Exp2: Convolution Neural Network

使用 pytorch 实现卷积神经网络，在 ImageNet 数据集熵进行图片分类（图片质量和数量均进行过压缩）。研究 dropout、normalization、learning rate decay、residual connection、网络深度等超参数对分类性能的影响。

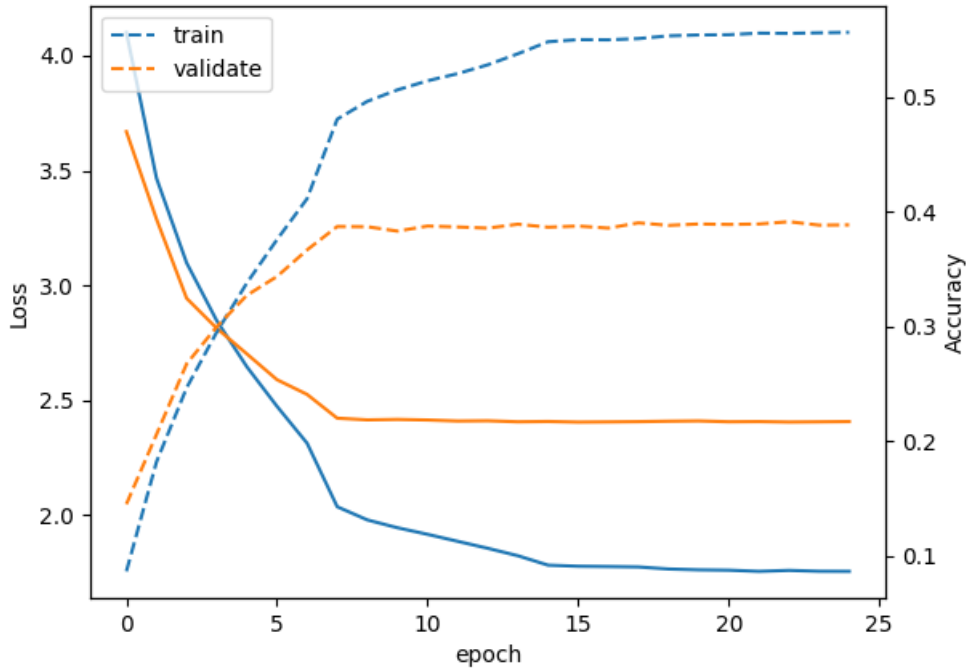
1. 基准模型 resnet18

主要程序如下：（参考 pytorch 官网教程）

```
data_dir = 'F:\\2021研一上学期\\深度学习\\ImageData2'
image_datasets = {x: datasets.ImageFolder(os.path.join(data_dir, x),
                                          data_transforms[x])
                  for x in ['train', 'val']}
dataloaders = {x: torch.utils.data.DataLoader(image_datasets[x],
batch_size=64, shuffle=True, num_workers=0)
               for x in ['train', 'val']}

if __name__ == '__main__':
    setup_seed(20211108)

    model_ft = models.resnet18()
    num_ftrs = model_ft.fc.in_features

    model_ft.fc = nn.Linear(num_ftrs, 100)

    model_ft = model_ft.to(device)

    criterion = nn.CrossEntropyLoss()

    optimizer_ft = optim.SGD(model_ft.parameters(), lr=0.001, momentum=0.9)

    # Decay LR by a factor of 0.1 every 7 epochs
    exp_lr_scheduler = lr_scheduler.StepLR(optimizer_ft, step_size=7,
gamma=0.1)

    model_ft = train_model(model_ft, criterion, optimizer_ft,
exp_lr_scheduler, num_epochs=25)
```

- 为了方便复现，固定随机数种子为本次实验截止日期，顺便提醒自己完成时间。（虽然最终并没有按时完成）
- 选择交叉熵作为损失函数，选择带动量的 SGD 作为优化器，学习率设置为 0.001；
- 使用 Learning rate decay 技巧，每过 7 个 epochs 便降低学习率，降低因子 gamma 取 0.1；

