DBNet GT 构造过程的问题和对应解决方法

感觉自己好久没有写干货了，刚好最近工作中一直在折腾DBNet+，今天就来简单聊聊DBNET

本篇文章会介绍一个在DBNet 使用过程中广为存在的问题，并且会对这个问题给出一些工程上的优化方法

对DBNet 还不是很了解的，可以看：

https://arxiv.org/pdf/1911.08947.pdf
arxiv.org/pdf/1911.08947.pdf

Part 1 DBNet 的GT 构造+方法和问题

1.1 DBNet 的问题

首先来回顾一下DBNet 当中概率图GT的构造过程和推理流程

按照原始DBNet 的论文，在构造概率图时，会使用如下的公式构造一个收缩后的GT：

$$D = \frac{A(1-r^2)}{L}$$

对应的代码如下（注意这里会涉及到一个shrink ratio+ ）

```
instance = poly[0].reshape(-1, 2).astype(np.int32)
area = plg(instance).area
peri = cv2.arcLength(instance, True)
distance = min( int(area * (1 - shrink_ratio * shrink_ratio) / (peri + 0.001) +0.!
pco = pyclipper.PyclipperOffset()
pco.AddPath(instance, pyclipper.JT_ROUND,  pyclipper.ET_CLOSEDPOLYGON)
shrunk = np.array(pco.Execute(-distance))
```

因此，构造的GT 相比于原先的GT 是个收缩一圈的，以下图为例，如果不看这张图对应着原始GT在图中的位置：

（明眼人可能一猜就能猜出来这是个什么票据）

那么，其对应的GT 就应该是如下这个样子：

到目前为止，看上去不错对吧?

然后，按照DBNet 的原始论文，在模型获得了如上图所示的预测结果以后，可以按照如下的公式，对DBNet 的预测结果进行反向的膨胀：

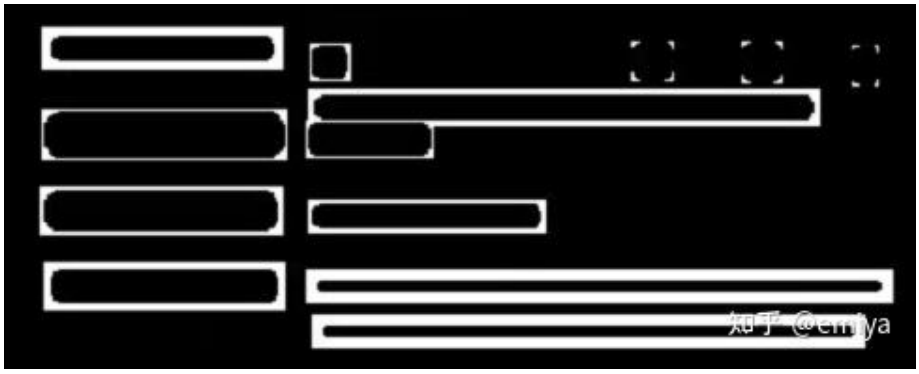
$$D' = \frac{A' \times r'}{L'}$$

对应的代码如下(注意这里会涉及到一个 unclip = 1.5)：

```
# mmocr 当中将dilate 的部分称之为 unclip ，下面统一我们也这么称呼

def unclip(box, unclip_ratio=1.5):
    poly = plg(box)
    distance = poly.area * unclip_ratio / poly.length
    offset = pyclipper.PyclipperOffset()
    offset.AddPath(box, pyclipper.JT_ROUND, pyclipper.ET_CLOSEDPOLYGON)
    expanded = np.array(offset.Execute(distance))
    return expanded
```

然后就会获得如下的恢复结果：

看上去，是不是哪里怪怪的？如果我们仔细的观察上述的预测结果，并和原始的GT 做差：

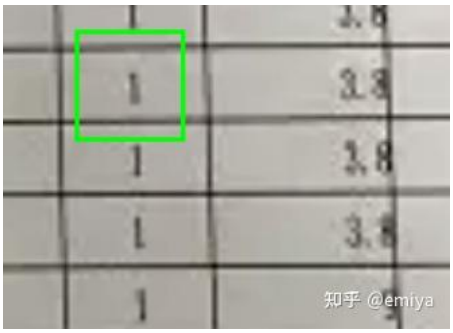


观察上面这张图，我们就会发现如下的问题：

> 1、对于长宽比差异不大的样本，dbnet 按照论文的公式还原的理论结果和GT差异是不大的
> 2、对于长宽比差异很大的框，dbnet 的还原结果和GT差异很大，预测会比GT缩小一圈

