YOLACT:

Real-time Instance Segmentation

Daniel Bolya, Chong Zhou, Fanyi Xiao, Yong Jae Lee

[University of California, Davis]

ICCV 2019

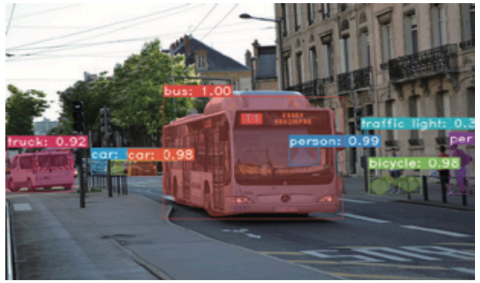
Sungman, Cho.


airplane: 0.67
airplane: 0.97
airplane: 0.5
truck: 0.91
truck: 0.84
truck: 0.79
truck: 0.36
truck: 0.4

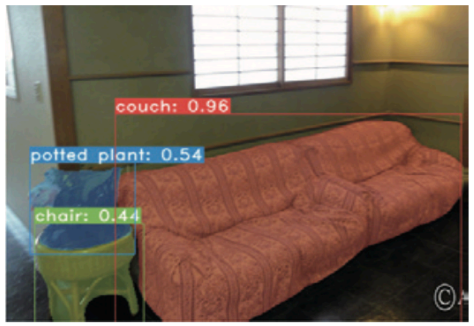

person: 1.00
person: 0.84
sports bal

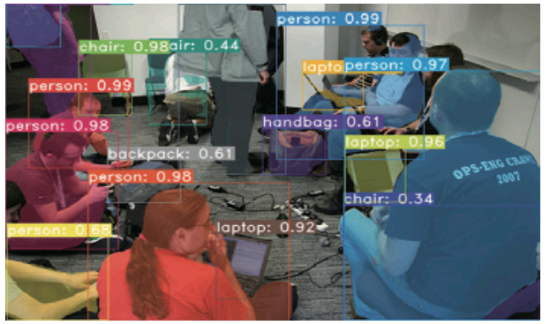

person: 1.00
person: 0.37
person: 0.83
person: 0.96
person: 0.88
baseball glove: 0.48
person: 0.82


person: 1.00
person: 0.99
person: 0.89
person: 0.76
person: 0.78
cake: 0.34


bus: 1.00
truck: 0.92
car: 0.98
person: 0.99
bicycle: 0.98


person: 0.40
person: 0.65
person: 0.79
person: 0.83
person: 0.96
bowl: 0.73
bowl: 0.83
bowl: 0.40
bowl: 0.43
cup: 0.33
bowl: 0.53
knife: 0.35

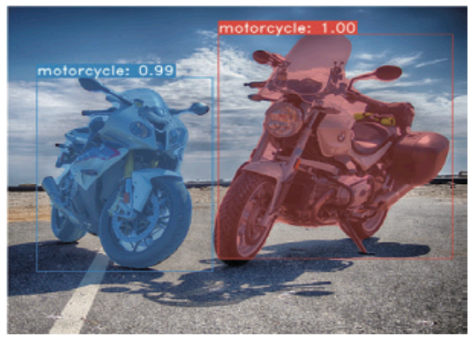

couch: 0.96
potted plant: 0.54
chair: 0.44

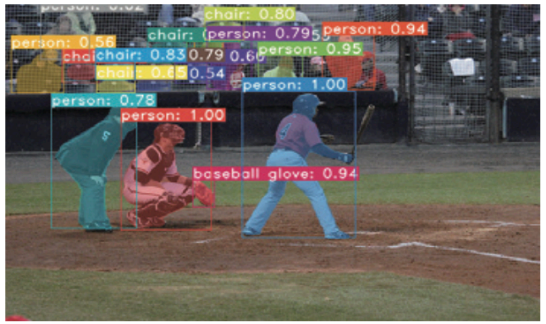

person: 0.99
chair: 0.98
person: 0.99
person: 0.97
handbag: 0.61
person: 0.98
backpack: 0.61
laptop: 0.96
person: 0.98
chair: 0.34
laptop: 0.92


person: 0.99
person: 1.00
person: 0.95


airplane: 0.92
truck: 0.90

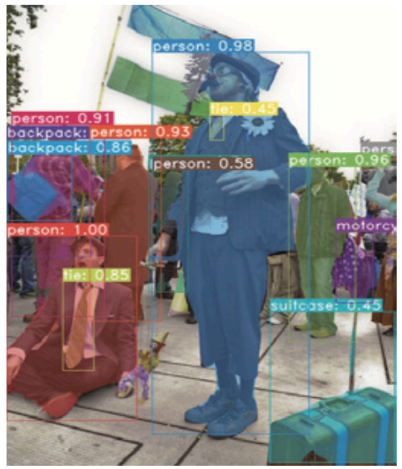

bus: 1.00
person: 0.68
car: 0.70

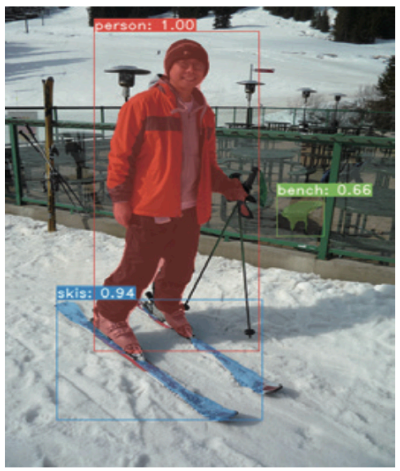

motorcycle: 1.00
motorcycle: 0.99


chair: 0.80
person: 0.94
person: 0.95
person: 1.00
person: 0.78
person: 1.00
baseball glove: 0.94


person: 0.95
frisbee: 0.96

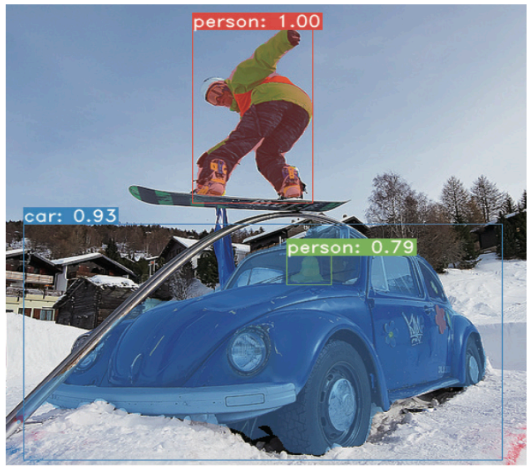

person: 1.00
person: 0.74

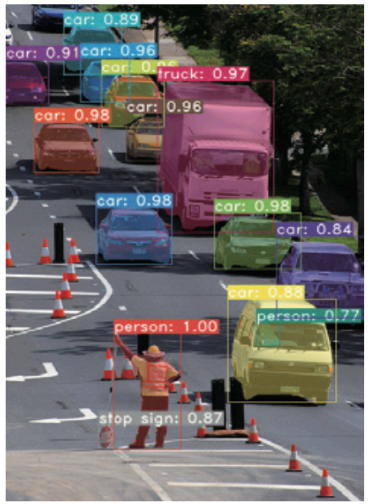

person: 0.98
tie: 0.45
person: 0.91
backpack: person: 0.93
backpack: 0.86
person: 0.58
person: 0.96
person: 1.00
tie: 0.85
suitcase: 0.45


