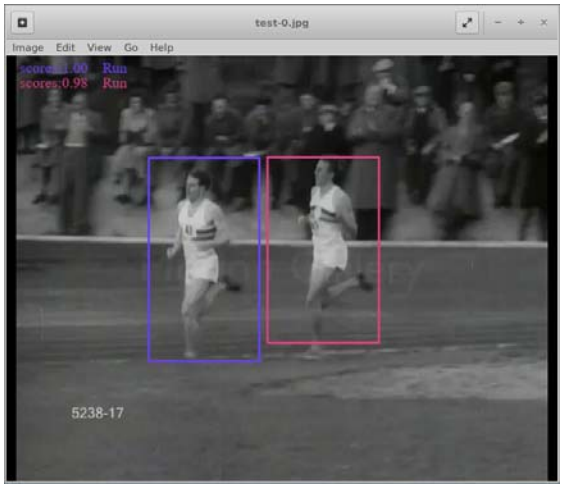
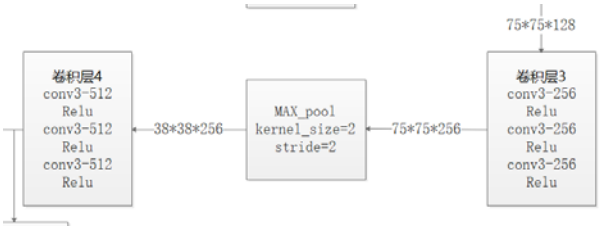
本周找齐了上周代码中的已知的所有 bug，使用作者的参数，自己写的网络和后处理已经复现了作者在论文中的效果：



仔细查看并且分析了一下作者的 caffe 源代码中的前向传播部分的代码，基本搞清楚了前向过程中计算 loss 的全部过程，但是有一个问题还很模糊，就是反向求导的过程什么样的需要手动完成？什么样的可以自动完成？如何让我自己的代码在 pytorch 中是否需要手动实现 backward？

上周的程序中找出来的问题简单记录一下，都是一些比较难发现的低级错误，只要注意到这些地方可以较快避免浪费时间：

- OpenCV 读取图片的格式是 BGR，但是 python 的图像库 PIL 读取到的图片是 RGB，所以使用作者的网络参数要注意调整通道位置
- 如下图所示，卷积层 3 的输出和卷积层 4 的输出中间的 maxpool 层，如果不进行处理，75*75 的 feature map 池化后最后一列会被丢弃，这样只能形成 37*37 的 feature map，在 pytorch 中，在 pool 层中参数 ceil_mode 设置为 True，可以保留最后一列，生成预期的 38*38 的结果。



- Caffe 中提到的对 conv4 有一个 normalize 层，这个层在通道间进行了 normalize 之后，还要乘上一个系数 scale，作者的 scale 参数是可训练的，所以在 pytorch 中可以继承 nn.module 重新定义一个 normalize 层，实现参数的加载和训练。
- 在最后计算 nms 的时候要注意先将超出图像区域的多余部分去掉，否则计算的 iou 会包含图像之外的，这显然没有意义
- 之前的结果不正确最为严重的一个问题是生成 prior tubes 的时候，feature map 的展开是按照行的方式展开，但是生成的 prior tubes 是按照列展开的，这也是之前代码中对结果影响最大的部分。

作者对于一段视频中的每一帧，都以该帧为起点，往后取 6 帧进行一次计算，这其中显然会有帧重复进行计算，重复最多的计算了 6 次，那么同一帧在多次计算中对图中的 ground truth

可能会有不同，那么可以考虑用平均的思想，于是有了，生成 tubes 的思想：

- 在这之前，以每一帧为起点，往后连续读取 6 帧，直到视频结束。每一段都经过 nms 后取出 top10 的 tubes。第一帧到第六帧，这里称为第一段，以取出来的 10 个 tubes 作为起点。
- 取出第二帧到第七帧的 tubes，这里称为第二段，然后依次考虑第一段中所有的 tubes，即取出第一段中第一个 tube，依次和第二段中所有的 tubes 计算 ious，挑出第二段中计算出来的 ious 大于 0.2 的 tubes，取出得分最高的 tube 和第一段中的第一个 tube 组成一个新的 tube。然后考虑第一段中第二个 tube 和第二段中剩下的 tubes 之间的关系。（如果某一个 tubes 在后面的帧中连续 5 帧都没有找到 ious 大于 0.2 的新 tubes，那么就认为这个 tubes 结束了。）
- 依次对每一个类都进行上一步操作。

