

一、起源

非极大值抑制 (Non-Maximum Suppression, 以下简称NMS算法) 的思想是搜索局部极大值, 抑制非极大值元素。针对不同的应用场景和检测算法, 由于矩形框的表征方式不同, NMS算法具有各种变体, 本文针对NMS及其各种变体 (主要是本文检测相关) 进行介绍。

二、经典NMS

1. 设定目标框的置信度阈值，常用的阈值是0.5左右
2. 根据置信度降序排列候选框列表
3. 选取置信度最高的框A添加到输出列表，并将其从候选框列表中删除
4. 计算A与候选框列表中的所有框的IoU值，删除大于阈值的候选框
5. 重复上述过程，直到候选框列表为空，返回输出列表


$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$


图1

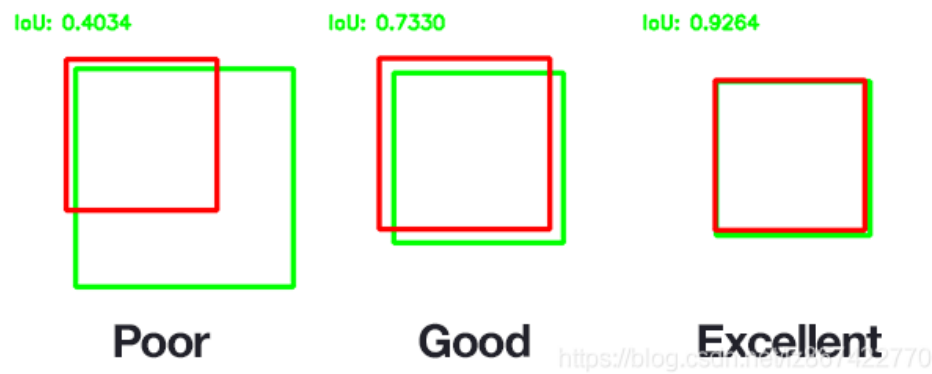


图2

下面，通过一个具体例子来说明经典NMS究竟做了什么。图3的左图是包含一个检测目标（王闹海）的实例图片。其中的绿色矩形框代表了经过目标检测算法后，生成的大量的带置信度的Bounding box，矩形框左下角的浮点数即代表该Bounding box的置信度。在这里，使用Python对经典NMS算法实现，并应用到该实例中去。当NMS的阈值设为0.2时，最后的效果如图3中右图所示。

