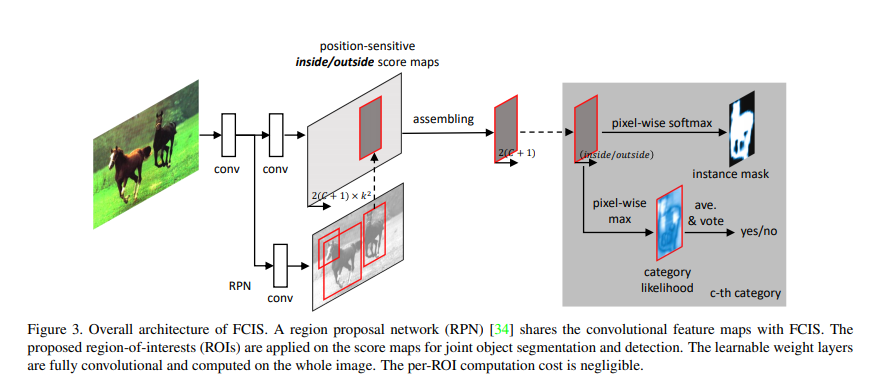
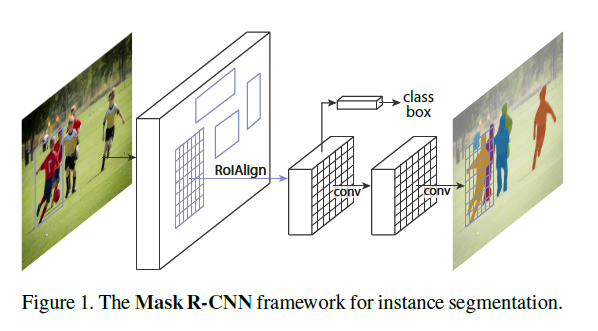
**2019&2020实例分割顶会论文阅读（一）**

**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：

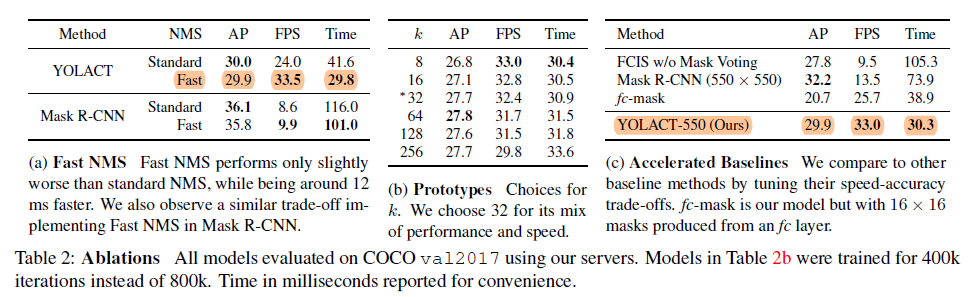
1. Mask RCNN（增加了Mask预测分支，改进了ROI，提出了ROI Align）
2. FCIS（清华与微软合作提出的，是首个用于图像实例分割任务的全卷积、端到端的解决方案，该架构在 COCO 2016 图像分割竞赛中获得了第一名）



**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。



**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）；

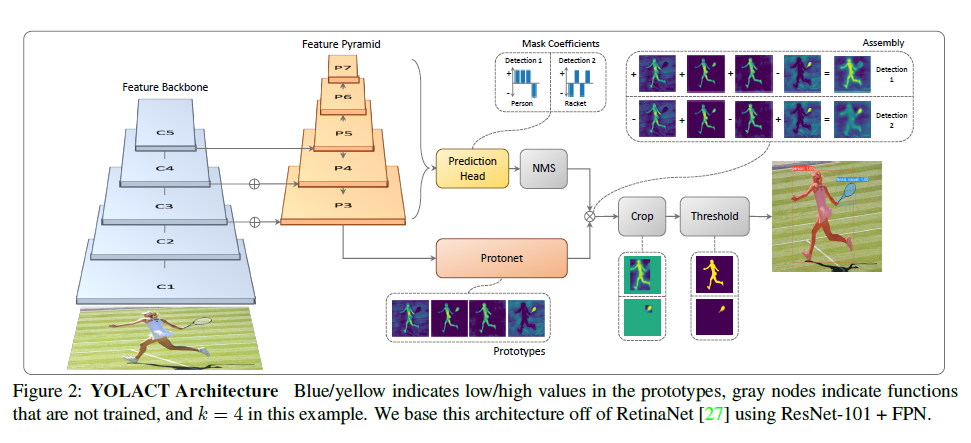
· 高质量的Mask（high-quality mask no re-pooling, since the masks use the full extent of theimage space without any loss of quality from repooling）；

