

# 加州大学提出：实时实例分割算法YOLACT，可达33 FPS/30mAP！ 现已开源！

原创：Amusi CVer 今天

点击上方“CVer”，选择“星标”和“置顶”  
重磅干货，第一时间送达

## 前戏

最近实例分割方向，出了很多paper，CVer也立即跟进报道（点击可访问）：

- CVPR2019 | 超越Mask R-CNN！华科开源图像实例分割新方法MS R-CNN

本文要介绍一篇很棒的实时实例分割论文：YOLACT，该论文是由 **加利福尼亚大学** 提出。截止2019年4月16日，据Amusi所了解，上述**MS R-CNN**应该是实例分割（Instance Segmentation）mAP 最高的算法；而本文 **YOLACT** 是实例分割中最快的算法（即FPS最大）。难能可贵都是这两篇paper，都已经开源！

注：YOLACT应该是Amusi了解到的第一个又快又好的实例分割算法，如果你有发现比这两个算法还fancy的paper，欢迎后台留言进行补充。


## 简介

《YOLACT: Real-time Instance Segmentation》

# YOLACT Real-time Instance Segmentation

Daniel Bolya      Chong Zhou      Fanyi Xiao      Yong Jae Lee

University of California, Davis

{dbolya, cczhou, fyxiao, yongjaelee}@ucdavis.edu  CVer

arXiv: <https://arxiv.org/abs/1904.02689>

github: <https://github.com/dbolya/yolact>

作者团队: 加利福尼亚大学

注: 2019年04月05日刚出炉的paper

Abstract: 我们提出了一个用于实时实例分割的简单全卷积模型, 在单个Titan Xp上以33 fps在MS COCO上实现了**29.8 mAP**, 这比以前的任何算法都要快得多。此外, 我们只在一个GPU上训练后获得此结果。我们通过将实例分割分成两个并行子任务: (1) 生成一组原型掩膜 (prototype mask); (2) 预测每个实例的掩膜系数 (mask coefficients)。然后我们通过将原型与掩膜系数线性组合来生成实例掩膜 (instance masks)。我们发现因为这个过程不依赖于 repooling, 所以这种方法可以产生非常高质量的掩膜。此外, 我们分析了 the emergent behavior of our prototypes, 并表明他们学会以 translation variant manner 定位实例, 尽管是完全卷积的。最后, 我们还提出了快速NMS (Fast NMS), 比标准NMS快12 ms, 只有一点点性能损失。