person: 1.00
bench: 0.66
skis: 0.94


person: 1.00
sports ball: 0.99
tennis racket: 0.99


person: 0.35
person: 0.81
person: 1.00

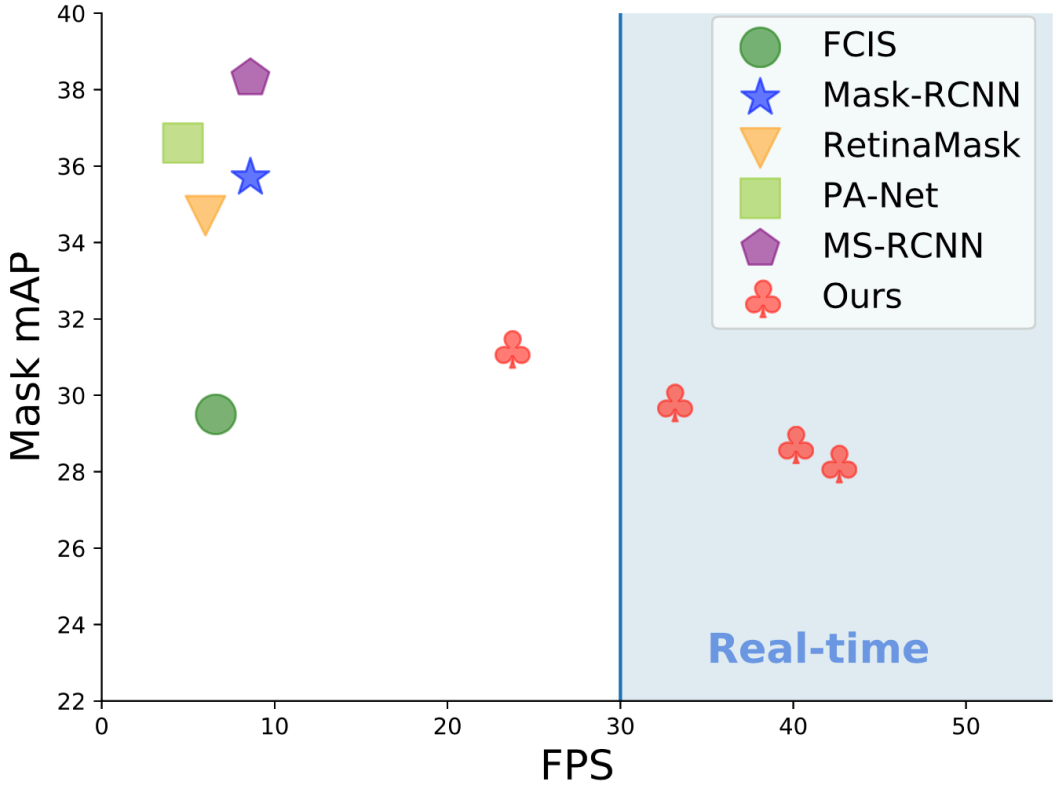

person: 1.00
car: 0.93
person: 0.79


car: 0.89
car: 0.91
car: 0.96
truck: 0.97
car: 0.98
car: 0.96
car: 0.98
car: 0.98
car: 0.84
car: 0.88
person: 0.77
person: 1.00
stop sign: 0.87

Introduction

Challenges

- Instance segmentation is hard – much harder than object detection.



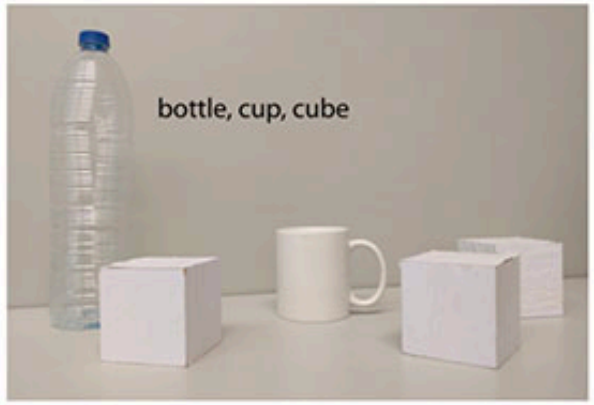
Contribution

- **Real-time Instance Segmentation.**

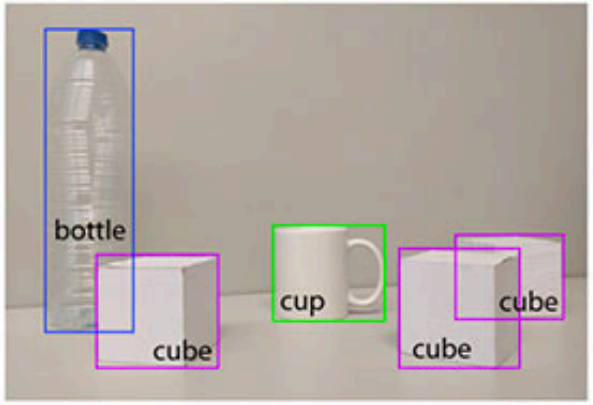
  (29.8mAP on MS COCO at 33.5fps)