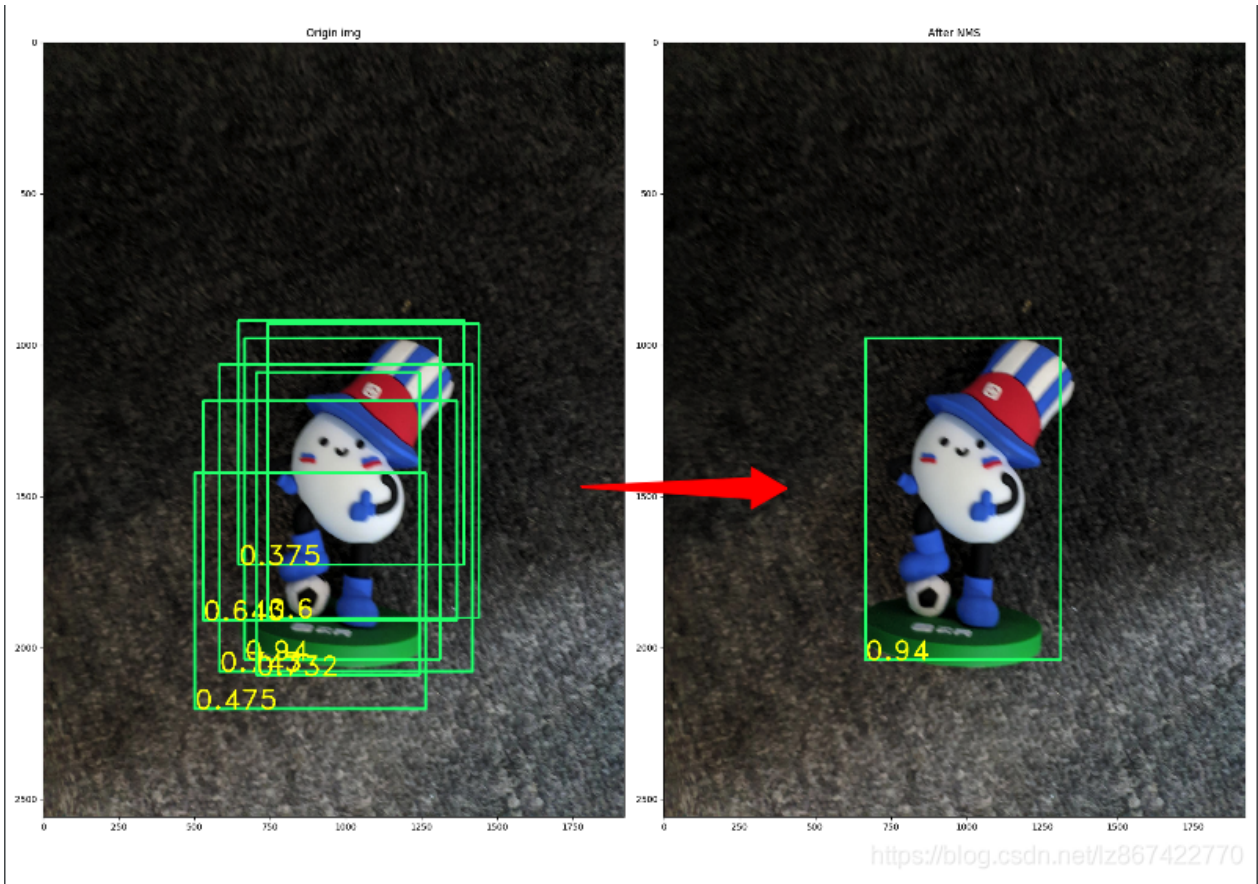


图3

Python代码如下：

本文所有代码和数据的github地址：<https://github.com/lzneu/letGo/tree/master/nms>

```
1. def nms(bounding_boxes, Nt):
2.     if len(bounding_boxes) == 0:
3.         return [], []
4.     bboxes = np.array(bounding_boxes)
5.
6.     # 计算 n 个候选框的面积大小
7.     x1 = bboxes[:, 0]
8.     y1 = bboxes[:, 1]
```

```

9.     x2 = bboxes[:, 2]
10.    y2 = bboxes[:, 3]
11.    scores = bboxes[:, 4]
12.    areas = (x2 - x1 + 1) * (y2 - y1 + 1)
13.
14.    # 对置信度进行排序, 获取排序后的下标序号, argsort 默认从小到大排序
15.    order = np.argsort(scores)
16.
17.    picked_boxes = [] # 返回值
18.    while order.size > 0:
19.        # 将当前置信度最大的框加入返回值列表中
20.        index = order[-1]
21.        picked_boxes.append(bounding_boxes[index])
22.
23.        # 获取当前置信度最大的候选框与其他任意候选框的相交面积
24.        x11 = np.maximum(x1[index], x1[order[:-1]])
25.        y11 = np.maximum(y1[index], y1[order[:-1]])
26.        x22 = np.minimum(x2[index], x2[order[:-1]])
27.        y22 = np.minimum(y2[index], y2[order[:-1]])
28.        w = np.maximum(0.0, x22 - x11 + 1)
29.        h = np.maximum(0.0, y22 - y11 + 1)
30.        intersection = w * h
31.
32.        # 利用相交的面积和两个框自身的面积计算框的交并比, 将交并比大于阈值的框删除
33.        ious = intersection / (areas[index] + areas[order[:-1]] - intersection)
34.        left = np.where(ious < Nt)
35.        order = order[left]
36.    return picked_boxes

```



三、soft-NMS

经典NMS是为了去除重复的预测框, 这种算法在图片中只有单个物体被检测的情况下具有很好的效果, 如上图所示, “腰子”王闹海被成功的检测出来。然而, 经典NMS算法存在着一些问题: 对于重叠物体无法很好的检测。当图像中存在两个重叠度很高的物体时, 经典NMS会过滤掉其中置信度较低的一个。如下图所示, 图中存在两个王闹海, 经典NMS过滤后的结果如下图4所示:

