Segmentation

AI 222 Advanced Machine Learning

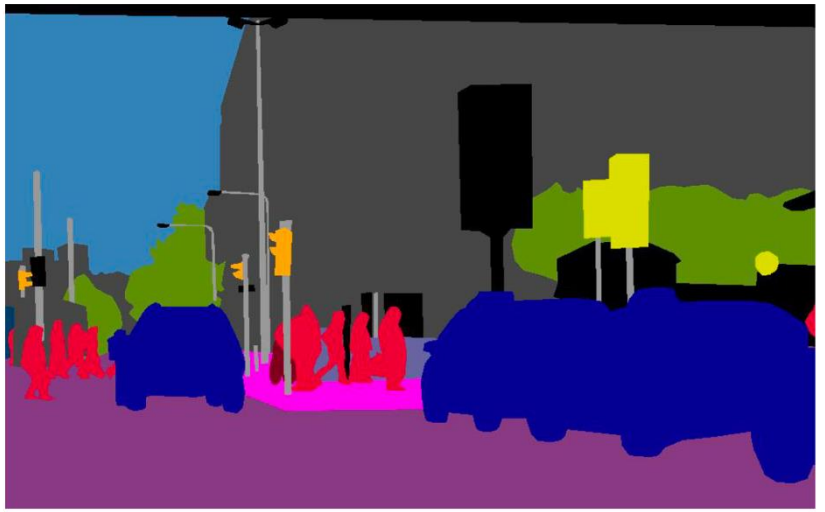
Rowel Atienza
University of the Philippines
github.com/roatienza

2024

Segmentation

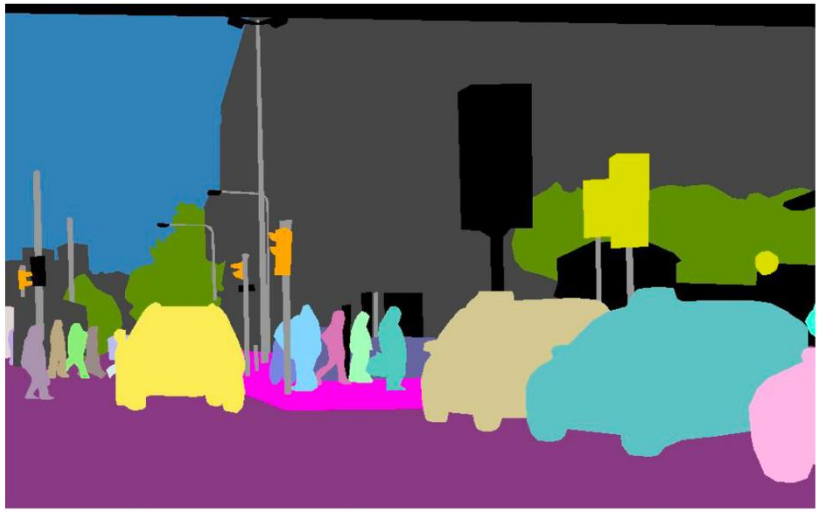
a) Ground truth image, b) Semantic segmentation: per pixel class label, c) Instance segmentation: per object mask and class label, d) Panoptic segmentation: per pixel class + instance segmentation

(a) image



(b) semantic segmentation

(c) instance segmentation

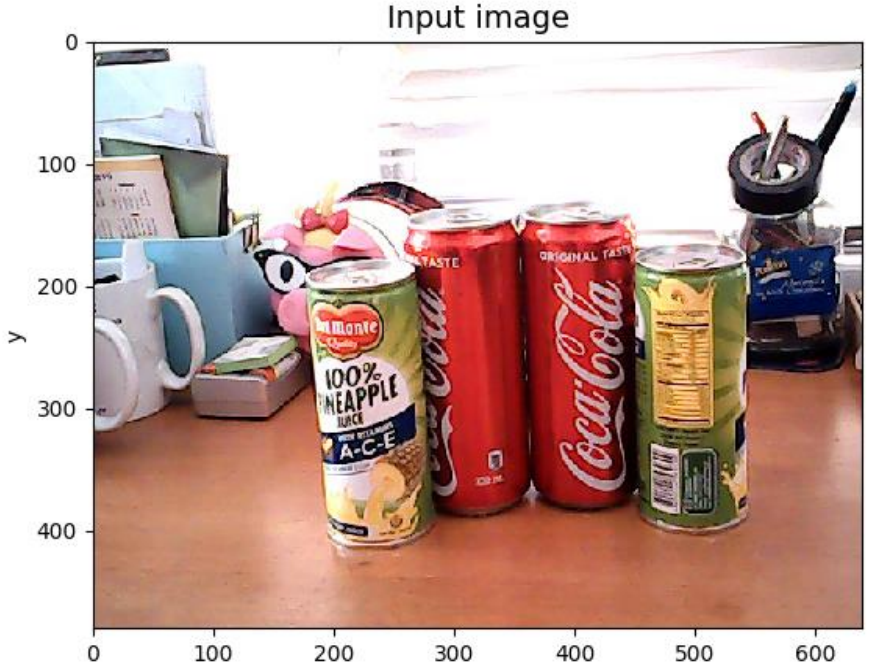


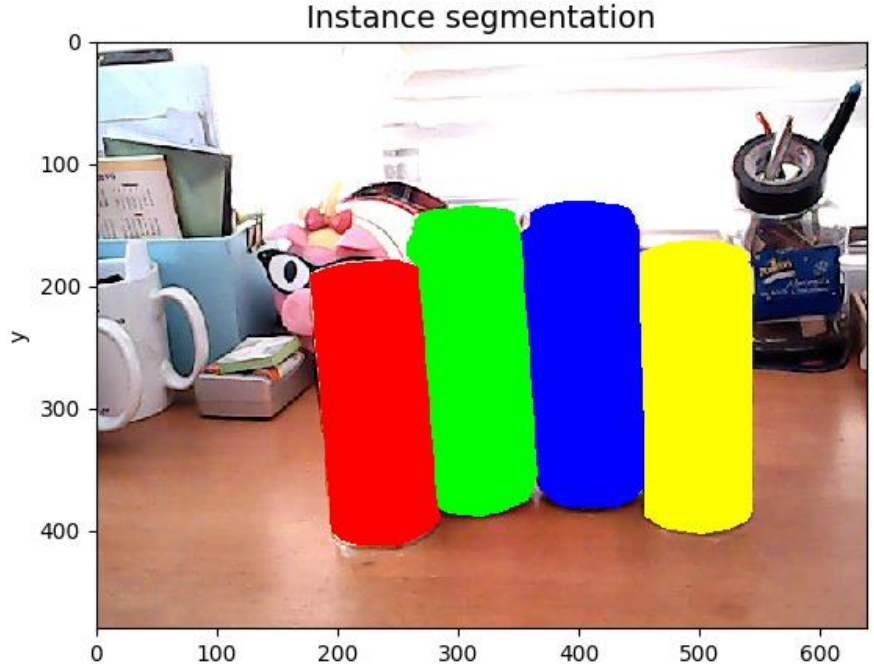
(d) panoptic segmentation

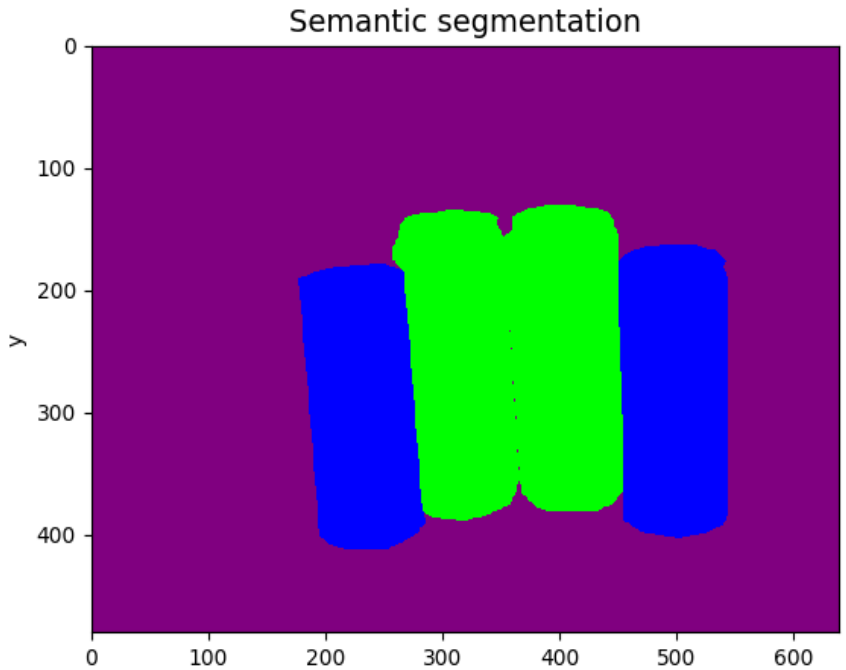
Kirillov, Alexander, et al. "Panoptic segmentation." *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019.

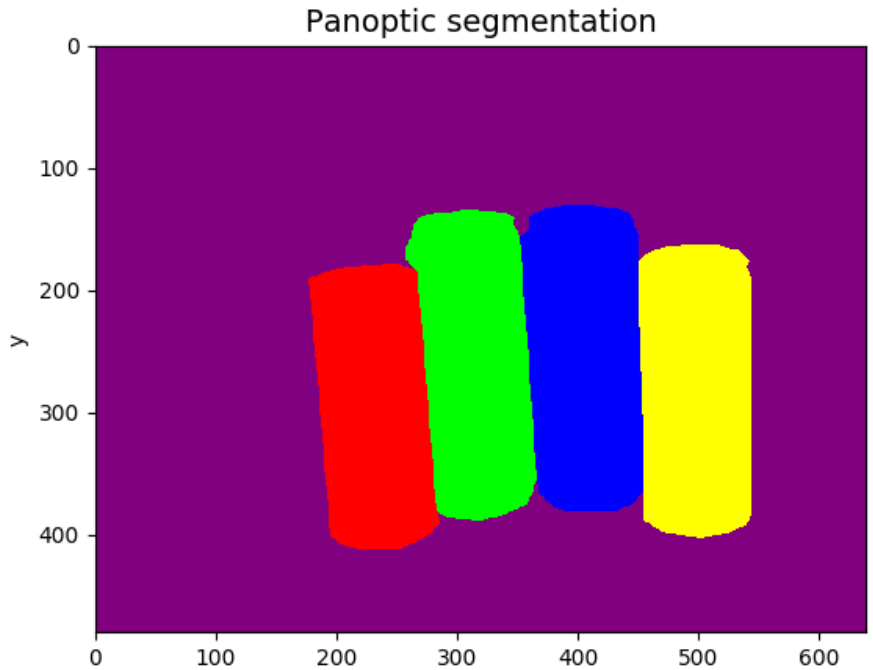
Stuff vs Things in Panoptic Segmentation

- Things – countable objects such as people, animals, tools – received the dominant share of attention.
- Stuff – amorphous regions of similar texture or material such as grass, sky, road.


Input image
0
100
200
300
400
y
0
100
200
300
400
500
600


Instance segmentation
0
100
200
300
400
y
0
100
200
300
400
500
600


Semantic segmentation
0
100
200
300
400
y
0
100
200
300
400
500
600

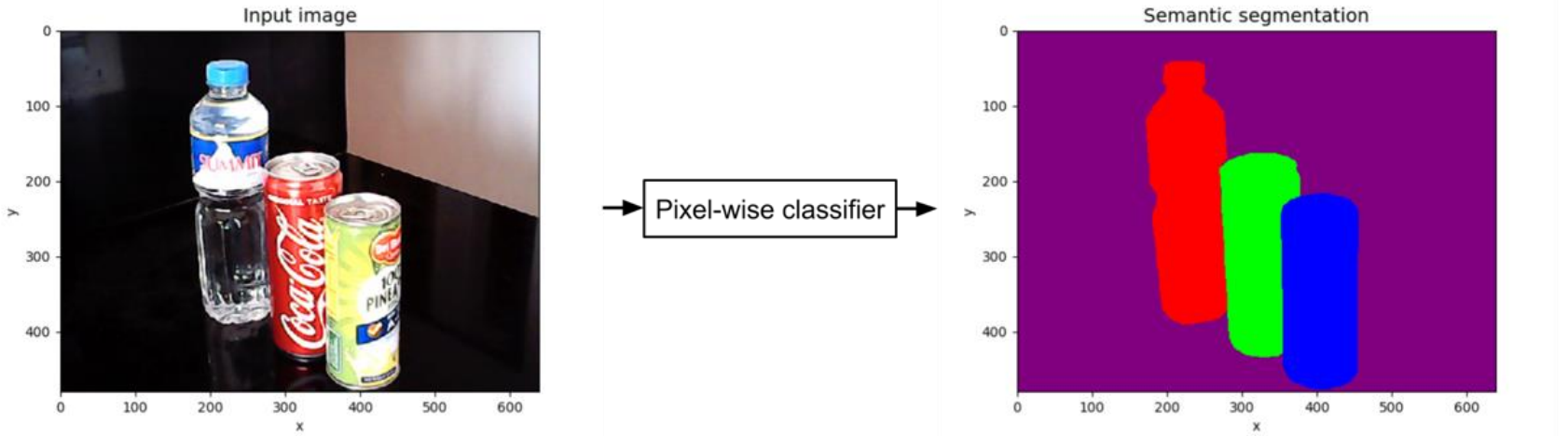

Panoptic segmentation
0
100
200
300
400
y
0
100
200
300
400
500
600

FCN

Long, Jonathan, Evan Shelhamer, and Trevor Darrell. "Fully convolutional networks for semantic segmentation." Proceedings of the IEEE conference on computer vision and pattern recognition. 2015.

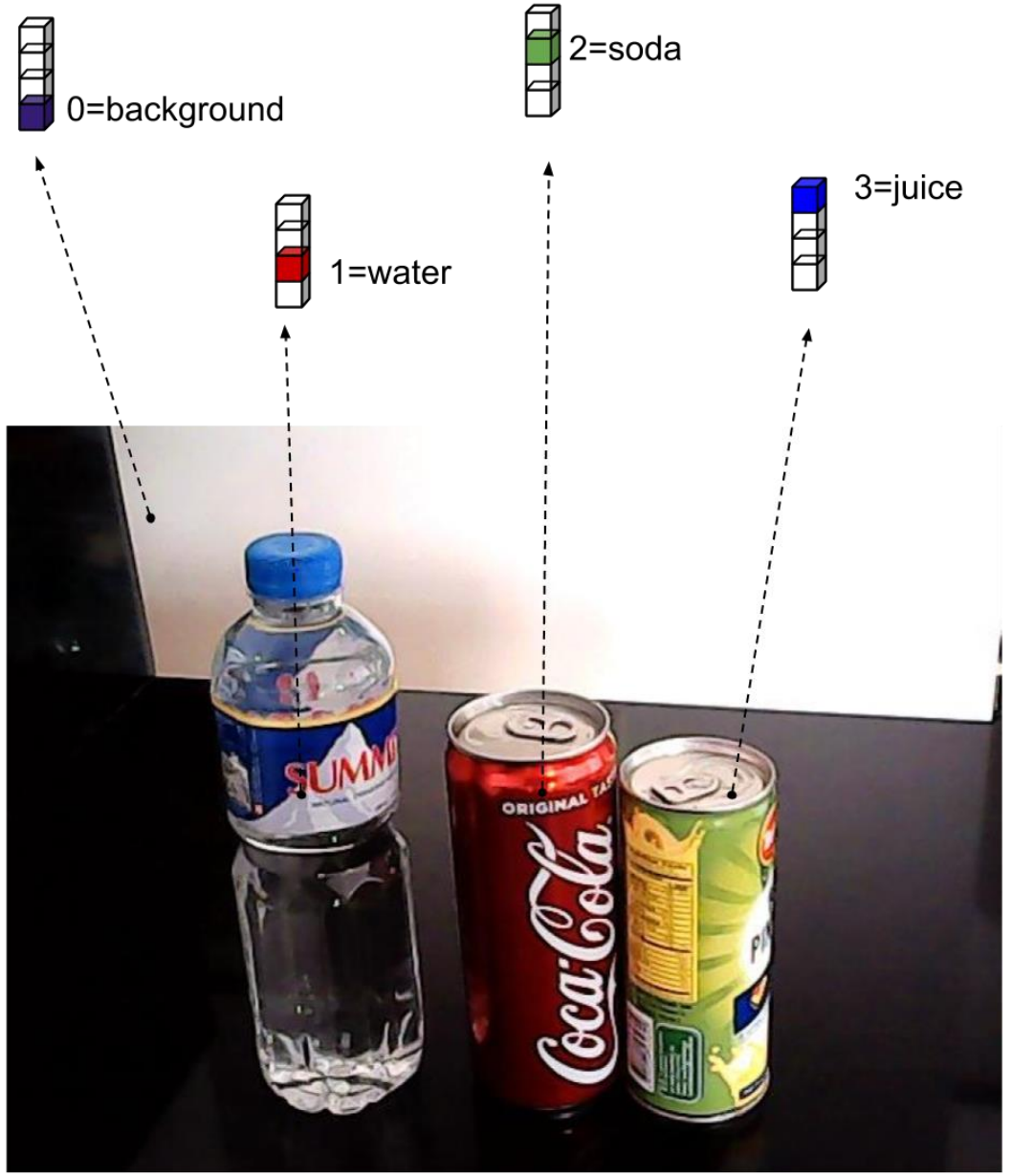
Semantic Segmentation

Semantic Segmentation



Assume 3 objects: Soda can, water bottle and juice can. Last object is the background

Semantic Segmentation



Semantic segmentation is per pixel classification

FCN

- Fully convolutional end-to-end and pixel-to-pixel semantic segmentation
- Uses many techniques that are common now but novel then (2015)
  - Skip connection
  - Upsampling and combining features from different levels
  - Fully convolutional
  - Pixel-wise loss

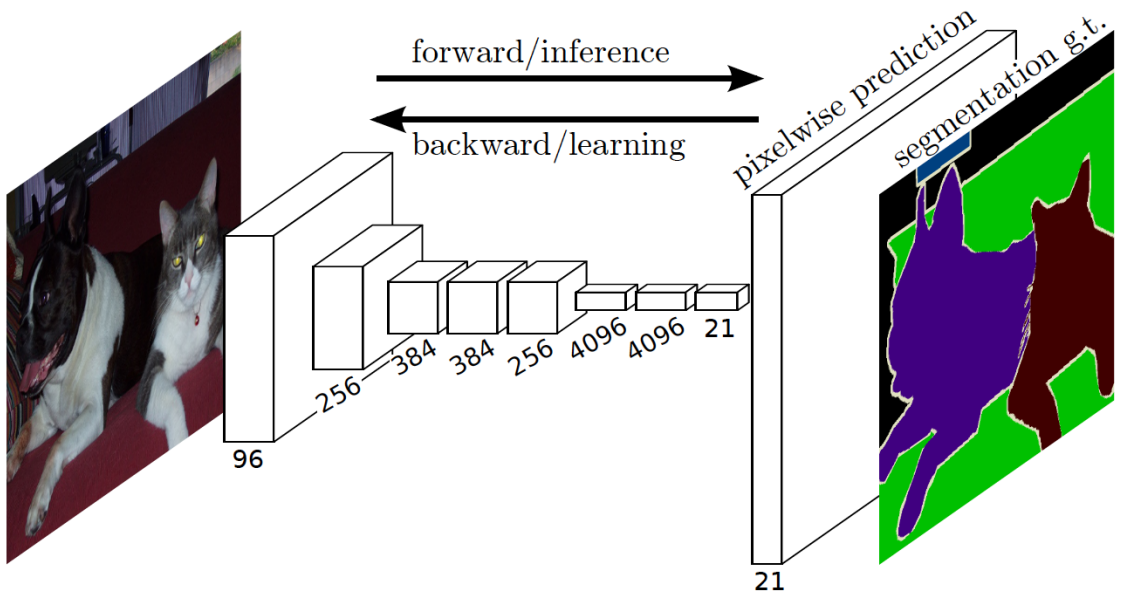


Figure 1. Fully convolutional networks can efficiently learn to make dense predictions for per-pixel tasks like semantic segmentation.

FCN

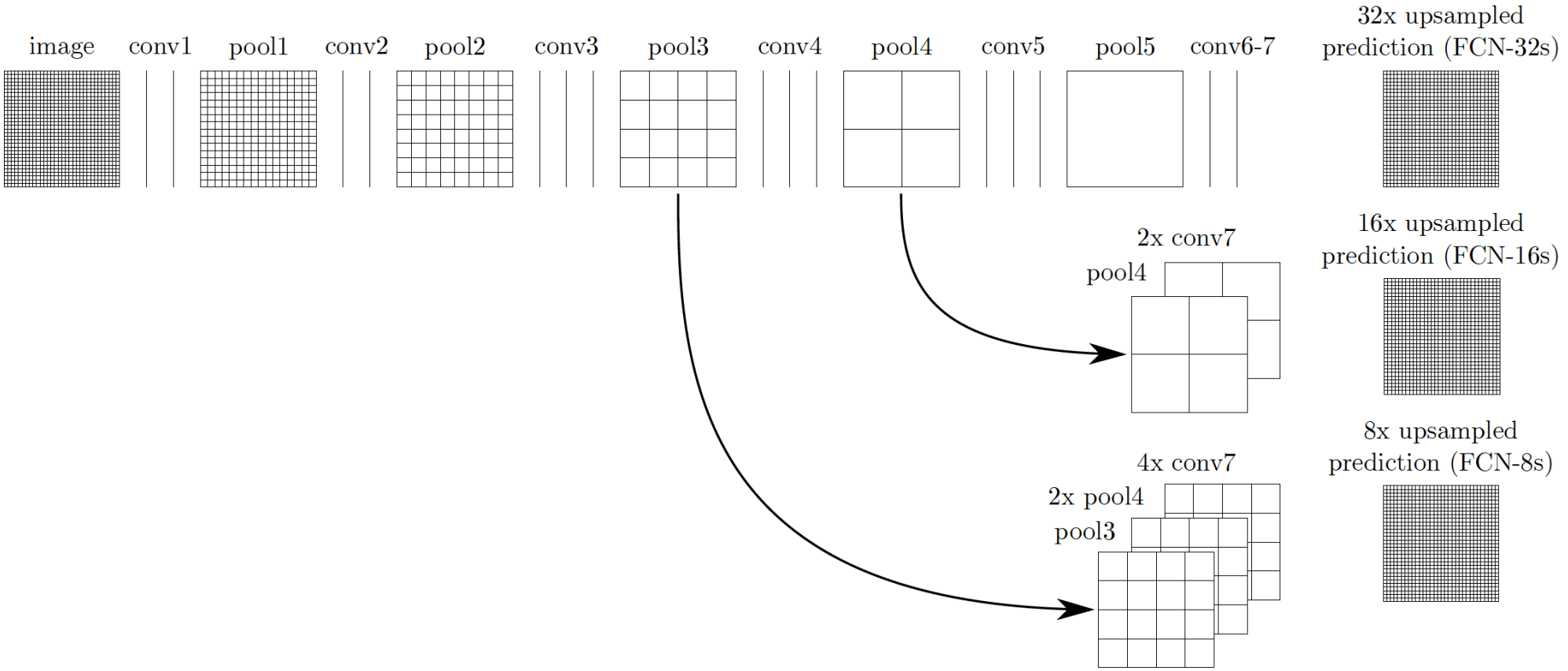


Figure 3. Our DAG nets learn to combine coarse, high layer information with fine, low layer information. Pooling and prediction layers are shown as grids that reveal relative spatial coarseness, while intermediate layers are shown as vertical lines. First row (FCN-32s): Our single-stream net, described in Section 4.1, upsamples stride 32 predictions back to pixels in a single step. Second row (FCN-16s): Combining predictions from both the final layer and the `pool4` layer, at stride 16, lets our net predict finer details, while retaining high-level semantic information. Third row (FCN-8s): Additional predictions from `pool3`, at stride 8, provide further precision.

