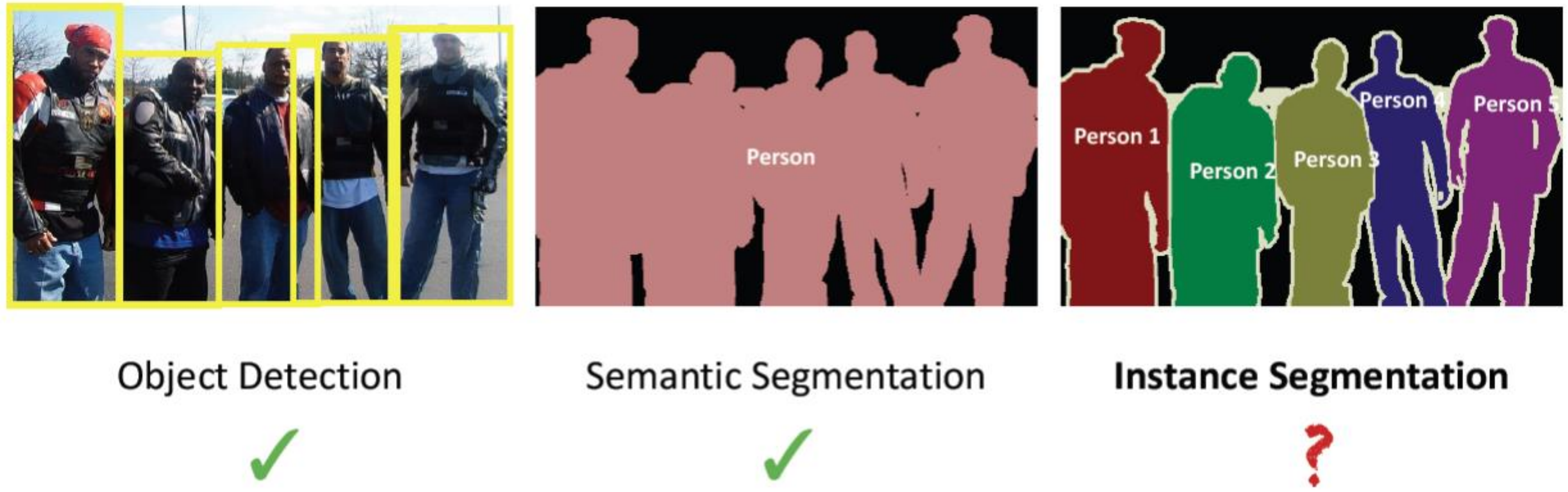
Deep Learning for Image Instance Segmentation ----YOLACT & YOLACT++

Jianping Fan
Dept of Computer Science
UNC-Charlotte

Course Website:
http://webpages.uncc.edu/jfan/itcs5152.html

github repo : https://github.com/dbolya/yolact

Definition of Image Instance Segmentation



Instance segmentation = object detection + semantic segmentation?

Slide from *Mask R-CNN Tutorial*, K. He. ICCV 2017

Scene understanding

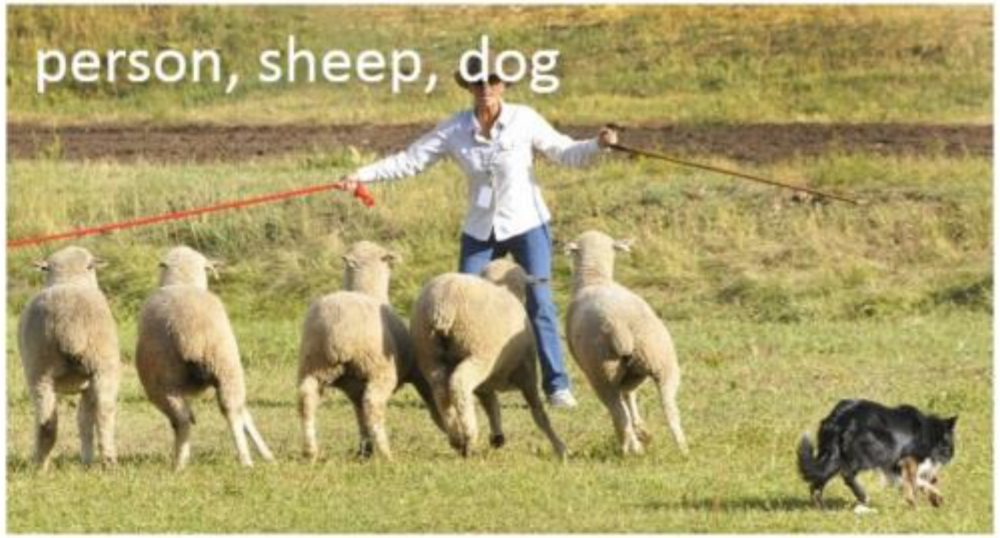
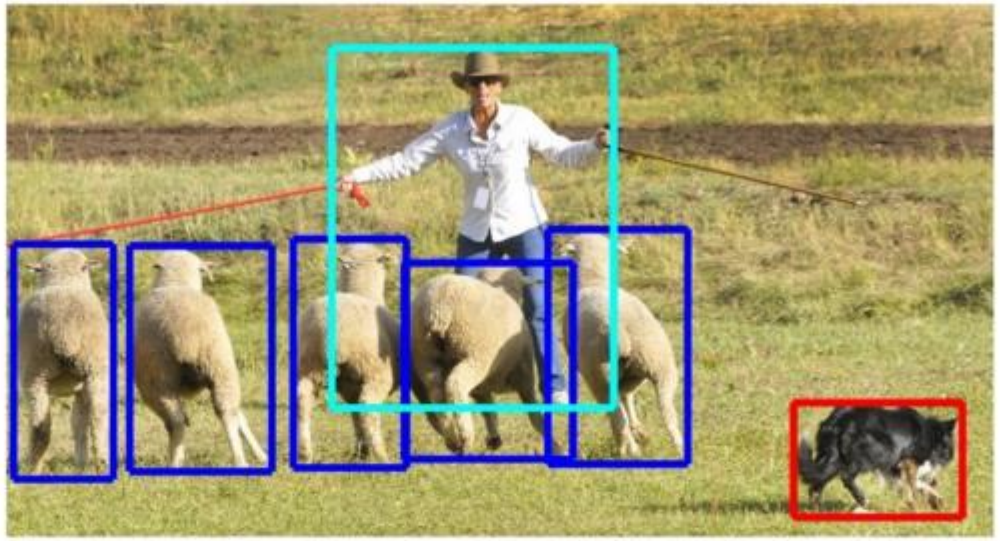
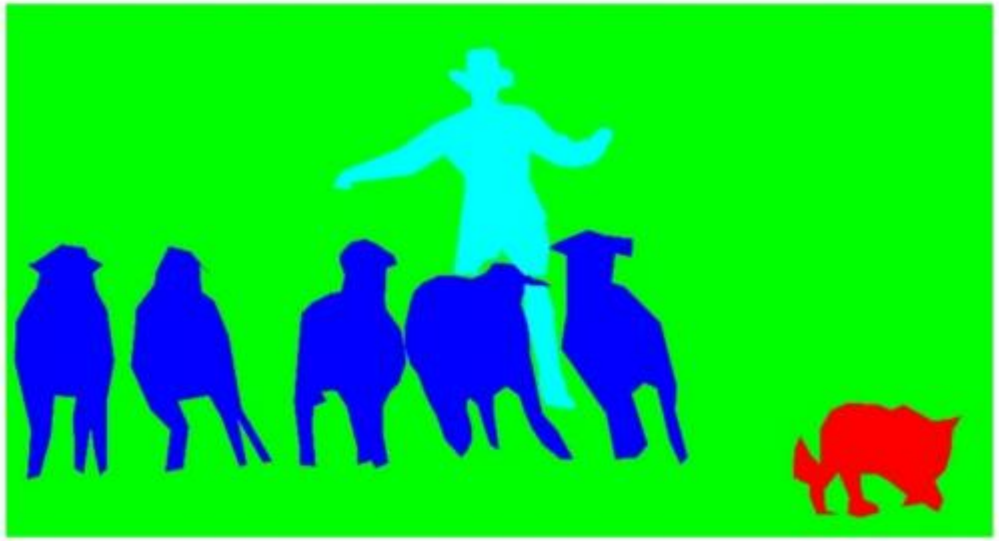