有的人会说，这个问题我遇到过！只需要把unclip ratio 设置的更大一些（比如从1.5设置到2.5），长宽比异常的框预测效果就会变好了！

但是按照上述的做法，也会遇到额外的问题，就是小的文本框可能会获得一个大一圈的预测，对识别的精度造成一定程度的下降，比如下面这个图，当面对图表/密集文字的时候，扩大的选框可能会将额外的背景框选进去，对识别造成干扰



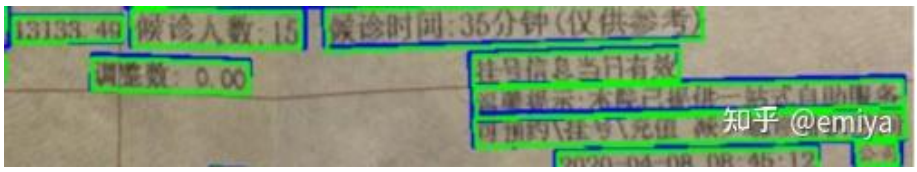
1.2 一个简单的解决思路

显然，造成上述问题的原因也是十分明显的，就是**对于不同的长宽比的矩形，使用相同shrink和dilate 的参数是不合理的,** 那么我们是否可以思考如下一种可能：

1、在测试的时候，我会希望使用一个统一的unclip_ratio ，方便对所有的预测结果统一的处理

2、训练的时候，**能否对所有不同长宽比的框，计算不同的 shrink_ratio， 使得按照统一的unclip_ratio 对收缩结果进行恢复，恢复后的框的面积和原始的面积之间的差差最不大?**

先说结论，结论当然是可以的，而且按照上述的思路训练出来的模型，在面对不同长宽比的框的时候预测的结果都很理想。如下图，蓝色的框是预测的结果，绿色的框是GT，所有的预测结果和GT 直接都非常的接近。



但是总的来说需要对原始的计算公式做一些微小的调整，在本文后续的内容中，会介动态计算 shrink ratio 的方法，以及对原始公式的一些微小调整

Part 2 动态计算Shrink Ratio

2.1 寻找合理的 shrink ratio

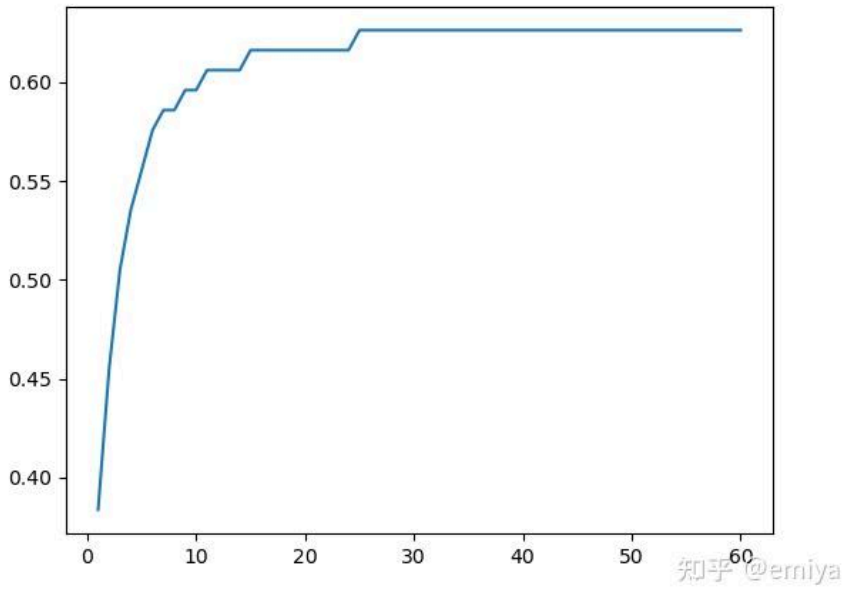
根据第一小节的设想，我们大致需要做如下的这么一件事情：

```
# step 1 设定目标的unclip r
unclip = 1.5

height = 1200
range_ = 60
for width_ratio in np.linspace(1.0 , range_ , range_):
    # step 2 遍历所有的长宽比的矩形框
    width = height * width_r
    polygon = make_polygon(height, width)

    min_diff = 100
    min_diff_r = None
    real_diff_r = None

    # step 3 遍历所有可能的 shrink_ratio
    for shrink_ratio in np.linspace(0,1,100):
        # 类似于dbnet 的shrink 函数，获取shrink 的框对应的最小外接矩形的高度和宽度
        s_height, s_width = get_s(height, width, shrink_ratio = shrink_ratio)
        # 类似于dbnet 的 dilate 函数，获取恢复的框的minAreaRect结果的高度和宽度
        lh, lw = get_l(s_height, s_width, unclip=unclip)
        # 计算恢复后的结果和原始GT 的差
        # 注意到这里，用了一个很别扭的 max(diff,0) ， 不要问为什么，我也忘了， 但是不这样
        diff = 1 - (lh * lw / height / width)
        diff_ = max(diff,0)
        if diff_ < min_diff:
            min_diff = diff_
            min_diff_r = shrink_ratio
            real_diff = diff
```