Loss Function and Performance

- Categorical cross entropy loss per pixel
- Pascal VOC 2012 test: 62.2 IoU

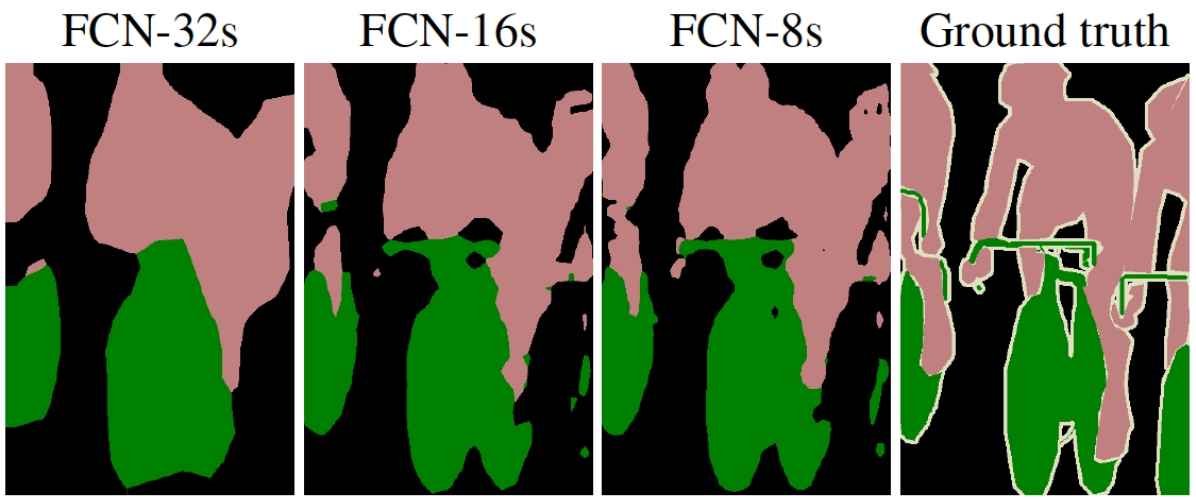


Figure 4. Refining fully convolutional nets by fusing information from layers with different strides improves segmentation detail. The first three images show the output from our 32, 16, and 8 pixel stride nets (see Figure 3).

PSPNet

Zhao, Hengshuang, et al. "Pyramid scene parsing network." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017.

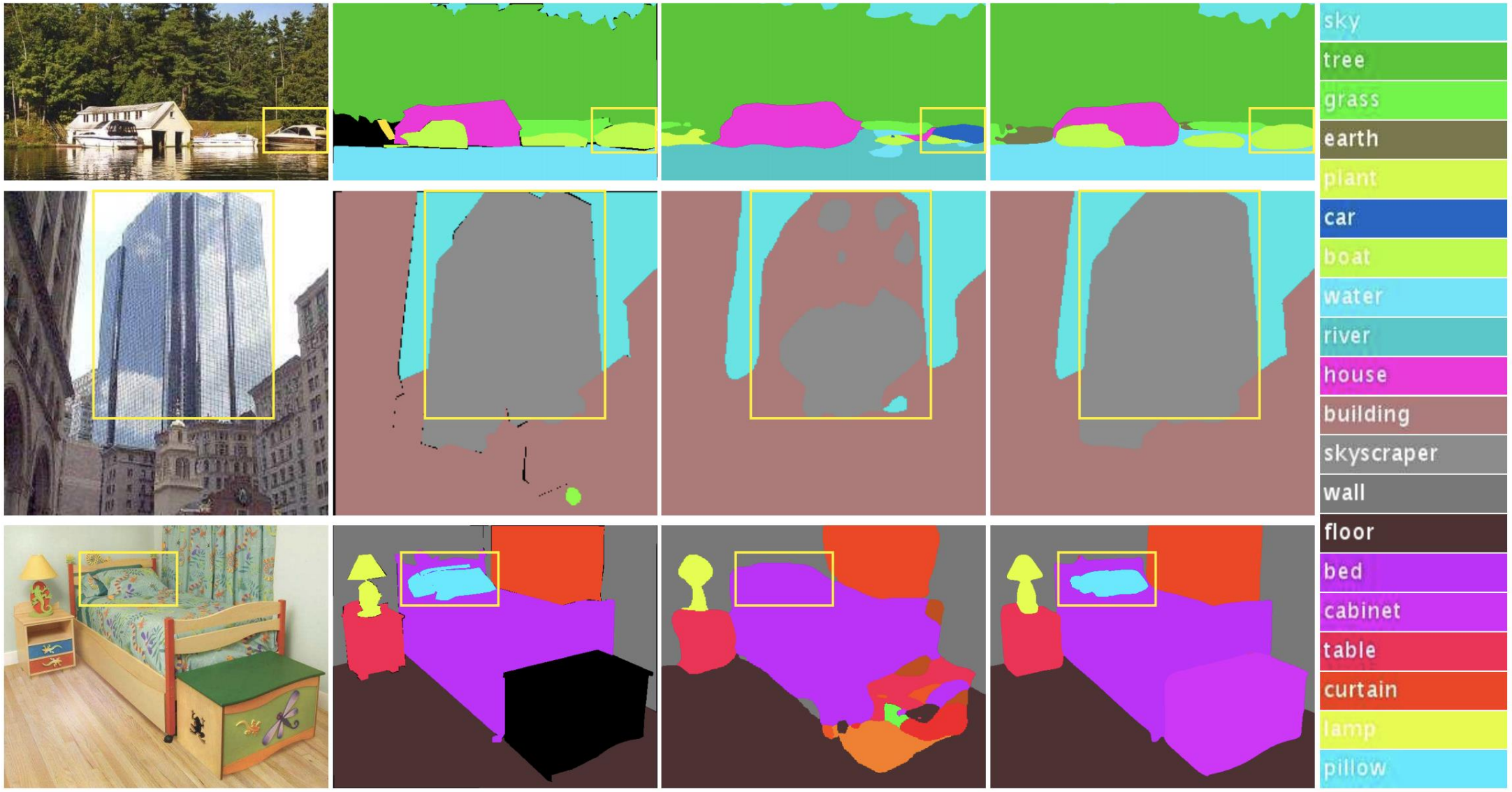
PSPNet

- FCN does not utilize global scene category
- Extend the pixel-level feature to the specially designed global pyramid pooling one
- a hierarchical global prior, containing information with different scales and varying among different sub-regions called pyramid pooling module
- 1st place on PASCAL VOC 2012 semantic segmentation benchmark
- 1st place on urban scene Cityscapes data

Common Failures in Semantic Segmentation

- Mismatched Relationship – car on lake
- Confusion Categories - mountain vs hill, skyscraper vs building
- Inconspicuous Classes – small objects in the scene like streetlight or pillow on a bed

Observations



(a) Image (b) Ground Truth (c) FCN (d) PSPNet (e) ColorMap

Figure 2. Scene parsing issues we observe on ADE20K [43] dataset. The first row shows the issue of mismatched relationship – cars are seldom over water than boats. The second row shows confusion categories where class "building" is easily confused as "skyscraper". The third row illustrates inconspicuous classes. In this example, the pillow is very similar to the bed sheet in terms of color and texture. These inconspicuous objects are easily misclassified by FCN.

Pyramid pooling module learns both global and local features for semantic segmentation

PSPNet

Global average pooling provides global contextual prior

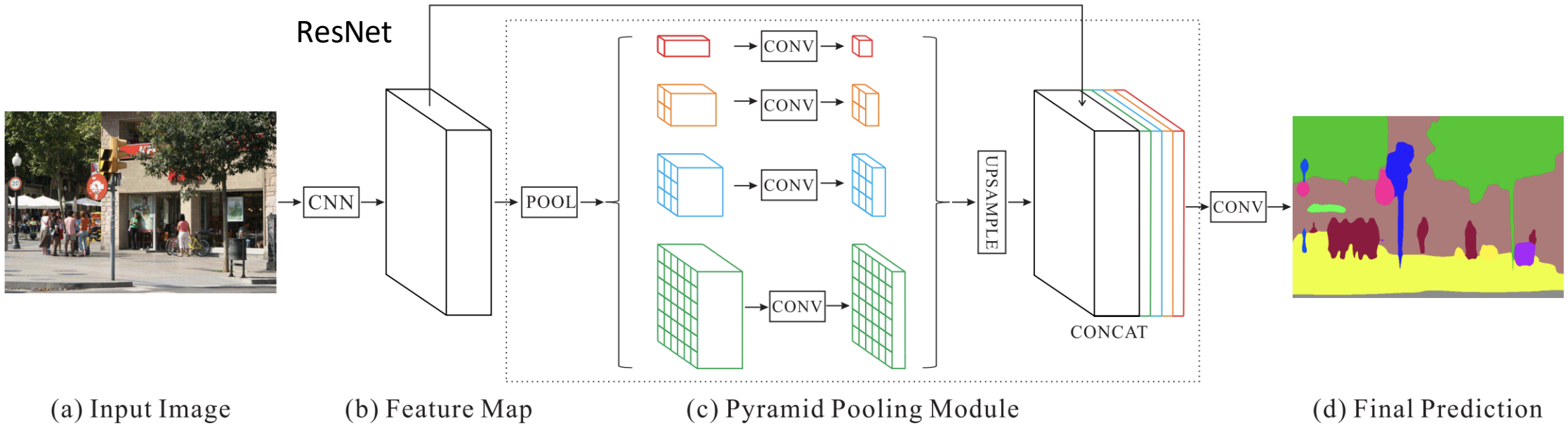


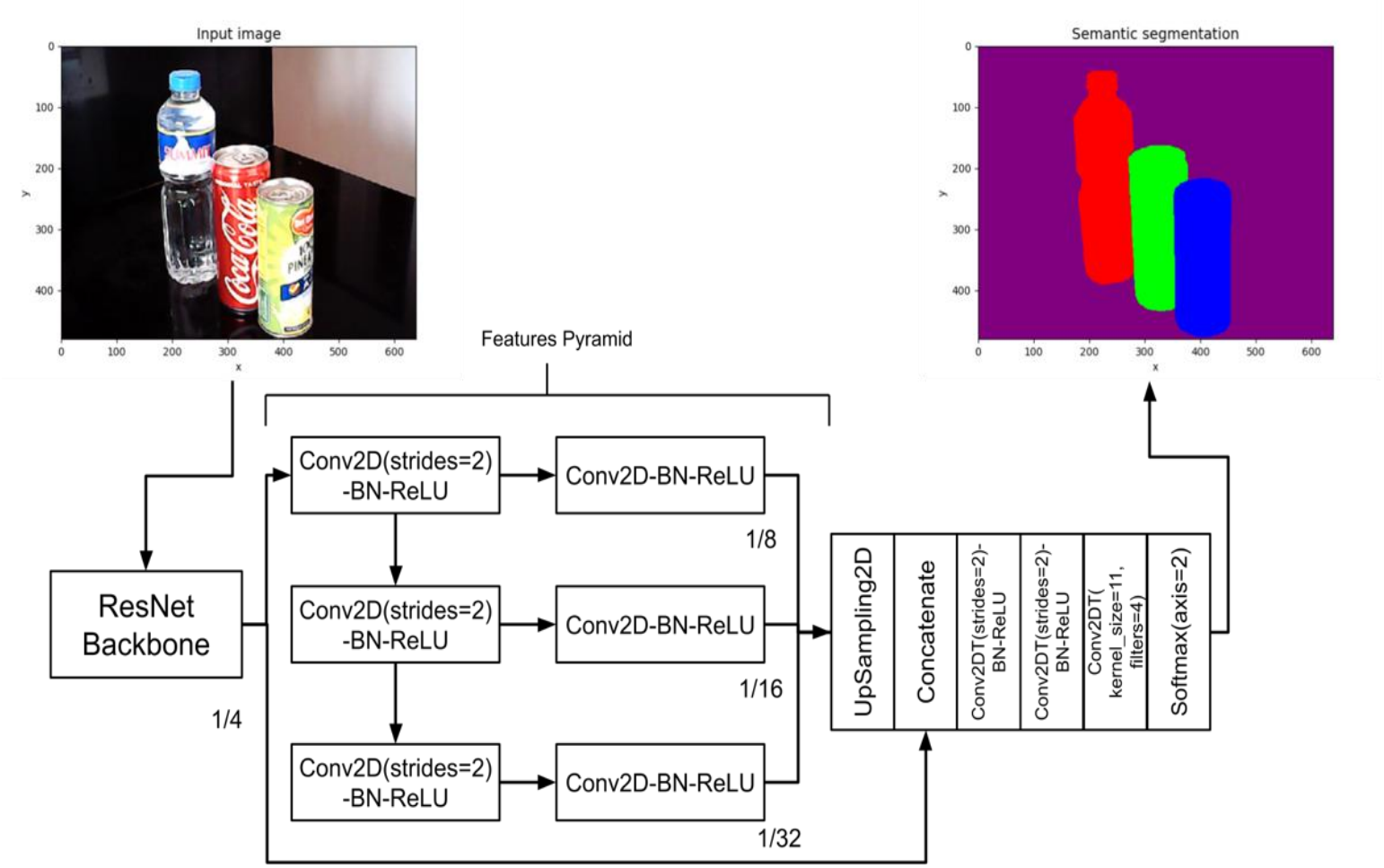
Figure 3. Overview of our proposed PSPNet. Given an input image (a), we first use CNN to get the feature map of the last convolutional layer (b), then a pyramid parsing module is applied to harvest different sub-region representations, followed by upsampling and concatenation layers to form the final feature representation, which carries both local and global context information in (c). Finally, the representation is fed into a convolution layer to get the final per-pixel prediction (d).

PSPNet

- Pascal VOC 2012 dataset: 85.4mIoU

Figure 7. Visual improvements on PASCAL VOC 2012 data. PSP-Net produces more accurate and detailed results.

FCN + PSPNet Sample Implementation



https://github.com/PacktPublishing/Advanced-Deep-Learning-with-Keras/tree/master/chapter12-segmentation

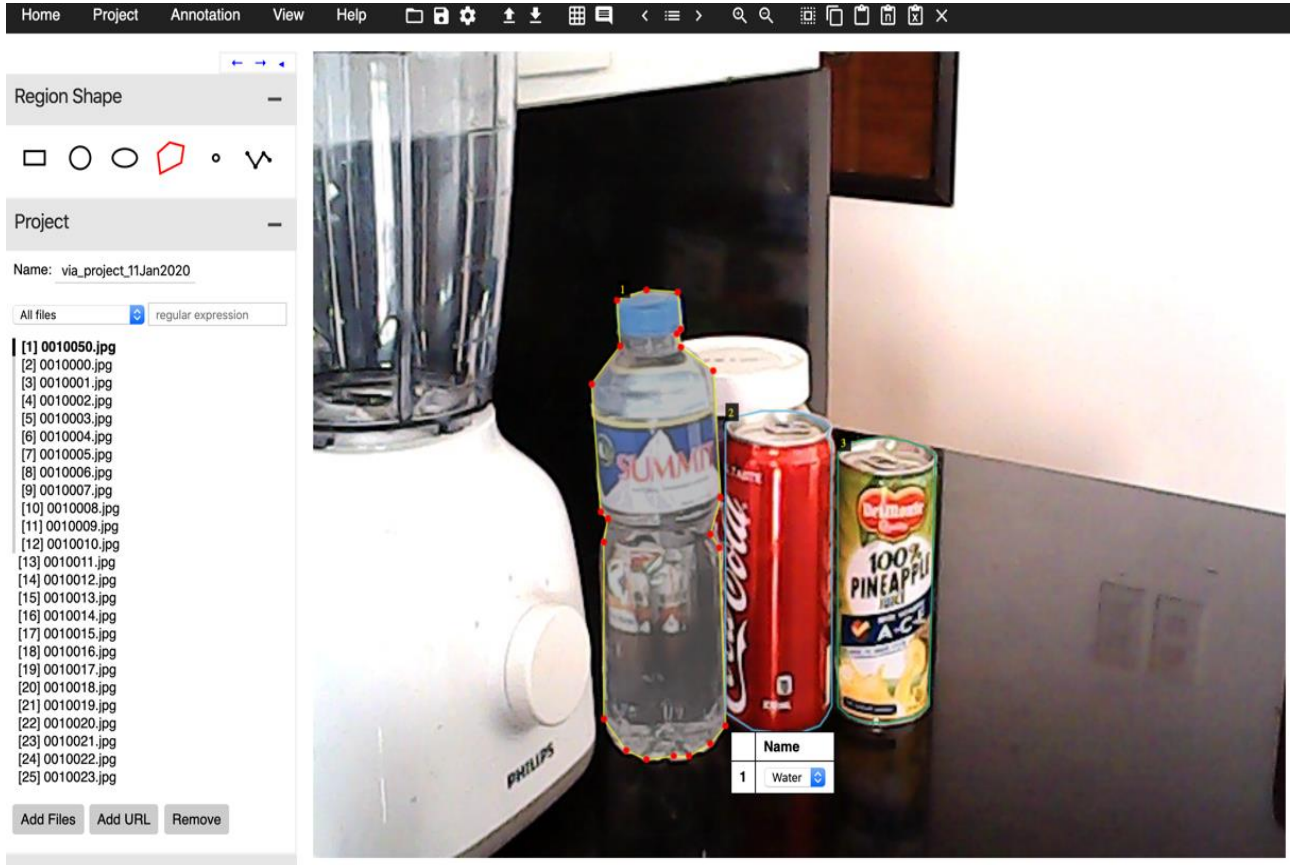
Features Pyramid in tf.keras

```
def features_pyramid(x, n_layers):
    outputs = [x]
    conv = AveragePooling2D(pool_size=2, name='pool1')(x)
    outputs.append(conv)
    prev_conv = conv
    n_filters = 512

    # additional feature map layers
    for i in range(n_layers - 1):
        postfix = "_layer" + str(i+2)
        conv = conv_layer(prev_conv,
                          n_filters,
                          kernel_size=3,
                          strides=2,
                          use_maxpool=False,
                          postfix=postfix)
        outputs.append(conv)
        prev_conv = conv

    return outputs
```

```
def build_fcn(input_shape,
              backbone,
              n_classes=4):
    inputs = Input(shape=input_shape)
    features = backbone(inputs)

    main_feature = features[0]
    features = features[1:]
    out_features = [main_feature]
    feature_size = 8
    size = 2
    # other half of the features pyramid
    # including upsampling to restore the
    # feature maps to the dimensions
    # equal to 1/4 the image size
    for feature in features:
        postfix = "fcn_" + str(feature_size)
        feature = conv_layer(feature,
                             filters=256,
                             use_maxpool=False,
                             postfix=postfix)
        postfix = postfix + "_up2d"
        feature = UpSampling2D(size=size,
                               interpolation='bilinear',
                               name=postfix)(feature)
        size = size * 2
        feature_size = feature_size * 2
        out_features.append(feature)

    # concatenate all upsampled features
    x = Concatenate()(out_features)
    # perform 2 additional feature extraction
    # and upsampling
    x = tconv_layer(x, 256, postfix="up_x2")
    x = tconv_layer(x, 256, postfix="up_x4")
    # generate the pixel-wise classifier
    x = Conv2DTranspose(filters=n_classes,
                        kernel_size=1,
                        strides=1,
                        padding='same',
                        kernel_initializer='he_normal',
                        name="pre_activation")(x)
    x = Softmax(name="segmentation")(x)

    model = Model(inputs, x, name="fcn")

    return model
```

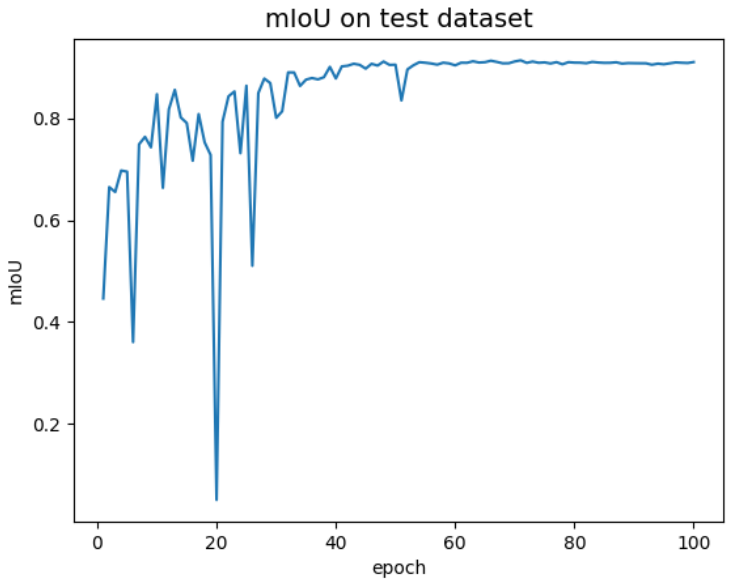
FCN + PSPNet

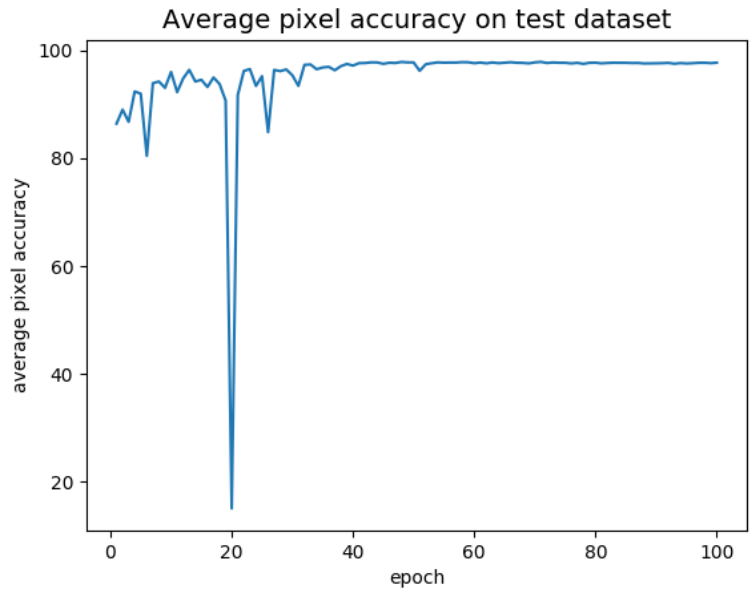
Dataset: 1k train, 50 test, VIA Annotation