- Propose Fast NMS

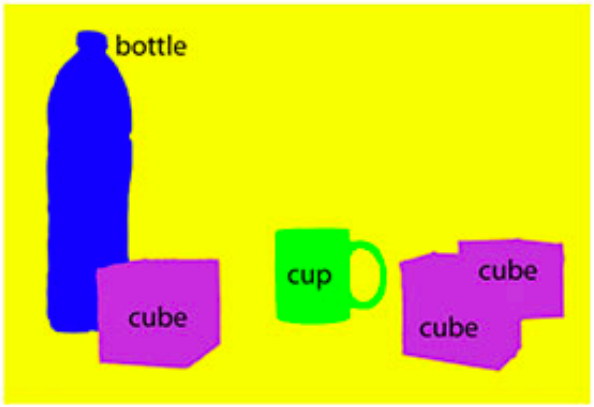
Related Works – Instance Segmentation



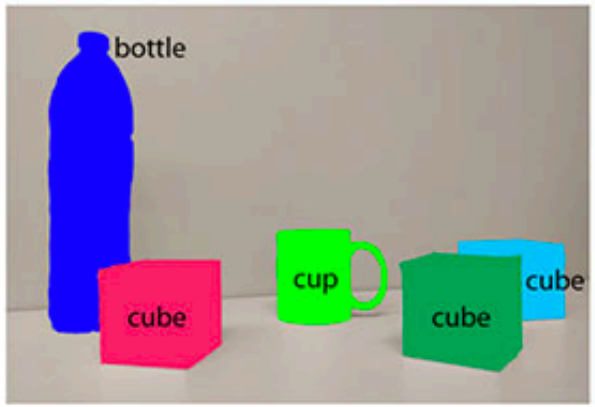
(a) Image classification



(b) Object localization



(c) Semantic segmentation



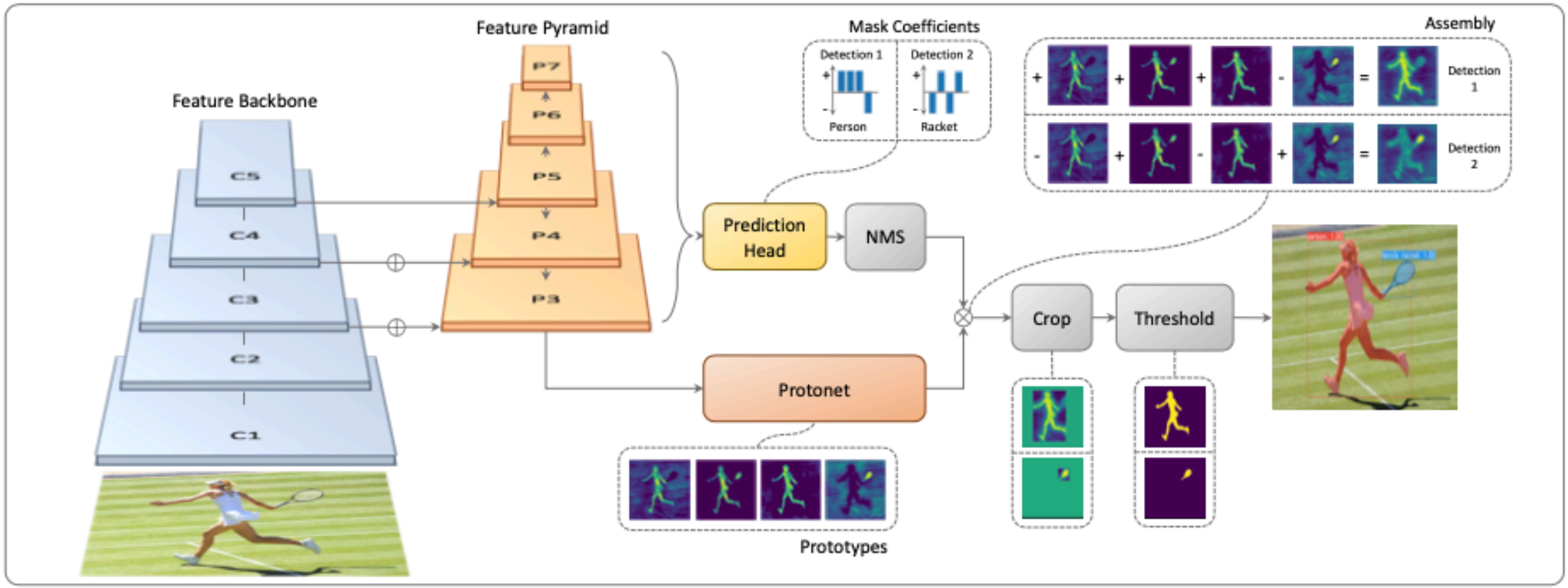
(d) Instance segmentation

- Box2Pix : 30 fps on PASCAL SBD 2012
- Straight to Shapes : 10.9 fps on Cityscapes
- Mask R-CNN : 13.5 fps on 550x550 images

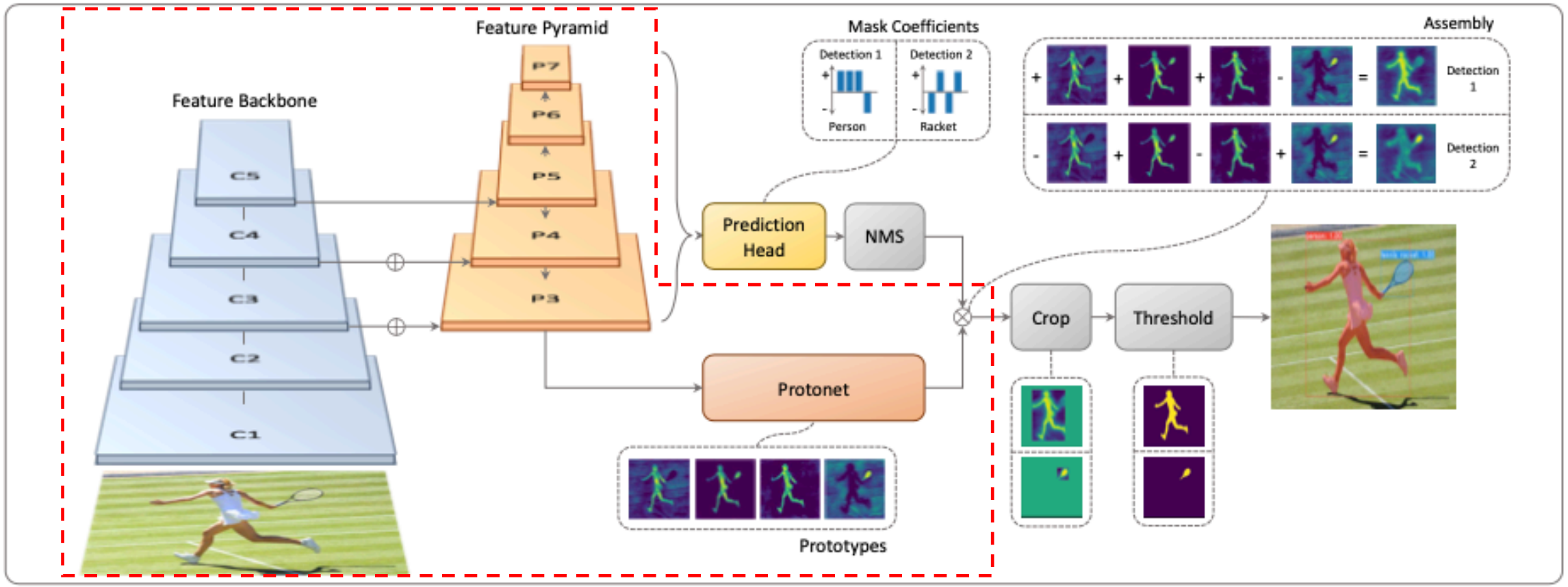
https://www.pyimagesearch.com/2018/11/19/mask-r-cnn-with-opencv/

