

Project 2

0 Introduction

本项目使用的数据集为 CIFAR-10。CIFAR-10 数据集是一个广泛应用于图像任务的数据集，由 60000 张 $32 \times 32 \times 3$ 的彩色图像组成，分别为 50000 张训练图像和 10000 张测试图像。数据集包含 10 个不同类别，每类有 6000 张图像。

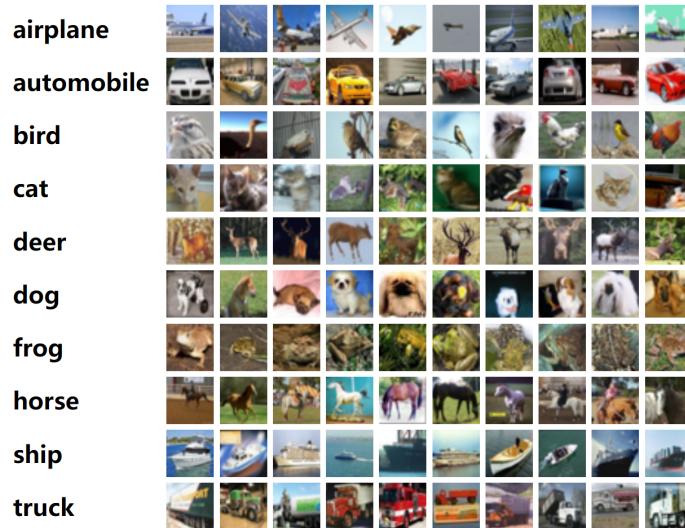


图 1: CIFAR-10 数据集

本项目主要做了以下任务：

1. 基于 ResNet18，研究损失函数、权重衰减、激活函数、优化器、学习率调整策略对于模型训练效果的影响，并在后续的训练任务中选择表现最优的一组策略。
2. 训练神经网络模型。使用现有的神经网络模型（LeNet, AlexNet, ResNet, ResNeXt, DenseNet, WideResNet, GENet）进行训练，对比分析不同模型的收敛速率、时间成本、训练效率和分类准确率。在学习率、优化器、损失函数、训练的 epoch 数量等基本超参数相同的情况下，表现最优秀的模型为 **GeResNeXt29_8x64d**，准确率达到 **96.88%**。

3. 基于 ResNet18，修改得到轻量化的网络 **ResNet9**，然后通过调整最大池化层顺序、标签平滑化、幽灵批标准化 (Ghost Batch Normalization, GBN) 等的 tricks 将 ResNet9 改进为 **FastResNet9**，模型的训练速度、收敛速度和训练效果都得到了较大提高。基础 ResNet9 训练 200 个 epoch 后的准确率为 93.10%，而 FastResNet9 训练 8 个 epoch 即可将其超越，训练 200 个 epoch 后准确率能够达到 **95.77%**，提升效果非常显著，甚至超过了标准 ResNet18。
4. 基于标准 VGG-A 网络，复现文献 *How does batch normalization help optimization?* [1] 中的三个实验，在实践中进一步理解 Batch Normalization 提升模型训练效果的原理。

需要注意的是，由于本项目时间非常紧张，训练模型花费的时间很长，所有模型我都只训练了一次（完整的），因此模型结果可能会受到随机性的影响，即本文结论中的最优模型并不意味着该模型在每次训练时都为最优。

1 Train a Network on CIFAR-10

1.1 Optimize the Network

在深度学习模型中，有 5 个策略/参数的选择对于神经网络来说非常重要，分别为：损失函数 (Loss Function)、权重衰减 (Weight Decay)、激活函数 (Activation Function)、优化器 (Optimizer)、学习率调整策略 (Learning Rate Scheduler)。

为了节省调参时间，我首先基于 ResNet18 这个经典网络模型，设定初始学习率为 0.01，以 CrossEntropyLoss、5e-4、SGD、ReduceLROnPlateau 训练 25 个 epoch 作为 baseline，对不同策略进行比较，然后选择最适用于长期训练 (max epoch=200) 的一组策略应用于 1.3 节的所有模型。

1.1.1 Loss Function

损失函数是用来衡量模型拟合程度好坏的函数，衡量的方式是比较网络输出和真实标签的差异。

选择 CrossEntropyLoss、BCEWithLogitsLoss、MSELoss、HuberLoss 和 MultiLabelSoftMarginLoss 这五种较常用的多分类问题损失函数，分别训练 25 个 epoch。

表 1: ResNet18 结果 (损失函数)

Loss Function	Train Accuracy	Test Accuracy
CrossEntropyLoss	94.95%	90.61%
BCEWithLogitsLoss	92.85%	84.83%
MSELoss	88.21%	84.81%
HuberLoss	83.99%	81.67%
MultiLabelSoftMarginLoss	92.86%	87.85%

由表1, 图2可知, CrossEntropyLoss 的训练、测试准确率都最高, 每个 epoch 的准确率基本都在其他损失函数的曲线上方, 故使用 CrossEntropyLoss 作为 CIFAR-10 分类的损失函数。

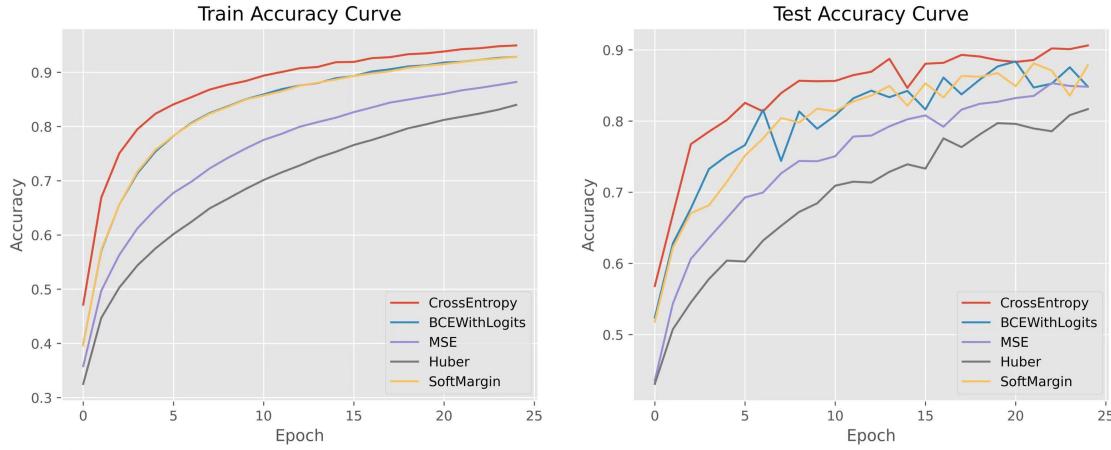


图 2: 学习曲线 (损失函数)

1.1.2 Weight Decay

权重衰减, 即 L2 正则化, 是一种最简单而又最常见的参数范数惩罚。这个正则化策略通过向损失函数添加一个正则项 $\Omega(\theta) = \lambda * \frac{1}{2} \|w\|_2^2$, 使权重更加接近原点, 避免过拟合。其中, λ 就是本节需要调整的 weight decay 参数, 它反映了对权重的惩罚程度。

以 1e-4, 5e-4, 1e-3, 5e-3, 1e-2 作为 weight decay 的值, 分别训练 25 个 epoch。

表 2: ResNet18 结果 (权重衰减)

Weight Decay	Train Accuracy	Test Accuracy
1e-4	95.60%	89.05%
5e-4	94.95%	90.61%
1e-3	94.10%	88.94%
5e-3	91.91%	88.89%
1e-2	88.54%	80.22%

由表2、图3可知, 1e-4 的训练准确率最高, 而 5e-4 的测试准确率最高, 这说明 1e-4 对于权重的惩罚力度较小, 出现了过拟合的问题, 比 5e-4 大的 weight decay 减弱了模型的拟合效果, 从以 1e-2, 5e-3 为 weight decay 的波动率非常大的学习曲线中可以看出。因此选择 5e-4 作为模型 weight decay 是最合适的选择。

