SOLOv2: Dynamic, Faster and Stronger

Xinlong Wang[1] Rufeng Zhang[2] Tao Kong[3*] Lei Li[3] Chunhua Shen[1*]
[1] The University of Adelaide, Australia [2] Tongji University, China [3] ByteDance AI Lab

Abstract

In this work, we aim at building a simple, direct, and fast instance segmentation framework with strong performance. We follow the principle of the SOLO method of Wang et al. "SOLO: segmenting objects by locations" [33]. Importantly, we take one step further by dynamically learning the mask head of the object segmenter such that the mask head is conditioned on the location. Specifically, the mask branch is decoupled into a mask kernel branch and mask feature branch, which are responsible for learning the convolution kernel and the convolved features respectively.

Moreover, we propose Matrix NMS (non maximum suppression) to significantly reduce the inference time overhead due to NMS of masks. Our Matrix NMS performs NMS with parallel matrix operations in one shot, and yields better results. We demonstrate a simple direct instance segmentation system, outperforming a few state-of-the-art methods in both speed and accuracy. A light-weight version of SOLOv2 executes at 31.3 FPS and yields 37.1% AP. Moreover, our state-of-the-art results in object detection (from our mask byproduct) and panoptic segmentation show the potential to serve as a new strong baseline for many instance-level recognition tasks besides instance segmentation. Code is available at: git.io/AdelaiDet

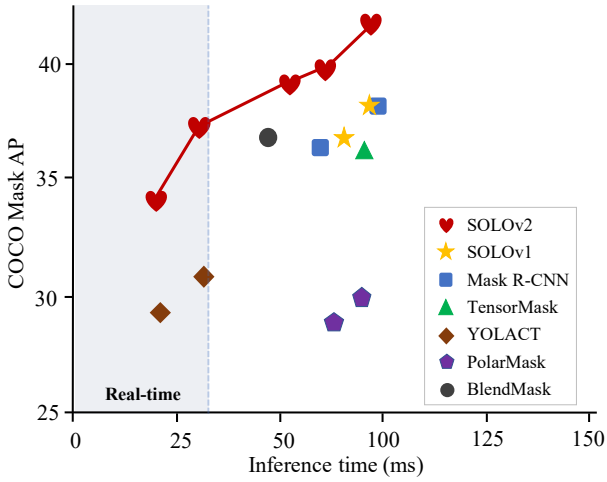


Figure 1 – Speed-accuracy trade-off on the COCO `test-dev` for a few recent instance segmentation methods. The proposed SOLOv2 outperforms a range of state-of-the-art algorithms. Inference time of all methods is tested on a Tesla V100-GPU machine.

1. Introduction

Generic object detection demands for the functions of localizing individual objects and recognizing their categories. For representing the object locations, bounding box stands out for its simplicity. Localizing objects using bounding boxes have been extensively explored, including the problem formulation, network architecture, post-processing and all those focusing on optimizing and processing the bounding boxes. The tailored solutions largely boost the performance and efficiency, thus enabling wide downstream applications recently. However, bounding boxes are coarse and unnatural. Human vision can effortlessly localize objects by their boundaries. Instance segmentation, *i.e.*, localizing objects using masks, pushes object localization to the limit at pixel level and opens up opportunities to more instance-level perception and applications. To date, most existing methods deal with instance segmentation in the view of bounding boxes, *i.e.*, segmenting objects in (anchor) bounding boxes. How to develop pure instance segmentation including the supporting facilities, *e.g.*, post-processing, is largely unexplored compared to bounding box detection and instance segmentation methods built on top it.

In the recently proposed SOLO[1], the task of instance segmentation is formulated as two sub-tasks of pixel-level classification, solvable using standard FCNs, thus dramatically simplifying the formulation of instance segmentation. SOLO takes an image as input, directly outputs instance masks and corresponding class probabilities, in a fully convolutional, box-free and grouping-free paradigm. As such,

*Correspondence should be addressed to: kongtao@bytedance.com, chunhua.shen@adelaide.edu.au

[1]Hereafter SOLO and SOLOv1 are used interchangeably, referring to the work of [33].

the research focus is shifted to how to generate better object masks. We need to develop techniques focusing on masks rather than boxes, to boost the performance as well as to accelerate the inference. In this work, we improve SOLO from these two aspects: mask learning and mask NMS.