Methodology

Two branches

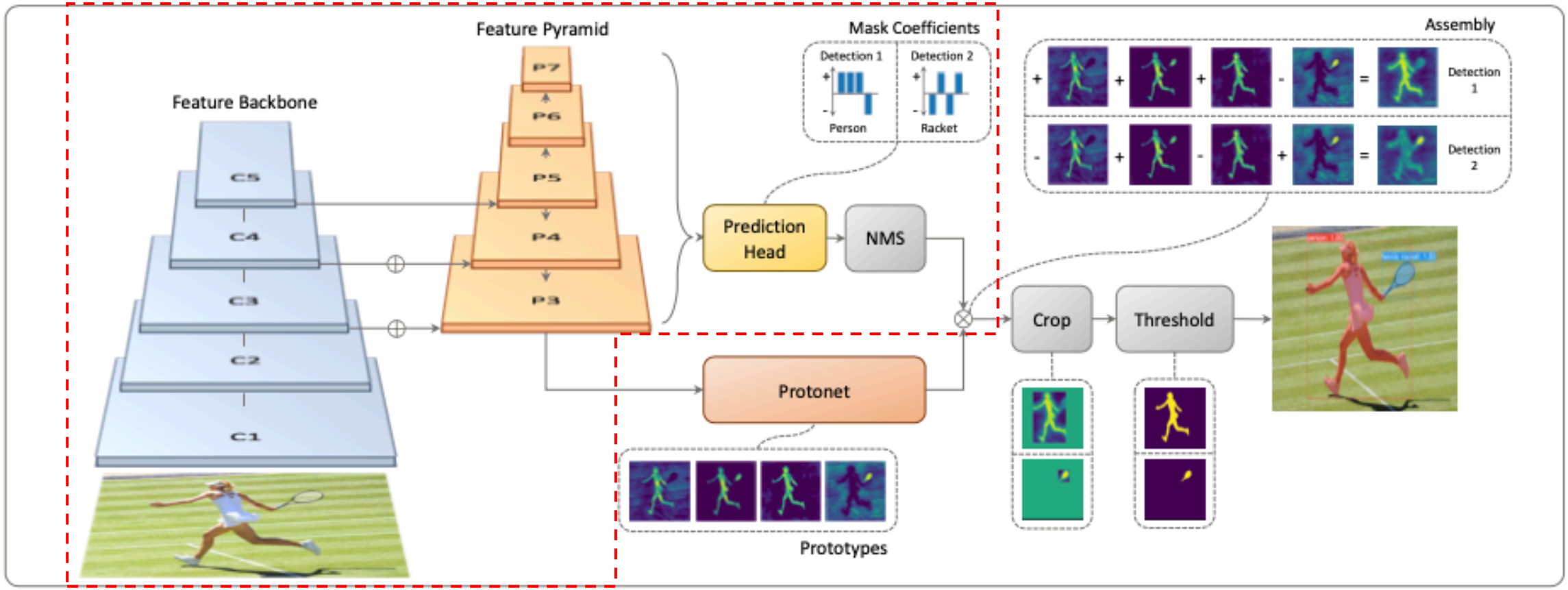


Two branches – FCN base.



Produce a set of image-sized "prototype masks" that do not depend on any one instance.

Two branches – Detection base.



Predict a vector of “mask coefficients” for each anchor that encode an instance’s representation

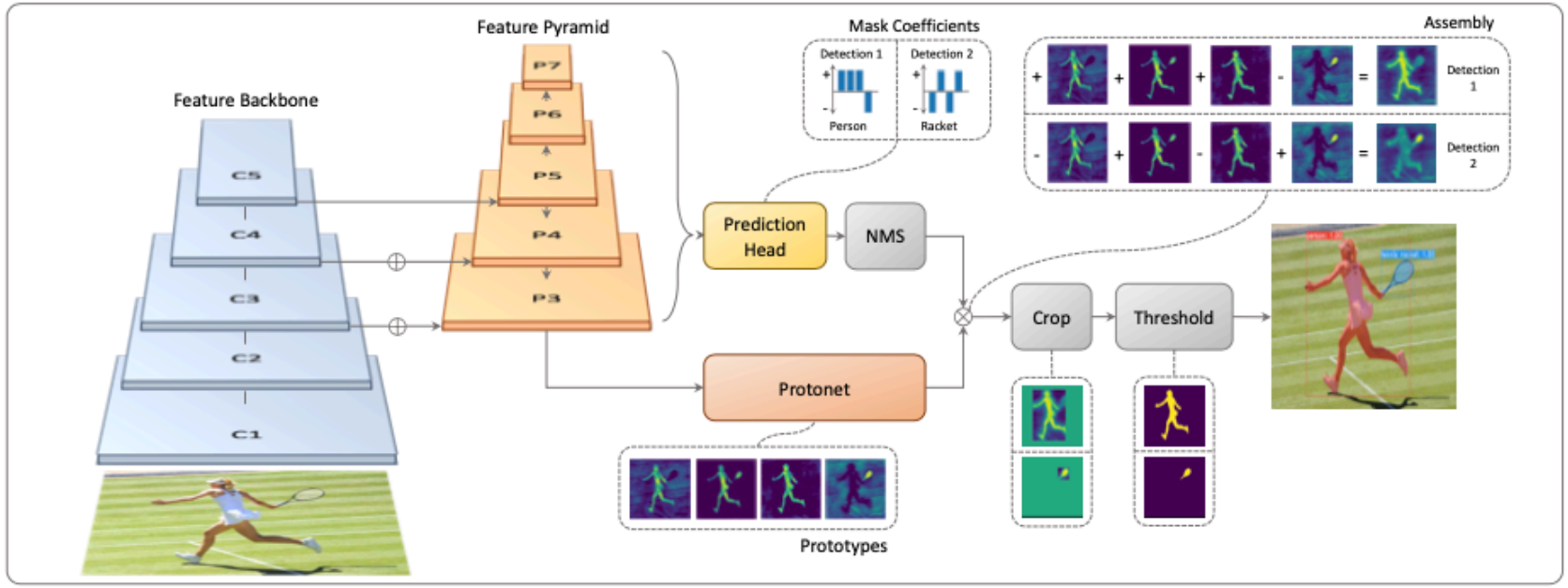
Rationale

- **masks are spatially coherent.**
  conv. layers naturally takes advantage of this coherence, a fc layer does not.

- Thus, making use of **fc layers**, which are good at **producing semantic vectors**, **conv. layers**, which are good at producing **spatially coherent masks**, to produce the " **mask coefficients** " and " **prototype masks** " , respectively.

Rationale

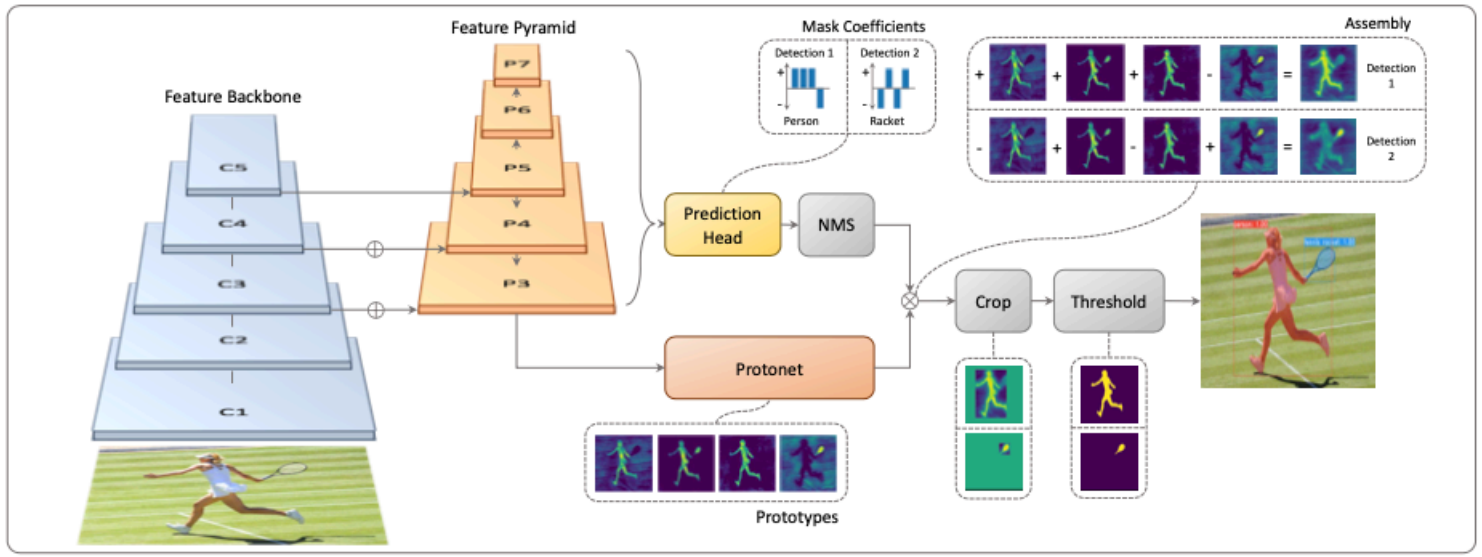
- Thus, making use of **fc layers**, which are good at **producing semantic vectors**, **conv. layers**, which are good at producing **spatially coherent masks**, to produce the " **mask coefficients** " and " **prototype masks** ", respectively.