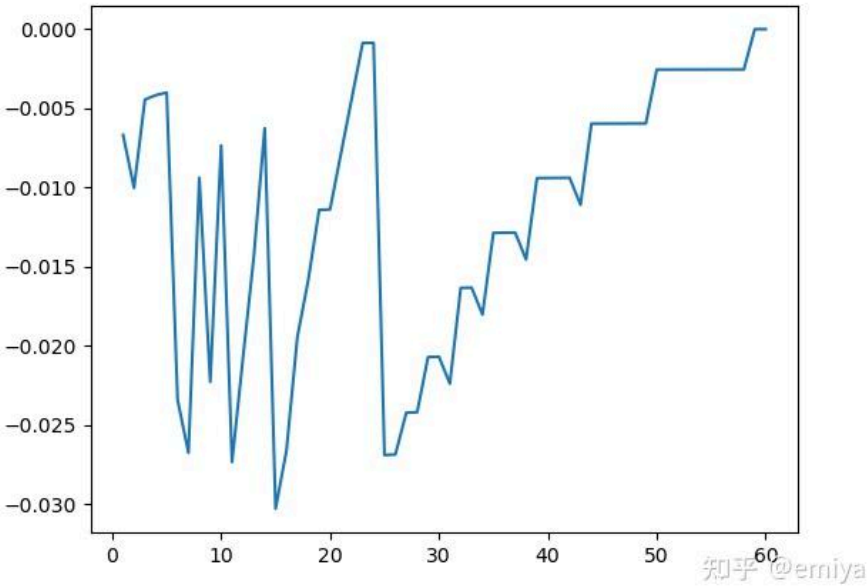
则通过上面这一端的代码，对于每一个 unclip ratio ，对于每一个长宽比的框，我们都能够获取的到一个对应的 min_diff_r

来简单的看一下效果：

可以看得出来，对于不同的长宽比的框，找出来的 shrink ratio 确实是不同的，而相比于默认的 0.6 的参数，对于长宽比较大的框，所需要的 shrink ratio 其实会大一些，而其实对于长宽比比较正常的框，使用shrink = 0.6 收缩的却又 " 略有不足" 。



当然，我们也可以额外的再多看一眼，对每个长宽比，按照最佳的shrink ratio 设置，所计算出来的 real_diff ，如下图，可以看到，最差的情况下，在长宽比约等于15的情况是，unclip的框所对应的面积只是原始GT面积的 103%，可以说也还是在可以接受的程度了（注意 real_diff 为负值意味着预测结果比原始的GT 更大）



额外提一句，在上述的伪代码当中有一个看上去有一些不对劲的地方，就是这里取了 diff_ =max(diff,0) ，即只考虑那些会使得收缩结果比原始结果稍微小一丢丢的shrink ratio，而那些收缩结果比GT稍微大一些但是也很好的情况都没有考虑进去。这里其实也是有大坑的，感兴趣的同学可以自己试试

2.2 小结

到这里，这篇专栏似乎就可以结束了，我们理清楚了如下的思路：

```
step 1 提前设定 unclipr = 1.
step 2 训练前计算出每一个长宽l
```

```
step 3 在训练过程中，对于每一个文本框 p
    step 3.1 利用 minAreaRect 计算文本框的长宽
    step 3.2 利用二分法，从预先计算好的 shrink-ratio 表当中查询最合适的shrink参数
    step 3.3 获取合理的 shrink 结果
```