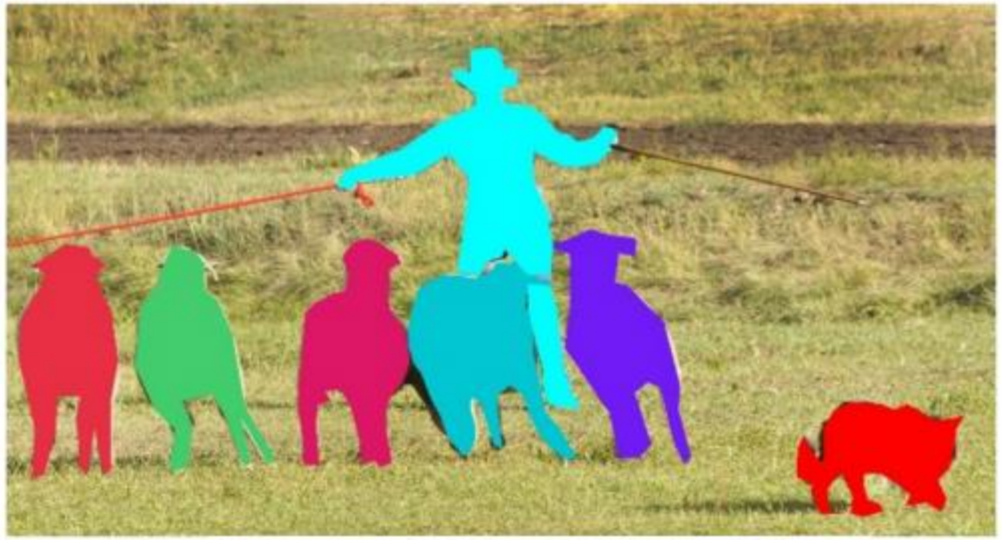
Image classification



Object detection

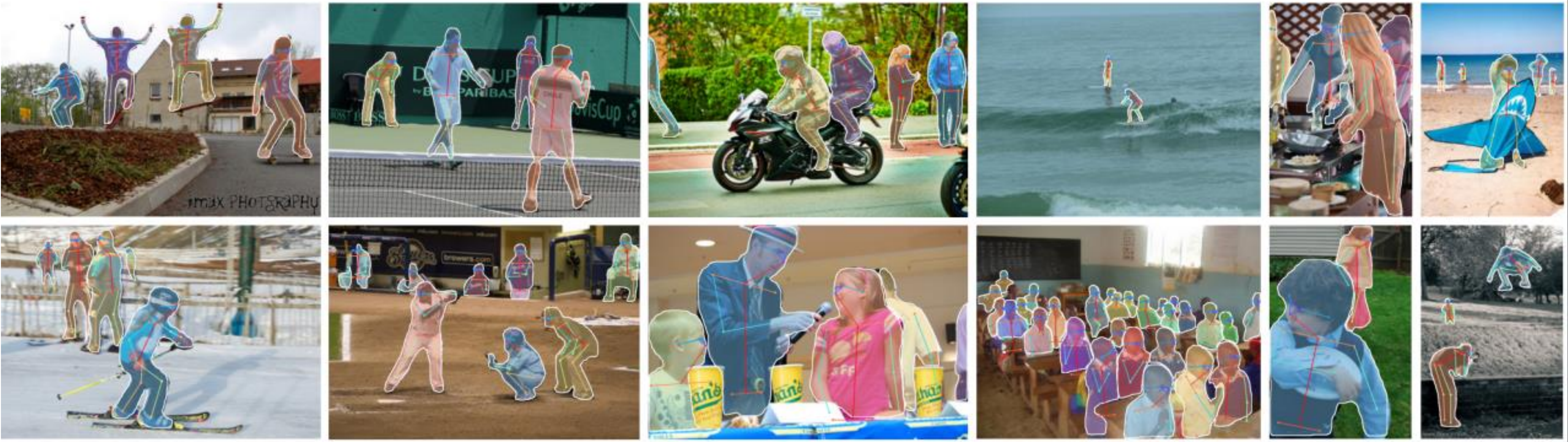


Semantic segmentation



Instance segmentation

Instance-level Object Understanding Today



He, Gkioxari, Dollár, Girshick. Mask R-CNN. In ICCV 2017

Two-Stage Approaches for Image Instance Segmentation

The two stage detector like Mask-RCNN is a representative two-stage instance segmentation approach that:

1. First generates candidate region-of-interests (ROIs)

2. Then classifies and segments those ROIs in the second stage.

The next few work followed on improving the FPN features or addressing the incompatibility between a mask’s confidence score and its localization accuracy.

2-Stage Models

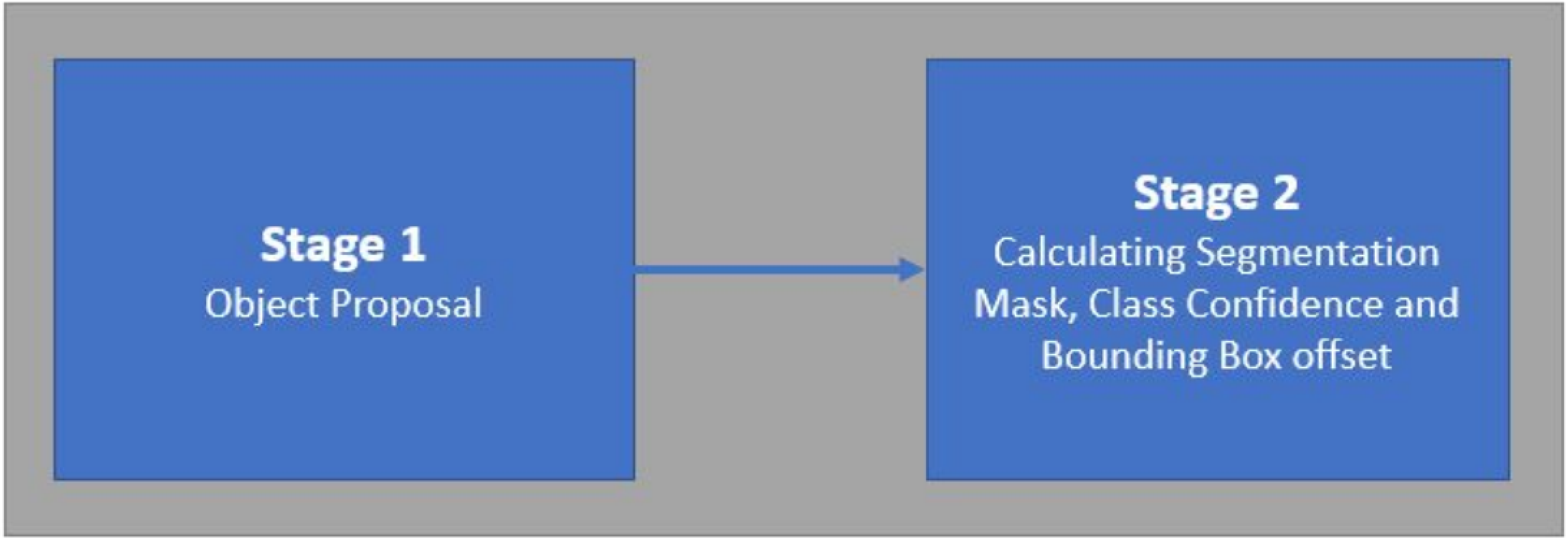
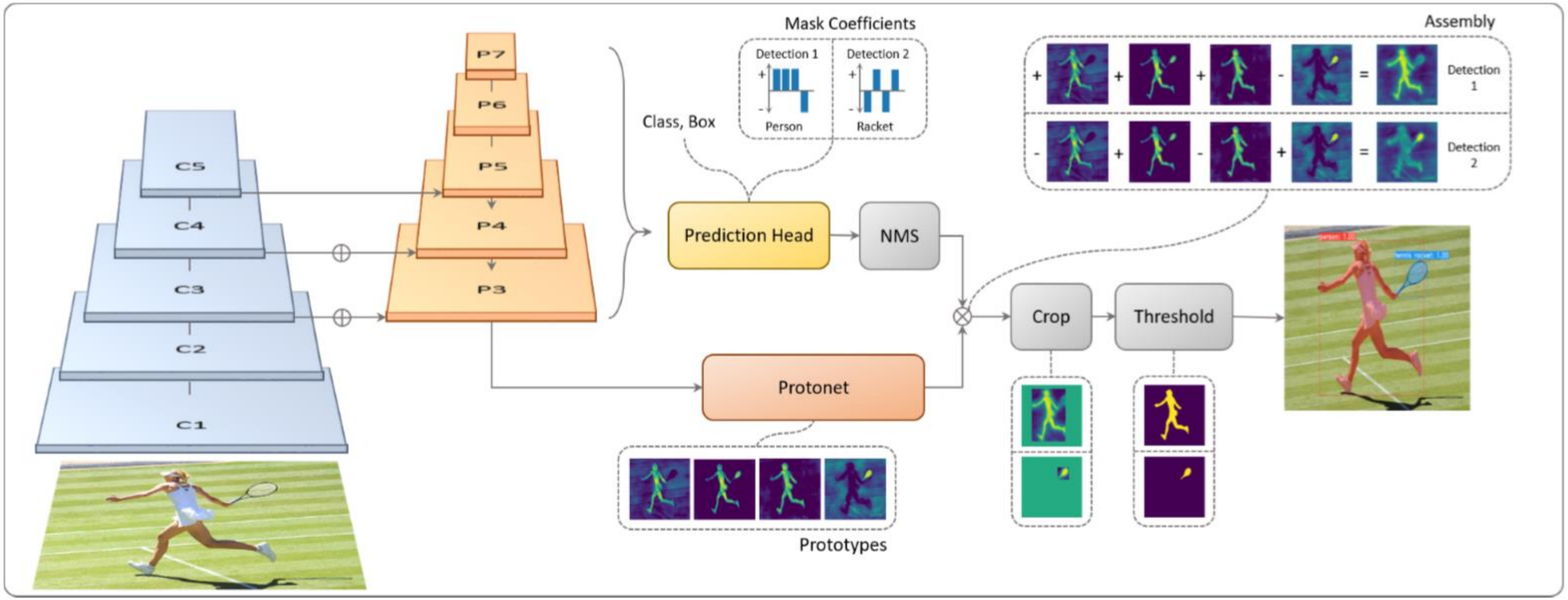

Stage 1
Object Proposal
Stage 2
Calculating Segmentation Mask, Class Confidence and Bounding Box offset

Fig. Working of Mask R-CNN

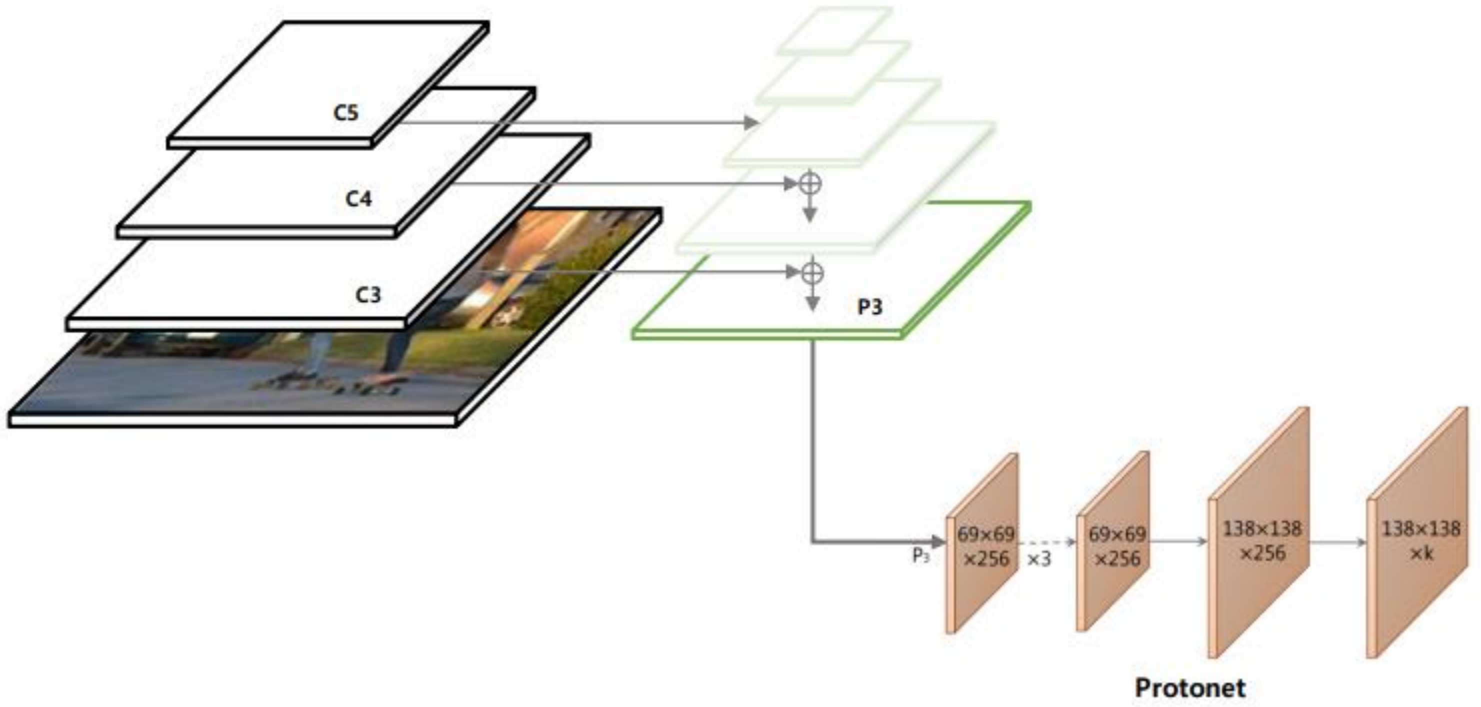
YOLACT Architecture

3

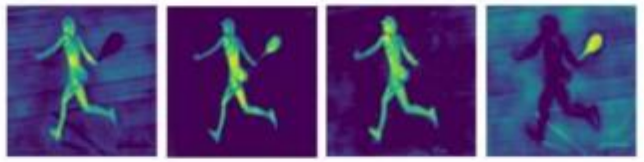


Performing mask segmentation of objects is much harder than obtaining bounding box of objects in object detection

