NMS 调研

导航

1 前言

以下是在学习基于深度学习方法实现目标检测的过程中，总结的论文、学习报告、博客等等，以及针对部分问题提出的个人看法，特此记录！

2 网络架构

2.1 RCNN 系列

2.1.1 RCNN

2.1.2 Fast RCNN

2.1.3 Faster RCNN

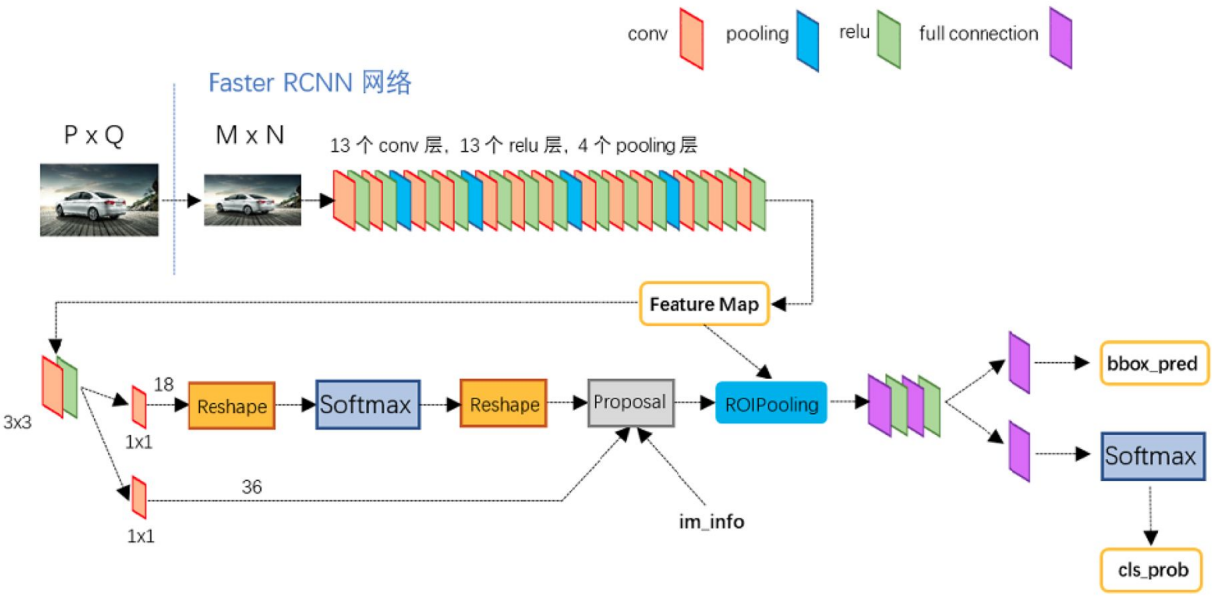


图 1: Faster RCNN 网络架构

Two-stages 中最为经典的物体检测算法。具体地，大致可以分为四个部分：

- Backbone (VGG16、Resnet101 等等) 输入 RGB 图片数据，压缩 16 倍【4 次下采样】，输出 Feature maps；
- Region Proposal Networks (RPN) 用于生成 region proposals **【可以理解为在原图尺度上，设置密密麻麻的候选 anchors，然后用 CNN 判断哪些 anchors 里面有目标，哪些没有】**【一张 $M \times N$ 的图，一共有 $ceil(M/16) \times ceil(N/16) \times num_{anchors}$ 个 anchors】【全部 anchors 拿去训练太多了，随机选取 128 个正例和 128 个负例】。

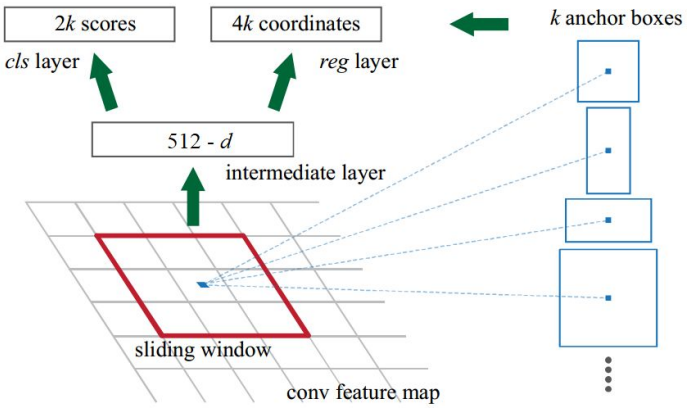


图 2: Anchors 示意图

feature map 经过 512 个 3×3 的卷积核之后得到 512 个通道的特征图。

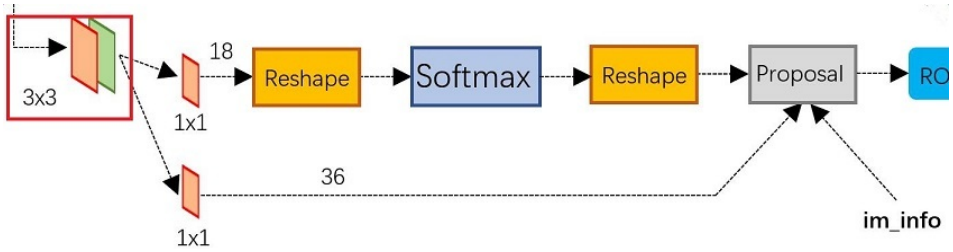


图 3: RPN 结构示意图

- 图3上面一层 (图2左边) **512-dimensions【**不是 512 减 d**】**特征图通过 softmax【softmax 常和 sigmoid 对比，前者**计算梯度简单，但要求类别相互独立】**转化为 **cls = 2k**，并判断 anchors 属于正例还是负例。具体的判断标准如下：
  - 与 Ground truth 的 IoU>0.7，记为正例；
  - 与 Ground truth 的 IoU<0.3，记为负例
- 图3下面一层 (图2右边) 利用 bounding box regression 修正 anchors 获得 proposals，并**剔除太小和超出边界的 proposals【**实现目标定位**】**；