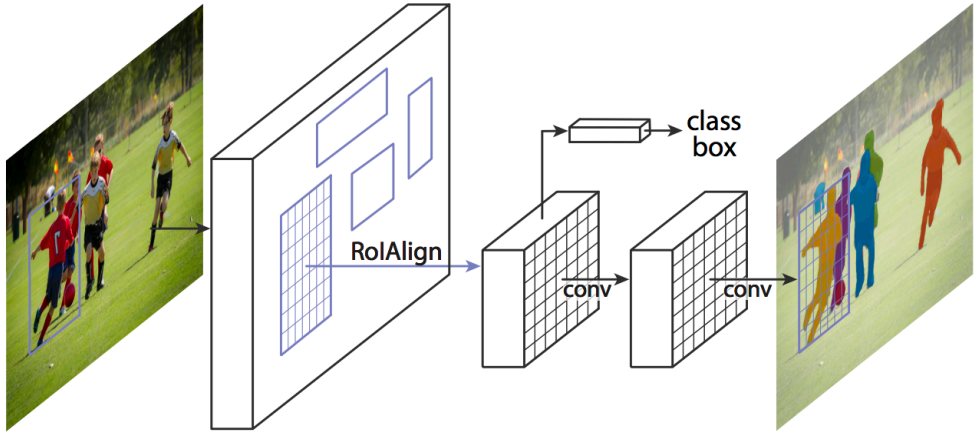
- Prototypes and mask coefficients can be computed independently.
- Assembly step can be implemented as a single matrix multiplication.