- 训练 25 个 epochs，查看 Loss 曲线和 Accuracy 曲线，以及验证集上的最佳准确率。



Best val Acc: 0.3916

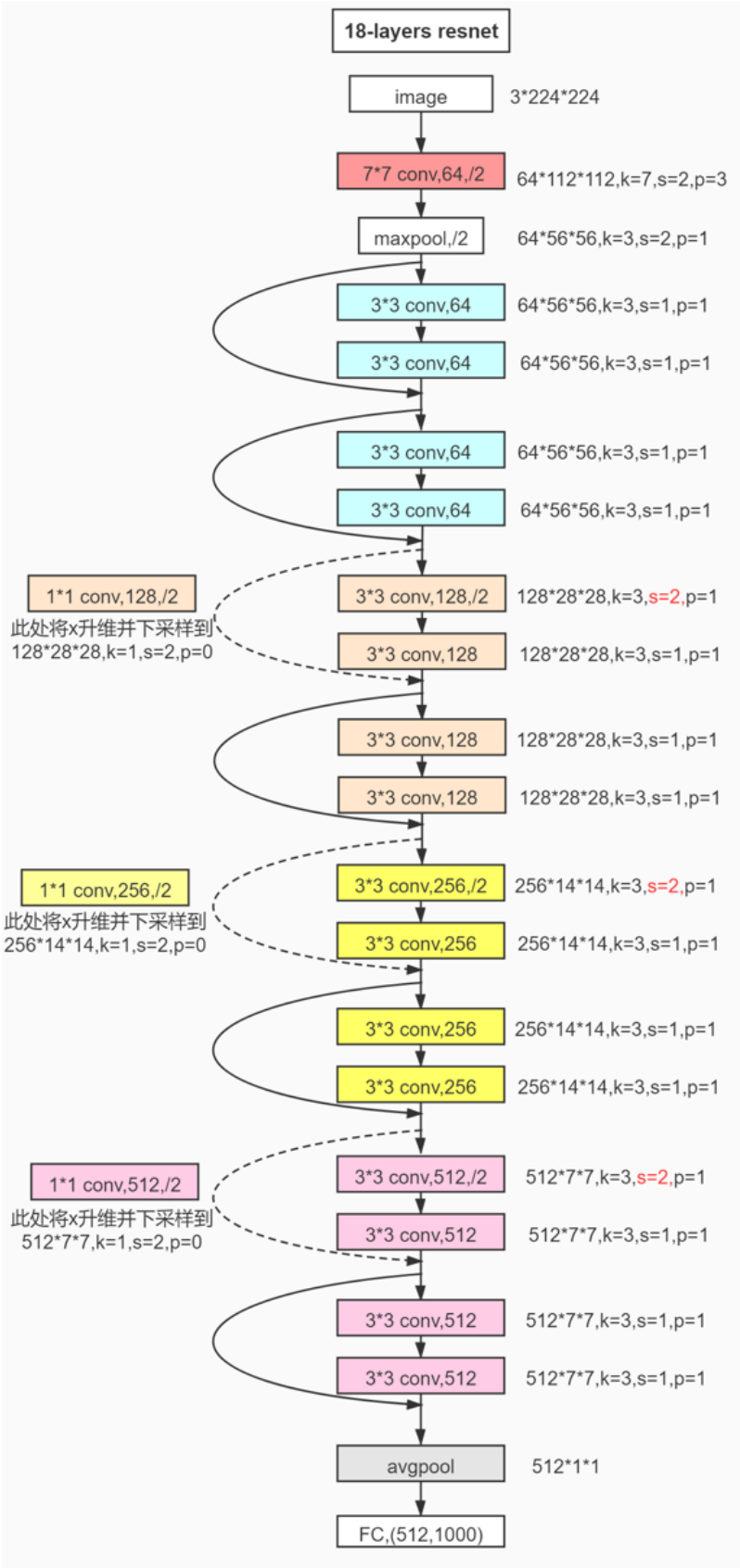
这个效果很一般，最佳准确率不到四成，而且明显存在过拟合，所以需要先对该基准模型进行改进。

2. 改进模型 xknet18

模型的改进主要从模型结构、模型参数、输入数据三个角度下手，根据实验结果筛选最佳的改进策略。

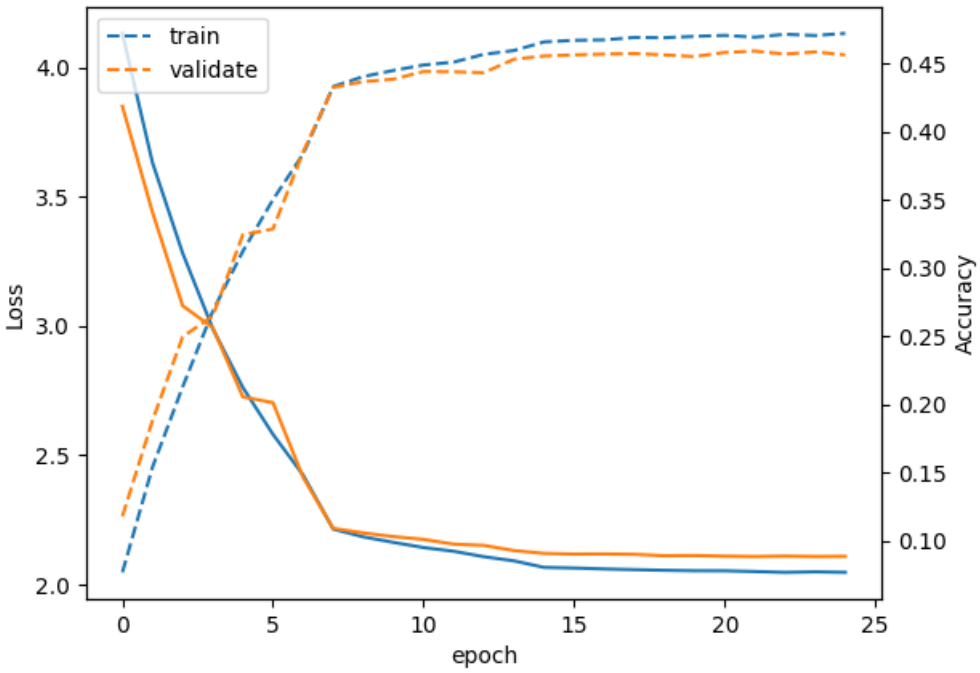
2.1 模型结构改进

从某博客上找到的 resnet18 结构如下：



这个网络结构是对原版的 ImageNet 数据集使用的，而我们所用的压缩版数据集，图片规格是 64*64 的，而且规模也比原数据集小很多，所以我对网络结构做了一些修改：

- 将 7*7 的 kernel 改为 3*3 的；
- 去掉第一个 maxpool 池化层.



