0.文件结构和执行方法

- /src：代码目录

  main.py：主函数入口，包括训练和测试代码

  debug_model.py: 测试模型能否过拟合小规模数据

  parameters.py: 全局超参数

  models.py, model_self.py:模型定义和调用，其中model_self.py是自己的taskC模型的具体定义

  data.py:数据读取

  cnn_layer_visualization.py, deep_dream.py, misc_functions.py:引用的可视化库代码

  plot_cnn_visualize.py,plot_confusion_matrix.py,plot_tsne.py,visualize.py:CNN可视化，混淆矩阵绘制，t_SNE可视化代码

  plot_curves.py:测试训练结果可视化代码，使用tensorboard

- /data：数据集目录
- /result: 可视化和训练测试结果目录

  generated/layer_visualization:CNN可视化结果

  generated/deep_dream:deep_dream可视化结果

  models:最优模型所在地

  reports:训练，测试结果

  visualizations:t-SNE和混淆矩阵可视化结果

执行方法：

运行主训练-测试程序（使用下文的baseline，效果最好）

```
python main.py
```

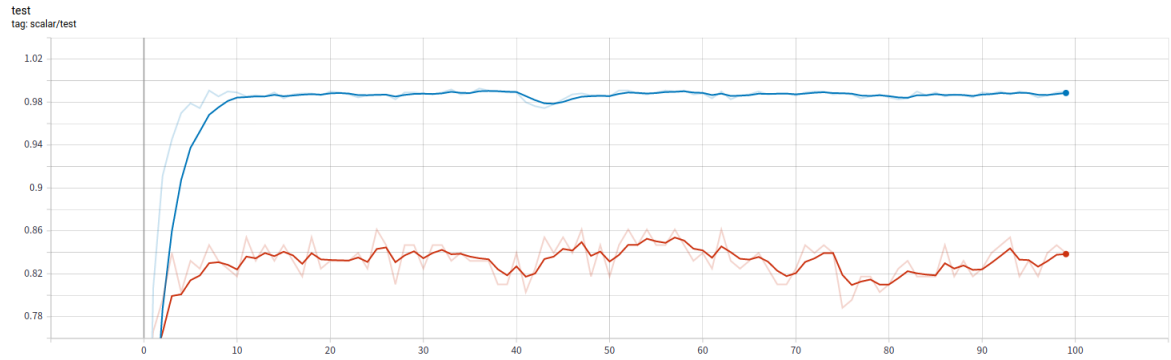
各种可视化（执行时间较长）

```
python visualize.py
```