} ***Fast***

Rationale

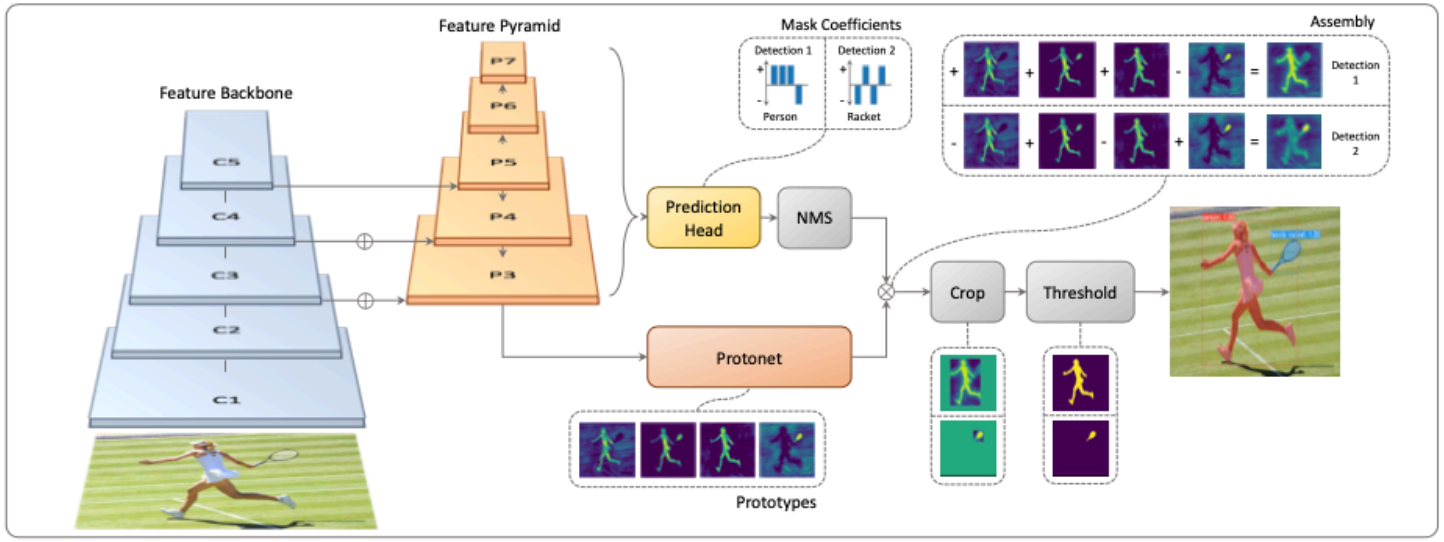


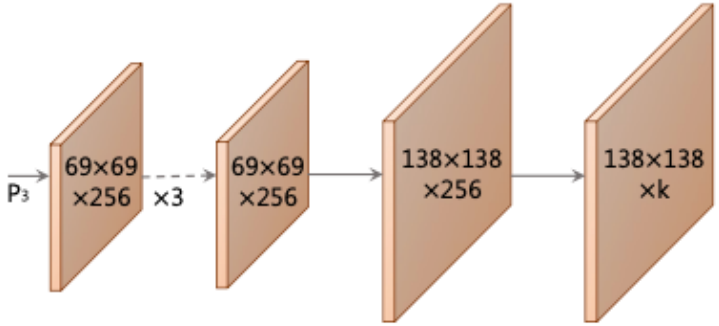
YOLACT



Mask R-CNN

Prototype Generation

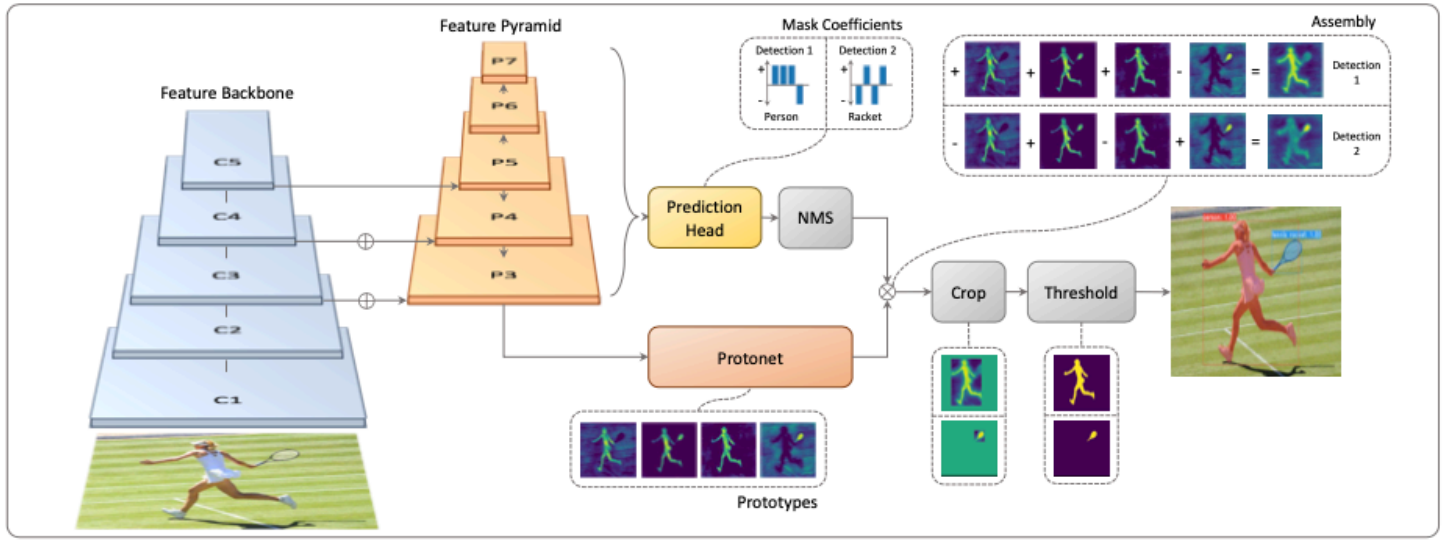


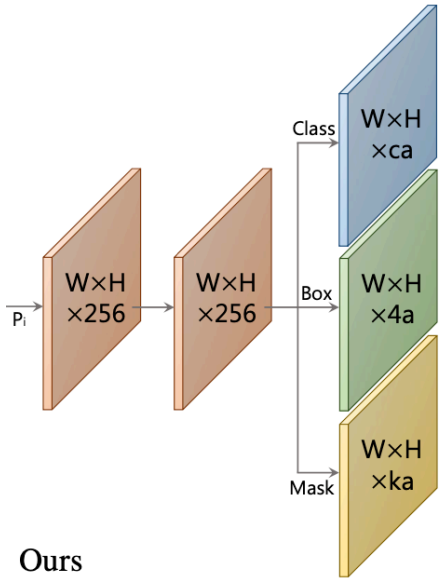


Protonet Architecture

- Implement protonet as an **FCN** whose last layer has k channels.
- Use FPN because its largest feature layers are the deepest.
- Upsample P3 to one fourth the dimensions of the input image to increase performance
- **Protonet output to be unbounded**, overpowering activations for prototypes it is very confident about (e.g., obvious background) → we have the option ReLU / no nonlinearity.

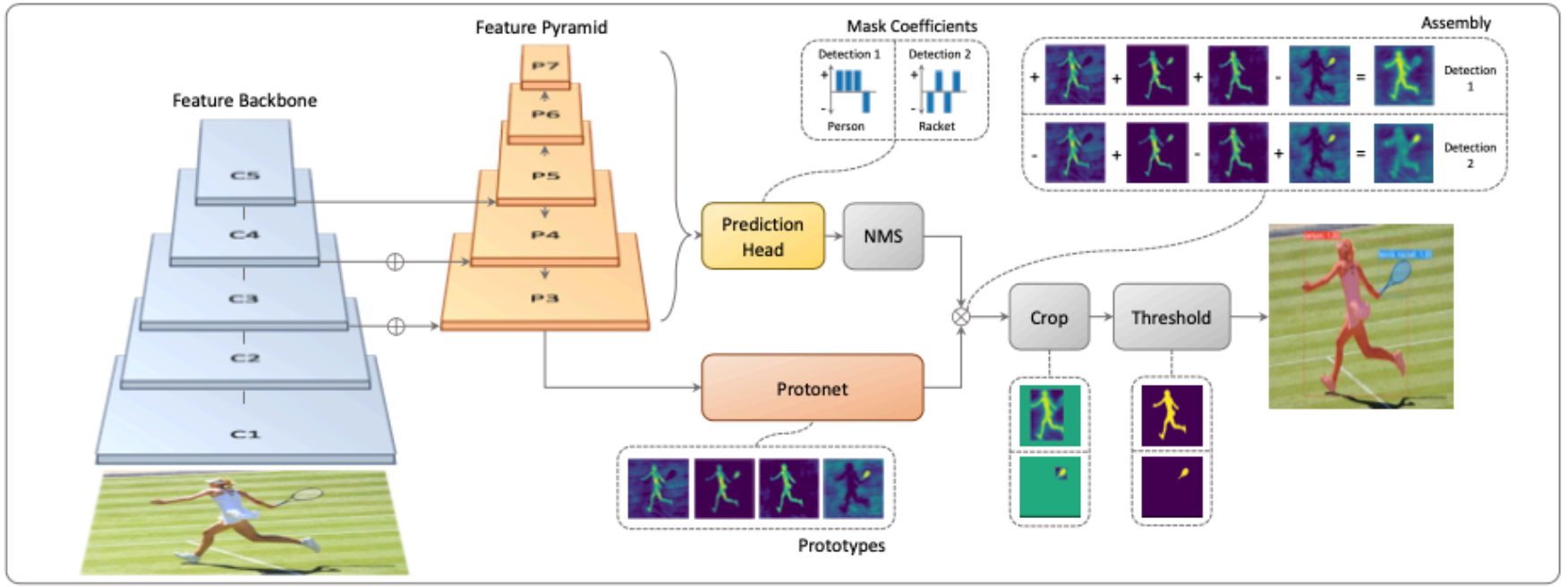
Mask Coefficients





- For mask coefficient prediction, we simply add a third branch in parallel that predicts k mask coefficients, one corresponding to each prototype.
- For nonlinearity, we find it important to be able to subtract our prototypes from the final mask.