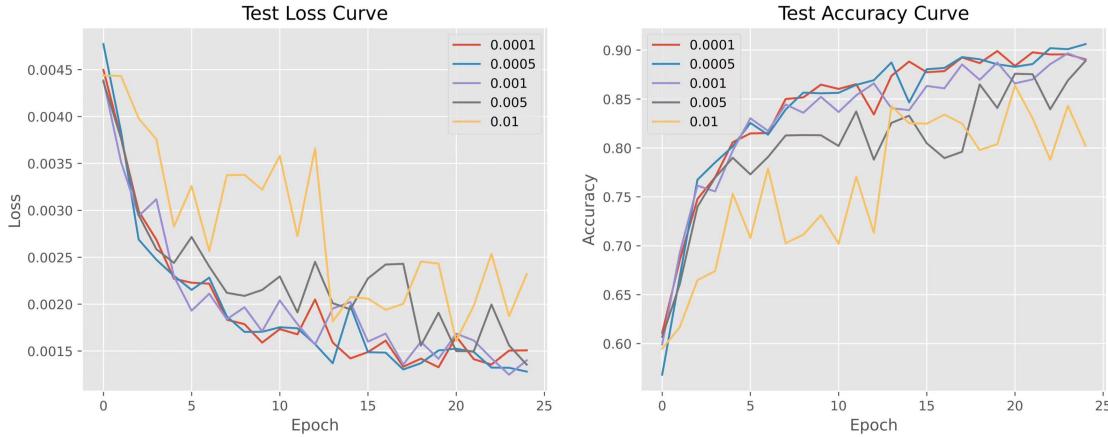


图 3: 学习曲线 (权重衰减)

1.1.3 Activation Function

激活函数的作用是将线性网络转化为非线性网络，增加模型的复杂度。

选择 RELU、Mish、LeakyRELU、ELU、GELU 作为 ResNet18 的激活函数，分别训练 25 个 epoch。

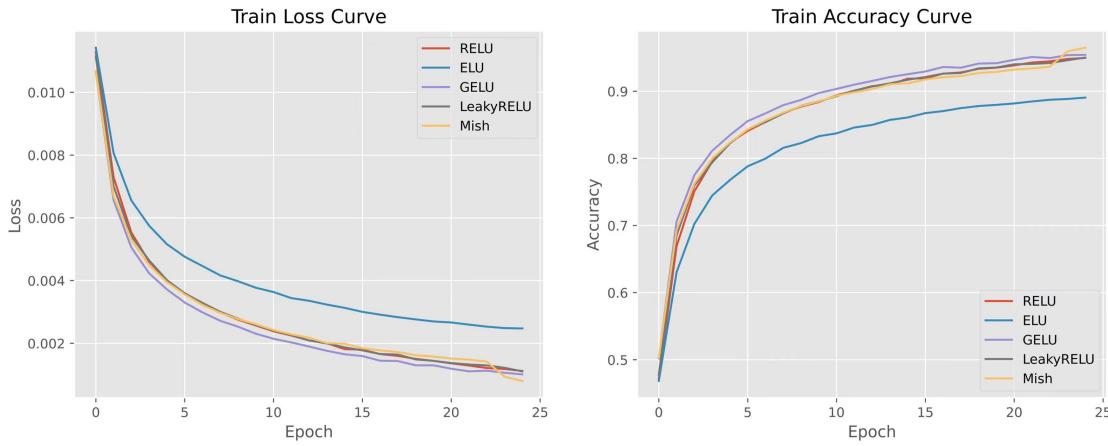


图 4: 学习曲线 (激活函数)

结合表3、图4可知，虽然 Mish 激活函数的准确率指标最高，但是它是在最后几个 epoch 发生突变反超的，不够稳定，所以暂不考虑 Mish。其次是 RELU 和 GELU，图像显示，GELU 在训练集上的损失曲线几乎始终低于及其他激活函数，且 GELU 函数在 0 点具备光滑的特性，说明 GELU 在求解和模型泛化性方面表现得很好。

不过因为标准 ResNet18 模型的激活函数为 RELU，该设置的背后蕴含了无数学者的智慧，所以在 1.3 节我就不自作主张更改 ResNet18 的激活函数了，我会在 1.4 节构建 ResNet9 时更换激活函数。这一小节仅仅是为了对比不同激活函数给模型带来的不同影响。

表 3: ResNet18 结果 (激活函数)

Activation Function	Train Accuracy	Test Accuracy
RELU	94.95%	90.61%
Mish	96.48%	91.69%
LeakyRELU	94.60%	89.34%
ELU	88.72%	86.93%
GELU	94.93%	90.36%

1.1.4 Optimizer

优化器的作用是在训练阶段更新模型中可学习的参数。选择 SGD、Adam、Adamdelta、Adagrad、Adamax 作为优化器，分别训练 25 个 epoch。

表 4: ResNet18 结果 (优化器)

Optimizer	Train Accuracy	Test Accuracy
SGD	94.95%	90.61%
Adam	72.61%	71.26%
Adamdelta	86.89%	82.81%
Adagrad	94.65%	88.62%
Adamax	87.34%	85.36%

由表4、图5可知，SGD 优化器的表现最好，Adam 优化器的表现很差，这可能是因为初始学习率 0.01 对于 Adam 优化器来说较低，模型的学习速度较慢。

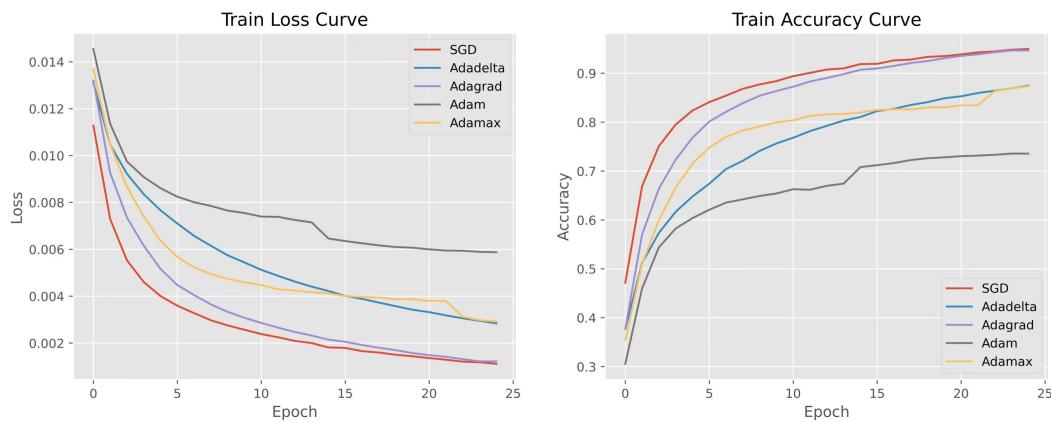


图 5: 学习曲线 (优化器)

1.1.5 Learning Rate Schedule

在训练过程中，学习率控制了每一步参数更新的步长，合适的学习率可以使代价函数以合适的速度收敛到最小值，让学习率随着训练的 epoch 数量增大而变化能够帮助模型更好地学习。

这里选择 ReduceLROnPlateau (Loss 自适应衰减)、StepLR (分段衰减)、LinearLR (线性衰减)、ExponentialLR (指数衰减)、CosineAnnealingLR (余弦退火衰减) 作为学习率调整策略，分别训练 25 个 epoch，学习率变化曲线见图6。

表 5: ResNet18 结果 (学习率调整策略)

LR Scheduler	Train Accuracy	Test Accuracy
ReduceLROnPlateauLR	94.95%	90.61%
StepLR	97.33%	90.74%
LinearLR	96.07%	90.59%
ExponentialLR	97.66%	90.60%
CosineAnnealingLR	98.44%	91.91%

由表5、图6可知，余弦退火衰减的策略表现最好，其他策略相差不大。如果训练 200 个 epoch，由于余弦退火衰减具有周期性，那么训练后期很可能会出现学习率过大的问题，而 Loss 自适应衰减策略则能够在长期的训练中保持较好的表现，因此在 1.3 节我决定使用 ReduceLROnPlateau 作为模型的学习率调整策略。

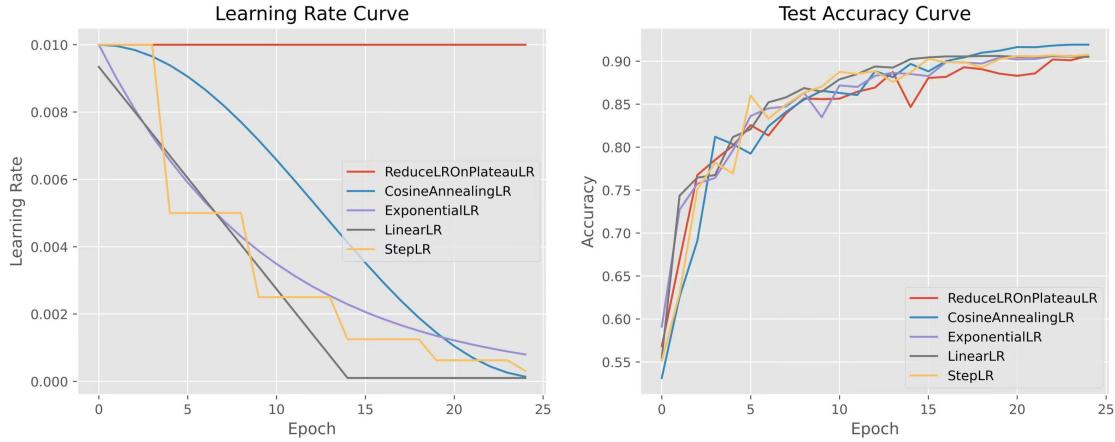


图 6: 学习率变化曲线 + 准确率曲线

1.2 Brief Introduction to Network Architecture

1.2 节、1.3 节使用的网络结构有：LeNet，AlexNet，ResNet，ResNeXt，DenseNet，WideResNet，GENet。我基于 ResNet18 改进的 ResNet9 网络将在 1.4 节介绍。

