这周机械学院毕业设计开题答辩，准备文献翻译以及开题报告改来改去折腾了很多时间：

- 对网络 loss 计算部分的代码进行了优化，训练速度提升了 20%（原来双卡一个 epoch 需要 17 小时，现在 4 卡一个 epoch 需要 6.5 小时，我认为这个是正常的现象，因为这个网络比较大（前面的特征提取部分要重复计算 6 次），显卡在计算的过程中就是需要花费这么长的时间）对 pytorch 的多卡训练的数据分配流程以及数据的流动有了更深刻的理解
- 对网络参数初始化以及通过参考 pytorch 的源代码网络对步长的分组调整的策略
- 经过这周的折腾我开始怀疑之前 UCFSports 很好的结果是抽奖出来的，给我造成了一种没有啥问题的错觉。要注意的东西实在是太多了。。
- **这周刚开始花了 3 天跑了一个完整的但是参数初始化的时候一开始没注意搞错了浪费了时间，只有 0.58 的 map，经过这些折腾之后现在在 UCF101 训练的 map 稳定提高，，经过 5 个 epoch 今天就达到了 0.6150，明天一定就可以到作者的水平了！这次一定行！**

这周对于训练速度慢的问题，试图找到原因并对其进行优化，主要做了以下工作

- 作者在 caffe 程序中是将寻找正样本的过程放在了训练过程中的 loss 计算，但是考虑到对正样本的寻找其实只和 ground truth 的坐标有关，而所有的 prior tubes 其实都是固定的，所以其实对正样本的寻找过程并不依赖网络的结果， 这一步骤放在训练阶段会占用部分训练时间。所以将 match tube 的工作放在 dataset 中直接完成，这样不用返回 ground truth，直接返回 encode 之后的结果用于计算 loss 就可以了。这样可以将在后面的 match 和 encode 的时间节约出来，（虽然其实才 0.1s）
- 因为之前没有采用上述的方法，所以如果一张图中有两个 ground truth 或者多个 ground truth 直接返回 ground truth 的坐标会在 dataloader 中合并为 batch 的时候出错，没有想到什么解决办法，所以之前都是每一段的 ground truth 单独返回，这样训练的样本数高达 38W 个。现在直接返回 encode 之后的数据，所以一张图有多个 ground truth 直接 encode 到 8396 个 prior tubes 中，样本数可以减少到 32W，这也减少了训练时间。同时使得难例挖掘更加合理，因为在有多个 ground truth 中如果只标注了一个，很有可能会把原来的正样本错认为难负样本。
- 在对代码各个部分花费的时间方面进行分析发现，花费时间最多的地方在网络的计算中，由于 pytorch 在 GPU 上的数据计算都是**非阻塞的**！如果后面的操作中没有对前面数据的依赖或者是需要用到显卡，程序会很快执行过去，但是一旦需要前面数据的地方就会阻塞等待！所以在找时间占用最大的地方为此纠结了很久，因为在 loss 的计算中为了等待显卡数据，会在一个明显不需要很多时间的地方等待大量时间，这对程序时间花费的分析带来了非常大的误导，浪费了很多时间。所以我觉得就目前的分析结果来看，这个网络对于这么多数据，一个 epoch 绝大部分都是在网络的计算上，所以可供优化提速的空间很少。
- 对于 eval 的过程，因为一个视频是作为整体进行 eval 计算的，所以视频间的差异导致不好直接使用多卡，之前一直用的单卡，依次 eval 要接近两个小时。这周改成多进程多卡分工，eval 时间降到 40min。

经过前面的优化出了一次训练结果，但是效果并不好，再次重新回去找曾经被自己忽略的各个部分，决定严格按照我能看到的作者的步骤来，并作出了以下修改：

- 步长的分组规则：用于提取 feature 的 weights 的步长设置为 base_lr/6，用于分类和回归的 weights 的步长设置为 base_lr，网络的所有 bias 的步长为 weights 步长的 2 倍，

且 weight_decay 设置为 0;

- 使用 basw_lr=0.0001 训练 10W 个 batch，之后使用 0.00001 训练 3W 个 batch，最后使用 0.000001 训练 2W 个 batch，时间上如果严格按照该思路，预计一次完整的训练需要 65 个小时。
- 之前的训练过程中，经常会出现训练完成了一个 epoch 之后就停止，修改一些参数后继续训练，这样做之后 optimizer 则重新初始化，通过查阅 SGD 的源代码，发现每次训练的过程中 momentum_buffer 有对之前梯度的历史信息，所以重新初始化优化器会导致历史信息被清零，显然该做法不可取。

本周对 object relation module 以及 siamese network 进行了学习和理解

object relation module:

总的来说这个模块的思想不难: 通过对四个矩阵 (WV (1*1 卷积层) ,WK (全连接层参数) ,WQ（全连接层参数）,WG（全连接层参数)）的学习考察两个物体的外观特征和几何特征之间的联系。对于选中的第 n 个物体，它和其他所有的 m 个物体之间的特征通过一个矩阵变化之后的加权平均,

$$\mathbf{f}_R(n) = \sum_m \omega^{mn} \cdot (W_V \cdot \mathbf{f}_A^m).$$

加权平均系数ω^{mn}代表第 m 个物体对 n 个物体产生影响的权重，则是根据第 n 个和第 m 个物体之间的几何位置关系和外观特征关系经过 softmax 之后得到。

作者最后将多个这样的 fR(n)进行 concat, 得到和输入的外观特征一模一样的 relation feature, 再和输入的外观矩阵进行加和，得到一个新的矩阵，因为这个 module 不会改变前后的关系所以可以在任意地方插入。作者为每一个物体都会收集 16 个这样的 relation feature。

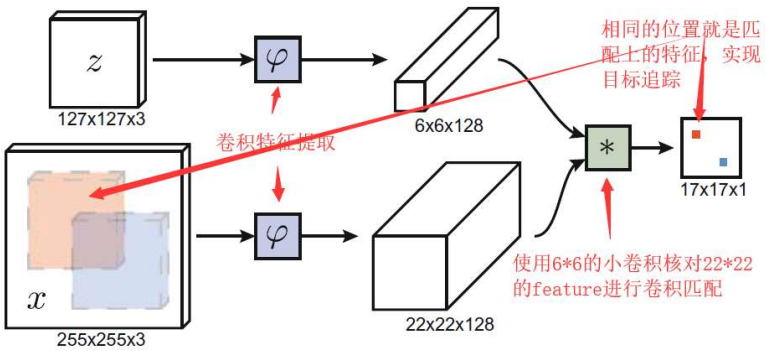
这其中还有几个不明白的地方:

- 四维的坐标映射到高维具体做了什么操作?
- 因为作者是一次性批量考虑所有的物体, 所以源代码中对于矩阵的操作非常复杂很难直接理解。还需要花更多时间理解作者的源代码才能上手这个 module

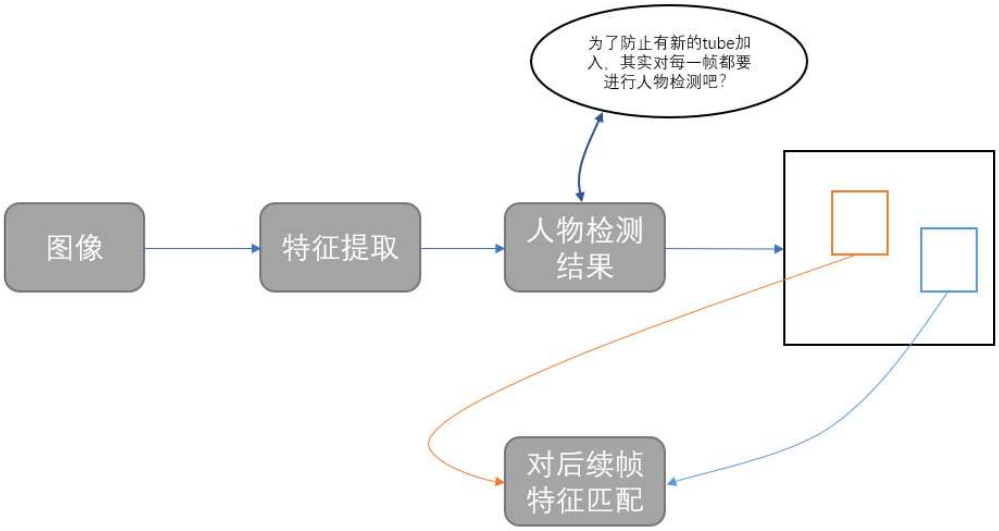
siamese network:

看了 Fully-convolutional Siamese networks

这个网络在这篇文章中是讲的目标追踪，其思想也很简单，就是给定一个模板之后，分别对模板和待搜索图中的特征进行提取，最后在大图中对这个模板的特征进行匹配，匹配上了的位置就是目标所在的区域。



基于该思想考虑做人物的追踪形成 tube，我认为是完全可以考虑的，而且每一个人物进行了特征提取之后的特征是全部被保留下来的，所以这些特征可以直接用于后续的行为分类。**但是我认为该方法的难点在于要如何确定追踪目标，因为如果追踪目标一旦确认错误这将直接导致后续全错。而且追踪目标的确认是要对每一帧都进行这样的工作，因为画面中可能会有旧的 tube 结束，也可能会有新的 tube 开始。**



基本流程如果表示成上面这样的话，其实既然对每一帧都进行了检测。。那为什么不直接根据前后检测的 IOU 就完成了 tube 的链接。。目前这种思路似乎并不可行。