下面理解了作者源代码中计算 loss 的整个过程，下面的各个函数的笔记是在看代码过程中对过程的提要，提醒写代码的时候要关注的一些细节。

整个过程中的核心思想有以下几点：

- 首先利用 ground truth 和 prior tubes 进行双向配对，找出正样本。
- 然后单独计算 conf loss，找出负样本。
- 最后把正负样本的预测数据和 ground truth 数据计算 loss 即可。

ACTMatchTube：

1、计算当前所有的 prior tubes 和 ground truth 的 iou，每一个 prior tube 都记录自己和每一个 ground truth 最大的 overlap

2、然后为每一个 ground truth 找到一个最匹配的 prior tube。

3、为剩下的 prior tubes 寻找到一个最匹配的 ground truth，要求 overlap>=overlap_threshold 即可配对成功

ACTComputeConfLoss：

1、依次遍历每一个 tube，对于每一个 tube 都要看看它有没有和 ground truth 匹配上，如果匹配上了 label 就置为相应的值，如果没有匹配上 label 就是 0。

2、每一个 tube 的 loss 的单独的计算方法：

$$\text{loss} = -\log\left(\frac{e^{c_{label}-c_{max}}}{\sum e^{c_i-c_{max}}}\right)$$

上式中，下标 i 代表每一个类，c_{label}是正确的 label 位置处 conf_conv 的输出，c_{max}是这一个 tube 的 conf_conv 在 11 个类上的输出值中的最大值。可以看到该 loss 的计算方法和传统的 softmax 后的计算有些许不同，它在这里将所有的 conf_conv 输出全部平移到了小于 0 的部分。

这个 conf loss 的计算结果仅仅只是用于难例挖掘。

IsEligibleMining：

如果 match_overlap<neg_overlap，那么这个函数就会返回 true，也就是说 overlap 太小了就会认定为负样本。用于比较的每一个 tube 的 match_overlap 都是这个 tube 和所有的 ground truth 进行了比较后的最大的那个 overlap。

ACTMineHardExamples：
首先计算 conf loss
找到所有的 overlap 小于 0.5 的 tube，作为负样本候选。
控制负样本的数量不超过正样本的三倍，综合第一步就可以得到负样本的数量。
按照 loss（conf loss）从高到低对负样本进行排序。
选出 loss 最高的负样本数量
最后选出的负样本中只要不是曾经和 ground truth 配对成功的就可以。

ACTEncodeLocPrediction：
依次取出前面配对成功的 prior tubes 和 ground truth，利用 prior tube 的四个参数，和所对应的 ground truth 的四个参数进行编码，得到论文中的（3）式。使用 encode 的结果可以计算 reg loss。

$$\mathcal{L}_{\text{reg}} = \frac{1}{K} \sum_{i \in \mathcal{P}} \sum_{c \in \{x,y,w,h\}} x_{ij}^{y} \sum_{k=1}^{K} \text{SmoothL1}\left(\hat{r}_i^{c_k} - \mathfrak{g}_{ij}^{c_k}\right) ,$$

$$\text{with} \quad \mathfrak{g}_{ij}^{x_k} = \frac{g_j^{x_k} - a_i^{x_k}}{a_i^{w_k}} \qquad \mathfrak{g}_{ij}^{y_k} = \frac{g_j^{y_k} - a_i^{y_k}}{a_i^{h_k}} , \tag{3}$$

$$\mathfrak{g}_{ij}^{w_k} = \log\left(\frac{g_j^{w_k}}{a_i^{w_k}}\right) \qquad \mathfrak{g}_{ij}^{h_k} = \log\left(\frac{g_j^{h_k}}{a_i^{h_k}}\right) .$$

将 prior tubes 和 ground truth 进行 decode 操作后的结果和这一个 prior tube 所对应的 conv 输出要一同返回，用于计算 reg loss。

ACTEncodeConfPrediction：
先取出 conf conv 在正样本上对所有类的打分以及正样本的正确 label
然后取出负样本的 conf conv 的所有打分，负样本的标签为 background。
返回以上的两个结果用于计算 con floss