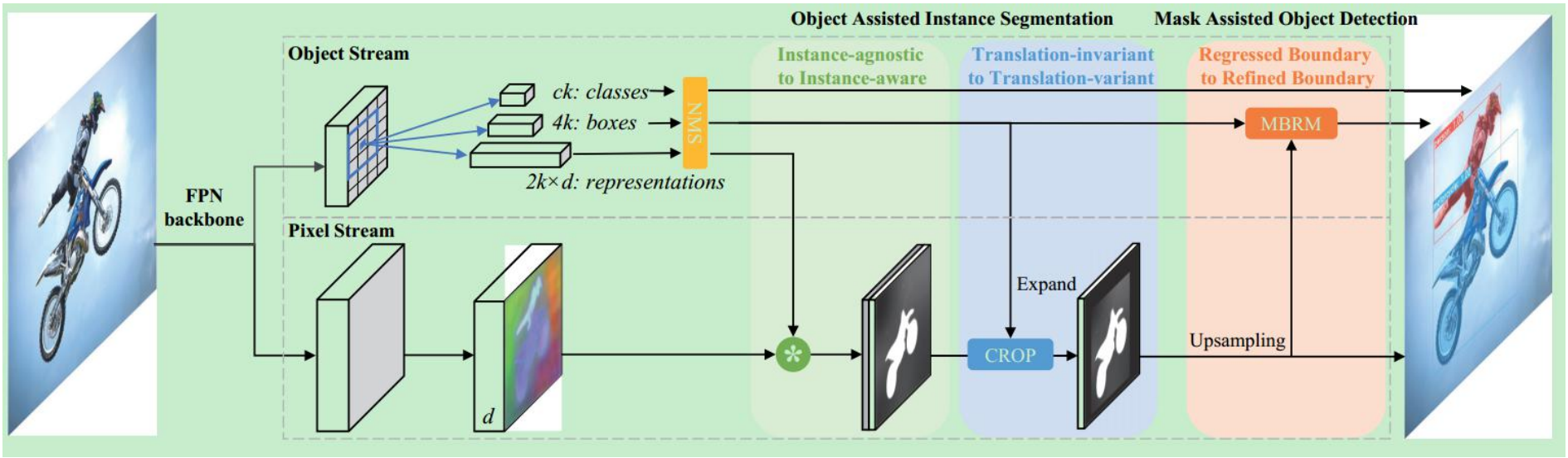
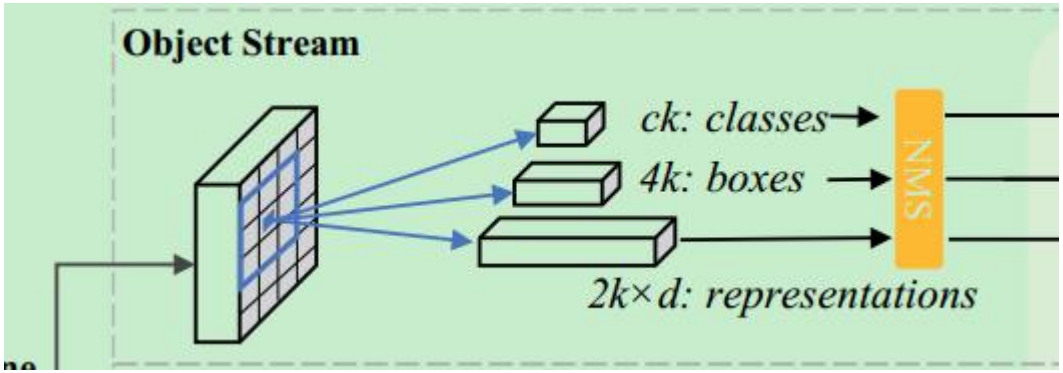
- RDSNet
- 关于带角度检测框的nms算法

