DiscoBox: Weakly Supervised Instance Segmentation and Semantic Correspondence from Box Supervision - Supplementary

Shiyi Lan[1]* Zhiding Yu[2]† Christopher Choy[2] Subhashree Radhakrishnan[2] Guilin Liu[2]
Yuke Zhu[2,3] Larry S. Davis[1] Anima Anandkumar[2,4]
[1]University of Maryland, College Park [2]NVIDIA [3]The University of Texas at Austin [4]Caltech

A. Mean-field inference

Recap that we denote the input image as $\boldsymbol{I}$, and have a set of box region proposals $\boldsymbol{R} = \{\boldsymbol{r}^n | n = 1, ..., N\}$. Each box proposal corresponds to an RoI feature map $\boldsymbol{f}^n$ of size $C \times H \times W$. Additionally, the box proposals correspond to a set of object masks $\boldsymbol{M} = \{\boldsymbol{m}^n | n = 1, .., N\}$, where each $\boldsymbol{m}^n$ is an $H \times W$ probability map.

Without loss of generality, we assume $\boldsymbol{x}^n \in \{0, 1\}^{H \times W}$ is the labeling of a box $\boldsymbol{r}^n$ from current batch, and $\boldsymbol{x}^s \in \{0, 1\}^{H \times W}$ is the labeling of an intra-class box $\boldsymbol{r}^s$, where $s \in \boldsymbol{N}_c(n)$ is an index of all the retrieved boxes. We denote $\boldsymbol{m}^n$ and $\boldsymbol{m}^s$ the predicted mask probability maps and $\boldsymbol{I}^n$ and $\boldsymbol{I}^s$ the cropped RGB images of $\boldsymbol{r}^n$ and $\boldsymbol{r}^s$. We further denote i, j, k the indices of box pixels and $\boldsymbol{N}_p(i)$ the set of 8-connected immediate neighbors of pixel i. We also assume a dense correspondence $\boldsymbol{T}_{ns}$ has been established between $\boldsymbol{r}^n$ and $\boldsymbol{r}^s$ based on the cost volume $\boldsymbol{C} = \boldsymbol{C}_u + \boldsymbol{C}_g$. The mean field inference method is shown in Algorithm 1:

Algorithm 1 Mean Field Inference.

```
1: procedure MEAN FIELD($\boldsymbol{m}^n, \boldsymbol{m}^s, \boldsymbol{I}^n, \boldsymbol{I}^s, \boldsymbol{T}_{ns}, \boldsymbol{C}$)
2:     $\varphi(x) = \mathbb{1}[x \leq 0.5] * 0.3 + \mathbb{1}[x > 0.5] * 0.7$    ▷ threshold function.
3:     $\boldsymbol{q}^n \leftarrow -\log(\varphi(\boldsymbol{m}^n)),\ \boldsymbol{q}^s \leftarrow -\log(\varphi(\boldsymbol{m}^s))$
4:     $\boldsymbol{k}(i, j) \leftarrow w_1 \exp(-\frac{|\boldsymbol{I}_i^n - \boldsymbol{I}_j^n|^2}{2\zeta^2})$    ▷ pairwise kernels
5:     while not converge do    ▷ iterate until convergence
6:         for $i \in \boldsymbol{r}^n$ do
7:             $\hat{q}_i^n \leftarrow 0$
8:             for $j \in \boldsymbol{N}(i)$ do    ▷ pairwise potentials
9:                 $\hat{q}_i^n \leftarrow \hat{q}_i^n + \boldsymbol{k}(i, j) q_j^n$
10:            end for
11:            for $s \in \boldsymbol{N}_c(n)$ do
12:                for $k \in \boldsymbol{r}^s$ do    ▷ cross-image potentials
13:                    $\hat{\mathbf{x}}_i^n \leftarrow \hat{\mathbf{x}}_i^n + w_2 \boldsymbol{T}_{ns}(i, k) \boldsymbol{C}(i, k) \boldsymbol{x}_k^s$
14:                end for
15:            end for
16:        end for
17:        $\boldsymbol{q}^n \leftarrow \varphi(\exp(-\hat{\boldsymbol{q}}^n - \boldsymbol{q}^n))$    ▷ local update
18:        normalize $\boldsymbol{q}^n$    ▷ normalization
19:        $\boldsymbol{q}^n \leftarrow -\log(\boldsymbol{q}^n)$
20:    end while
21:    $\boldsymbol{x}^n \leftarrow \mathbb{1}[\exp(-\boldsymbol{q}^n) > 0.5]$
22:    return $\boldsymbol{x}^n$
23: end procedure
```