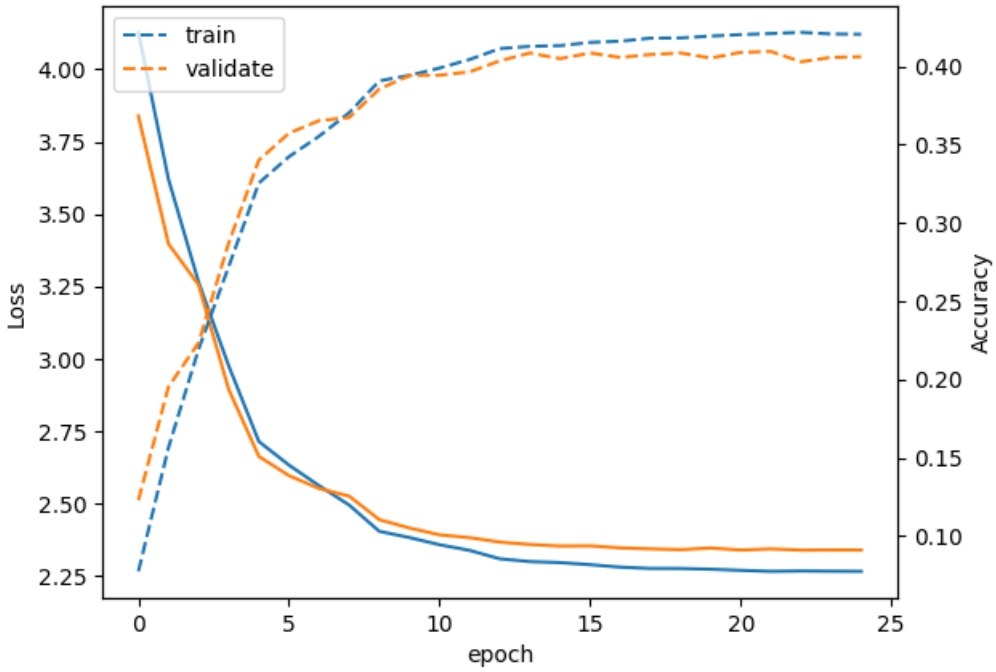
Training complete in 166m 30s
Best val Acc: **0.4592**

可以看到准确率上升了将近 7 个百分点。 而且 train 和 validate 的曲线贴合地比较好，大致可以说明没有过拟合或欠拟合。

2.2 参数调整

还是保持一次实验 25 epochs，调整以下参数：

- 将各卷积层的 channel 减小到原来的 1/2；
- 改变学习率下降参数：step_size=4, gamma=0.3



Training complete in 435m 16s

Best val Acc: 0.4096

由于我们的数据集像素是 64*64 的，比原版的 224*224 小了不少，所以将网络的宽度减小。但是没想到效果竟降低了很多，所以我决定不做这个改动了。

2.3 数据增强

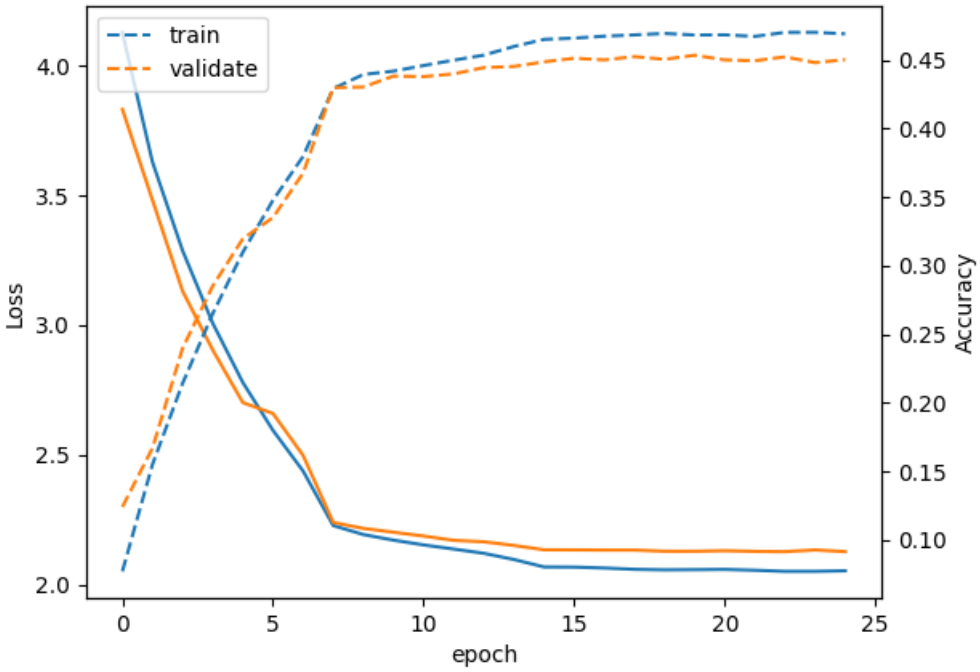
由于计算资源十分有限，所以只是随机选了一小部分数据集做增强，简单体验一下这个方法。

- 水平翻转（取 $p = 0.05, 0.1$，即随机选取约 5000 张图片进行水平翻转）
- 垂直翻转（取 $p = 0.05, 0.1$，即随机选取约 5000 张图片进行垂直翻转）