- RoI Pooling。2 个输入：Feature maps (蓝色) & proposals (黄色)；

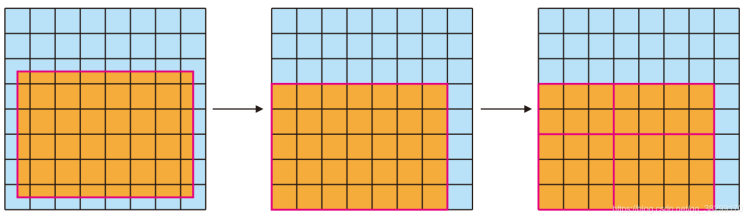


图 4: RoI Pooling 输出为 2×2 时的流程图

提醒：

- **为什么需要 RoI Pooling？**
  后面接全连接层要求**固定输入的大小**。因此当大小不定时，**要么从图像中 crop 一部分【破坏了图像的完整性】，要么将图像 warp 为需要的大小【破坏了图像的形状】**
- **不匹配问题！见图4**
  * Region proposal 的 xywh 通常是小数，但为了方便操作会将其整数化；
  * 将整数化后的边界区域平均分割为 $k \times k$ 个单元，还需要对每个单元的边界整数化。

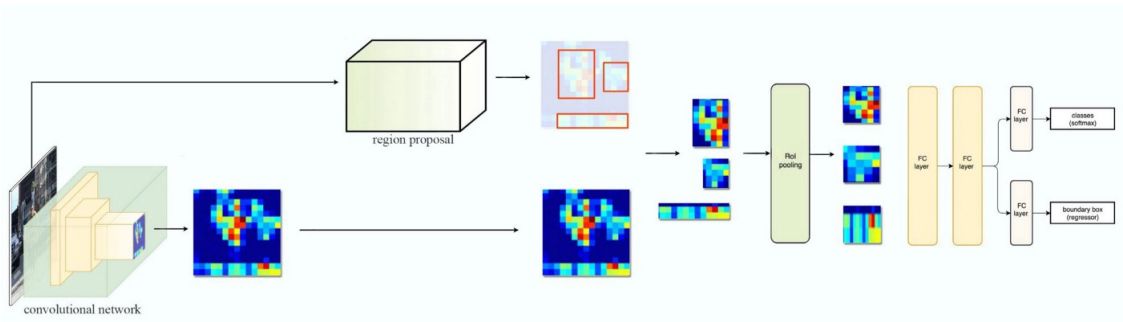
  此时，候选框和最开始的位置存在一定的偏差。【在2.1.4节中提出了相应的解决办法】

- Classification。计算 proposal 的类别，并再次 bounding box regression 获得检测框的精确位置。

具体的训练步骤：

1. 在已经训练好的 model 上，训练 RPN；
2. 在步骤 1 的基础上，收集 proposals；
3. 训练 Faster RCNN；
4. 第二次训练 RPN；
5. 在步骤 4 的基础上，收集 proposals；
6. 第二次训练 Faster RCNN

测试阶段：



2.1.4 Mask RCNN

2.2 YOLO 系列

2.2.1 YOLOV1

2.2.2 YOLOV2

2.2.3 YOLOV3

输入： $(bs, 416, 416, 3)$

具体地，将原图按纵横比例缩放至 416×416，取 $\mathbf{min}(w/img\_w, h/img\_h)$ 保证图片不会出现扭曲，多余的部分用 $(128, 128, 128)$ 填充。相应地，标签也需要进行相应调整以及归一化处理【避免回归框时，大边框过度主导梯度方向】

```
# annotation_line: ROOT/000005.jpg 263,211,324,339,1
def get_random_data(self, annotation_line, input_shape, jitter=.3, hue=.1, sat=0.7, val=0.4, random=True):
    line    = annotation_line.split()
    image   = Image.open(line[0])
    image   = cvtColor(image)      # 读取图像并转换成RGB图像
    iw, ih = image.size            # 获取图像的宽高
    h, w    = input_shape          # 获取目标的宽高
    # 获取目标框信息
    box     = np.array([np.array(list(map(int, box.split(',')))) for box in line[1:]])
    if not random:
        scale       = min(w/iw, h/ih)
        nw          = int(iw*scale)
        nh          = int(ih*scale)
        dx          = (w-nw)//2
        dy          = (h-nh)//2
        image       = image.resize((nw,nh), Image.BICUBIC)
        new_image   = Image.new('RGB', (w,h), (128,128,128)) #  多余的部分加上灰条
        new_image.paste(image, (dx, dy))
        image_data = np.array(new_image, np.float32)

        #   调整真实框
        if len(box) > 0:
            np.random.shuffle(box)
            box[:, [0,2]] = box[:, [0,2]]*nw/iw + dx
            box[:, [1,3]] = box[:, [1,3]]*nh/ih + dy
            box[:, 0:2][box[:, 0:2]<0] = 0
            box[:, 2][box[:, 2]>w]      = w
            box[:, 3][box[:, 3]>h]      = h
            box_w = box[:, 2] - box[:, 0]
            box_h = box[:, 3] - box[:, 1]
            box   = box[np.logical_and(box_w>1, box_h>1)] # 丢弃无效框
        return image_data, box
```

Backbone： Darknet-53【要求输入图片的大小是 32 的倍数】，包含 53 个卷积层，没有池化层【4 次下采样通过卷积操作实现。卷积相较于池化，前者更注重保留信息，且增加了非线性变换的次数，后者更注重筛选信息】

	类型	卷积核数量	卷积核尺寸	输出
	卷积层	32	3×3	256×256
	卷积层	64	3×3/2	128×128
1×	卷积层	32	1×1	
	卷积层	64	3×3	
	残差层			128×128
	卷积层	128	3×3/2	64×64
2×	卷积层	64	1×1	
	卷积层	128	3×3	
	残差层			64×64
	卷积层	256	3×3/2	32×32
8×	卷积层	128	1×1	
	卷积层	256	3×3	
	残差层			32×32
	卷积层	512	3×3/2	16×16
8×	卷积层	256	1×1	
	卷积层	512	3×3	
	残差层			16×16
	卷积层	1 024	3×3/2	8×8
4×	卷积层	512	1×1	
	卷积层	1 024	3×3	
	残差层			8×8

图 5: Darknet-53 结构图。残差结构缓解梯度消失的问题，同时复用特征，强化学习网络

Backbone 输出： $bs \times 52 \times 52 \times 256$、$bs \times 26 \times 26 \times 512$、$\times 13 \times 13 \times 1024$ 三类特征图【小尺度特征图用于检测大尺寸物体；大尺寸特征图用于检测小尺度特征图】

Neck： 由五个卷积层组成，负责生成特征金字塔。具体地：将 $bs \times 13 \times 13 \times 1024$ 的特征层上采样变为 $bs \times 26 \times 26 \times 256$，再与 $bs \times 26 \times 26 \times 512$ 堆叠，得到维度为 $bs \times 26 \times 26 \times 768$ 的特征图，经由五个卷积层后，变为 $bs \times 26 \times 26 \times 256$。