We first introduce a dynamic scheme, which enables dynamically segmenting objects by locations. Specifically, the mask learning can be divided into two parts: convolution kernel learning and feature learning. When classifying the pixels into different location categories, the classifiers are predicted by the network and conditioned on the input. About feature learning, as envisioned in SOLO, more techniques developed in semantic segmentation could be applied to boost the performance. Inspired by the semantic FPN in [17], we construct a unified and high-resolution mask feature representation for instance-aware segmentation. A step-by-step derivation of mask learning from SOLOv1 to SOLOv2 is shown in Section 3.

We further propose an efficient and effective matrix NMS algorithm. As a post-processing step for suppressing the duplicate predictions, non-maximum suppression (NMS) serves as an integral part in state-of-the-art object detection systems. Take the widely adopted multi-class NMS for example. For each class, the predictions are sorted in descending order by confidence. Then for each prediction, it removes all other highly overlapped predictions. The sequential and recursive operations result in non-negligible latency. For mask NMS, the drawback is magnified. Compared to bounding box, it takes more time to compute the IoU of each mask pair, thus leading to a large overhead. We address this problem by introducing Matrix NMS, which performs NMS with parallel matrix operations in one shot. Our Matrix NMS outperforms the existing NMS and the varieties in both accuracy and speed. As a result, Matrix NMS processes 500 masks in less than 1 ms in simple PyTorch implementation, and outperforms the recently proposed Fast NMS [2] by 0.4% AP.

With the improvements, SOLOv2 outperforms SOLOv1 by 1.9% AP while being 33% faster. The Res-50-FPN SOLOv2 achieves 38.8% mask AP at 18 FPS on the challenging MS COCO dataset, evaluated on a single V100 GPU card. A light-weight version of SOLOv2 executes at 31.3 FPS and yields 37.1% mask AP. Interestingly, although the concept of bounding box is thoroughly eliminated in our method, our bounding box byproduct, *i.e.*, by directly converting the predicted mask to its bounding box, yield 42.4% AP for bounding box object detection, which even *surpasses many state-of-the-art, highly-engineered object detection methods*.

We believe that, with simple, fast and sufficiently strong solutions, instance segmentation should be an advanced alternative to the widely used object bounding box detection, and SOLOv2 may play an important role and predict its wide applications.

2. Related Work

Here we review some recent work closest to ours.

2.1. Instance Segmentation

Instance segmentation is a challenging task, as it requires instance-level and pixel-level predictions simultaneously. The existing approaches can be summarized into three categories. Top-down methods [20, 12, 27, 13, 6, 2, 3, 38] solve the problem from the perspective of object detection, *i.e.*, detecting first and then segment the object in the box. In particular, recent methods of [3, 38, 35] build their methods on the anchor-free object detector FCOS [32], showing promising performance. Bottom-up methods [29, 9, 26, 10] view the task as a label-then-cluster problem, *e.g.*, learn the per-pixel embeddings and then cluster them into groups. The latest direct method [33] aims at dealing with instance segmentation directly, without dependence on box detection or embedding learning. In this work, We inherit the core design of SOLO and further explore the direct instance segmentation solutions.

We specifically compare our method with the recent YOLACT [2]. YOLACT learns a group of coefficients which are normalized to $[-1, 1]$ for each anchor box. During the inference, it first performs a bounding box detection and then use the predicted boxes to crop the assembled masks. BlendMask improves YOLACT, achieving a better balance betweem accuracy and speed [3].

While our method is evolved from SOLO [33] through directly decoupling the original mask prediction to kernel learning and feature learning. No anchor box is needed. No normalization is needed. No bounding box detection is needed. We directly map the input image to the desired object classes and object masks. Both the training and inference are much simpler. As a result, our proposed framework is much simpler, yet achieving significantly better performance (6% AP better at a comparable speed); and our best model achieves 41.7 AP vs. YOLACT's best 31.2% AP.