p=0.05, p=0.05

Training complete in 183m 50s

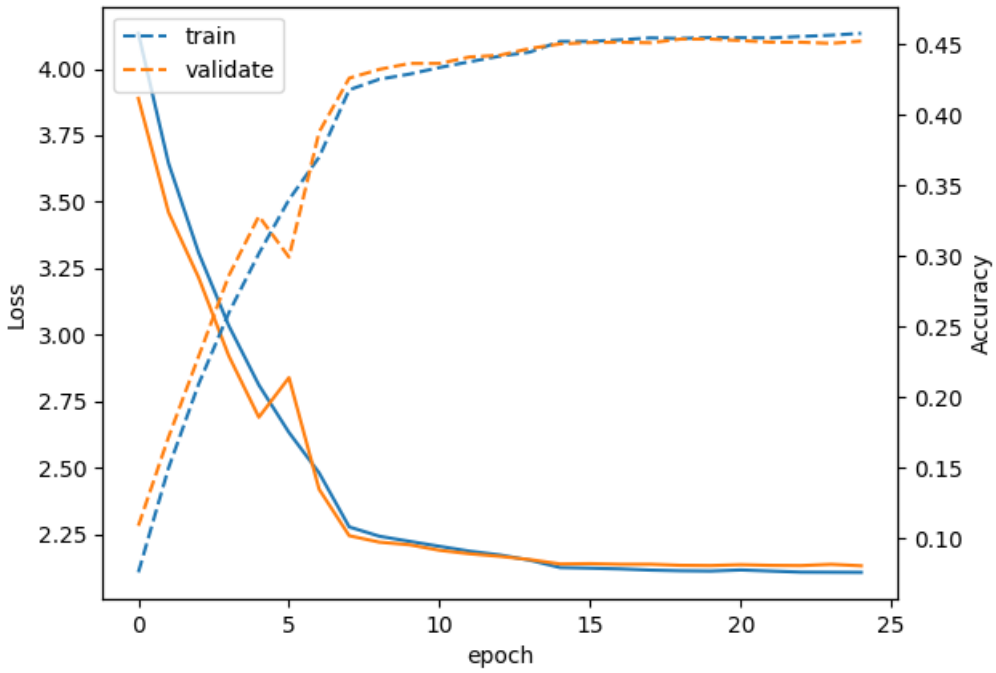
Best val Acc: 0.4534



p=0.1, p=0.1

Training complete in 204m 16s

Best val Acc: 0.453600



可以发现使用数据增强后，效果比 2.2 的，模型有所提升，但是两个不同的翻转概率并没有对结果造成什么影响。

最后用该模型预测 test set，结果写入 test.txt

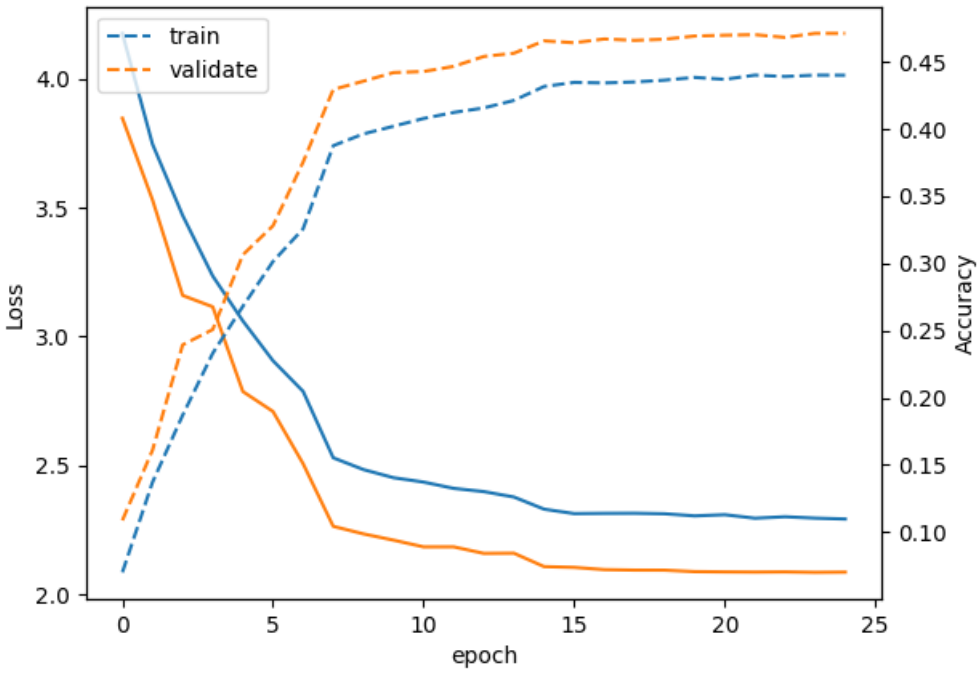
3. 研究关键超参数对分类性能的影响

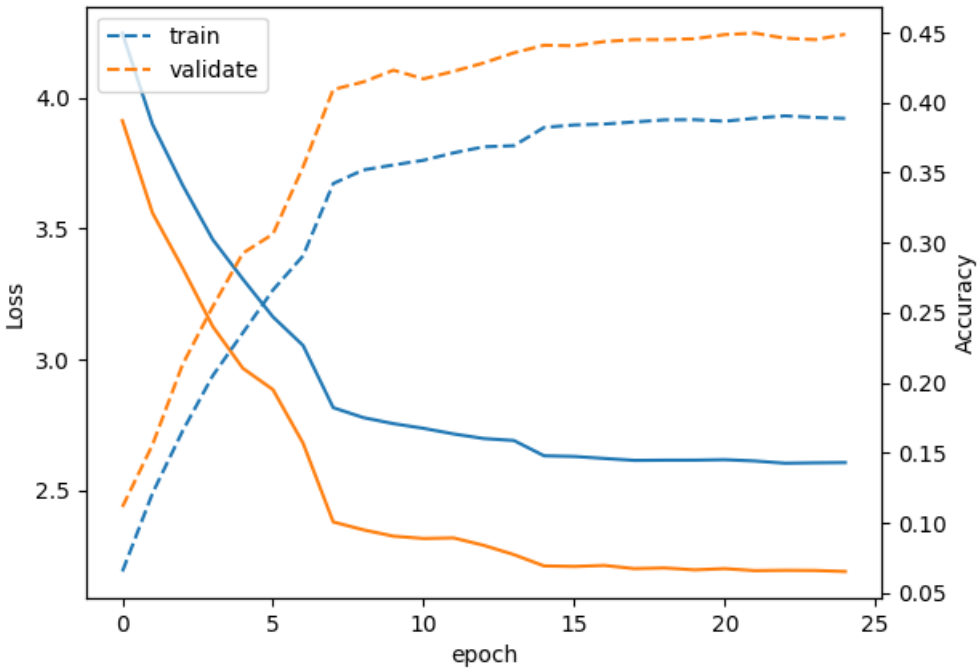
> 这一部分是我手动在原代码上进行更改后运行得到的结果，很抱歉对助教来讲要运行很不方便，我以人格担保实验的真实性。