· 普适性强（general）

**5. YOLACT Architecture**

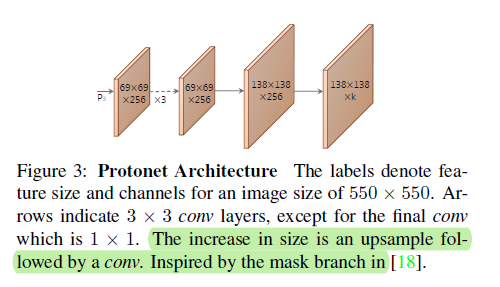
结构组成：Prototype Generation + Mask Coefficients + Mask Assembly

+ Emergent Behavior + Backbone Detector



**Prototype Generation**

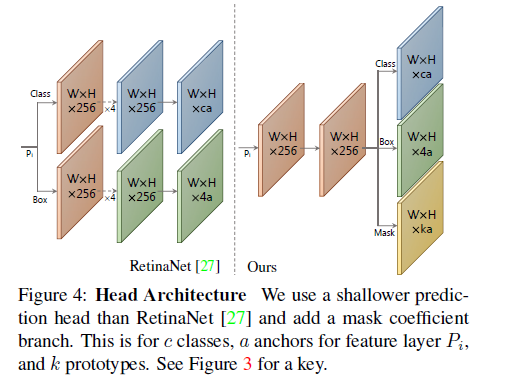
模板产生的分支（protonet）针对每幅图像预测k个模板mask，本文用FCN的方式来实现protonet，FCN最后一层有k个channel，每个channel对应一个模板，且每个channel都被送入主要特征层。



**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**）



**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。

IMG_256

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。

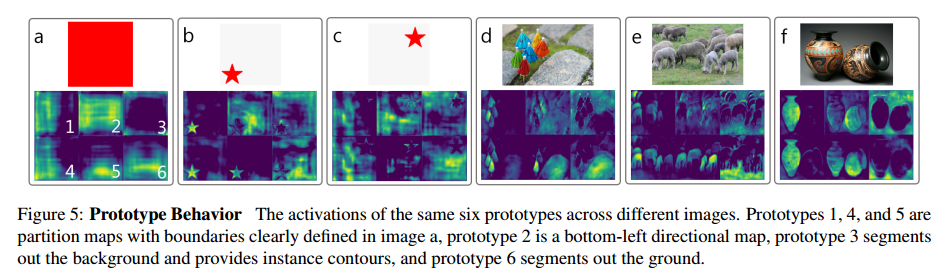
由于实例分割的特殊性，一张图有很多目标，如果把这些目标都在原图的大小计算损失函数，大的目标损失函数值之和较大，而小的目标损失函数值之和，因为像素数量不一样嘛，所以再求平均之后（除以像素总数），小目标的梯度相对于大目标就很小了，导致网络可能不重视小目标的分割。为了避免这个问题，作者的办法是：