2.2. Dynamic Convolutions

In traditional convolution layers, the learned convolution kernels stay fixed and being independent on the input, *e.g.*, the weights are the same for every image and every location of the image. Some previous works explore the idea of bringing more flexibility into the traditional convolutions. Spatial Transform Networks [15] predicts a global parametric transformation to warp the feature map, allowing the network to adaptively transform feature maps conditioned on the input. Dynamic filter [16] is proposed to actively predict the parameters of the convolution filters. It applies dynamically generated filters to an image in a sample-specific way. Deformable Convolutional Networks [8] dynamically learn

the sampling locations by predicting the offsets for each image location. We bring the dynamic scheme into instance segmentation and enable learning instance segmenters by locations.

2.3. Non-Maximum Suppression

NMS is widely adopted in many computer vision tasks and becomes an essential component of object detection systems. Some recent works are proposed to improve the traditional NMS. They can be divided into two groups, either for improving the accuracy or speeding up. Instead of applying the hard removal to duplicate predictions according to a threshold, Soft-NMS [1] decreases the confidence scores of neighbors according to their overlap with higher scored predictions. The detection accuracy is slightly improved over the traditional NMS but inference is slow due to the sequential operations. Adaptive NMS [25] applies dynamic suppression threshold to each instance, which is tailored for pedestrian detection in a crowd. To accelerate the inference, Fast NMS proposed in [2] enables deciding the predictions to be kept or discarded predictions in parallel. Note that it speeds up at the cost of performance deterioration. Different from the previous methods, our Matrix NMS addresses the issues of hard removal and sequential operations at the same time. As a result, *the proposed Matrix NMS is able to process 500 masks in less than 1 ms* in simple PyTorch implementation, which is negligible compared with the time of network evaluation, and yields 0.4% AP better than Fast NMS.

3. A Revisit to SOLOv1

The core idea of SOLOv1 framework is to segment objects by locations. The input image is conceptually divided into $S \times S$ grids. If the center of an object falls into a grid cell, that grid cell is responsible for predicting the semantic category as well as assigning the per-pixel location categories. There are two branches: category branch and mask branch. Category branch predicts the semantic categories, while the mask branch segments the object instance. Concretely, category branch ouputs $S \times S \times C$ shaped tensor, where C is the number of object classes. Mask branch generates output tensor $M \in \mathbb{R}^{H \times W \times S^2}$. The k^{th} in S^2 channel is responsible to segment instance at grid (i, j), where $k = i \cdot S + j$.

We zoom in to show what happens in the last layer of the mask branch. The last layer is a 1×1 convolution layer which takes feature $F \in \mathbb{R}^{H \times W \times E}$ as input and produces S^2 output channels, *i.e.*, the tensor M. The convolution kernel is $G \in \mathbb{R}^{1 \times 1 \times E \times S^2}$. The operation can be written as:

$$M = F * G. \tag{1}$$

This layer can be viewed as S^2 classifiers. Each classifier is responsible for classifying whether the pixels belonging to this location category.

As discussed in [33], the prediction M is somewhat redundant as in most cases the objects are located sparsely in the image. It means that only a small part of S^2 classifiers actually functions during a single inference. Decoupled SOLO [33] addresses this issue by decoupling the S^2 classifiers into two groups of S classifiers, corresponding to S horizontal and S vertical location categories respectively. Thus the output space is decreased from $H \times W \times S^2$ to $H \times W \times 2S$.

From another perspective, since output M is redundant, and feature F is fixed, why not directly learning the convolution kernel G? In this way, we can simply pick the valid ones from predicted S^2 classifiers and perform the convolution dynamically. The number of model parameters also decreases. What's more, as the predicted kernel is generated dynamically conditioned on the input, it benefits from the flexibility and adaptive nature. Additionally, each of S^2 classifiers is conditioned on the location. It is in accordance with the core idea of segmenting objects by locations and goes a step further by predicting the segmenters by locations. We illustrate the detailed method in Section 4.1.

4. SOLOv2

We present the details of the proposed SOLOv2 design in this section.