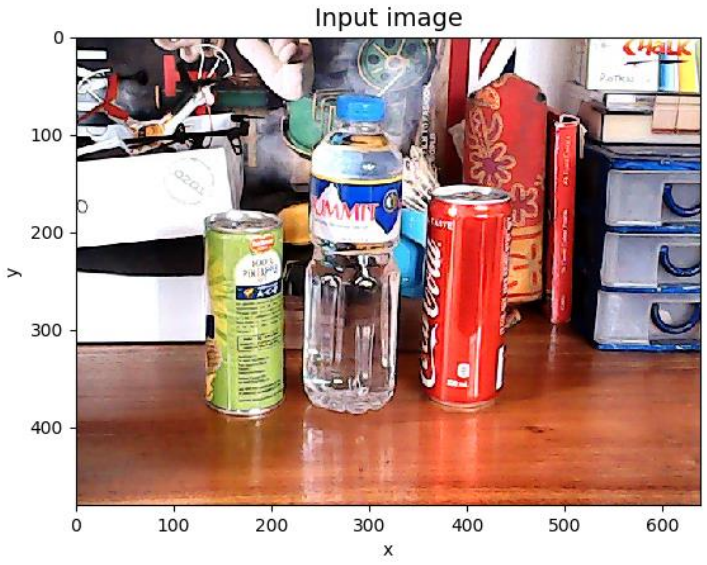
FCN+PSPNet Evaluation

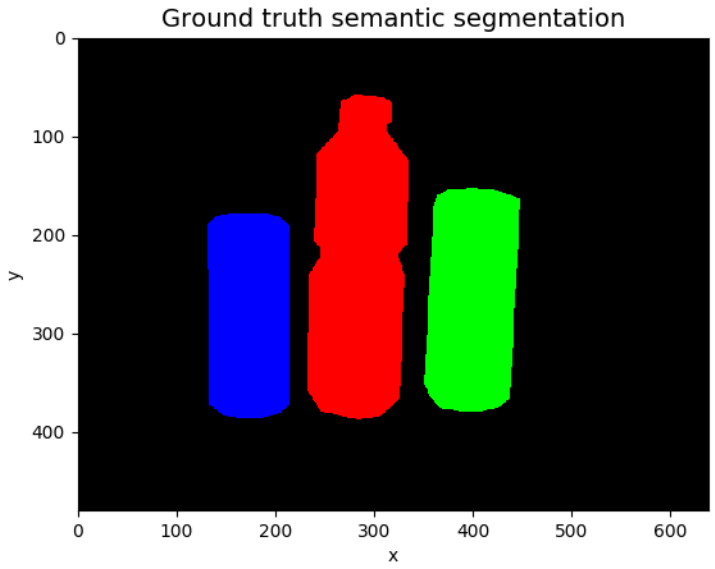
- Max mIoU = 0.91
- Max Pixel Accuracy = 97.9%

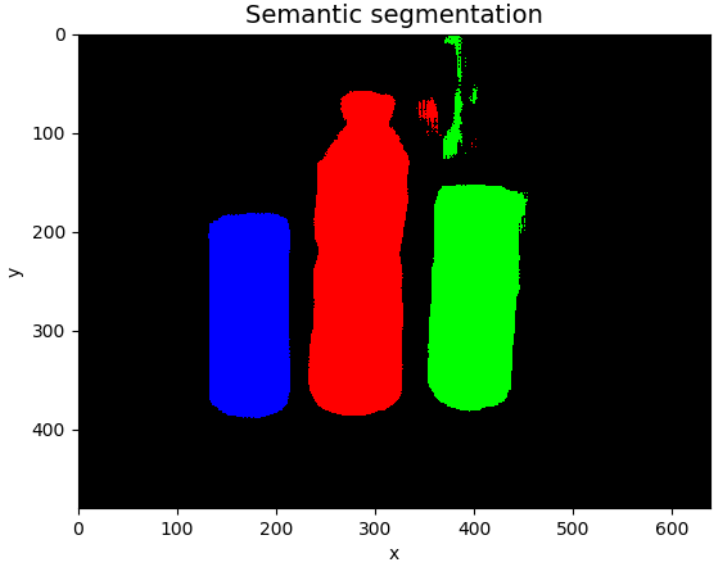




Sample Output







Instance Segmentation

Mask R-CNN

He, Kaiming, et al. "Mask r-cnn." Proceedings of the IEEE international conference on computer vision. 2017.

Instance Segmentation (Mask R-CNN)

- Simultaneous classification, localization and pixel-level classification
- ROI Pool is not interested in pixel-level alignment
  - ROI Pool is a coarse feature extraction method
    - ROI Pool performs coarse spatial quantization
  - To address this problem, Mask RC-CNN introduces RoIAlign
- Mask R-CNN – extends object detection by using FCN (semantic segmentation) on each ROI
  - ROIAlign is quantization free
  - Decouple mask and class predictions
  - FCN performs multi-class prediction per pixel

Instance Segmentation (Mask R-CNN)

- Simultaneous classification, localization and pixel-level classification
- Segmentation methods based on ROI or RPN performs segmentation first before recognition
  - Mask R-CNN performs mask prediction and recognition in parallel
  - Similar to FCIS [Li et al (2017)], segment proposal and object detection in parallel

Li, Yi, et al. "Fully convolutional instance-aware semantic segmentation." *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017.

Mask R-CNN

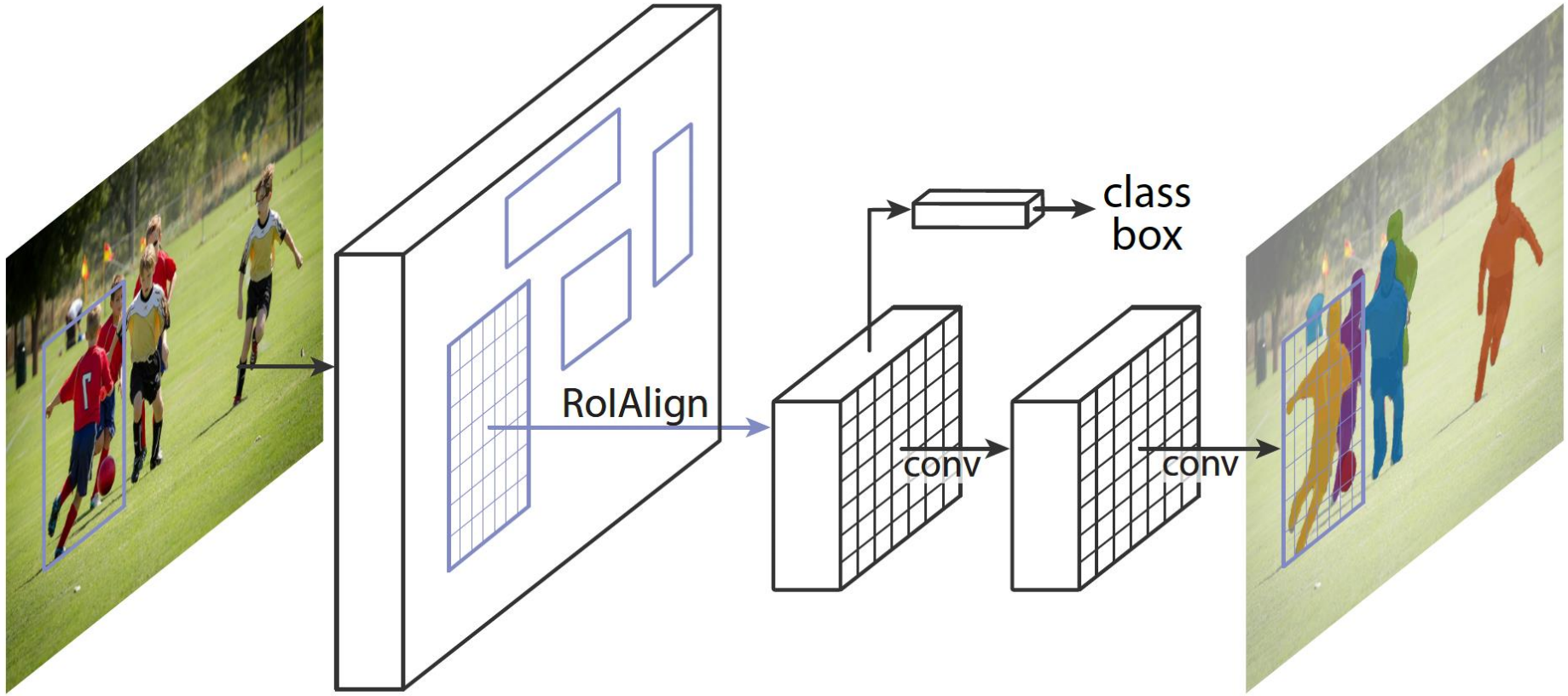


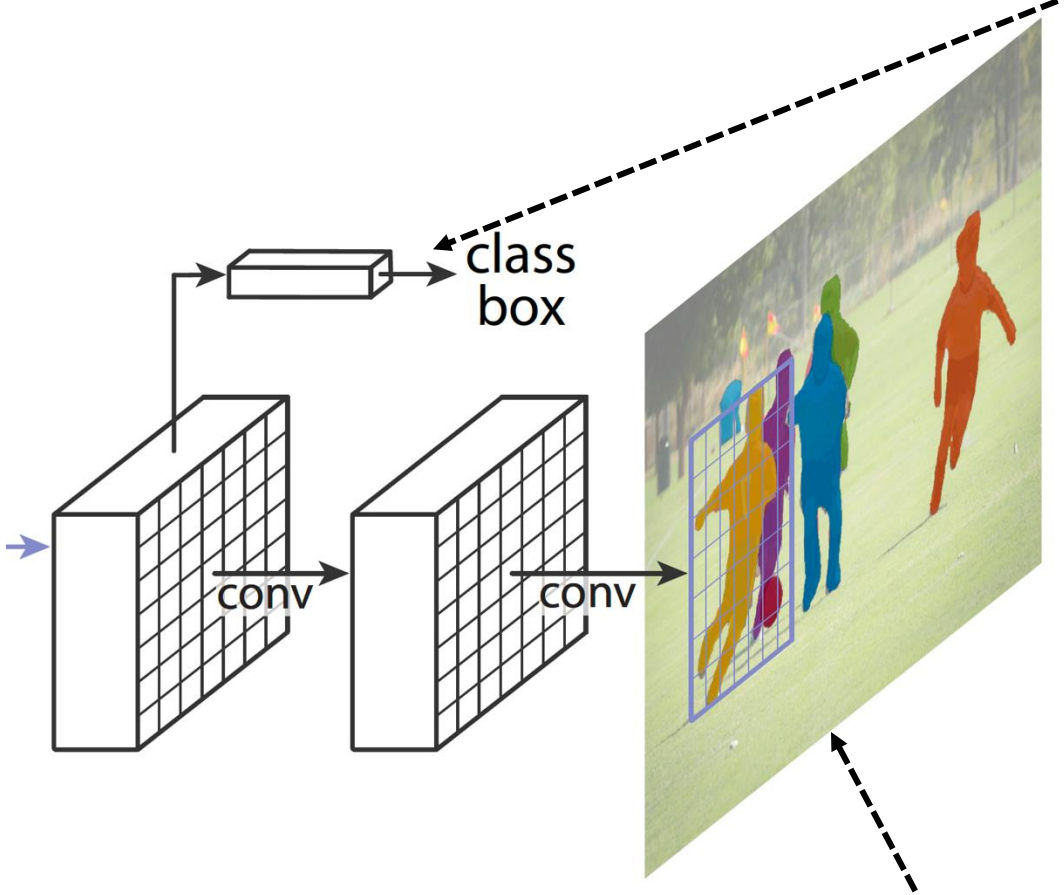
Figure 1. The **Mask R-CNN** framework for instance segmentation.

Loss Function

- $\mathcal{L} = \mathcal{L}_{cls} + \mathcal{L}_{off} + \mathcal{L}_{mask}$
- Mask loss is a binary cross entropy per pixel loss
  - Sigmoid output per pixel
- Every class has a mask prediction
  - Total binary mask predictions per ROI is $Km \times m$ where K is the number of categories
  - Each mask prediction is independent and does not compete with other mask predictions
  - The class label prediction selects which mask prediction contributes to $\mathcal{L}_{mask}$

Mask R-CNN

Class prediction per ROI selects which of the K mask predictions is valid and contributes to $\mathcal{L}_{mask}$. For example if the person class is the 5th then only the 5th mask prediction is selected.



Each ROI has a K-$m \times m$ mask
But only the k-th mask contribute to $\mathcal{L}_{mask}$

K mask predictions based on FCN

Mask

- Prediction: K $m \times m$ masks for each ROI using FCN
- Results to fewer parameters
- Predictions are in the form of maps thus preserving spatial dimensions

ROIAlign

- Instead of extracting 7×7 feature map per ROI, an ROI is divided into bins (eg 2×2) and a number (eg 4) of points are sampled per bin. The exact location of each feature point is used to generate the feature map.
- Key is unlike in ROI, no quantization is applied in mask ROI, bins and sampling points

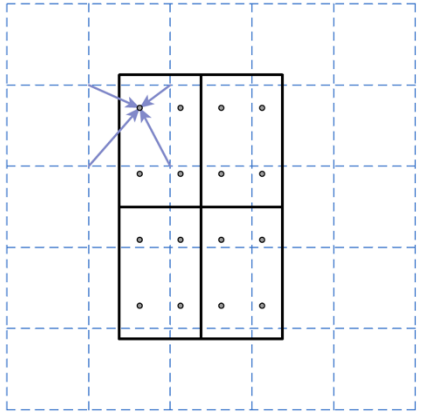


Figure 3. **RoIAlign:** The dashed grid represents a feature map, the solid lines an RoI (with 2×2 bins in this example), and the dots the 4 sampling points in each bin. RoIAlign computes the value of each sampling point by bilinear interpolation from the nearby grid points on the feature map. No quantization is performed on any coordinates involved in the RoI, its bins, or the sampling points.

Network Architecture

- Backbone network
  - ResNet50 and ResNeXt
- Extended network:
  - Faster R-CNN
  - Faster R-CNN with FPN

After prediction, the $m \times m$ mask is then resized to RoI dimensions

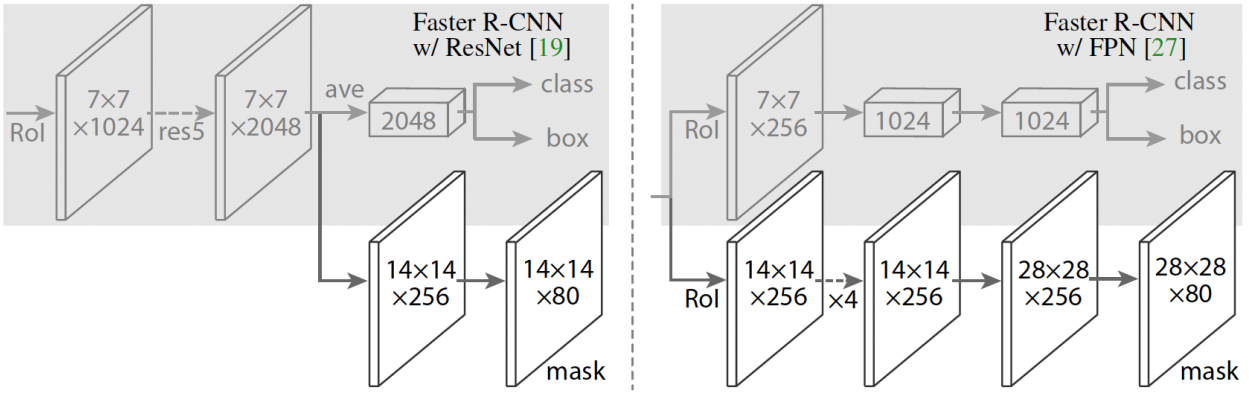


Figure 4. **Head Architecture**: We extend two existing Faster R-CNN heads [19, 27]. Left/Right panels show the heads for the ResNet C4 and FPN backbones, from [19] and [27], respectively, to which a mask branch is added. Numbers denote spatial resolution and channels. Arrows denote either conv, deconv, or *fc* layers as can be inferred from context (conv preserves spatial dimension while deconv increases it). All convs are 3×3, except the output conv which is 1×1, deconvs are 2×2 with stride 2, and we use ReLU [31] in hidden layers. *Left*: 'res5' denotes ResNet's fifth stage, which for simplicity we altered so that the first conv operates on a 7×7 RoI with stride 1 (instead of 14×14 / stride 2 as in [19]). *Right*: '×4' denotes a stack of four consecutive convs.

Training

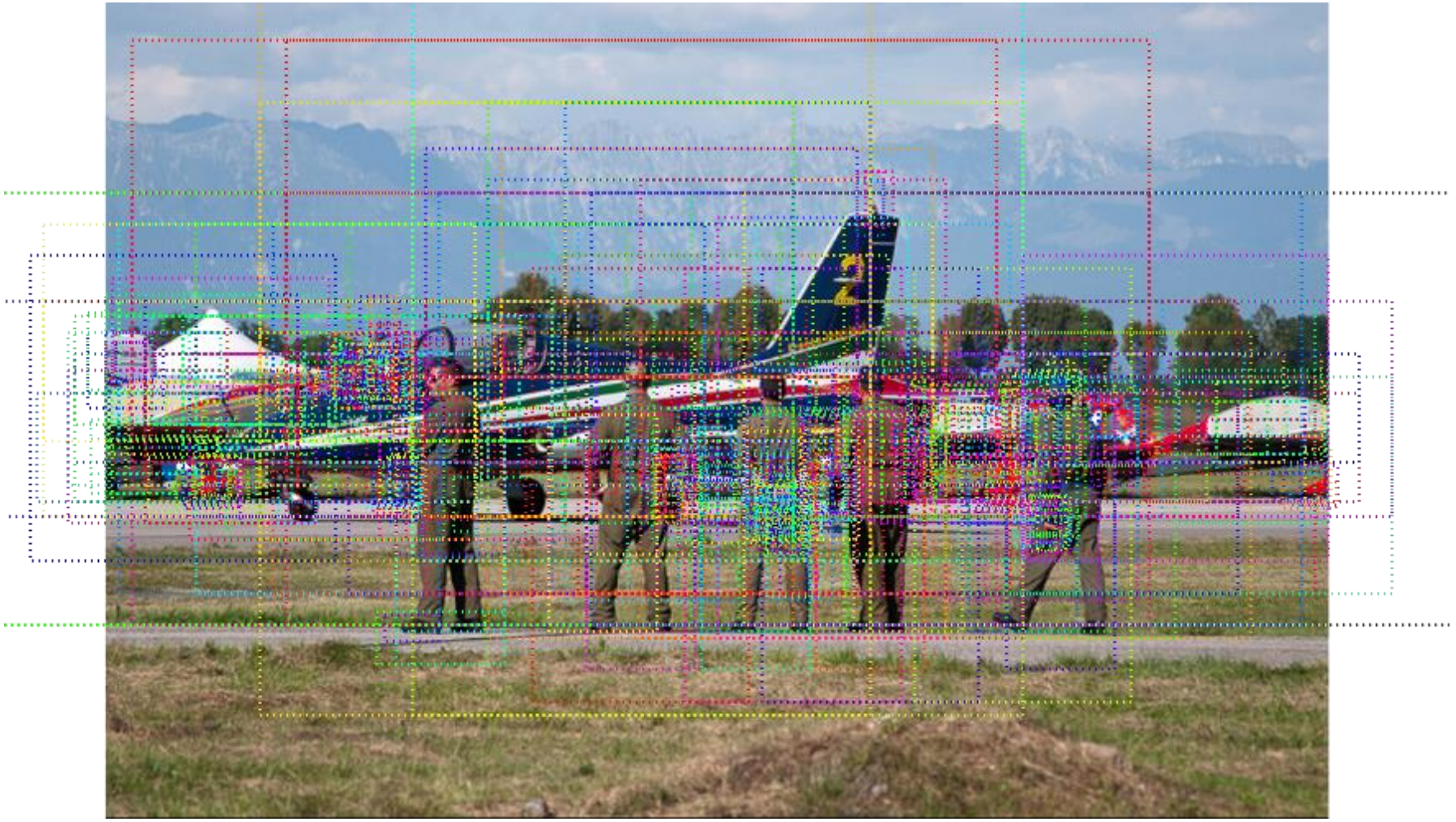
- $\mathcal{L}_{mask}$ is defined only on positive ROIs (IoU>0.5)
- 1:3 positive to negative ROI
- 5 scales, 3 aspect ratios