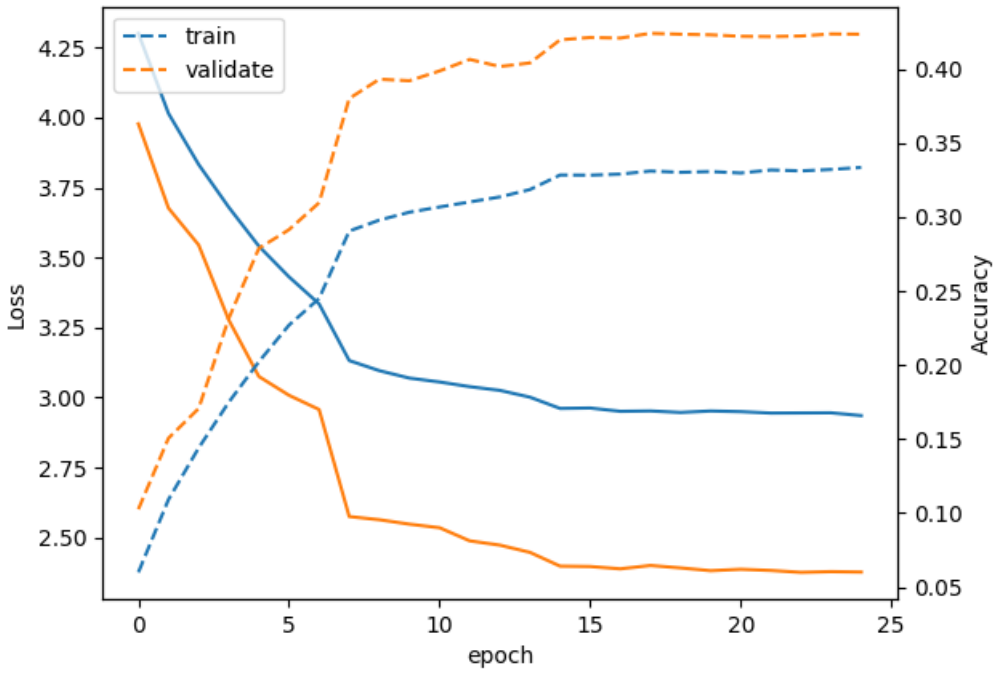
以下 5 个模块，只是在上述模型的基础上做了相应的更改，没有叠加不同超参的更改，所以参照是统一的，均为上述“改进模型”。

3.1 dropout

由于 Batch Normalization 和 Dropout 并用会导致方差偏移，所以我先将原有的 BN 层去掉，然后全连接层后面添加一个 Dropout 层，p = [0.1,0.2,0.3]，观察实验结果


train
validate
Loss
Accuracy
epoch
4.0
3.5
3.0
2.5
2.0
0.45
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0
5
10
15
20
25


train
validate
Loss
Accuracy
epoch
4.0
3.5
3.0
2.5
0.45
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0
5
10
15
20
25

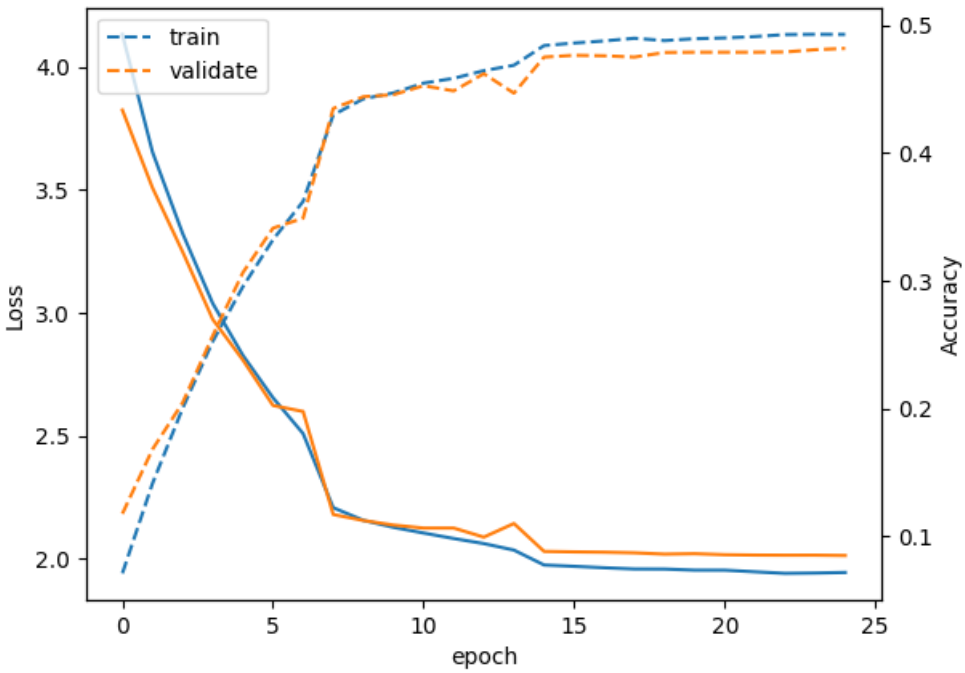


根据以上三张图，可以发现 Dropout 的作用还是很明显的，当我们逐渐增大 drop 的概率时，最优的 Accuracy 是持续下降的，而且模型的欠拟合越来越严重。所以对于我们这次的模型，其实不加 Dropout 效果会更好。

