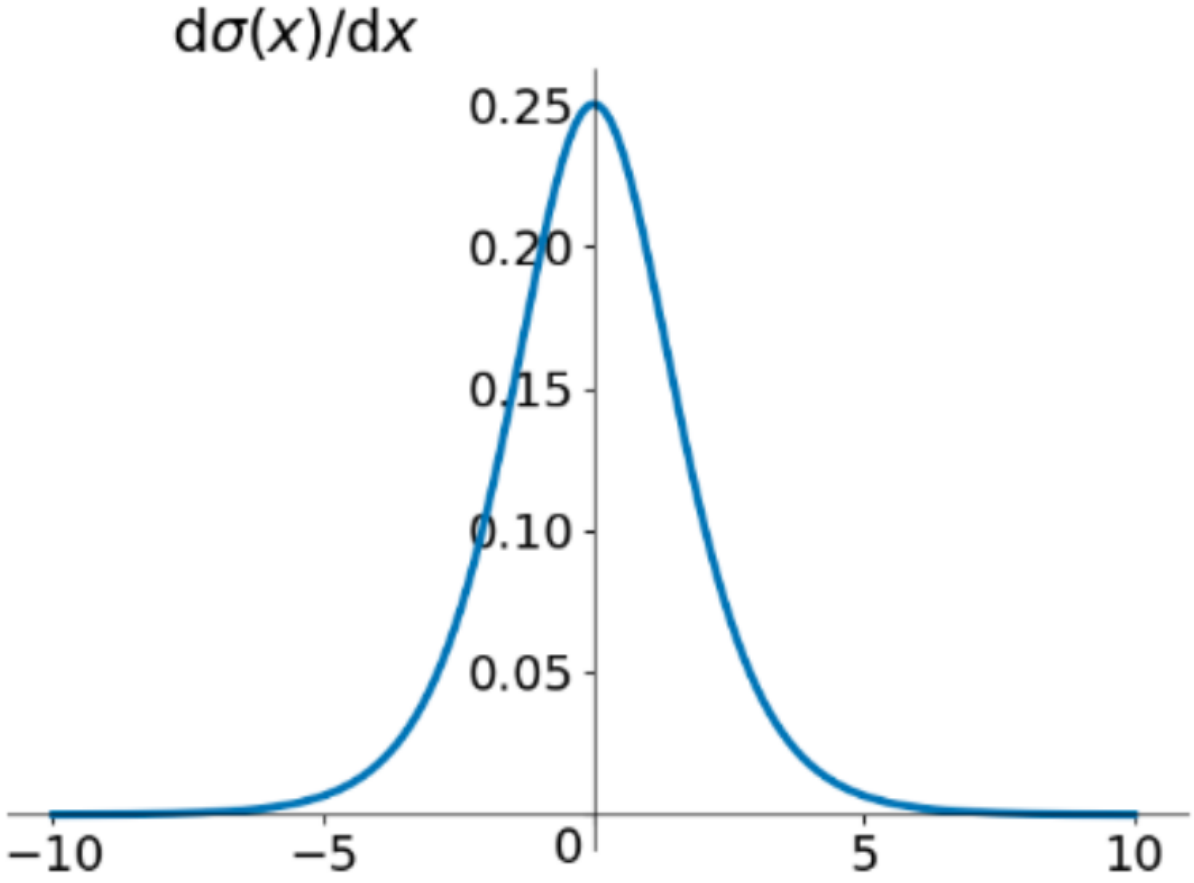
而从实验结果来看：

- CosineAnnealingLR在训练和测试准确率上都取得了最好的结果,这说明它能够帮助模型较好地拟合训练数据,同时也能较好地泛化到测试集。这主要得益于它平滑的学习率下降策略,能够更好地引导模型收敛。
- OneCycleLR的表现也非常出色,训练和测试准确率都很高,仅次于CosineAnnealingLR。这说明它能在训练前期快速逃离局部最优,后期又能较好地收敛,兼顾了训练效率和最终效果。
- CyclicLR在训练和测试准确率上的表现相对较差,可能是因为它的周期性学习率变化对于这个特定任务而言并不太适合,导致模型难以有效收敛。

1.4 不同激活函数的模型性能比较

在该部分中，我们尝试多种激活函数，探究不同激活函数对模型性能的影响。这里我们依然选用simp_Res9作为研究对象，其他超参数不变，分别使用ReLU、Mish、ELU、RReLU、GELU五种激活函数进行实验探究。

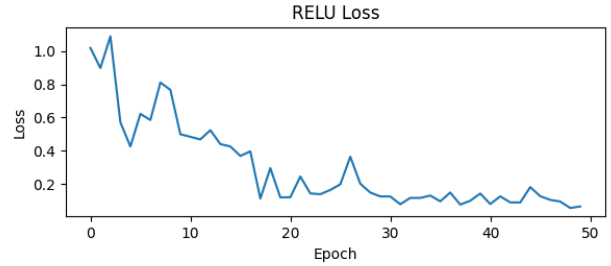
这里不选择对sigmoid函数进行探究，因为由sigmoid函数的导数图象可以看到：

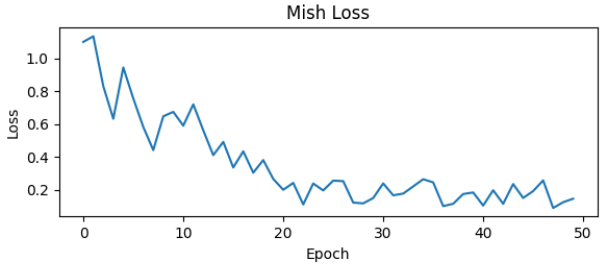


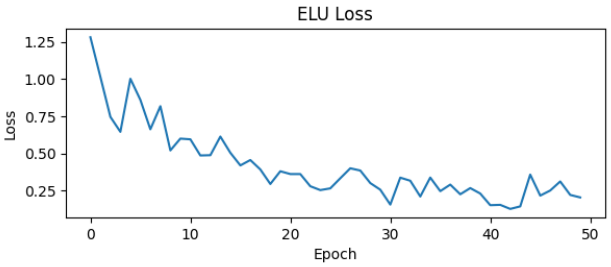
$$\frac{\partial C}{\partial b_1} = \sigma'(z_1)w_2\sigma'(z_2)w_3\sigma'(z_3)w_4\sigma'(z_4)\frac{\partial C}{\partial a_4}$$

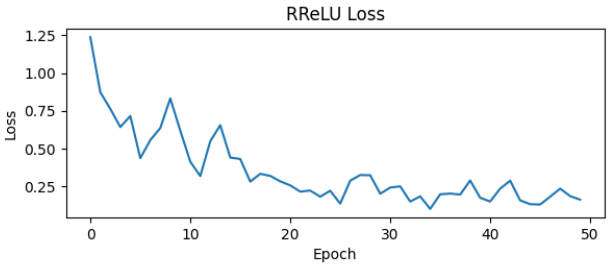
当x=0时，能够取导数最大值0.25，而在反向传播求导公式中，w是我们初始化地均值为0标准差为1的权重，那么大量小于1的值连乘，会使结果迅速变小，出现梯度消失现象，无法对参数进行更新，因此不选用sigmoid激活函数。