Head： 由两个卷积层【卷积核分别为 3×3、1×1，后一个卷积操作改变通道数的同时实现预测】组成，输出结果分别为 $bs \times 13 \times 13 \times (3 * (4 + 1 + cls\_num))$ 、$bs \times 26 \times 26 \times (3 * (4 + 1 + cls\_num))$ 、$bs \times 52 \times 52 \times (3 * (4 + 1 + cls\_num))$

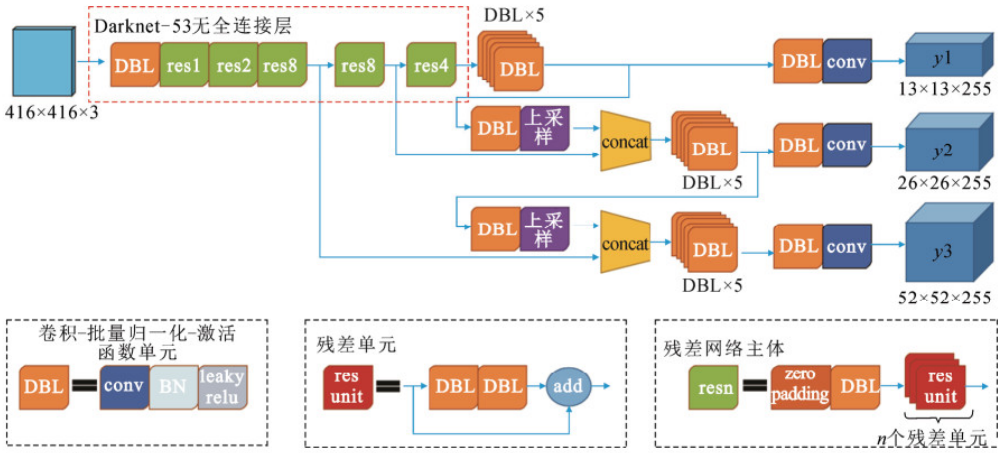


图 6: YOLOV3 结构图

预测： $bs \times N \times N \times [3 \times (4 + 1 + \text{cls\_num})]$

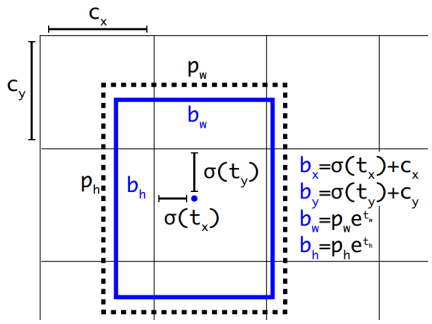


图 7: 检测框解码

预测结果（t_x，t_y，t_w，t_h）解码【中心点平移到蓝色点的过程】：
网络输出获得边界框预测：

$$b_x = (\sigma(t_x) + c_x)/W \tag{1}$$

$$b_y = (\sigma(t_y) + c_y)/H \tag{2}$$

$$b_w = p_w e^{tw}/W \tag{3}$$

$$b_h = p_h e^{th}/H \tag{4}$$

其中，(c_x, c_y) 为网格从顶左部的坐标；p_w，p_h 分别是锚框的维度；W，H 分别是特征图的维度。

```
# 真实框标签化
# b: batch-size; k: 第k+1个框; (i, j): 网格点坐标; t: 第t+1个真实框
y_true[b, k, j, i, 0] = batch_target[t, 0] - i.float()
y_true[b, k, j, i, 1] = batch_target[t, 1] - j.float() # 不需要除以anchors[best_n]的宽高
y_true[b, k, j, i, 2] = math.log(batch_target[t, 2] / anchors[best_n][0])
y_true[b, k, j, i, 3] = math.log(batch_target[t, 3] / anchors[best_n][1]) # 消除框的大小对误差的影响
```

```
y_true[b, k, j, i, 4] = 1 # 置信度标签
y_true[b, k, j, i, c + 5] = 1 # 类别标签
```

在 faster-RCNN 中，t_x，t_y，t_w，t_h 依次可以表示为：

$$t_x = (G_x - P_x)/P_w \tag{5}$$

$$t_y = (G_y - P_y)/P_h \tag{6}$$

$$t_w = \log(G_x/P_x) \tag{7}$$

$$t_h = \log(G_x/P_x) \tag{8}$$

其中，(P_x, P_y) 和 (G_x, G_y) 分别为预设的 anchor box 和 ground truth 在特征图上的中心点坐标；(P_w, P_h) 和 (G_w, G_h) 分别为预设的 anchor box 和 ground truth 在特征图上的宽和高【归一化变绝对尺度为相对尺度】。

提醒：

- 没有直接回归 bounding box 的长宽而是**尺度缩放到对数空间，去掉不等式约束【相对形变必须大于 0】**
- 当输入的 Proposal 与 ground truth 相差较小时，可以认为边框回归是一种**线性变换**

$$t_w = \log(G_w/P_w) = \log\left(1 + \frac{G_w - P_w}{P_w}\right) = \frac{G_w - P_w}{P_w} \quad (P_w - G_w \to 0)$$

- sigmoid 压缩 t_x，t_y 到 $(0, 1)$，可以有效地确保目标中心位于预测的网格单元中

提醒： **关于多尺度特征图的标签分配：**

- YOLOV3 不像 YOLOV1【每个 grid 负责中心落在该 grid 中的 ground truth；YOLOV3 三类特征图可能会存在中心重合】
- 两种分配思路：

  i>. 先根据 ground truth 的大小和所有 anchor 的 IOU 分配合适的 anchor 作为正例

```
# gt_box:(num_true_box, 4)
# anchor_shapes: (9, 4)
# best_ns: [每个真实框最大的重合度, 每一个真实框最重合的先验框的序号]
best_ns = torch.argmax(self.calculate_iou(gt_box, anchor_shapes), dim=-1)
# 为 每一个真实框最重合的先验框 适配对应尺度的特征图, l: 0/1/2
for t, best_n in enumerate(best_ns):
    if best_n not in self.anchors_mask[l]: continue
```

  ii>. 直接在 3 种尺度上，选 ground truth 对应 IOU 最大的 anchor

提醒： **预测框【总共 $\sum_{i=0}^{3} 3 \times N_i \times N_i$】的三种情况**

- 正例：任取一个 ground truth，与所有的预测框计算 IOU，IOU 最大的预测框即为正例【一个预测框只能分配一个 ground truth；正例计算检测框 loss、置信度 loss、类别 loss】；
- 忽略样例：正例除外，与任意一个 ground truth 的 IOU 大于阈值，则为忽略样例【忽略样本样例不产生任何 loss】；

```
# obj_mask：obj_mask = y_true[..., 4] == 1，其他为False
# noobj_mask：正例对应的noobj_mask[b, k, j, i] = 0；noobj_mask[b][anch_ious_max > self.
    ignore_threshold] = 0，其他为1，即为负例
# 只有正例和负例计算置信度损失
loss_conf = torch.mean(self.BCELoss(conf, obj_mask.type_as(conf))[noobj_mask.bool() | obj_mask
    ])
```