1.2.1 LeNet

LeNet5 是最早的卷积神经网络之一，是最近大量神经网络架构的起点。LeNet5 利用卷积、参数共享、池化等操作提取特征，节省了大量的计算成本，最后使用全连接神经网络进行分类识别 [2]。

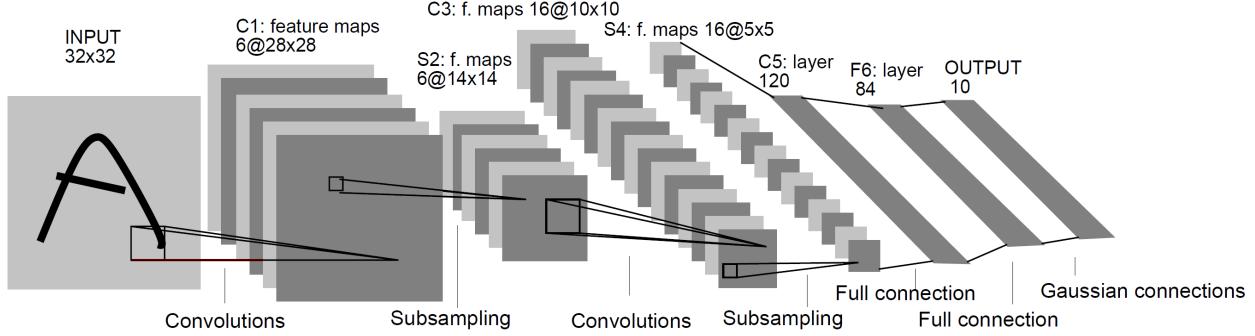


图 7: LeNet5 网络结构示意图

LeNet5 由 7 层 CNN（不包含输入层）组成，如图7所示，输入的原始图像大小为 32×32 ，卷积层为 C_i ，子采样层（池化层）为 S_i ，全连接层为 F_i 。

1.2.2 AlexNet

AlexNet 在 LeNet 的基础上加深了网络的结构，能够学习更丰富更高维的图像特征。

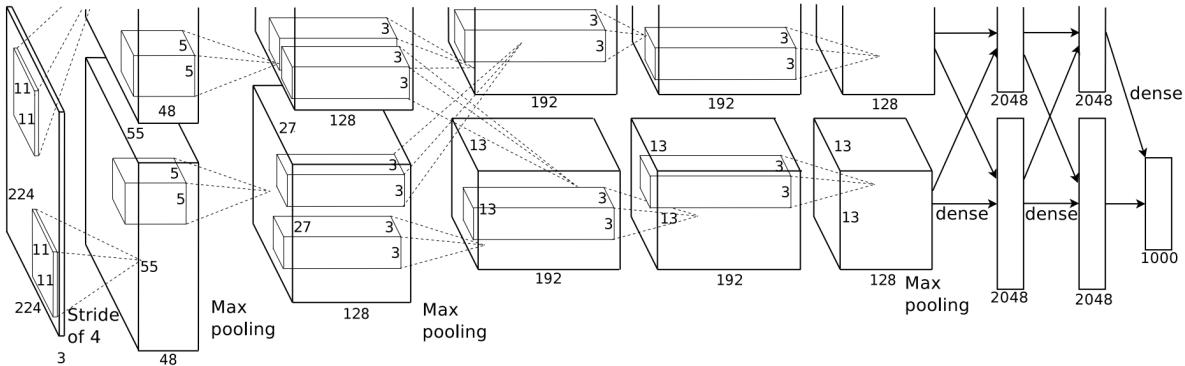


图 8: AlexNet 网络结构示意图

AlexNet 由 5 个卷积层（Conv + RELU + LRN + Pooling）和 3 个全连接层（InnerProduct + RELU + Dropout）组成，如图8所示。其特点在于使用层叠的卷积层（卷积层 + 卷积层 + 池化层）来提取图像特征，使用 RELU 替换 Sigmoid、Tanh 等激活函数，使用 Dropout 和数据增强来抑制过拟合，且允许多 GPU 并行运算 [3]。

1.2.3 ResNet

卷积神经网络叠加多层，往往会导致梯度消失、梯度爆炸和退化等问题发生，残差网络 ResNet 的提出正是为了解决这两个问题 [4]。ResNet 通过数据预处理和批归一化 (Batch Normalization, BN) 来避免梯度消失和梯度爆炸，通过引入残差 (Residual) 结构来缓解模型的退化，即人为地让神经网络某些层跳过下一层神经元的连接，隔层相连，弱化每层之间的强联系，如图9所示。

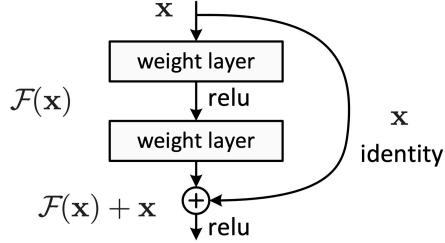


图 9: ResNet 残差结构示意图

1.2.4 ResNeXt

受到 Inception 的 split-transform-merge 结构的启发，ResNeXt 将 ResNet 的三层卷积 block 从原来的多 filter 卷积核拆分为多个结构相同的少 filter 卷积核，如图10所示。相同的拓扑结构能够简化网络设计，使超参数相对较少，让模型在不明显增加参数量级的情况下提升准确率 [5]。

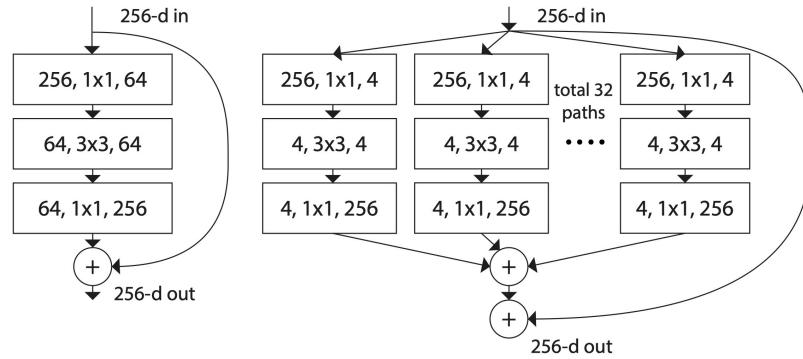


图 10: ResNeXt 网络结构示意图

1.2.5 DenseNet

传统卷积网络在前向过程中每层只有一个连接，ResNet 增加了残差结构，主要目的是为了增加不同层之间的信息流动。DenseNet 基于此提出了一种新的连接模式——Dense connections。传统 L 层的网络仅有 L 个连接， L 层 DenseNet 中则有 $L(L + 1)/2$ 个连接，这加强了特征传播，实现特征复用，相比于 ResNet 大幅减少了参数量，同时还能保证模型精度。

标准 DenseNet 中，Block 结构为 BN + RELU + 3×3 Conv。DenseNet-BC 在每个 Block 前增加了 Bottleneck 结构 (BN + RELU + 1×1 Conv)，且引入了一个超参数 θ ，将 m 个 feature map 映射为 θm 个 (θ 通常取 0.5)，以此来减少参数量 [6]。