用tensorboard可视化（需要安装tensorboard）

```
python plot_curves.py
```

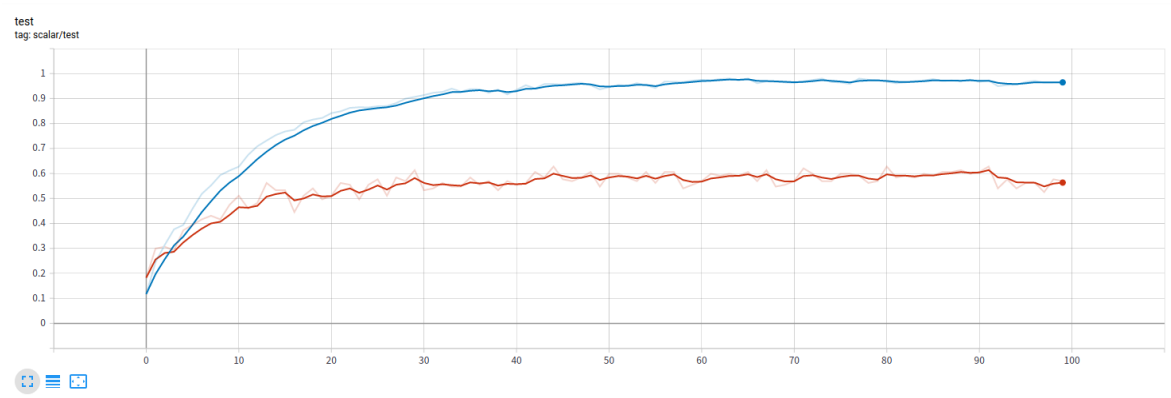
1.taskA



蓝线为训练准确率，红线为验证准确率。

可以看出，模型在有预训练的情况下收敛的很快，准确率也较高，最高达到85%左右。

2.taskB



蓝线为训练准确率，红线为验证准确率。

没有预训练的模型准确率不高，而且收敛很慢，勉强达到60%

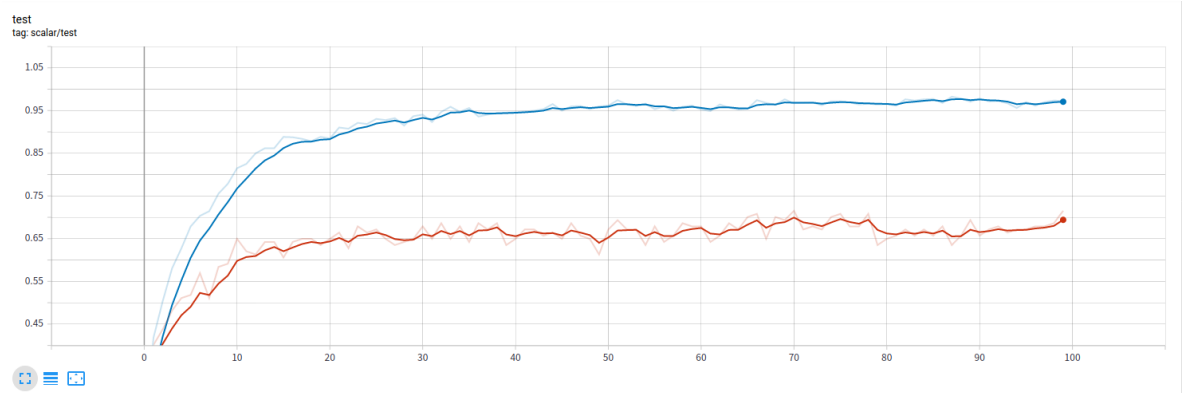
3.taskC

3.1 基本情况

模型使用3层卷积层（每一层都是卷积+ReLU激活函数+Batch Normalization），一层池化层和一层全连接，具体如下：

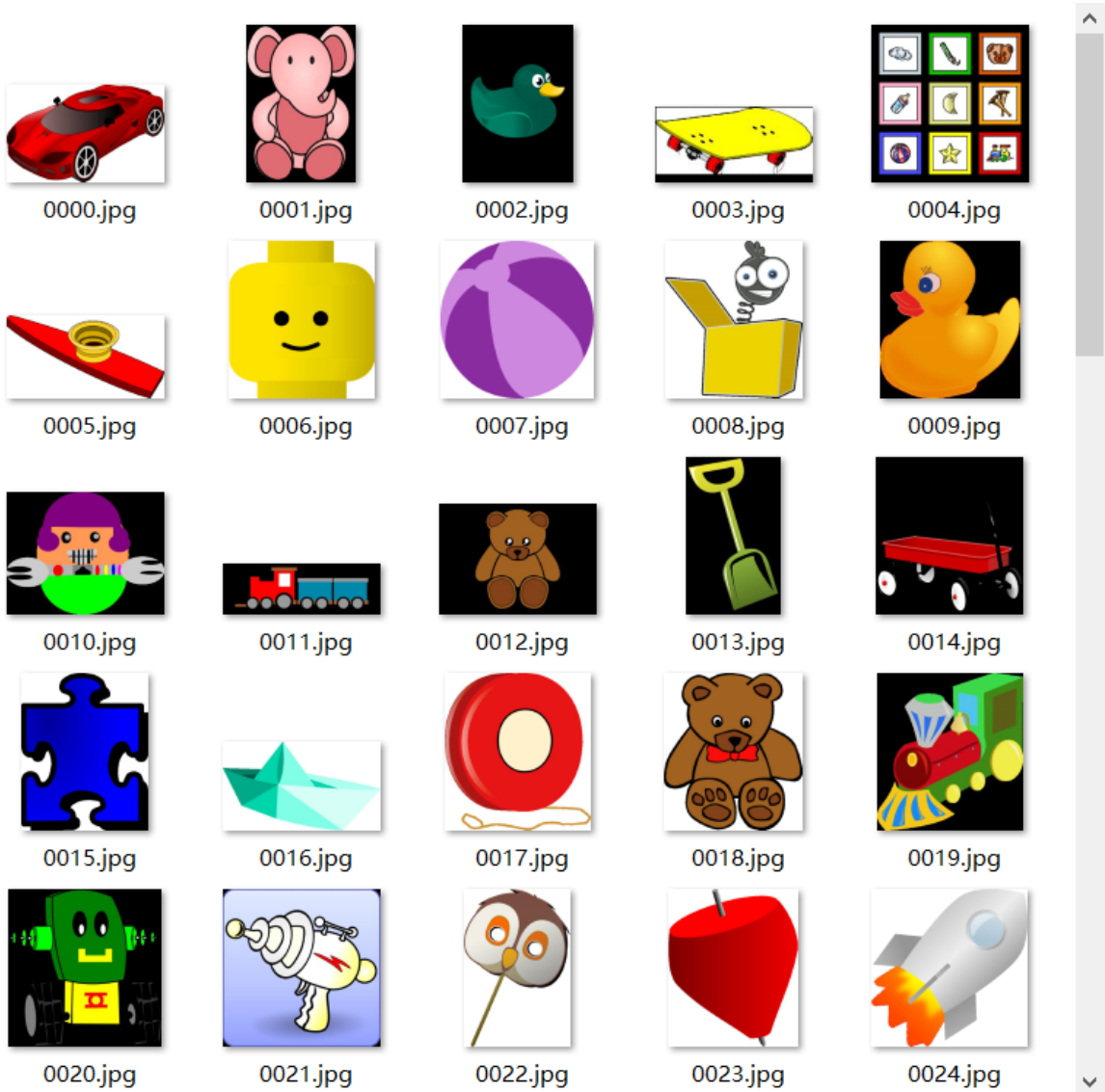
模型训练的其他超参数如下：

- 学习率10^-4，不进行衰减
- 使用参数为(0.64,1),(1,1)的RandomResizedCrop，其余数据增广方法和提供的代码一致
- 使用pytorch默认的Xavier初始化
- 不dropout
- 使用不带weight_decay的Adam优化算法



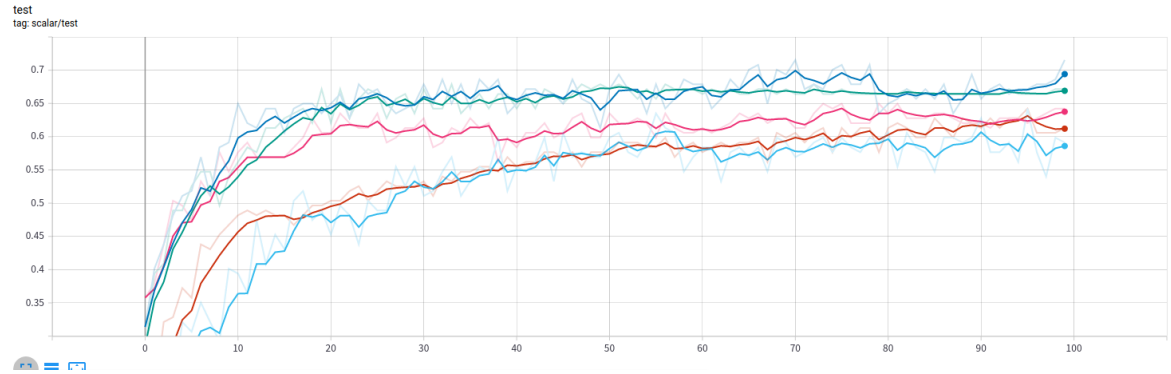
蓝线为训练准确率，红线为验证准确率。

模型准确率勉强达到70%，并不高。因为这组数据集并不大，只有1092张图片，而且样本方差也比较大，比如这个玩具类，包含了各种类型的玩具。用这样的数据集训练非常容易过拟合。



