2019&2020 实例分割顶会论文阅读（一）

YOLACT：Real-time Instance Segmentation（ICCV2019）

1. Background

当前实例分割方面效果最好的是 Mask RCNN 和 FCIS：

1、Mask RCNN（增加了 Mask 预测分支，改进了 ROI，提出了 ROI Align）

2、FCIS（清华与微软合作提出的，是首个用于图像实例分割任务的全卷积、端到端的解决方案，该架构在 COCO 2016 图像分割竞赛中获得了第一名）

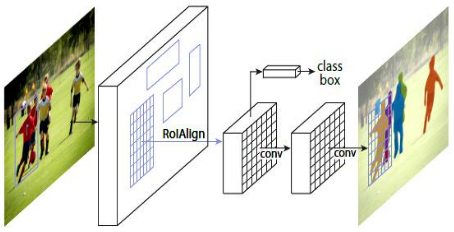


Figure 1. The **Mask R-CNN** framework for instance segmentation.

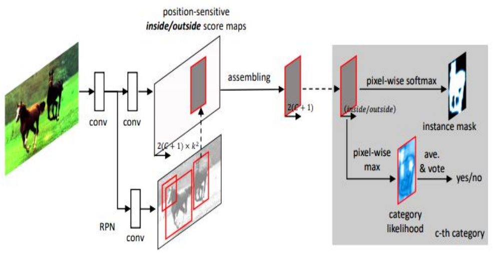
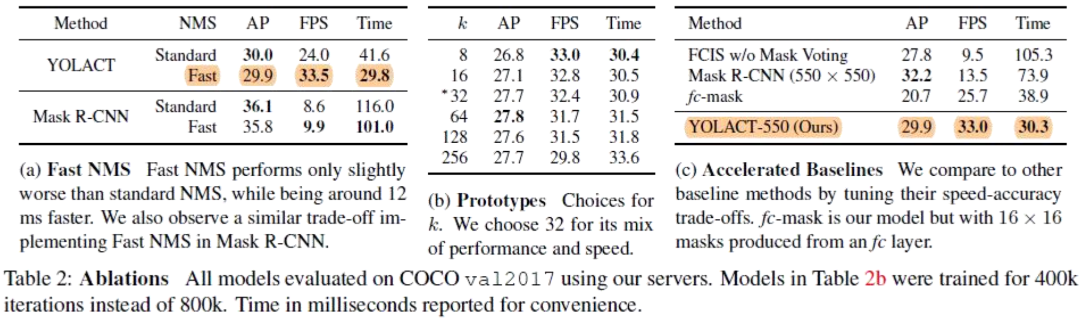


Figure 3. Overall architecture of FCIS. A region proposal network (RPN) [34] shares the convolutional feature maps with FCIS. The proposed region-of-interests (ROIs) are applied on the score maps for joint object segmentation and detection. The learnable weight layers are fully convolutional and computed on the whole image. The per-ROI computation cost is negligible.

2. Source idea

最新的两阶段实例分割方法在很大程度上依赖于特征定位来生成 Mask。 也就是说，这些方法（例如通过 RoIpool / align）在某些边界框区域中“re-pooling”特征，然后将这些定位后的特征送到 Mask predictor 中。其中每个 ROI 都需要进行 re-pooling features，并通过后续计算对其进行处理，这使得即使减小图像大小，它们也无法获得实时速度（30 fps）（请参见表 2c）。这种串行的运行方式同样很难加速。

虽然存在像 FCIS 一样并行执行这些步骤的一阶段方法，但是它们在定位之后需要大量的后处理，因此仍然远远不够实时。



Method	NMS	AP	FPS	Time
YOLACT	Standard	**30.0**	24.0	41.6
	Fast	29.9	**33.5**	**29.8**
Mask R-CNN	Standard	**36.1**	8.6	116.0
	Fast	35.8	**9.9**	**101.0**

(a) **Fast NMS** Fast NMS performs only slightly worse than standard NMS, while being around 12 ms faster. We also observe a similar trade-off implementing Fast NMS in Mask R-CNN.

k	AP	FPS	Time
8	26.8	**33.0**	**30.4**
16	27.1	32.8	30.5
*32	27.7	32.4	30.9
64	**27.8**	31.7	31.5
128	27.6	31.5	31.8
256	27.7	29.8	33.6

(b) **Prototypes** Choices for k. We choose 32 for its mix of performance and speed.