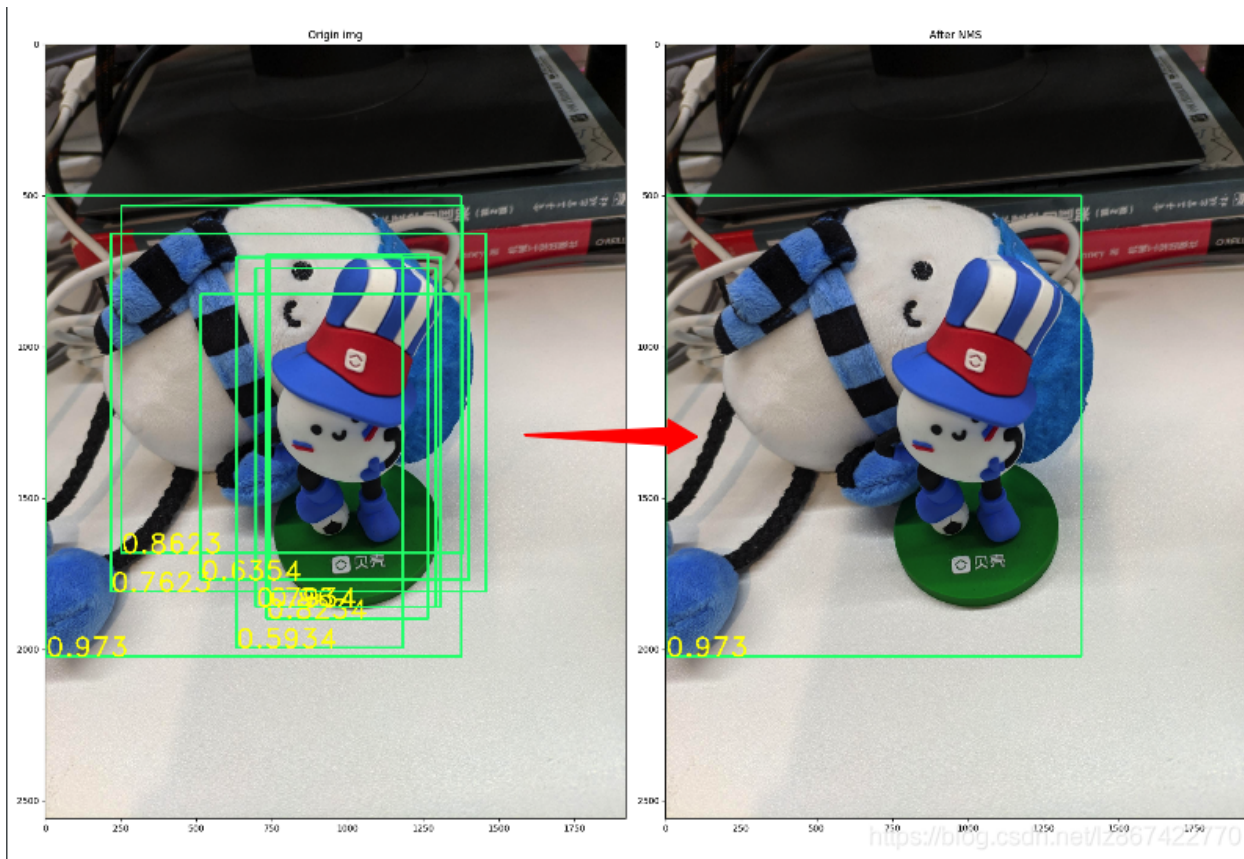


图4

而我们期望的结果是两个腰子都被算法成功检测出来。

为了解决这类问题, Jan Hosang,等人提出了Soft-NMS算法。Soft-NMS的算法伪代码如图5所示。其中红色框为经典NMS的步骤, 而绿色框中的内容为Soft-NMS改进的步骤。可以看出, 相对于经典NMS算法, Soft-NMS仅仅修改了一行代码。当选取了最大置信度的Bounding box之后, 计算其余每个Bounding box与Bounding box的IoU值, 经典NMS算法的做法是直接删除IoU大于阈值的Bounding box; 而Soft-NMS则是使用一个基于IoU的衰减函数, 降低IoU大于阈值 N_t 的Bounding box的置信度, IoU越大, 衰减程度越大。