3.2 学习率的选择和实验

在网络和其余超参数都和上文基准C模型一样的情况下，我对五种学习率选择进行了实验，结果如下



深蓝色：基准学习率10^-4；浅蓝色：高学习率10^-3；深红色：低学习率10^-5；深绿色：10^-4学习率，快速衰减（每20epoch减半）；粉色：10^-4学习率，慢速衰减（每10epoch减半）

可以看出，10^-4的学习率是最好的，学习率过低(10^-5)，过高（10^-3)都差很多。

而且，学习率衰减效果不是特别好，慢速衰减的效果差于不衰减，快速衰减的效果和不衰减差不多。

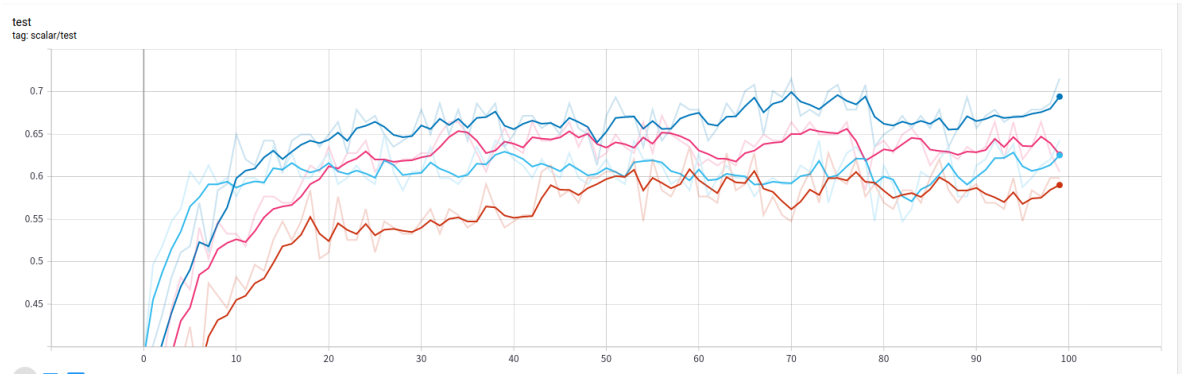
3.3 数据增强

示例代码中，就用RandomResizedCrop进行了随机裁剪，还用了RandomHorizontalFlip随机翻转。除此之外，我还了解到，有旋转，颜色变化，灰度化，高级裁剪等多种数据增强操作。

因此，我将在3.3.1,3.3.2中，证明裁剪和翻转操作的有效性，并且对比这两种操作的多种不同参数来获得一个好的baseline。在3.3.3-3.3.6中，我将在baseline基础上加上其他数据增强操作并且进行试验。

3.3.1 裁剪

示例代码给的数据增强里有RandomResizedCrop，其第二个参数代表留下的图片大小范围，第三个参数代表图片缩放尺度。在网络和其余超参数都和上文基准C模型一样的情况下，我们对这种裁剪方法进行了实验。



红色：高自由度裁剪，第二个参数为(0.08,1)，第三个参数锁死(1,1)

深蓝色：低自由度裁剪，第二个参数为(0.64,1)，第三个参数锁死(1,1)

浅蓝色：不裁剪，第二个参数为(1,1)，第三个参数锁死(1,1)

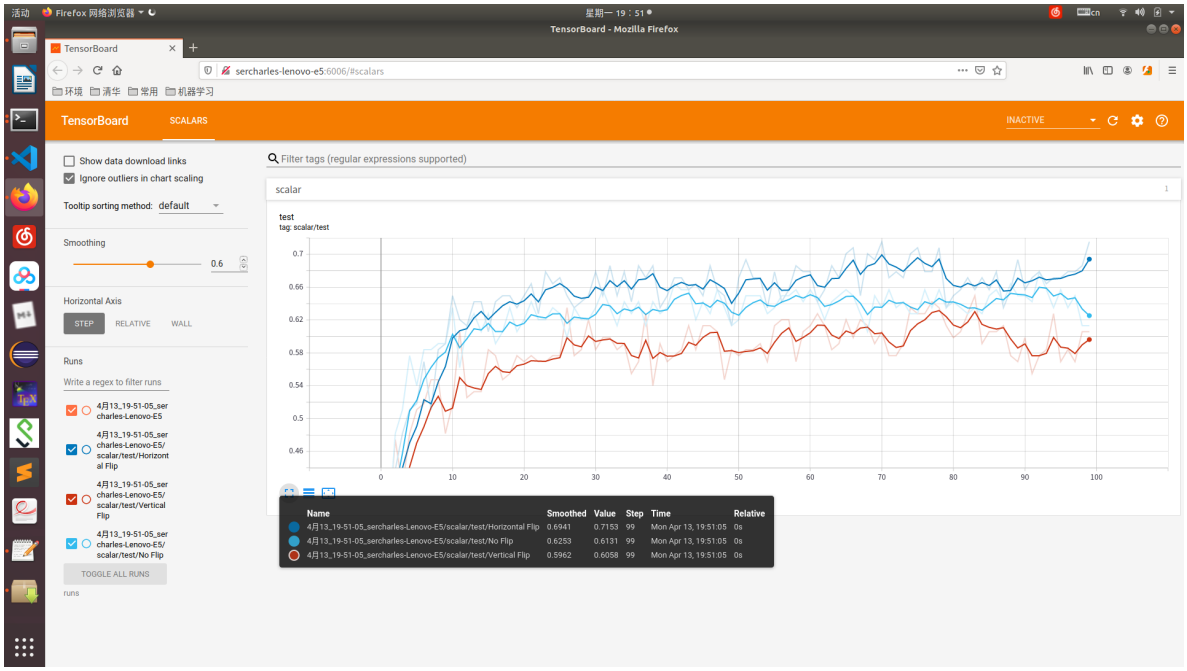
粉色：第二个参数为(0.64,1)，但是第三个参数不设置，默认为(0.75, 1.3333333333333333)