- 负例：正例除外，与全部 ground truth 的 IOU 都小于阈值，则为负例【正例样本与 ground truth 的 IOU 可能小于 0.5，不冲突；只计算置信度 loss】

损失函数：由四个部分组成：

- 边界框损失：
  在 YOLOV1 中，计算长宽损失**多一个根号**，【对于相同的误差值，大物体误差对检测的影响应小于小物体误差对检测的影响。例如，5 个像素点的偏差，对于 416×416 的预测框几乎没有影响，但是对于 10×10 的预测框影响就很大】【但是加了根号后对小框的损失来说有些过大。例如 $\left(\sqrt{20}-\sqrt{10}\right)^2 = 3.43$、$\left(\sqrt{110}-\sqrt{100}\right)^2 = 0.48$】；
  在 YOLOV3 中，用以下方法控制不同大小的目标对应的权重：

```
box_loss_scale[b, k, j, i] = batch_target[t, 2] * batch_target[t, 3] / in_w / in_h
box_loss_scale = 2 - box_loss_scale
```

  以上是计算的回归损失【与性能评估指标存在不一致性】，现阶段多采用 IOU 及其变体进行计算。这样做也更加有利于目标检测【大多数对象是相对于图像本身而言的小目标体】，具体详见附录。

- 置信度损失：可以理解为粗划分【二分类】，另外由于检测框输出结果存在明显的类别失衡问题【大部分为负例】，在2.2.4节改 BCE 为 Focal loss。

- 类别预测损失

附：IoU 及其变体的发展

- IoU

$$IoU = \frac{|A \cap B|}{|A \cup B|}, IoU \in [0,1] \tag{9}$$

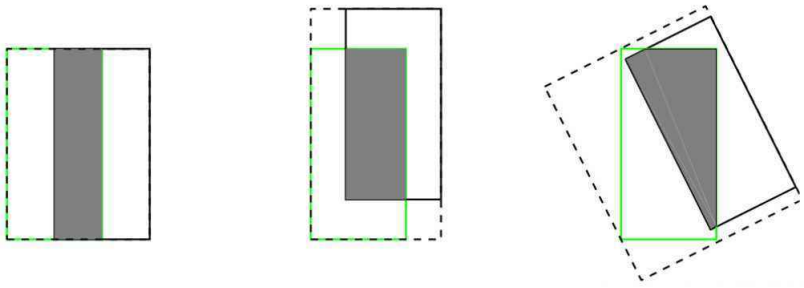


图 8: 相同的 IoU 对应不同的重叠方式。绿色方框为真实框；灰色区域为重叠区域；黑色方框为预测框

> **提醒**：
>
> – 可以反映预测框和真框的相似度，满足尺度不变性【对尺度不敏感】；
>
> – 如果两个框没有相交【IoU=0】，影响梯度回传；
>
> – 无法精确的反映两个框的重合度大小图8

- GIoU

$$GIoU = IoU - \frac{|C - (A \cup B)|}{|C|}, GIoU \in [-1, 1] \tag{10}$$

其中，$|C|$ 为预测框和真实框的最小闭包区域面积；$|C - (A \cup B)|$ 为不属于两个框的区域。当且仅当真实框和预测框重合时，$GIoU = 1$；不存在交集时，$GIoU = -1$。

- DIoU

$$DIou = Iou - \frac{\rho^2(b, b^{gt})}{c^2} \tag{11}$$

其中，b，b^{gt} 分别为预测框和真实框的中心点；ρ 为欧式距离函数；c 为预测框和真实框的最小闭包区域的对角线距离。

> **提醒**：
>
> – 和 GIou 一样，当预测框和真实框不存在重叠时，仍然可以正常梯度下降；
>
> – 可以直接衡量预测框和真实框的距离；
>
> – 当预测框和真实框为包含状态时，GIoU 退化为 IoU，DIoU 仍**有可能**继续更新梯度；
>
> – DIoU 也可以应用于 NMS 中。

- CIoU

$$CIoU = IoU - \frac{\rho^2(b, b^{gt})}{c^2} - \alpha v \tag{12}$$

$$v = \frac{4}{\pi^2}\left(\arctan\frac{w^{gt}}{h^{gt}} - \arctan\frac{w}{h}\right)^2 \tag{13}$$

$$\alpha = \begin{cases} 0, & if \quad IoU < 0.5 \\ \frac{v}{1 - IoU + v}, & if \quad IoU \geq 0.5 \end{cases} \tag{14}$$

提醒：

- 当预测框和真实框重叠较少时，即 $IoU < 0.5$，CIoU 退化为 DIoU；
- 当 $IoU \geq 0.5$ 时，预测框和真实框的中心点存在重合的可能性，因此需要引入纵横比进一步调整预测框，依据于：
- 当预测框和真实框的纵横比不一致时，v 会比较大，可以继续保持 loss 的梯度。

- EIoU

$$EIoU = IoU - \frac{\rho^2(b, b^{gt})}{c^2} - \frac{\rho^2(w, w^{gt})}{C_w^2} - \frac{\rho^2(h, h^{gt})}{C_h^2} \tag{15}$$

