合肥工業大学
HEFEI UNIVERSITY OF TECHNOLOGY

IoU 损失汇总

汇报人:柯水洲

日期：2021年1月4日

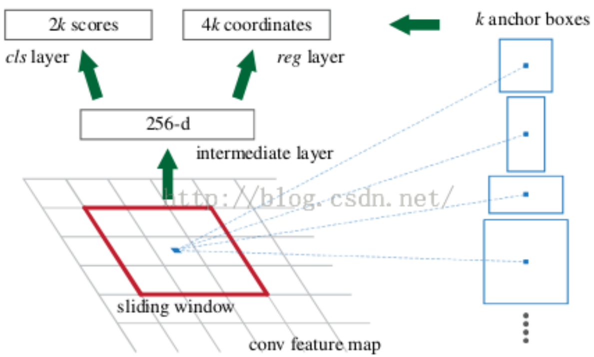
- 目标检测任务的损失函数由两部分构成：
  - Classification Loss
  - Bounding Box Regression Loss
- Smooth L2 Loss —> IoU Loss —> GIoU Loss —> DIoU Loss —> CIoU Loss
- IoU就是我们所说的交并比，是目标检测中最常用的指标，在anchor-based的方法中，他的作用不仅用来确定正样本和负样本，还可以用来评价输出框（predict box）和ground-truth的距离。

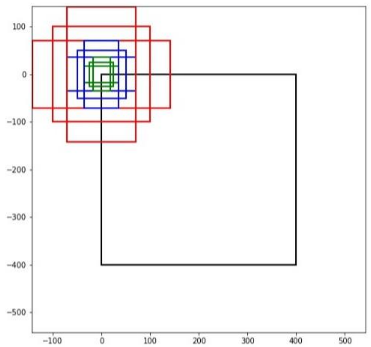
Anchor

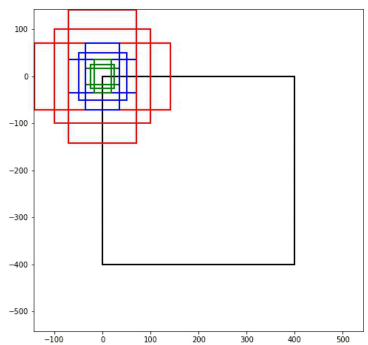
- 就是在图像上预设好的不同大小，不同长宽比的参照框。
- RCNN、Fast RCNN（selective search）;Faster RCNN(RPN)
- 实际的SSD模型，在300x300的输入下，anchor数量也特别多，其在38x38、19x19、10x10、5x5、3x3、1x1的六个特征图上，每个点分别设置4、6、6、6、6、4个不同大小和长宽比的anchor，所以一共有38x38x4+ 19x19x6+ 10x10x6+5x5x6+ 3x3x4+ 1x1x4= 8732个anchor。

假设原图 400×400 。我们选取三种长宽比 $[2, 1, 0.5]$,三种scale $[0.5, 0.25, 0.125]$,得到9个不同的anchor.

	2 : 1	1 : 1	1： 2
200×200	282.8×141.4	200×200	141.4×282.8
100×100	141.4×70.7	100×100	70.7×141.4
50×50	70.7×35.3	50×50	35.3×70.7