但是，实际上，按照上述的做法，**在一些特殊的情况下，可能会失效。**所以接下来的内容才是本文的重点

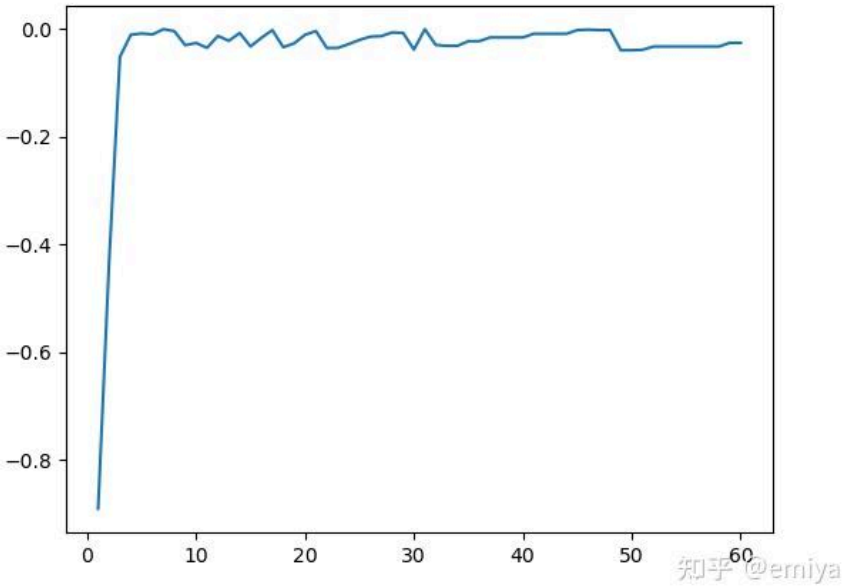
Part 3 DBNet GT 计算公式的不足

3.1 进一步探索 GT 计算的问题

上述的过程看上去都是非常的合理的，但是，**当我们来到 unclip_r = 3.5 , 似乎一些都不是那么的合理了**：

（unclip 取2.5以上，似乎都会有这个问题，3.5能看的更明显一些）

我们仍然按照上述的代码，对 unclip = 3.5 的情况进行分析unclip结果和GT之间的误差：



这时候就发现，好像哪里不对劲？？**对于unclip = 3.5 的情况下，会发现无论按照什么样的shrink ratio，对于长宽比在 1:1 ~ 1:3 的情况下，都无法获得合理的恢复的结果。**如下图是运行时打印的长宽比/找到的shrinkratio/unclip结果和GT的差值，可以看到对于长宽比在 1:3 以后的情况，unclip 的结果和原始结果差异在 5% 以内，但是对于前两行，恢复的结果会比原始的GT框大非常多

```
1.0000:0.0000:-0.8906,
2.0000:0.0000:-0.4356,
3.0000:0.0000:-0.0516,
4.0000:0.2121:-0.0104,
5.0000:0.2828:-0.0080,
6.0000:0.3232:-0.0097,
```

直观的理解下来大概是这样的，首先，DBNet 涉及的两个公式是非常不对称的，shrink 是作用在原始的GT 上，而 unclip 是

unclip = 3.5 的情况，可以认为是一种膨胀的“相对厉害”的情况，则按照原始的计算公式，对于正方形的框（第一行），哪怕是使用原始公式当中最厉害的收缩参数 shrink_ratio = 0.0 , 所获得的收缩的框再按照 unclip = 3.5 扩张回来，会获得的结果也是原来的结果的189%，即“扩大了好大一圈”。

3.2 修改公式，解决小长宽比GT构造问题

说起来复杂，解决起来倒是也很简单，就是给原始的DBNet GT构造方法继续“打补丁”，总的来说就是，**对于小长宽比的矩形，我们在进行收缩的时候，需要使用比原始论文更加“过分”的收缩参数继续收缩**