B. Sinkhorn's algorithm

Given any cost volume $\boldsymbol{C}$, we use Sinkhorn's Algorithm [1] to find out the one-to-one assignment approximately. We define a threshold function $\varphi^o(x) = \mathbb{1}[x > 0.5] * 0.4 + 0.6$, and use the function to obtain $\boldsymbol{\mu}_a = \varphi^o(\boldsymbol{m}^a)$ and $\boldsymbol{\mu}_b = \varphi^o(\boldsymbol{m}^b)$. The Sinkhorn's algorithm to obtain the transport matrix is described in the Algorithm 2:

C. Additional implementation details

C.1. Data loading and augmentation

Disco-Box follows the original data loading and augmentation settings in YOLACT++ and SOLOv2:

YOLACT++. In YOLACT++ [2], an input image is resized without changing its aspect ratio so that its longer side is equal to L. Random cropping is then applied on the resized image crop size $L \times L$. In case the crop goes outside the image, the outside part of the crop is filled with the mean RGB values. Color jittering and random flipping are then applied on top of random cropping as data augmentation. The original paper of YOLACT++ has reported results with different L such as 550 and 700. We follow the same setting of loading and augmentation with $L = 550$.

SOLOv2. SOLOv2 [3] also resizes the longer image side to L. Random cropping is then applied on the resized image with crop size $L \times W = 1333 \times 800$, where L and W always correspond to the longer and shorter side of the resized input image, respectively. Random flipping is applied on top of random cropping as data augmentation. Our Disco-Box

*Work done during an internship at NVIDIA Research.
†Corresponding author: <zhidingy@nvidia.com>.

Algorithm 2 Sinkhorn's Algorithm

```
procedure $\mathcal{H}(\boldsymbol{C}, \boldsymbol{\mu}_a, \boldsymbol{\mu}_b, \varepsilon, t_{max})$
    $\boldsymbol{K} = e^{(-\boldsymbol{C}/\varepsilon)}$
    $b \leftarrow \mathbf{1}$
    $t \leftarrow 0$
    while $t \leq t_{max}$ and not converge do
        $\boldsymbol{a} = \frac{\mu_a}{\boldsymbol{K}\boldsymbol{b}}$
        $\boldsymbol{b} = \frac{\mu_b}{\boldsymbol{K}^\intercal \boldsymbol{a}}$
    end while
    $\boldsymbol{T} = diag((a))\boldsymbol{K}diag(\boldsymbol{b})$
    return $\boldsymbol{T}$
end procedure
```

with SOLOv2 architecture follows the same data loading and augmentation strategy with SOLOv2.

C.2. Learning and optimization

All our experiments are conducted on DGX-1 machine with 8 NVIDIA Tesla V100 GPUs. Note that we follow the same learning and optimization settings as YOLACT++ and SOLOv2 as elaborated below:

YOLACT++. We follow the original setting of [2] for DISCO-BOX with YOLACT++ architecture. We adopt a batch size of 8 on each GPU. On COCO, the initial learning rate is set to 1×10^{-3} and is decreased to 1×10^{-4} and 1×10^{-5} after 280K and 360K iterations, respectively. Warm-up is used in the first 500 iterations to prevent gradient explosion. On PASCAL VOC 2012, the same warm-up and initial learning rate is applied, and the learning rate is decreased to 1×10^{-4} and 1×10^{-5} after 60K and 100K iterations, respectively. It takes about 5 days to train on COCO and about 12 hours on VOC12.

SOLOv2. We follow the original training settings of [3] where multi-scale training are applied for ResNeXt-101-DCN and ResNet-101-DCN backbones. On COCO, the batch size is set to 2 on each GPU. The initial learning rate is set to 1×10^{-2} and decreased to 1×10^{-3} and 1×10^{-4} after 26 epochs and 32 epochs, respectively. Warm-up is used in the first 2000 iterations. It takes about 4 days to train on COCO with a ResNet-50 backbone.

C.3. Object retrieval with memory bank

Pushing objects. We maintain an independent first-in-first-out queue with size 100 for every category as the memory bank. The RoI features and predicted masks of objects in each batch are pushed into the memory bank. Note that the RoI features and masks are obtained by the teacher network with respect to the ground truth bounding boxes.

Retrieving objects. To construct intra-class pairs for every batch during training, we define an object in the current batch as a *query*, and use it to retrieve the RoI features and mask probability maps of intra-class objects from a memory bank. This allows us to conveniently construct pairs without significant extra computation. Note that we empirically set the maximum number of retrieved objects to be 10 for every query object. This is done by random sampling from the memory bank. We ignore the pairs when the bank size (and thus the number of retrieved objects) is smaller than 5.