Overall Model Architecture



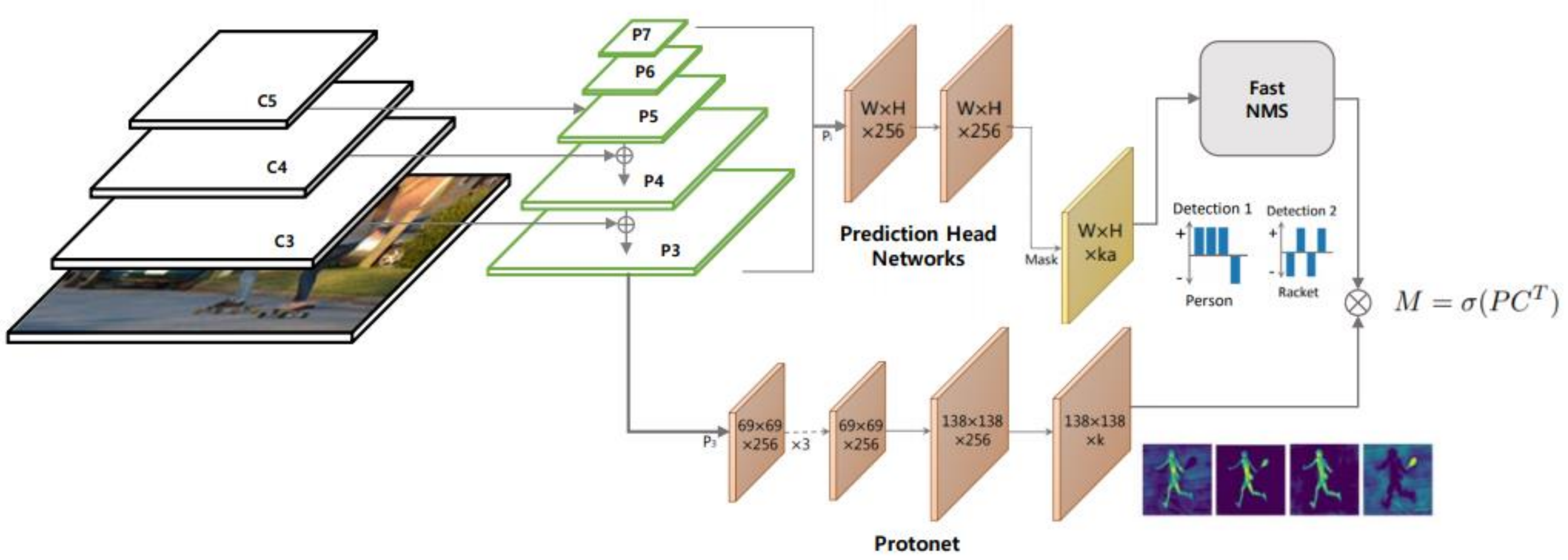
No explicit loss on the prototypes



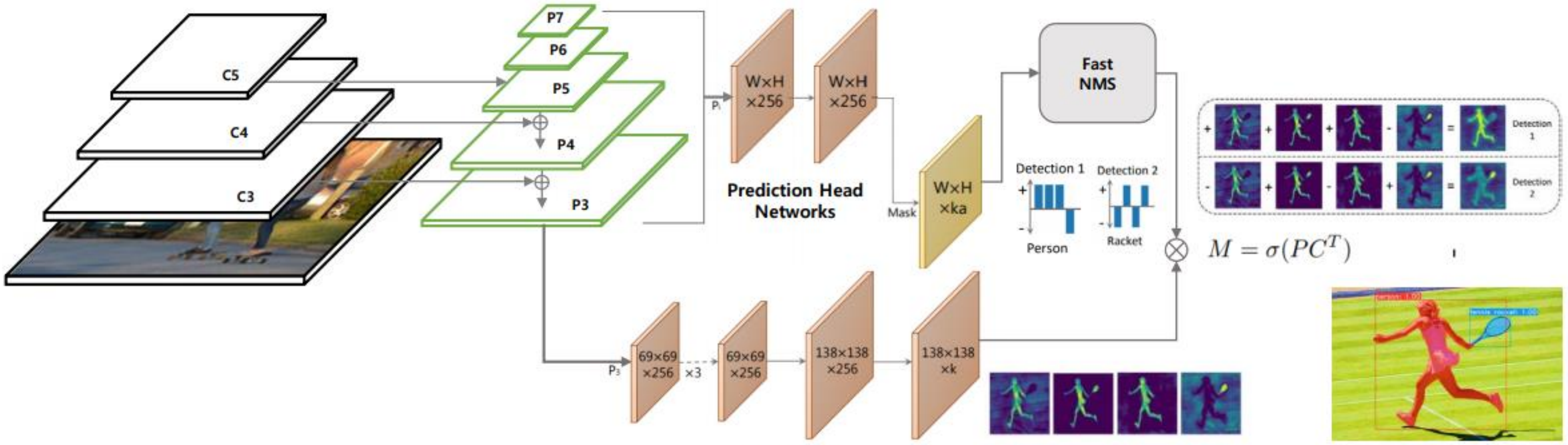
Unbounded, large overpowering activations

ReLU for interpretability

Overall Model Architecture



Overall Model Architecture

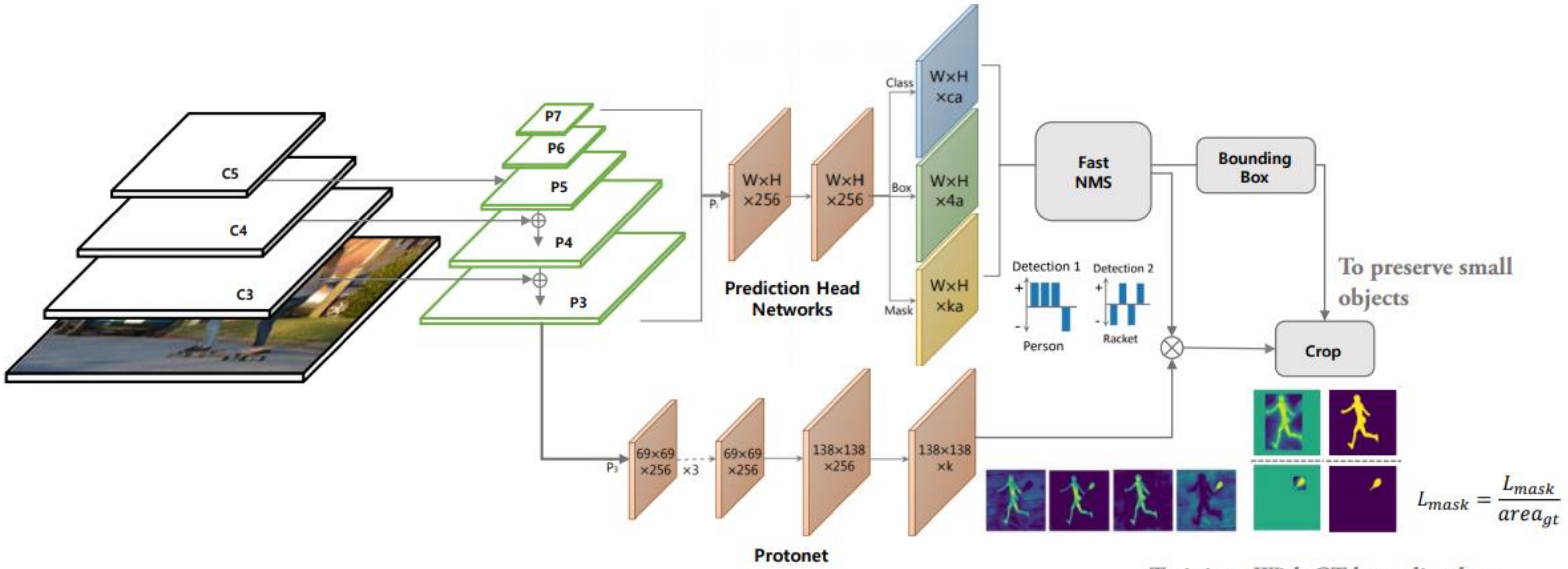


$M = \sigma(PC^T)$

Binary Cross Entropy

$L_{mask} = BCE(M, M_{gt})$

Overall Model Architecture



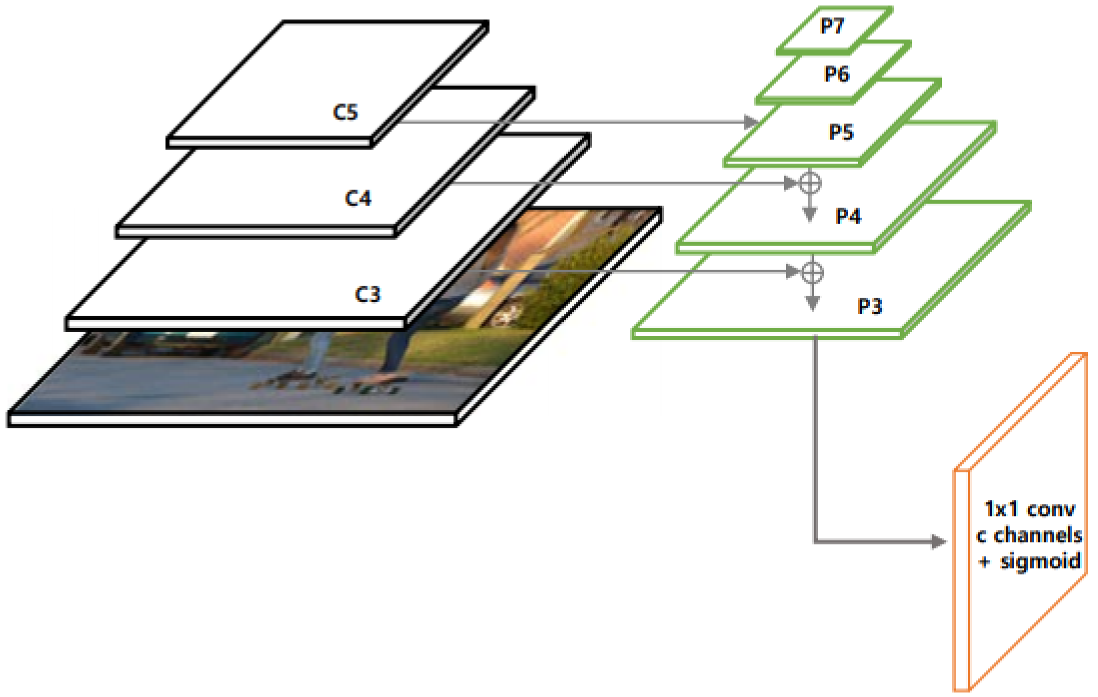
To preserve small objects

$$L_{mask} = \frac{L_{mask}}{area_{gt}}$$

Training : With GT bounding box

Test : With predicted bounding box

Overall Model Architecture



Only for training : Train underlying layers with Semantic Segmentation

Increase feature richness, **No inference time penalty**

The drawback of the two-stage architectures (such as Mask RCNN) are:

1. Two stage detectors have high accuracy but low performance
2. Dependent on Feature Localisation to generate/ produce

   masks of the objects

To address these issues, YOLACT uses a single stage detector extension which performs instance segmentation by breaking into subtasks , they forgo explicitly the localization step.