换一个角度来理解, 经典NMS同样可以用一个基于IoU的衰减函数来表示, 只是这个衰减函数是个0-1函数, 当IoU大于阈值 N_t , 直接降低该Bounding box的置信度到0。关于经典NMS和Soft-NMS的衰减函数区别, 可以通过如下的公式来表示[1]:

- 经典NMS:

$$s_i = \begin{cases} s_i, & \text{iou}(\mathcal{M}, b_i) < N_t \\ 0, & \text{iou}(\mathcal{M}, b_i) \geq N_t \end{cases},$$

- Soft-NMS-线性:

$$s_i = \begin{cases} s_i, & \text{iou}(\mathcal{M}, b_i) < N_t \\ s_i(1 - \text{iou}(\mathcal{M}, b_i)), & \text{iou}(\mathcal{M}, b_i) \geq N_t \end{cases},$$

- Soft-NMS-高斯:

$$s_i = s_i e^{-\frac{\text{iou}(\mathcal{M}, b_i)^2}{\sigma}}, \forall b_i \notin \mathcal{D}$$

Input : $\mathcal{B} = \{b_1, \dots, b_N\}$, $\mathcal{S} = \{s_1, \dots, s_N\}$, N_t
 \mathcal{B} is the list of initial detection boxes
 \mathcal{S} contains corresponding detection scores
 N_t is the NMS threshold

begin

$\mathcal{D} \leftarrow \{\}$

while $\mathcal{B} \neq \text{empty}$ **do**

$m \leftarrow \text{argmax } \mathcal{S}$

$\mathcal{M} \leftarrow b_m$

$\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{M}; \mathcal{B} \leftarrow \mathcal{B} - \mathcal{M}$

for b_i **in** \mathcal{B} **do**

if $\text{iou}(\mathcal{M}, b_i) \geq N_t$ **then**

$\mathcal{B} \leftarrow \mathcal{B} - b_i; \mathcal{S} \leftarrow \mathcal{S} - s_i$

end

NMS

$s_i \leftarrow s_i f(\text{iou}(\mathcal{M}, b_i))$

Soft-NMS

end

end

return \mathcal{D}, \mathcal{S}

end

<https://blog.csdn.net/flz867422770>

图5

接下来，继续使用图4的实例来验证Soft-NMS的效果，使用Python对Soft-NMS算法的实现如下

Python代码：

```
1. def soft_nms(bboxes, Nt=0.3, sigma2=0.5, score_thresh=0.3, method=2):
2.     # 在 bboxes 之后添加对于的下标[0, 1, 2...], 最终 bboxes 的 shape 为 [n, 5], 前四个为坐标, 后一个为下标
3.     res_bboxes = deepcopy(bboxes)
4.     N = bboxes.shape[0] # 总的 box 的数量
5.     indexes = np.array([np.arange(N)]) # 下标: 0, 1, 2, ..., n-1
6.     bboxes = np.concatenate((bboxes, indexes.T), axis=1) # concatenate 之后, bboxes 的操作不会对外部变量产生影响
7.     # 计算每个 box 的面积
8.     x1 = bboxes[:, 0]
9.     y1 = bboxes[:, 1]
10.    x2 = bboxes[:, 2]
```

```

11. y2 = bboxes[:, 3]
12. scores = bboxes[:, 4]
13. areas = (x2 - x1 + 1) * (y2 - y1 + 1)
14.
15. for i in range(N):
16.     # 找出 i 后面的最大 score 及其下标
17.     pos = i + 1
18.     if i != N - 1:
19.         maxscore = np.max(scores[pos:], axis=0)
20.         maxpos = np.argmax(scores[pos:], axis=0)
21.     else:
22.         maxscore = scores[-1]
23.         maxpos = 0
24.     # 如果当前 i 的得分小于后面的最大 score, 则与之交换, 确保 i 上的 score 最大
25.     if scores[i] < maxscore:
26.         bboxes[[i, maxpos + i + 1]] = bboxes[[maxpos + i + 1, i]]
27.         scores[[i, maxpos + i + 1]] = scores[[maxpos + i + 1, i]]
28.         areas[[i, maxpos + i + 1]] = areas[[maxpos + i + 1, i]]
29.     # IoU calculate
30.     xx1 = np.maximum(bboxes[i, 0], bboxes[pos, 0])
31.     yy1 = np.maximum(bboxes[i, 1], bboxes[pos, 1])
32.     xx2 = np.minimum(bboxes[i, 2], bboxes[pos, 2])
33.     yy2 = np.minimum(bboxes[i, 3], bboxes[pos, 3])
34.     w = np.maximum(0.0, xx2 - xx1 + 1)
35.     h = np.maximum(0.0, yy2 - yy1 + 1)
36.     intersection = w * h
37.     iou = intersection / (areas[i] + areas[pos] - intersection)
38.     # Three methods: 1.linear 2.gaussian 3.original NMS
39.     if method == 1: # linear
40.         weight = np.ones(iou.shape)
41.         weight[iou > Nt] = weight[iou > Nt] - iou[iou > Nt]
42.     elif method == 2: # gaussian
43.         weight = np.exp(-(iou * iou) / sigma2)
44.     else: # original NMS
45.         weight = np.ones(iou.shape)
46.         weight[iou > Nt] = 0
47.     scores[pos:] = weight * scores[pos:]
48. # select the boxes and keep the corresponding indexes
49. inds = bboxes[:, 5][scores > score_thresh]
50. keep = inds.astype(int)
51. return res_bboxes[keep]