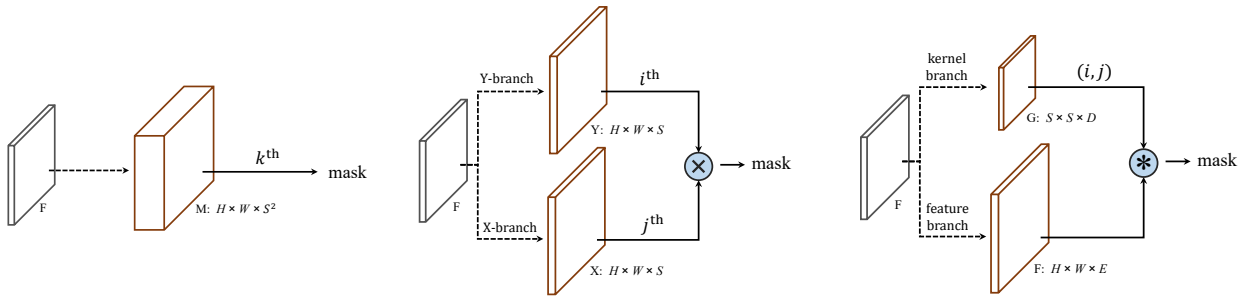
4.1. Dynamic Instance Segmentation

We inherit the most of settings from SOLOv1, *e.g.*, grid cells, multi-level prediction, CoordConv [24] and loss function. Based on that, we introduce the dynamic scheme, in which the original mask branch is decoupled into a mask kernel branch and a mask feature branch, for predicting the convolution kernel and the convolved features respectively. We show the comparisons with SOLOv1 in Figure 2.

4.1.1 Mask Kernel Branch

The mask kernel branch lies in the prediction head, along with the semantic category branch. The head works on a pyramid of feature maps generated by FPN [21]. Both the branches in the head consist of 4×convs for feature extraction an a final one conv for prediction. Weights for the head are shared across different feature map levels. We add the spatial functionality to the kernel branch by giving the first convolution access to the normalized coordinates, *i.e.*, concatenating two additional input channels.

For each grid, the kernel branch predicts the D-dimensional output to indicate predicted convolution kernel weights, where D is the number of parameters. For generating the weights of a 1×1 convolution with E input channels, D equals E. As for 3×3 convolution, D equals $9E$. These



(a) Vanilla head (b) Decoupled head (c) Dynamic head

Figure 2 – Dynamic SOLOv2 head compared to SOLOv1 heads. F is input feature. Dashed arrows denote convolutions. $k = i \cdot S + j$. '$\otimes$' denotes element-wise multiplication; and '$\circledast$' denotes convolution.

generated weights are conditioned on the locations, *i.e.*, the grid cells. If we divide the input image into $S \times S$ grids, the output space will be $S \times S \times D$, There is no activation function on the output.

4.1.2 Mask Feature Branch

The mask feature branch needs to predict instance-aware feature maps $F \in \mathbb{R}^{H \times W \times E}$, where E is the dimension of mask feature. F will be convolved by the output of mask kernel branch. If all the predicted weights are used, *i.e.*, S^2 classifiers, the outputted instance mask after the final convolution will be in $H \times W \times S^2$, which is the same as the output space of SOLOv1.

Since the mask feature and mask kernel are decoupled and separately predicted, there are two ways to construct the mask feature branch. We can put it into the head, along with the kernel branch. It means that we predict the mask features for each FPN levels. Or, to predict a unified mask feature representation for all FPN levels. We have compared the two implementations in Section 5.1.3 by experiments. Finally, we employ the latter one for its effectiveness and efficiency.

For learning a unified and high-resolution mask feature representation, we apply feature pyramid fusion inspired by the semantic segmentation in [17]. After repeated stages of 3×3 conv, group norm [34], ReLU and $2\times$ bilinear upsampling, the FPN features P2 to P5 are merged into a single output at 1/4 scale. The last layer after the element-wise summation consists of 1×1 convolution, group norm and ReLU. The details are illustrated in Figure 3. It should be noted that we feed normalized pixel coordinates to the deepest FPN level (at 1/32 scale), before the convolutions and bilinear upsamplings. The provided accurate position information is important for enabling position sensitivity and predicting instance-aware features.

