





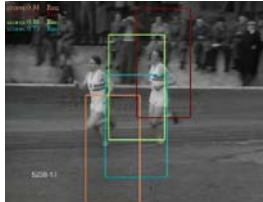
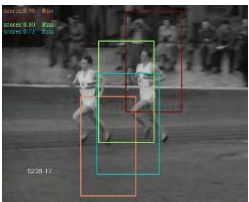
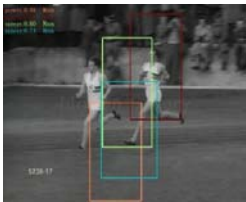


本周主要对作者提供的 ACT-Detector 的源代码的测试思路进行了细节理解和考虑，利用作者提供的训练参数，对网络的输出部分（featuremap 上数据具体结构、后处理及 nms）编程用 python+pytorch 初步实现了对连续 6 帧的处理和输出，效果如下，但是对于一些图的识别还没能达到作者提供的水平，特别像下面的第三个，作者在论文中提供的效果和这个就不是一个层次的。。但是现在还没有找到问题所在。应该是一些细节地方还没有做到位，还需要花时间去阅读作者的源码：

			Biking 不知道为什么多出来一个框。。
			Diving 这个的 confidence 只有 0.05
			Run 这个是找的作者论文上的，但是结果的差距有点大。。

核心思路：

在连续的 6 帧中，每一帧的 feature map 上的每一个点假设有 n 个 prior box，那么连续的 6 帧中同一个中心位置，长宽都相同的 6 个 prior box 组成一个 prior tube，这也就对应着作者在文中提到的假设短时间内运动的目标不会走出这个 tube。

然后 loc 卷积层会为每一个 prior box 计算出一组回归参数，利用这些回归参数对每一个 prior box 进行微调，这样原来的 prior tubes 就变成了新的 tubes。

然后将这些 tubes 在每一个分类中的打分取出来，假设我产生了 8732 个 tubes，那么在类别 1 就有 8732 个 conf 的输出，（在做这一步之前一定要对这个 conf 进行 softmax），然后将这 8732 个概率值和 confidence_threshold 比较，小于它的就不要了，然后把满足 confidence_threshold 条件的 tubes 进行 nms，即从前面选出的集合（称为集合 M）中先取出得分最高的 tube 放入集合 N，然后按照得分的从高到低依次取出集合 M 中剩下的，如果发现和集合 N 中的任意一个 tubes 的 IOU 值大于了 nms_threshold 就丢弃这个 tube，如果发现 IOU 都小于 nms_threshold，那么就将这个 tube 放入集合 N 中，直到集合 M 为空。

对每一个类（背景类不需要）进行上述操作，完成之后将所有的类的 tubes 合并，根据其打分的高低进行排序，取前 topk 个。

下面记录一些实现过程中的细节：

1、Conv4_3 的 normalize layer:

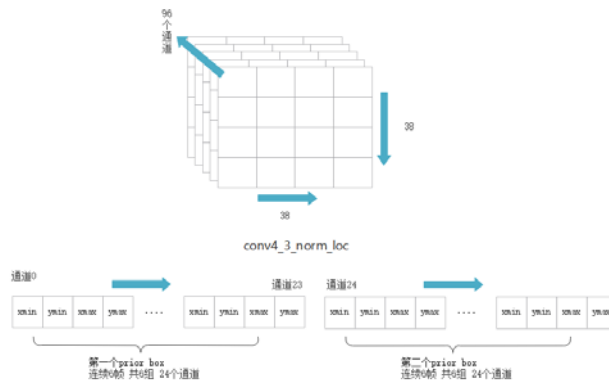
在 prototxt 中，该层的参数 across_spatial 为 false，所以对形状为 $m \times n \times c$ 的 feature map 进行操作时，其实是在 $m \times n$ 个 c 维向量上进行的操作，也就是在通道间进行的 normalize。而当 across_spatial 为 true 时才是针对一个 feature map 的 norm 操作。这里的操作在 pytorch 中可以直接用 F.normalize 来实现。

另外 scale_filler 参数，在 prototxt 中其给定值为 20，也就是说，在经过 normalize 之后

还要对结果乘上 20 用来放大, 但是 channel_shared 参数又是 false, 查看了 caffe 的源代码, 这又让每一个通道的 scale 是不共享的。而从模型中获取的这一层参数也对应于每一个通道的值是不同的, 但基本上都在 16.38 附近。在测试中也发现这个 scale 是否 channel shared 对结果的影响不大、