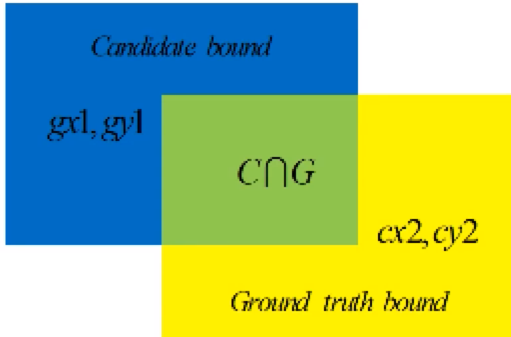




IOU

- 尺度不变性

$$IoU = \frac{|A \cap B|}{|A \cup B|}$$



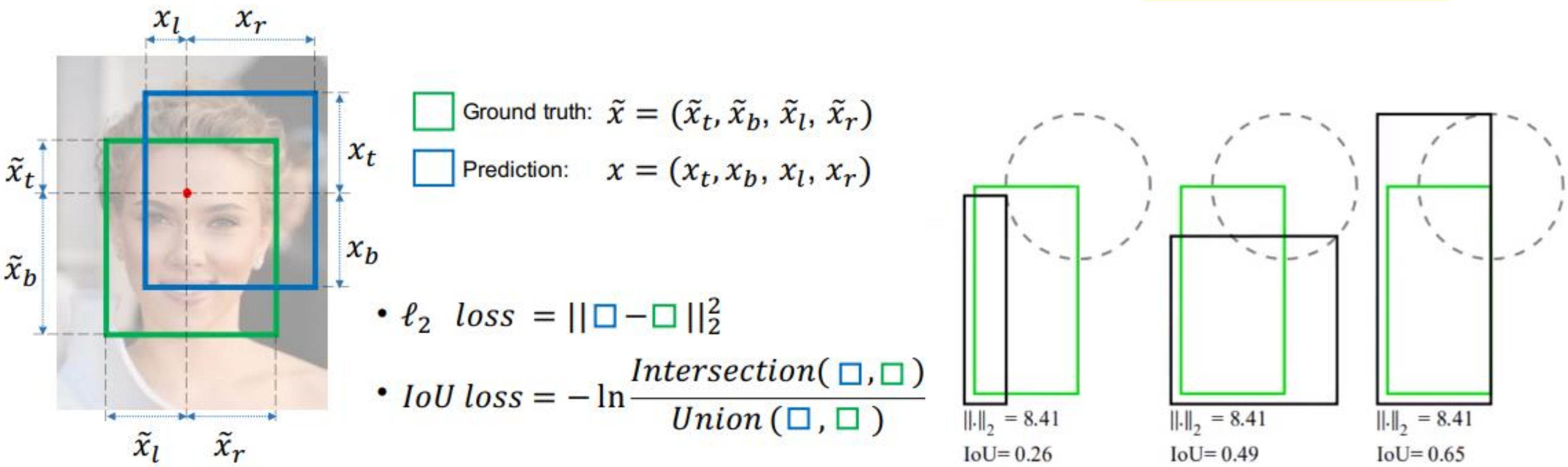


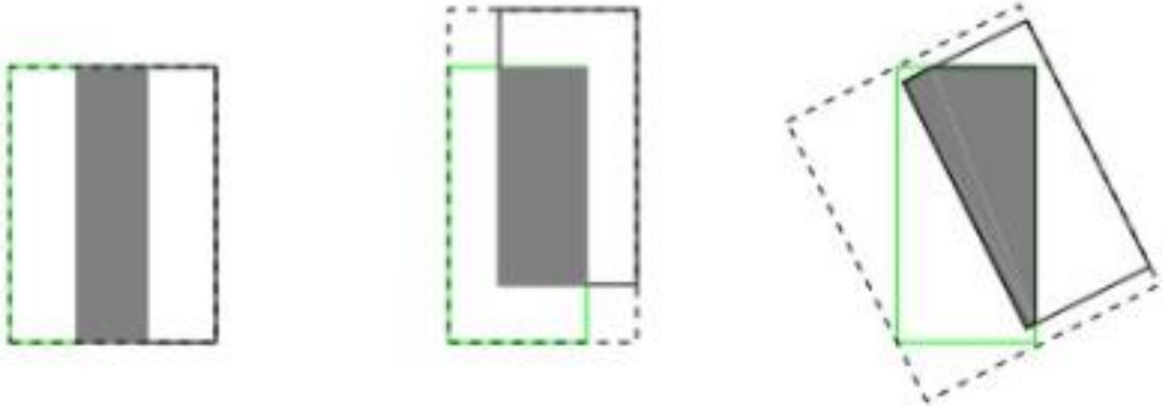
Figure 1: Illustration of IoU loss and ℓ_2 loss for pixel-wise bounding box prediction.

```python
import numpy as np
def Iou(box1, box2, wh=False):
    if wh == False:
        xmin1, ymin1, xmax1, ymax1 = box1
        xmin2, ymin2, xmax2, ymax2 = box2
    else:
        xmin1, ymin1 = int(box1[0]-box1[2]/2.0), int(box1[1]-box1[3]/2.0)
        xmax1, ymax1 = int(box1[0]+box1[2]/2.0), int(box1[1]+box1[3]/2.0)
        xmin2, ymin2 = int(box2[0]-box2[2]/2.0), int(box2[1]-box2[3]/2.0)
        xmax2, ymax2 = int(box2[0]+box2[2]/2.0), int(box2[1]+box2[3]/2.0)
    # 获取矩形框交集对应的左上角和右下角的坐标（intersection）
    xx1 = np.max([xmin1, xmin2])
    yy1 = np.max([ymin1, ymin2])
    xx2 = np.min([xmax1, xmax2])
    yy2 = np.min([ymax1, ymax2])
    # 计算两个矩形框面积
    area1 = (xmax1-xmin1) * (ymax1-ymin1)
    area2 = (xmax2-xmin2) * (ymax2-ymin2)
    inter_area = (np.max([0, xx2-xx1])) * (np.max([0, yy2-yy1])) #计算交集面积
    iou = inter_area / (area1+area2-inter_area+1e-6)  #计算交并比

    return iou
```