3.2 normalization

之前的实验都是默认使用了 Batch Normalization，下面我尝试了不使用 Normalization、使用 Instance Normalization 两种方式

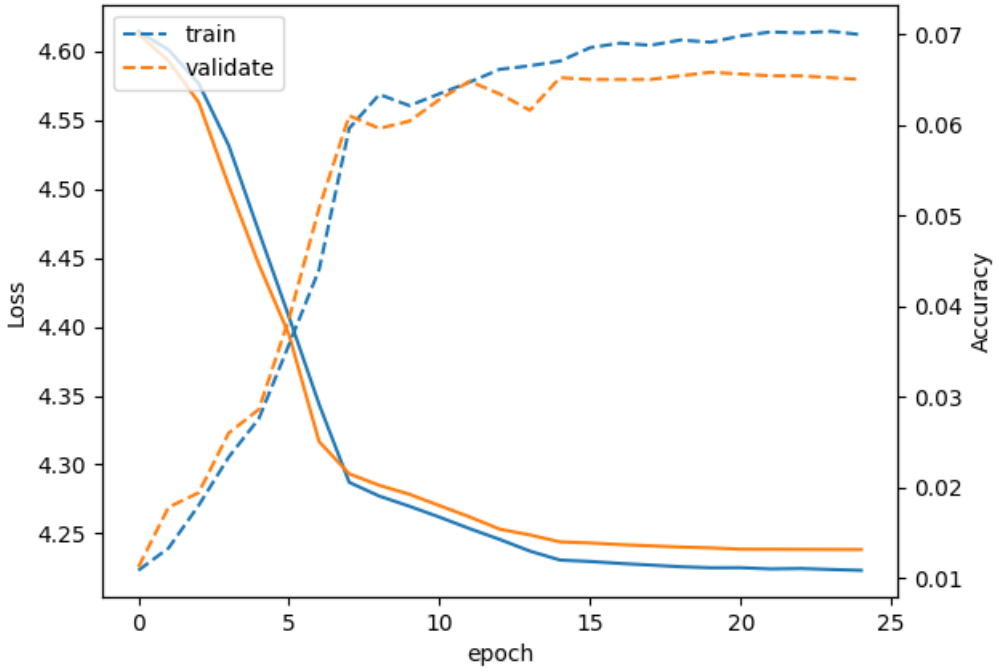
- 不用 normalization layer，观察实验结果



很神奇地发现，不用 normalization layer 结果反而变好了，这是让我有点困惑的地方。不过换个角度想，这个提升也并不是很明显，而且我的 epochs 只有 25，一方面可能是由于一定的随机性，一方面可能是模型训练得还不够。由于计算资源和时间的限制，我做不到更多的训练回合，更无法通过多次实验来缓解随机性的问题。

当然，也有可能是因为去掉正则化层之后，模型稍微过拟合了。

- 使用 Instance Normalization，观察实验结果



验证集上的最佳准确率只有 0.0658，远远低于使用 BN 的效果。Instance Normalization 进行标准化计算时，取的单通道，单样本上的数据进行计算，可能是因为得到的归一化统计量将不具有代表性，导致结果如此不好?

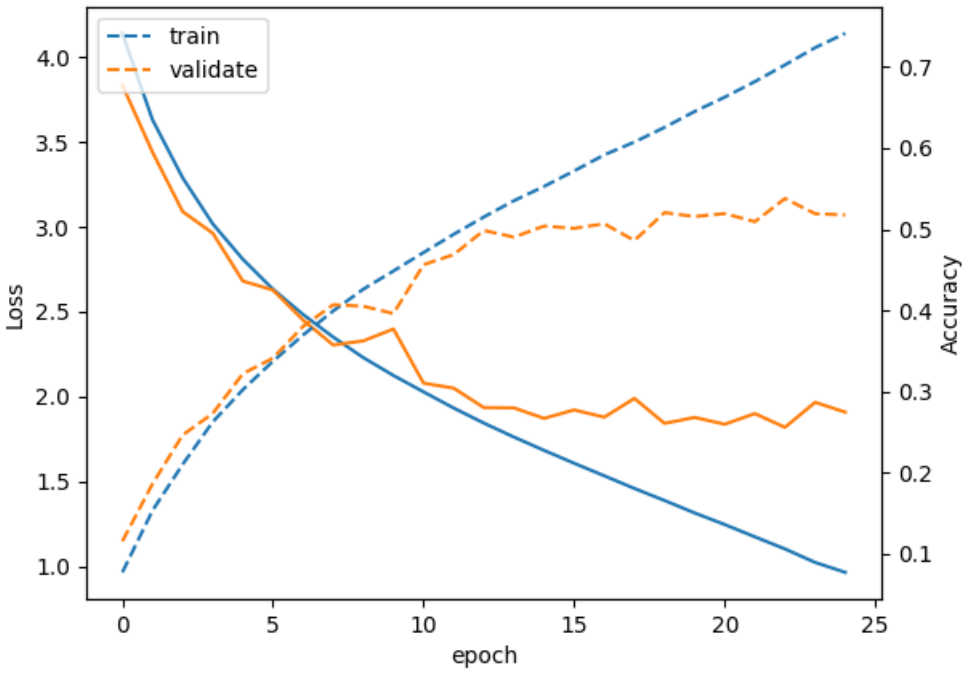
查找资料发现:

BN 适用于判别模型中，比如图片分类模型。因为 BN 注重对每个 batch 进行归一化，从而保证数据分布的一致性，而判别模型的结果正是取决于数据整体分布。但是 BN 对 batchsize 的大小比较敏感，由于每次计算均值和方差是在一个 batch 上，所以如果 batchsize 太小，则计算的均值、方差不足以代表整个数据分布;

IN 适用于生成模型中，比如图片风格迁移。因为图片生成的结果主要依赖于某个图像实例，所以对整个 batch 归一化不适合图像风格化中，在风格迁移中使用 Instance Normalization 不仅可以加速模型收敛，并且可以保持每个图像实例之间的独立。

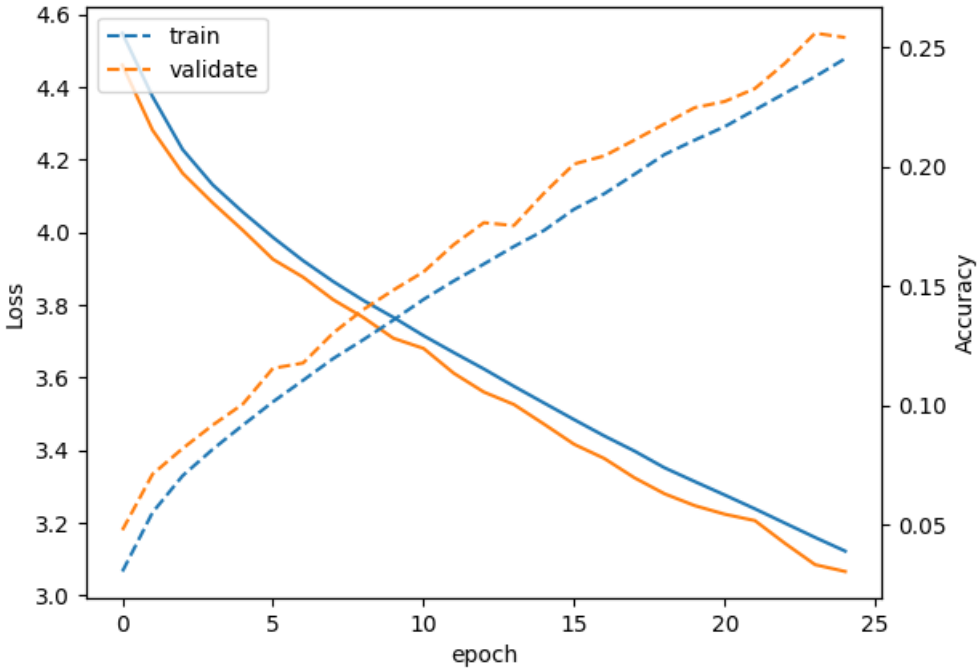
3.3 learning rate decay

不用 learning rate decay 技巧，保持 lr=0.001，观察实验结果



这个结果就有点一言难尽了。虽然准确率看着挺高，但是明显很不健康，一方面是因为迭代不够，但是即便增大 epochs，想必 train 和 validate 的曲线还是有很大的 gap，这就是由于学习率（后期）太大导致模型不收敛。

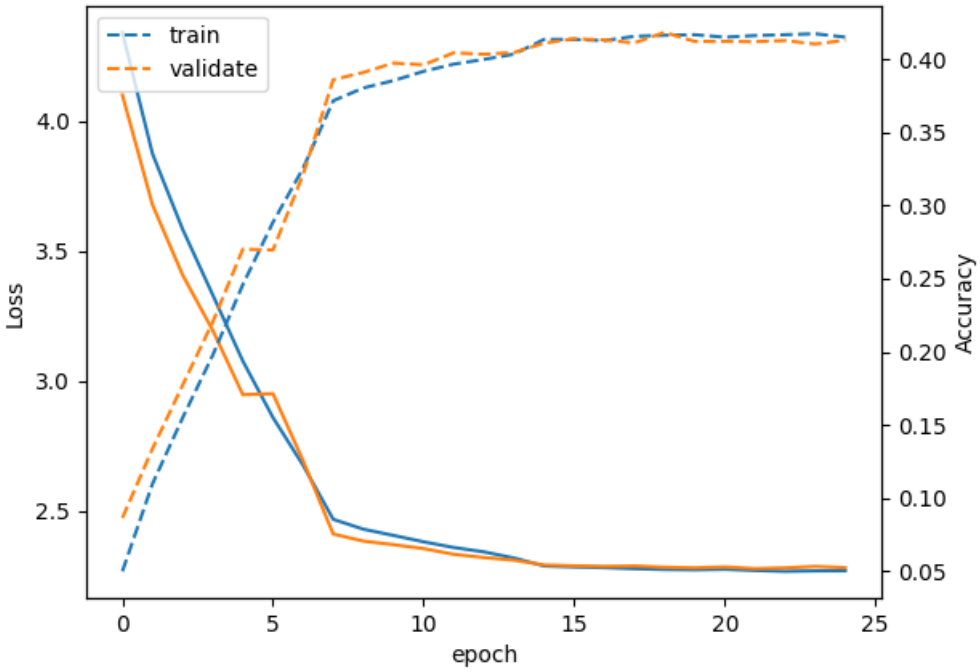
那如果我一开始就把 learning rate 设置得小一些呢？尝试一下 lr=0.0001，观察实验结果



这个相对就比较正常了，但是明显是训练还不到火候，可能需要再来 25 个回合。不过我发现这两个图有一个共性，那就是 train 的曲线都非常的丝滑，不知为啥可以这么丝滑？