其中，C_w 和 C_h 分别为覆盖预测框和真实框的最小外接框的宽度和高度。

提醒：

- CIoU 无法很好的度量预测框的宽和高具体的回归程度；
- EIoU 还结合了 Focal loss【$L_{EIoU} = IoU^{\gamma}(1 - EIoU)$】，**来降低与目标重叠较少的大量锚对回归框损失的贡献**。

```
......
iou = inter / union
if CIoU or DIoU or GIoU:
    cw = torch.max(b1_x2, b2_x2) - torch.min(b1_x1, b2_x1)  # 最小外接框的宽
    ch = torch.max(b1_y2, b2_y2) - torch.min(b1_y1, b2_y1)  # 最小外接框的高
    if CIoU or DIoU:
        c2 = cw**2 + ch**2 + eps
        rho2 = ((b2_x1 + b2_x2 - b1_x1 - b1_x2)**2 + (b2_y1 + b2_y2 - b1_y1 - b1_y2) ** 2)/4
        if CIoU:
            v = (4/math.pi**2) * torch.pow(torch.atan(w2/(h2 + eps)) - torch.atan(w1/(h1 + eps)), 2)
            with torch.no_grad(): alpha = v / (v - iou + (1 + eps))
            return iou - (rho2 / c2 + v * alpha)
        return iou - rho2 / c2
    c_area = cw * ch + eps
    return iou - (c_area - union) / c_area
return iou
```

- α-IoU

$$\begin{aligned} L_{\alpha-IoU} &= 1 - IoU^{\alpha} \\ L_{\alpha-GIoU} &= 1 - IoU^{\alpha} + \left(\frac{|C - (A \cup B)|}{|C|}\right)^{\alpha} \\ &\cdots \end{aligned} \tag{16}$$

> **提醒**：
>
> – $\alpha = 3$ 时，增加了 high IoU 目标的损失和梯度，从而提高了回归框的精度【直观来看，添加幂次运算，可以放大损失的敏感度】【测试了一下，没啥用】；
>
> – 不会引入额外的参数，也不会增加训练或推理的时间。

- SIoU【Submitted on 25 May 2022】额外考虑了回归框与真实框之间的向量角度

  动机：迄今为止，提出的方法都没有考虑到真实框与预测框之间的匹配方向，进而导致训练收敛速度慢，预测框在训练的过程中“四处游荡”。

  具体请参考AI 视觉网奇

*** Focal loss**

用于处理目标检测中正负样本比例失衡的问题。【一张图像可以有成千上万个候选框，但是只有很少的一部分包含目标】**【也可以处理 long-tail 问题】**

以二分类为例：

$$CE = -ylogy' - (1-y)log(1-y') = \begin{cases} -logy', & y = 1 \\ -log(1-y'), & y = 0 \end{cases} \tag{17}$$

其中，$y' \in (0,1)$ 表示输出类别对应的概率。

通常会在交叉熵的前面加一个参数 α 控制正负样本对 loss 的贡献，即

$$CE = \begin{cases} -\alpha logy', & y = 1 \\ -(1-\alpha)log(1-y'), & y = 0 \end{cases} \tag{18}$$

此外，【当某类样本的概率越大，也就也意味着更容易被识别为该样本】。改进式 (18)：

$$FL = \begin{cases} -\alpha(1-y')^{\gamma} logy', & y = 1 \\ -(1-\alpha)y'^{\gamma} log(1-y'), & y = 0 \end{cases} \tag{19}$$

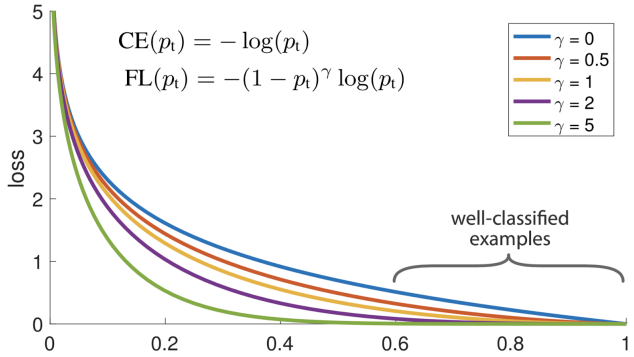


图 9: 正样本概率

由图9不难发现【仔细看图】：y' 越大，FL 越小，它对总体的 FL 的贡献就越小。【此时标签为 1】**【降低高置信度样本的损失**，通俗地讲就是，容易被识别的类别就只要求预测概率超过 0.5 就可以了，不用过度追求预测概率为 1，应该更加关注难以分类的样本】

```
def forward(self, pred, true):
    loss = self.loss_fcn(pred, true)
    pred_prob = torch.sigmoid(pred)  # prob from logits
    p_t = true * pred_prob + (1 - true) * (1 - pred_prob)
    alpha_factor = true * self.alpha + (1 - true) * (1 - self.alpha)
    modulating_factor = (1.0 - p_t) ** self.gamma
    loss *= alpha_factor * modulating_factor
    if self.reduction == 'mean': return loss.mean()
    elif self.reduction == 'sum': return loss.sum()
    else: return loss
```