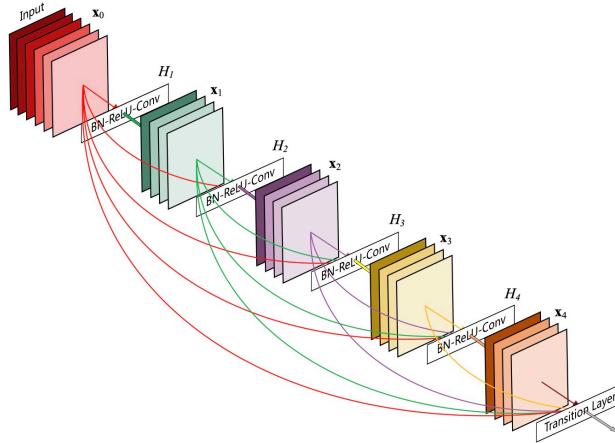


图 11: DenseNet 网络结构示意图

1.2.6 WideResNet

ResNet 每提高一个百分点的模型精度，就需要增加一倍左右的层数，随着深度增加，ResNet 存在特征复用减少的问题，导致网络训练效率很低。

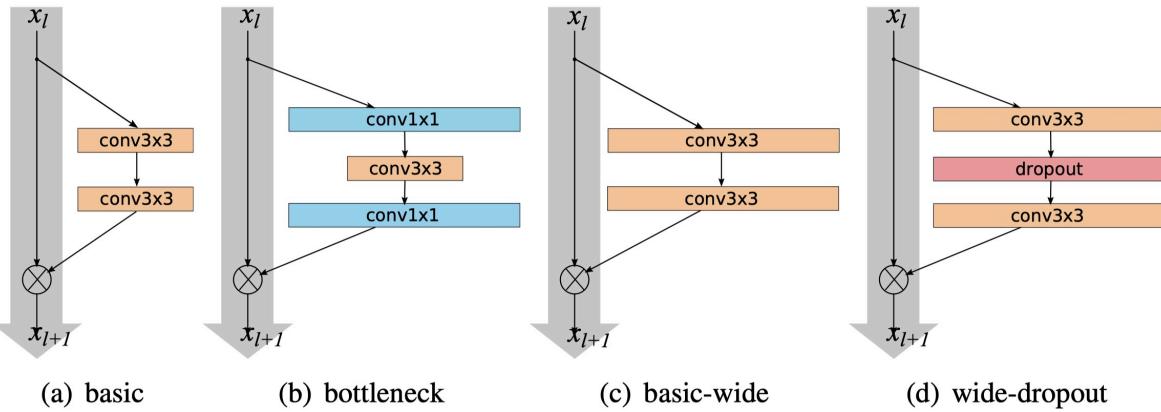


图 12: WideResNet 残差结构示意图

宽残差网络 WideResNet 通过减少网络深度、增加网络宽度来提高模型性能，相同参数时 WideResNet 的训练速度高于 ResNet。不同于 ResNet 的 BN 正则化，WideResNet 使用 Dropout 进行正则化 [7]。WideResNet 带 dropout 的残差块如图 12d 所示。

1.2.7 GENet

GENet 从上下文的角度出发，提出了 Gather-Excite 操作（图13），利用空间注意力来更好的挖掘特征之间的上下文信息。其中，Gather 操作是指使用不同大小的滤波器从局部的空间位置上聚集特征，Excite 操作则将 Gather 操作后收集到的上下文信息恢复至原始尺寸。这是一种类似于编解码即 Encoder-Decoder 的模型，以增加较小的参数量和计算量的代价来提升网络的性能 [8]。

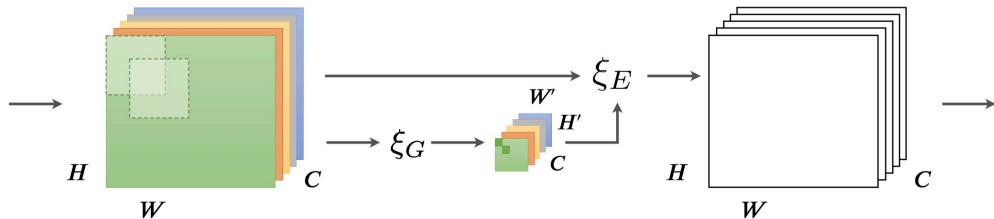


图 13: GENet 的 Gather-Excite 操作示意图

1.3 Training Results

所有模型通用的参数配置如表6所示。

表 6: 模型参数设置

Batch Size	Initial LR	LR Scheduler	Lambda	Optimizer	Loss	Epoch
128	0.1	ReduceLROnPlateau	5e-4	SGD	CrossEntropy	200

```
[ ]: criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.1, momentum=0.9, weight_decay=5e-4)
lr_scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(
    optimizer, mode='min', factor=0.5, patience=5, cooldown=5, min_lr=1e-6)
```

对训练集和测试集使用相同的均值和标准差进行标准化处理，应用的数据增强技术包括：随机裁剪、以 0.5 为概率的随机水平翻转。数据增强和权重衰减都是防止模型过拟合的手段，能够增强模型的泛化性能。

```
[ ]: transform_train = T.Compose(
    [
        T.RandomCrop(32, padding=4),
        T.RandomHorizontalFlip(),
        T.ToTensor(),
        T.Normalize((0.4914, 0.4822, 0.4465), (0.2470, 0.2435, 0.2616)),
    ]
)
```

标准 ResNet18 模型的第一层使用 7×7 的卷积核，而 CIFAR-10 的图片大小为 32×32 ，比较小，如果用这么大的卷积核会损失许多信息，对模型精度造成负面影响。因此，我将第一层的卷积核修改为 3×3 的大小，padding 和 stride 参数都修改为 1。同理，我还删除了第一层卷积层之后的 3×3 池化层。

表 7: 模型结果 (at 25 epochs and 200 epochs)

Model	Nparams	Test Acc (25)	Test Acc (200)	Time(s/epoch)
LeNet5	0.06M	59.87%	76.32%	2.76
AlexNet	2.47M	69.23%	77.82%	3.05
ResNet18	11.17M	85.84%	95.29%	21.96
ResNet34	21.28M	83.74%	95.58%	51.59
ResNet50	23.52M	79.00%	94.66%	79.24
ResNeXt29_2x32d	2.30M	86.46%	94.02%	40.15
ResNeXt29_2x64d	9.13M	82.35%	94.45%	85.47
ResNeXt29_32x4d	4.77M	81.86%	95.03%	78.08
DenseNet100_bc	0.77M	89.02%	95.20%	74.25
DenseNet121	6.96M	89.92%	95.35%	89.91
WideResNet16x8	10.97M	84.63%	95.18%	42.83
WideResNet28x10	36.49M	84.10%	96.63%	158.49
GeResNeXt29_8x64d	36.33M	85.50%	96.88%	439.67

模型结果见表7，其中 GeResNeXt29_8x64d 的表现最优秀，准确率高达 96.88%，训练曲线见图14。该模型的缺点是参数量较大，运算速度很慢，训练 200 个 epoch 需要花费超过 1 天的时间。

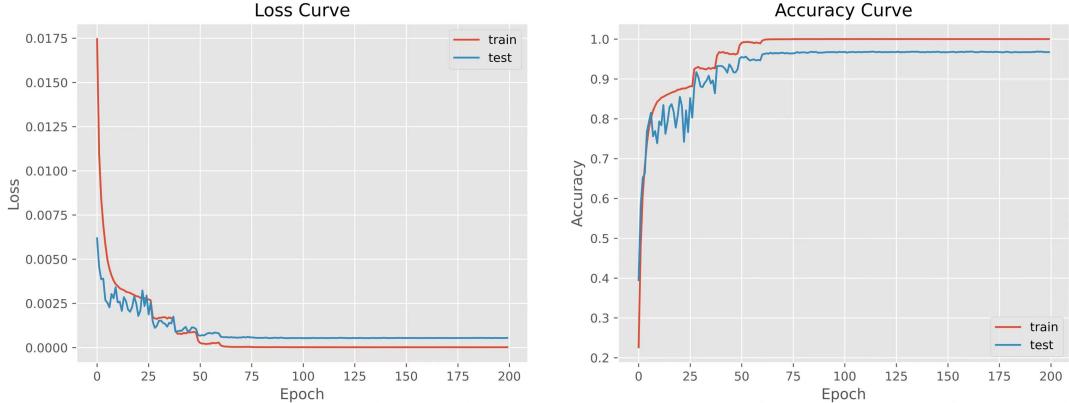


图 14: GeResNeXt29_8x64d 学习曲线

相较而言，性价比较高的模型为 ResNet34 和 WideResNet28x10，前者牺牲 1.3% 的准确率，单 epoch 训练速度是 GeResNeXt29_8x64d 的九分之一；后者牺牲 0.25% 的准确率，单 epoch 训练速度是 GeResNeXt29_8x64d 的三分之一。效果最差的两个模型是 AlexNet 和 Lenet5，因为这两个网络出现得比较早，结构比较简单，CIFAR-10 数据集对二者而言复杂度太高了，用它们很难拟合出准确率高的模型。

从 ResNet 和 ResNeXt 的模型结果可以得出，参数量越大并不意味着模型训练效果越好，ResNet34 和 ResNeXt29_32x4d 的参数量并不是最多的，但训练效果却最好。这可能是因为 CIFAR-10 数据集的复杂度不够高，相同模式下略复杂的网络结构就能满足训练需求，也不会产生过拟合的问题。

