本周完成了 loss 计算部分的代码工作，以及网络训练代码的组织和对之前仅用于验证的部分模块的重写，同时完全理解了目标检测过程中对于 loss 计算的思路，上上周虽然看懂了作者每一个函数的作用，但是对这些函数之间的配合关系以及作者为何要这样做表示非常懵。另一方面，最大的收获是又看了一遍深度学习课程中关于求导和优化这一部分，通过理解 pytorch 中计算图求导的思想，想清了作者的相同网络同时通过 6 帧的训练方法。磕磕绊绊到现在基本上对于目标检测的整个程序流程已经完全理解了。

首先对于网络 loss 的计算，其实理解了非常简单，其根本思想是找出正负样本，这里面的正样本就是和 ground truth 匹配上的，那么这个正样本是带有标签的，是有具体的动作在里面的；负样本就是背景，负样本是通过难例挖掘得到的，，而难例挖掘就是通过将计算 confidence loss 找到那些既本身就应该是背景然后网络的输出对这个 tube 还判断错误的，所以我们称之为难例。总的来说对于正负样本的选取根本思想都是为了让网络只关注于它最应该学习的东西。由于正负样本数量有限，所以每次更新可能都只会对网络中的部分参数进行更新。上述过程中这些对不同的 tube 的正负样本的选择操作并不会影响到原来网络的梯度！因为只是利用网络的输出进行一些额外的计算、选择和判断操作，这些计算的结果并不会加入到最终的 loss 的计算过程中，而计算最终的 loss 的过程其实是对网络的部分输出进行的一些常规操作。，这些操作逻辑上都很清晰，完全可以被计算图记录，很好地实现 backward。

在开会的时候听学长提起视频中动作可能只有部分的问题，在实现的过程中发现 ucf101 数据集中视频并不是每一帧都有 ground truth 的，在很多视频中，它关心的往往只有一个动作，比如投篮只有投篮这一小段时间，所以如果对一个视频进行训练时，假设随机取 6 帧，一定会取到部分有 ground truth 而部分缺失，或者 6 帧中都没有 ground truth，这将直接导致正样本的 tube 是残缺的或者直接数量为 0，如果严格按照 1:3 的比例，，那么负样本的数量也为 0 那还训个啥。。好在作者在论文中提到了这个问题，在训练的过程中要求每一帧都是有 ground truth 的，所以正样本残缺的问题可以解决了。

按照作者提供的代码，注意到作者为 6 帧的每一帧都建立了一个提取特征的 ssd 网络，刚开始我以为作者的训练也是这样 6 个并行的网络同时进行训练，但是对作者训练完成之后的参数进行了对比，发现作者提供的训练好的参数 6 个网络间完全一致！这说明作者的这 6 个在结构上并行的网络在 finetune 的时候并没有分别更新参数，而是完全参数共享的！刚开始我自己的网络结构也是仿照作者的结构写的，所以如果直接按照这种方式对网络进行训练，6 个网络在结构上并行进行参数更新一定会导致参数更新不一致的情况，并不能实现参数共享。为了解决这个问题，直接考虑改变这种并行的网络的结构，考虑将其简化为 1 个网络，然后按照顺序将这 6 帧依次送入同一个网络中，这样对这个网络实现了参数共享。但是引发了对梯度问题的思考，也就是说这样 6 帧多次送入网络的方法会对梯度的计算带来怎样的影响。这个问题有点像是多个 batch 的输入对网络梯度的影响。在论文 Bag of Tricks for Image Classification with Convolutional Neural Networks 中有提到输入的 batch 的增加会减少梯度的噪声因而可以适当增加学习率，所以总的来说应该是对网络的训练更加有益了，但是我对这个解释的方法存有一点点疑问，之前总觉得对同一个网络进行了多个输入不会造成梯度的累加吗，在教学视频中回顾，注意到了在损失函数的最后其实有对样本的个数做了平均，所以梯度其实被平均了，所以总的来说还是有益的！。

后面的工作中，对于验证部分的代码还需要重新改写，考虑 validation 的 dataset 可以写成每次读取一个完整的视频的所有帧，并且把视频帧一次性返回，这样 label 其实可以不必返回，返回空就好啦，然后计算完了之后直接把所有的数据分配到帧，这样每一个视频只保存一个缓存文件。这个可以手动交给多个单独的进程去完成，这样速度会快很多。

最后稍微记录一下 caffe 模型数据的提取问题，之前确实实现了 caffe 数据的提取，但是那是将 caffe 模型读取出来之后强行把模型中的参数一层一层扒出来用 numpy 数组保存起来然后再把它们复制到 pytorch 模型中，这个过程相当复杂。而且对于 caffemodel 中如果参数不全的话可能会没有办法 load，之前强行加载参数的程序在 load init weights 的时候就会出现加载错误的情况，最后不得已回到之前比赛中使用过的 caffemodel 转 pytorch 的方法。

这里面的关键问题是作者在 caffemodel 中重写了很多奇奇怪怪的层，所以这就直接导致原本的 caffe_pb2.py 这个文件失效，在查找其他问题的时候突然注意到其实这个文件每个版本的 caffe 都会自己生成的，所以我已经编译好的 caffe 版本可以直接找到对应的 caffe_pb2.py 这个文件，这就基本上解决了问题，成功把作者提供的部分 init weights 从中提取了出来。并为此详细写了一个博客：

https://blog.csdn.net/yuanqia4079/article/details/84885724