首先，低自由度裁剪的效果比高自由度和不裁剪都要好。

其次，锁死第二个参数为(1,1)很有效，能够大大改进模型效果。

3.3.2 翻转

示例代码里使用了RandomHorizontalFlip，除此之外，还有RandomVerticalFlip。在网络和其余超参数都和上文基准C模型一样的情况下，我们对翻转方法进行了实验。



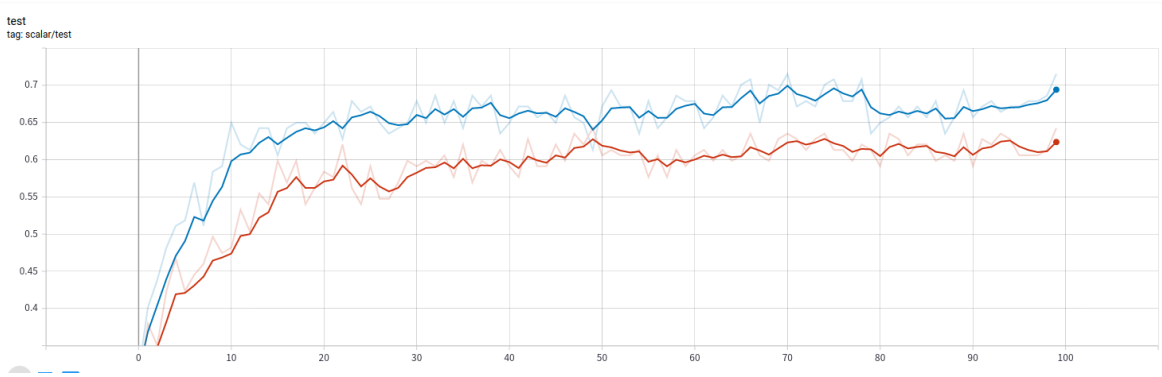
深蓝：水平翻转；浅蓝：不翻转；红色：垂直翻转

可以看出，水平翻转效果是最好的。

综合3.3.1，3.3.2，我们得到了baseline：使用第二个，第三个参数分别为(0.64,1)和(1,1)的RandomResizedCrop，还有RandomHorizontalFlip。