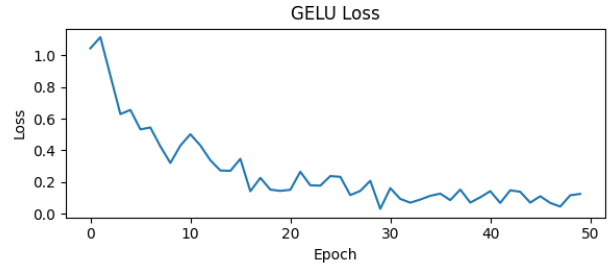
下面是不同激活函数的训练效果的比较：

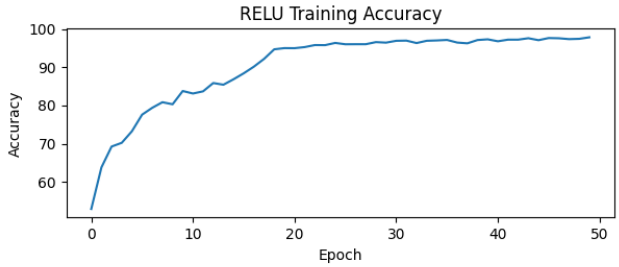

RELU Loss
Loss
Epoch

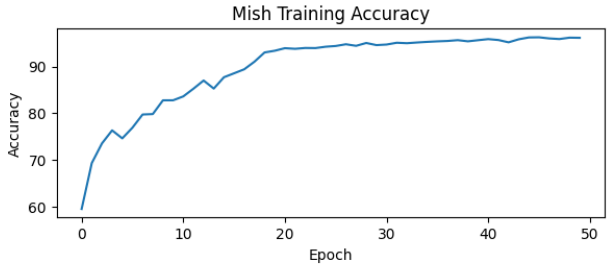

Mish Loss
Loss
Epoch

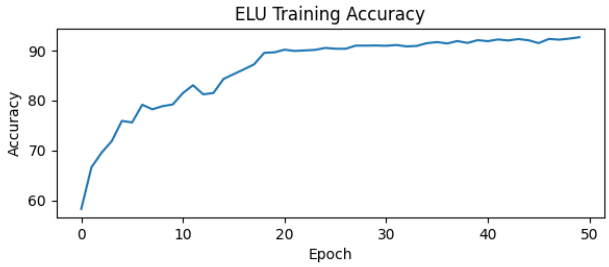

ELU Loss
Loss
Epoch

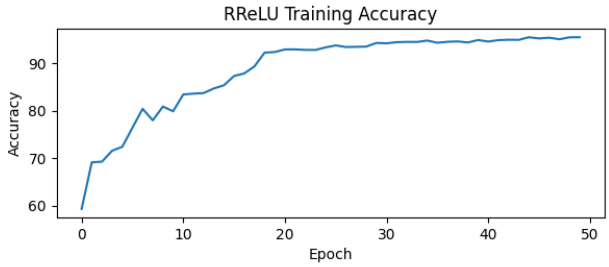

RReLU Loss
Loss
Epoch

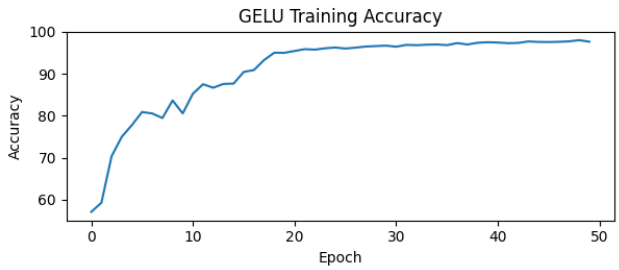

GELU Loss
Loss
Epoch

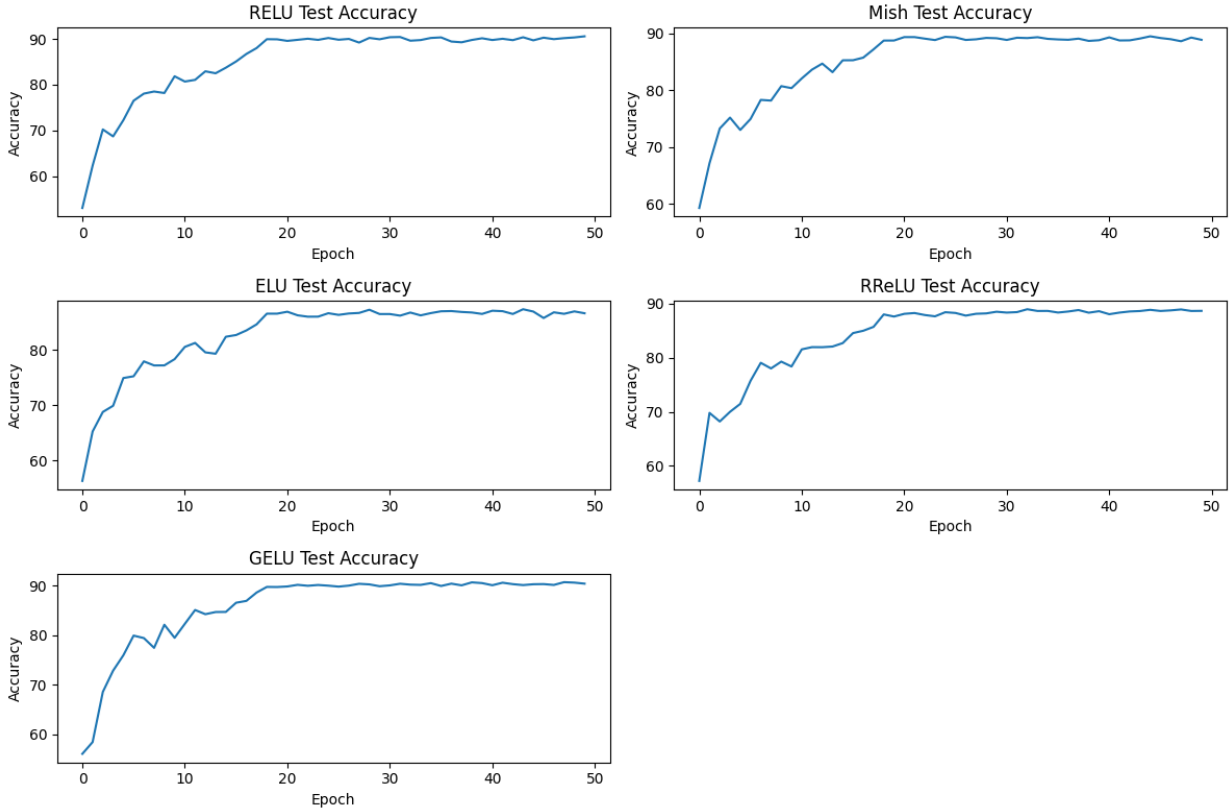

RELU Training Accuracy
Accuracy
Epoch


Mish Training Accuracy
Accuracy
Epoch


ELU Training Accuracy
Accuracy
Epoch


RReLU Training Accuracy
Accuracy
Epoch


GELU Training Accuracy
Accuracy
Epoch



其训练出的模型的最终效果如下：

activation	train accuracy（%）	test accuracy（%）
ReLU	97.994	90.580
Mish	96.282	88.860
ELU	92.860	86.660
RReLU	95.568	88.660
GELU	97.648	90.460