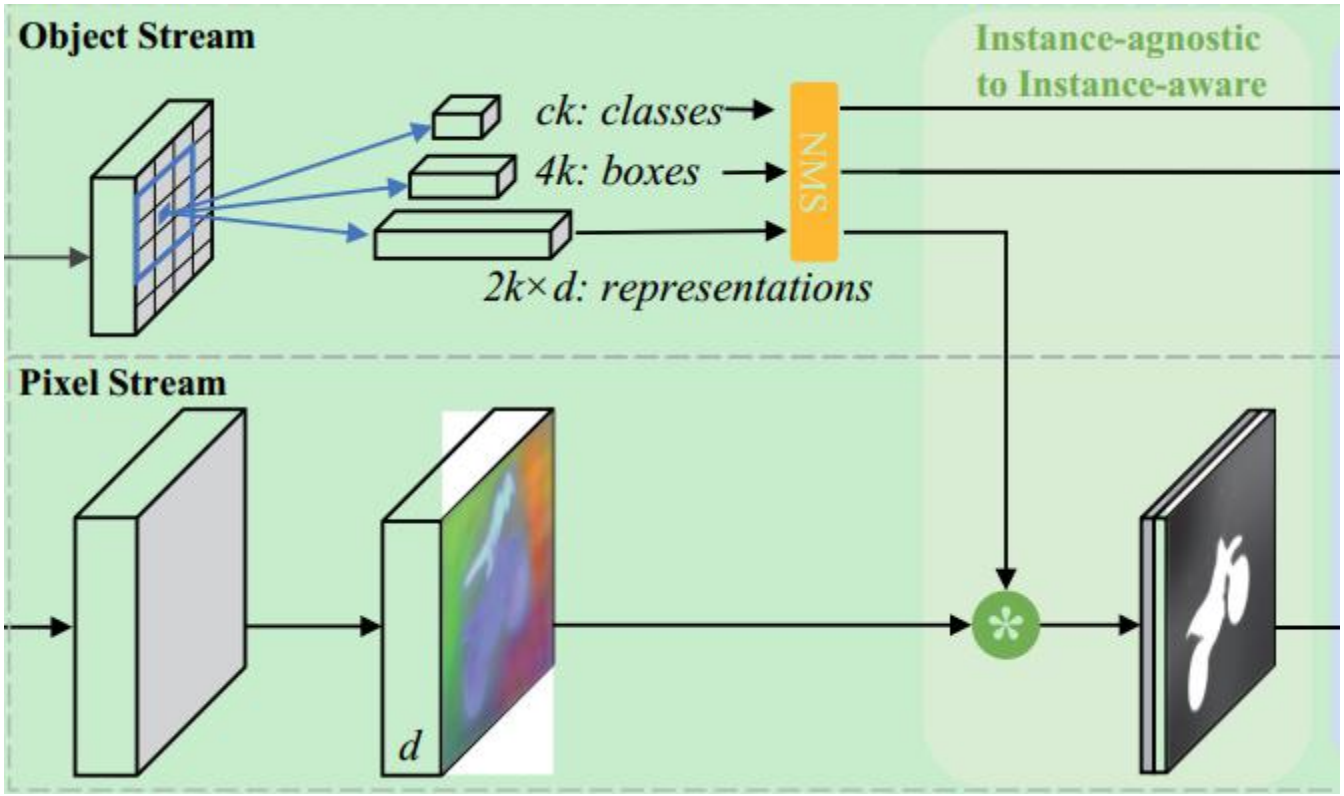

Object Assisted Instance Segmentation
Mask Assisted Object Detection
Object Stream
Instance-agnostic
to Instance-aware
Translation-invariant
to Translation-variant
Regressed Boundary
to Refined Boundary
ck: classes
4k: boxes
2k×d: representations
NMS
MBRM
FPN
backbone
Pixel Stream
d
Expand
CROP
Upsampling

- 检测流
- 分割流

检测流辅助分割

- c是类别
- k是在一个位置上的anchor个数
- d是分割流中的特征图通道数





- 对于每一个目标对应的1X1像素，将其分为前景和背景与分割的特征图进行卷积，之后在通道方向上进行softmax，得到特征图M，这个特征图M就是相应目标的mask