3.3.3 旋转

在网络和其余超参数都和上文基准C模型一样的情况下，我们对在基准模型（做crop和flip）的条件下，是否加上旋转做了实验

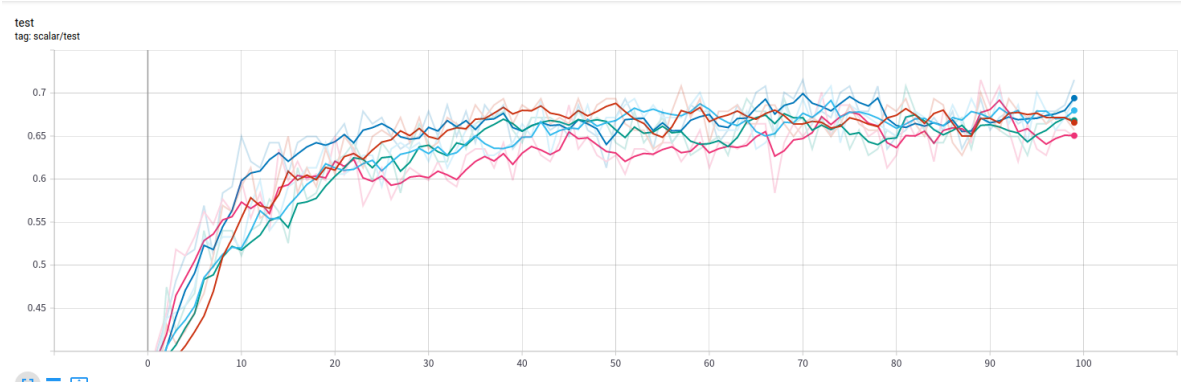


红色：随机30度；蓝色：不旋转

可以看出，旋转对于模型效果起了负面作用。

3.3.4 颜色

在网络和其余超参数都和上文基准C模型一样的情况下，我们对在基准模型（做crop和flip）的条件下，是否加上几种颜色上的变换做了实验

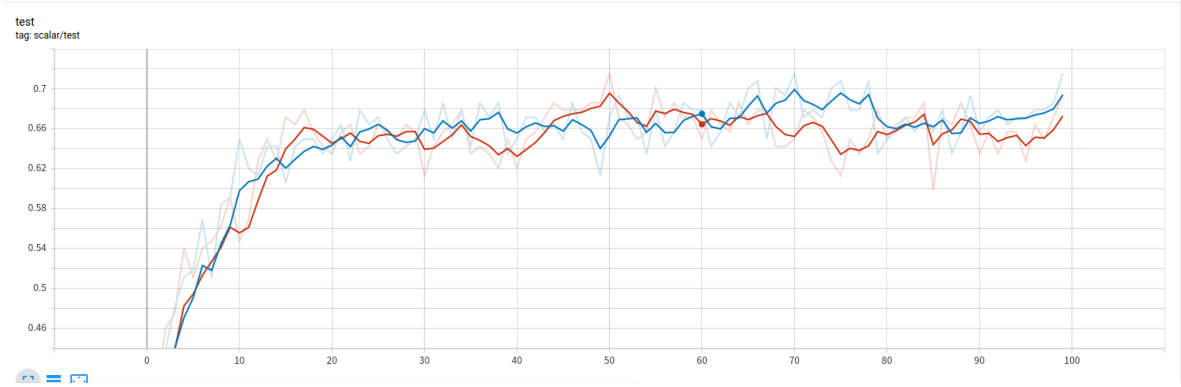


深蓝：不变；红色：亮度变化；浅蓝：对比度变化；绿色：色调变化；粉色：饱和度变化；

可以看出，单独加上任何一种变化，都没有办法显著改进模型。

3.3.5 灰度

在网络和其余超参数都和上文基准C模型一样的情况下，我们对在基准模型（做crop和flip）的条件下，是否将图片随机灰度化做了实验

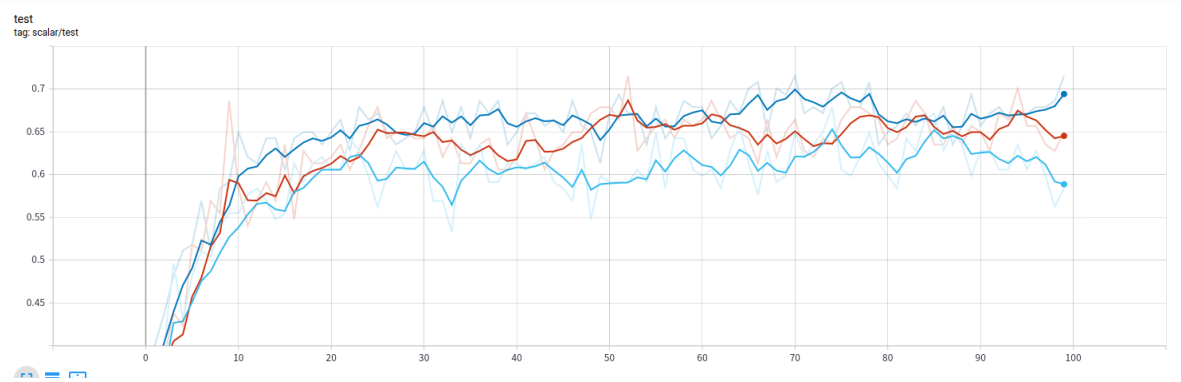


蓝色：不变；红色：0.1概率随机灰度化

灰度化对模型没有改进。

3.3.6 高级裁剪

在网络和其余超参数都和上文基准C模型一样的情况下，我们对在基准模型（做crop和flip）的条件下，对高级裁剪（Fivecrop，Tencrop）做了实验



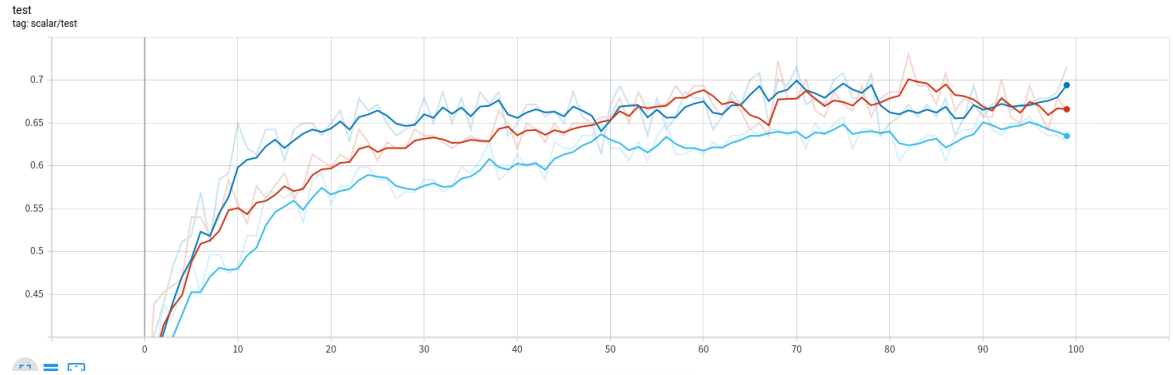
深蓝：没有操作；红色：5-crop；浅蓝：10-crop

可以看出，高级裁剪对模型没有显著改进，而且因为要把图像规模扩增到原来的5/10倍，大大增加了时间，空间开销，不可取。

3.4 优化策略

课上讲过，Adam优化算法和SGD优化算法是两个常用的优化算法。而且，在优化中，为了防止过拟合，往往加上weight_decay，也就是在损失函数后加上L2正则项约束来限制参数大小。

在网络和其余超参数都和上文基准C模型一样的情况下，我选取了SGD优化算法，Adam优化算法，带weight_decay的Adam算法进行对比。



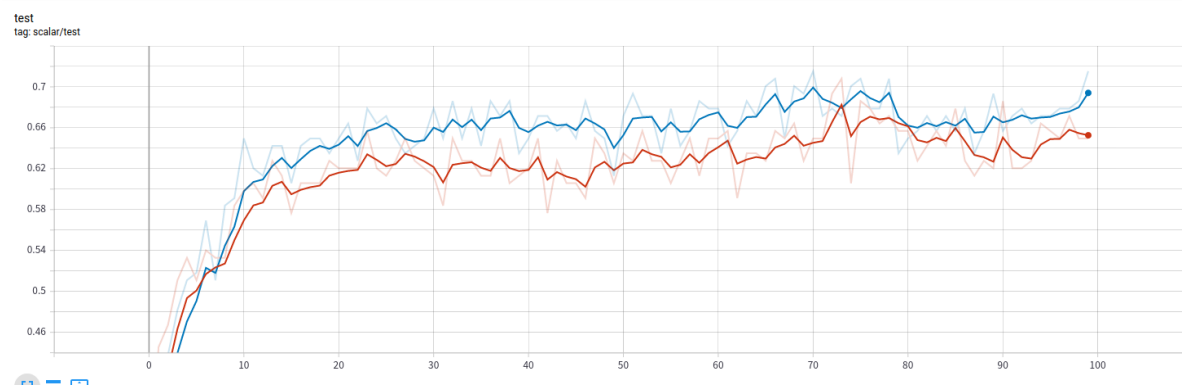
深蓝：Adam算法；浅蓝：SGD算法；红色：带weight_decay的Adam算法