Inference

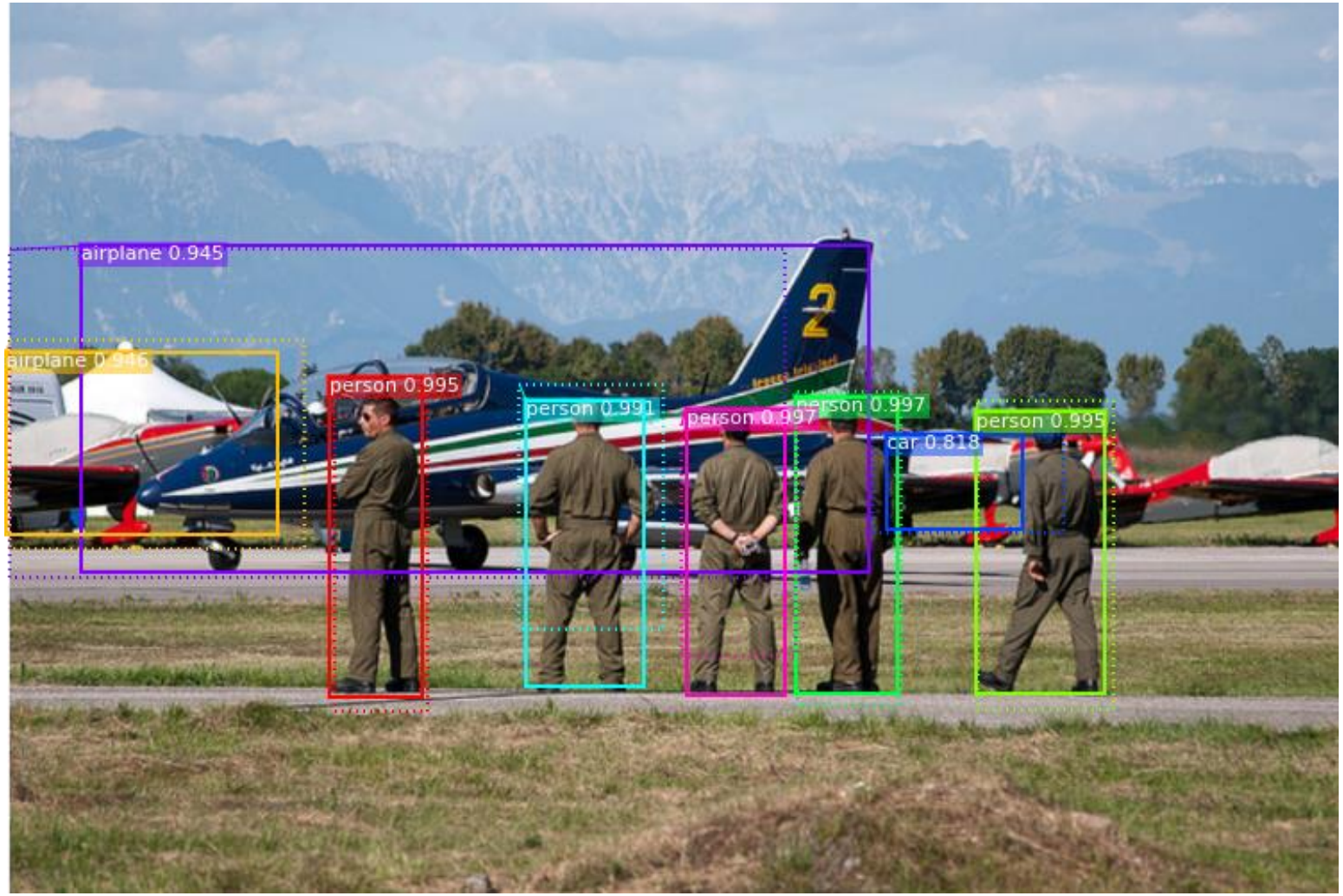
- Mask branch applied on top scoring 100 branches based on score
  - Each branch has class and bounding box prediction
- For each branch, only the k-th $m \times m$ mask corresponding to its class k is chosen
  - Recall that each of 100 predictions produces K $m \times m$ masks
- The chosen k-th mask is then resized by its RoI and binarized by 0.5 threshold

Mask RCNN in Summary

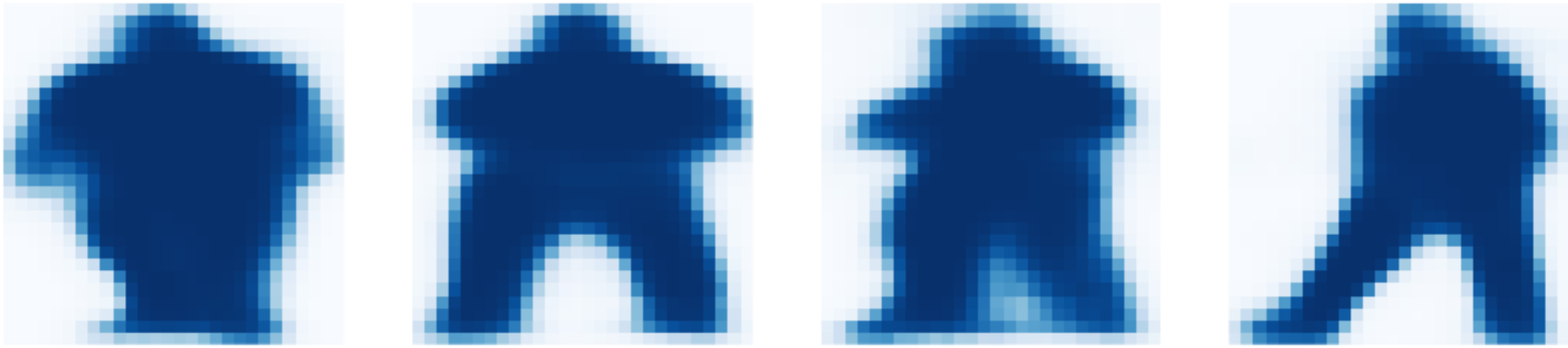
Region Proposal Network and displays positive and negative anchors



Bounding Box Refinement



Mask Generation



Composing the different pieces into a final result

Performance

- COCO test-dev
  - 37.1 AP ResNeXt-101-PFN
  - 35.7 AP ResNet-101-PFN
  - 33.1 AP ResNet-101-C4

	backbone	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
MNC [10]	ResNet-101-C4	24.6	44.3	24.8	4.7	25.9	43.6
FCIS [26] +OHEM	ResNet-101-C5-dilated	29.2	49.5	-	7.1	31.3	50.0
FCIS+++ [26] +OHEM	ResNet-101-C5-dilated	33.6	54.5	-	-	-	-
Mask R-CNN	ResNet-101-C4	33.1	54.9	34.8	12.1	35.6	51.1
Mask R-CNN	ResNet-101-FPN	35.7	58.0	37.8	15.5	38.1	52.4
Mask R-CNN	ResNeXt-101-FPN	**37.1**	**60.0**	**39.4**	**16.9**	**39.9**	**53.5**

Table 1. **Instance segmentation** *mask* AP on COCO `test-dev`. MNC [10] and FCIS [26] are the winners of the COCO 2015 and 2016 segmentation challenges, respectively. Without bells and whistles, Mask R-CNN outperforms the more complex FCIS+++, which includes multi-scale train/test, horizontal flip test, and OHEM [38]. All entries are *single-model* results.

Figure 5. More results of **Mask R-CNN** on COCO test images, using ResNet-101-FPN and running at 5 fps, with 35.7 mask AP (Table 1).

Mask Scoring R-CNN

Huang, Zhaojin, et al. "Mask scoring r-cnn." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2019.

Mask Scoring R-CNN

- By default, Mask R-CNN uses classifier score to score the mask prediction
  - However, mask quality as measured by IoU between mask prediction and ground truth is not correlated with classification
- Mask Scoring R-CNN
  - Learn a method for scoring mask quality
  - Mask scoring calibrates misalignment between mask score and mask quality
  - Improve instance segmentation by prioritizing high quality masks

Mask Scoring R-CNN

Score that is based on bounding box IoU as used by Mask R-CNN is not a good scoring method for a mask prediction.

Figure 1. Demonstrative cases of instance segmentation in which bounding box has a high overlap with ground truth and a high classification score while the mask is not good enough. The scores predicted by both Mask R-CNN and our proposed MS R-CNN are attached above their corresponding bounding boxes. The left four images show good detection results with high classification scores but low mask quality. Our method aims at solving this problem. The rightmost image shows the case of a good mask with a high classification score. Our method will retrain the high score. As can be seen, scores predicted by our model can better interpret the actual mask quality.

Mask Scoring R-CNN: Why not learn a score based on IoU of mask prediction and mask ground truth?

MaskIoU and Mask Score

- MaskIoU - pixel-level Intersection-over-Union (IoU) between the predicted mask and its ground truth mask to describe instance segmentation quality
- Mask score = MaskIoU x Class score , is aware of both semantic categories and the instance mask completeness

IoU and and MaskIoU

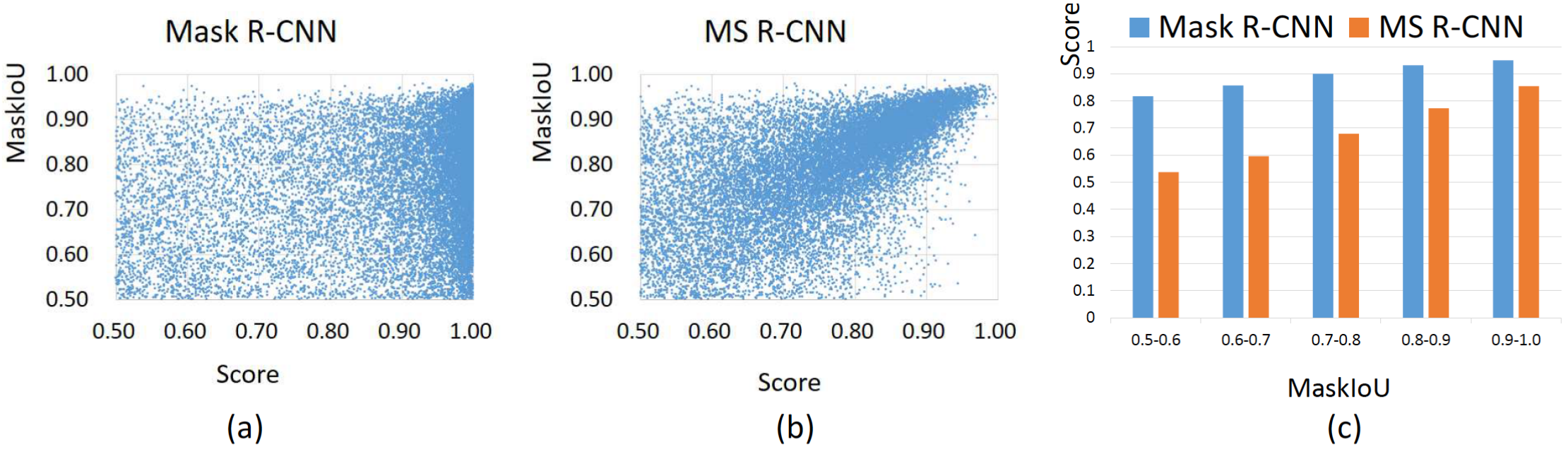


Figure 2. Comparisons of Mask R-CNN and our proposed MS R-CNN. (a) shows the results of Mask R-CNN, and the mask score has less relationship with MaskIoU. (b) shows the results of MS R-CNN; we penalize every detection with a high score and a low MaskIoU, and the mask score can correlate with MaskIoU better. (c) shows the quantitative results, where we average the score between each MaskIoU interval; we can see that our method can have a better correspondence between score and MaskIoU.

Classification score in Mask R-CNN is not correlated with MaskIoU

MS R-CNN

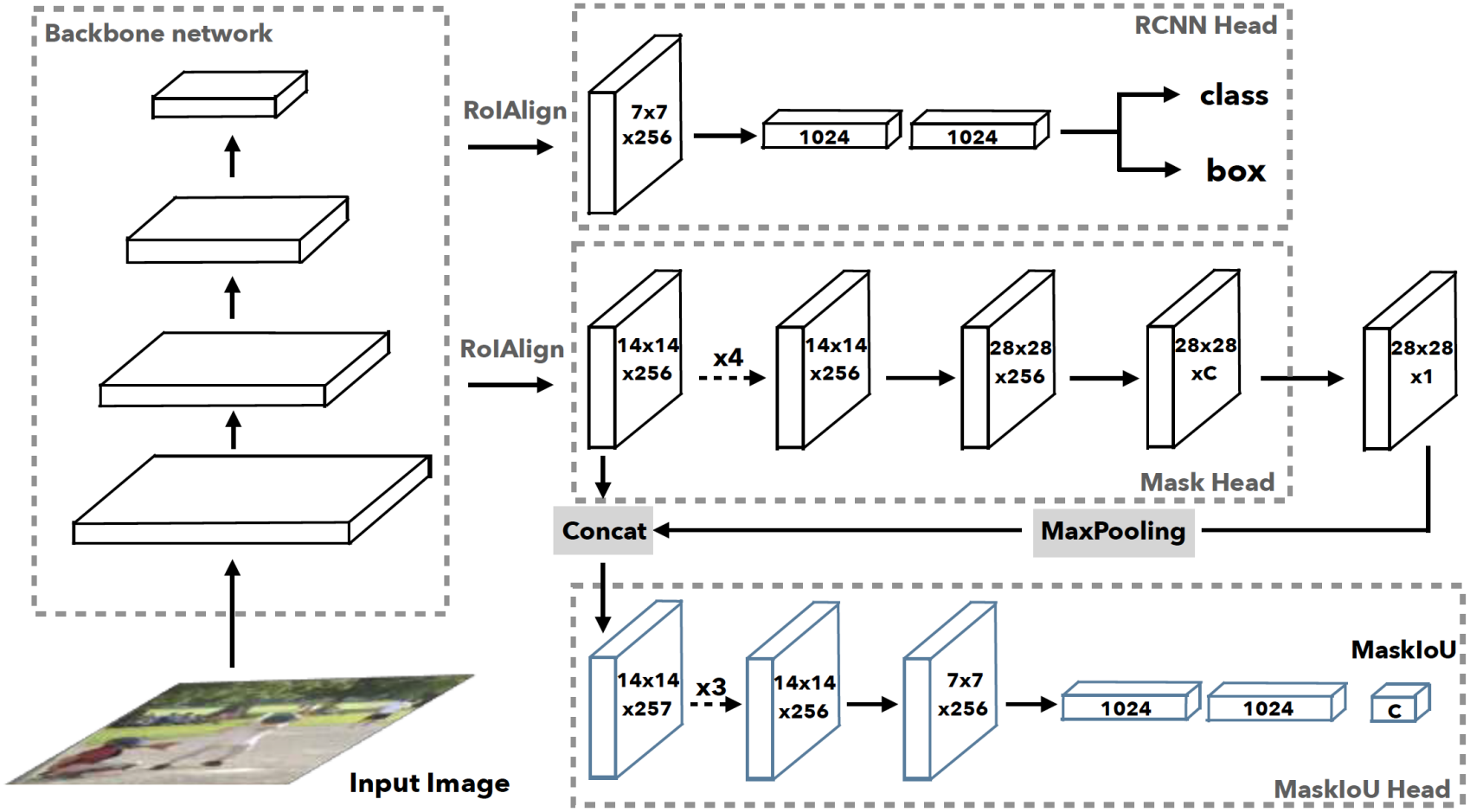
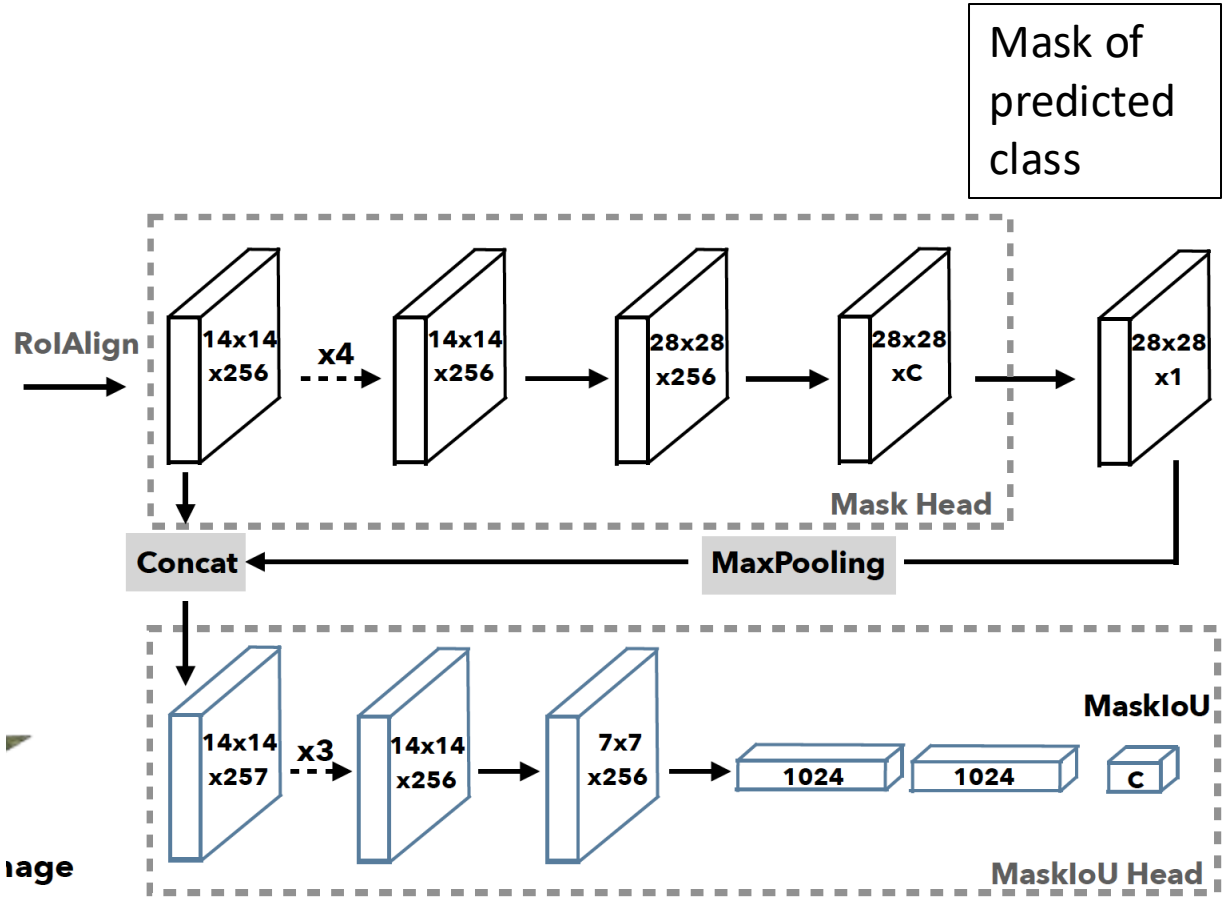


Figure 3. Network architecture of Mask Scoring R-CNN. The input image is fed into a backbone network to generate RoIs via RPN and RoI features via RoIAlign. The R-CNN head and Mask head are standard components of Mask R-CNN. For predicting MaskIoU, we use the predicted mask and RoI feature as input. The MaskIoU head has 4 convolution layers (all have kernel-size=3 and the final one uses stride=2 for downsampling) and 3 fully connected layers (the final one outputs C classes MaskIoU.)

MaskIoU and MaskIoU Head

- $s_{mask} = s_{cls} \cdot s_{iou}$
- s_{cls} focuses on classifying the proposal belong to which class
  - Output of classifier
- s_{iou} focuses on regressing the MaskIoU
  - Output of MaskIoU Head
- MaskIoU head regresses the IoU between ground truth and predicted masks



Ground truth is IoU between predicted mask and ground truth mask. Loss function is L2.

Inference

- Specifically, suppose the R-CNN stage of Mask R-CNN outputs N bounding boxes, and among them, top-k (i.e. k = 100) scoring boxes after SoftNMS [2] are selected.
- Then the top-k boxes are fed into the Mask head to generate multi-class masks.
  - This is the standard Mask R-CNN inference procedure.
  - We follow this procedure as well and feed the top-k target masks to predict the MaskIoU
- The predicted MaskIoU are multiplied with classification score, to get the new calibrated mask score as the final mask confidence

Performance

- COCO 2017 test
  - ResNet101: 35.4 AP MS R-CNN vs 34.3 AP Mask R-CNN
  - ResNet101 FPN: 38.3 AP MS R-CNN vs 37.0 AP Mask R-CNN

Table 3. Comparing different instance segmentation methods on COCO 2017 test-dev.

Method	Backbone	AP	AP@0.5	AP@0.75	AP_S	AP_M	AP_L
MNC [7]	ResNet-101	24.6	44.3	24.8	4.7	25.9	43.6
FCIS [23]	ResNet-101	29.2	49.5	-	-	-	-
FCIS+++ [23]	ResNet-101	33.6	54.5	-	-	-	-
Mask R-CNN [15]	ResNet-101	33.1	54.9	34.8	12.1	35.6	51.1
Mask R-CNN [15]	ResNet-101 FPN	35.7	58.0	37.8	15.5	38.1	52.4
Mask R-CNN [15]	ResNeXt-101 FPN	37.1	60.0	39.4	16.9	39.9	53.5
MaskLab [3]	ResNet-101	35.4	57.4	37.4	16.9	38.3	49.2
MaskLab+ [3]	ResNet-101	37.3	59.8	36.6	19.1	40.5	50.6
MaskLab+ [3]	ResNet-101 (JET)	38.1	61.1	40.4	19.6	41.6	51.4
Mask R-CNN	ResNet-101	34.3	55.0	36.6	13.2	36.4	52.2
MS R-CNN		35.4	54.9	38.1	13.7	37.6	53.3
Mask R-CNN	ResNet-101 FPN	37.0	59.2	39.5	17.1	39.3	52.9
MS R-CNN		38.3	58.8	41.5	17.8	40.4	54.4
Mask R-CNN	ResNet-101 DCN+FPN	38.4	61.2	41.2	18.0	40.5	55.2
MS R-CNN		39.6	60.7	43.1	18.8	41.5	56.2

YOLACT

Bolya, Daniel, et al. "YOLACT: Real-time Instance Segmentation." arXiv preprint arXiv:1904.02689 (2019).