以 25 个 epoch 为界作为训练初期，发现收敛速度最快的模型是 DenseNet121，这说明收敛速度快的模型训练到最后准确率不一定最高，这和学习率的设置存在一定关联。

1.4 Construction of FastResNet9

在 1.2 节和 1.3 节，我使用了现有的模型在 CIFAR-10 上训练，发现训练速度和模型精度不可兼得。这一小节将基于 ResNet18 模型进行改进，在不损失太多模型精度的条件下尽可能地提高训练速度。

1.4.1 ResNet9

ResNet18 的层数还是比较多，为了简化模型结构，我减少了 2 个残差块，构建了轻量化的 ResNet9，结构如图15所示。我在尝试了不同激活函数进行训练后，将 ResNet18 的 RELU 激活函数更改为了于零点较平滑的 GELU 函数，一定程度上提高了训练效果。

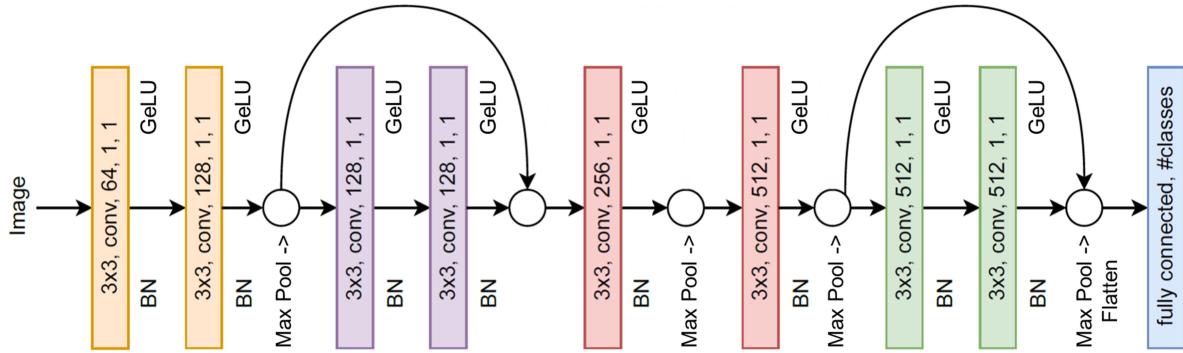


图 15: ResNet9 网络结构示意图

模型初始学习率为 0.01，其他参数设置见表6，训练结果见表8。相比于 ResNet18，ResNet9 单 epoch 训练时间下降了约 9 秒，速度有了很明显的提升，但是分类准确率下降了 2.22%，效果并不理想。

表 8: 模型结果 (ResNet9)

Model	Nparams	Test Accuracy	Time(s/epoch)
ResNet9	6.57M	93.07%	12.38
ResNet18	11.17M	95.29%	21.96

1.4.2 Tricks in FastResNet9

基于上一节的 ResNet9 模型，我又做了以下改进：

1. 调整最大池化层 (MaxPool2d) 的顺序

一般而言，一个 block 内的顺序应该为：Conv→BN→Activation→Pooling（图16a）。池化层的作用是对特征图进行降维，减少参数量，如果将 MaxPool 层向前移动，在 BN 和激活函数前先进行 pooling，就能节省计算时间（图16b）。经过实践，该操作使训练速度大幅提高，收敛速率略微降低，但这并不会降低模型的学习能力，因为可以通过增加几个 epoch 来保证模型的收敛性和精度。

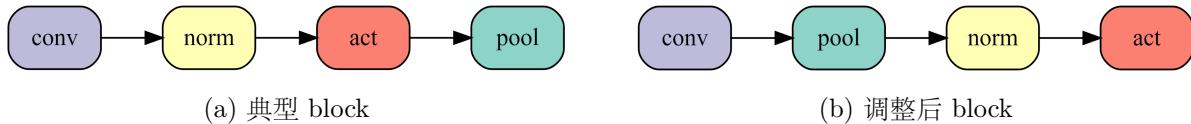


图 16: Conv-Pool-Act Block

2. 标签平滑 Label Smoothing

标签平滑损失在分类问题中往往用于提升模型的训练速度和泛化能力，该损失的计算方式见图17。它能够缓解模型过于武断的问题，让输出分布更加稳定，同时训练时间进一步减少。

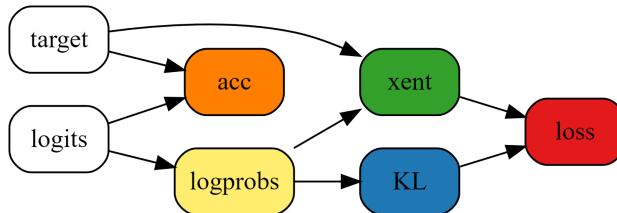


图 17: Label Smoothing

3. 幽灵批归一化 Ghost Batch Normalization

批归一化 (Batch Normalization) 在 batch size 较大的时候效果会降低，batch size 为 32 的时候效果最好，这是因为过大的批量会降低正则化的效果，而过小的批量又会带来很大的噪声干扰。Ghost BatchNorm 的做法就是将大的 batch 分成多个 batch size 为 32 的子集，然后将 BatchNorm 分别应用于每个子集上（图18）。这个操作能够提高模型的准确率。

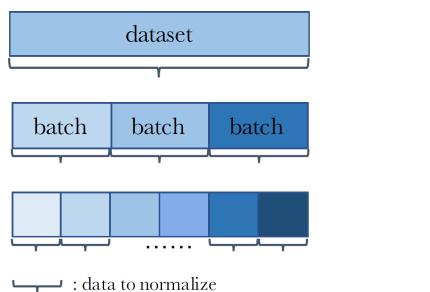


图 18: Ghost Batch Normalization

4. 测试数据增强 Test Data Augmentation

在对训练集的数据增强部分，本项目进行了随机裁剪和随机水平翻转，因此我联想到对测试集也可以做数据增强操作，即将原始测试集和水平翻转后测试集的输出取平均，作为最终的预测结果。这个操作会增加一点训练时间，但是能够提高模型的准确率。

```
[ ]: raw_inputs = valid_data[idx: idx + args.batch_size]
flipped_inputs = torch.flip(raw_inputs, [-1])
y_hat1 = net_valid(raw_inputs).detach()
y_hat2 = net_valid(flipped_inputs).detach()
y_hat = torch.mean(torch.stack([y_hat1, y_hat2], dim=0), dim=0)
```

1.4.3 Results of FastResNet9

训练结果见表9。FastResNet9 的各项结果都非常好，训练 10 个 epoch 后就达到了 94.03% 的准确率，200 个 epoch 后更是进一步提高到了 95.77%，且单 epoch 训练时间缩短到 7.91 秒，模型性能非常令人满意。

表 9: 模型结果 (FastResNet9)

Model	Nparams	Test Acc (10 epochs)	Test Acc (200 epochs)	Time(s/epoch)
ResNet9	6.57M	83.31%	93.07%	12.38
FastResNet9	6.57M	94.03%	95.77%	7.91
ResNet18	11.17M	86.16%	95.29%	21.96

学习曲线在训练初期非常陡峭（图19），说明 FastResNet9 收敛速度非常快；曲线在训练中后期则非常平稳，损失和准确率都在缓慢下降或上升，模型效果很理想。

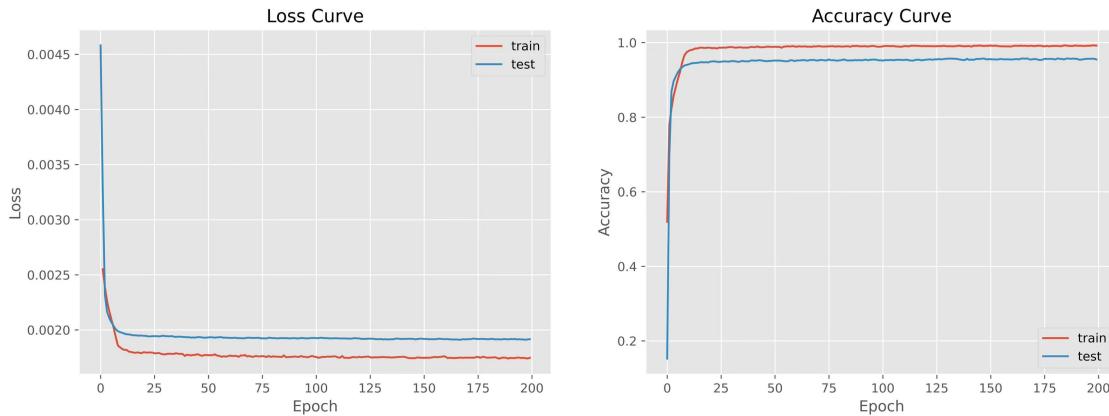


图 19: FastResNet9 学习曲线

1.5 Network Interpretation

这一节中，我只对 FastResNet9、ResNet9 和 ResNet18 模型的前两层卷积层进行可视化，因为卷积层越深，特征图大小越小，人眼很难从像素块里分辨特征。案例图（图20）是一张大小为 32×32 的可爱小猫照片。