可以看出，Adam算法比SGD算法要强很多，因此选用Adam算法更好。同时，weight_decay对于模型并没有显著的优化效果。

3.5 初始化策略

参考这篇文章，可以看出，pytorch的默认初始化是Xavier初始化。而课上讲过，Kaiming初始化对于使用ReLU激活函数的CNN效果更好。

因此，在网络和其余超参数都和上文基准C模型一样的情况下，我对比了使用默认初始化和Kaiming初始化的情况。



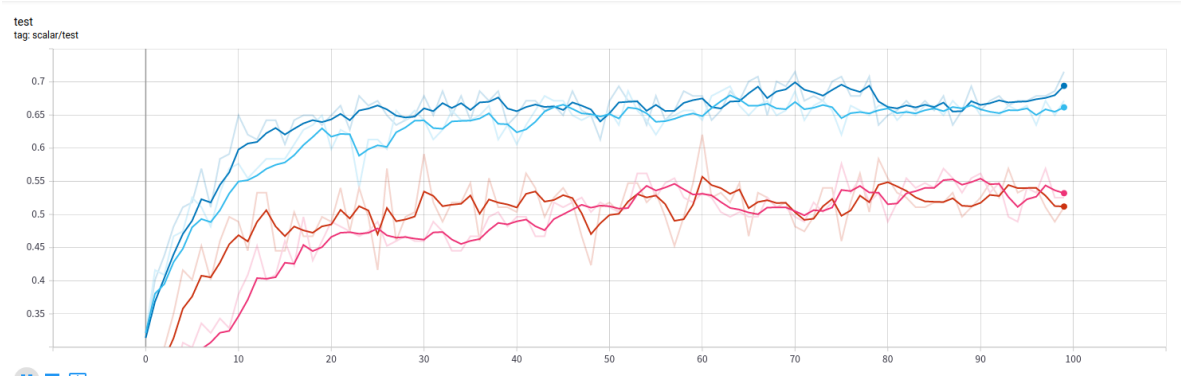
蓝色：默认Xavier初始化；红色：Kaiming初始化

可以看出，默认初始化的效果更好一些。

3.6 dropout

Dropout是一种有效的防止过拟合的方法。因为最后的全连接层只有一层，如果dropout的话，在验证的时候如果屏蔽正确的通道，那么结果也不会正确，因此，在最后一层使用dropout理论上会严重降低模型准确率（实际上经过测试，dropout50%的通道，准确率低于50%），就没有使用这个进行对比。

在网络和其余超参数都和上文基准C模型一样的情况下，我对比了四种情况：没有dropout，三层卷积都dropout，只dropout第一卷积层，只dropout第三卷积层。



深蓝：无dropout；浅蓝：只dropout第三层；深红：只dropout第一层；粉色：全部dropout

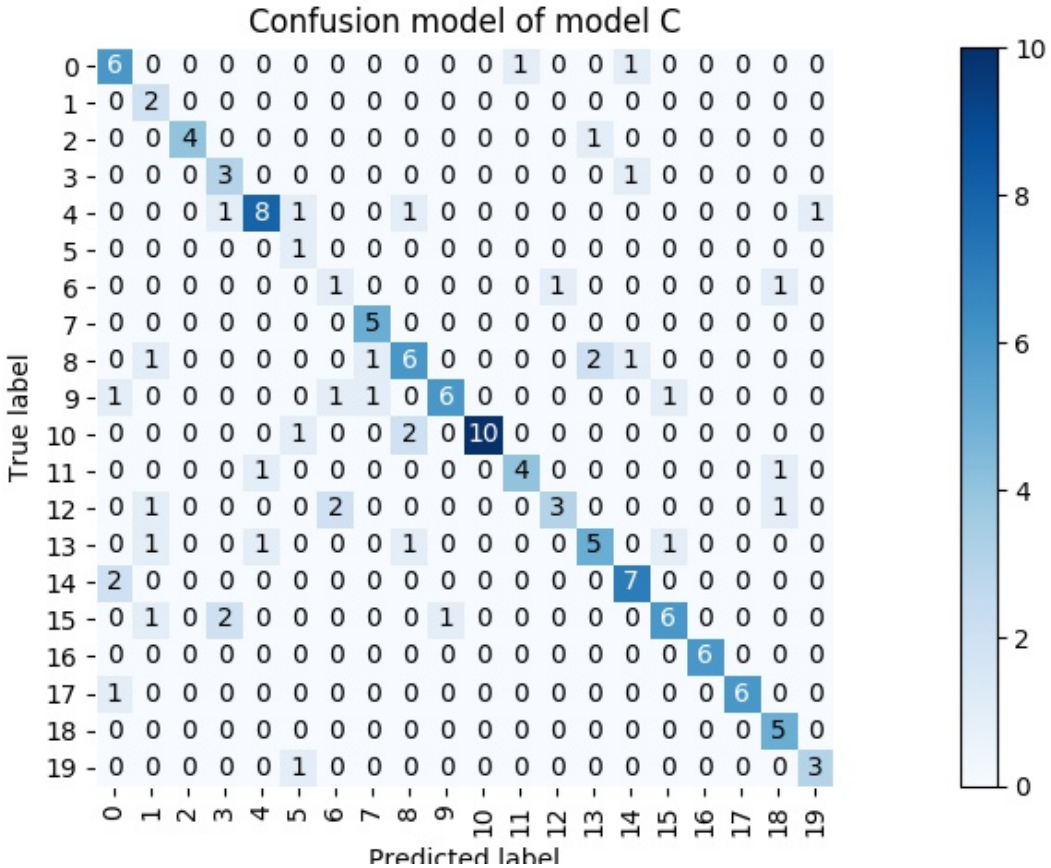
可以看出，dropout全部层和只dropout第一层效果都比不dropout较差，只dropout第三层也没有改善模型准确率。一般来说，卷积层进行dropout效果都不太好，一般dropout都用于全连接层，这个实验与结论大致吻合。