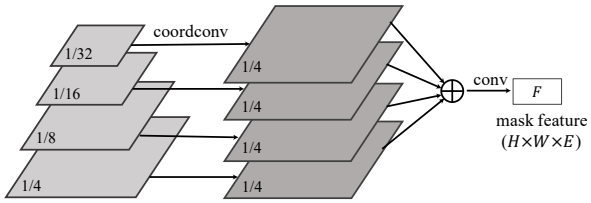


Figure 3 – Unified mask feature branch. Each FPN level (left) is upsampled by convolutions and bilinear upsampling until it reaches 1/4 scale (middle). In the deepest FPN level, we concatenate the x, y coordinates and the original features to encode spatial information. After element-wise summation, a 1×1 convolution is attached to transform to designated output mask feature $F \in \mathbb{R}^{H \times W \times E}$.

4.1.3 Forming Instance Mask

For each grid cell at (i, j), we first obtain the mask kernel $G_{i,j,:} \in \mathbb{R}^D$. Then $G_{i,j,:}$ is convoluted with F to get the instance mask. In total, there will be at most S^2 masks for each prediction level. Finally, we use the proposed Matrix NMS to get the final instance segmentation results.

4.1.4 Learning and Inference

The label assignment and loss functions are the same as SOLOv1. The training loss function is defined as follows:

$$L = L_{cate} + \lambda L_{mask}, \tag{2}$$

where L_{cate} is the conventional Focal Loss [23] for semantic category classification, L_{mask} is the Dice Loss for mask prediction. For more details, we refer readers to [33].

During the inference, we forward input image through the backbone network and FPN, and obtain the category score $\mathbf{p}_{i,j}$ at grid (i, j). We first use a confidence threshold of 0.1 to filter out predictions with low confidence. The corresponding predicted mask kernels are then used to perform convolution on the mask feature. After the `sigmoid`

operation, we use a threshold of 0.5 to convert predicted soft masks to binary masks. The last step is the Matrix NMS.

4.2. Matrix NMS

Motivation Our Matrix NMS is motivated from Soft-NMS [1]. Soft-NMS decays the other detection scores as a monotonic decreasing function $f(\texttt{iou})$ of their overlaps. By decaying the scores according to IoUs recursively, higher IoU detections will be eliminated with a minimum score threshold. However, such process is sequential like traditional Greedy NMS and could not be implemented in parallel.

Matrix NMS views this process from another perspective by considering how a predicted mask m_j being suppressed. For m_j, its decay factor is affected by: (a) The penalty of each prediction m_i on m_j ($s_i > s_j$); and (b) the probability of m_i being suppressed. For (a), the penalty of each prediction m_i on m_j could be easily computed by $f(\texttt{iou}_{i,j})$. For (b), the probability of m_i being suppressed is not so elegant to be computed. However, the probability usually has positive correlation with the IoUs. So here we directly approximate the probability by the most overlapped prediction on m_i as

$$f(iou_{\cdot,i}) = \min_{\forall s_k > s_i} f(\texttt{iou}_{k,i}). \tag{3}$$

To this end, the final decay factor becomes

$$decay_j = \min_{\forall s_i > s_j} \frac{f(\texttt{iou}_{i,j})}{f(\texttt{iou}_{\cdot,i})}, \tag{4}$$

and the updated score is computed by $s_j = s_j \cdot decay_j$.

We consider two most simple decremented functions, denoted as `linear`

$$f(\texttt{iou}_{i,j}) = 1 - \texttt{iou}_{i,j}, \tag{5}$$

and `Gaussian`

$$f(\texttt{iou}_{i,j}) = \exp\left(-\frac{\texttt{iou}_{i,j}^2}{\sigma}\right). \tag{6}$$

Implementation All the operations in Matrix NMS could be implemented in one shot without recurrence. We first compute a $N \times N$ pairwise IoU matrix for the top N predictions sorted descending by score. For binary masks, the IoU matrix could be efficiently implemented by matrix operations. Then we get the most overlapping IoUs by column-wise max on the IoU matrix. Next, the decay factors of all higher scoring predictions are computed, and the decay factor for each prediction is selected as the most effect one by column-wise min (Eqn. (4)). Finally, the scores are updated by the decay factor. For usage, we just need threshing and selecting top-k scoring masks as the final predictions.