基于训练数据，对五种激活函数的效果分析如下：

1. **Loss曲线分析**:
    - ReLU: Loss曲线下降较快,但在训练过程中可能出现"震荡"现象,表明模型训练不太稳定。
    - Mish: Loss曲线下降更加平稳,表明训练更加稳定。相比ReLU,Mish能够更好地避免"死亡神经元"的问题。
    - ELU: Loss下降相对平缓,但最终达到的loss值较低。ELU的平稳性可能会使模型训练过程更加稳定。
    - RReLU: Loss曲线波动较大,表明训练过程不太稳定。可能需要仔细调整负斜率的取值范围。
    - GELU: Loss下降速度适中,最终loss值较低。GELU的平滑特性可能会带来更优的训练收敛性。
2. **准确率曲线分析**:
    - ReLU: 训练准确率上升较快,但测试准确率提升相对较慢,可能存在过拟合问题。

- Mish: 训练和测试准确率都提升较快,而且最终精度较高。Mish能够较好地平衡训练集拟合和泛化性能。
- ELU: 训练准确率提升较慢,但测试准确率提升较快,泛化性能较好。
- RReLU: 训练准确率波动较大,测试准确率提升也较慢,可能存在过拟合和训练不稳定的问题。
- GELU: 训练和测试准确率都提升较快,最终精度较高。GELU可能能够兼顾拟合和泛化。

根据提供的最终准确率数据,我们可以进一步分析这五种激活函数在ResNet模型训练中的表现:

1. **ReLU**:
   - 训练准确率最高,达到97.994%,表明模型在训练集上拟合效果很好。
   - 测试准确率也相对较高,达到90.580%,说明模型具有较强的泛化能力。
   - 训练和测试准确率差距较大(约7.414个百分点),可能存在一定程度的过拟合。
2. **Mish**:
   - 训练准确率较高,达到96.282%,表明模型也能很好地拟合训练数据。
   - 测试准确率为88.860%,略低于ReLU,但仍保持在较高水平。
   - 训练和测试准确率差距较小(约7.422个百分点),说明Mish可以较好地平衡拟合和泛化。
3. **ELU**:
   - 训练准确率为92.860%,相对较低,可能存在欠拟合的问题。
   - 测试准确率为86.660%,也处于相对较低水平。
   - 训练和测试准确率差距较小(约6.2个百分点),说明ELU可能更倾向于追求较强的泛化性能。
4. **RReLU**:
   - 训练准确率为95.568%,相对较高,但仍略低于ReLU。
   - 测试准确率为88.660%,与Mish相当,但略低于ReLU和GELU。
   - 训练和测试准确率差距约6.908个百分点,表明RReLU可能存在一定程度的过拟合。
5. **GELU**:
   - 训练准确率为97.648%,与ReLU相当,说明GELU同样能够很好地拟合训练数据。
   - 测试准确率为90.460%,仅次于ReLU,体现了较强的泛化性能。
   - 训练和测试准确率差距约7.188个百分点,介于ReLU和Mish之间,平衡了拟合和泛化。

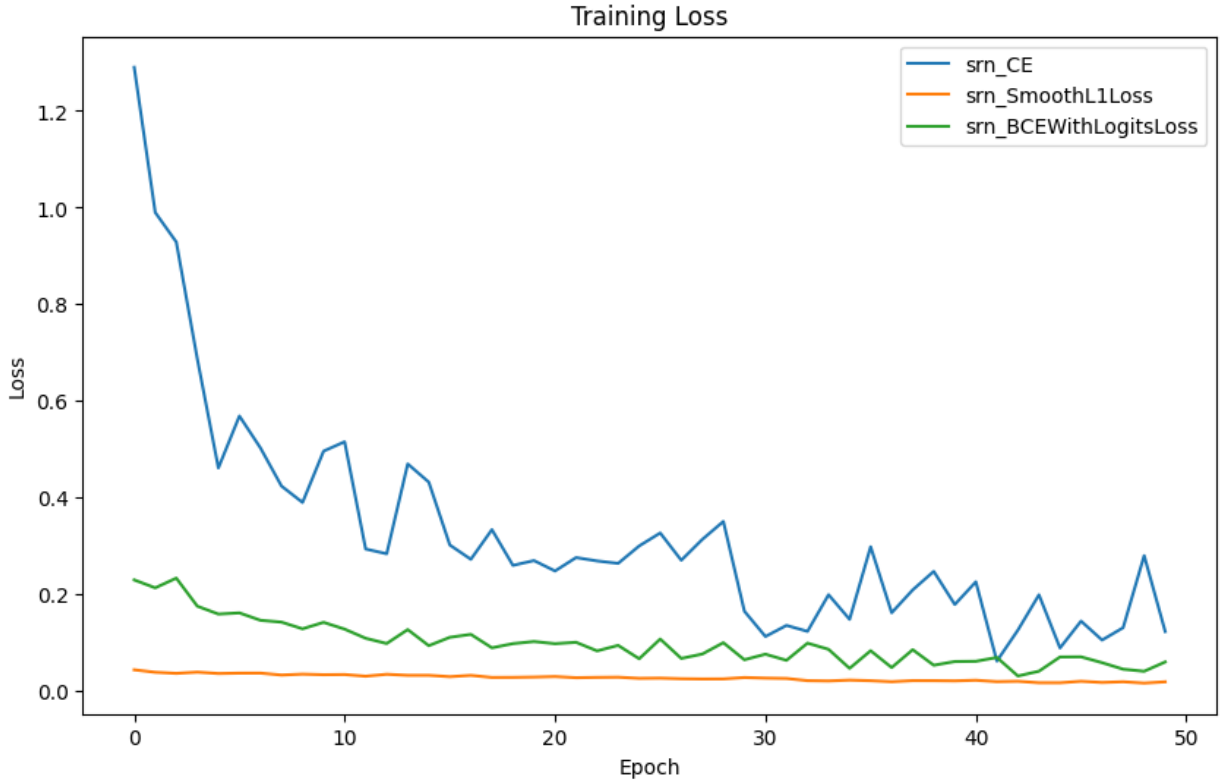
我们可以得出以下结论:

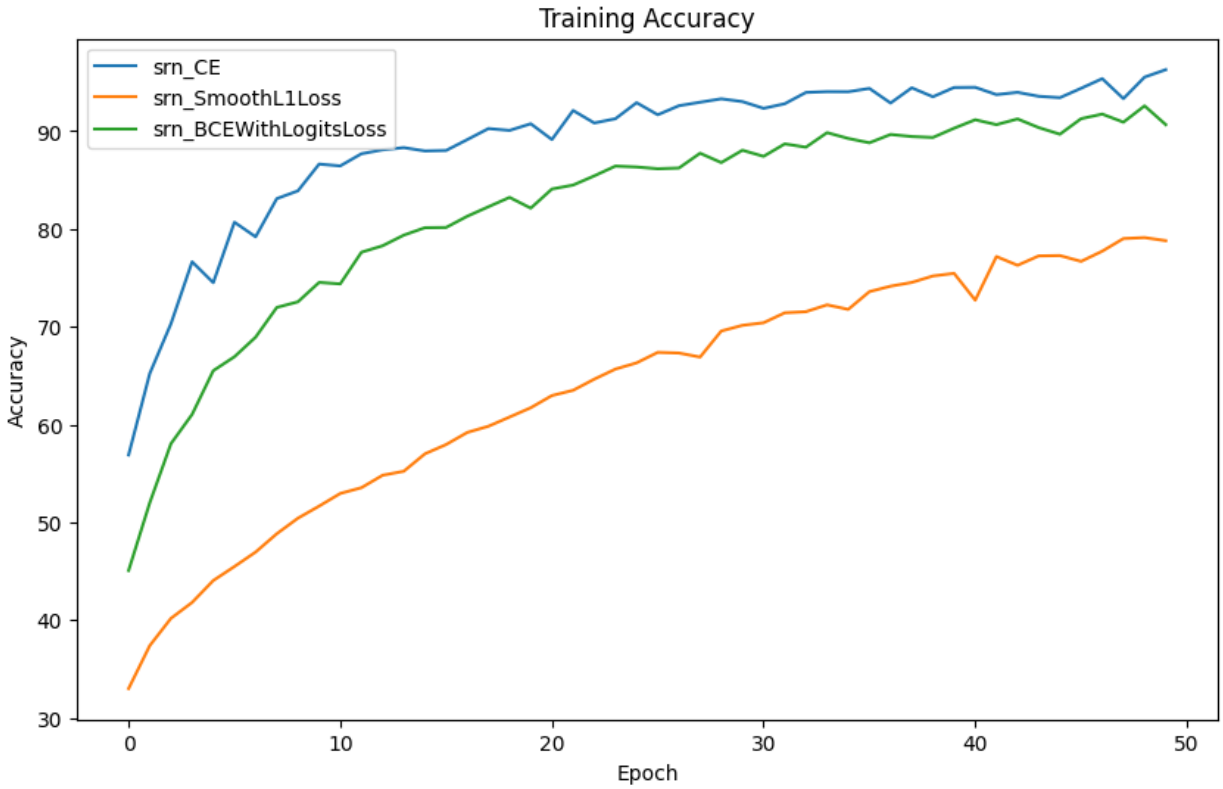
1. Mish和GELU激活函数在loss曲线和准确率曲线上表现较为出色,训练更加稳定,兼顾拟合和泛化。
2. ELU虽然收敛较慢,但最终效果不错,可能适合一些需要更强泛化能力的任务。
3. ReLU和RReLU相对而言训练效果较差,存在过拟合和不稳定的问题。

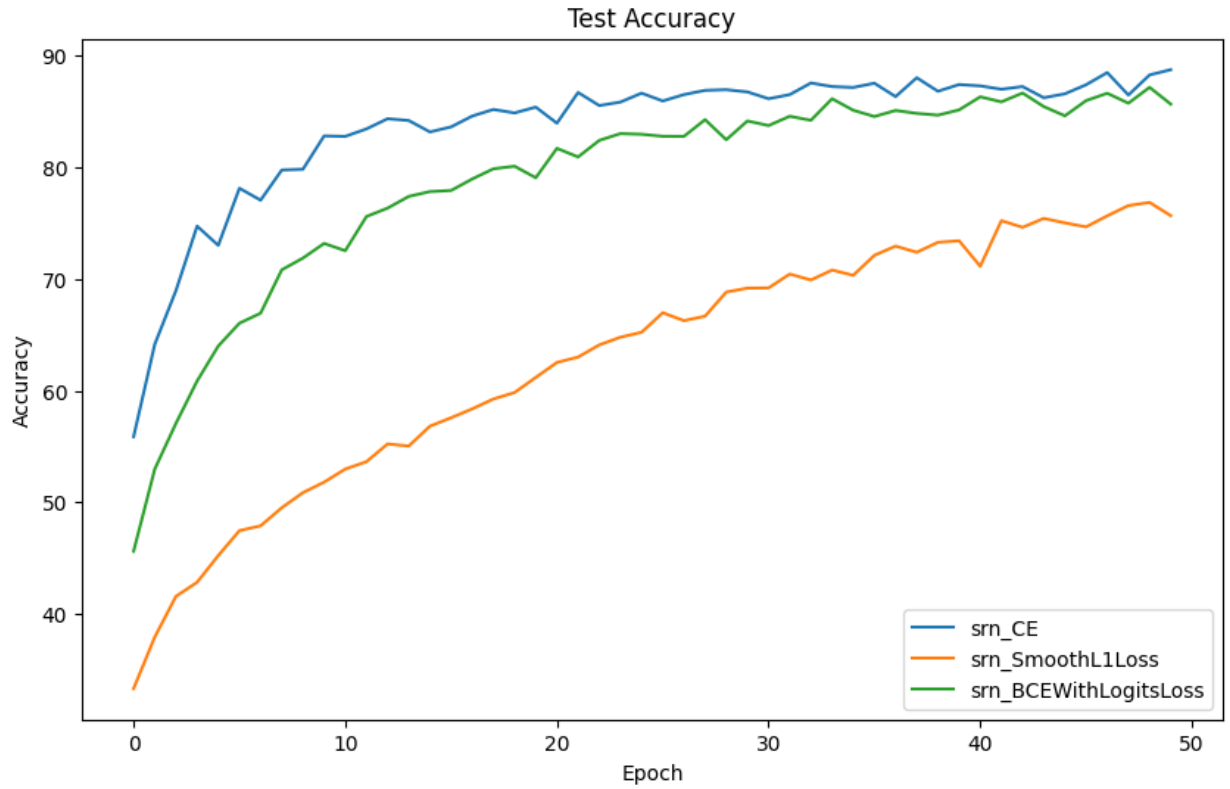
1.5 不同损失函数的训练效果比较

在该部分中，我们尝试多种损失函数，探究不同损失函数对模型性能的影响。这里我们依然选用simp_Res9作为研究对象，其他超参数不变，分别使用Cross Entropy、SmoothL1Loss、BCEWithLogitsloss三种损失函数进行实验探究。

下面是不同损失函数的训练效果的比较:


Training Loss
srn_CE
srn_SmoothL1Loss
srn_BCEWithLogitsLoss
Loss
Epoch
0.0
0.2
0.4
0.6
0.8
1.0
1.2
0
10
20
30
40
50


Training Accuracy
srn_CE
srn_SmoothL1Loss
srn_BCEWithLogitsLoss
Accuracy
Epoch
30
40
50
60
70
80
90
0
10
20
30
40
50



下面是最终的训练结果比较：

loss	train accuracy（%）	test accuracy（%）
CE	96.246	88.720
SmoothL1Loss	78.678	75.660
BCEWithLogitsLoss	90.704	85.640