- 两种分割算法：一种proposal based，检测算法的扩展(Mask RCNN)，一种基于分割算法，先预测每个店的特征，然后通过聚类获得掩码，这种方法受限于聚类算法，性能有限。作者想直接获得每个簇的中心，这样可以抛弃聚类算法，直接进行end to end的训练。

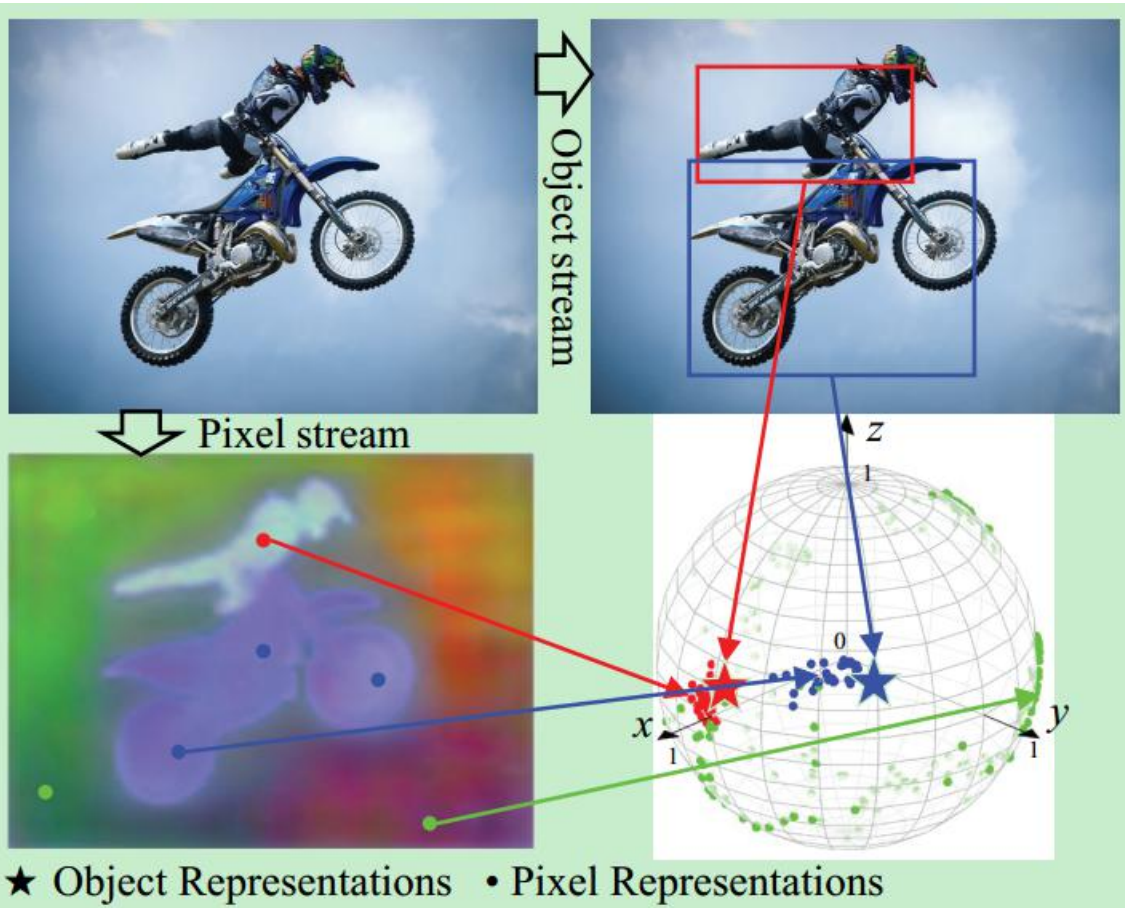


Figure 3: Illustration for the representations of the objects[2] and pixels, both of which are embedded into d-dimension feature space in the object and pixel streams respectively. Pixel representations are close to corresponding instance representation in feature space and different objects have distinct representations. Dimension reduction (from d to 3) and L2 normalization are performed to representations.