Figure 4 shows the pseudo-code of Matrix NMS in Pytorch style. In our code base, Matrix NMS is $9\times$ times faster than traditional NMS and being more accurate (Table 7). We show that Matrix NMS serves as a superior alternative of traditional NMS both in accuracy and speed, and can be easily integrated into the state-of-the-art detection/segmentation systems.

```
def matrix_nms(scores, masks, method='gauss', sigma=0.5):
    # scores: mask scores in descending order (N)
    # masks: binary masks (NxHxW)
    # method: 'linear' or 'gauss'
    # sigma: std in gaussian method

    # reshape for computation: Nx(HW)
    masks = masks.reshape(N, HxW)
    # pre-compute the IoU matrix: NxN
    intersection = mm(masks, masks.T)
    areas = masks.sum(dim=1).expand(N, N)
    union = areas + areas.T - intersection
    ious = (intersection / union).triu(diagonal=1)

    # max IoU for each: NxN
    ious_cmax = ious.max(0)
    ious_cmax = ious_cmax.expand(N, N).T
    # Matrix NMS, Eqn.(4): NxN
    if method == 'gauss': # gaussian
        decay = exp(-(ious^2 - ious_cmax^2) / sigma)
    else: # linear
        decay = (1 - ious) / (1 - ious_cmax)
    # decay factor: N
    decay = decay.min(dim=0)
    return scores * decay
```

Figure 4 – Python code of Matrix NMS. `mm`: matrix multiplication; `T`: transpose; `triu`: upper triangular part

5. Experiments

To evaluate the proposed method SOLOv2, we conduct experiments on three basic tasks, instance segmentation, object detection and panoptic segmentation on MS COCO [22]. We also present experimental results on the recently proposed LVIS dataset, which has more than 1K categories and thus is considerably more challenging.

5.1. Instance segmentation

For instance segmentation, we report lesion and sensitivity studies by evaluating on the COCO 5K `val2017` split. We also report COCO mask AP on the `test-dev` split, which is evaluated on the evaluation server.

Training details. SOLOv2 is trained with stochastic gradient descent (SGD). We use synchronized SGD over 8 GPUs with a total of 16 images per mini-batch. Unless otherwise specified, all models are trained for 36 epochs (i.e., $3\times$) with an initial learning rate of 0.01, which is then divided by 10 at 27th and again at 33th epoch. Weight decay of 0.0001 and momentum of 0.9 are used. All models are initialized from ImageNet pre-trained weights. We use scale jitter where the shorter image side is randomly sampled from 640 to 800 pixels.

5.1.1 Main Results

We compare SOLOv2 to the state-of-the-art methods in instance segmentation on MS COCO `test-dev` in Table 1. SOLOv2 with ResNet-101 achieves a mask AP of 39.7%, which is much better than SOLOv1 and other state-of-the-art instance segmentation methods. Our method shows its superiority espically on large objects (*e.g.* +5.0 AP_L than Mask R-CNN).

We also provide the speed-accuracy trade-off on COCO to compare with some dominant instance segmenters (Figure 1). We show our models with ResNet-50, ResNet-101, ResNet-DCN-101 and two light-weight versions described in Section 5.1.3. The proposed SOLOv2 outperforms a range of state-of-the-art algorithms, both in accuracy and speed. The running time is tested on our local machine, with a single V100 GPU, Pytorch 1.2 and CUDA 10.0. We download code and pre-trained models to test inference time for each model on the same machine.

5.1.2 SOLOv2 Visualization

We visualize what SOLOv2 learns from two aspects: mask feature behavior and the final outputs after being convolved by the dynamically learned convolution kernels.

We visualize the outputs of mask feature branch. We use a model which has 64 output channels (*i.e.*, $E = 64$ for the last feature map prior to mask prediction) for easy visualization. Here we plot each of the 64 channels (recall the channel spatial resolution is $H \times W$) as shown in Figure 5.