基于训练数据，对三种损失函数的效果分析如下：

1. **Cross Entropy Loss**:
    - Loss曲线:通常会呈现较快的下降趋势,最终收敛到较低的损失值。这种损失函数专门针对分类任务设计,能够有效地优化模型的分类性能。
    - 准确率曲线:训练准确率和测试准确率通常都会快速提升,并最终达到较高的水平。Cross Entropy Loss能够有效地引导模型学习分类决策边界。
2. **Smooth L1 Loss**:
    - Loss曲线:下降趋势相对平缓,最终收敛的损失值通常高于Cross Entropy Loss。Smooth L1 Loss对异常值/outlier更加鲁棒,适用于回归问题。
    - 准确率曲线:训练准确率提升相对较慢,但测试准确率提升可能更快。这是因为Smooth L1 Loss更关注整体预测误差,而非单个样本的分类准确性。
3. **BCE with Logits Loss**:
    - Loss曲线:下降趋势与Cross Entropy Loss类似,但最终收敛的损失值通常较高。这种损失函数适用于二分类问题,能够直接输出概率值。

- 准确率曲线:训练准确率和测试准确率的提升速度介于Cross Entropy Loss和Smooth L1 Loss之间。它能平衡分类性能和概率输出的需求。

根据提供的最终准确率数据,我们可以进一步分析这三种损失函数在ResNet模型训练中的表现:

1. **Cross Entropy (CE) Loss**:
   - 训练准确率为96.246%,相当高,表明模型在训练集上拟合效果很好。
   - 测试准确率为88.720%,也处于较高水平,说明模型具有较强的泛化能力。
   - 训练和测试准确率差距约7.526个百分点,这种差距较小,表明模型没有明显的过拟合问题。
2. **Smooth L1 Loss**:
   - 训练准确率为78.678%,相对较低,可能存在一定程度的欠拟合。
   - 测试准确率为75.660%,也较低,说明模型的泛化性能较弱。
   - 训练和测试准确率差距约3.018个百分点,差距较小,但整体准确率偏低。
3. **BCE with Logits Loss**:
   - 训练准确率为90.704%,相对较高,但低于Cross Entropy Loss。
   - 测试准确率为85.640%,介于Cross Entropy Loss和Smooth L1 Loss之间。
   - 训练和测试准确率差距约5.064个百分点,表明模型有一定的过拟合倾向。

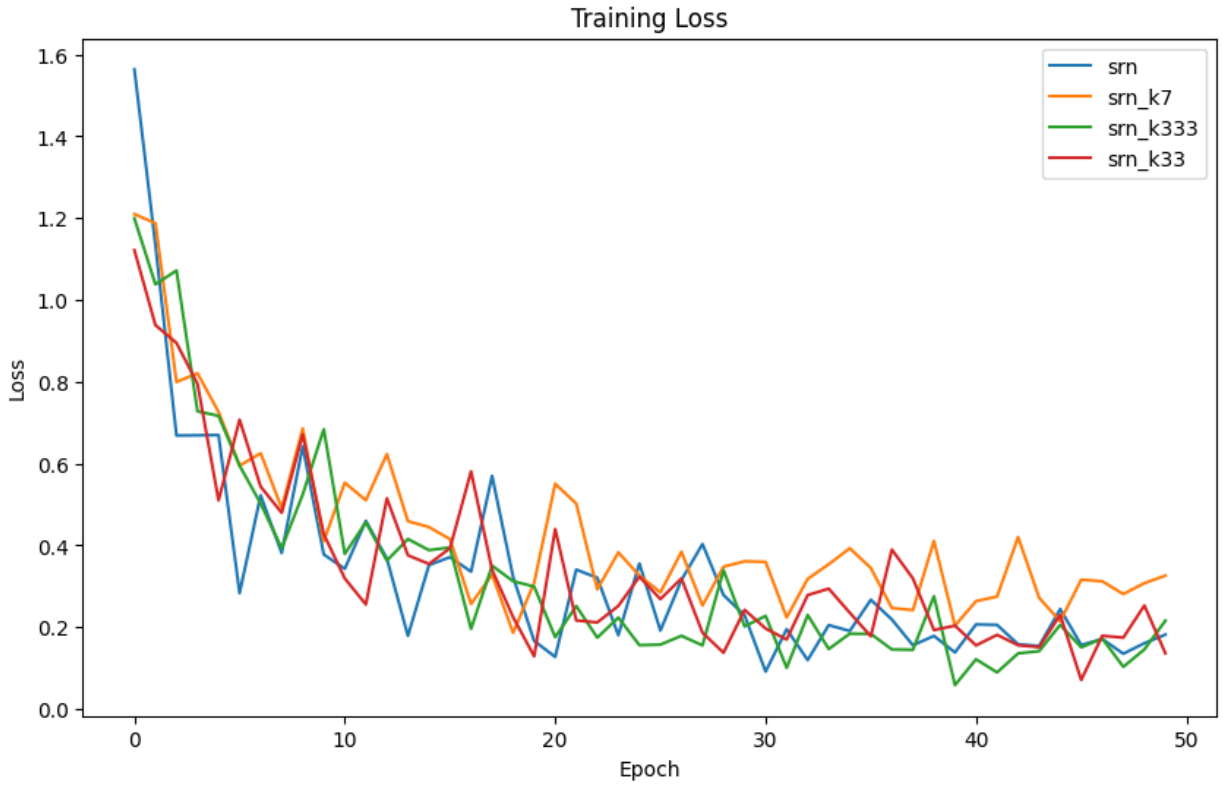
综合分析:

1. Cross Entropy Loss表现最佳,不仅在训练集上拟合效果良好,在测试集上的泛化性能也较强。这种损失函数更适合分类任务。
2. Smooth L1 Loss的整体表现较差,无论是训练还是测试准确率都较低。可能由于该损失函数更适用于回归问题,不太适合分类任务。
3. BCE with Logits Loss介于两者之间,既不如Cross Entropy Loss那么出色,也没有Smooth L1 Loss那么糟糕。这种损失函数在平衡分类性能和概率输出方面可能更有优势。