Method	AP	FPS	Time
FCIS w/o Mask Voting	27.8	9.5	105.3
Mask R-CNN (550 × 550)	**32.2**	13.5	73.9
fc-mask	20.7	25.7	38.9
YOLACT-550 (Ours)	29.9	**33.0**	**30.3**

(c) **Accelerated Baselines** We compare to other baseline methods by tuning their speed-accuracy trade-offs. *fc*-mask is our model but with 16 × 16 masks produced from an *fc* layer.

Table 2: **Ablations** All models evaluated on COCO `val2017` using our servers. Models in Table 2b were trained for 400k iterations instead of 800k. Time in milliseconds reported for convenience.

原文： **These two-stage methods require re-pooling features for each ROI and processing them with subsequent computations, which make them unable to obtain real-time speeds (30 fps) even when decreasing image size (see Table 2c).**

3. Subtasks

为了解决上述问题，研究人员提出了快速的、单阶段的实例分割模型，旨在将 Mask 分支添加到单阶段目标检测框架中，因此，作者将实例分割任务分解为两个更简单的并行任务，将其组合以形成最终的 Mask。

（1）第一个分支使用 FCN 生成一组图像大小的“原型掩码”(“prototype masks)，它们不依赖于任何一个实例。

（2）第二个向目标检测分支添加额外的 head 以预测用于编码原型空间中的实例表示的每个 anchor 的“掩模系数”（“mask coefficients）的向量。

（3）对经过 NMS 后的每个实例，通过线性组合这两个分支的工作来为该实例构造掩码。

原文：(1) generating a set of prototype masks and (2) predicting per-instance mask coefficients. Then we produce instance masks by linearly combining the prototypes with the mask coefficients.

1、The first branch uses an FCN [31] to produce a set of image-sized “prototype masks” that do not depend on any one instance.

2、The second adds an extra head to the object detection branch to predict a vector of “mask coefficients” for each anchor that encode an instance’s representation in the prototype space.

3、Finally, for each instance that survives NMS, we construct a mask for that instance by linearly combining the work of these two branches.

4. Advantages

- 更快（faster，because of its parallel structure and extremely light-weight assembly process. YOLACT adds only a marginal amount of computational overhead to a one-stage backbone detector, making it easy to reach 30 fps even when using ResNet-101）；

- 高质量的 Mask（high-quality mask no re-pooling, since the masks use the full extent of theimage space without any loss of quality from repooling）；

- 普适性强（general）

5. YOLACT　Architecture

结构组成：Prototype Generation + Mask Coefficients + Mask Assembly + Emergent Behavior + Backbone Detector