YOLACT: Real-time instance segmentation

- Instance segmentation based on 2-stage detectors are slow
  - Mask R-CNN is based on Faster R-CNN
  - Processing is serial (stage 1 (ROI/RPN) then 2 (Fast RCNN)) by nature
- Idea – can we get rid of RPN?
  - Classification and detection over the entire image (what)
  - Prototype masks over the entire image (where)

YOLACT

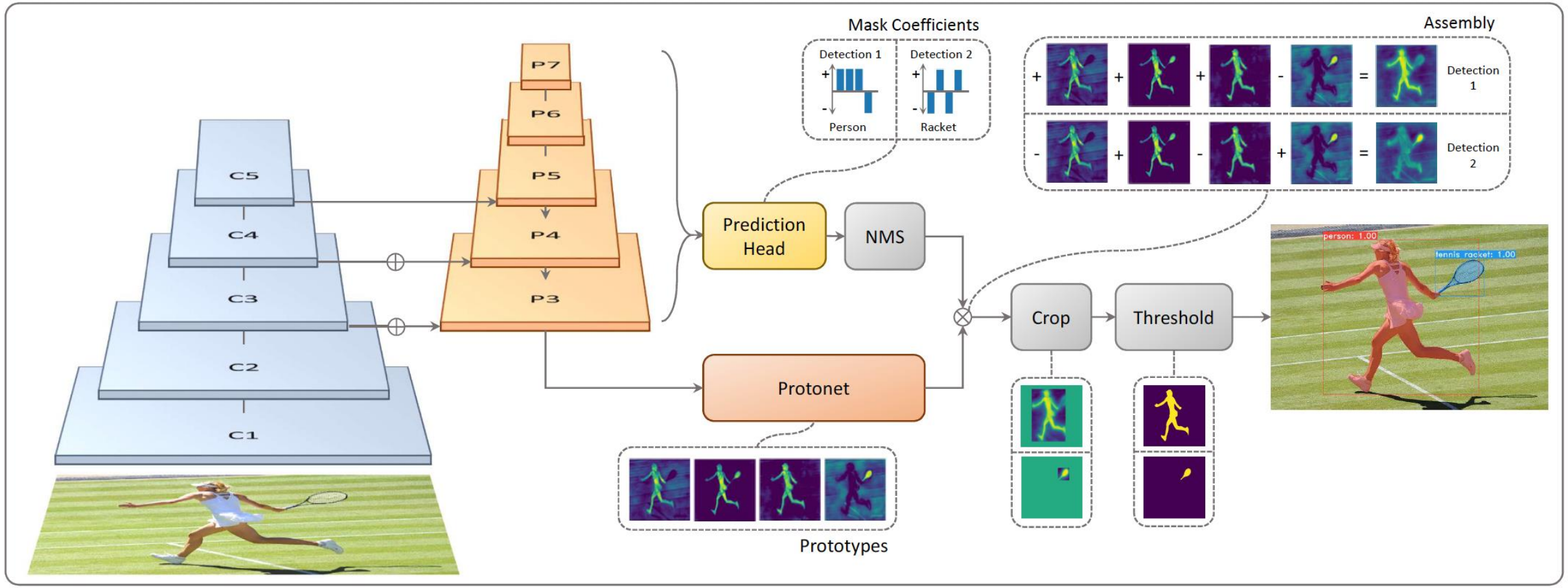
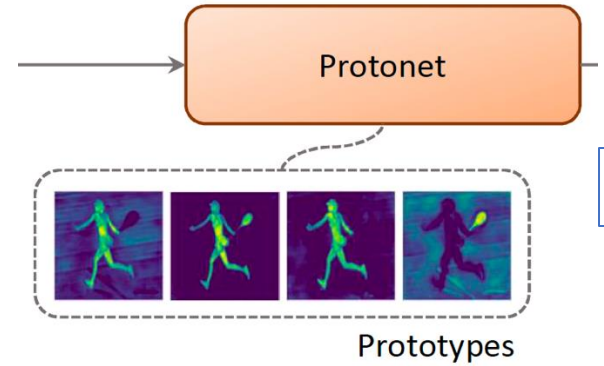


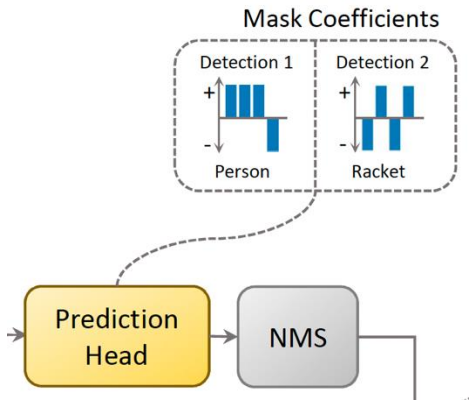
Figure 2: **YOLACT Architecture** Blue/yellow indicates low/high values in the prototypes, gray nodes indicate functions that are not trained, and $k = 4$ in this example. We base this architecture off of RetinaNet [25] using ResNet-101 + FPN.

Key Ideas

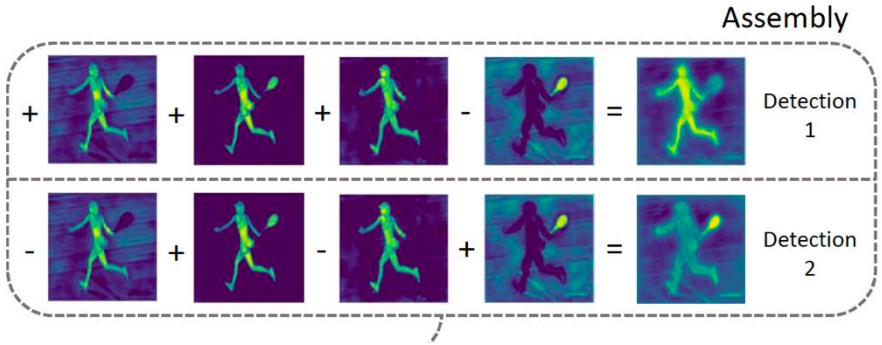
- Break instance segmentation into 2 parallel tasks
  - Produce image-size prototype masks
  - Add 2nd head in object detection to predict vector of mask coefficients for each anchor
- Combine the results of the 2 parallel tasks



prototype masks



mask coefficients



Combine after NMS

ProtoNet

- k prototypes outputs
- No explicit loss
- Supervision after mask assembly
- Final output is ¼ of input image size

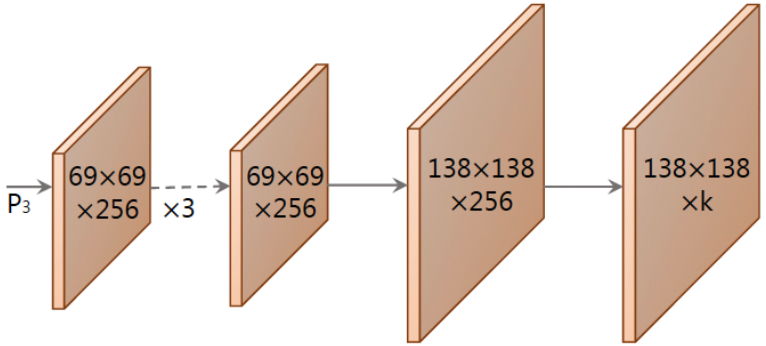


Figure 3: **Protonet Architecture** The labels denote feature size and channels for an image size of 550×550. Arrows indicate 3×3 *conv* layers, except for the final *conv* which is 1×1. The increase in size is an upsample followed by a *conv*. Inspired by the mask branch in [16].

Mask Coefficients

- Typical detection outputs per anchor:
  - c-dim class
  - 4-dim bounding box offsets
- YOLACT add 3rd output per anchor called masking coefficients
  - k coefficients (1 per prototype)
  - Use of tanh activation
- New output dimensions per anchor: $c + 4 + k$

Mask Assembly

- To produce instance masks, combine the outputs of prototype mask and mask coefficients

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

Where P is $h \times w \times k$ prototype masks, C is $n \times k$ mask coefficients with n is the number of surviving masks after NMS

Loss Functions

- $\mathcal{L} = \mathcal{L}_{cls} + \mathcal{L}_{off} + \mathcal{L}_{mask}$
- Mask loss is a binary cross entropy per pixel loss
  - Sigmoid output per pixel
- $\mathcal{L}_{mask} = BCE(M, M_{gt})$
- $\mathcal{L}_{mask}$ is divided by the area of ground truth mask M_{gt}
- $\mathcal{L}_{off}$ is smooth L1
- $\mathcal{L}_{cls}$ is categorical cross entropy
- OHEM sample mining with ratio of 1:3 (positive:negative)

OHEM (Online Hard Example Mining)

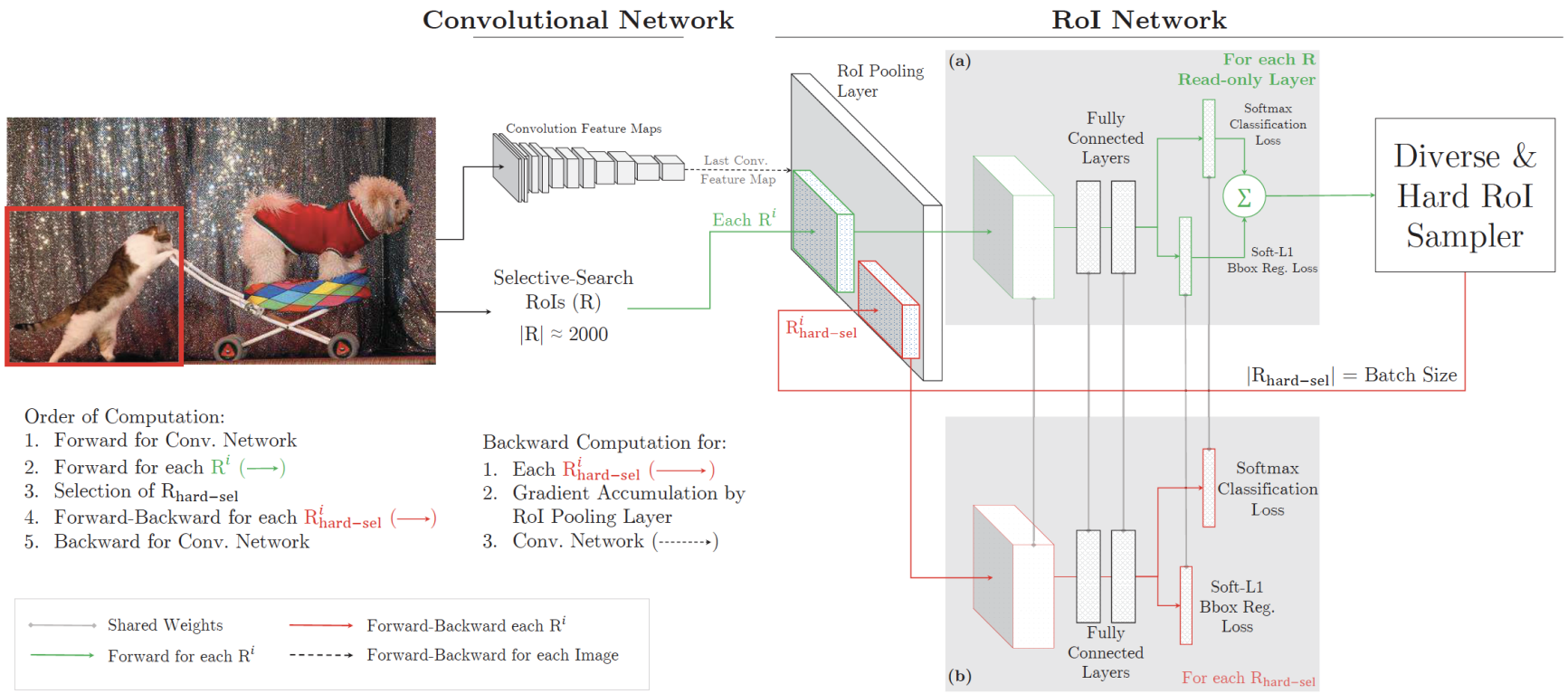


Figure 2: Architecture of the proposed training algorithm. Given an image, and selective search RoIs, the conv network computes a conv feature map. In (a), the *readonly* RoI network runs a forward pass on the feature map and all RoIs (shown in green arrows). Then the Hard RoI module uses these RoI losses to select B examples. In (b), these hard examples are used by the RoI network to compute forward and backward passes (shown in red arrows).

Shrivastava, Abhinav, Abhinav Gupta, and Ross Girshick. "Training region-based object detectors with online hard example mining." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

Emergent Behavior in Prototype Masks

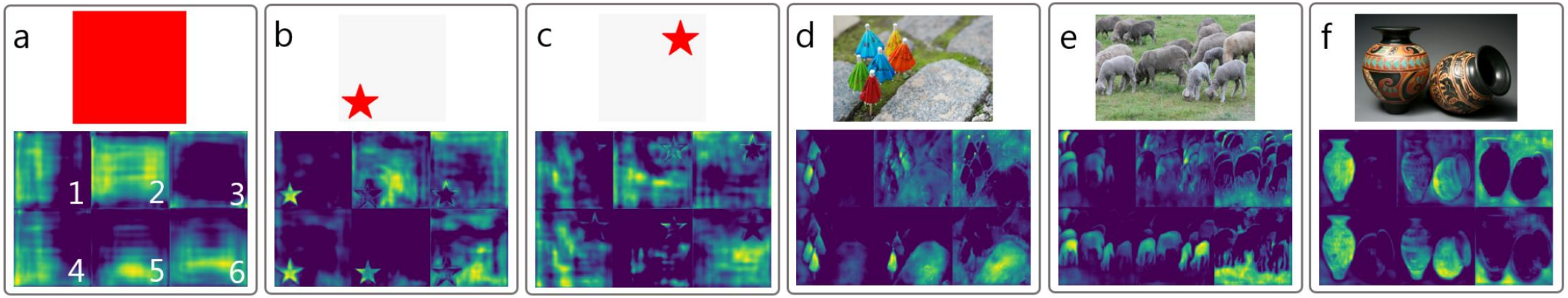


Figure 5: **Prototype Behavior** The activations of the same six prototypes across different images. Prototypes 1, 4, and 5 are partition maps with boundaries clearly defined in image a, prototype 2 is a bottom-left directional map, prototype 3 segments out the background and provides instance contours,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

Emergent Behavior

- Prototypes activate on certain partitions of the image
  - 1, 4, 5 masks on boundaries
  - 2 is directional map
  - 3 background and contours
  - 6 mask on ground
- Combination of prototype masks results into instance level masking
  - Mask 4 – Mask 5 removes the red umbrella

YOLACT Head

- P_3 to P_7 PFN with ResNet-101 head
- Output of each P_i goes to 3 detectors per anchor a:
  - c-dim class
  - 4-dim bounding box offset
  - k-dim mask coefficient
- Anchor sizes: $[24, 48, 96, 192, 384]$
- Aspect ratios: $[1, 2, \frac{1}{2}]$

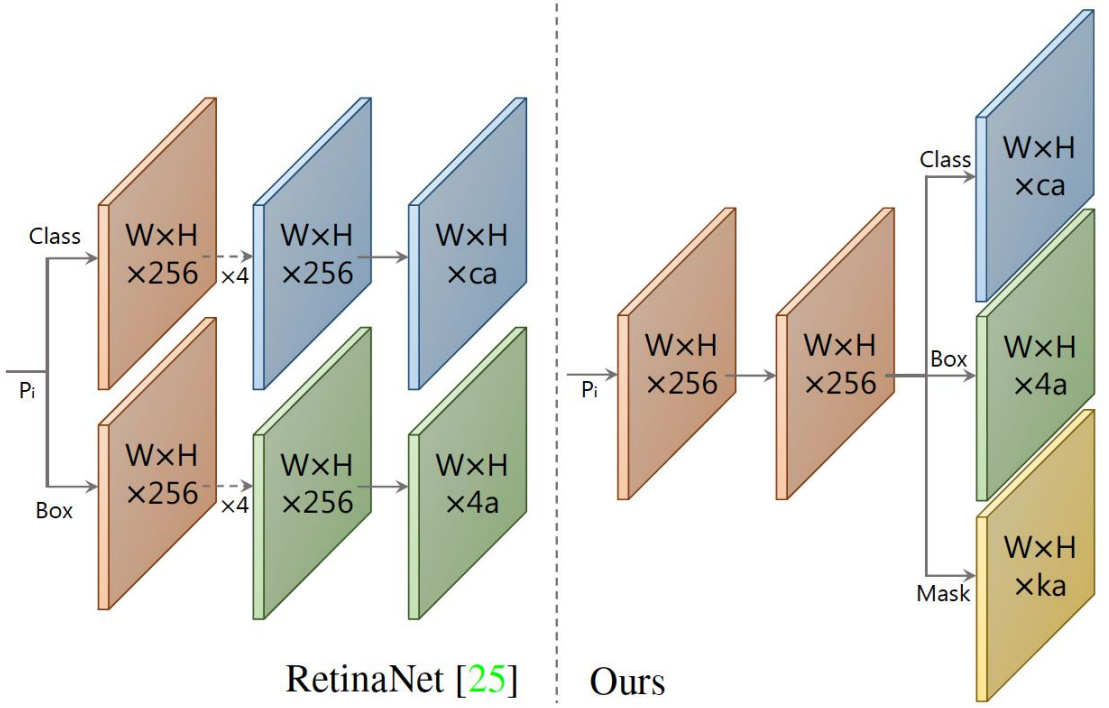


Figure 4: **Head Architecture** We use a shallower prediction head than RetinaNet [25] and add a mask coefficient branch. This is for c classes, a anchors for feature layer P_i, and k prototypes. See Figure 3 for a key.

Fast NMS

- NMS is a for loop
  - Can we do it in parallel?
- Fast NMS computes a sorted matrix of IoU
  - $X \in \mathbb{R}^{c \times n \times n}$, where c is the number of classes, n (top n) detections
  - X is sorted in descending order by score for c classes
  - Threshold IoU by t
- Fast NMS:
  1. Setup all IoUs: $X_{kij} = 0 \quad \forall k, j, i \geq j$
     1. Pytorch `torch.triu()`
  2. Max IoU per detection: $K_{kj} = \max_i \left(X_{kij} \right) . \quad \forall k, j$
  3. Thresholding: $\mathrm{K} = K_{kj} \geq t$

Performance

- 3.8x faster than the fastest instance segmentation (Mask R-CNN and Mask Scoring R-CNN)
  - 33fps at 550^2 image resolution
  - 29.8 AP vs 38.3 and 35.7 AP of Mask Scoring and Mask R-CNN
  - COCO test-dev

Figure 6: **YOLACT** evaluation results on COCO's `test-dev` set. This base model achieves 29.8 mAP at 33.0 fps. All images have the confidence threshold set to 0.3.

Detection

Cascade R-CNN

Cai, Zhaowei, and Nuno Vasconcelos. "Cascade r-cnn: Delving into high quality object detection." Proceedings of the IEEE conference on computer vision and pattern recognition. 2018.

Cascade R-CNN

- Detection networks are dependent on IoU threshold
  - $IoU \geq 0.5$ for positive anchor boxes
  - $IoU \geq 0.5$ introduces close false positives in which humans can easily reject
- Low IoU threshold results into noisy detection (many false positives)
- High IoU threshold degrades detection performance
  - High IoU is prone to overfitting resulting to low detection rates
- Cascade R-CNN
  - Can we learn IoU thresholds?
  - Sequence of detectors with increasing IoU thresholds
  - Lower-level detector has a better idea on higher-level detector IoU
    - Low-level detectors are resampling object proposals

Cascade R-CNN

- Localization is directly affected by input IoU
- Detection is directly affected by IoU threshold
- Quality of hypothesis is directly affected by the quality of what it processes and evaluates
  - However, it is not simply a matter of increasing the IoU as shown in the figures.
  - Gray line in c) shows that the detection stage has a better idea on the ground truth that can be used for the next stage; Most of the time all lines are above the gray line

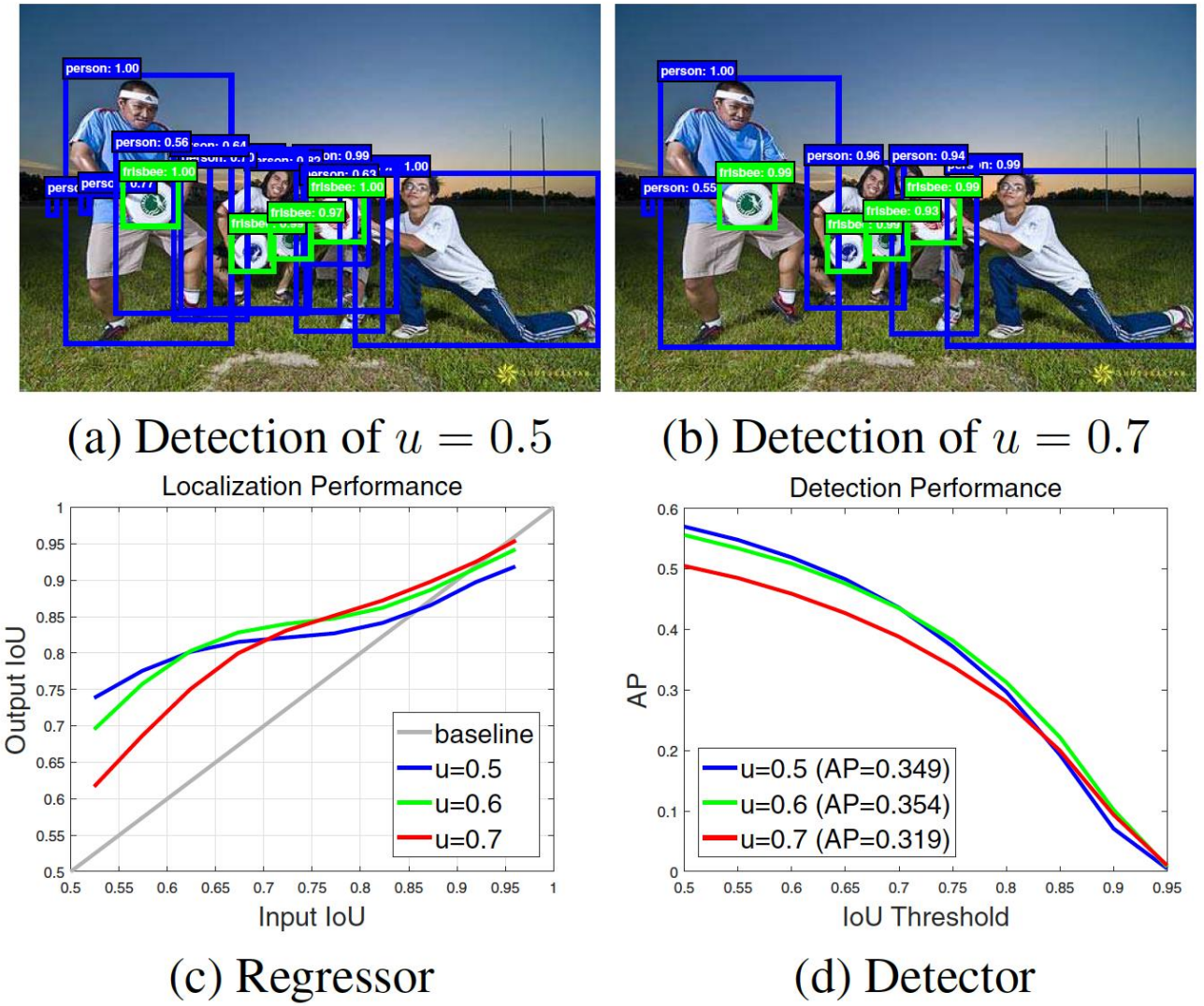
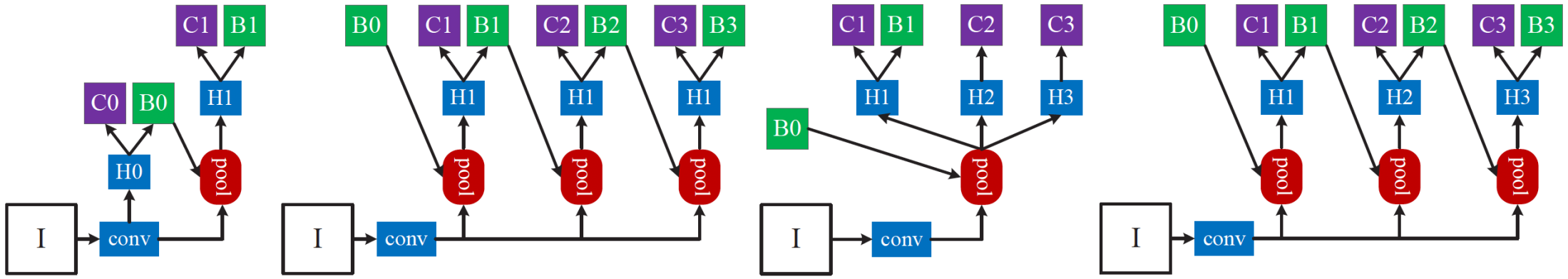


Figure 1. The detection outputs, localization and detection performance of object detectors of increasing IoU threshold u.