4.可视化

可视化的代码集中在visualize.py中，运行这个文件就可以进行全部可视化了。

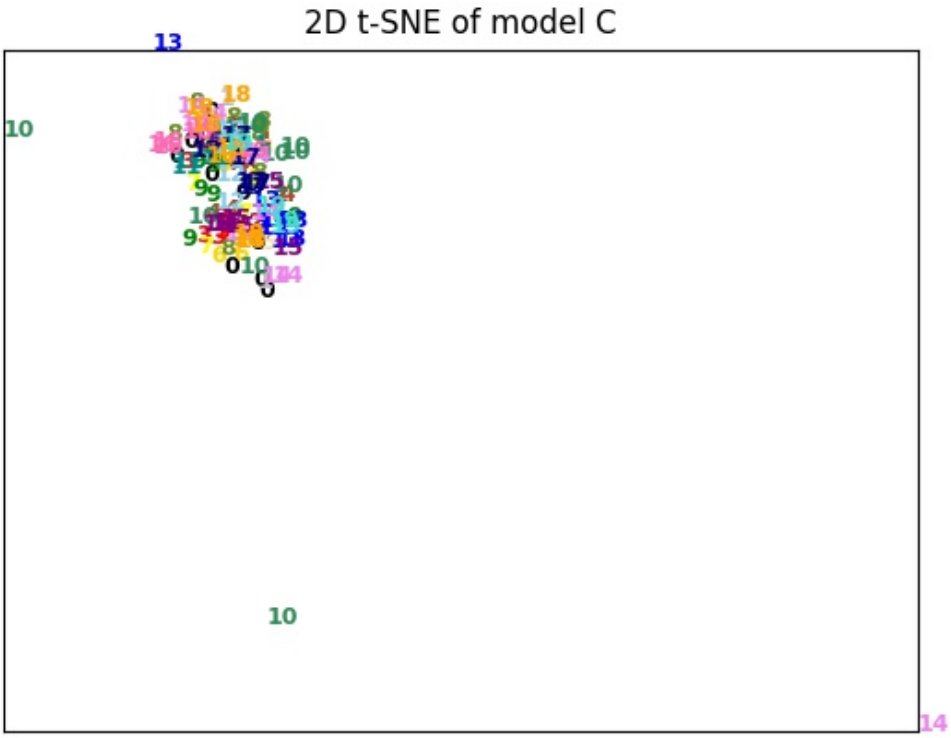
4.1 confusion matrix

绘制confusion matrix的代码在plot_confusion_matrix.py中。

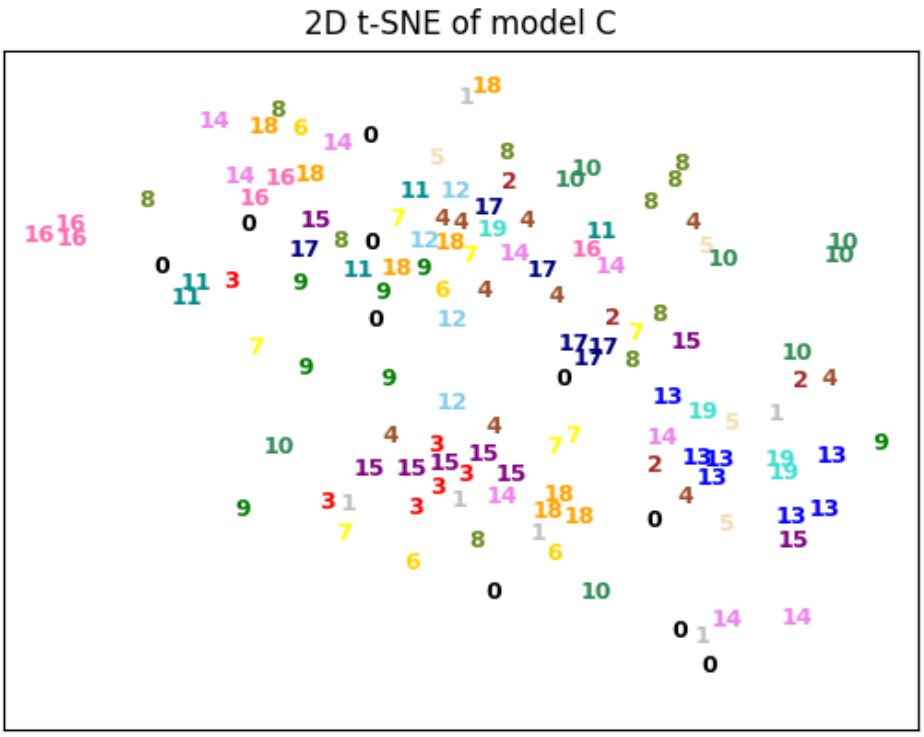


4.2 t-SNE可视化

绘制t-SNE可视化的代码在plot_tsne.py文件中



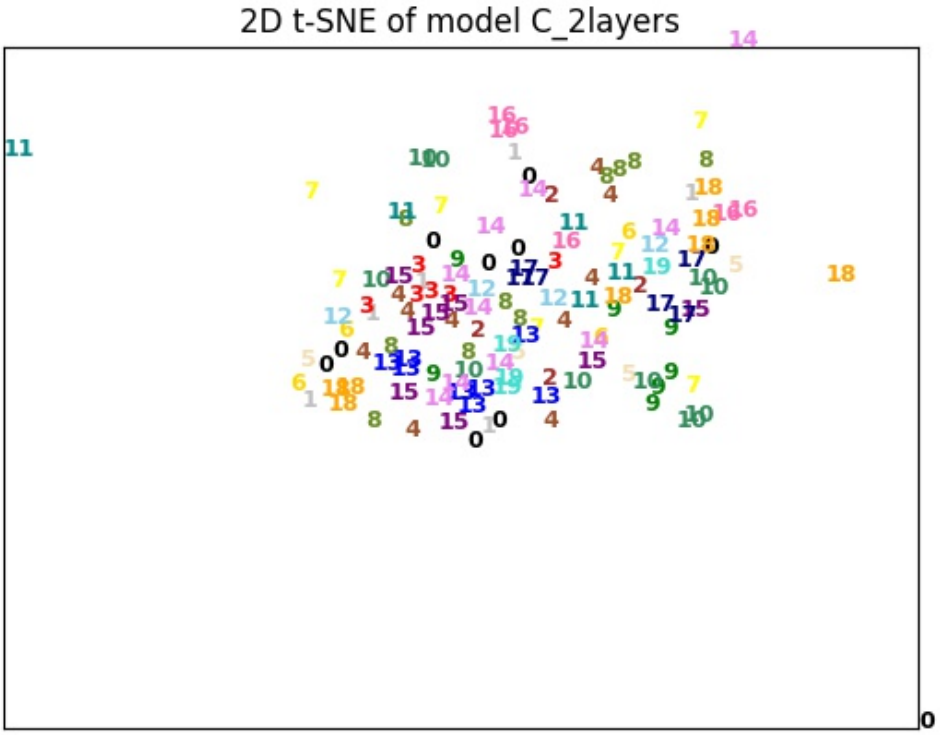
放大后如下：



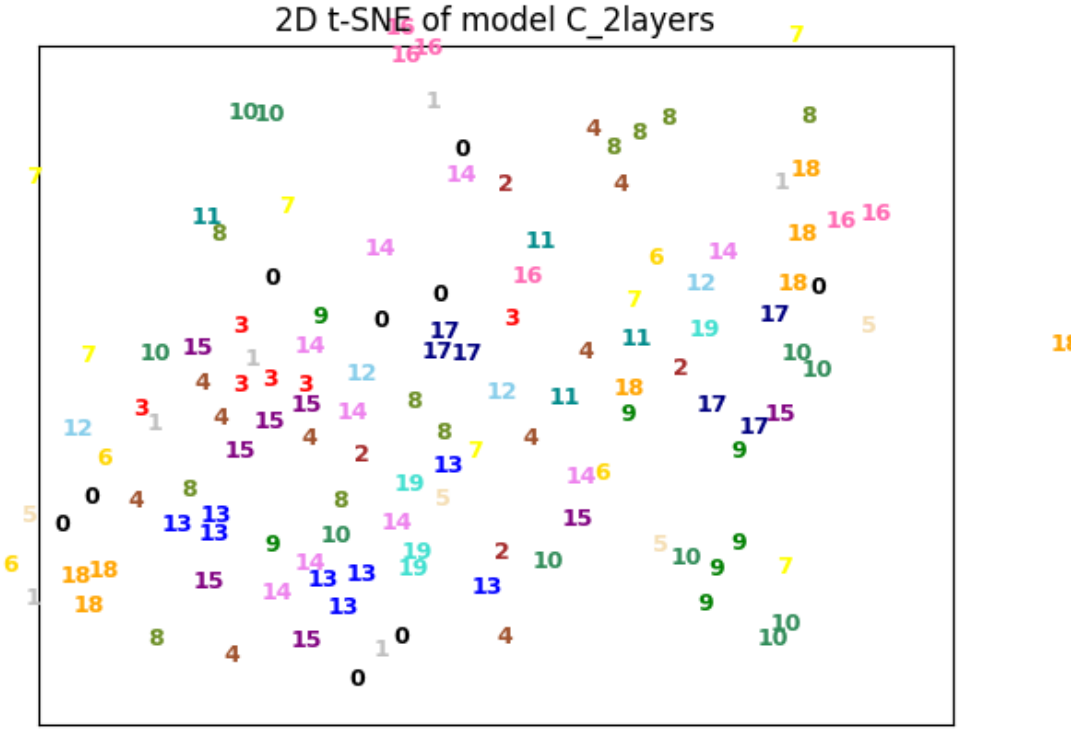
在只有一层全连接的情况下，这个分类并不够明显。

因此，我猜测，使用两层全连接的话，经过一次线性变换，分类会更加明显一些。

我把网络的最后一层全连接拆分成了一个input·512的全连接和一个512·20的全连接，在最后一层全连接之前进行可视化，结果如下：

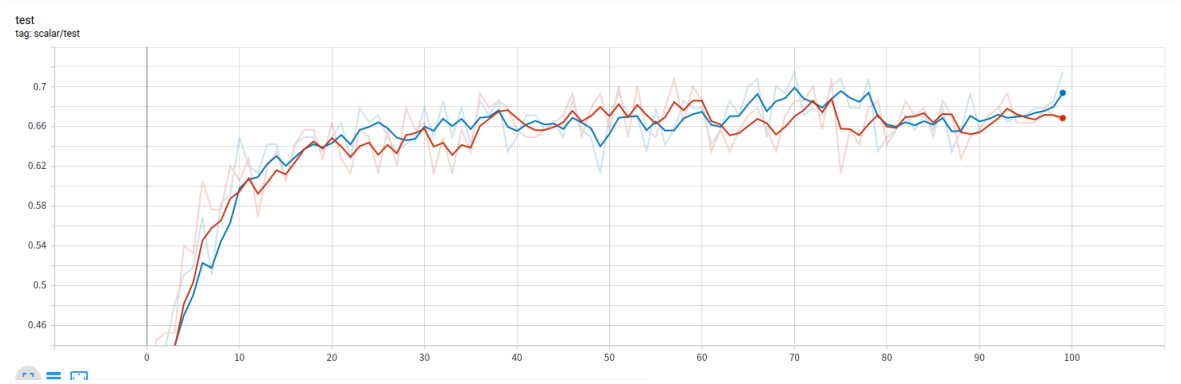


放大后如下：



这个分类就比之前更加清晰一些，尤其是在没放大的时候。

但是增加一层全连接，网络的过拟合更加严重了，准确率下降，而且训练和测试开销大大增加，我并没有最终选择两层全连接的网络。



红色：2层全连接网络；蓝色：一层全连接网络

4.3 其余的可视化

我使用了pytorch cnn可视化的这个项目，使用它的cnn_layer_visualization模块对三层卷积层的每个channel进行了可视化，结果在result/generated/layer_visualization中。我还挑了一张图片，用这个项目的deep dream模块对三层卷积层的每个channel进行了可视化，结果在result/generated/deep_dream中。

5.总结

这次实验收获很大，基本熟悉了pytorch的读取数据，数据增广，模型实现，读取和保存模型，训练，测试等多个方面。我还对深度学习训练中遇到的常见问题，比如欠拟合和过拟合有了一定的了解，并且养成了制定合理的训练计划，以及在模型训练前先用少量样本进行debug（如果能过拟合再继续训练）的习惯。除此之外，我还了解了Tensorboard，t-SNE，confusion matrix，CNN层可视化等常用的可视化工具。