There are two main patterns. The first and the foremost, the mask features are position-aware. It shows obvious behavior of scanning the objects in the image horizontally and vertically. Interestingly, it is indeed in accordance to the target in the decoupled-head SOLO: Segmenting objects by their independent horizontal and vertical location categories. The other obvious pattern is that some feature maps are responsible for activating all the foreground objects, *e.g.*, the one in white boxes.

The final outputs are shown in Figure 8. Different objects are in different colors. Our method shows promising results in diverse scenes. It is worth pointing out that the details at the boundaries are segmented well, especially for large objects. We compare against Mask R-CNN on object details in Figure 6. Our method shows great advantages.

5.1.3 Ablation Experiments

We investigate and compare the following four aspects in our methods: (a) the kernel shape used to perform convolution on mask features; (b) CoordConvs used in the mask kernel branch and mask feature branch; (c) unified mask feature representation and (d) the effectiveness of Matrix NMS.

Kernel shape We consider the kernel shape from two aspects: number of input channels and kernel size. The comparisons are shown in Table 3. 1×1 conv shows equivalent performance to 3×3 conv. Changing the number of input channels from 128 to 256 attains 0.4% AP gains. When it grows beyond 256, the performance becomes stable. In this work, we set the number of input channels to be 256 in all other experiments.

Effectiveness of coordinates Since our method segments objects by locations, or specifically, learns the object segmenters by locations, the position information is very important. For example, if the mask kernel branch is unaware of the positions, the objects with the same appearance may have the same predicted kernel, leading to the same output mask. On the other hand, if the mask feature branch is unaware of the position information, it would not know how to assign the pixels to different feature channels in the order that matches the mask kernel. As shown in Table 4, the model achieves 36.3% AP without explicit coordinates input. The results are reasonably good because that CNNs can implicitly learn the absolute position information from the commonly used zero-padding operation, as revealed in [14]. The pyramid zero-paddings in our mask feature branch should have contributed considerably. However, the implicitly learned position information is coarse and inaccurate. When making the convolution access to its own input coordinates through concatenating extra coordinate channels, our method enjoys 1.5% absolute AP gains.

Unified Mask Feature Representation For mask feature learning, we have two options: to learn the feature in the head separately for each FPN level or to construct a unified representation. For the former one, we implement as SOLOv1 and use seven 3×3 conv to predict the mask features. For the latter one, the detailed implementation is illustrated in Figure 3. We compare these two implementations in Table 5. As shown, the unified representation achieves better results, especially for the medium and large objects. This is easy to understand: in SOLOv1 large-size objects are assigned to high-level feature maps of low spatial resolutions, leading to coarse boundary prediction.

Dynamic vs. Decoupled The dynamic head and decoupled head both serve as the efficient varieties of the SOLO head. We compare the results in Table 6. All the settings are the same except the head type, which means that for the dynamic head we use the separate features as above. The dynamic head achieves 0.7% AP better than the decoupled head. We believe that the gains have come from the dynamic scheme which learns the kernel weights dynamically, conditioned on the input.

Matrix NMS Our Matrix NMS can be implemented totally in parallel. Table 7 presents the speed and accuracy compar-

	backbone	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
box-based:							
Mask R-CNN [12]	Res-101-FPN	35.7	58.0	37.8	15.5	38.1	52.4
Mask R-CNN*	Res-101-FPN	37.8	59.8	40.7	**20.5**	40.4	49.3
MaskLab+ [5]	Res-101-C4	37.3	59.8	39.6	16.9	39.9	53.5
TensorMask [6]	Res-50-FPN	35.4	57.2	37.3	16.3	36.8	49.3
TensorMask [6]	Res-101-FPN	37.1	59.3	39.4	17.4	39.1	51.6
YOLACT [2]	Res-101-FPN	31.2	50.6	32.8	12.1	33.3	47.1
box-free:							
PolarMask [35]	Res-101-FPN	30.4	51.9	31.0	13.4	32.4	42.8
SOLOv1 [33]	Res-101-FPN	37.8	59.5	40.4	16.4	40.6	54.2
SOLOv2	Res-50-FPN	38.8	59.9	41.7	16.5	41.7	56.2
SOLOv2	Res-101-FPN	39.7	60.7	42.9	17.3	42.9	57.4
SOLOv2	Res-DCN-101-FPN	**41.7**	**63.2**	**45.1**	18.0	**45.0**	**61.6**