分割流辅助检测

- 将检测框的定位视为一个分类任务，例如对于检测框的左侧来说，将检测框的左侧在图像中的列数(行数)在整个图像中的位置视为一个分类问题。

$$x = \underset{i}{\operatorname{argmax}} P(X = i|M') \,, \tag{2}$$

where X is the discrete random variable for the horizontal coordinate of the left boundary, $M' \in \mathbb{R}^{h \times w}$ is the foreground channel of M in Eq. (1) up-sampled to the input image size, $h \times w$, with all the dimensions of size 1 removed, and $P(X = i|M')$ denotes the posterior probability given the corresponding instance mask M'.

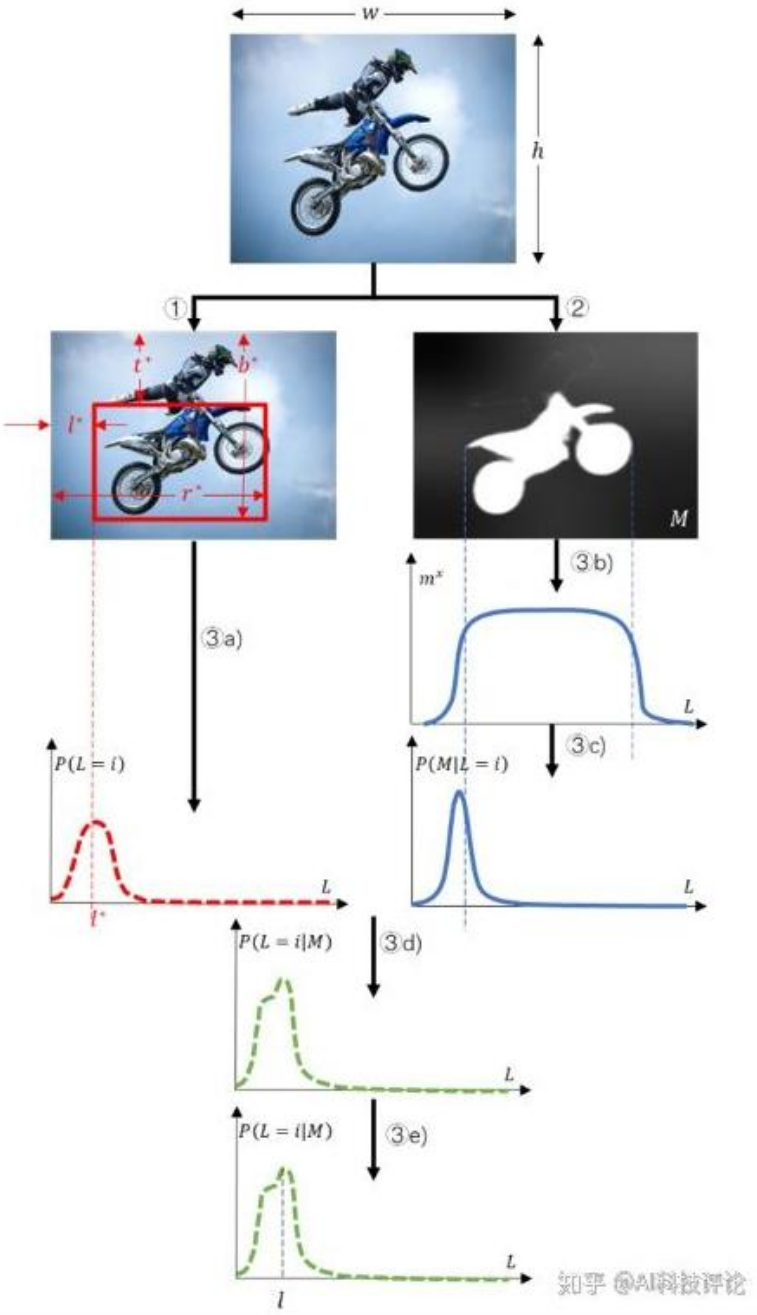
$$P(X=i|M') = \frac{P(X=i)P(M'|X=i)}{\sum_{t=1}^{w} P(X=t)P(M'|X=t)}, \quad (3)$$