The network learns to localize masks on its own where visually , spatially and semantically similar instances appear in the prototypes .
The **number of prototype masks** in YOLACT is independent of the number of categories, this leads to distributed representation in the prototype space , this behavior leads to following advantages:

1. Some prototype spatially partition the image
2. Some localize the instances
3. Some detect instance contours
4. Some encode position-sensitive directional maps
5. Some do the combo of the above operations

YOLACT adds a mask branch to the one-stage detectors without an explicit localization step, where the complex task of instance segmentation is divided into two simpler, parallel tasks that can be assembled to form the final masks.

1. **First branch** obtains a set of image-sized “prototype masks” that do not depend on any one instance by using an FCN method.

2. **Second branch** adds an 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. Then by linearly combining the First branch and Second branch we generate the masks of instances which have be passed from NMS

Prototypes

The concept of using prototypes have been used extensively in the vision community , they are mainly used for obtaining the features whereas the current author has used to assemble masks for instance segmentation which are specific to each image then having global prototypes for entire dataset

Protonet: Network for Proto-type Generation

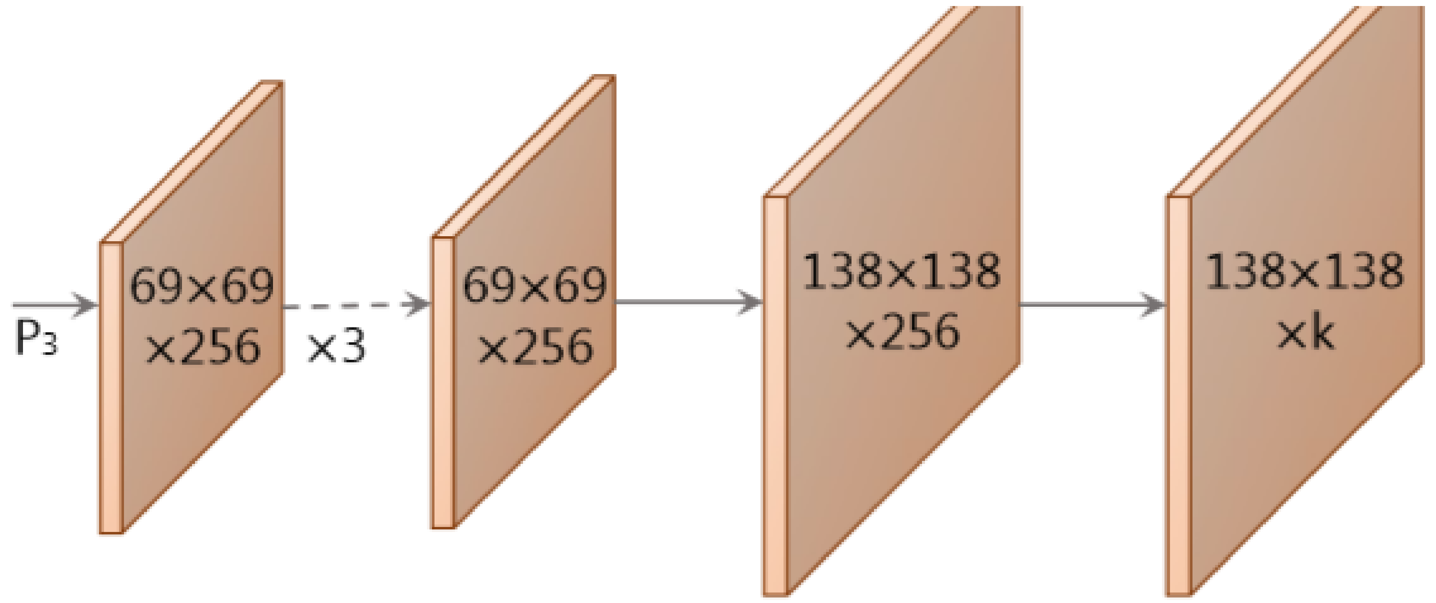
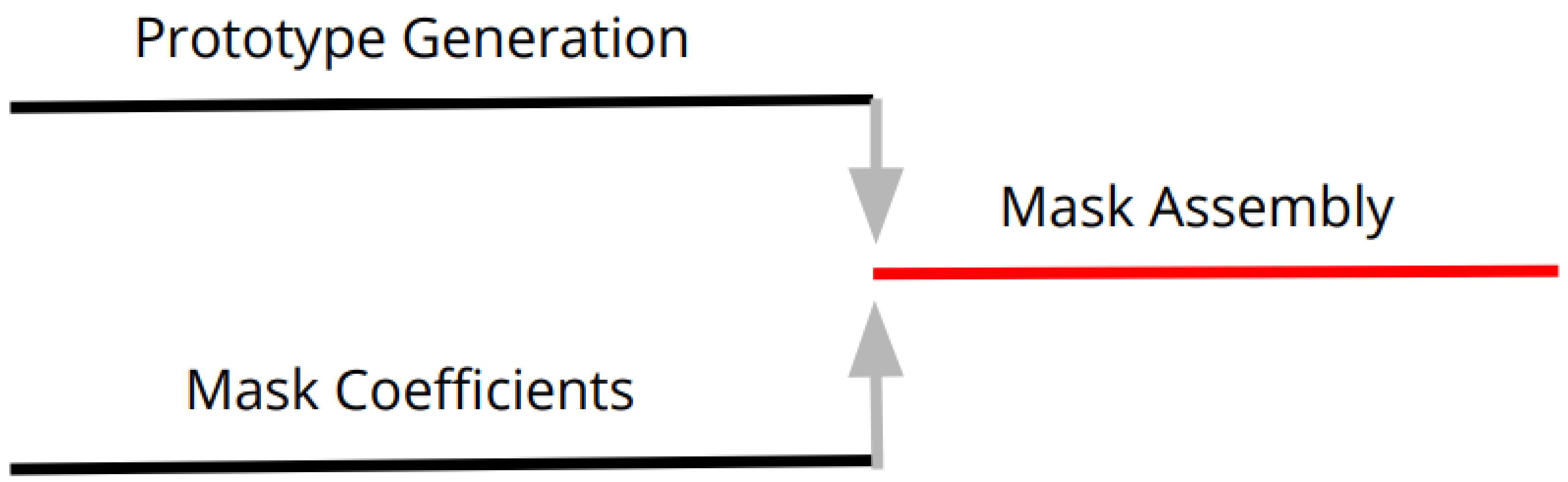


Fig. 3: **Protonet Architecture** The labels denote feature size and channels for an image size of 550×550. Arrows indicate 3×3 *conv* layers, except for the final *conv* which is 1×1. The increase in size is an upsample followed by a *conv*. Inspired by the mask branch in [2].

The prototype generation branch (protonet) predicts a set of k prototype masks for the entire image according to the following **design choices**:

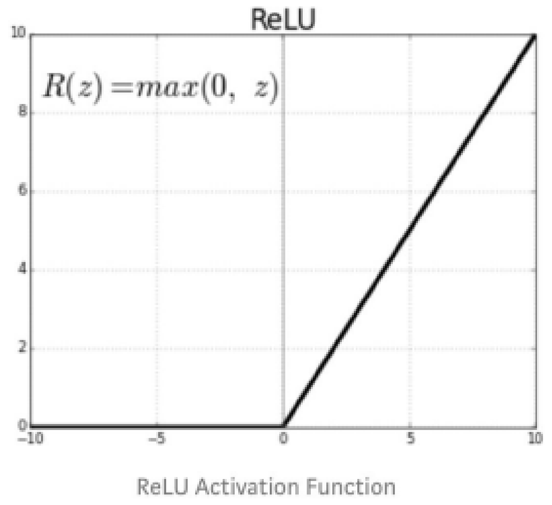
1. Taking protonet from deeper backbone features which produces robust and high quality masks so from FPN -P3 the last layer having k channels is considered**,** then it is up-sampled to one fourth the dimensions of the input image to increase performance on small objects.
2. Individual prototype losses are not considered explicitly but instead the final mask loss after assembly.
3. Relu or non -linearity operation is performed on the protonet’s output to keep it unbound as it allows the network to produce large, overpowering activation's on prototypes it is very confident about for the background

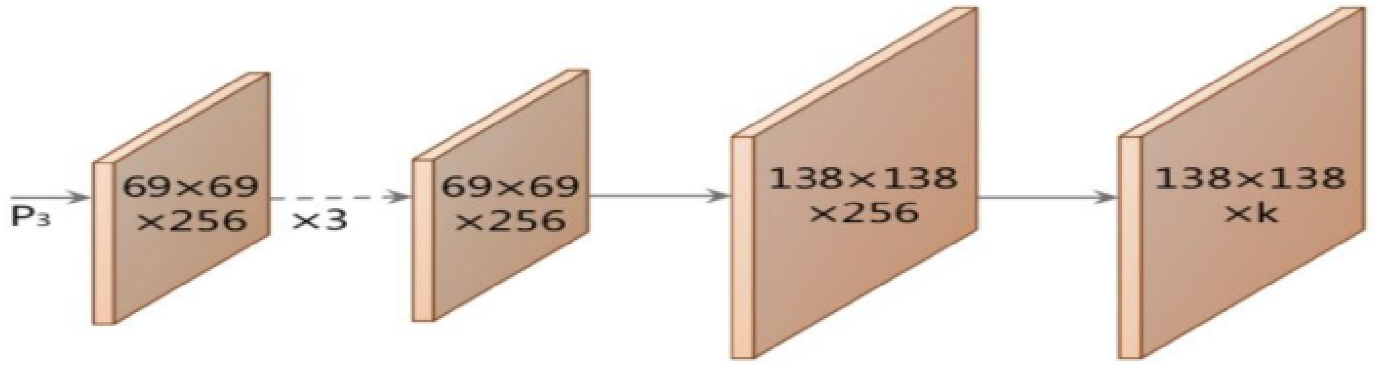


Slide credit to Daniel Bolya, Chong Zhou, Fanyi Xiao, Yong Jae Lee

Step 1 : Prototype Generation

- Also known as Pronet
- It is a Fully Convolutional Network
- Last layer produces k prototypes
- No explicit loss
- We leave the prototypes output being unbounded