```

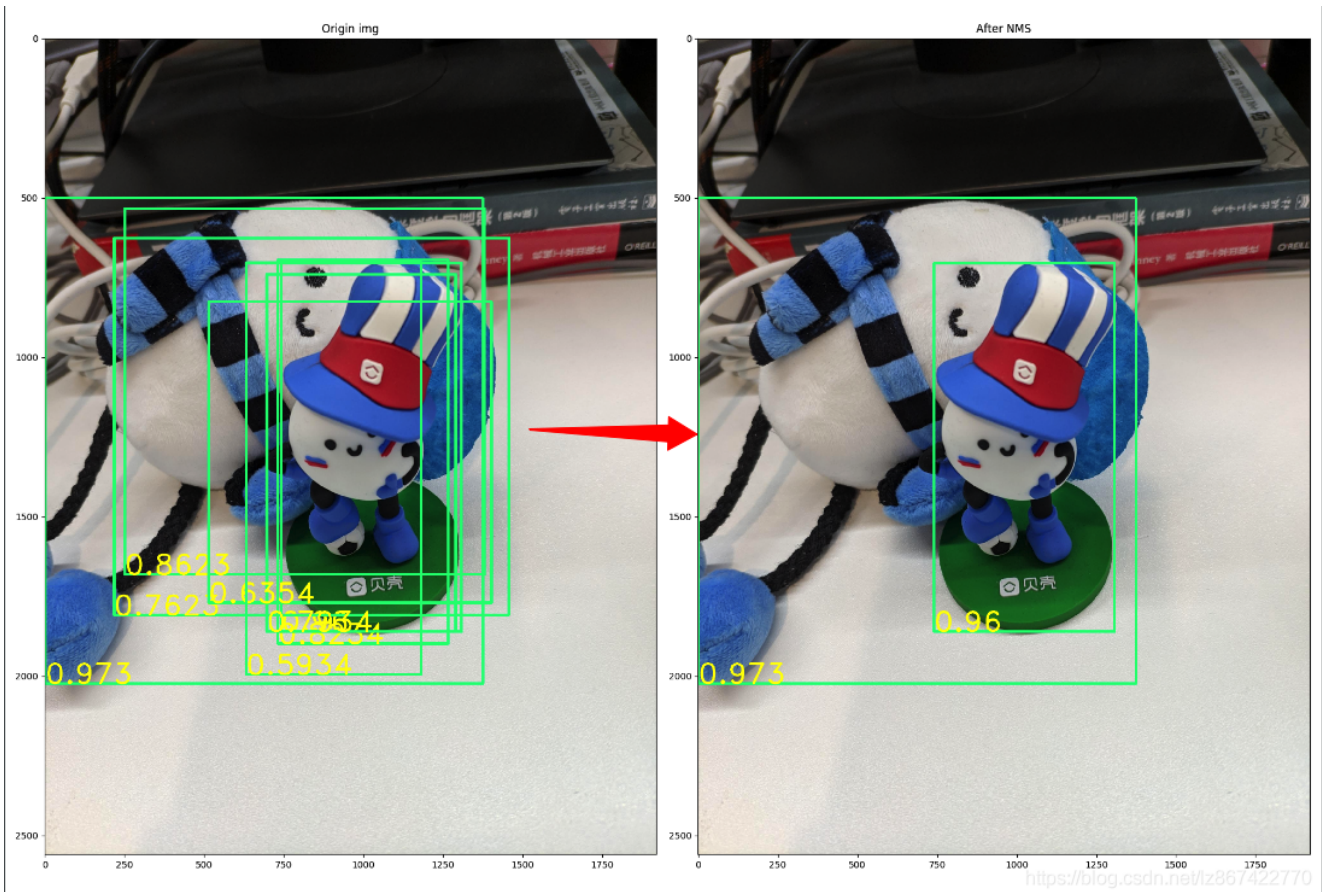


图6

四、Locality-Aware NMS

以上的NMS或相关算法均为针对目标检测问题而提出的，而文本检测作为目标检测的一个特殊领域，具有其特殊之处，比如文本检测中面临的成千上万的候选几何体，如果使用计算复杂度为 $O(n^2)$ 的经典NMS类算法，由于n的数量过大，这是不可接受的。为此旷世公司于2017在EAST（一种非常好用的One-stage的文本检测算法，敬请关注交易只能团队下期文章！）论文[2]中提出了Locality-Aware NMS算法。

Locality-Aware NMS算法考虑了文本检测任务中**临近像素高度相关**的特征，提出了一种基于行的几何体合并方法，它的步骤主要是两个：

1. 合并同一**行区域**中的几何体
2. 对合并后的Bounding box集合进行标准的NMS操作

Locality-Aware NMS提升了算法的时间复杂度，在最优情况下可以提升至 $O(n)$ 。即使是最坏情况，也不会比经典NMS的计算量更大。

Python代码如下：

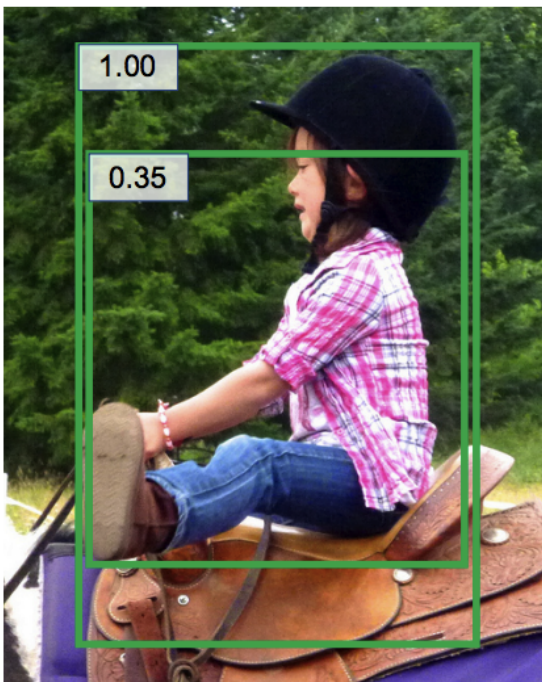
```
1. import numpy as np
2. from shapely.geometry import Polygon
3.
4. def intersection(g, p):
5.     # 取g,p中的几何体信息组成多边形
6.     g = Polygon(g[:8].reshape((4, 2)))
7.     p = Polygon(p[:8].reshape((4, 2)))
8.     # 判断g,p是否为有效的多边形几何体
9.     if not g.is_valid or not p.is_valid:
10.         return 0
11.     # 取两个几何体的交集和并集
12.     inter = Polygon(g).intersection(Polygon(p)).area
13.     union = g.area + p.area - inter
14.     if union == 0:
15.         return 0
16.     else:
17.         return inter / union
18.
19. def weighted_merge(g, p):
20.     # 取g,p两个几何体的加权（权重根据对应的检测得分计算得到）
21.     g[:8] = (g[:8] * g[8] + p[:8] * p[8]) / (g[8] + p[8])