即在原始的收缩参数的计算公式上额外 的设置一个 scale

$$D = \frac{A(1-r^2)*scale}{L}$$

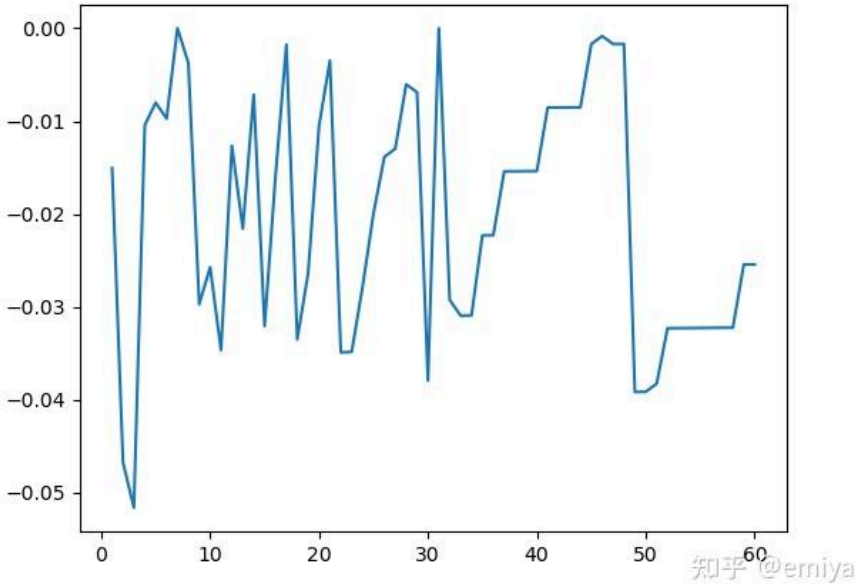
并且在原始的计算流程当中引入 scale:

> scale =1.5 if 1<= max(height,width)/min(height,width) <=2 else 1

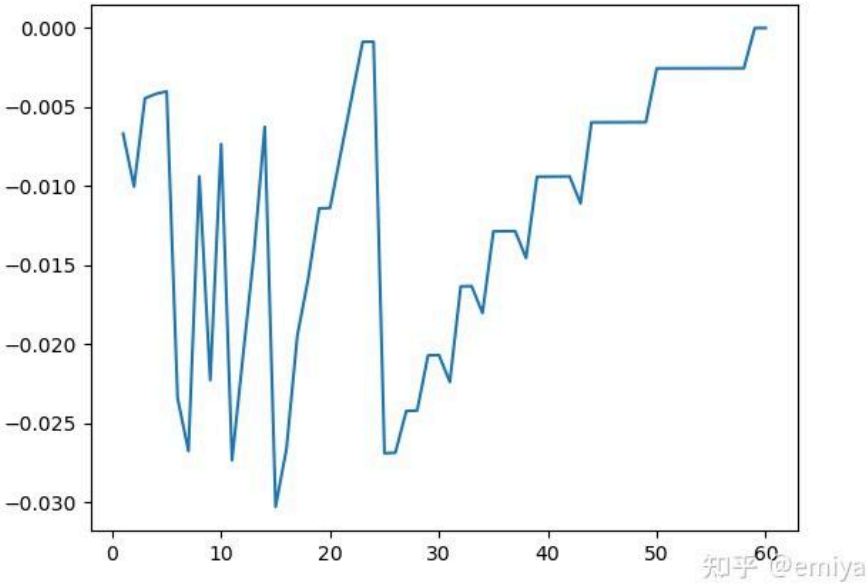
```
# step 1 设定目标的unclip r
unclip = 3.5

height = 1200
range_ = 60
for width_ratio in np.linspace(1.0 , range_ , range_):
    # step 2 遍历所有的长宽比的矩形框
    width = height * width_r
    polygon = make_polygon(height, width)

    min_diff = 100
    min_diff_r = None
    real_diff_r = None

    # step 3 遍历所有可能的 shrink_ratio
    for shrink_ratio in np.linspace(0,1,100):
        # 类似于dbnet 的shrink 函数，获取shrink 的框对应的最小外接矩形的高度和宽度
        scale =  1.5 if 1 <= max(height,width) / min(height,width) <=2  else 1
        s_height, s_width = get_s(height, width, shrink_ratio = shrink_ratio , sc
        lh, lw = get_l(s_height, s_width, unclip=unclip)
        # 计算恢复后的结果和原始GT 的差
        # 注意到这里，用了一个很别扭的 max(diff,0) ， 不要问为什么，我也忘了， 但是不这样
        diff = 1 - (lh * lw / height / width)
        diff_ = max(diff,0)
        if diff_ < min_diff:
            min_diff = diff_
            min_diff_r = shrink_ratio
            real_diff = diff

def get_s(height, width, max_r=0.5, scale = 1.5):
    s_offset = Area(height, width) * (1 - max_r**2) * scale / length(height, widt

    polygon = np.array([0, 0, width, 0, width, height, 0, height]).reshape(-1,2).
    pco = pyclipper.PyclipperOffset()
    pco.AddPath(polygon, pyclipper.JT_ROUND,
                pyclipper.ET_CLOSEDPOLYGON)
    shrunk = np.array(pco.E
    if shrunk.shape[0] > 0:
```

```
        _ , (w,h) , _ = cv2.minAreaRect(shrunk)
        return h , w
    else:
        s_height_ = height - 2 * s_offset
        s_width_ = width - 2 * s_offset
    return s_height_, s_width_


def get_l(s_height, s_width, unclip = 1.5):
    # s_width = s_height * ratio
    d = Area(s_height, s_width) * unclip / length(s_height, s_width)
    l_height = s_height + 2 * d
    l_width = s_width + 2 * d

    poly = np.array([0,0,s_width,0,s_width,s_height,0,s_height]).reshape(4,2)

    poly = unclipf(poly, unclip_ratio=unclip)

    if poly.shape[0] == 0 :
        return l_height, l_width
    else:
        _ , (w , h) , _ = cv2.minAreaRect(poly)
        return h , w

def unclipf(box, unclip_ratio=1.5):
    poly = plg(box)
    distance = poly.area * unclip_ratio / poly.length
    offset = pyclipper.PyclipperOffset()
    offset.AddPath(box, pyclipper.JT_ROUND, pyclipper.ET_CLOSEDPOLYGON)
    expanded = np.array(offset.Execute(distance))
    return expanded



def Area(height, width):
    return height * width


def length(height, width):
    return 2 * (height + width)
```