- 如果两个框没有相交，根据定义，IoU=0，不能反映两者的距离大小（重合度）。同时因为loss=0，没有梯度回传，无法进行学习训练。

- IoU无法精确的反映两者的重合度大小。如下图所示，三种情况IoU都相等，但看得出来他们的重合度是不一样的，左边的图回归的效果最好，右边的最差。



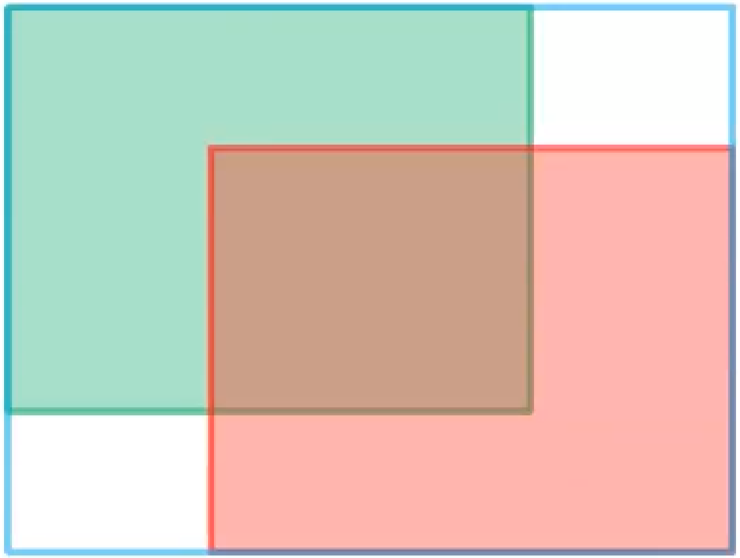
GIoU(Generalized Intersection over Union)

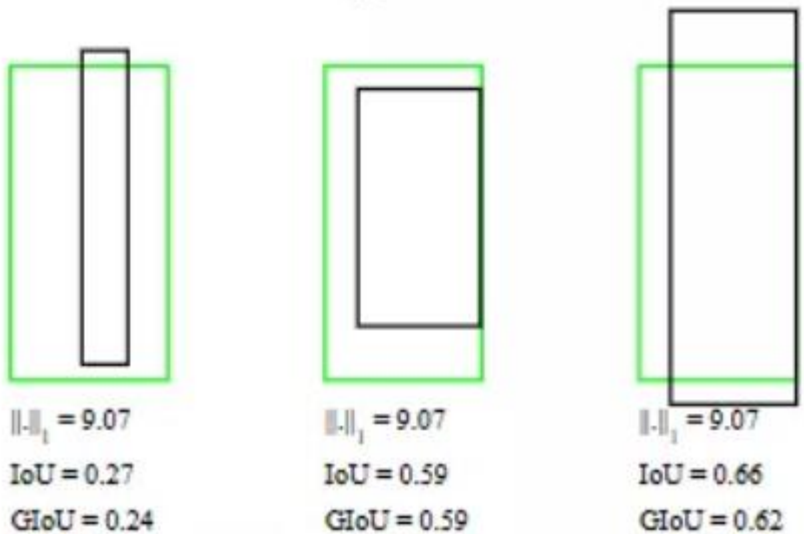
- 与IoU只关注重叠区域不同，GIoU不仅关注重叠区域，还关注其他的非重合区域，能更好的反映两者的重合度。
- 尺度不变性