2.2.4 YOLOV4

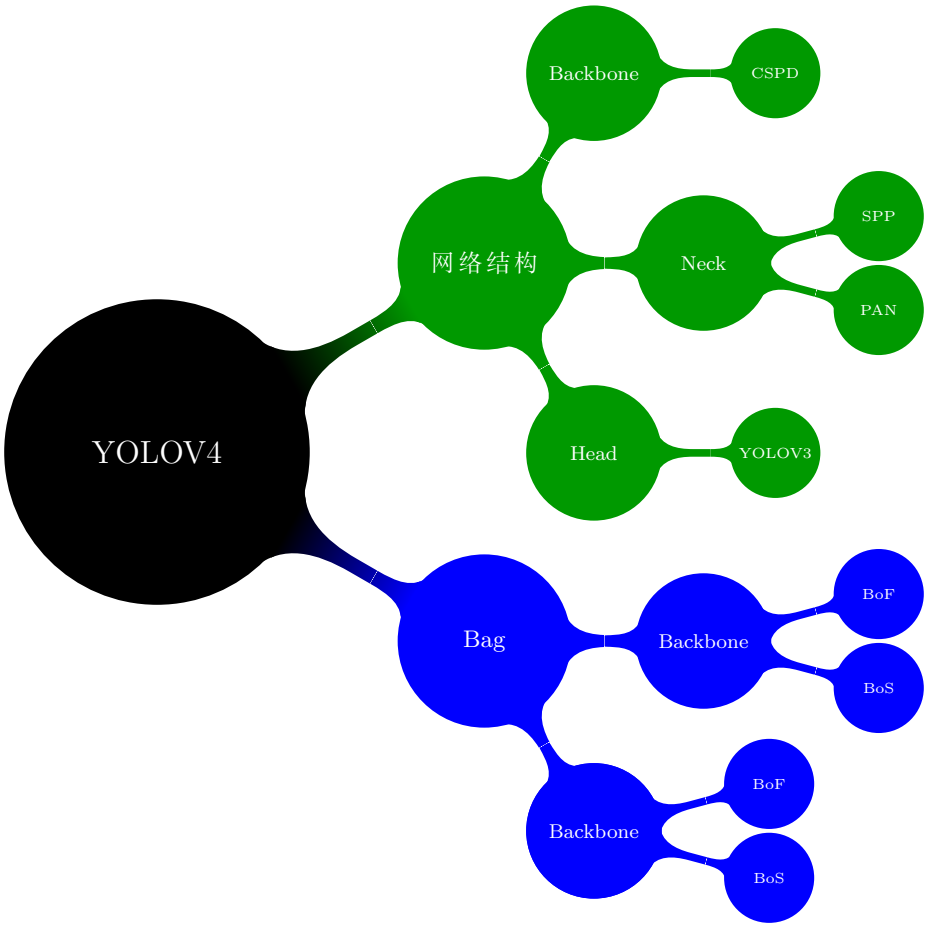


图 10: YOLOV4 的主要结构

CSPDarknet53：Cross Stage Paritial Network (CSP)：【网上写得太花哨】，可以参考 GoogLeNet，优点可以总结为两点：

> 提醒：
>
> - 特征复用，强化学习；
> - 降低内存成本

稀疏化训练：稀疏度训练有助于通道修剪，以删除不重要的通道。具体地，除检测头以外，YOLOV4 中的每个卷积层后面都有一个 BN 层，用于加速收敛。

$$y = \gamma \times \frac{x - \overline{x}}{\sqrt{\sigma^2 + \epsilon} + \beta} \tag{20}$$

其中，$\overline{x}$ 和 σ^2 是小批量的输入特征的均值和方差，γ 和 β 表示可训练的缩放因子和偏置。对缩放因子 γ 施加 L_1 正则化即可实现稀疏化训练：

$$loss = loss_{yolov4} + \alpha \sum_{\gamma} f(\gamma) \tag{21}$$

其中，$f(\gamma) = |\gamma|$ 表示 L_1 范数。

2.2.5 YOLOV5

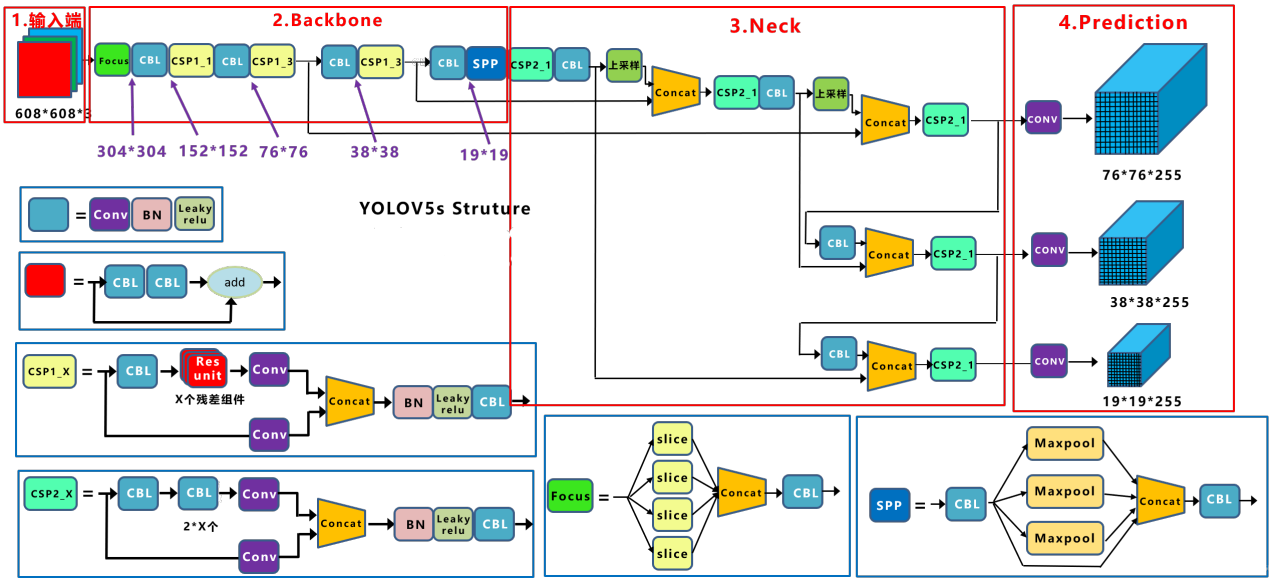


图 11: YOLOV5S 结构图

图像加权采样

1. 读取 ground truth，统计训练样本中不同类别的矩形框个数；

```
def labels_to_class_weights(labels, nc):
    if labels[0] is None: return torch.Tensor()
```

```
        labels = np.concatenate(labels, 0)
        classes = labels[:, 0].astype(int)
        weights = np.bincount(classes, minlength=nc)
        weights[weights == 0] = 1
        weights = 1 / weights # 标签越多，权重越少
        weights /= weights.sum() # 赋予每个类别的权重为该类别的目标框数
        return torch.from_numpy(weights).float()
```

2. 结合类别权重，计算训练样本中各个输入图片的权重；

```
def labels_to_image_weights(labels, nc, class_weights=np.ones(nc)):
    weights = image_weights, k=1)  # weighted image sample
    class_counts = np.array([np.bincount(x[:, 0].astype(int), minlength=nc) for x in labels])
    return (class_weights.reshape(1, nc) * class_counts).sum(1) # 图片中每个标签的权重之和即为图片的
        权重
```

3. 依据图片权重采样，训练

```
......
for epoch in range(start_epoch, epochs):
    ......
    # 持续更新
    if opt.image_weights:
        # 初始化maps = np.zeros(nc)
        # 标签数量越多，且标签识别精度越高，赋予的标签的权重越小
        cw = model.class_weights.cpu().numpy() * (1 - maps) ** 2 / nc
        iw = labels_to_image_weights(dataset.labels, nc=nc, class_weights=cw)  # 图像权重
        dataset.indices = random.choices(range(dataset.n), weights=iw, k=dataset.n)
    ......
```