D. Additional visualization

Finally, we provide visualization of instance segmentation results which are not presented in the main paper due to limited space. Fig. 1 shows the results obtained by DISCO-BOX SOLOv2/ResNeXt-101-DCN on COCO. In addition, Fig. 2 shows the results obtained by YOLACT++/ResNet-50-DCN on VOC12. One could see that the predicted masks are of high qualities in general, even though some examples can be cluttered and challenging.

References

[1] Philip A Knight. The Sinkhorn–Knopp algorithm: Convergence and applications. *SIAM Journal on Matrix Analysis and Applications*, 2008. 1

[2] Daniel Bolya, Chong Zhou, Fanyi Xiao, and Yong Jae Lee. YOLACT++: Better real-time instance segmentation. *IEEE Trans. PAMI*, 2020. 1, 2

[3] Xinlong Wang, Rufeng Zhang, Tao Kong, Lei Li, and Chunhua Shen. Solov2: Dynamic and fast instance segmentation. *NeurIPS*, 2020. 1, 2

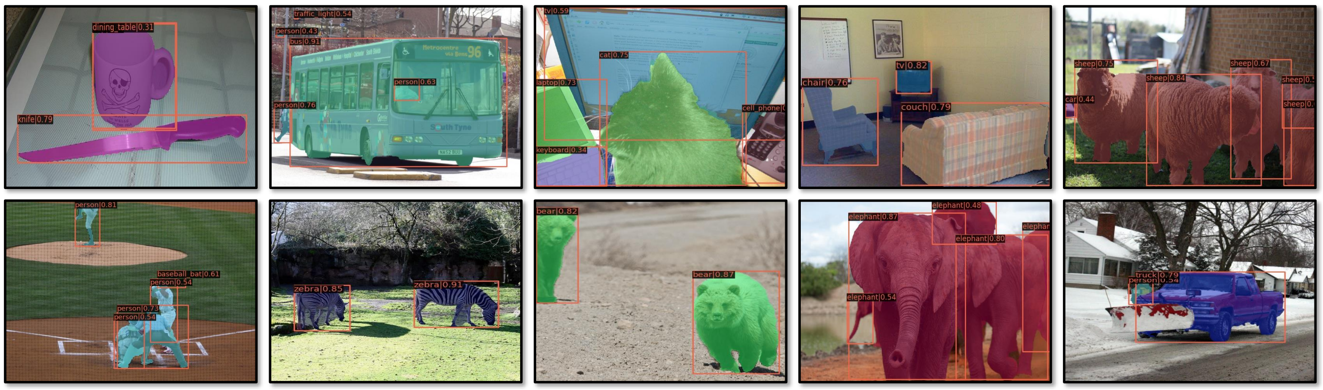
Figure 1. Visualization of instance segmentation on COCO (SOLOv2/ResNeXt-101-DCN).

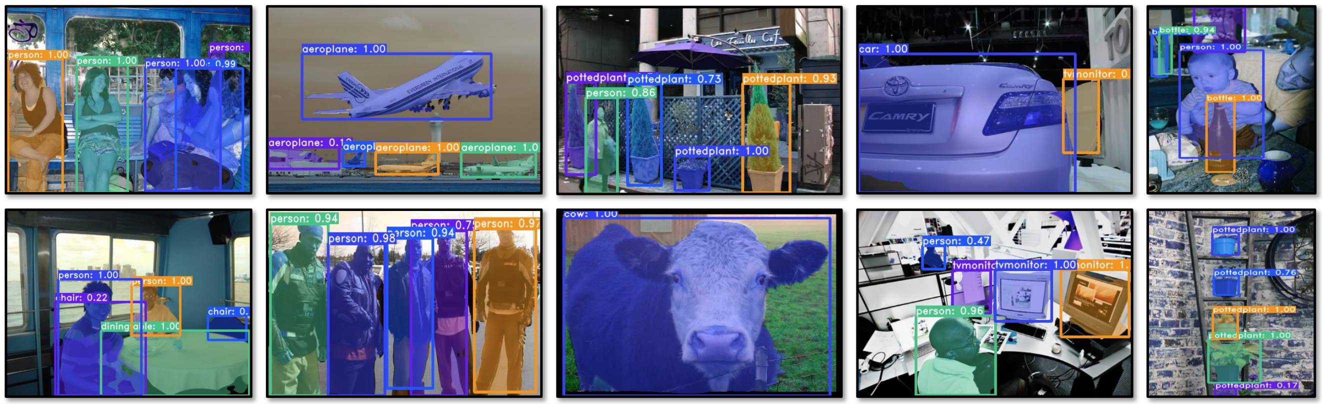
Figure 2. Visualization of instance segmentation on VOC12 (YOLACT++/ResNet-50-DCN).