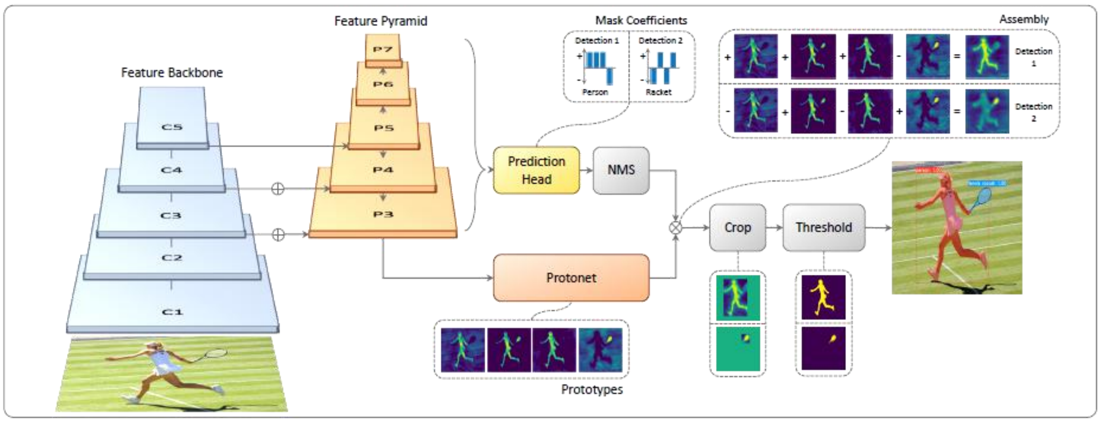


Figure 2: **YOLACT Architecture** Blue/yellow indicates low/high values in the prototypes, gray nodes indicate functions that are not trained, and $k = 4$ in this example. We base this architecture off of RetinaNet [27] using ResNet-101 + FPN.

Prototype Generation

模板产生的分支（protonet）针对每幅图像预测 k 个模板 mask，本文用 FCN 的方式来实现 protonet，FCN 最后一层有 k 个 channel，每个 channel 对应一个模板，且每个 channel 都被送入主要特征层。

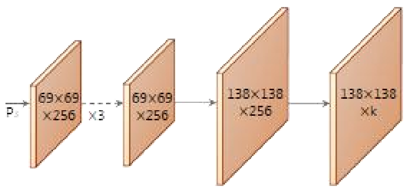


Figure 3: **Protonet Architecture** The labels denote feature size and channels for an image size of 550×550. Arrows indicate 3×3 *conv* layers, except for the final *conv* which is 1×1. The increase in size is an upsample followed by a *conv*. Inspired by the mask branch in [18].

Mask Coefficients

ProtoNet 预测了 k 个 mask，每一个 mak 都对应一个系数，就是在 prediction head 分支预测出来。需要注意的是，这个分支同样需要计算坐标值的回归值 4 个，类别概率 c 个，mask 的系数 k 个。就是说，一个 anchor 对应 4+c+k 个输出。

另外作者认为，为了能够通过线性组合来得到最终想要的 mask，能够从最终的 mask 中减去原型 mask 是很重要的。换言之就是，mask 系数必须有正有负。所以，在 mask 系数预测时使用了 tanh 函数进行非线性激活，因为 tanh 函数的值域是(-1,1).（**Then for non-linearity, we find it important to be able to subtract out prototypes from the final mask. Thus, we apply tanh to the k mask coefficients, which produces more stable outputs over no non-linearity**）

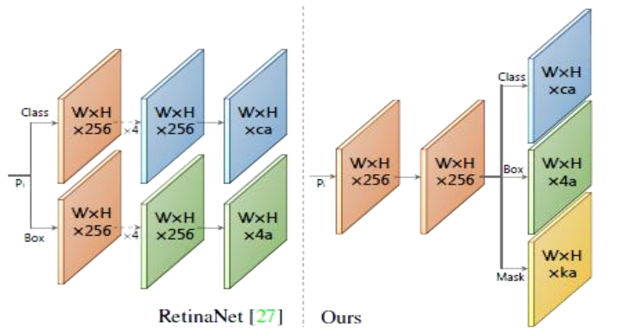


Figure 4: **Head Architecture** We use a shallower prediction head than RetinaNet [27] and add a mask coefficient branch. This is for c classes, a anchors for feature layer P_i, and k prototypes. See Figure 3 for a key.

Mask Assembly

通过基本的矩阵乘法配合 sigmoid 函数来处理两分支的输出，从而合成 mask。

$$M = \sigma(PC^T)$$

其中，P 是 h×w×k 的原型 mask 集合，C 是 n×k 的系数集合，代表有 n 个通过 NMS 和阈值过滤的实例，每个实例对应有 k 个 mask 系数。

A. Loss Function

Loss 由分类损失、框回归损失和 mask 损失三部分组成，其中分类损失和框回归损失同 SSD，mask 损失为预测 mask 和 ground truth mask 的逐像素二进制交叉熵，系数分别为 1, 1.5, and 6.125。

B. Mask Crop

为了改善小目标的分割效果，在推理时会首先根据检测框进行裁剪，再阈值化。而在训练时，会使用 ground truth 框来进行裁剪，并通过除以对应 ground truth 框面积来平衡 loss 尺度。