3 后处理

动机：在执行目标检测任务时，可能对同一目标有多次检测，因此有必要“清理检测”，使得每个对象只得到一个检测。

3.1 传统 NMS

非极大值抑制（Non-Maximum Suppressin，NMS）局部最大搜索

Algorithm 1: 非极大值抑制

输入： 候选框集合 $\mathcal{B}$ 及其对应的置信度 $\mathcal{S}$；阈值 N_t【常用 0.3-0.5】

最终检测结果 $\mathcal{D} \leftarrow \{\}$

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

步骤一，从 $\mathcal{B}$ 中选择最大 score 的检测框 m；

步骤二，将其从 $\mathcal{B}$ 中移除并加入到最终检测结果 $\mathcal{D}$ 中；

步骤三，将 $\mathcal{B}$ 中剩余检测框 b_i 与 m 的 IOU 大于 N_t 的框从 $\mathcal{B}$ 中移除；

end

输出： 最终检测结果

提醒：

- 在使用 NMS 之前需要预选框预处理【NMS 耗时】；
- 将相邻检测框的分数强制归零，当一个真实物体在重叠区域出现时，会致使该目标检测失败，降低平均检测率；
- 阈值选取过于主观；
- 只能 CPU 计算

3.2 多类别 NMS

提醒：

- 1的做法是把所有 boxes 放在一起做 NMS，没有考虑类别【某一类的 boxes 不应该因为它与另一类最大得分 boxes 的 iou 值超过阈值而被筛掉】
- 实现方式：为每个类别单独做 NMS【给 box 坐标添加一个偏移量，偏移量由类别索引决定】

3.3 Soft-NMS

不像 NMS 一样粗暴的删除 IOU 大于阈值的检测框，Soft-NMS 衰减分数。区别在于步骤三，前者：

$$s_i = \begin{cases} s_i, & iou\left(m, b_i\right) < N_t \\ 0, & iou\left(m, b_i\right) \geq N_t \end{cases} \tag{22}$$

后者有两种实现形式：

- 线性加权

$$s_i = \begin{cases} s_i, & iou\left(m, b_i\right) < N_t \\ s_i\left(1 - iou\left(m, b_i\right)\right), & iou\left(m, b_i\right) \geq N_t \end{cases} \tag{23}$$

- 高斯加权

$$s_i = s_i e^{-\frac{iou(m,b_i)^2}{\sigma}}, \forall b_i \notin \mathcal{D} \tag{24}$$

提醒：

- 不需要重新训练原有的模型，二值化时等价于 NMS；
- $\mathcal{B}$ 为空后，所有的候选框都还在，只是置信度发生了变化，还需要阈值处理一次。

3.4 Softer-NMS

动机： NMS 仅使用了置信度得分，没法反映出候选框的定位精确度【分类置信度和定位置信度非正相关】Softer-NMS 引入了两个假设：

- 假设 1：候选框的定位置信度（x_e）是高斯分布；

$$P_\Theta(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-x_e)^2}{2\sigma^2}} \tag{25}$$

- 假设 2：真实框的定位置信度（x_g）是狄拉克分布，即标准方差为 0 的高斯分布极限

$$P_D(x) = \delta(x - x_g) \tag{26}$$

用 KL 散度来衡量两个分布的相似度：

$$\begin{aligned} L_{reg} &= \arg\min_\Theta D_{KL}\left(P_D(x) || P_\Theta(x)\right) && (27)\\ &= \int P_D(x) \log P_D(x) dx - \int P_D(x) \log P_\Theta dx && (28)\\ &= -\log P_\Theta(x_g) + C_1 && (29)\\ &= \frac{(x_g - x_e)^2}{2\sigma^2} + \frac{\log \sigma^2}{2} + C_2 && (30) \end{aligned}$$

其中 C_1，C_2 均为常量。**根据链式法则**分别对 x_e 和 σ 求偏导：

$$\begin{aligned} \frac{dL_{reg}}{dx_e} &= \frac{x_e - x_g}{\sigma^2} && (31)\\ \frac{dL_{reg}}{d\sigma} &= -\frac{(x_e - x_g)^2}{\sigma^3} + \frac{1}{\sigma} && (32) \end{aligned}$$

设定 $\alpha = \log(\sigma^2)$【动机和2.2.3节中回归框长宽缩放到对数空间一样】，此时

$$L_{reg} \propto \frac{e^{-\alpha}}{2}(x_g - x_e)^2 + \frac{1}{2}\alpha \tag{33}$$

提醒：

- σ 是网络预测的标准方差；
- 狄拉克函数具体挑选性：$\int f(x)\delta(x - t_0)dx = f(t_0)$【见公式 (28)】；
- 可以理解为所有 $iou\left(m, b_i\right) \geq N_t$ 的候选框预测坐标的加权平均。

3.5 Weighted-Boxes-Fusion, WBF

提醒：

- 与单一模型相比，结合多个模型的预测会更好
- NMS 和 soft-NMS 只是简单地删除部分预测框
- 比 NMS 慢 3 倍

具体的流程，可总结为：维护两个容器 Clusters 和 Fusions【一个 Fusion 对应一簇 Cluster】，详细步骤如下：

Algorithm 2: WBF

输入： 候选框集合 $\mathcal{B}$，按其对应的置信度 $\mathcal{S}$ 降序排列；检测结果容器 $\mathcal{L}$ 和融合容器 $\mathcal{F}$，阈值 N_t

最终检测结果 $\mathcal{D} \leftarrow \{\}$

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

步骤一，按顺序从 $\mathcal{B}$ 中取出一个框；

步骤二，从 $\mathcal{F}$ 中找到与之匹配的框，要求属于同一类别且 mIoU $> N_t$。如果没有找到，则将其加到 $\mathcal{L}$ 和 $\mathcal{F}$ 的尾部；反之，则将其加到 $\mathcal{L}$，加入位置为其在 $\mathcal{F}$ 中匹配到的位置；

步骤三，重新计算 $\mathcal{F}$ 中的 box 坐标和置信度分数，融合 $\mathcal{L}$ 中的所有 box，融合公式为：

$$\mathcal{S} = \frac{\sum_{i=1}^{T} \mathcal{S}_i}{T} \tag{34}$$

$$\mathcal{X}1,2 = \frac{\sum_{i=1}^{T} \mathcal{S}_i * \mathcal{X}1,2_i}{\sum_{i=1}^{T} \mathcal{S}_i} \tag{35}$$

$$\mathcal{Y}1,2 = \frac{\sum_{i=1}^{T} \mathcal{S}_i * \mathcal{Y}1,2_i}{\sum_{i=1}^{T} \mathcal{S}_i} \tag{36}$$