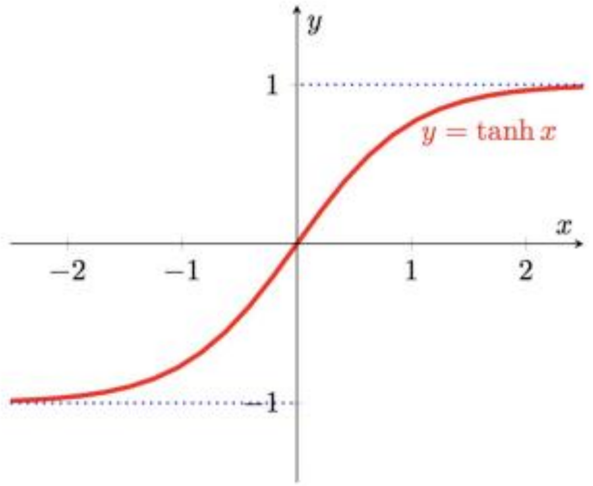


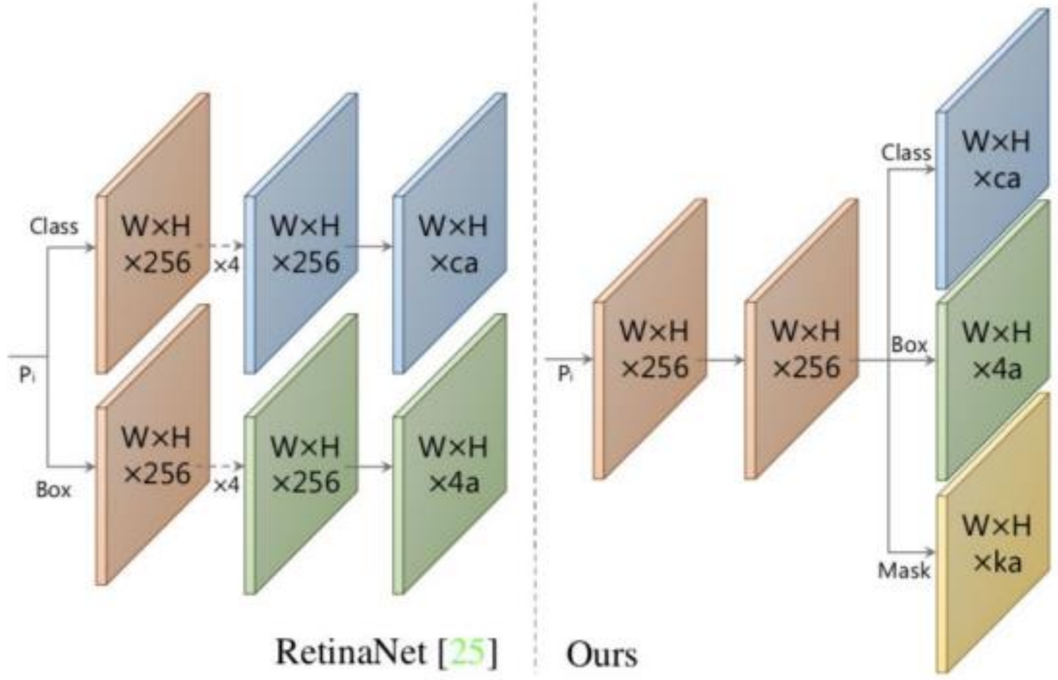
Slide credit to Daniel Bolya, Chong Zhou, Fanyi Xiao, Yong Jae Lee

Step 1 : Mask Coefficients (In Parallel)

- Produces c + 4 + k coefficients

- Use tanh on k mask coefficients





Slide credit to Daniel Bolya, Chong Zhou, Fanyi Xiao, Yong Jae Lee

Prediction Head

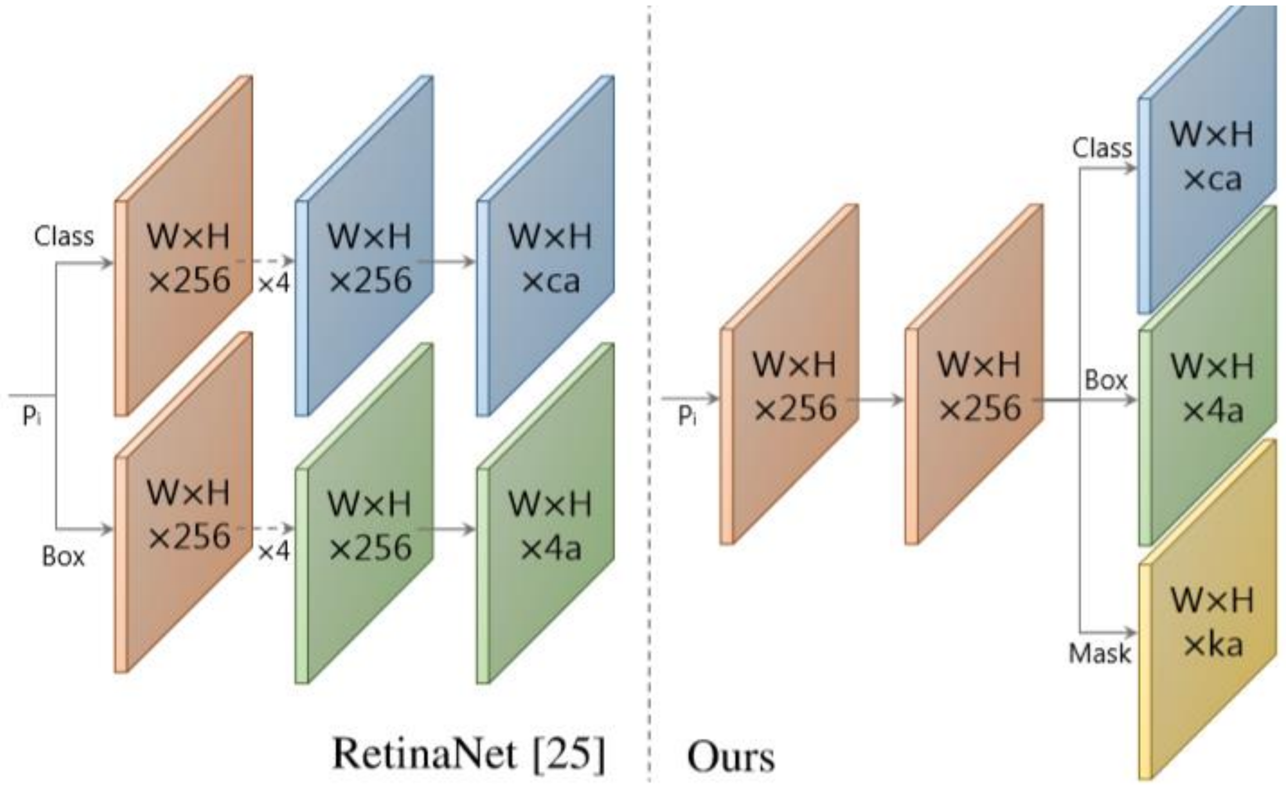


Fig. 4: **Head Architecture** We use a shallower prediction head than RetinaNet [25] and add a mask coefficient branch. This is for c classes, a anchors for feature layer P_i, and k prototypes. See Figure 3 for a key.

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

Mask Coefficients

In anchor based object detectors there are two branches in their prediction head.

1. To predict c class confidences
2. The other to predict 4 bounding box regressors.

To obtain the mask coefficient prediction, a third branch is simply added in parallel that predicts k mask coefficients, one corresponding to each prototype, thus instead of producing 4 + c coefficients per anchor, we produce 4 + c + k.

Mask Assembly

The mask assembly steps produce the instance masks are given below:

1. Combining the prototype branch and mask coefficient branch by using a linear combination of the former with the latter as coefficients.
2. Applying a sigmoid nonlinearity to produce the final masks.
3. The combination is done using using a single matrix multiplication and sigmoid: $M = \sigma(PC^T)$

where P is an h×w ×k matrix of prototype masks and C is a n × k matrix of mask coefficients for n instances surviving NMS and score thresholding.

Step 2 : Mask Assembly

- M => Mask
- P => Prototype (h x w x k)
- C => Mask Coefficients (n x k)
- Sigmoid