·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。

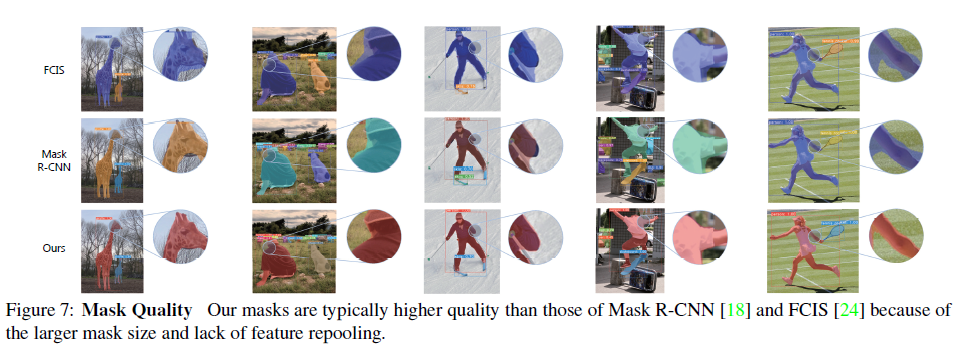
## **Emergent Behavior**

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。



**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。



**6. Other improvements**

**Fast NMS**

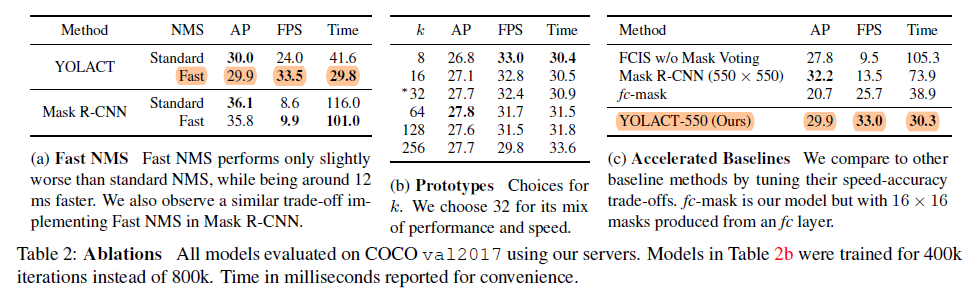
主要思想是将传统NMS计算方法转为矩阵运算，从而受益于一些快速矢量运算库。

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；

1. 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；

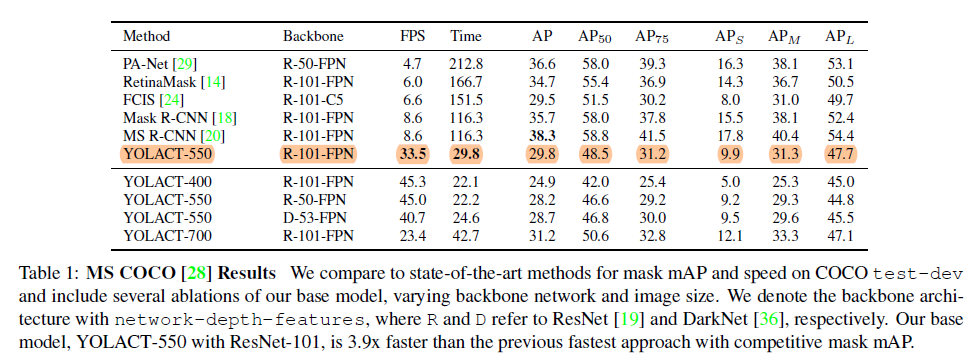
（4）去除与得分高的候选框重叠比例较大的框，具体做法是对上一步得到的矩阵，按列取最大值，然后对取完最大值的矩阵按阈值划分，只留下小于指定阈值的部分，即为Fast NMS过滤的结果。

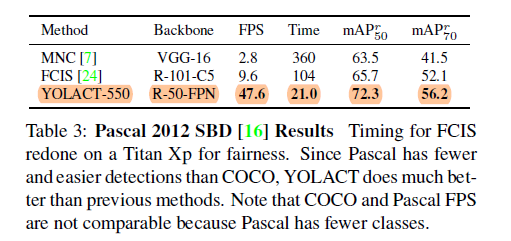


**7. Result**

**（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.





（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