步骤四，将 $\mathcal{B}$ 中所有的框都处理完后，重新缩放 $\mathcal{F}$ 中的置信度分数：

$$\mathcal{C} = \mathcal{C} * \frac{min\,(T, N)}{N} \tag{37}$$

如果簇中的框数少，则意味着只有少数模型可以预测它，因此需要降低这种情况的置信度分数

end

输出： 最终检测结果

4 半监督学习

4.1 Softer Teacher[2]

FixMatch 提出了一个经典的数据增广方式来做 SSL：未标注的数据使用弱数据增强图像打伪标签，增强图像训练网络。

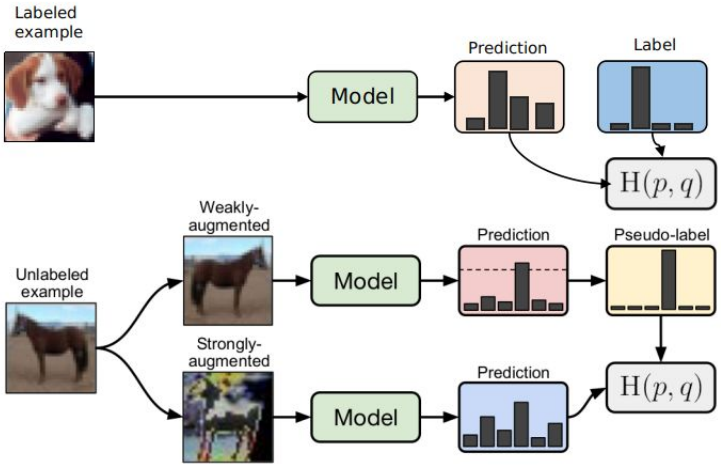


图 12: 经典的半监督学习方式

受 FixMatch 启发，STAC：

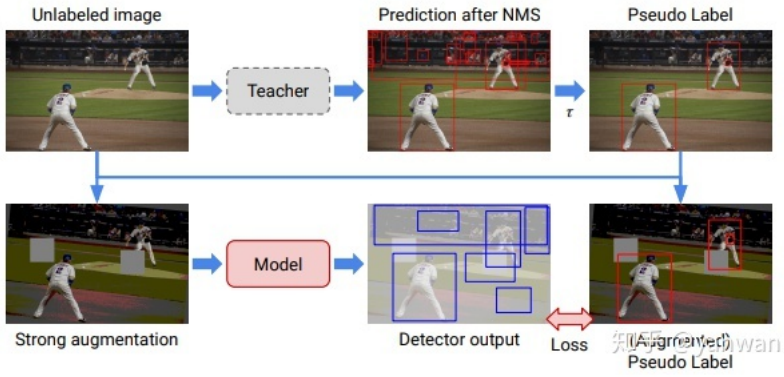


图 13: Caption

Softer Teacher 是一种端到端的半监督目标检测方法:

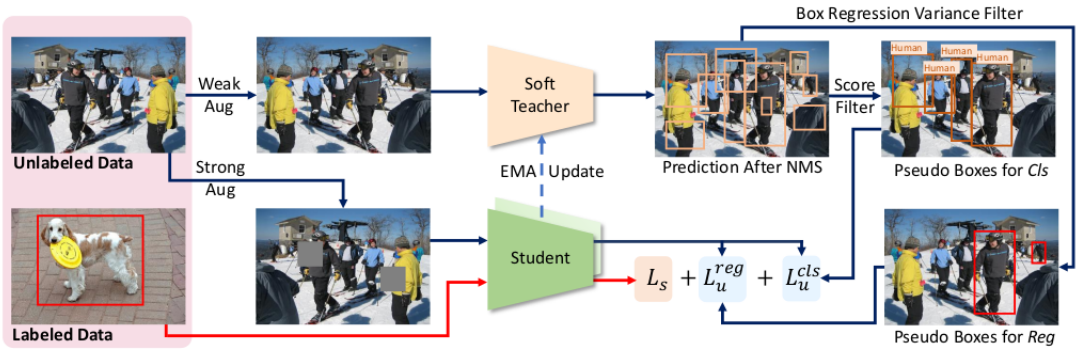


Figure 2. The overview of the end-to-end pseudo-labeling framework for semi-supervised object detection. Unlabeled images and labeled images form the training data batch. In each training iteration, a soft teacher is applied to perform pseudo-labeling on weak augmented unlabeled images on the fly. Two sets of pseudo boxes are produced: one is used for classification branch by filtering boxes according to the foreground score, and the other is used for box regression branch by filtering boxes according to box regression variance. The teacher model is updated by student model via exponential mean average (EMA) manner. The final loss is the sum of supervised detection loss L_s and unsupervised detection loss L_u.

图 14: Caption

提醒： 现有半监督学习方法的缺点：初始检测器的性能影响伪标签的质量，会影响最终的性能；本文的两个创新点：

- 未标注边界框的分类损失由 teacher 网络产生；
- 框抖动方法【以框回归的方差作为定位可靠性度量】，用于选择可靠的伪框

损失函数由监督损失和无标注数据的无监督损失组成：

$$L = L_s + \alpha L_u \tag{38}$$

为了解决较高阈值导致的前景框被错分为背景类别的问题，Softer teacher 通过对所有背景样本评估其真实背景的可靠性，让可靠性低的背景样本损失更低：

$$L_u^{cls} = \frac{1}{N_b^{fg}} \sum_{i=1}^{N_b^{fg}} l_{cls}\left(b_i^{fg}, G_{cls}\right) + \sum_{j=1}^{N_b^{bg}} w_j l_{cls}\left(b_j^{bg}, G_{cls}\right),\ w_j = \frac{r_j}{\sum_{k=1}^{N_b^{bg}} r_k} \tag{39}$$

一般地，通过 teacher 模型的预测得分就可以确定伪候选框的类别和坐标。事实上，前景得分区分正负样本与最后的回归准确度并无正向关系。因此本文通过度量回归预测的一致性来估计伪候选框的定位可靠性。具体地，将 teacher 模型预测的前景得分大于 0.5 的框多次抖动，然后继续送入 teacher 模型进行回归，计算抖动框的回归方差，方差越小，定位可靠性越高。

$$\overline{\sigma}_i = \frac{1}{4} \sum_{k=1}^{4} \hat{\sigma_k},\ \hat{\sigma_k} = \frac{\sigma_k}{0.5\left(h(b_i) + w(b_i)\right)} \tag{40}$$

参考文献

[1] S. Ren, K. He, R. Girshick & J. Sun, Faster R-CNN: Towards real-time object detection with region proposal networks, *Computer Vision and Pattern Recognition*, 2015.

[2] M. Xu, Z. zhang, H., Hu, et al. End-to-End semi-supervised object detection with soft teacher, *Computer Vision and Pattern Recognition*, 2021.