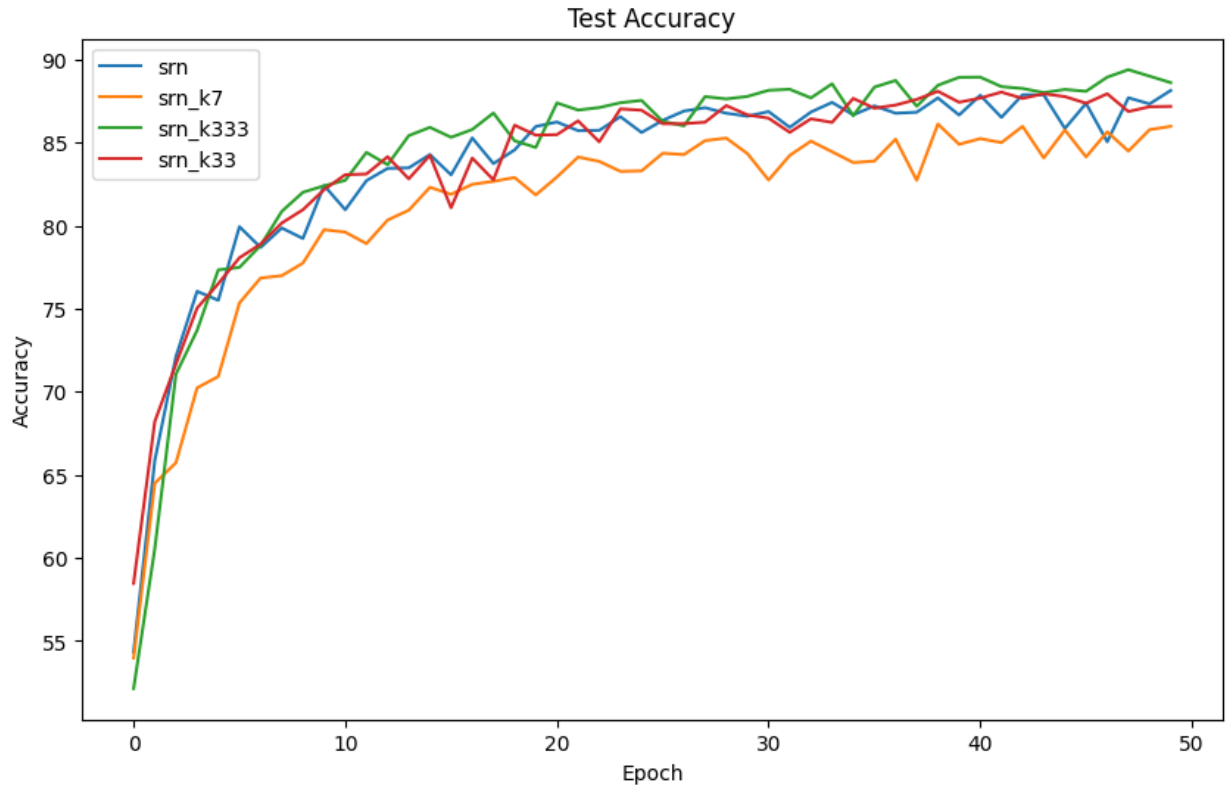
1.6 不同卷积核的模型性能比较

下面我们分别对第一卷积块的大小进行调整，比较探究卷积块尺寸对模型效果的影响。这里我们依然选用simp_Res9作为研究对象，其他超参数不变，分别使用【3x3】、【7x7】、【3x3，3x3，3x3】以及【3x3，3x3】的卷积组合进行实验。

下面是不同卷积核的模型训练效果的比较：


Training Loss
srn
srn_k7
srn_k333
srn_k33
Loss
Epoch
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1.6
0
10
20
30
40
50


Training Accuracy
srn
srn_k7
srn_k333
srn_k33
Accuracy
Epoch
60
70
80
90
0
10
20
30
40
50



下面是模型最终性能：

kernel size	train accuracy（%）	test accuracy（%）
【3x3】	94.872	88.140
【7x7】	93.548	85.980
【3x3，3x3，3x3】	85.516	88.610
【3x3，3x3】	94.148	87.180

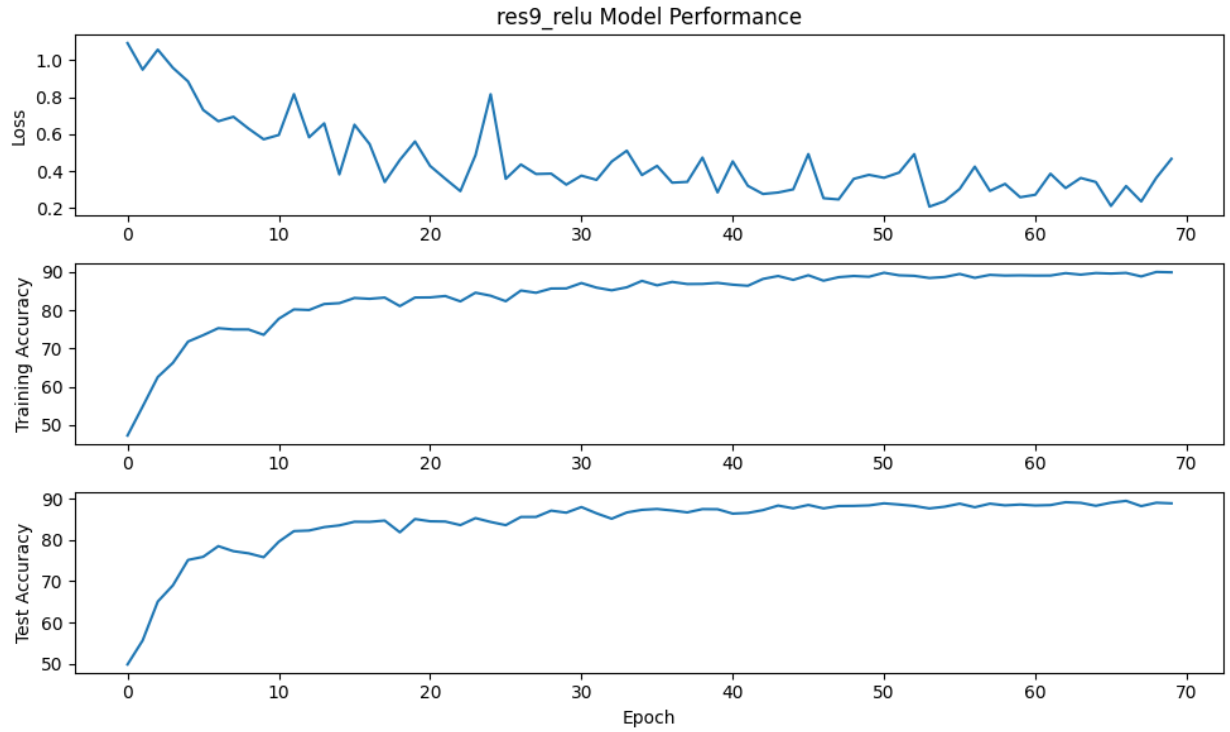
对此我们可以分析如下：

1. 单个3x3卷积核的表现最佳,既能有效学习局部特征,又能保持良好的泛化性能。这种基本卷积结构是ResNet的核心所在。
2. 7x7卷积核尽管感受野较大,但在ResNet中可能会导致特征提取能力降低,出现过拟合问题。
3. 多层3x3卷积能进一步增加网络深度,提升特征表达能力,但训练收敛可能较慢,需要更精心的优化。
4. 双层3x3卷积在保持强特征提取能力的同时,也增加了网络深度,是一种不错的折衷方案。

1.7 数据增强对模型效果的影响

在之前的实验中，我们对训练数据进行了随机截取、随机翻转和标准化等基础的数据增强。但模型的泛化能力依然较弱，在模型训练准确率稳步提高直到趋近于100%时，测试准确率往往于90%左右波动。故我们尝试对数据进行进一步的增强。这里我们使用random erasing方法，将图像的某一部分进行遮挡，以防止模型过度学习而发生过拟合。