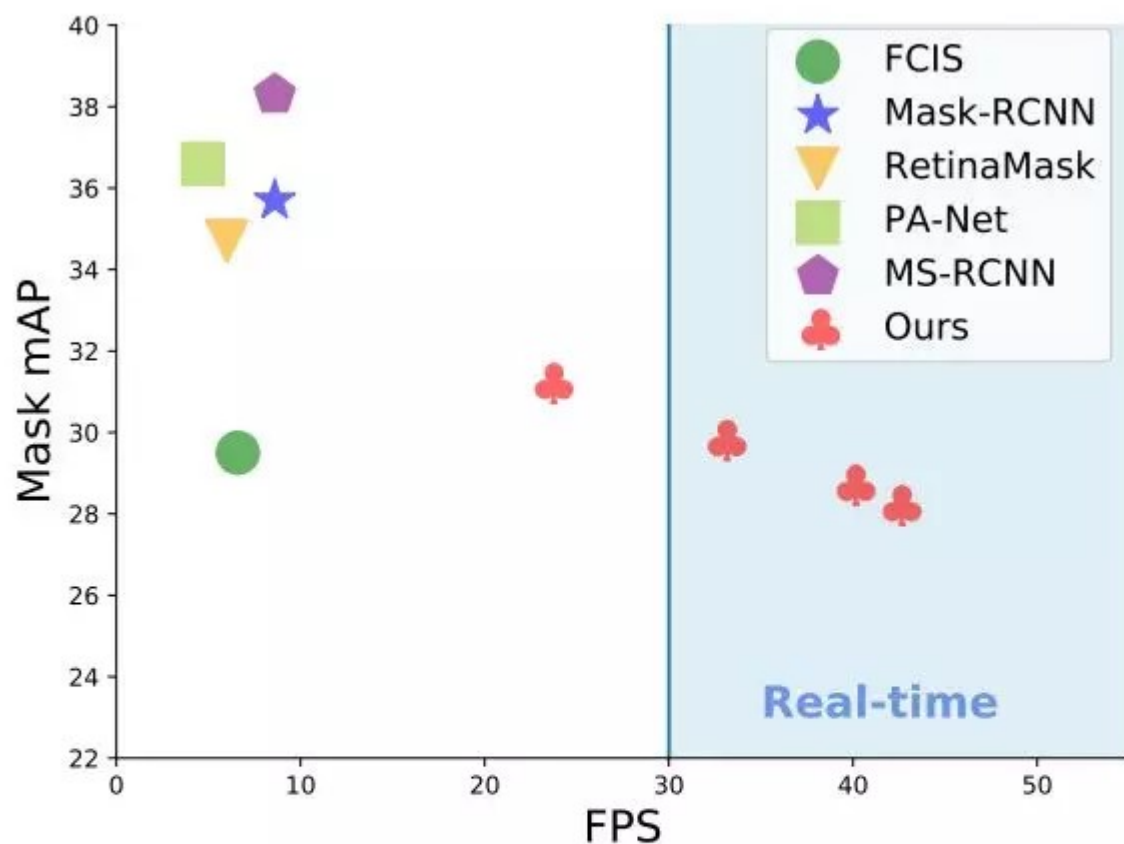


Figure 1: Speed-performance trade-off for various instance segmentation methods on COCO. To our knowledge, ours is the first *real-time* (above 30 FPS) approach with around 30 mask mAP on COCO test-dev.

CVer

## 正文

### 背景

先介绍一下为什么叫**YOLOACT**，因为其全称为：**You Only Look At CoefficientTs**。这里应该是致敬YOLO。

原作者在论文中引用了YOLOv3中的这句话，“**Boxes are stupid anyway though, I’m probably a true believer in masks except I can’t get YOLO to learn them.**”

*“Boxes are stupid anyway though, I’m probably a true believer in masks except I can’t get YOLO to learn them.”*

– Joseph Redmon, YOLOv3 [24] CVer

注：日常催更，YOLOv4该来了吧！

## 本文算法 (YOLACT)

YOLACT的目标是将掩模分支添加到现有的一阶段 (one-stage) 目标检测模型，其方式与 Mask R-CNN对 Faster-CNN 操作相同，但没有明确的定位步骤 (如, feature repooling)。为此，我们将实例分割的复杂任务分解为两个更简单的并行任务，这些任务可以组合以形成最终的掩模。第一个分支使用FCN生成一组图像大小的“原型掩模” (“prototype masks)，它们不依赖于任何一个实例。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。最后，对经过NMS后的每个实例，我们通过线性组合这两个分支的工作来为该实例构造掩模。

YOLACT将问题分解为两个并行的部分，利用 **fc层** (擅长产生语义向量) 和 **conv层** (擅长产生空间相干掩模) 来分别产生“掩模系数”和“原型掩模”。然后，因为原型和掩模系数可以独立地计算，所以 backbone 检测器的计算开销主要来自合成 (assembly) 步骤，其可以实现为单个矩阵乘法。通过这种方式，我们可以在特征空间中保持空间一致性，同时仍然是一阶段和快速的。