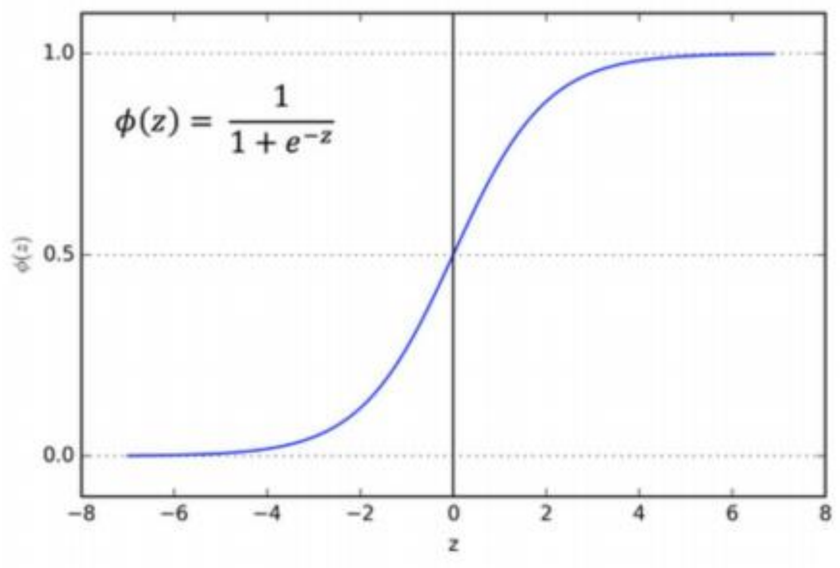


Fig: Sigmoid Function

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

Slide credit to Daniel Bolya, Chong Zhou, Fanyi Xiao, Yong Jae Lee

Protonet Behavior

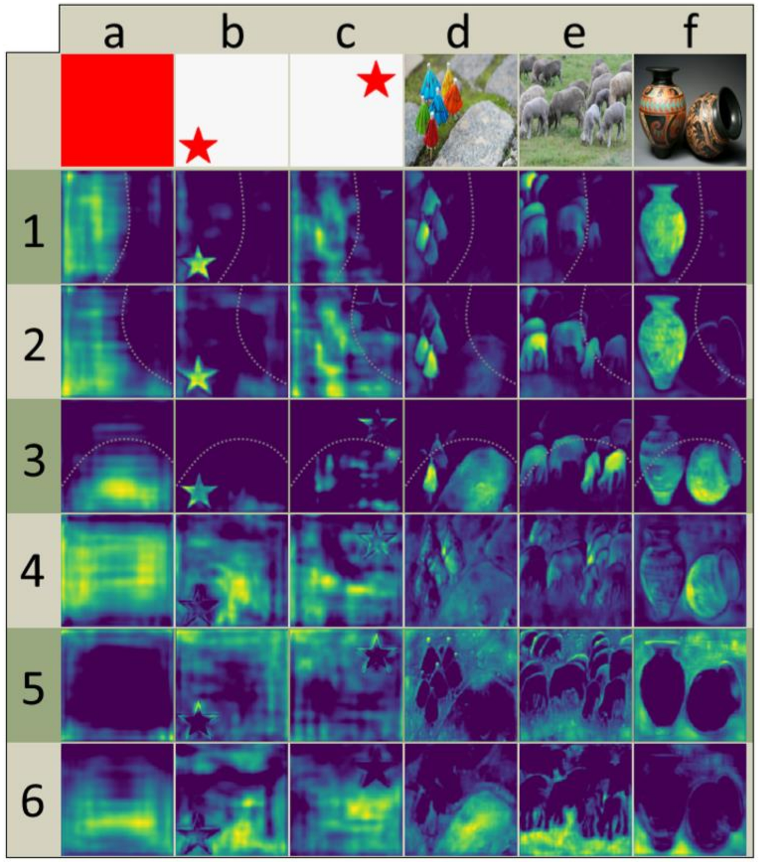
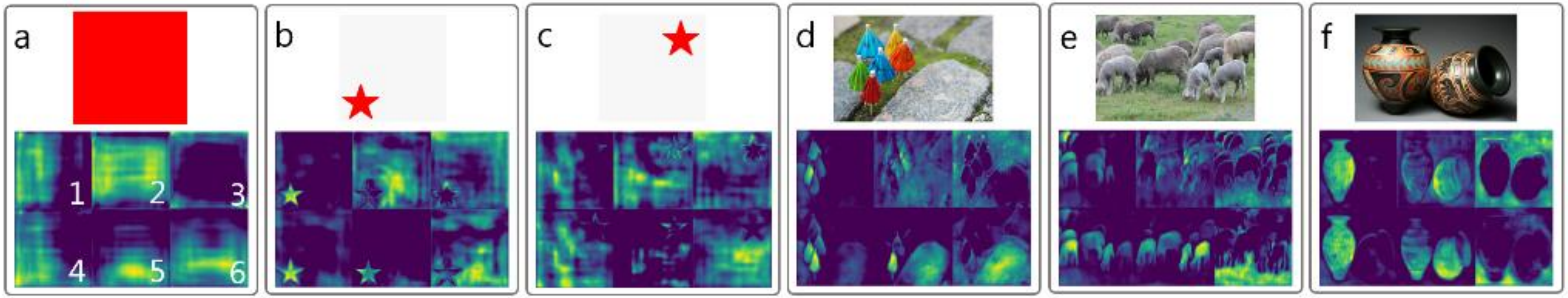


Fig. 5: **Prototype Behavior** The activations of the same six prototypes (y axis) across different images (x axis). Prototypes 1-3 respond to objects to one side of a soft, implicit boundary (marked with a dotted line). Prototype 4 activates on the bottom-left of objects (for instance, the bottom left of the umbrellas in image d); prototype 5 activates on the background and on the edges between objects; and prototype 6 segments what the network perceives to be the ground in the image. These last 3 patterns are most clear in images d-f.

Protonet Behavior



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.

OTHER IMPROVEMENTS

- **Fast NMS**

- **Semantic Segmentation Loss**

1. generating a set of prototypes masks

2. predicting the sub instance mask co efficient

Fast NMS

	1	2	3	4	5
1		X12	X13	X14	X15
2			X23	X24	X25
3				X34	X35
4					X45
5					

Standard NMS : In most object detectors NMS is used to suppress duplicate detections. The NMS operation is performed sequentially, that is for each of the c classes in the dataset, sort the detected boxes descending by confidence, and then for each detection remove all those with lower confidence than it that have an IoU overlap greater than some threshold. Though its is fast it is a large barrier when it comes to obtained 30 fps

Fast NMS: To remove the sequential nature of the traditional NMS the author introduces the Fast NMS where every instance can be decided to be kept or discarded in parallel , to perform this we use already -removed detections to suppress other detections, which is not possible in traditional NMS.

Steps of Fast NMS

The following steps are followed

1. Compute a $c \times n \times n$ pairwise IoU matrix X for the top n detections

2. Batched sorting in descending order by score for each of c classes.

3. Computation of IoU which can be easily vectorized. Then, find which detections to remove by checking if there are any higher-scoring detections with a corresponding IoU greater than some threshold t.

Implementation of Fast NMS :

1. First setting the lower triangle and diagonal of X to 0, wich can be performed in one batched triu call.

$$X_{kij} = 0 \qquad \forall k, j, i \geq j$$

2. Taking the column-wise max,to compute a matrix K of maximum IoU values for each detection.

$$K_{kj} = \max_i (X_{kij}) \qquad \forall k, j$$

3. Thresholding this matrix with t (K < t) will indicate which detections to keep for each class.

NMS (Non Maximum Suppression)

- Most Object Detectors uses traditional NMS or Sequential NMS
    - Makes sure each object is detected only once
    - Sort the detected boxes in descending order by Confidence
    - Discard values less than a certain threshold
    - Discard the values greater than threshold IoU values

- Fast-NMS
    - A new version of NMS
    - Decides to either discard or keep parallely
    - Allows already removed detections to suppress other detections

Slide credit to Daniel Bolya, Chong Zhou, Fanyi Xiao, Yong Jae Lee

Fast-NMS

- First we compute a pairwise IoU matrix (X) using,
  - c*n*n
  - c = classes
  - n = top n detections sorted in descending order
- Second, remove detections for
  - Confidence values less than a threshold ‘t’
  - IoU values greater than a threshold ‘t’

Slide credit to Daniel Bolya, Chong Zhou, Fanyi Xiao, Yong Jae Lee

Fast-NMS contd..

- Second step implemented using:
  - Setting lower triangle and diagonal of X to be 0.

$$X_{kij} = 0 \qquad \forall k, j, i \geq j$$

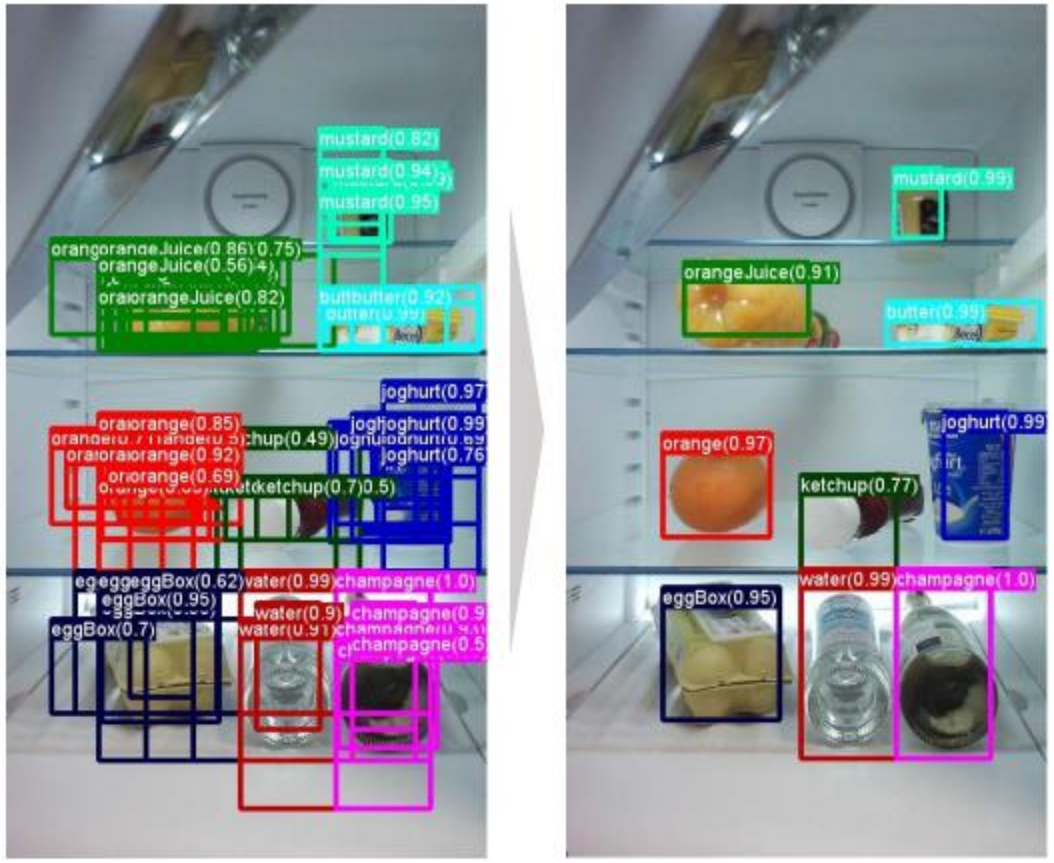
    - Where
      - X = pairwise IoU matrix
  - Taking the column-wise max
    - Where K = Matrix of maximum IoU values

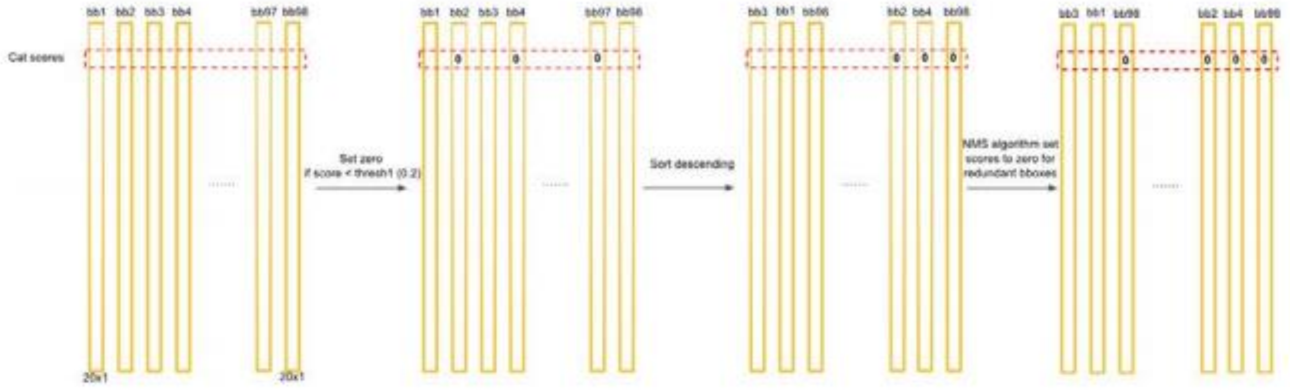
$$K_{kj} = \max_i (X_{kij}) \qquad \forall k, j$$

- Detections to keep given by threshold matrix t (K<t)

Slide credit to Daniel Bolya, Chong Zhou, Fanyi Xiao, Yong Jae Lee

NMS: Non-Maximum Suppression





Sort scores in descending error.
The boxes with an IoU > threshold with the boxes of high scores will be erased.