下面是其他超参数不变，对simp_Res9使用增强后的数据训练70个epoch的效果：

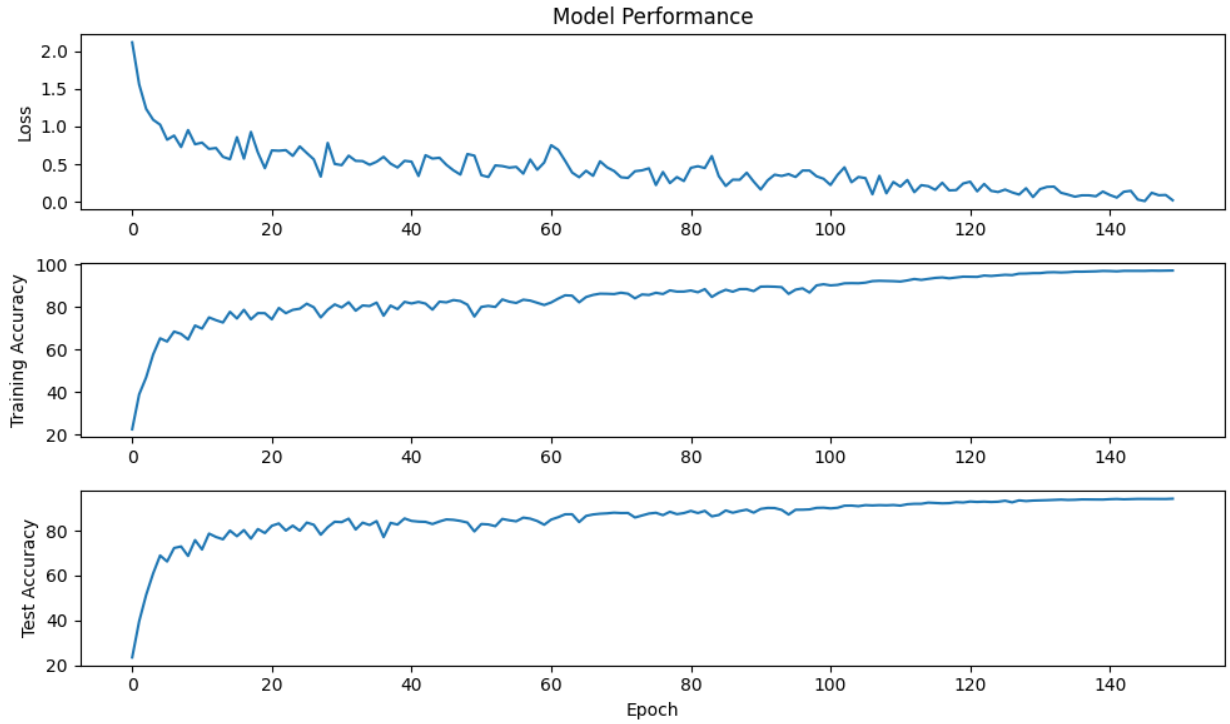


其在训练集上的最终准确率为 **89.888%**，在测试集上的最终准确率为**88.860%**。

可以看到没有显著的过拟合现象，数据增强在一定程度上起到了增强模型泛化能力的作用。

1.8 对模型最优性能的探索

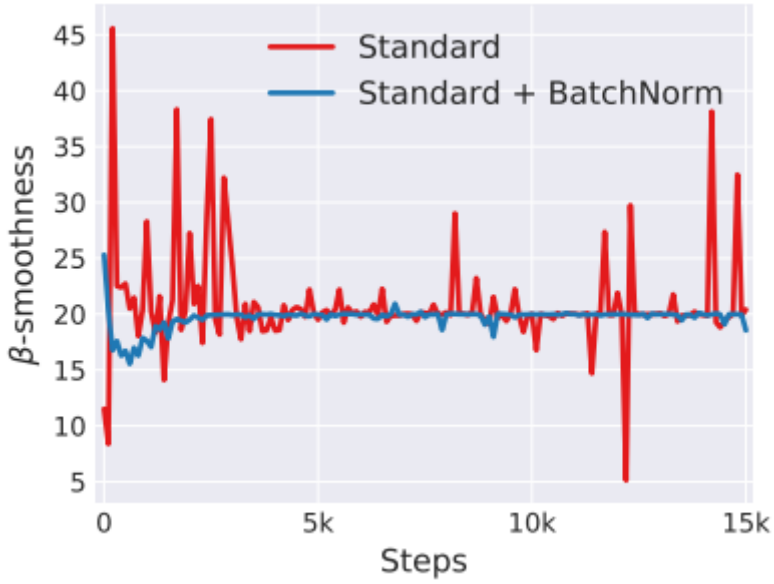
下面我们基于前面的探索结果，对simp_Res9模型使用多种手段进行优化，将第一层的卷积核改为【3x3，3x3，3x3】，选择激活函数为GELU，选择学习率策略为CosineAnnealingLR，并使用数据增强，训练150个epoch。模型的训练效果如下：



模型最终在训练集上的准确率为 **97.118%**，在测试集上的准确率为 **94.290%。**表现良好，且在训练时间和模型体积上相较于传统ResNet均有较大的优势。

2 Batch Normalization

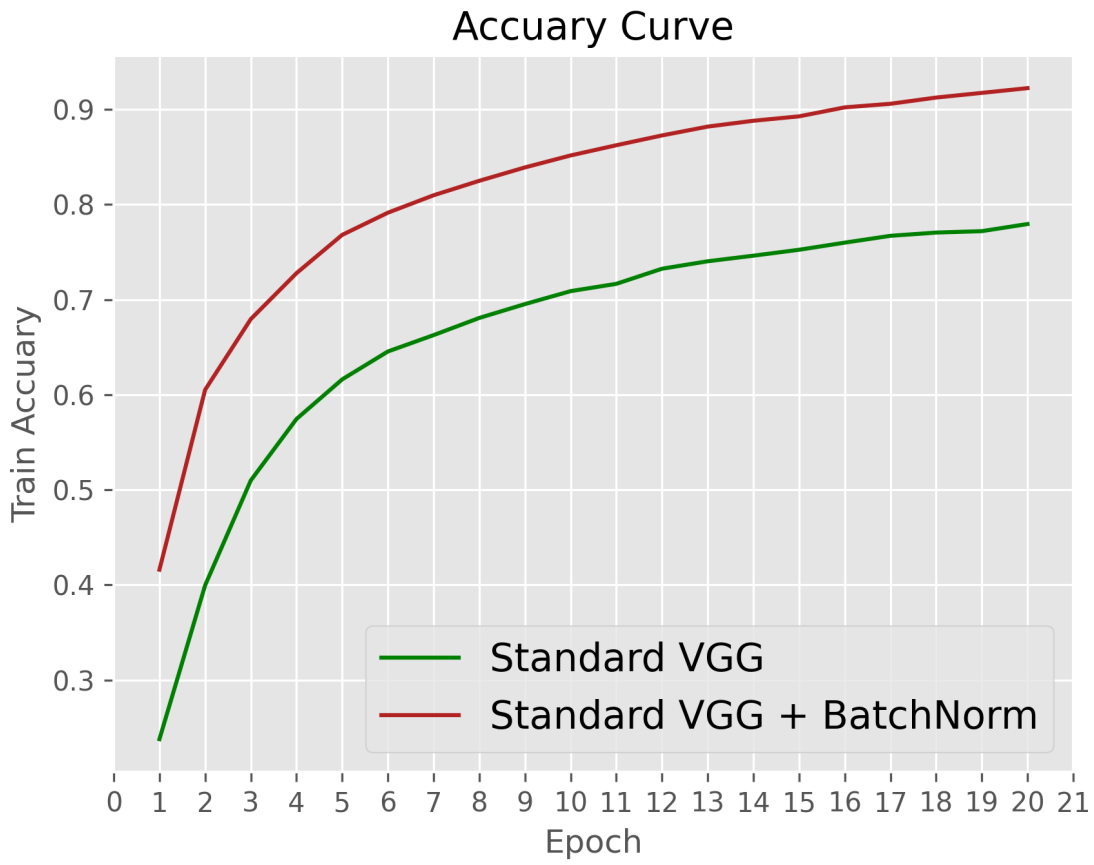
BN效果好是因为BN的存在会引入mini-batch内其他样本的信息，就会导致预测一个独立样本时，其他样本信息相当于正则项，使得loss曲面变得更加平滑，更容易找到最优解。**相当于一次独立样本预测可以看多个样本，学到的特征泛化性更强，更加general。**这个结论在之前的How Does Batch Normalization Help Optimization这篇文章中有明确提出和实验证明。