Cascade R-CNN

- In Faster R-CNN, H0 is RPN, H1 is detection head, C1 is classifier and B1 is bounding box offset predictor



(a) Faster R-CNN (b) Iterative BBox at inference (c) Integral Loss (d) Cascade R-CNN

Figure 3. The architectures of different frameworks. "I" is input image, "conv" backbone convolutions, "pool" region-wise feature extraction, "H" network head, "B" bounding box, and "C" classification. "B0" is proposals in all architectures.

Detection Problem (Review)

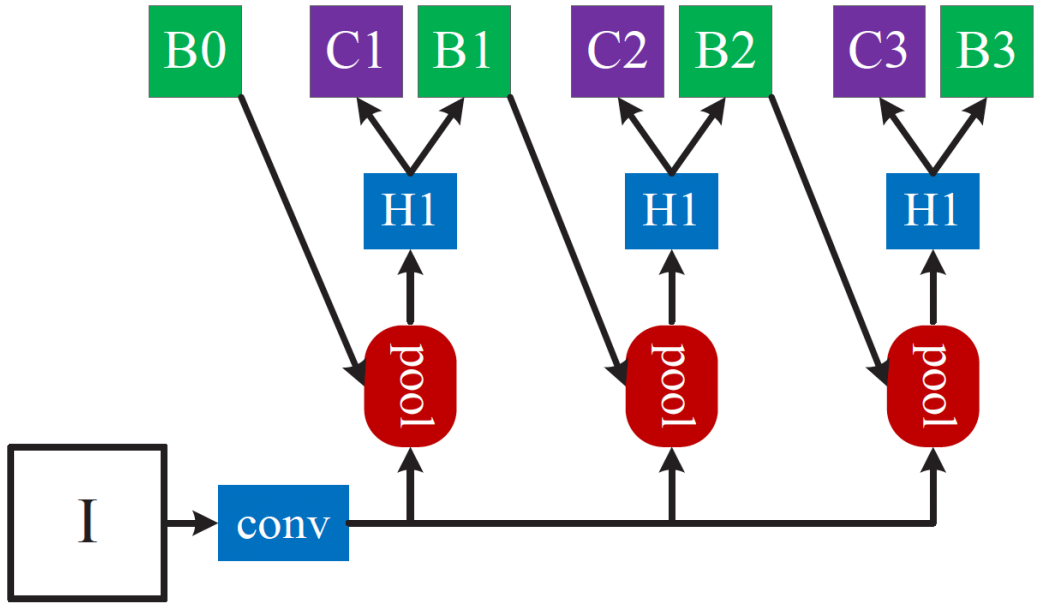
- Localization
  - $\mathbf{b} = (b_x, b_y, b_w, b_h)$ where (b_x, b_y) is the bounding box centroid and (b_w, b_h) is the width and height.
  - Given ground truth $\mathbf{g} = (g_x, g_y, g_w, g_h)$, the object detection network regresses $f(x, \mathbf{b})$ by minimizing a loss function $\mathcal{L}_{off}$ (eg smooth L1) using supervised learning from dataset $(\mathbf{g}_i, \mathbf{b}_i)$. $\mathcal{L}_{off}$ operates on Δ.
  - $\Delta = \left(\frac{\frac{g_x - b_x}{b_w}}{\sigma_x}, \frac{\frac{g_y - b_y}{b_h}}{\sigma_y}, \frac{\log\frac{g_w}{b_w}}{\sigma_w}, \frac{\log\frac{g_h}{b_h}}{\sigma_h} \right) = (\delta_x, \delta_y, \delta_w, \delta_h)$

Iterative BBox

- Multi-step localization:

$$f'(x, \mathbf{b}) = f \circ f \ldots \circ f(x, \mathbf{b})$$

- Same head, H1
- Problem is H1 is trained on IoU threshold of $u = 0.5$
  - Not optimal for higher threshold on upper-level detector



(b) Iterative BBox at inference

Iterative Box

- Distribution changes after the 1st detector
- As IoU threshold is increased in the upper-level detector the number of outliers increases (red dots)

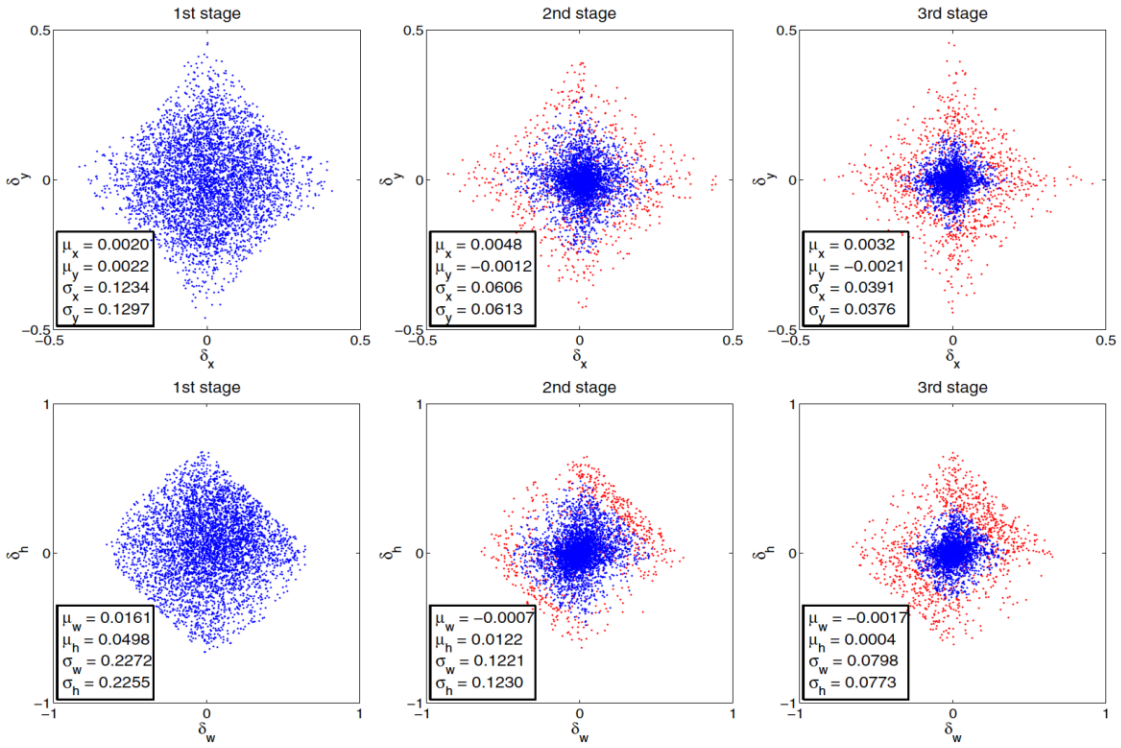


Figure 2. Sequential Δ distribution (without normalization) at different cascade stage. Red dots are outliers when using increasing IoU thresholds, and the statistics are obtained after outlier removal.

Classification

- Classification
  - The class b_c of bounding box $\mathbf{b}$ is matched to ground truth class g_c by minimizing a loss function $\mathcal{L}_{cls}$ (eg softmax CE) using supervised learning by the IoU threshold criterion:

$$\mathrm{b}_c = \begin{cases} \mathrm{g}_c & if\ IoU(g,b) > u \\ 0 & else \end{cases}$$

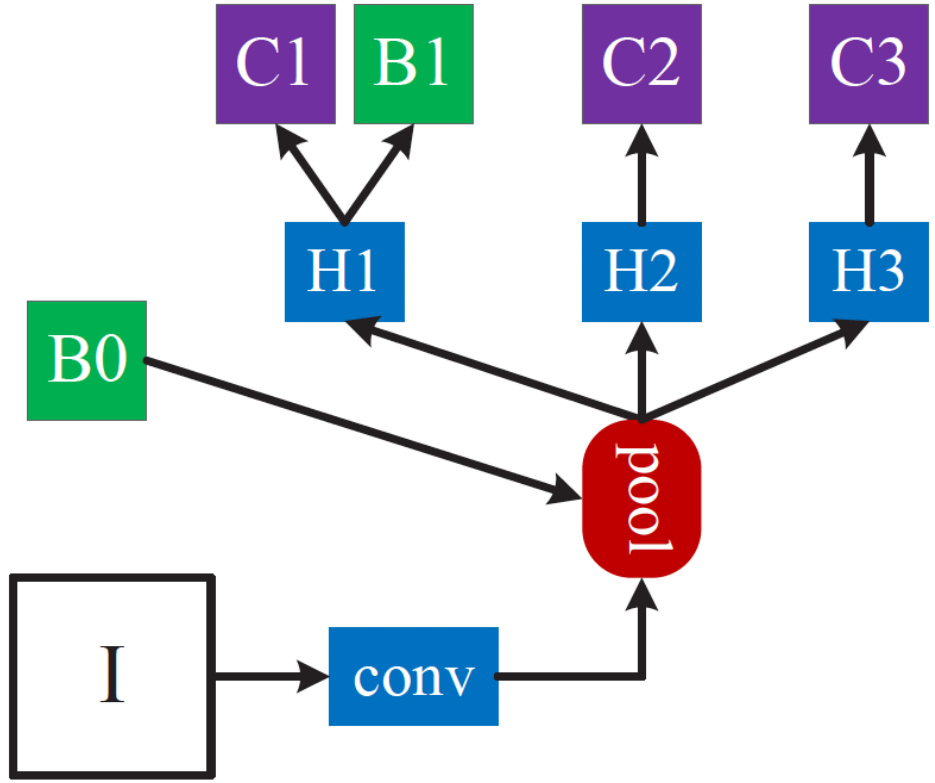
  - where 0 is the background class
  - g_c is the class label of the ground truth object g
- When u is high, few positive samples for training
- When u is low, no incentive to reject false positives
- If we settle at $u = 0.5$, many close false positives as classified by humans are detected

Integral Loss

- The idea is to use different thresholds in evaluating the loss function:

$$\mathcal{L}'_{cls} = \sum_{u \in U} \mathcal{L}_{cls}$$

- Where $U = \{0.5, 0.55, \dots, 0.75\}$ as designed by the COCO Challenge
  - Also known as Integral Loss



(c) Integral Loss

Integral Loss

- In 1st stage, the number of positive samples decreases rapidly with u
- Many low-quality outputs, few high-quality outputs
  - Overfits on low-quality outputs
- Problem is high-quality classifiers are forced to evaluate outputs of low-quality classifiers

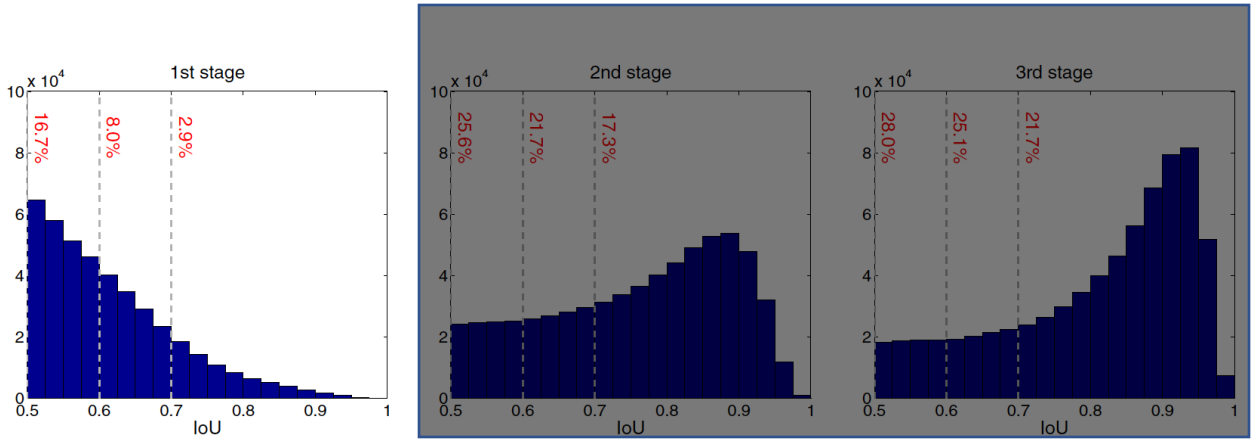


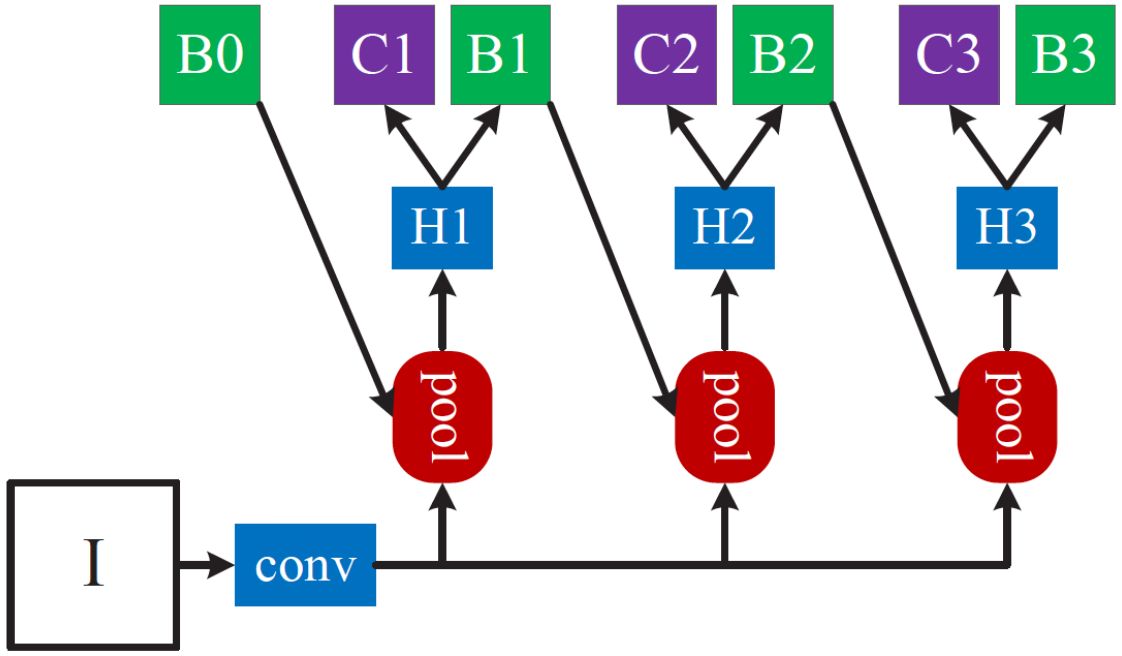
Figure 4. The IoU histogram of training samples. The distribution at 1st stage is the output of RPN. The red numbers are the positive percentage higher than the corresponding IoU threshold.

Cascade R-CNN

- Specialized regressor:

$$f(x, \mathbf{b}) = f_T \circ f_{T-1} \ldots \circ f_1(x, \mathbf{b})$$

  - Where T is the number of cascade stages
- Each f_t is specialized in optimizing its own $\{\mathbf{b}^t\}$
  - Unlike iterative BBox which reuses its optimizer for different bounding boxes
- Note that the regressor f_t optimizes wrt to $\{\mathbf{b}^t\}$ instead of the initial bounding boxes $\{\mathbf{b}^1\}$



(d) Cascade R-CNN

Cascade R-CNN

- Due to resampling, the number of positive samples at higher IoU threshold increases
  - Addressing imbalance in Integral Loss
  - More high-quality positive samples
  - No overfitting since there is no imbalance
-

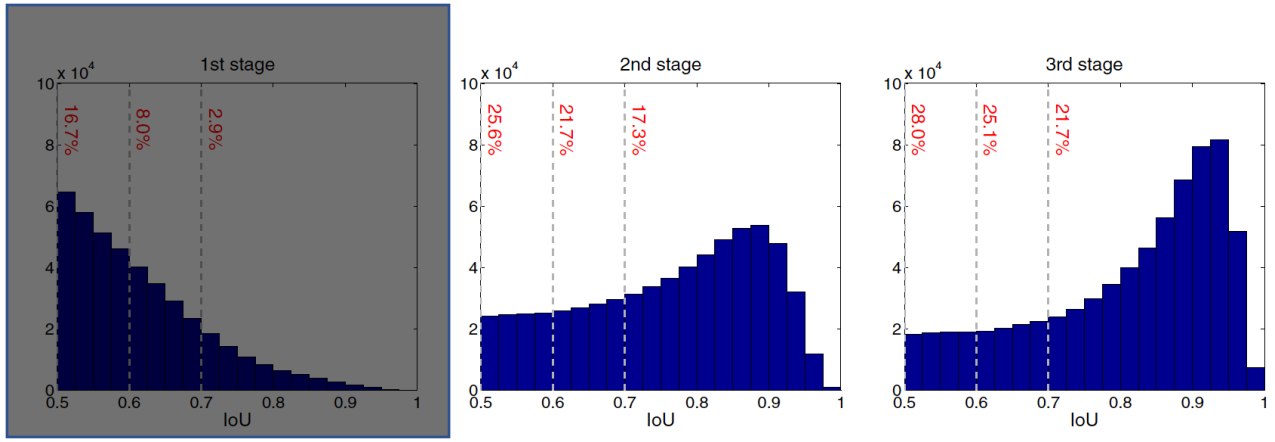


Figure 4. The IoU histogram of training samples. The distribution at 1st stage is the output of RPN. The red numbers are the positive percentage higher than the corresponding IoU threshold.

Loss Function

- $\mathcal{L} = \mathcal{L}_{cls}^{t} + \lambda[c \geq 1]\mathcal{L}_{off}^{t}$
- Where t is the stage number (eg. 3)
- $\mathcal{L}_{cls}^{t}$ is the classifier loss at stage t
- $\mathcal{L}_{off}^{t}$ is the bounding box offset loss at stage t
- The threshold increases with stage: $u^{t} > u^{t-1}$
- The bounding box at stage at stage t is $\mathbf{b}^{t} = f_{t-1}(x^{t-1},\ \mathbf{b}^{t-1})$ is a function of previous inputs and bounding boxes .

Performance

- COCO 2017 test-dev (ResNet 101) : 42.8 AP
  - 41.5AP FCOS
  - 42.8 AP FSAF

HTC

Chen, Kai, et al. "Hybrid task cascade for instance segmentation." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2019.