22.     # 合并后的几何体的得分为两个几何体得分的总和
23.     g[8] = (g[8] + p[8])
24.     return g
25.
26. def standard_nms(S, thres):
27.     # 标准NMS
28.     order = np.argsort(S[:, 8])[::-1]
29.     keep = []
30.     while order.size > 0:
31.         i = order[0]
32.         keep.append(i)
33.         ovr = np.array([intersection(S[i], S[t]) for t in order[1:]])
34.         inds = np.where(ovr <= thres)[0]
35.         order = order[inds + 1]
36.     return S[keep]
37.
38.
39. def nms_locality(polys, thres=0.3):
40.     '''
41.     locality aware nms of EAST
42.     :param polys: a N*9 numpy array. first 8 coordinates, then prob
43.     :return: boxes after nms
44.     '''
45.     S = [] # 合并后的几何体集合
46.     p = None # 合并后的几何体
47.     for g in polys:
48.         if p is not None and intersection(g, p) > thres: # 若两个几何体的相交面积大于指定的阈值，则进行合并
49.             p = weighted_merge(g, p)
50.         else: # 反之，则保留当前的几何体
51.             if p is not None:
52.                 S.append(p)
53.             p = g
54.     if p is not None:
55.         S.append(p)
56.     if len(S) == 0:
57.         return np.array([])
58.     return standard_nms(np.array(S), thres)
```