由于实例分割的特殊性，一张图有很多目标，如果把这些目标都在原图的大小计算损失函数，大的目标损失函数值之和较大，而小的目标损失函数值之和，因为像素数量不一样嘛，所以再求平均之后（除以像素总数），小目标的梯度相对于大目标就很小了，导致网络可能不重视小目标的分割。为了避免这个问题，作者的办法是：

• 在训练的时候，得到原图大小的 mask，按照 ground truth，把各个目标 mask 都 crop 出来，计算损失函数，再除以各自 crop 出来的区域像素总数。

• 在预测的时候，根据 bboxes 分支的目标预测框，把 mask crop 出来，再去和系数做线性组合，然后经过 sigmiod 和阈值分割。然和按照位置把得到的 mask 还原回原图位置上。

Emergent Behavior

实例分割任务有一个共识，FCNs 是平移不变，需要在模型中添加转移方差。因此，在 Mask R-CNN 和 FCIS 中，通过显式方法添加了转移方差：方向图、位置存档，或是把 mask 预测分支放在第二个 stage，都使得它们不需要再处理定位问题。在 YOLACT 中唯一添加转移方差的地方是使用预测框裁剪 feature map 时。但这只是为了改善对小目标的分割效果,作者发现对大中型目标，不裁剪效果就很好了。

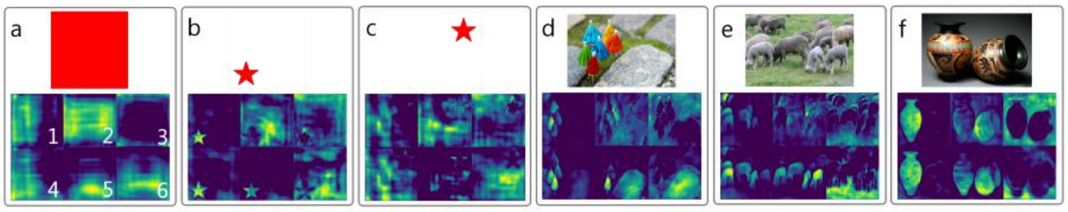


Figure 5: **Prototype Behavior** The activations of the same six prototypes across different images. Prototypes 1, 4, and 5 are partition maps with boundaries clearly defined in image a, prototype 2 is a bottom-left directional map, prototype 3 segments out the background and provides instance contours, and prototype 6 segments out the ground.

Backbone Detector

YOLACT 使用 ResNet-101 结合 FPN 作为默认主干网络，默认输入图像尺寸为 550×550。使用 Smooth-L1 loss 训练 bounding box 参数；使用 softmax 交叉熵训练分类部分，共(C+1)个类别。同时，使用 OHEM 方式选取训练样本，正负样本比例设为 1:3。

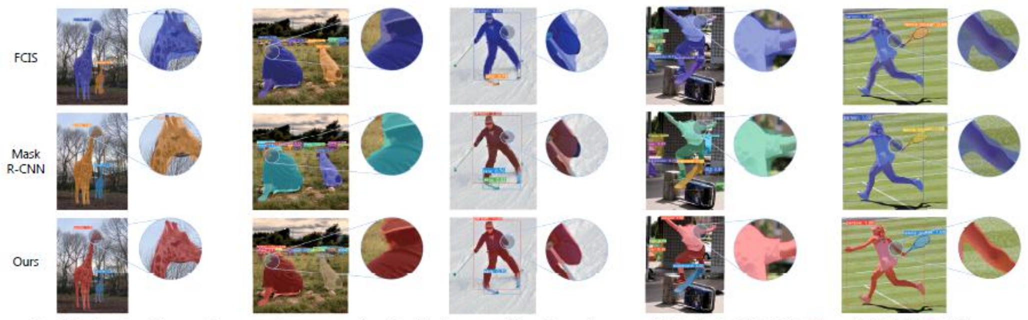


Figure 7: **Mask Quality** Our masks are typically higher quality than those of Mask R-CNN [18] and FCIS [24] because of the larger mask size and lack of feature repooling.