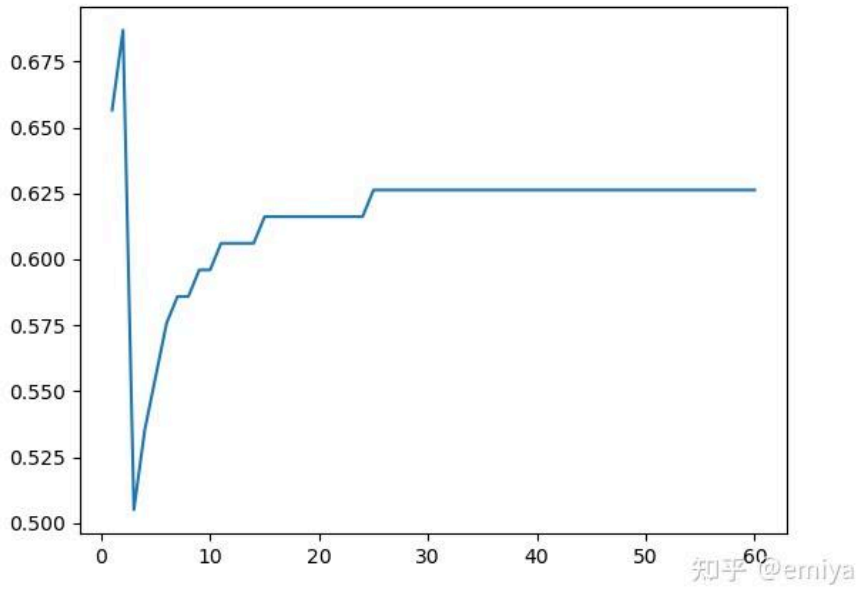
根据修改的上述代码，再可视化的来看 长宽比和 real_diff 之间的关系，可以看到哪怕是对 unclip = 3.5 ，一样能够获取得到合理的shrink ratio， 使得最终的 real_diff 是可接受的



而这样的扰动对于 unclip = 1.5 的情况的误差也不会有太大的影响：



但是会稍微影响到最佳的shrink ratio 的选择，毕竟在小长宽比的区间乘上了一个scale



3.3 one more thing

到这里这篇文章就真的讲完了，估计认真看到这里的话应该可以帮到你。不过如果你认真的按照这个这个专栏做到这里的话，可以开始思考如下几个问题（只可意会，不可言传~）

> 1、文章当中为什么要提到 unclip = 3.5？
> 2、使用unclip = 3.5，会造成正样本的数量急剧下降，如何解决
> 3、使用unclip = 3.5，对于小目标会非常不友好（如果目标很小，构造GT 的时候会收缩消失），如何解决?

最后，欢迎点个赞或者收藏 :)

编辑于 2022-11-03 15:20 · 上海

OCR（光学字符识别）　计算机视觉　深度学习（Deep Learning）

阿珺

文章太赞了[发呆

2023-03-29 · 北京

回复 喜欢

bare

只看懂了前部分，后部分为啥要用3.5不太懂，能解答一下吗，感谢

2023-01-18 · 安徽

回复 喜欢

bare ‣ **emiya**

请问特殊情况是指有重叠的情况么

2023-02-12 · 安徽

回复 喜欢

emiya 作者

在一些特殊的情况下，1.5 的 kernel size 不够用

2023-02-12 · 中国香港

回复 喜欢