We keep the box if IoU < threshold

Segmentation Loss

- Since each pixel can be assigned to more than one class we use sigmoid and c channels
- This loss is given a weight of 1 and results in a +0.4 mAP boost.

Loss Function

Three losses are used to train the model:

1. classification loss Lcls
2. box regression loss L box
3. mask loss L mask

To compute mask loss, they simply take the pixel-wise binary cross entropy between assembled masks M and the ground truth masks

M gt : L mask = BCE(M, M gt).

Losses

Three types of losses:

- Classification Loss

$$L_{conf}(x,c) = -\sum_{i \in Pos}^{N} x_{ij}^{p} log(\hat{c}_i^p) - \sum_{i \in Neg} log(\hat{c}_i^0) \quad \text{where} \quad \hat{c}_i^p = \frac{\exp(c_i^p)}{\sum_p \exp(c_i^p)}$$

Confidence Loss

- Box Regression Loss

$$L_{1;smooth} = \begin{cases} |x| & \text{if } |x| > \alpha; \\ \frac{1}{|\alpha|} x^2 & \text{if } |x| \leq \alpha \end{cases}$$

$$L_{loc}(x,l,g) = \sum_{i \in Pos}^{N} \sum_{m \in \{cx,cy,w,h\}} x_{ij}^{k} \text{smooth}_{\text{L1}}(l_i^m - \hat{g}_j^m)$$

$$\hat{g}_j^{cx} = (g_j^{cx} - d_i^{cx})/d_i^w \qquad \hat{g}_j^{cy} = (g_j^{cy} - d_i^{cy})/d_i^h$$

$$\hat{g}_j^w = \log\left(\frac{g_j^w}{d_i^w}\right) \qquad \hat{g}_j^h = \log\left(\frac{g_j^h}{d_i^h}\right)$$

Localization Loss

Slide credit to Daniel Bolya, Chong Zhou, Fanyi Xiao, Yong Jae Lee

Losses (Continue…)

- Mask Loss - Pixel-wise Binary Cross Entropy

Mgt: Lmask = BCE(M, Mgt).

Slide credit to Daniel Bolya, Chong Zhou, Fanyi Xiao, Yong Jae Lee

YOLACT++

- Fast Mask Re-Scoring Network

- Deformable Convolution with Intervals

- Optimized Prediction Head

Fast Mask Re-Scoring Network

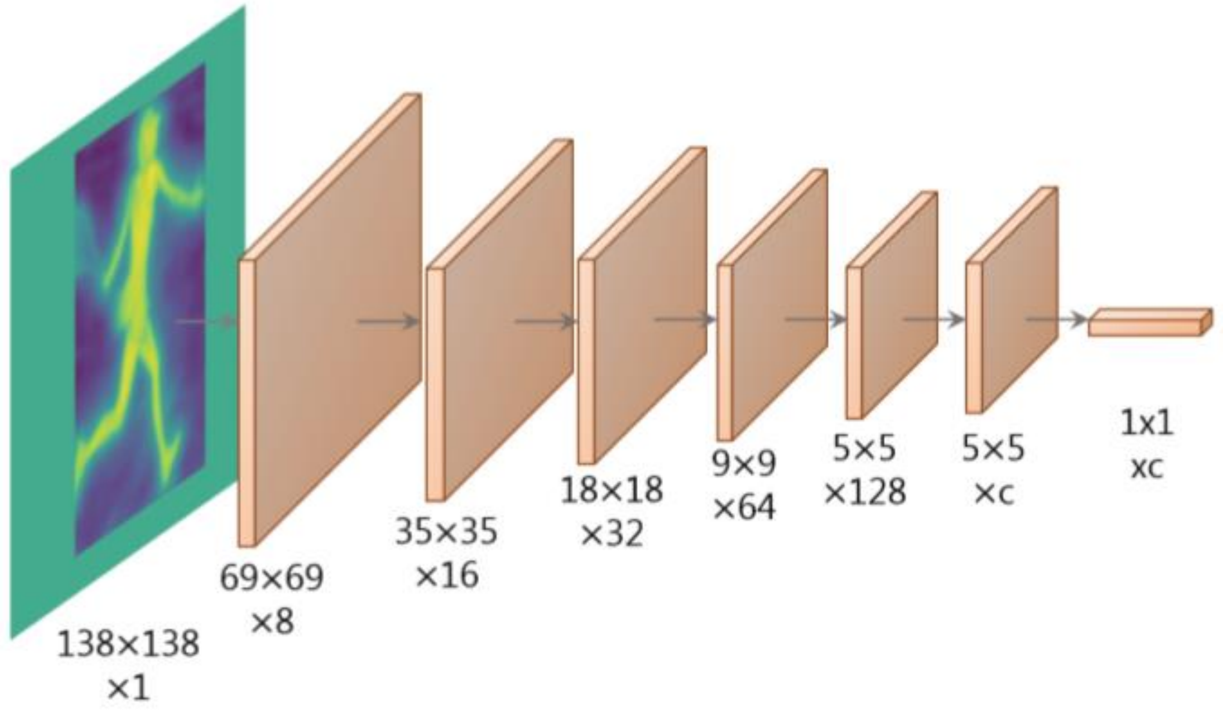


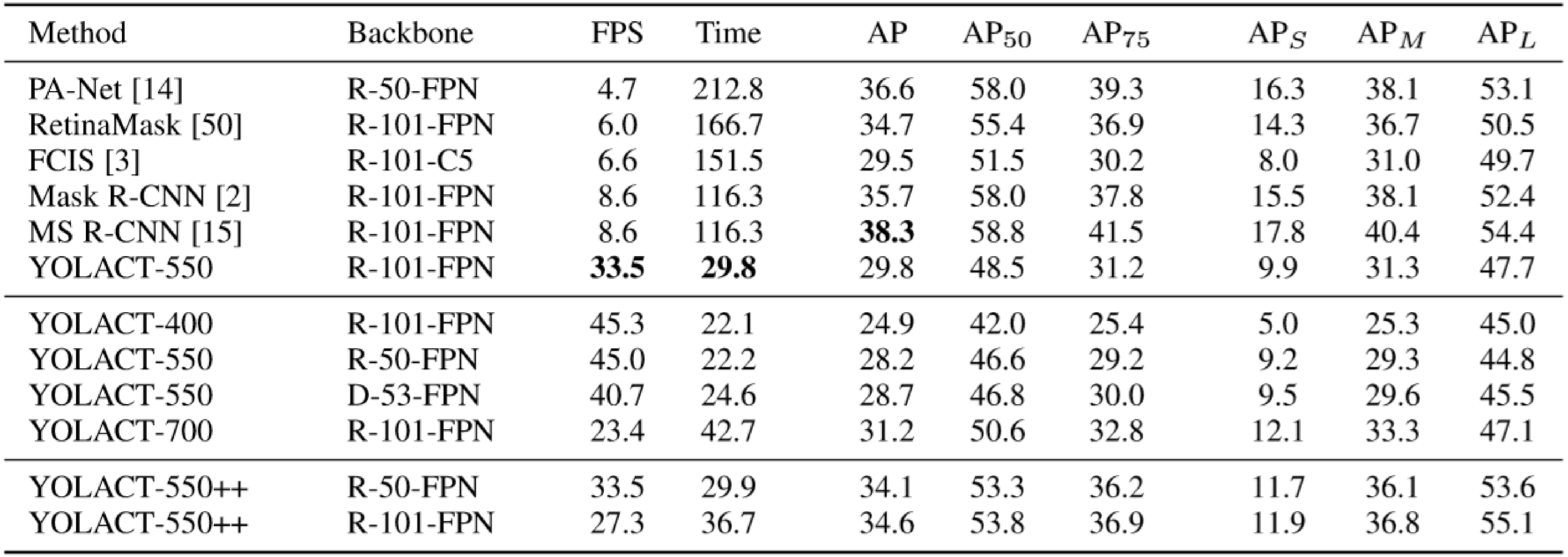
Fig. 6: **Fast Mask Re-scoring Network Architecture** Our mask scoring branch consists of 6 conv layers with ReLU non-linearity and 1 global pooling layer. Since there is no feature concatenation nor any fc layers, the speed overhead is only ~1 ms.

Optimized Prediction Head

- keeping the scales unchanged while increasing the anchor aspect ratios from [1,1/2,2] to [1,1/2,2,1/3,3]

- keeping the aspect ratios unchanged while increasing the scales per FPN level by threefold $([1\text{x},\ 2^{\frac{1}{3}}\text{x},\ 2^{\frac{2}{3}}\text{x}])$.

RESULTS



Method	Backbone	FPS	Time	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
PA-Net [14]	R-50-FPN	4.7	212.8	36.6	58.0	39.3	16.3	38.1	53.1
RetinaMask [50]	R-101-FPN	6.0	166.7	34.7	55.4	36.9	14.3	36.7	50.5
FCIS [3]	R-101-C5	6.6	151.5	29.5	51.5	30.2	8.0	31.0	49.7
Mask R-CNN [2]	R-101-FPN	8.6	116.3	35.7	58.0	37.8	15.5	38.1	52.4
MS R-CNN [15]	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
YOLACT-550++	R-50-FPN	33.5	29.9	34.1	53.3	36.2	11.7	36.1	53.6
YOLACT-550++	R-101-FPN	27.3	36.7	34.6	53.8	36.9	11.9	36.8	55.1