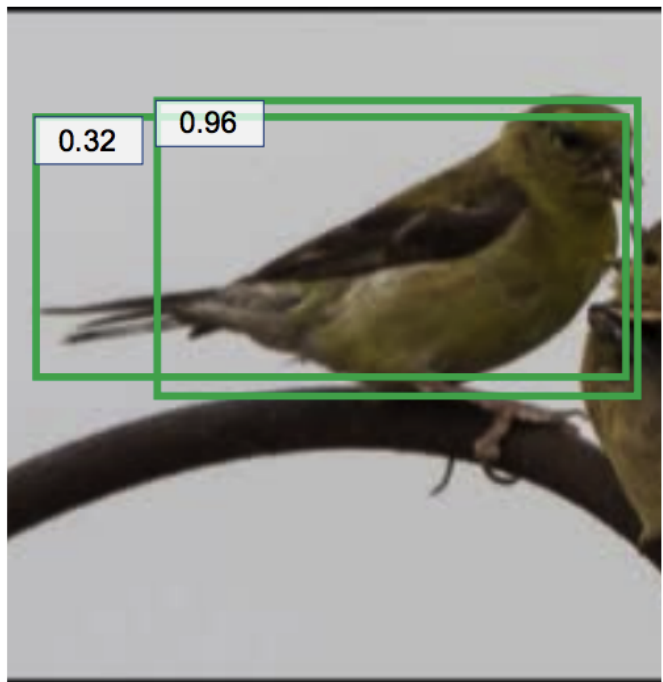
五、Softer-NMS

以上的NMS算法，无论是经典NMS还是Soft-NMS算法，都是在一种假设前提下：**置信度最高的Bounding box就是目标的候选位置最精确的物体位置**。但是事实上，这个假设可能并不成立，或者说并不那么精确。针对这个问题，来自卡内基梅隆大学与旷视科技的研究人员在文中提出了一种新的非极大抑制算法Softer-NMS，显著改进了目标检测的定位精度，代码已经开源，目前Github上的Star已超100，可谓短短两天已经引起了不小的关注。作者关注了两种目前NMS会出现的问题：

1. 所有的候选Bounding box都够精确，无法确定该选择哪一个，见图7a
2. 具有高置信度的Bounding box不够精确，见图7b



(a)



(b)

<https://blog.csdn.net/iz867422770>

图7

Softer-NMS正是为了这两个问题而设计的。关于Softer-NMS的相关解析，敬请关注交易智能团队后续分享。

参考文献：

- [1] Bodla N , Singh B , Chellappa R , et al. Improving Object Detection With One Line of Code[J]. 2017.
- [2] Zhou X , Yao C , Wen H , et al. EAST: An Efficient and Accurate Scene Text Detector[J]. 2017.
- [3] He Y , Zhu C , Wang J , et al. Bounding Box Regression with Uncertainty for Accurate Object Detection[J]. 2018.
- [4] 非极大值抑制 (Non-Maximum Suppression, NMS) – 康行天下 – 博客园
- [5] 非极大值抑制(Non-Maximum Suppression) – 云+社区 – 腾讯云
- [6] Intersection over Union (IoU) for object detection – PylmageSearch
- [7] NMS 算法源码实现 | 从零开始的BLOG
- [8] <https://www.52cv.net/?p=1434>