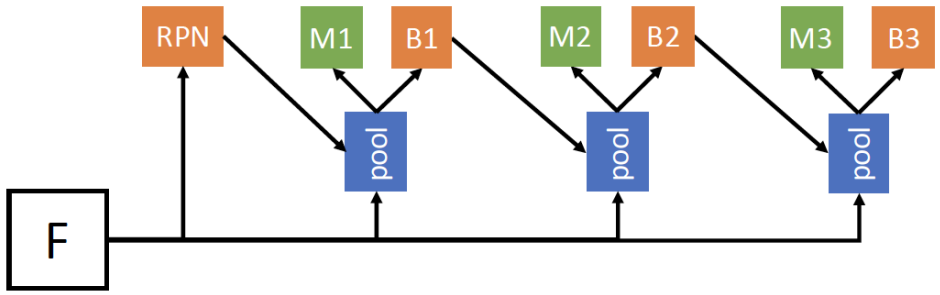
Cascade

- 2 key ideas of Cascade R-CNN:
  - Progressive refinement of prediction
  - Adaptive handling of training distributions
  - Applying Cascade R-CNN on Mask R-CNN has limited gains
- HTC Solution: Improve information flow
  - Interweave detection and segmentation features together for a joint multi-stage processing

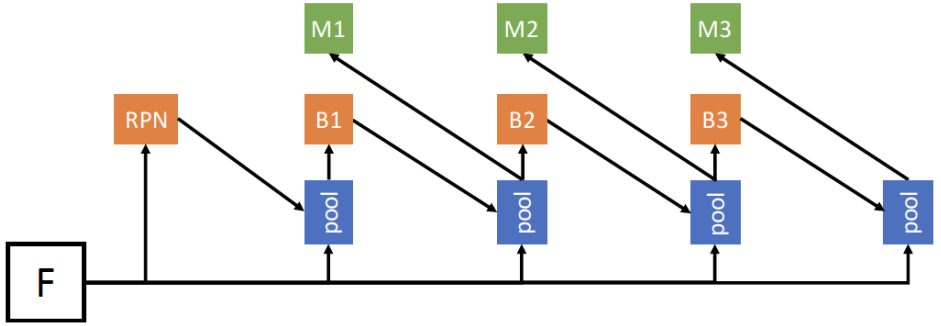
Hybrid Task Cascade (HTC) Key Ideas

- Interleave bounding box regression and mask prediction instead of executing them in parallel
- Incorporate a direct path to reinforce the information flow between mask branches by feeding the mask features of the preceding stage to the current one
- Explore more contextual information by adding an additional semantic segmentation branch and fusing it with box and mask branches

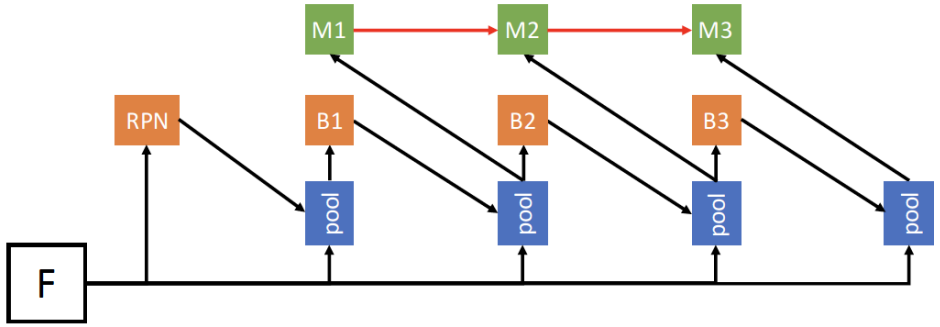
Cascade Architecture



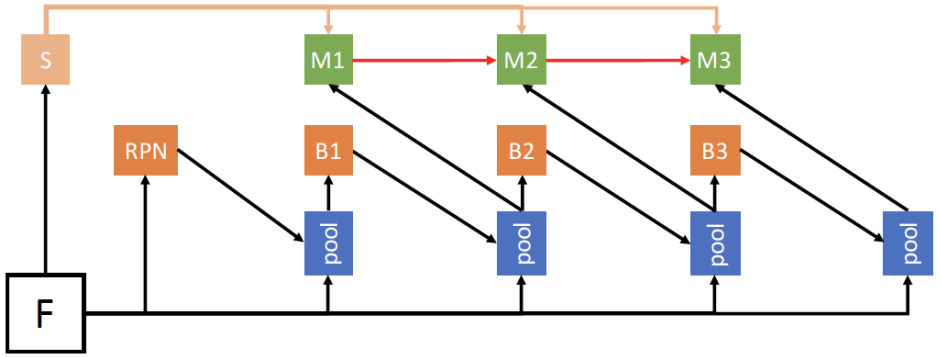
(a) Cascade Mask R-CNN



(b) Interleaved execution



(c) Mask information flow

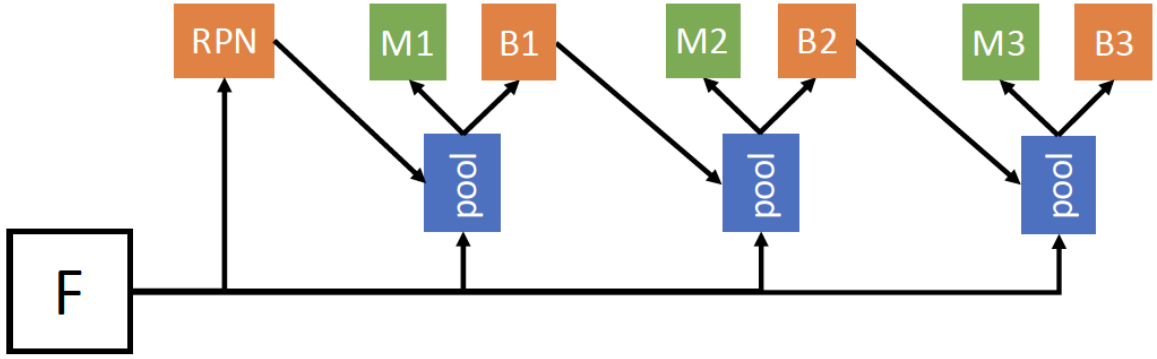


(d) Hybrid Task Cascade (semantic feature fusion with box branches is not shown on the figure for neat presentation.)

Figure 1: The architecture evolution from Cascade Mask R-CNN to Hybrid Task Cascade.

Cascade Mask R-CNN

- $\mathbf{x}_t^{box} = \mathcal{P}(\mathbf{x}, \mathbf{r}_{t-1})$
- $\mathbf{x}_t^{mask} = \mathcal{P}(\mathbf{x}, \mathbf{r}_{t-1})$
- $\mathbf{r}_t = B_t(\mathbf{x}_t^{box})$
- $\mathbf{m}_t = M_t(\mathbf{x}_t^{mask})$
- $\mathbf{x}$ are backbone features
- $\mathbf{x}_t^{box}$ box features and $\mathbf{x}_t^{mask}$ mask features from $\mathbf{x}$ and ROIs
- $\mathcal{P}$ is pooling operation
- B_t and M_t are box and mask heads
- $\mathbf{r}_t$ and $\mathbf{m}_t$ are box and mask outputs

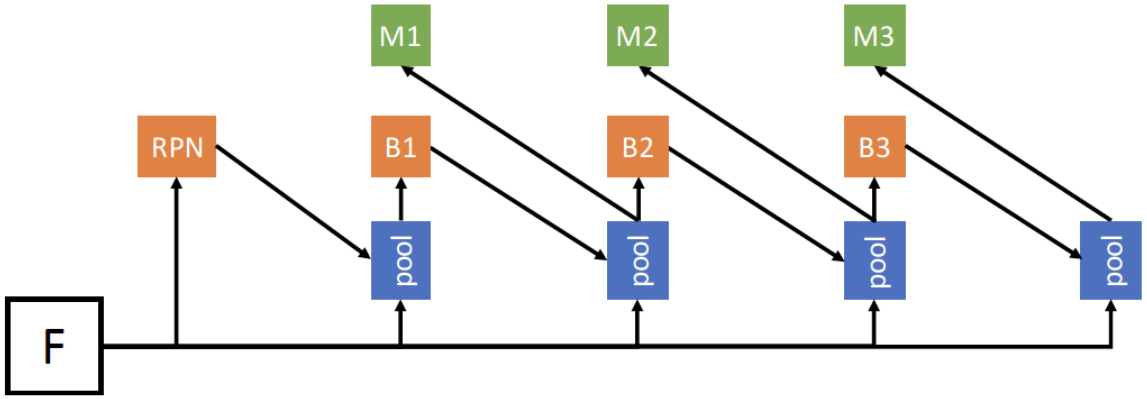


(a) Cascade Mask R-CNN

Small improvement over Mask R-CNN

Interleaved Execution

- $\mathbf{x}_t^{box} = \mathcal{P}(\mathbf{x}, \mathbf{r}_{t-1})$
- $\mathbf{x}_t^{mask} = \mathcal{P}(\mathbf{x}, \mathbf{r}_t)$
- $\mathbf{r}_t = B_t(\mathbf{x}_t^{box})$
- $\mathbf{m}_t = M_t(\mathbf{x}_t^{mask})$
- $\mathbf{x}$ are backbone features
- $\mathbf{x}_t^{box}$ box features and $\mathbf{x}_t^{mask}$ mask features from $\mathbf{x}$ and ROIs
- $\mathcal{P}$ is pooling operation
- B_t and M_t are box and mask heads
- $\mathbf{r}_t$ and $\mathbf{m}_t$ are box and mask outputs

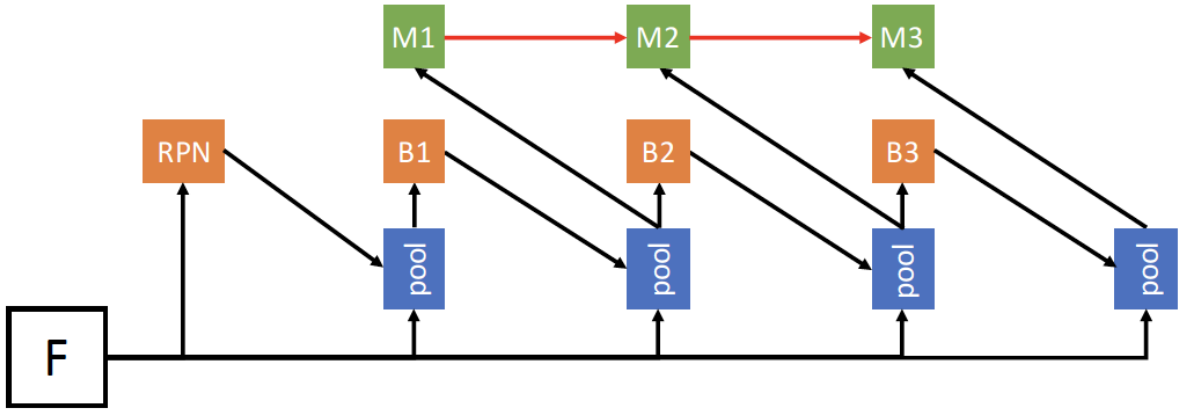


(b) Interleaved execution

Interleaved execution – improved performance over Cascade Mask R-CNN

Mask Information Flow

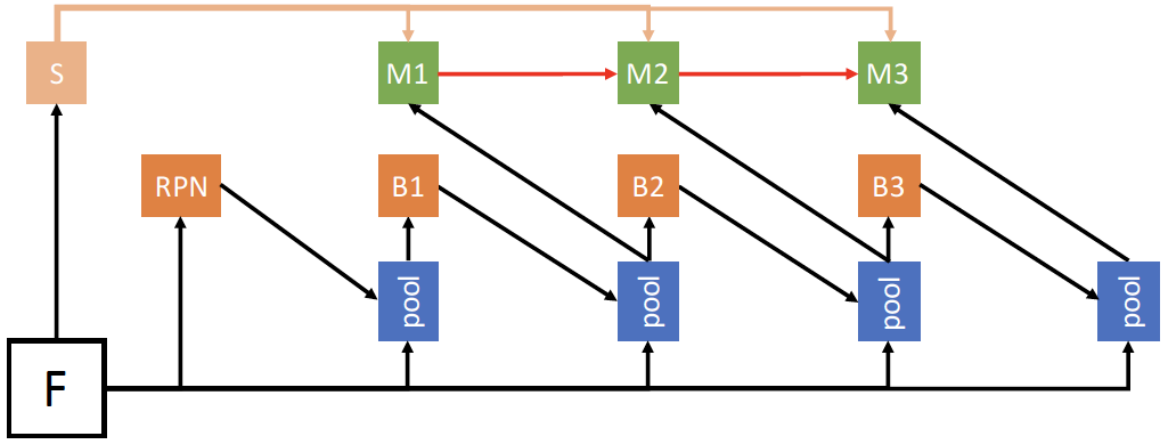
- $\mathbf{x}_t^{box} = \mathcal{P}(\mathbf{x}, \mathbf{r}_{t-1})$
- $\mathbf{x}_t^{mask} = \mathcal{P}(\mathbf{x}, \mathbf{r}_t)$
- $\mathbf{r}_t = B_t(\mathbf{x}_t^{box})$
- $\mathbf{m}_t = M_t(\mathbf{x}_t^{mask}, \mathbf{m}_{t-1}^-)$
- $\mathbf{x}$ are backbone features
- $\mathbf{x}_t^{box}$ box features and $\mathbf{x}_t^{mask}$ mask features from $\mathbf{x}$ and ROIs
- $\mathcal{P}$ is pooling operation
- B_t and M_t are box and mask heads
- $\mathbf{m}_{t-1}^-$ is the intermediate features of M_{t-1}
- $\mathbf{r}_t$ and $\mathbf{m}_t$ are box and mask outputs



(c) Mask information flow

HTC using Mask Information Flow & Segmentation Branch

- $\mathbf{x}_t^{box} = \mathcal{P}(\mathbf{x}, \mathbf{r}_{t-1}) + \mathcal{P}(\mathcal{S}(\mathbf{x}), \mathbf{r}_{t-1})$
- $\mathcal{S}$ is segmentation head
- $\mathbf{x}_t^{mask} = \mathcal{P}(\mathbf{x}, \mathbf{r}_t) + \mathcal{P}(\mathcal{S}(\mathbf{x}), \mathbf{r}_t)$
- $\mathbf{r}_t = B_t(\mathbf{x}_t^{box})$
- $\mathbf{m}_t = M_t\left(\mathcal{F}(\mathbf{x}_t^{mask}, \mathbf{m}_{t-1}^{-})\right)$
- $\mathbf{x}$ are backbone features
- $\mathbf{x}_t^{box}$ box features and $\mathbf{x}_t^{mask}$ mask features from $\mathbf{x}$ and ROIs
- $\mathcal{P}$ is pooling operation
- B_t and M_t are box and mask heads
- $\mathbf{m}_{t-1}^{-}$ is the intermediate features of M_{t-1}
- $\mathbf{r}_t$ and $\mathbf{m}_t$ are box and mask outputs



(d) Hybrid Task Cascade (semantic feature fusion with box branches is not shown on the figure for neat presentation.)

HTC – Mask Information Flow

- $\mathcal{F}\left(\mathbf{x}_t^{mask}, \mathbf{m}_{t-1}\right) = \mathbf{x}_t^{mask} + \mathcal{G}(\mathbf{m}_{t-1}^{-})$
- $\mathbf{m}_1^{-} = M_1^{-}\left(\mathbf{x}_t^{mask}\right)$
- $\mathbf{m}_2^{-} = M_2^{-}\left(\mathcal{F}\left(\mathbf{x}_t^{mask}, \mathbf{m}_1^{-}\right)\right)$
- …
- $\mathbf{m}_{t-1}^{-} = M_t^{-}\left(\mathcal{F}\left(\mathbf{x}_t^{mask}, \mathbf{m}_{t-1}^{-}\right)\right)$

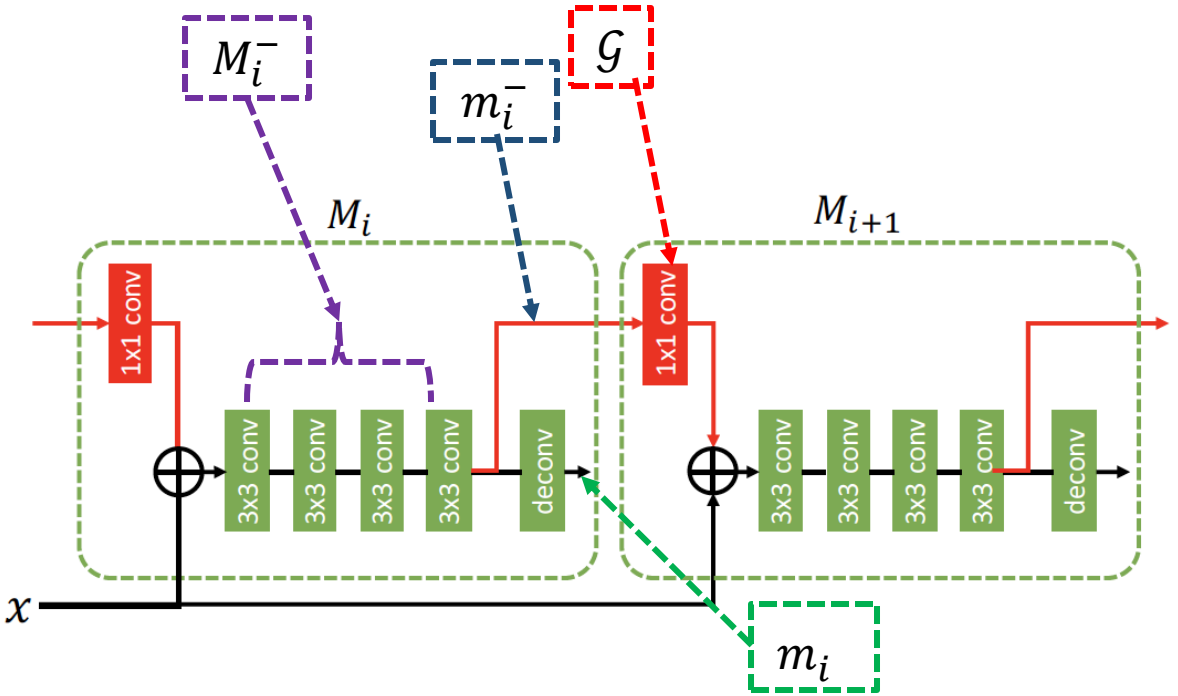


Figure 2: Architecture of multi-stage mask branches.

HTC – Segmentation Branch

- Feature Pyramid for fine pixel prediction
- Pyramid of features
- Element-wise fusion
- 4-layer of further processing
- Semantic features are combined with box/mask features via ROIAlign

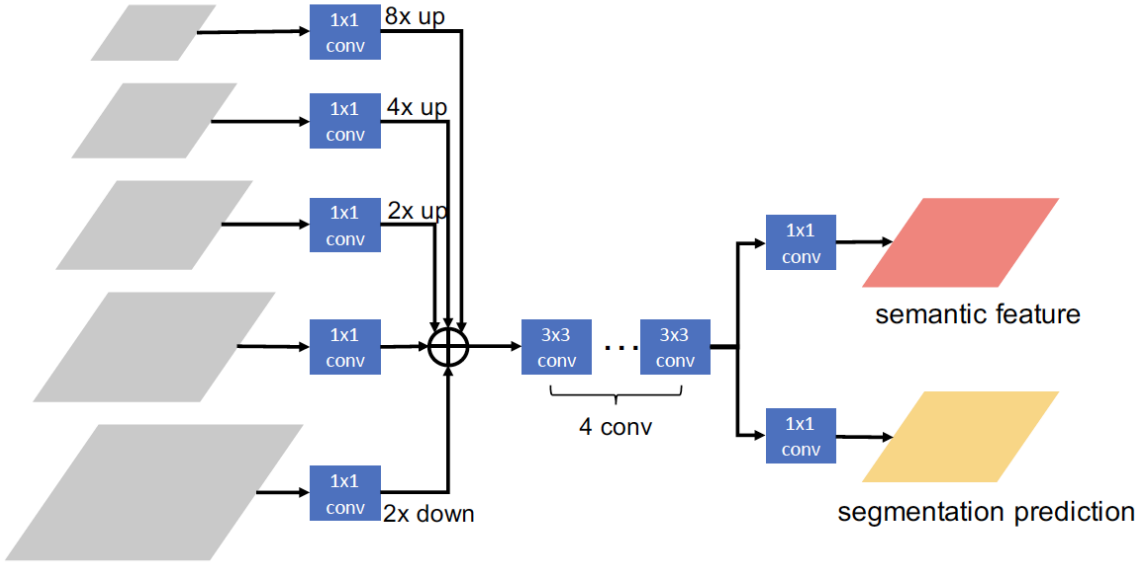
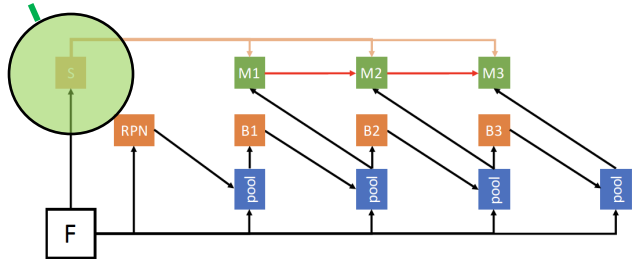


Figure 3: We introduce complementary contextual information by adding semantic segmentation branch.



(d) Hybrid Task Cascade (semantic feature fusion with box branches is not shown on the figure for neat presentation.)

Loss Functions

- $\mathcal{L} = \sum_{t=1}^{T} \alpha_t (\mathcal{L}_{bbox}^t + \mathcal{L}_{mask}^t) + \beta \mathcal{L}_{seg}$
  - $\mathcal{L}_{bbox}^t$ is bounding box prediction at stage t which combines class $\mathcal{L}_{cls}^t$ and bounding box regression $\mathcal{L}_{reg}^t$
  - Same convention as Cascade R-CNN
- $\mathcal{L}_{bbox}^t(c_t, \mathbf{r}_t, \widehat{c_t}, \widehat{\mathbf{r}_t}) = \mathcal{L}_{cls}^t(c_t, \widehat{c_t}) + \mathcal{L}_{reg}^t(\mathbf{r}_t, \widehat{\mathbf{r}_t})$
- $\mathcal{L}_{mask}^t\ (\mathbf{m}_t, \widehat{\mathbf{m}_t}) = BCE\ (\mathbf{m}_t, \widehat{\mathbf{m}_t})$
  - $\mathcal{L}_{mask}^t$ is mask prediction at stage t
  - Same convention as Mask R-CNN
- $\mathcal{L}_{seg}\ (\mathbf{s}, \hat{\mathbf{s}}) = CE(\mathbf{s}, \hat{\mathbf{s}})$
- $T = 3$, $\alpha = [1, 0.5, 0.25]$, and $\beta = 1$ by default

Performance

- COCO 2017 test-dev (Mask)
  - 38.4 AP ResNet-50-FPN
  - 39.7 AP ResNet-101-FPN
  - 47.1 AP ResNeXt-101-FPN

Figure 4: Examples of segmentation results on COCO dataset.