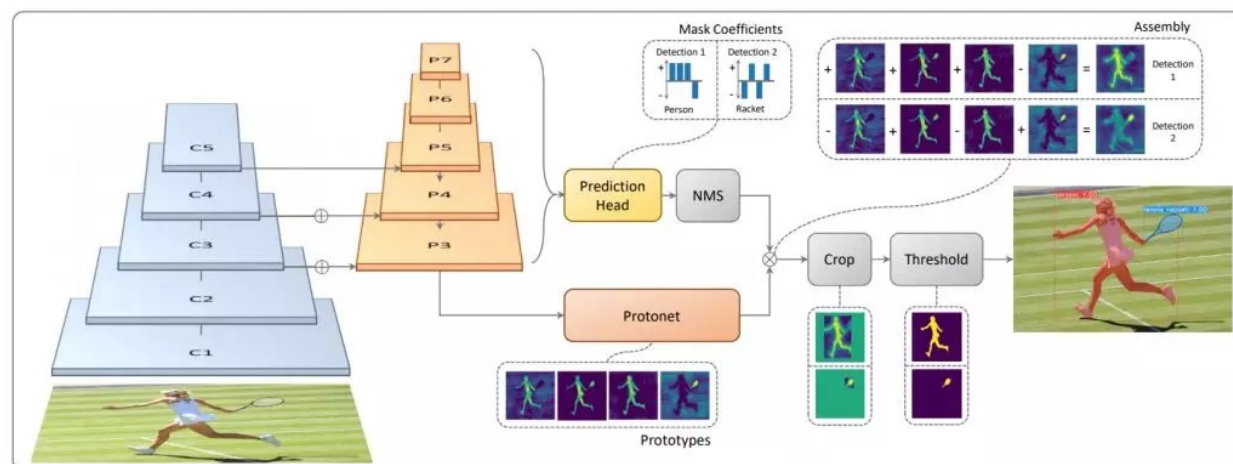


Figure 2: **YOLACT Architecture** Blue/yellow indicates low/high values in the prototypes, gray nodes indicate functions that are not trained, and  $k = 4$  in this example. We base this architecture off of RetinaNet [25] using ResNet-101 + FPN.

## 1 原型生成 (Prototype Generation)

本文将 protonet 实现为FCN，其最后一层有k个 channels（每个原型一个）并将其附加到 backbone 特征层

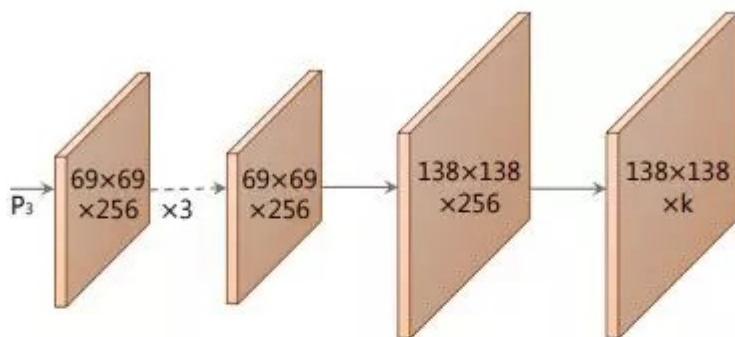


Figure 3: **Protonet Architecture** The labels denote feature size and channels for an image size of  $550 \times 550$ . Arrows indicate  $3 \times 3$  conv layers, except for the final conv which is  $1 \times 1$ . The increase in size is an upsample followed by a conv. Inspired by the mask branch in [16].

## 2 掩膜系数 (Mask Coefficients)

典型的基于Anchor (anchor-based) 的目标检测器在其预测 head 中具有两个分支：一个分支用于预测 c 类置信度 (confidences)，另一个分支用于预测 4 个边界框回归量。对于掩膜系数预测，本文简单地添加并行的第三个分支，其预测 k 个掩膜系数，一个对应于每个原型。因此，不是每个 anchor 产生  $4 + c$  个系数，而是产生  $4 + c + k$ 。