$$GIoU = IoU - \frac{|A_c - U|}{|A_c|}$$

$$L_{GIoU} = 1 - GIoU$$

$$0 \le L_{GIoU} \le 2$$





Hamid Rezatofighi .et al Generalized Intersection over Union: A Metric and A Loss for Bounding Box Regression CVPR2019

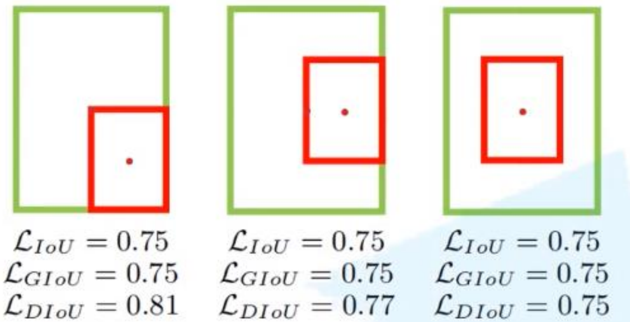
```python
def Giou(rec1,rec2):
    #分别是第一个矩形左右上下的坐标
    x1,x2,y1,y2 = rec1
    x3,x4,y3,y4 = rec2
    iou = Iou(rec1,rec2)
    area_C = (max(x1,x2,x3,x4)-min(x1,x2,x3,x4))*(max(y1,y2,y3,y4)-min(y1,y2,y3,y4))
    area_1 = (x2-x1)*(y1-y2)
    area_2 = (x4-x3)*(y3-y4)
    sum_area = area_1 + area_2

    w1 = x2 - x1   #第一个矩形的宽
    w2 = x4 - x3   #第二个矩形的宽
    h1 = y1 - y2
    h2 = y3 - y4
    W = min(x1,x2,x3,x4)+w1+w2-max(x1,x2,x3,x4)    #交叉部分的宽
    H = min(y1,y2,y3,y4)+h1+h2-max(y1,y2,y3,y4)    #交叉部分的高
    Area = W*H    #交叉的面积
    add_area = sum_area - Area    #两矩形并集的面积

    end_area = (area_C - add_area)/area_C    #闭包区域中不属于两个框的区域占闭包区域的
    giou = iou - end_area
    return giou
```

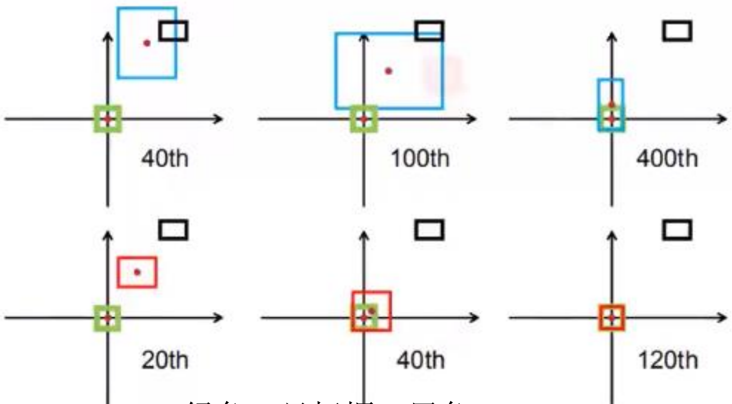
DIoU(Distance-IoU)