Rotated Mask R-CNN

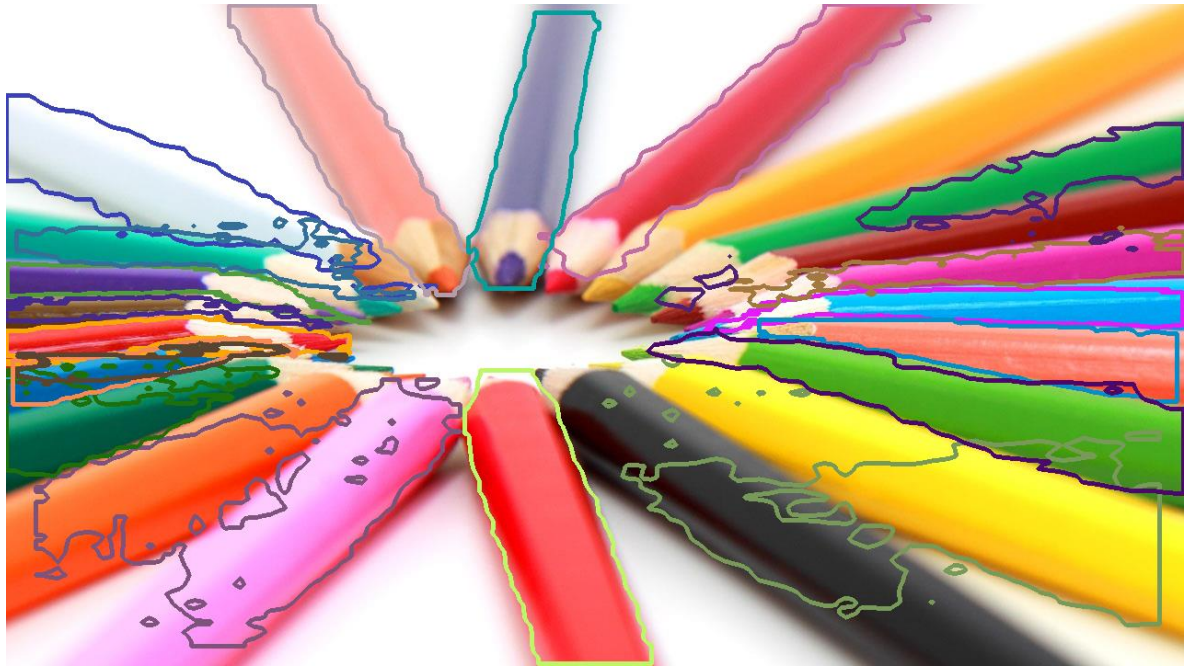
https://github.com/mrlooi/rotated_maskrcnn

Use of Rotated Bounding Boxes

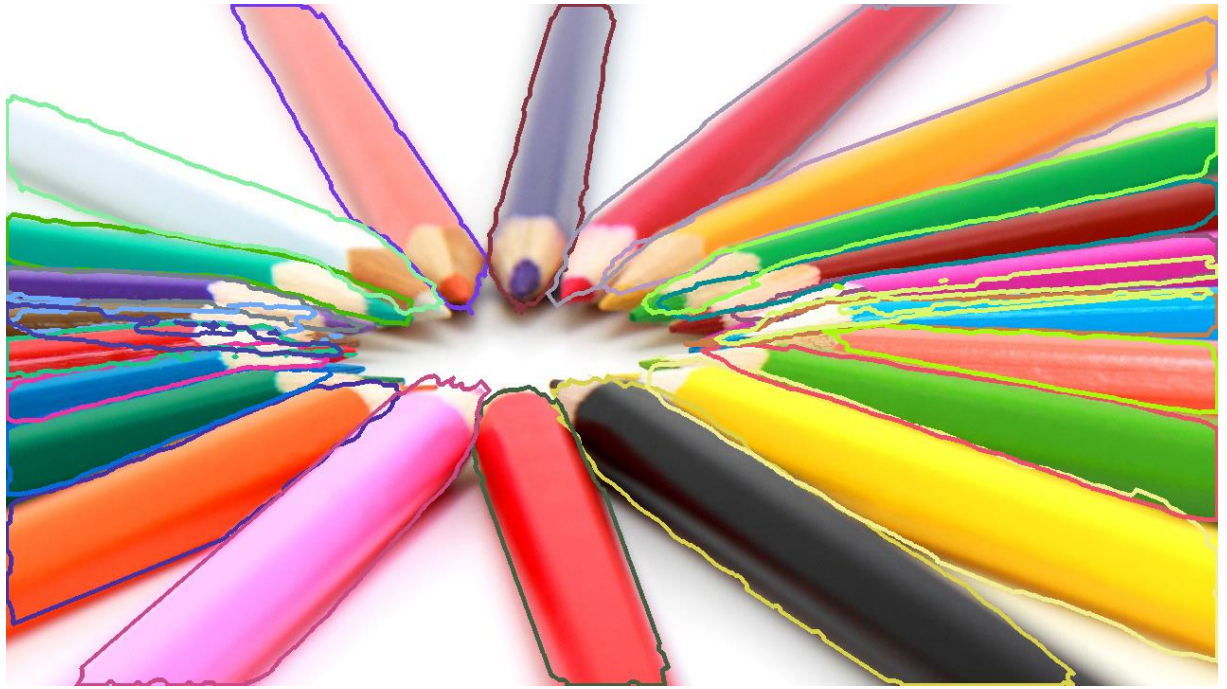
Authors: Paper to follow

Use of Rotated Bounding Boxes

Authors: Paper to follow



Mask R-CNN



Rotated Mask R-CNN

Segment Anything Model

Ravi, Nikhila, et al. "Sam 2: Segment anything in images and videos." *arXiv preprint arXiv:2408.00714* (2024).

SAM2 – Image and Video Segmentation

- Successor of SAM
- Foundation model for promptable image and video segmentation
  - Simply Promptable Visual Segmentation (PVS)
- Inputs: Video/Image + Prompts: input points, boxes, or masks of object of interest
- Outputs: Segmentation/Masklet (spatio-temporal mask)
- New feature – memory to track the masklet across video frames

SAM2

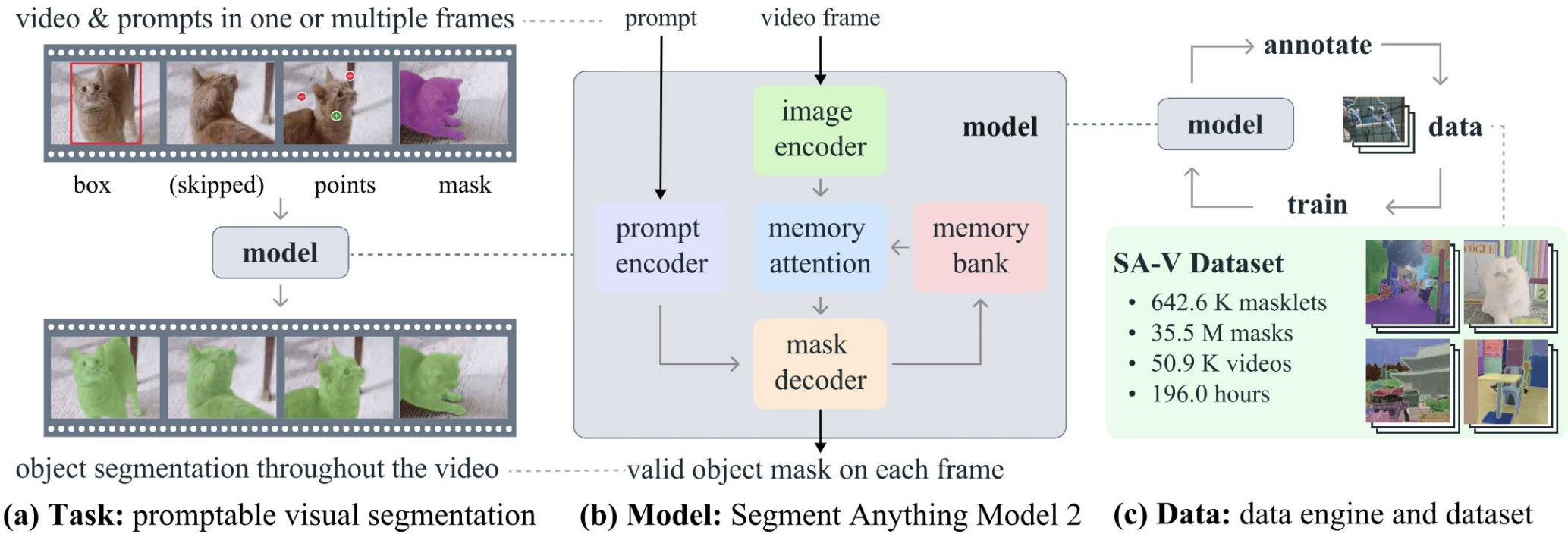


Figure 1 We introduce the Segment Anything Model 2 (SAM 2), towards solving the promptable visual segmentation task (a) with our foundation model (b), trained on our large-scale SA-V dataset collected through our data engine (c). SAM 2 is capable of interactively segmenting regions through prompts (clicks, boxes, or masks) on one or multiple video frames by utilizing a streaming memory that stores previous prompts and predictions.

Data Engine

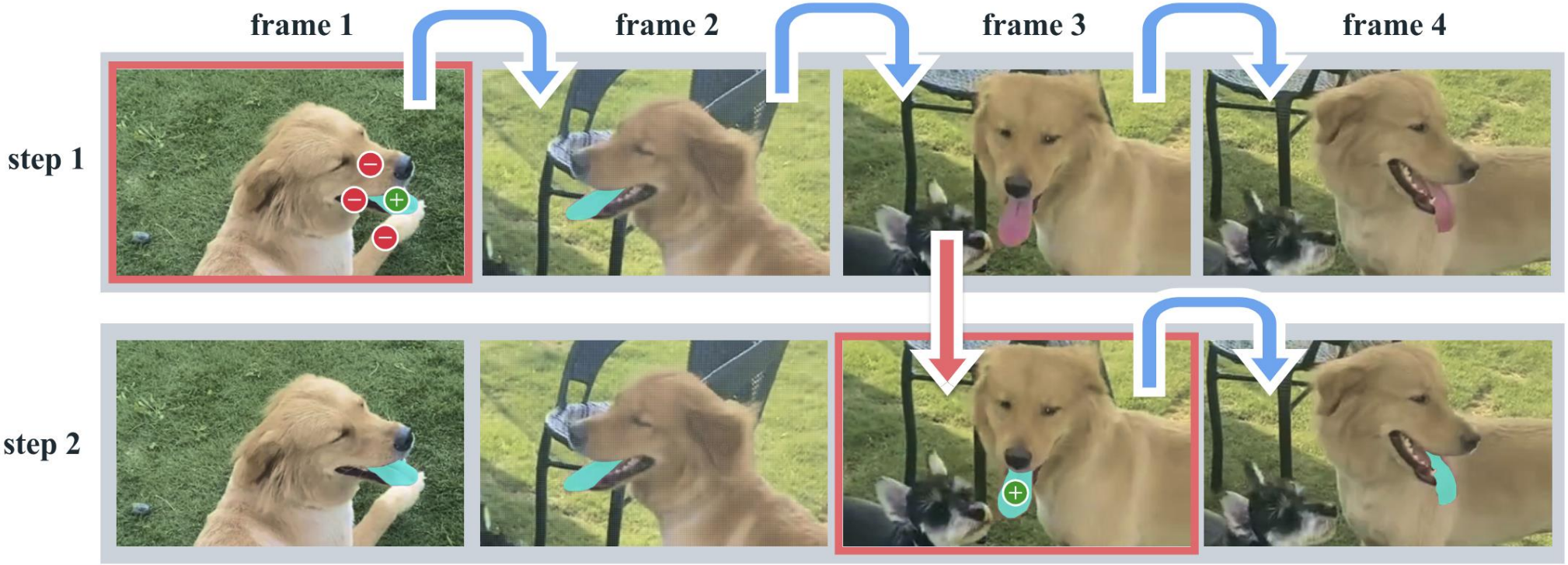


Figure 2 Interactive segmentation with SAM 2. Step 1 (selection): we prompt SAM 2 in frame 1 to obtain the segment of the target object (the tongue). Green/red dots indicate positive/negative prompts respectively. SAM 2 automatically propagates the segment to the following frames (blue arrows) to form a *masklet*. If SAM 2 loses the object (after frame 2), we can correct the masklet by providing an additional prompt in a new frame (red arrow). Step 2 (refinement): a single click in frame 3 is sufficient to recover the object and propagate it to obtain the correct masklet. A decoupled SAM + video tracker approach would require several clicks in frame 3 (as in frame 1) to correctly re-annotate the object as the segmentation is restarted from scratch. With SAM 2's memory, a single click can recover the tongue.

Model

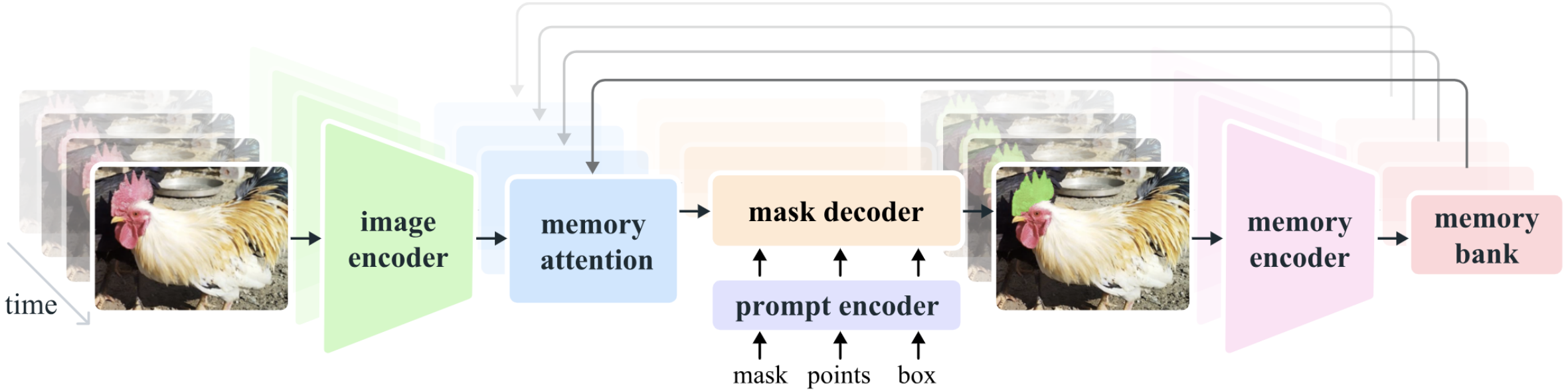


Figure 3 The SAM 2 architecture. For a given frame, the segmentation prediction is conditioned on the current prompt *and/or* on previously observed memories. Videos are processed in a *streaming* fashion with frames being consumed one at a time by the image encoder, and cross-attended to memories of the target object from previous frames. The mask decoder, which optionally also takes input prompts, predicts the segmentation mask for that frame. Finally, a memory encoder transforms the prediction and image encoder embeddings (not shown in the figure) for use in future frames.

Image Encoder

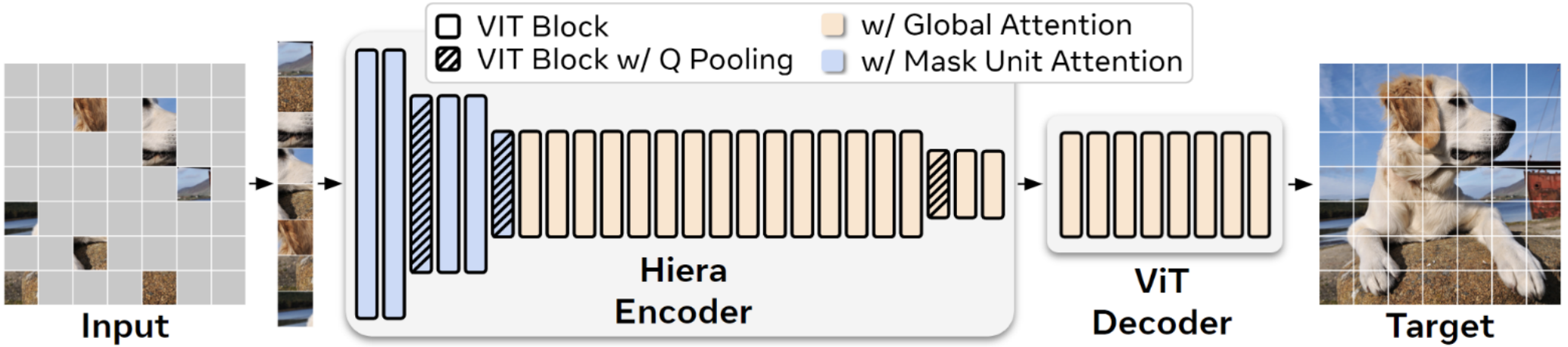


Figure 2. **Hiera Setup.** Modern hierarchical transformers like Swin (Liu et al., 2021) or MViT (Li et al., 2022c) are more parameter efficient than vanilla ViTs (Dosovitskiy et al., 2021), but end up slower due to overhead from adding spatial bias through vision-specific modules like shifted windows or convs. In contrast, we design Hiera to be as simple as possible. To add spatial bias, we opt to *teach* it to the model using a strong pretext task like MAE (pictured here) instead. Hiera consists entirely of standard ViT blocks. For efficiency, we use local attention within "mask units" (Fig. 4, 5) for the first two stages and global attention for the rest. At each stage transition, Q and the skip connection have their features doubled by a linear layer and spatial dimension pooled by a 2×2 maxpool. Hiera-B is shown here (see Tab. 2 for other configs).

Ryali, Chaitanya, et al. "Hiera: A hierarchical vision transformer without the bells-and-whistles." International Conference on Machine Learning. PMLR, 2023.

Memory Attention

- Standard transformer
- Inputs: Cross-Attention of Current Image + Memory Bank Frames + Object Pointers
- Object Pointers – Mask decoder tokens

Prompt Encoder

- Points or Boxes : Positional Encoding + Learnable Embeddings
- Mask prompts : Convolution

Mask Decoder

- 2-way transformer blocks
  - A transformer block with four layers:
    - Self-attention of sparse inputs,
    - Cross attention of sparse inputs to dense inputs,
    - Block on sparse inputs, and
    - Cross attention of dense inputs to sparse inputs.
- Can predict multiple masks due to ambiguities

Memory Encoder

- Convolution of downsampled mask with summing element-wise with current frame embeddings from image encoder

Memory Bank

- FIFO queue of memories of up to N recent frames
- FIFO queue of up to M recent prompted frames

Training

- Image and Video
- Simulate interactive prompting
- Using ground truth masks, points and boxes

Data Engine

- Phase 1 – SAM1 with interactive with human annotators using brush and eraser
- Phase 2 – SAM1 with human in the loop to correct wrong predictions + SAM2 to propagate predictions to succeeding frames
- Phase 3 – SAM2 with additional prompts (points and bbox) with occasional human in the loop to refine predictions

Data Engine

	Model in the Loop	Time per Frame	Edited Frames	Clicks per Clicked Frame	Phase 1 Mask Alignment Score (IoU>0.75)			
					All	Small	Medium	Large
Phase 1	SAM only	37.8 s	100.00 %	4.80	-	-	-	-
Phase 2	SAM + SAM 2 Mask	7.4 s	23.25 %	3.61	86.4 %	71.3 %	80.4 %	97.9 %
Phase 3	**SAM 2**	**4.5 s**	**19.04 %**	**2.68**	**89.1** %	**72.8** %	**81.8** %	**100.0** %

Table 1 Evolution of data engine phases showing the average annotation time per frame, the average percent of edited frames per masklet, the number of manual clicks per clicked frame, and Mask Alignment to Phase 1 by mask size.

Dataset SA-V

	#Videos	Duration	#Masklets	#Masks	#Frames	Disapp. Rate
DAVIS 2017 (Pont-Tuset et al., 2017)	0.2K	0.1 hr	0.4K	27.1K	10.7K	16.1 %
YouTube-VOS (Xu et al., 2018b)	4.5K	5.6 hr	8.6K	197.3K	123.3K	13.0 %
UVO-dense (Wang et al., 2021b)	1.0K	0.9 hr	10.2K	667.1K	68.3K	9.2 %
VOST (Tokmakov et al., 2022)	0.7K	4.2 hr	1.5K	175.0K	75.5K	41.7 %
BURST (Athar et al., 2022)	2.9K	28.9 hr	16.1K	600.2K	195.7K	37.7 %
MOSE (Ding et al., 2023)	2.1K	7.4 hr	5.2K	431.7K	638.8K	41.5 %
Internal	62.9K	281.8 hr	69.6K	5.4M	6.0M	36.4 %
SA-V Manual	50.9K	196.0 hr	190.9K	10.0M	4.2M	42.5 %
SA-V Manual+Auto	50.9K	196.0 hr	642.6K	35.5M	4.2M	27.7 %

Table 3 Comparison of our datasets with open source VOS datasets in terms of number of videos, duration, number of masklets, masks, frames, and disappearance rate. SA-V Manual contains only manually annotated labels. SA-V Manual+Auto combines manually annotated labels with automatically generated masklets.

Benchmark on Image Segmentation

		1 (5) click mIoU				
Model	Data	SA-23 All	SA-23 Image	SA-23 Video	14 new Video	FPS
SAM	SA-1B	58.1 (81.3)	60.8 (82.1)	54.5 (80.3)	59.1 (83.4)	21.7
SAM 2	SA-1B	58.9 (81.7)	60.8 (82.1)	56.4 (81.2)	56.6 (83.7)	**130.1**
SAM 2	our mix	**61.9 (83.5)**	**63.3 (83.8)**	**60.1 (83.2)**	**69.6 (85.8)**	**130.1**

Table 5 Zero-shot accuracy on the Segment Anything (SA) task across 37 datasets. The table shows the average 1- and 5-click mIoU of SAM 2 compared to SAM by domains (image/video). We report the average metrics on the 23 datasets used by SAM (SA-23) and the average across 14 additional zero-shot video datasets (as detailed in §F.3).

VOS Metric

- $\mathcal{J}$ – region metric that computes the number of pixels of the intersection between the two masks and divides it by the size of the union (also called Intersection over Union - IoU, or Jaccard index).
- $\mathcal{F}$ - the accuracy in the boundaries, via a bipartite matching between the boundary pixels of both masks.
  - The final boundary measure is the F measure between the precision and recall of the matching

Benchmark on Video Object Segmentation

Method	$\mathcal{J}\&\mathcal{F}$					$\mathcal{G}$
	MOSE val	DAVIS 2017 val	LVOS val	SA-V val	SA-V test	YTVOS 2019 val
STCN (Cheng et al., 2021a)	52.5	85.4	-	61.0	62.5	82.7
SwinB-AOT (Yang et al., 2021b)	59.4	85.4	-	51.1	50.3	84.5
SwinB-DeAOT (Yang & Yang, 2022)	59.9	86.2	-	61.4	61.8	86.1
RDE (Li et al., 2022a)	46.8	84.2	-	51.8	53.9	81.9
XMem (Cheng & Schwing, 2022)	59.6	86.0	-	60.1	62.3	85.6
SimVOS-B (Wu et al., 2023b)	-	88.0	-	44.2	44.1	84.2
JointFormer (Zhang et al., 2023b)	-	90.1	-	-	-	87.4
ISVOS (Wang et al., 2022)	-	88.2	-	-	-	86.3
DEVA (Cheng et al., 2023b)	66.0	87.0	55.9	55.4	56.2	85.4
Cutie-base (Cheng et al., 2023a)	69.9	87.9	66.0	60.7	62.7	87.0
Cutie-base+ (Cheng et al., 2023a)	71.7	88.1	-	61.3	62.8	87.5
SAM 2 (Hiera-B+)	76.6	90.2	**78.0**	76.8	77.0	88.6
SAM 2 (Hiera-L)	**77.9**	**90.7**	**78.0**	**77.9**	**78.4**	**89.3**

Table 6 VOS comparison to prior work. SAM 2 performs well in accuracy ($\mathcal{J}\&\mathcal{F}$, $\mathcal{G}$) for video segmentation based on first-frame ground-truth mask prompts. SAM 2 performs significantly better on SA-V val/test.

End