3.4 residual connection

去掉 shortcut 结构，查看实验结果



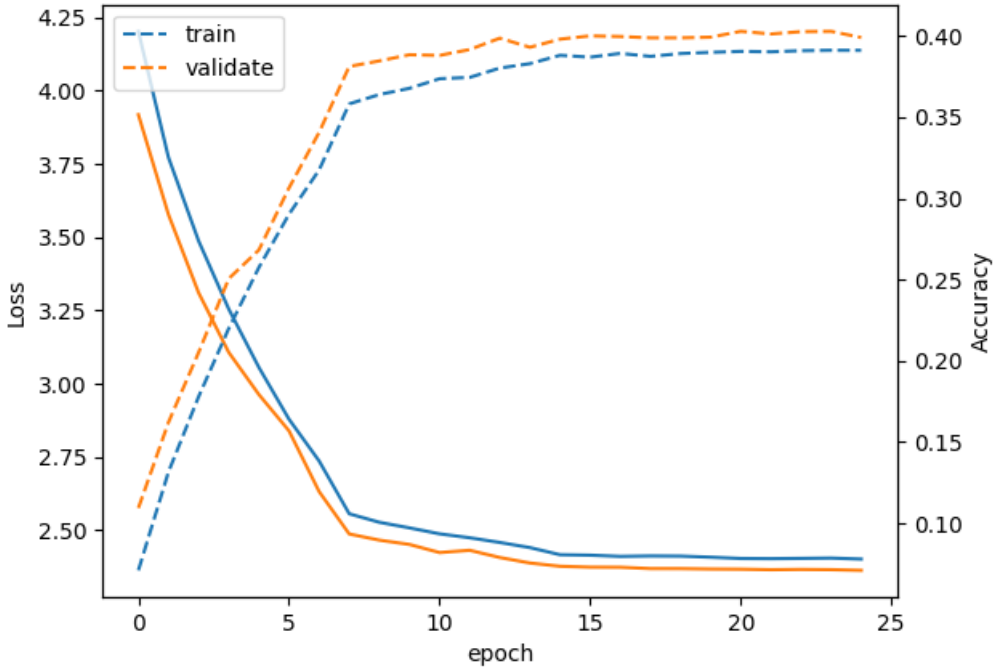
Best val Acc: 0.4184

结果明显不如修改之前的，准确率大概下降了 5 个百分点，这个很合理，毕竟没了 resnet 的精髓与核心模块，效果肯定要差不少。

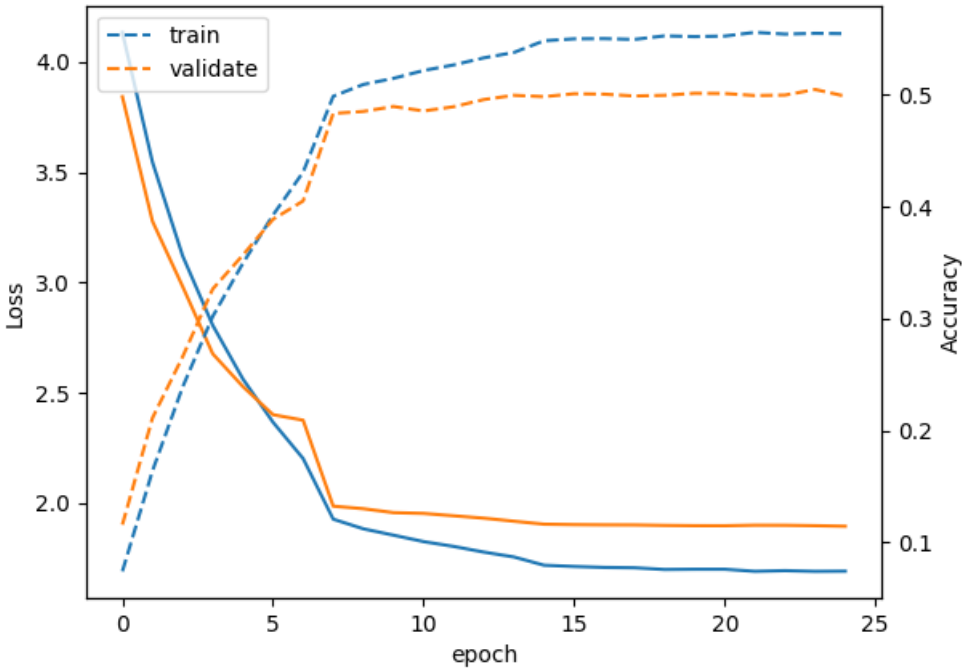
3.5 网络深度

在 2.1 的基础上，同样保持 4 个大层，并保持每个 block 的结构不变，改变每个层的 blocks 数量，分别得到 14 层和 34 层的 resnet 网络，观察实验结果

- resnet14



- resnet34



同时对比 2.1 中 resnet18 的验证集准确率约为 0.46，还是可以明显看出，网络加深之后模型的性能有较大提升。不过 resnet34 的训练时间约为 resnet14 的两倍，时间成本大大增加。

4. 总结

通过此次实验，我 ~~深深地体会到了计算资源对于科研实验的重要性，~~体会到了深度神经网络的强大之处。很多我用人眼都看不出来的图片，神经网络可以从 100 个类中精准地找到图片真正的归宿，而且我只是用了一个18层的网络简单训练了 25 个回合而已，这无疑是很令人震撼的。

不过更多的是自己工程能力的提升，通过这次实验，我第一次用深度神经网络来完成一个图片分类任务，并且研究了各种超参数对网络性能的影响，虽然过程中还有些东西不太理解，但也收获了很多，是对自己的一次很好的锻炼。