从特征图中看，第一层卷积层中（图21），FastResNet9 学习到了比较有解释力的特征，猫的轮廓或边界被刻画得非常清晰；第二层（图22）相较于第一层，特征模糊一些，但是仍然很容易观察到两只尖尖的猫耳，说明该模型的解释性还是不错的。

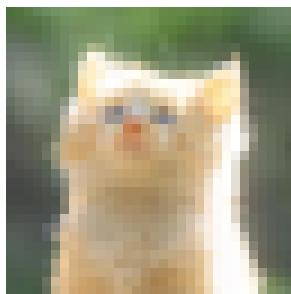


图 20: 可爱小猫照片

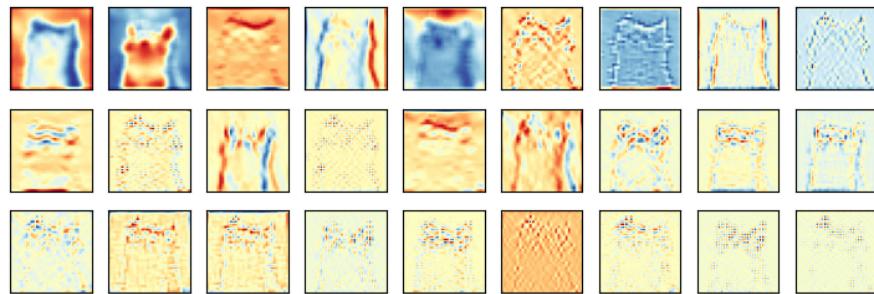


图 21: FastResNet9 第一层特征图

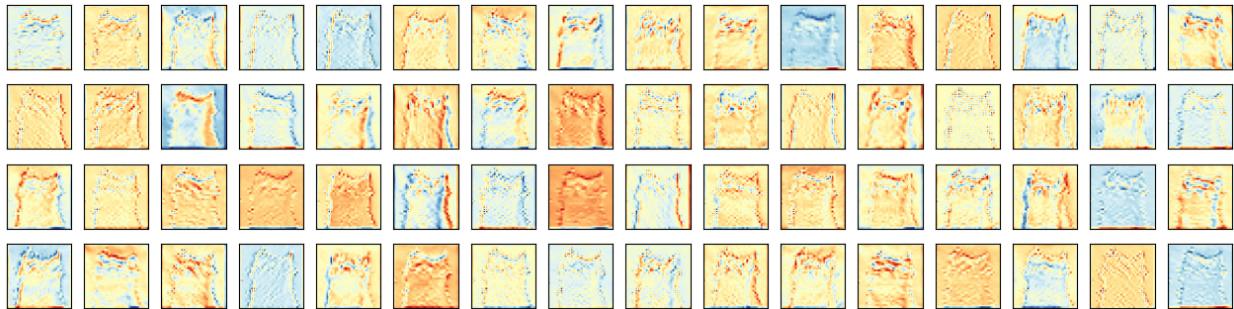


图 22: FastResNet9 第二层特征图

然而，一旦将 FastResNet9 与 ResNet9 和 ResNet18 学习到的特征相比较，就会发现后二者学习到的特征更加具有解释性（图23-图26）：第一层卷积，模型不但非常清晰地刻画了小猫的轮廓，还展现出了光影（高光）与小猫头部的曲面；第二层卷积的特征图就没有那么清晰了，但也比 FastResNet9 的第二层特征要强。

这里就出现了一个问题，明明 1.4 节中结果显示 FastResNet9 的准确率比其他两个模型都要高，为什么在特征学习上却比不过其他两个模型呢？我的猜想是，FastResNet9 提取出的特征更加精简，几乎没有冗余，模型性能自然就高；而且该模型的优秀表现也不一定全部来源于特征提取，还可能由 1.4.2 节中提到的训练技巧导致。ResNet9 和 ResNet18 提取的特征中有一部分是重复的，增加了计算耗时。

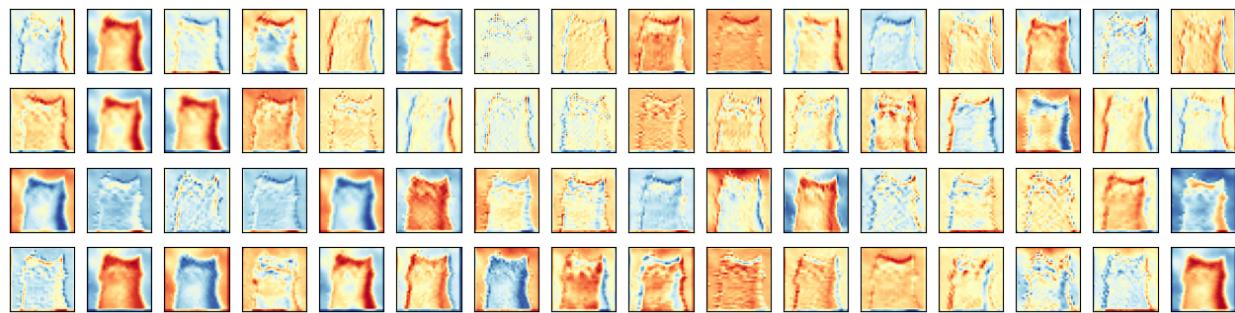


图 23: ResNet9 第一层特征图

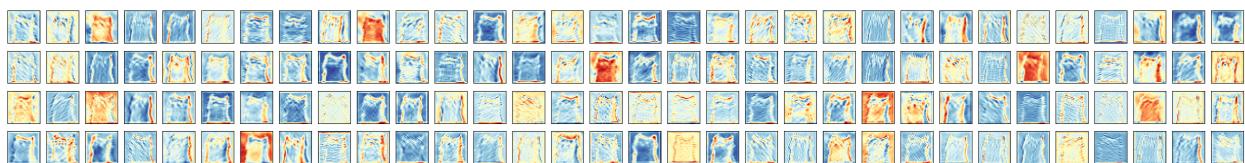


图 24: ResNet9 第二层特征图

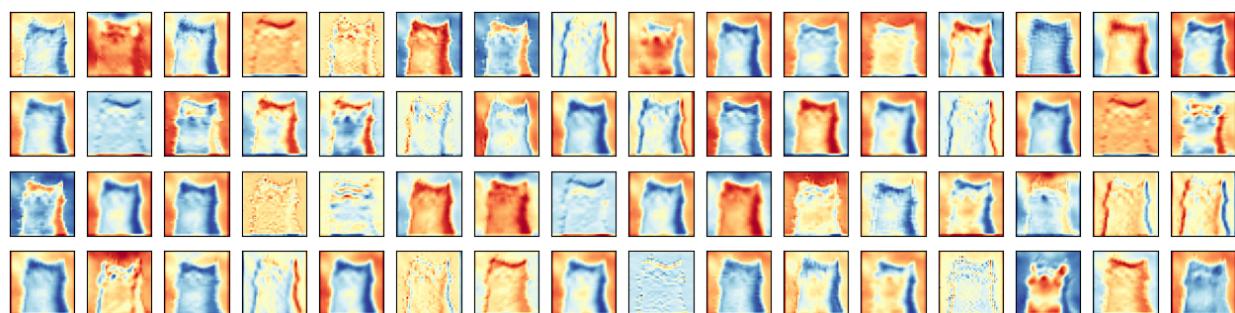


图 25: ResNet18 第一层特征图

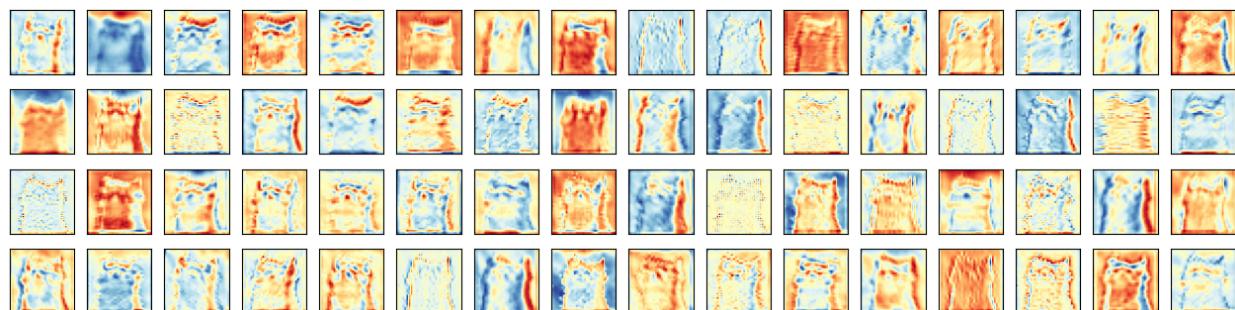


图 26: ResNet18 第二层特征图

2 Batch Normalization

Batch Normalization (BatchNorm, BN) 通过在神经网络中添加额外的层，控制层输入的均值和方差，使每一层神经网络的输入保持相同分布。在实际应用中，BatchNorm 能加速模型收敛，降低模型对学习率的敏感性，大大提高模型训练效果。最早提出 BN 的学者认为 BN 的优点在于能够解决 Interval Covariate Shift (ICS) 问题，可是实践中并无直接证据表明 BN 的优秀效果与 ICS 有关。