Assuming that the boundary is only related to the maximum of each row in M', and it only affects its neighboring pixels, the likelihood probability can be defined as

$$P(M'|X=i) = P(m^x|X=i) \quad (4)$$

$$= P(m^x_{i-s,\dots,i+s}|X=i), \quad (5)$$

where $m^x_i = \max_{1\le j\le h} M'_{ij}$, and s is a hyper-parameter, describing the influence scope of the boundary on its neighboring pixels. Ideally, a pixel on the boundary only affects


w
h
①
②
t*
b*
l*
r*
M
③a)
③b)
③c)
③d)
③e)
m^x
L
$P(L = i)$
$P(M|L = i)$
$P(L = i|M)$
l^*
l
知乎 @AI科技评论

For $P(X = i)$, we simply adopt a discrete Gaussian distribution

$$P(X = i) = \alpha e^{-(i-\mu)^2/2\sigma_x^2} , \quad (6)$$

$$\mu = x_r , \ \sigma_x = \gamma w_b , \quad (7)$$

xr为检测流中回归出的左侧边界位置，wb为检测框的宽度，γ为回归边界的权重。

No.		Method	TE	OHEM	IE	FPS	AP^m
1	YOLACT	LC				33	29.9
2	$RDSNet_s$	Corr				32	$31.0^{+1.1}$
3					√		30.0
4			√				30.7
5			√		√		30.8
6			√	√			31.6
7			√	√	√		**31.8**$^{+1.9}$

Table 3: Demonstration of the effectiveness of the cropping module on COCO `val2017`. LC: Linear Combination, Corr: Correlation, TE: Expand during training, IE: Expand during inference. Our finally adopted choice (last row) yields the highest mAP. It should be noted that by using Corr instead of LC, RDSNet already outperforms YOLACT by 1.1 in mAP.

关于用于有角度检测框的nms算法

```python
def nms(pred_box, thresh):
    # 输入一张图片预测出的检测框，维度为：(N, 9),最后一个为预测的score
    # boxs = pred_box.copy()
    scores = pred_box[:, 8]
    order = scores.argsort()[::-1]

    keep = []
    while order.size > 0:
        boxs = pred_box[:, :8].copy()
        boxs = boxs[order]
        i = order[0]
        keep.append(i)

        ious = []
        max_box = boxs[0]
        for box in boxs[1:]:
            iou = intersection(max_box, box)
            ious.append(iou)
        ious = np.array(ious)
        index = np.where(ious <= thresh)[0]
        order = order[index+1]

    return keep
```

- 两层循环
- 使用了其他的包

1) The centers of both the ground truth rectangle and the oriented anchor box should be located in the same grid cell.
2) The difference between rotation angles of ground truth rectangle and matched oriented anchor box should be less than $90°/k$ (k is the anchor number in one cell).

```python
def nms(pred_box, thresh):
    # 输入一张图片预测出的检测框，维度为：(N, 9)，最后一个为预测的score
    # boxs = pred_box.copy()
    scores = pred_box[:, 8]
    order = scores.argsort()[::-1]

    keep = []
    while order.size > 0:
        boxs = pred_box[:, :8].copy()
        boxs = boxs[order]
        i = order[0]
        keep.append(i)

        ious = []
        max_box = boxs[0]
        for box in boxs[1:]:
            iou = intersection(max_box, box)
            ious.append(iou)
        ious = np.array(ious)
        index = np.where(ious <= thresh)[0]
        order = order[index+1]

    return keep
```