— DIoU能够最小化两个边界框之间的距离，因此可以加快收敛速度

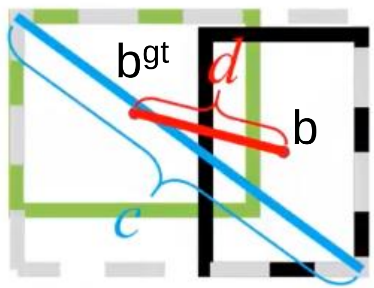


$$DIoU = IoU - \frac{\rho^2(b, b^{gt})}{c^2} = IoU - \frac{d^2}{c^2}$$

$$-1 \le DIoU \le 1$$



绿色：目标框；黑色：anchor
蓝色：GIoU的预测框
红色：DIoU的预测框



ρ代表两者间欧氏距离

Zhaohui Zheng.et al Distance-IoU Loss: Faster and Better Learning for Bounding Box Regression AAAI, 2020

```python
def Diou(bboxes1, bboxes2):
    rows = bboxes1.shape[0]
    cols = bboxes2.shape[0]
    dious = torch.zeros((rows, cols))
    if rows * cols == 0:#
        return dious
    exchange = False
    if bboxes1.shape[0] > bboxes2.shape[0]:
        bboxes1, bboxes2 = bboxes2, bboxes1
        dious = torch.zeros((cols, rows))
        exchange = True
    # #xmin,ymin,xmax,ymax->[:,0],[:,1],[:,2],[:,3]
    w1 = bboxes1[:, 2] - bboxes1[:, 0]
    h1 = bboxes1[:, 3] - bboxes1[:, 1]
    w2 = bboxes2[:, 2] - bboxes2[:, 0]
    h2 = bboxes2[:, 3] - bboxes2[:, 1]

    area1 = w1 * h1
    area2 = w2 * h2

    center_x1 = (bboxes1[:, 2] + bboxes1[:, 0]) / 2
    center_y1 = (bboxes1[:, 3] + bboxes1[:, 1]) / 2
    center_x2 = (bboxes2[:, 2] + bboxes2[:, 0]) / 2
    center_y2 = (bboxes2[:, 3] + bboxes2[:, 1]) / 2
```

```python
inter_max_xy = torch.min(bboxes1[:, 2:],bboxes2[:, 2:])
inter_min_xy = torch.max(bboxes1[:, :2],bboxes2[:, :2])
out_max_xy = torch.max(bboxes1[:, 2:],bboxes2[:, 2:])
out_min_xy = torch.min(bboxes1[:, :2],bboxes2[:, :2])

inter = torch.clamp((inter_max_xy - inter_min_xy), min=0)
inter_area = inter[:, 0] * inter[:, 1]
inter_diag = (center_x2 - center_x1)**2 + (center_y2 - center_y1)**2
outer = torch.clamp((out_max_xy - out_min_xy), min=0)
outer_diag = (outer[:, 0] ** 2) + (outer[:, 1] ** 2)
union = area1+area2-inter_area
dious = inter_area / union - (inter_diag) / outer_diag
dious = torch.clamp(dious,min=-1.0,max = 1.0)
if exchange:
    dious = dious.T
return dious
```

CIoU(Distance-IoU)

– 一个优秀的回归定位损失应考虑：重叠面积、中心点距离、长宽比

$$CIoU = IoU - (\frac{\rho^2(b, b^{gt})}{c^2} + \alpha \upsilon)$$

$$\upsilon = \frac{4}{\pi^2}(\arctan\frac{w^{gt}}{h^{gt}} - \arctan\frac{w}{h})^2$$

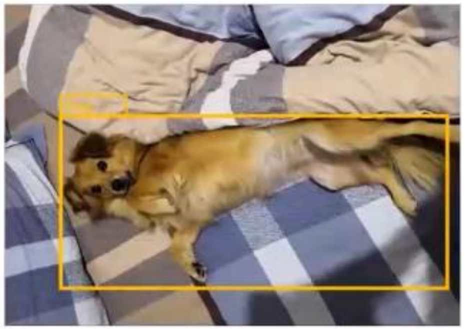
$$\alpha = \frac{\upsilon}{(1 - IoU) + \upsilon}$$