Mask Assembly



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

$P : h \times w \times k$ *Prototype mask*

$C : h \times w \times k$ *Mask coefficients*

Loss / Cropping Masks

$$L_{cls} : L_{box} : L_{mask} = 1 : 1.15 : 6.125$$

Loss / Cropping Masks

$$\underbrace{L_{cls} : L_{box}}_{\text{SSD}} : L_{mask} = 1 : 1.15 : 6.125$$

Loss / Cropping Masks

$$L_{cls} : L_{box} : \boxed{L_{mask}} = 1 : 1.15 : 6.125$$

$$M_{gt}\colon L_{mask} = \mathrm{BCE}(M, M_{gt}).$$

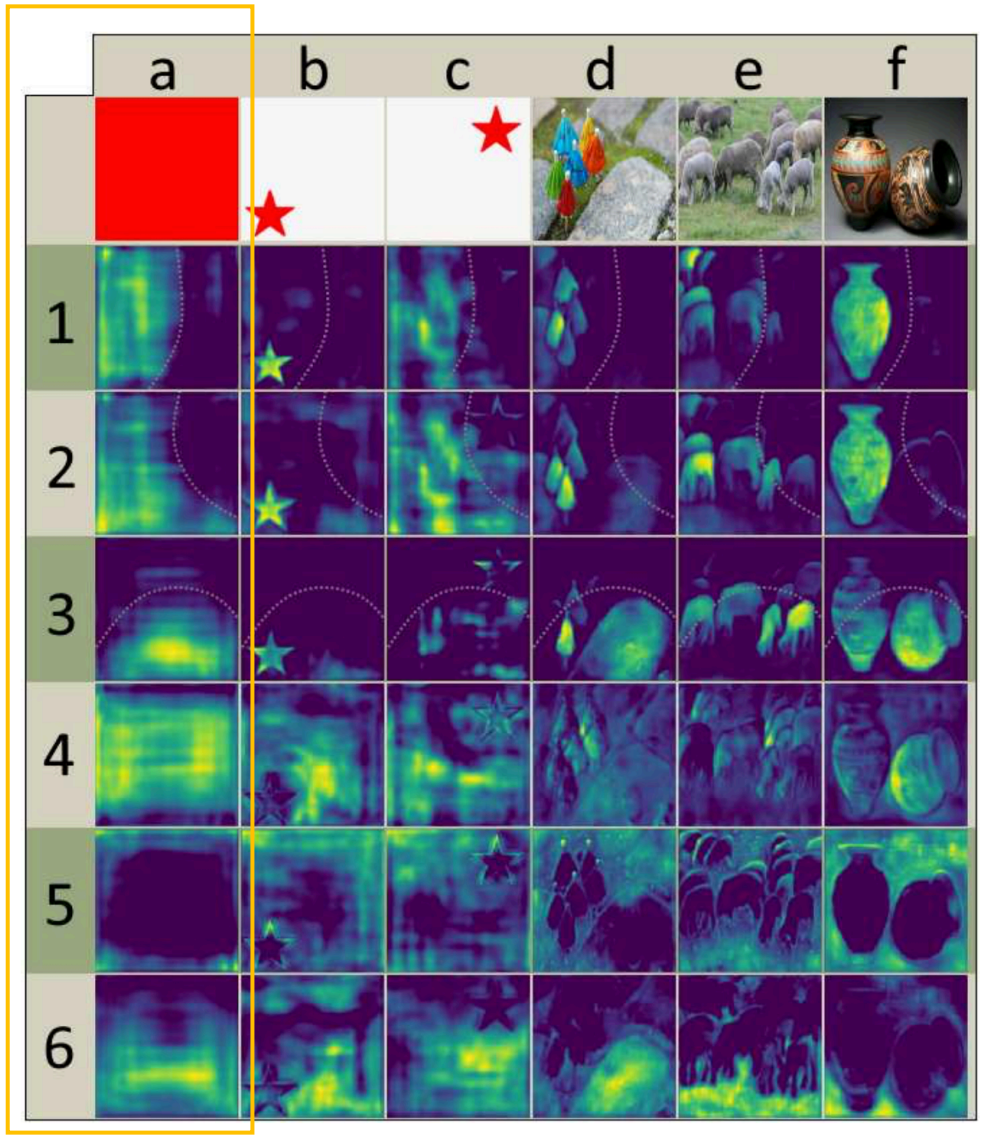
Loss / Cropping Masks

$$L_{cls} : L_{box} : L_{mask} = 1 : 1.15 : 6.125$$

Cropping Masks We crop the final masks with the predicted bounding box during evaluation. During training, we instead crop with the ground truth bounding box, and divide L_{mask} by the ground truth bounding box area to preserve small objects in the prototypes.

To be Translation Variant

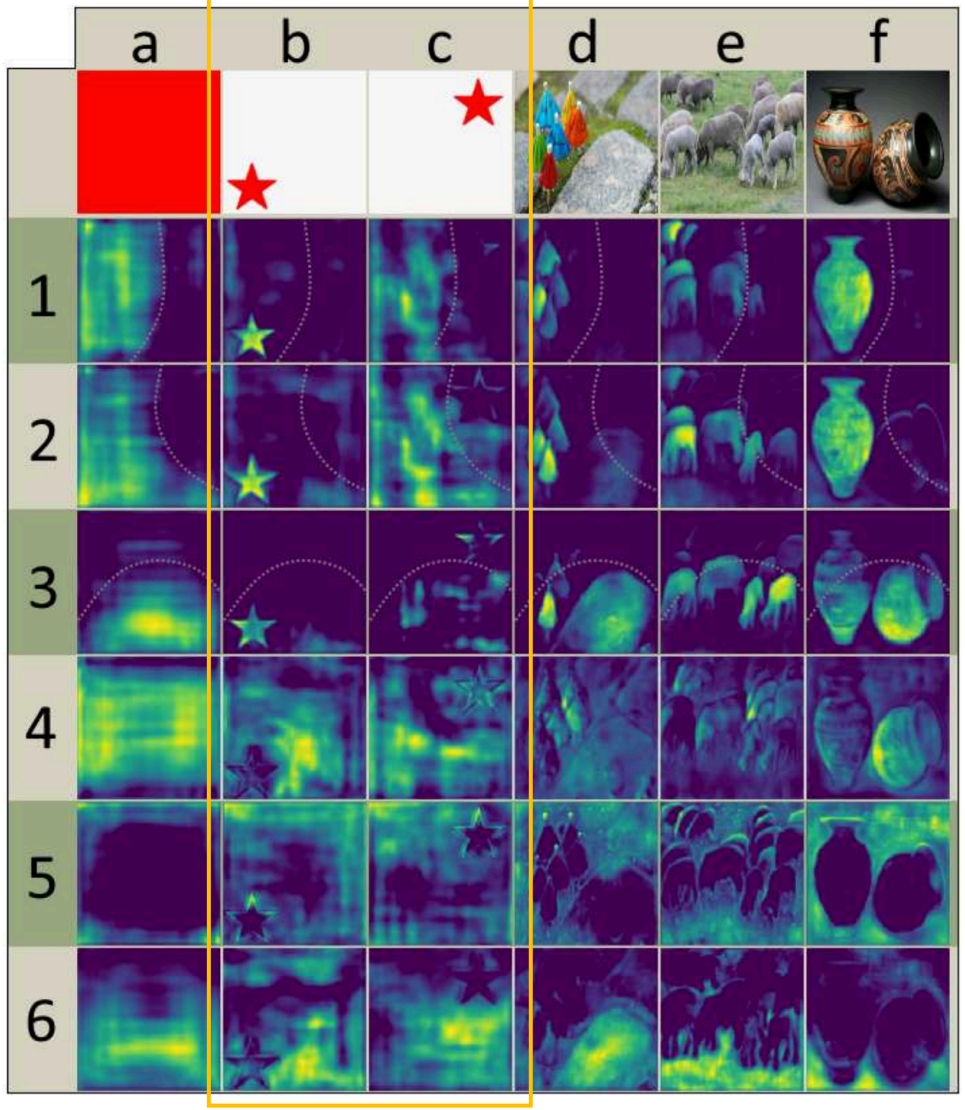
- FCN is translation invariant, the task needs tran slation variance. (FCIS, Mask R-CNN)
- YOLACT learns how to localize instances on its own via different activations in its prototypes.