在 *How Does Batch Normalization Help Optimization?* [1] 中，作者提出 BN 对缓解 ICS 问题的作用非常微弱，其真实作用在于让优化过程的 landscape 更加平滑，使梯度更新更加稳定，加速模型收敛。加入 BN 层后，梯度的 Lipschitz 性质得到改善，即：

$$|f(x_1) - f(x_2)| \leq L\|x_1 - x_2\|, \quad \forall x_1 \text{ and } x_2 \quad (1)$$

这说明模型输出的变化量始终被限制在常系数范围内，增强了梯度的稳定性和预测性，使得基于梯度下降的深度学习训练效果更好。

在这一小节，本项目将基于 VGG-A 模型，复现文章中对 optimization landscape 进行的 3 个实验（图27）。由于实验目的仅仅是为了对比添加 BN 层前后模型的平稳性，我并没有在模型中加入学习率衰减、L2 正则化、Dropout 等策略。

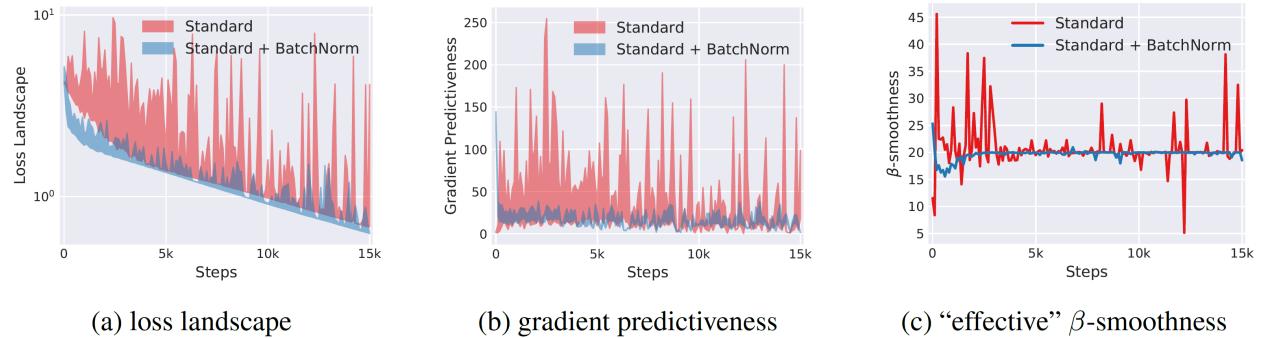


图 27: 论文原文中的 3 个实验

2.1 Impact of BatchNorm on Accuracy

在复现实验之前，我首先查看了标准 VGG-A 模型 with or without BatchNorm 在模型训练速率和效果上的区别。为了控制实验效果，节省计算时间，这里使用最基本的 SGD 优化器，训练 20 个 epoch，模型参数设置如下表所示。

表 10: VGG-A 模型超参数 (2.1 节)

Optimizer	Loss	Epoch	Batch Size	Learning Rate
SGD	CrossEntropy	20	128	0.1

结果显示（图28），加入 BN 层之后，模型在训练集、测试集上的准确率提高更快，最终效果也更好（标准 VGG 准确率为 79.5%，VGG+BN 准确率为 82.8%），说明 BN 能够加速模型收敛，且在泛化性上优于不加 BN 层的 VGG 网络。

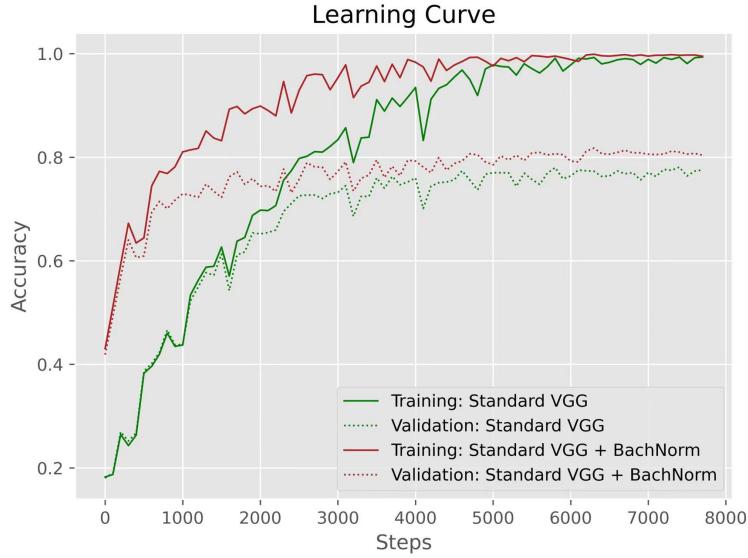


图 28: Standard VGG vs VGG+BN

2.2 How does BN help optimization?

本项目在复现论文实验时，没有使用指导文件中给的学习率 [1e-3, 2e-3, 1e-4, 5e-4]，是因为我在实验中发现这组学习率无法让模型在 20 个 epoch 内收敛，甚至不添加 BN 层的标准 VGG 网络在训练初期的 400 个 step 里完全没能得到有效优化（图29）。另外，图29中显示 VGG+BN 的波动性比标准 VGG 更大，与论文中的结论相悖，说明这组学习率太小，从而导致模型收敛速率太慢，无法从图中提取出有效规律，实验效果很不理想。

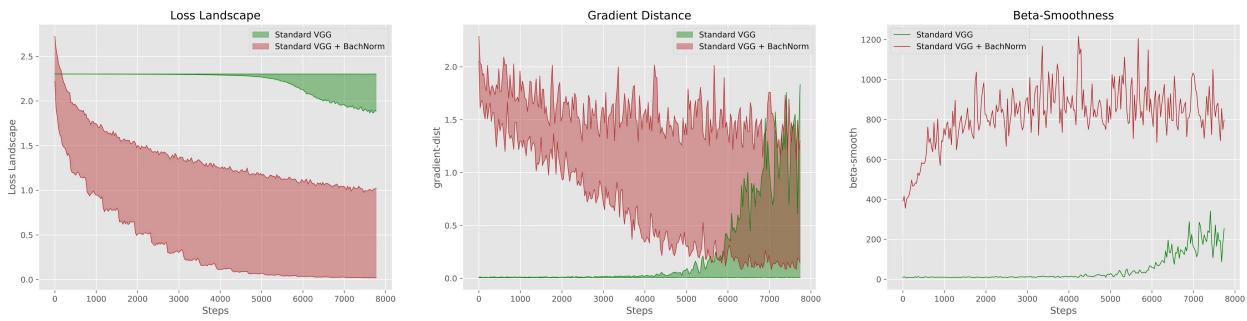


图 29: 学习率过小的 landscapes

因此，我重新选择了一组较大的学习率 [0.05, 0.075, 0.1, 0.125, 0.15]，以此避免模型无法快速收敛的问题。同时这一组学习率也不会因为过大而引起梯度爆炸，能够满足本项目复现实验的要求。VGG 模型的所有超参数如下表所示。

其中, gap=30 表示每 30 个 step 计算一次 landscape, 一方面是为了节省计算时间, 另一方面是为让图像更美观。

表 11: VGG-A 模型超参数 (2.2 节)

Optimizer	Loss	Epoch	Batch Size	Learning Rate	Gap
SGD	CrossEntropy	20	128	0.05, 0.075, 0.1, 0.125, 0.15	30

2.2.1 Loss Landscape

第一个实验, 我对于训练过程中的每 30 个 step 计算其损失的均值。

结果显示 (图30), 未添加 BN 层的网络 loss landscape 波动范围更大, 且均值的绝对值也更大, 这些差异在训练初期非常明显 (因为在训练后期模型已经接近收敛)。