2、feature map 的输出和 tubes 的对应关系:

想要用作者的模型参数来直接实现效果, 拿出 feature map 中的数据来使用时必须考虑其原来的定义。下面拿 conv4_3_norm 的 localization 卷积层的输出来理解: 在这一层中, 有 4 个 priorbox, 4 个坐标以及 6 帧, 所以这一层的输出有 $4 \times 4 \times 6 = 96$ 个通道, 通过阅读作者提供的 caffe 源代码, 知道这 96 个通道的定义如下图:



类似的, 在 conf 层中, 则是以每一个 prior tube 为一组 (一组为 num_class + 1 个, 因为还有背景), 为每一个 prior tube 在每一个分类上进行打分。假设该 conf 卷积层的输出有 100 个通道, 其中有效分类为 24 类, 那么加上背景就是 25 个类, 那么这 100 个通道的 0-25 就是对应的前置 tube1, 25-50 就是对应前置 tube2。

3、prior box 顺序的对应关系:

Feature map 上的每一个点都至少有 4 个 prior box, 那么这 4 个 prior boxes 的顺序究竟如何?

```
layer {
  name: "conv4_3_norm_concat_mbox_priorbox"
  type: "PriorBox"
  bottom: "conv4_3_norm_concat"
  bottom: "data_stream0"
  top: "conv4_3_norm_concat_mbox_priorbox"
  prior_box_param {
    min_size: 30.0
    max_size: 60.0
    aspect_ratio: 2
    flip: true
    clip: false
    variance: 0.1
    variance: 0.1
    variance: 0.2
    variance: 0.2
    step: 8
    offset: 0.5
  }
}
```

Prototxt 对 prior box 层的定义, 里面传入了 min_size,max_size 以及一个 aspect_ratio, 那么根据 SSD 论文中提到的, 这三个参数一共可以产生 4 个 prior box, 其中第一个 prior box 为 $\min_size \times \min_size$, 继续上面的例子, 则这个 box 的参数对应 conf 卷积层输出的前 25 个通道, 然后是 $\sqrt{\min_size \times \max_size} \times \sqrt{\min_size \times \max_size}$, 这个 prior box 对应 conf 卷积层输出的 25-50 通道, 然后分别是 $width = \min_size \times \sqrt{2}$, $height = \min_size \times \sqrt{1/2}$ 和 $width = \min_size \times \sqrt{1/2}$, $height = \min_size \times \sqrt{2}$ 两个 prior box。

4、Tubes 的 decode 的方法:

得到 tubes 中每一个 prior box 的回归参数之后要对文中提到的下式进行反解得到回归

后的 priorbox，但是作者 prior box 层中还传入了 variance 参数，这个参数总是在 feature map 的输出前面乘上对估计值进行修正。

$$\text{with } g_{ij}^{x_k} = \frac{g_j^{x_k} - a_i^{x_k}}{a_i^{w_k}}, \quad g_{ij}^{y_k} = \frac{g_j^{y_k} - a_i^{y_k}}{a_i^{h_k}},$$

$$g_{ij}^{w_k} = \log\left(\frac{g_j^{w_k}}{a_i^{w_k}}\right), \quad g_{ij}^{h_k} = \log\left(\frac{g_j^{h_k}}{a_i^{h_k}}\right).$$

```
new_center_x = var[0] * feature_flat[
    4 * tubes.sequence_length * k + 4 * j + 0, i] * width + center_x
new_center_y = var[1] * feature_flat[
    4 * tubes.sequence_length * k + 4 * j + 1, i] * height + center_y
new_width = np.exp(var[2] * feature_flat[4 * tubes.sequence_length * k + 4 * j + 2, i]) * width
new_height = np.exp(var[3] * feature_flat[4 * tubes.sequence_length * k + 4 * j + 3, i]) * height
current_tube[j, 0] = new_center_x - new_width / 2.0 # x_min
current_tube[j, 1] = new_center_y - new_height / 2.0 # y_min
current_tube[j, 2] = new_center_x + new_width / 2.0 # x_max
current_tube[j, 3] = new_center_y + new_height / 2.0 # y_max
```

5、Tubes 的 IOU 的计算方法

在进行 nms 时需要考虑 IOU，这里的 IOU 不再是针对 prior box，而是针对整个 tube，对于整个 tube 的 IOU 是取一个 tube 中所有回归后的 box 计算每一个 box 在对应帧上的 IOU，计算每一个 tube 的 6 帧的平均 IOU 作为这个 tube 的 IOU。