To be Translation Variant

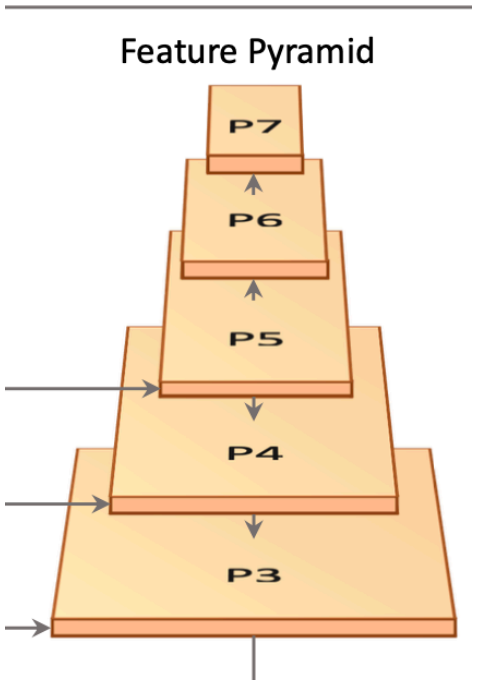
- FCN is translation invariant, the task needs tran slation variance. (FCIS, Mask R-CNN)
- YOLACT learns how to localize instances on its own via different activations in its prototypes.

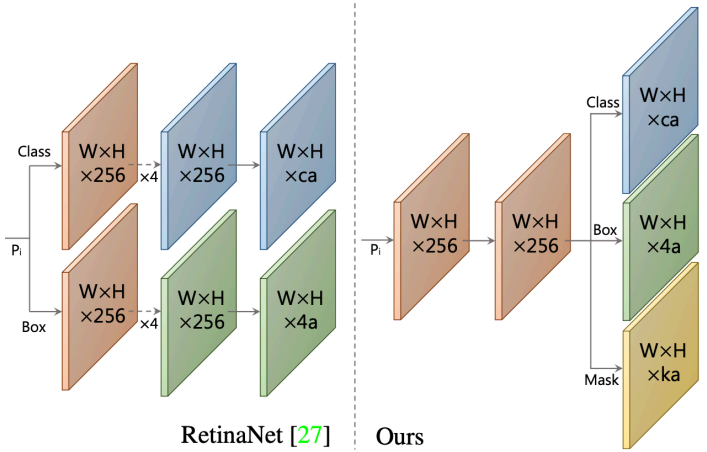
conv output will be the same. On the other hand, the consistent rim of padding in modern FCNs like ResNet gives the network the ability to tell how far away from the image's edge a pixel is. Conceptually, one way it could accomplish this is to have multiple layers in sequence spread the padded 0's out from the edge toward the center (e.g., with a kernel like $[1, 0]$). This means ResNet, for instance, *is inherently translation variant*, and our method makes heavy use of that property (images b and c exhibit clear translation variance).



YOLACT Detector

- Backbone : ResNet-101
- Image size : 550 x 550
- Modify FPN by not producing P2 and producing P6, P7
- 3 anchors : [1, 0.5, 2]
- Smooth-L1 loss to bbox regression.
- Cross entropy with c positive labels and 1 background label.
- Using OHEM with a 3:1(neg:pos)
- Do not use focal loss.





Fast NMS

- A version of NMS where every instance can be decided to be kept or discarded in parallel. (Fast NMS entirely in standarad GPU-accelerated matrix operations)

  1. Compute $c \times n \times n$ pairwise IoU matrx X.

  2. Batched sorting on the GPU is readily available and computing IoU can be easily vectorized.

  3. Remove detections (threshhold t)

     I. Setting the lower triangle and diagonal of X to 0: $X_{kij} = 0,\ \forall k, j, i\ \geq j$

     II. Taking column-wise max $K_{kj} = \max_i(X_{kij})\ \ \forall k, j$

     III. thresholding with t $(K < t)$

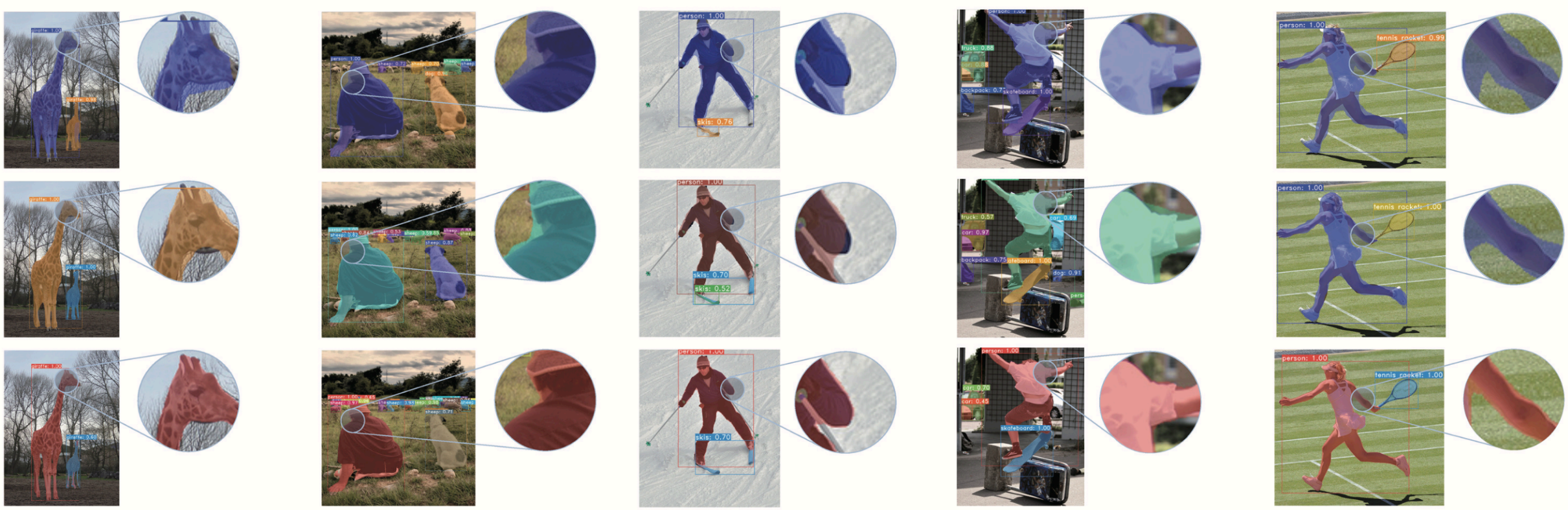
Fast NMS is 15.0 ms faster than traditional NMS with a perforamnce loss of only 0.3 mAP.
(in Mask R-CNN)

Results

FCIS

Mask R-CNN

Ours



Results

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

Results

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**

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