v衡量长宽比的相似性
α代表权重

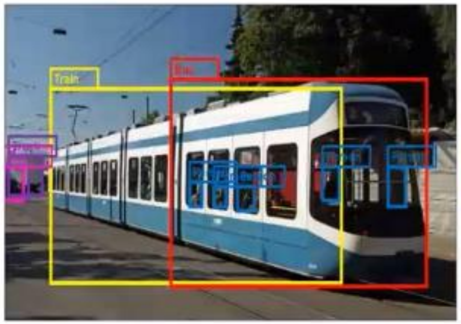
$\mathcal{L}_{GIoU}$

$\mathcal{L}_{CIoU}$

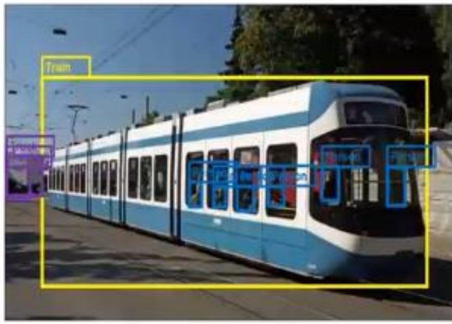
$\mathcal{L}_{GIoU}$



$\mathcal{L}_{CIoU}$



$\mathcal{L}_{GIoU}$



$\mathcal{L}_{CIoU}$

```python
def bbox_overlaps_ciou(bboxes1, bboxes2):
    rows = bboxes1.shape[0]
    cols = bboxes2.shape[0]
    cious = torch.zeros((rows, cols))
    if rows * cols == 0:
        return cious
    exchange = False
    if bboxes1.shape[0] > bboxes2.shape[0]:
        bboxes1, bboxes2 = bboxes2, bboxes1
        cious = torch.zeros((cols, rows))
        exchange = True

    w1 = bboxes1[:, 2] - bboxes1[:, 0]
    h1 = bboxes1[:, 3] - bboxes1[:, 1]
    w2 = bboxes2[:, 2] - bboxes2[:, 0]
    h2 = bboxes2[:, 3] - bboxes2[:, 1]

    area1 = w1 * h1
    area2 = w2 * h2

    center_x1 = (bboxes1[:, 2] + bboxes1[:, 0]) / 2
    center_y1 = (bboxes1[:, 3] + bboxes1[:, 1]) / 2
    center_x2 = (bboxes2[:, 2] + bboxes2[:, 0]) / 2
    center_y2 = (bboxes2[:, 3] + bboxes2[:, 1]) / 2
    inter_max_xy = torch.min(bboxes1[:, 2:],bboxes2[:, 2:])
    inter_min_xy = torch.max(bboxes1[:, :2],bboxes2[:, :2])
    out_max_xy = torch.max(bboxes1[:, 2:],bboxes2[:, 2:])
    out_min_xy = torch.min(bboxes1[:, :2],bboxes2[:, :2])

    inter = torch.clamp((inter_max_xy - inter_min_xy), min=0)
    inter_area = inter[:, 0] * inter[:, 1]
    inter_diag = (center_x2 - center_x1)**2 + (center_y2 - center_y1)**2
    outer = torch.clamp((out_max_xy - out_min_xy), min=0)
    outer_diag = (outer[:, 0] ** 2) + (outer[:, 1] ** 2)
    union = area1+area2-inter_area
    u = (inter_diag) / outer_diag
    iou = inter_area / union
    with torch.no_grad():
        arctan = torch.atan(w2 / h2) - torch.atan(w1 / h1)
        v = (4 / (math.pi ** 2)) * torch.pow((torch.atan(w2 / h2) - torch.atan(w1 / h1)), 2
        S = 1 - iou
        alpha = v / (S + v)
        w_temp = 2 * w1
    ar = (8 / (math.pi ** 2)) * arctan * ((w1 - w_temp) * h1)
    cious = iou - (u + alpha * ar)
    cious = torch.clamp(cious,min=-1.0,max = 1.0)
    if exchange:
        cious = cious.T
    return cious
```

Q&A

汇报人:柯水洲

日期：2021年1月4日