## 3 掩膜合成 (Mask Assembly)

为了生成实例掩码，我们需要将原型分支和掩膜系数分支进行合成，使用前者与后者的线性组合作为系数。这个运算可以由单个矩阵相乘完成：

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

其中，P是  $h \times w \times k$  的原型掩膜，C是  $n \times k$  的掩膜系数。



## 4 YOLACT 检测器 (Detector)

Backbone: ResNet-101 + FPN

Image Size: 550\*550

注：本文其实尝试了ResNet-101和DarkNet-53，这部分很好理解，backbone越轻量级，YOLACT运行速度越快，但mAP越低。不过使用ResNet-101就能达到33 FPS，真的很赞。

## 4 快速非极大值抑制 (Fast NMS)

### 实验结果

YOLACT在COCO test-dev上的mAP和FPS，其中基于ResNet-101的YOLACT-550 比之前具有相同mAP的算法快了 3.8倍。

Method	Backbone	FPS	Time	AP	AP <sub>50</sub>	AP <sub>75</sub>	AP <sub>S</sub>	AP <sub>M</sub>	AP <sub>L</sub>
PA-Net [27]	R-50-FPN	4.7	212.8	36.6	58.0	39.3	16.3	38.1	53.1
RetinaMask [12]	R-101-FPN	6.0	166.7	34.7	55.4	36.9	14.3	36.7	50.5
FCIS [22]	R-101-C5	6.6	151.5	29.5	51.5	30.2	8.0	31.0	49.7
Mask R-CNN [16]	R-101-FPN	8.6	116.3	35.7	58.0	37.8	15.5	38.1	52.4
MS R-CNN [18]	R-101-FPN	8.6	116.3	<b>38.3</b>	58.8	41.5	17.8	40.4	54.4
YOLACT-550	R-101-FPN	<b>33.0</b>	<b>30.3</b>	29.8	48.5	31.2	9.9	31.3	47.7
YOLACT-400	R-101-FPN	44.0	22.7	24.9	42.0	25.4	5.0	25.3	45.0
YOLACT-550	R-50-FPN	42.5	23.5	28.2	46.6	29.2	9.2	29.3	44.8
YOLACT-550	D-53-FPN	40.0	25.0	28.7	46.8	30.0	9.5	29.5	45.5
YOLACT-700	R-101-FPN	23.6	42.4	31.2	50.6	32.8	12.1	33.5	47.1

### Fast NMS的有效性

Method	NMS	AP	FPS	Time	$k$	AP	FPS	Time	Method	AP	FPS	Time
YOLACT	Standard	<b>30.0</b>	23.8	42.1	32	27.7	<b>32.4</b>	<b>30.9</b>	FCIS w/o Mask Voting	27.8	9.5	105.3
	Fast	29.9	<b>33.0</b>	<b>30.3</b>	64	<b>27.8</b>	31.7	31.5	Mask R-CNN (550 × 550)	<b>32.2</b>	13.5	73.9
Mask R-CNN	Standard	<b>36.1</b>	8.6	116.0	128	27.6	31.5	31.8	<i>fc</i> -mask	20.7	25.7	38.9
	Fast	35.8	<b>9.9</b>	<b>101.0</b>	256	27.7	29.8	33.6	YOLACT-550 (Ours)	29.9	<b>33.0</b>	<b>30.3</b>

(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.

(b) **Prototypes** Choices for  $k$  in our method. YOLACT is robust to varying  $k$ , so we choose the fastest ( $k = 32$ ).

(c) **Accelerated Baselines** We compare to other baseline methods by tuning their speed-accuracy trade-offs. *fc*-mask is our model but with  $16 \times 16$  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.

## YOLACT部分实例分割结果



## CVer图像分割交流群

扫码添加CVer助手，可申请加入**CVer-图像分割交流群**。一定要备注：**图像分割+地点+学校/公司+昵称**（如图像分割+上海+上交+卡卡）



▲ 长按加群

这么硬的**论文速递**，麻烦给我一个**在看**



▲ 长按关注我们

**麻烦给我一个在看！**

[阅读原文](#)