本周大量时间花在 debug 和重构之前代码上面。前期主要借鉴学长在组会中提到的用很小批量的数据来进行过拟合以验证代码的正确性。使用的之前参考作者源代码中提供的网络结构，作者的网络结构设计也是为 6 帧中的每一帧都分配了一个单独的提帧网络，刚开始认为作者如此做应该有他的道理，所以包括训练部分和验证部分都按照这个思路设计，但是后面通过对这 6 个单独网络中的参数进行对比发现完全一模一样，我认为完全没有必要为每一帧都单独设计一个网络，故改写了部分的网络结构。

分析了之前写的训练部分代码中计算量大两个函数 ACTMatchTube 和 ACTComputeConfLoss，最初的设计是使用 numpy 在 cpu 中使用循环实现，其运算速度特别慢，GPU 长期空闲，训练程序基本无法使用。后面将其中不合理的循环设计放到了 GPU 中用矩阵切片操作的方法来实现，速度得到明显提升，GPU 也基本可以跑满。训练代码基本可用并且已经训练两天，在这之中尝试了训练过程中对前面预训练部分 freeze，只训练最后的分类和定位回归部分，以及全部网络均进行训练两种方法，观察了不同的训练方法对训练效果的影响。

重写了 map 部分的程序，map 部分需要计算 map 就需要对每一个视频的每一帧都取连续的 6 帧进行计算，粗略估计计算 map 时单个视频的计算量和单个视频的实际帧数有关，计算量可以达到训练时的几十倍甚至上百倍，不像训练的时候每一个 epoch 中对于每一个视频其实只是随机选取的一段 ground truth 进行计算。之前计算 map 的部分参考的作者提供给的源代码，作者的代码中处理一段视频会反复去磁盘读取连续的 6 帧，然后将该帧计算后的 nms 结果存盘，之后再读取计算其 map，该种方法的计算速度非常慢，平均速度基本为 1fps，重复读取明显就是极不合理的设计，现在的设计中以视频为单位一次读取一全部帧然后按照单卡最大 batch 送入 GPU 中进行计算之后不存盘直接将 nms 后的数据放在内存中等待全部计算结束后使用。这样的做法已经将计算 map 的速度提升到 8fps，但是速度还是很慢，UCF101 数据集的 test 视频有 910 个，这样的速度对每一帧都进行。经过分析，在对 feature 进行提取的过程中这种思路依然做了很多重复计算。分析后认为在 eval 阶段完全可以以视频为单位将网络提取 feature map 和预测 loc 和 conf 这两个部分分为两步进行，GPU 计算完成后直接将 GPU 计算的结果交给另一个进程将每一帧的计算结果进行分配，存入内存等待计算 map，这样将两个部分完全分割开应该可以显著提高 eval 的计算速度。接下来需要马上再改写 map 部分的计算代码进行提速，尽快验证网络是否过拟合。

实验中发现 confidence 部分的 loss 相对而言偏高，且下降慢，根据论文，是在计算完了 conf 部分所有正负样本的 loss 之后，选择了直接除以正样本的个数，训练开始阶段这样输出的 loss 差不多刚好是-ln(1/numclass)的 4 倍，这和正负样本比例为 1:3 基本吻合。分析后认为是计算 cross_entropy 时没有对每一个样本取平均，也就是说梯度累加过多导致步长变大容易导致发散，通过对正负样本的总个数取了平均之后情况基本可以接受，当然这就和论文中提到的公式有所出入了。

实验中将前面提取 feature 部分的网络进行了 pre-train，故参考之前的操作考虑将已经初始化过的部分网络进行了 freeze，只对采用随机数初始化的 loc 和 conf 网络单独进行训练，最后总的 loss 停留在了 2.7 左右无法继续下降，之后将网络中所有的参数全部参加更新继续训练，但是将前面的参数全部放开之后可用 batch size 急剧减小，基本上减少为原来的六分之一，我分析后认为可能是前面提取 feature map 的网络中的参数需要进行 6 次重复计算其计算图需要占用大量显存资源，直接导致显存依次无法容纳更多的数据，所以计算速度相对也降低了。现在网络的总 loss 已经下降到 1.0，还有继续下降的趋势。

显卡驱动重新折腾:
https://blog.csdn.net/yuanqia4079/article/details/84993628