6. Other improvements

Fast NMS

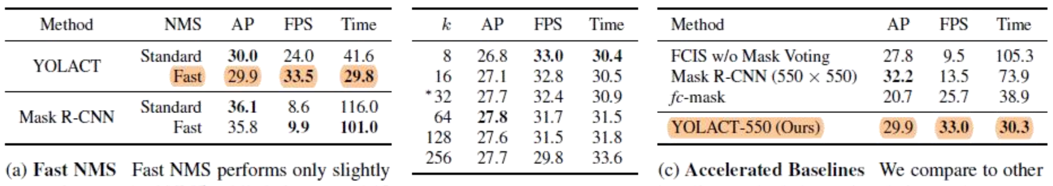
主要思想是将传统 NMS 计算方法转为矩阵运算，从而受益于一些快速矢量运算库。

算法流程：（1）对每一个类别 ci，取 top-n 个候选目标，并按得分降序排列；

（2）计算一个 c×n×n 的 IOU 矩阵，其中每个 n×n 矩阵表示对该类 n 个候选框，两两之间的 IOU；

（3）自己与自己的 IOU=1，IOU(A,B)=IOU(B,A)，所以对上一步得到的 IOU 矩阵进行一次处理。具体做法是将每一个通道，的对角线元素和下三角部分置为 0；

（4）去除与得分高的候选框重叠比例较大的框，具体做法是对上一步得到的矩阵，按列取最大值，然后对取完最大值的矩阵按阈值划分，只留下小于指定阈值的部分，即为 Fast NMS 过滤的结果。



Method	NMS	AP	FPS	Time
YOLACT	Standard	**30.0**	24.0	41.6
	Fast	29.9	**33.5**	**29.8**
Mask R-CNN	Standard	**36.1**	8.6	116.0
	Fast	35.8	**9.9**	**101.0**

(a) **Fast NMS** Fast NMS performs only slightly worse than standard NMS, while being around 12 ms faster. We also observe a similar trade-off implementing Fast NMS in Mask R-CNN.

k	AP	FPS	Time
8	26.8	**33.0**	**30.4**
16	27.1	32.8	30.5
*32	27.7	32.4	30.9
64	**27.8**	31.7	31.5
128	27.6	31.5	31.8
256	27.7	29.8	33.6

(b) **Prototypes** Choices for k. We choose 32 for its mix of performance and speed.

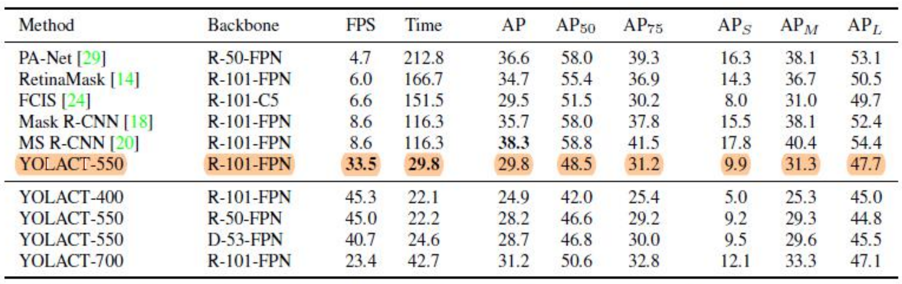
Method	AP	FPS	Time
FCIS w/o Mask Voting	27.8	9.5	105.3
Mask R-CNN (550 × 550)	**32.2**	13.5	73.9
fc-mask	20.7	25.7	38.9
YOLACT-550 (Ours)	29.9	**33.0**	**30.3**

(c) **Accelerated Baselines** We compare to other baseline methods by tuning their speed-accuracy trade-offs. *fc*-mask is our model but with 16 × 16 masks produced from an *fc* layer.

Table 2: **Ablations** All models evaluated on COCO `val2017` using our servers. Models in Table 2b were trained for 400k iterations instead of 800k. Time in milliseconds reported for convenience.