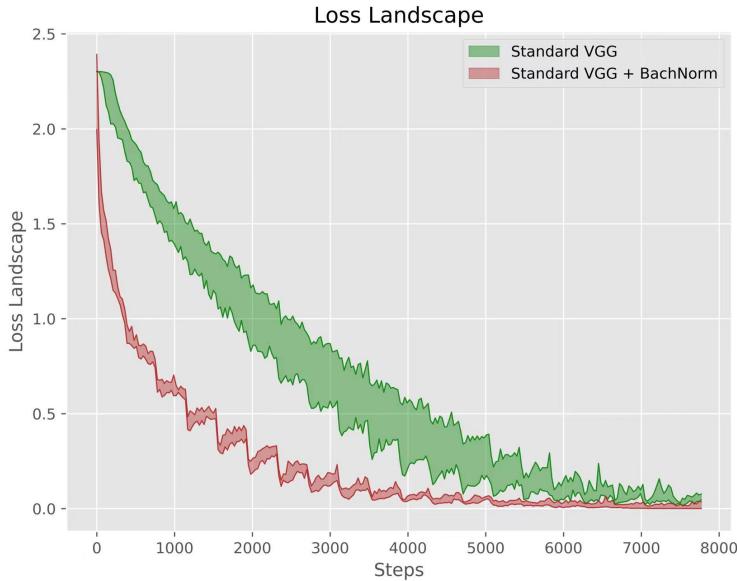


图 30: Loss Landscape

论文中作者对此分析指出, 由于梯度更新的 Lipschitz 性质, 模型训练受到学习率的影响减小。这意味着 BatchNorm 能够使训练在以下两个方面变得更加高效:

1. 减少调参的负担。模型的收敛性对学习率的敏感度降低, 使模型收敛的学习率范围增大, 让任何基于梯度的模型能够采取更大的学习率, 而不会遭遇 loss landscape 的突然变化 (如梯度消失和梯度爆炸), 从而减少花费在调整超参数上的时间。
2. 梯度更新更加充分。在给定合适学习率的情况下, 梯度的 Lipschitz 性质使模型能够充分地利用该学习率进行更新, 提高收敛速度。

有趣的是，BatchNorm 的平滑效应并不是独一无二的（论文原文 3.3 节）。事实上，使用 $l_p - norm$ ($p = 1, 2, \dots, \infty$) 进行标准化都能达到很好的效果，在深度线性网络中， $l_1 - norm$ 的性能甚至比 BatchNorm 都好。然而我们知道， $l_p - norm$ 归一化方法并不能保证层输入的分布符合类高斯分布，甚至会让数据产生严重的偏置，因此可以得到结论：BatchNorm 并不是通过归一化的方式来提升训练效果的。

2.2.2 Gradient Distance

第二个实验，我使用梯度更新前后之差的 $l_2 - norm$ 作为 gradient distance landscape，反映梯度的预测水平（同样是每 30 个 step 计算一次）。

$$\text{grad_dist}^{(t+1)} = \|\text{grad}^{(t+1)} - \text{grad}^{(t)}\|_2, \quad t = 1, 2, \dots \quad (2)$$

结果显示（图31），未添加 BN 层的 VGG 网络梯度预测误差波动范围更大。图31与论文结果的不同之处在于，VGG+BN 模型在训练初期的梯度预测误差均值比标准 VGG 更大，波动范围也较大，在训练后期 VGG+BN 的梯度预测误差则明显变得非常平稳，且均值低于标准 VGG。

这一定程度上说明了 BN 加速模型收敛的机制，即在训练初期，梯度的波动幅度较大有利于模型跳出局部最优，探索到通向全局最优的路径，故 BN 对于梯度平稳程度的限制也较小；而在训练后期，模型逐渐收敛，梯度更新的稳定性成为了该阶段的重点，故 BN 带来的平滑效应越来越大。

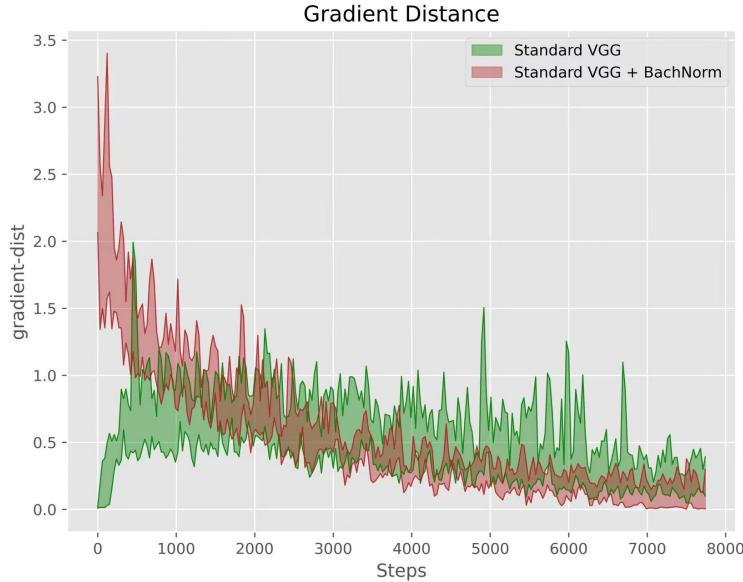


图 31: Gradient Distance Landscape

2.2.3 Effective Beta-Smoothness

第三个实验需要对比 VGG-A 模型添加 BN 层与否情况的 effective β -smoothness， β 越小越平滑。

一个函数 $f(x)$ 是 β -smooth 的当且仅当其梯度是 β -Lipschitz 的，即：

$$\|\nabla f(x_1) - \nabla f(x_2)\| \leq \beta \|x_1 - x_2\| \quad (3)$$

神经网络的 SGD 优化求解中，对权重 w 从 t 到 $t+1$ 的更新有：

$$w_{t+1} = w_t - lr * \nabla f(w_t) \quad (4)$$

将上式代入 β -Lipschitz 公式中，有：

$$\|\nabla f(w_{t+1}) - \nabla f(w_t)\| \leq \beta \|w_{t+1} - w_t\| = lr * \beta \|\nabla f(w_t)\| \quad (5)$$

解得：

$$\beta_t = \max \frac{\|\nabla f(w_{t+1}) - \nabla f(w_t)\|}{\|w_{t+1} - w_t\|} = \max \frac{\|\nabla f(w_{t+1}) - \nabla f(w_t)\|}{lr * \|\nabla f(w_t)\|} \quad (6)$$

结果显示（图32），训练初期，含有 BN 层的模型的 β 比标准 VGG 的 β 更高，而之后的大部分时间中，含有 BN 层的模型的 β 都更小，验证了 BN 的平滑效应。这里前后两阶段平滑性的不同也符合 2.2.2 节中我对于 BN 加速模型收敛机制的猜想。

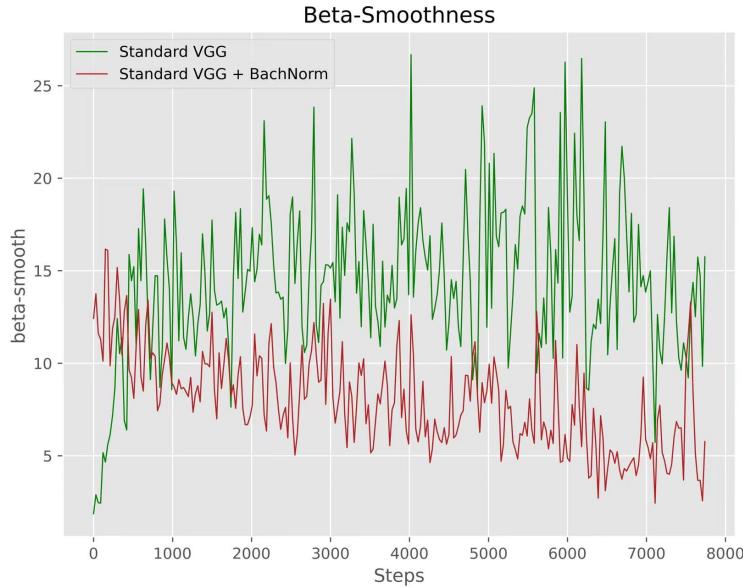


图 32: Effective Beta-Smoothness

参考文献

- [1] Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. How does batch normalization help optimization?, 2018.
- [2] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. 1998.
- [3] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. 2012.
- [4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. 2016.
- [5] Saining Xie, Ross Girshick, Piotr Dollar, Z. Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. 2017.
- [6] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Weinberger. Densely connected convolutional networks. 2017.
- [7] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks, 2017.
- [8] Jie Hu, Li Shen, Samuel Albanie, Gang Sun, and Andrea Vedaldi. Gather-excite: Exploiting feature context in convolutional neural networks, 2019.