Yolact and Yolact++

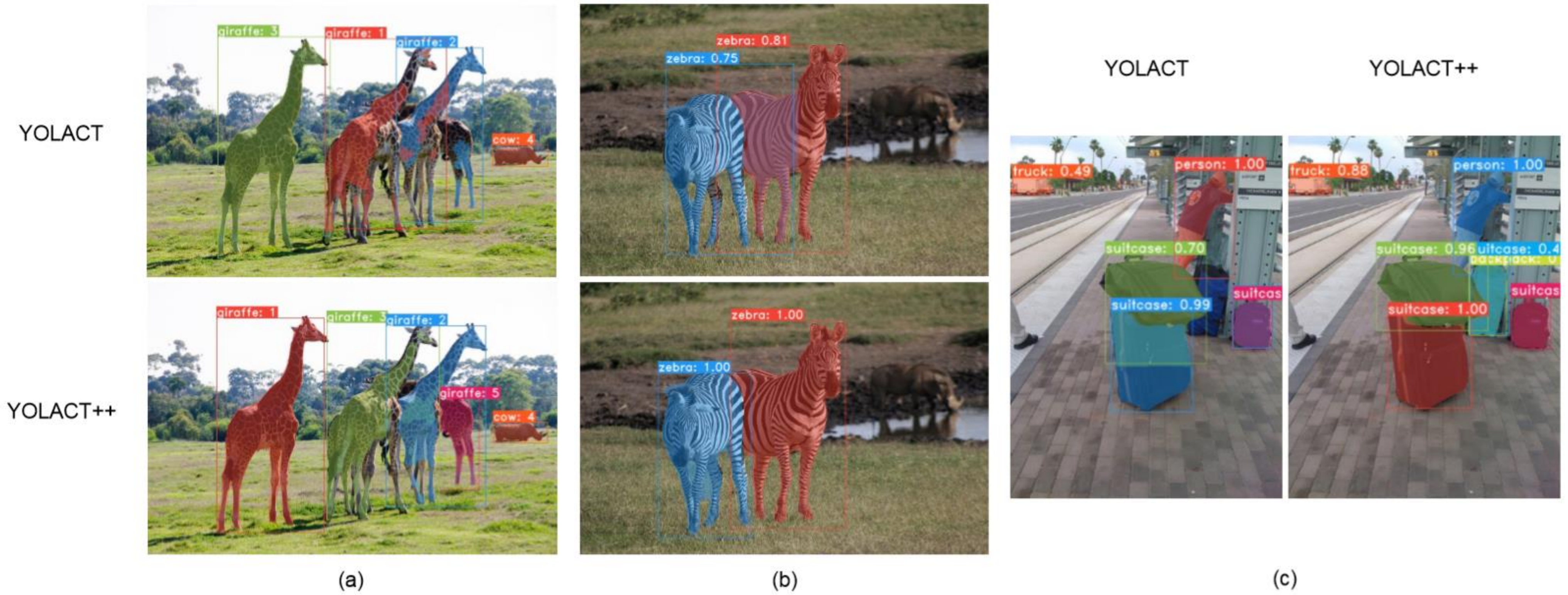
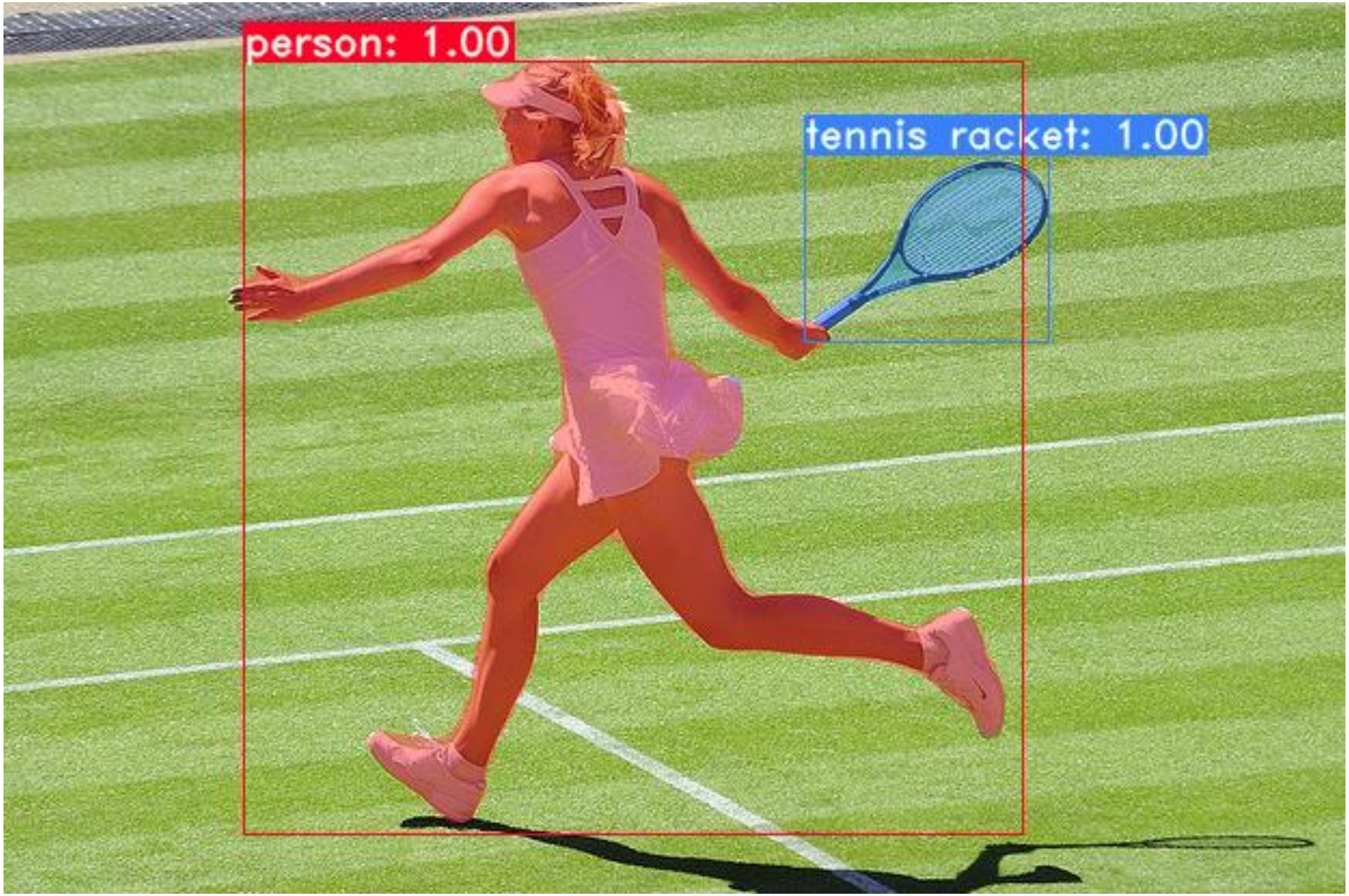
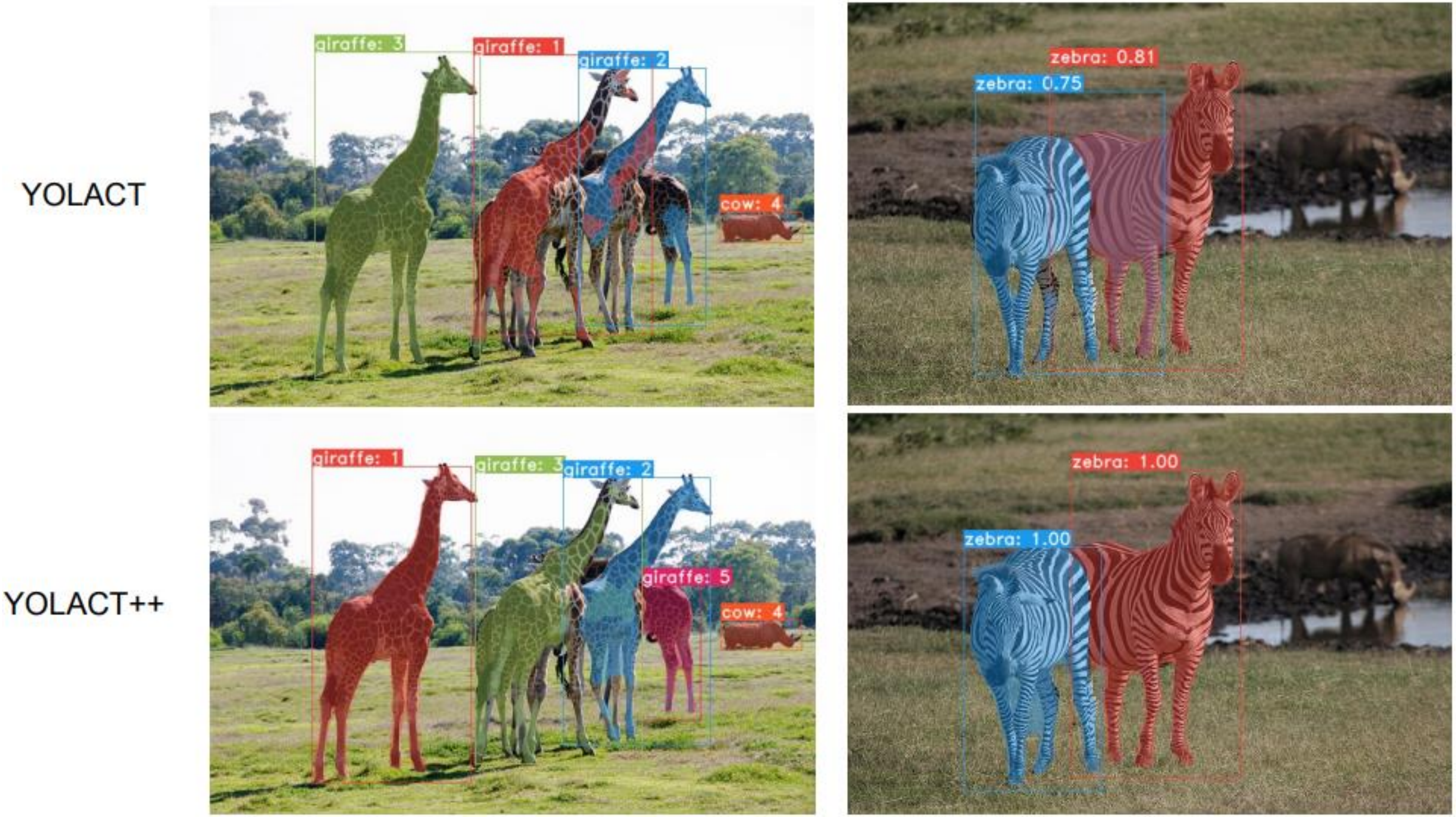


Fig. 10: **YOLACT vs. YOLACT++** (a) shows the rank of each detection in the image. As YOLACT++ has a fast mask re-scoring branch, its detections with better masks are ranked higher than those of YOLACT (see the leftmost giraffe). Since YOLACT++ is equipped with deformable convolutions in the backbone and has a better anchor design, the box recall, mask quality, and classification confidence are all increased. Specifically, (b) shows that both the box prediction and instance segmentation mask of the left zebra is more precise. (c) shows increased detection recall and improved class confidence scores.


person: 1.00
tennis racket: 1.00


YOLACT
giraffe: 3
giraffe: 1
giraffe: 2
cow: 4
zebra: 0.81
zebra: 0.75
YOLACT++
giraffe: 1
giraffe: 3
giraffe: 2
giraffe: 5
cow: 4
zebra: 1.00
zebra: 1.00

The **advantages of YOLACT** include:

1. Lightweight assembly process due to parallel structure
2. Marginal amount of computational overhead to one-stage detectors like ResNet101
3. Masks quality are high
4. Generic concept of adding of generating prototypes and mask coefficients

Why is YOLACT faster ?

- Single Stage Model

- Prototype Masks and Mask Coefficients are calculated parallely and independently.

- Other methods have an explicit localization step (Ex : ROIAlign in Mask R-CNN)

- YOLACT learns about localizing instances by itself and bypasses the explicit localization step.