图中展示了用L-Lipschitz函数来衡量采用和不采用BN进行神经网络训练时两者的区别，可以看出未采用BN的训练过程中，L值波动幅度很大，而采用了BN后的训练过程L值相对比较稳定且值也比较小，尤其是在训练的初期，这个差别更明显。这证明了BN通过参数重整确实起到了平滑损失曲面及梯度的作用。

2.1 Batch Normalization对模型训练效果的影响

我们分别测试不同学习率下VGG模型分别使用与不使用BN时的最终性能差异（由测试集上的准确率衡量），实验结果如下：



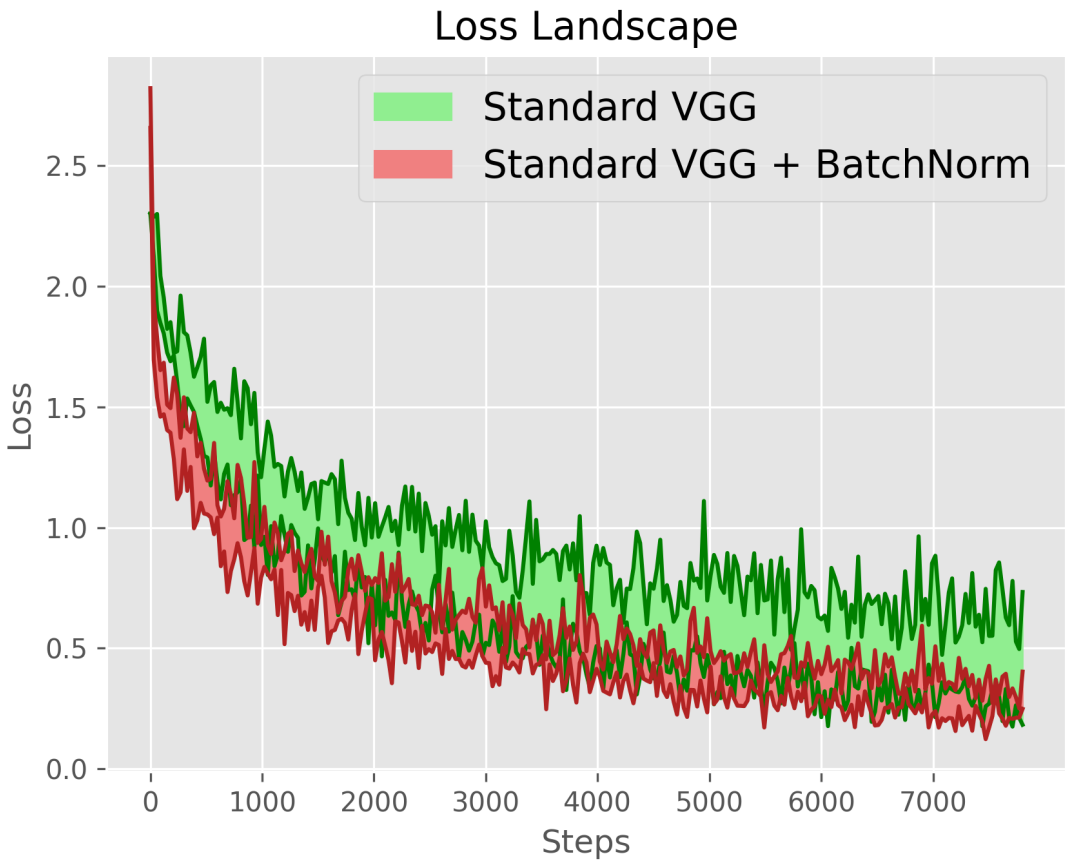
lr	with BN	without BN
2e-3	86.40%	75.98%
1e-4	81.77%	81.71%
5e-4	87.14%	83.90%

可以看到：

1. Batch Normalization能够显著提升模型在较大学习率下的泛化性能,抑制过拟合的发生。这是因为BN能够使训练过程更加稳定,减少对初始参数和学习率的敏感性。
2. 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。
3. 在较小学习率下,BN的效果相对较小,因为较小的学习率本身就能使模型训练更加稳定,BN带来的附加效果不太明显。
4. 在中等学习率下,BN仍然能够提升模型的性能,但效果不如在大学习率下那么显著。

2.2 Batch Normalization 对训练下loss landscape的影响

下面我们对模型每一步的损失分布进行可视化对比，画出模型的 Loss Landscape如下：



可以发现:

1. 损失函数分布均值的影响:
    - 使用BN的模型在各个迭代步骤中,损失函数的分布均值总是小于不使用BN的模型。
    - 这表明BN能够有效降低模型训练过程中的损失值,使得模型更快地收敛到最优解。
    - 这是因为BN能够对特征进行归一化处理,减小内部协变量偏移,从而使得损失函数下降更加平稳。
2. 损失函数分布波动的影响:
    - 使用BN的模型在各个迭代步骤中,损失函数的分布波动总是小于不使用BN的模型。
    - 这说明BN能够有效稳定模型的训练过程,减小损失函数值的波动幅度。
    - 稳定的损失函数分布意味着模型的训练过程更加平稳,更不容易陷入局部最优解或梯度爆炸/消失等问题。
3. 满足L-Lipschitz性质:
    - 前述两点表明,使用BN后模型的训练损失呈现更加稳定的性质。
    - 这符合L-Lipschitz连续性的要求,即损失函数对于输入的变化具有界限。
    - 满足L-Lipschitz性质意味着模型更容易优化,也更有利于理论分析和收敛性证明。