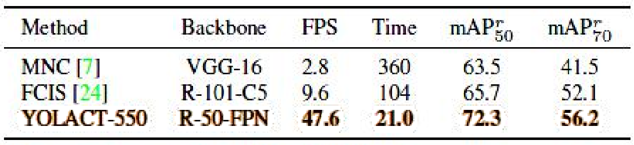
7. Result

（1）**29.8 mAP** on MS COCO **at 33.5 fps** evaluated **on a single Titan Xp**, which is significantly faster than any previous competitive approach.



Method	Backbone	FPS	Time	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
PA-Net [29]	R-50-FPN	4.7	212.8	36.6	58.0	39.3	16.3	38.1	53.1
RetinaMask [14]	R-101-FPN	6.0	166.7	34.7	55.4	36.9	14.3	36.7	50.5
FCIS [24]	R-101-C5	6.6	151.5	29.5	51.5	30.2	8.0	31.0	49.7
Mask R-CNN [18]	R-101-FPN	8.6	116.3	35.7	58.0	37.8	15.5	38.1	52.4
MS R-CNN [20]	R-101-FPN	8.6	116.3	**38.3**	58.8	41.5	17.8	40.4	54.4
YOLACT-550	R-101-FPN	**33.5**	**29.8**	29.8	48.5	31.2	9.9	31.3	47.7
YOLACT-400	R-101-FPN	45.3	22.1	24.9	42.0	25.4	5.0	25.3	45.0
YOLACT-550	R-50-FPN	45.0	22.2	28.2	46.6	29.2	9.2	29.3	44.8
YOLACT-550	D-53-FPN	40.7	24.6	28.7	46.8	30.0	9.5	29.6	45.5
YOLACT-700	R-101-FPN	23.4	42.7	31.2	50.6	32.8	12.1	33.3	47.1

Table 1: **MS COCO [28] Results** We compare to state-of-the-art methods for mask mAP and speed on COCO `test-dev` and include several ablations of our base model, varying backbone network and image size. We denote the backbone architecture with `network-depth-features`, where `R` and `D` refer to ResNet [19] and DarkNet [36], respectively. Our base model, YOLACT-550 with ResNet-101, is 3.9x faster than the previous fastest approach with competitive mask mAP.



Method	Backbone	FPS	Time	mAP^r_{50}	mAP^r_{70}
MNC [7]	VGG-16	2.8	360	63.5	41.5
FCIS [24]	R-101-C5	9.6	104	65.7	52.1
YOLACT-550	**R-50-FPN**	**47.6**	**21.0**	**72.3**	**56.2**

Table 3: **Pascal 2012 SBD [16] Results** Timing for FCIS redone on a Titan Xp for fairness. Since Pascal has fewer and easier detections than COCO, YOLACT does much better than previous methods. Note that COCO and Pascal FPS are not comparable because Pascal has fewer classes.

（2）作者认为 YOLACT 与 Mask R-CNN 的 AP 差距 6 个百分点的原因在于检测器性能不好，而不是在于他们生成 Mask 的方法。

（3）做了两个图片大小分别为 400x400,700x700 的对比实验，结果与预料的一样。小尺寸会降低性能，大尺寸会降低速度，因此折中 550x550。

8. Training Details

We train all models with batch size 8 on one GPU using ImageNet [10] pretrained weights.We find that this is a sufficient batch size to use batch norm, so we leave the pretrained batch norm unfrozen but do not add any extra bn layers. We train with SGD for 800k iterations starting at an initial learning rate of 10e-3 and divide by 10 at iterations 280k, 600k, 700k, and 750k, using a weight decay of 5x10e-4, a momentum of 0.9, and all data augmentations used in SSD [30].

For Pascal, we train for 120k iterations and divide the learning rate at 60k and 100k. We also multiply the anchor scales by 4/3, as objects tend to be larger. Training takes 4-6 days (depending on config) on one Titan Xp for COCO and less than 1 day on Pascal.

9. Contributions

（1） first real-time (> 30 fps) instance segmentation algorithm （2） provide experiments to study the speed vs. performance trade-offs obtained with different backbone architectures, numbers of prototypes, and image resolutions.（3）provide a novel Fast NMS approach that is 12ms faster than traditional NMS with a negligible performance penalty.

吕锋
2020.06.08