Table 1 – Instance segmentation mask AP (%) on COCO `test-dev`. All entries are *single-model* results. Mask R-CNN* is our improved version with scale augmentation and longer training time (6×). 'DCN' means deformable convolutions used.

	backbone	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
YOLOv3 [30]	DarkNet53	33.0	57.9	34.4	18.3	35.4	41.9
SSD513 [28]	ResNet-101	31.2	50.4	33.3	10.2	34.5	49.8
DSSD513 [28]	ResNet-101	33.2	53.3	35.2	13.0	35.4	51.1
RefineDet [39]	ResNet-101	36.4	57.5	39.5	16.6	39.9	51.4
Faster R-CNN [21]	Res-101-FPN	36.2	59.1	39.0	18.2	39.0	48.2
RetinaNet [23]	Res-101-FPN	39.1	59.1	42.3	21.8	42.7	50.2
FoveaBox [18]	Res-101-FPN	40.6	60.1	43.5	23.3	45.2	54.5
RPDet [37]	Res-101-FPN	41.0	62.9	44.3	23.6	44.1	51.7
FCOS [32]	Res-101-FPN	41.5	60.7	45.0	**24.4**	44.8	51.6
CenterNet [40]	Hourglass-104	42.1	61.1	45.9	24.1	45.5	52.8
SOLOv2	Res-50-FPN	40.4	59.8	42.8	20.5	44.2	53.9
SOLOv2	Res-101-FPN	42.6	61.2	45.6	22.3	46.7	56.3
SOLOv2	Res-DCN-101-FPN	**44.9**	**63.8**	**48.2**	23.1	**48.9**	**61.2**

Table 2 – Object detection box AP (%) on the COCO `test-dev`. Although our bounding boxes are directly generated from the predicted masks, the accuracy outperforms most state-of-the-art methods. Speed-accuracy trade-off of typical methods is shown in Figure 7.

Kernel shape	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
$3 \times 3 \times 64$	37.4	58.0	39.9	15.6	40.8	**56.9**
$1 \times 1 \times 64$	37.4	58.1	40.1	15.5	41.1	56.3
$1 \times 1 \times 128$	37.4	58.1	40.2	**15.8**	41.1	56.6
$1 \times 1 \times 256$	**37.8**	**58.5**	**40.4**	15.6	41.3	56.8
$1 \times 1 \times 512$	37.7	58.3	**40.4**	15.4	**41.5**	56.6

Table 3 – Kernel shape. We use kernel shape of $1 \times 1 \times 256$ in other experiments, as the performance becomes stable when the shape goes beyond that.

Kernel	Feature	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
		36.3	57.4	38.6	**15.6**	39.8	54.7
✓		36.3	57.3	38.5	15.1	40.0	54.1
	✓	37.1	58.0	39.4	15.2	40.5	55.9
✓	✓	**37.8**	**58.5**	**40.4**	**15.6**	**41.3**	**56.8**

Table 4 – Explicit coordinates. Precise coordinates input can considerably improve the results.

Mask Feature	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Separate	37.3	58.2	40.0	**15.7**	40.8	55.5
Unified	**37.8**	**58.5**	**40.4**	15.6	**41.3**	**56.8**

Table 5 – Mask feature representation. We compare the separate mask feature representation outputted by 7 convs in parallel head and the unified representation from Figure 3.

ison of Hard-NMS, Soft-NMS, Fast NMS and our Matrix NMS. Since all methods need to compute the IoU matrix, we pre-compute the IoU matrix in advance for fair comparison. The speed reported here is that of the NMS process alone, excluding computing IoU matrices.

Hard NMS and Soft-NMS are widely used in current object detection and segmentation models. Unfortunately, both methods are recursive and spend much time budget (22 ms). Our Matrix NMS only needs <1 ms and is almost cost free! Here we also show the performance of

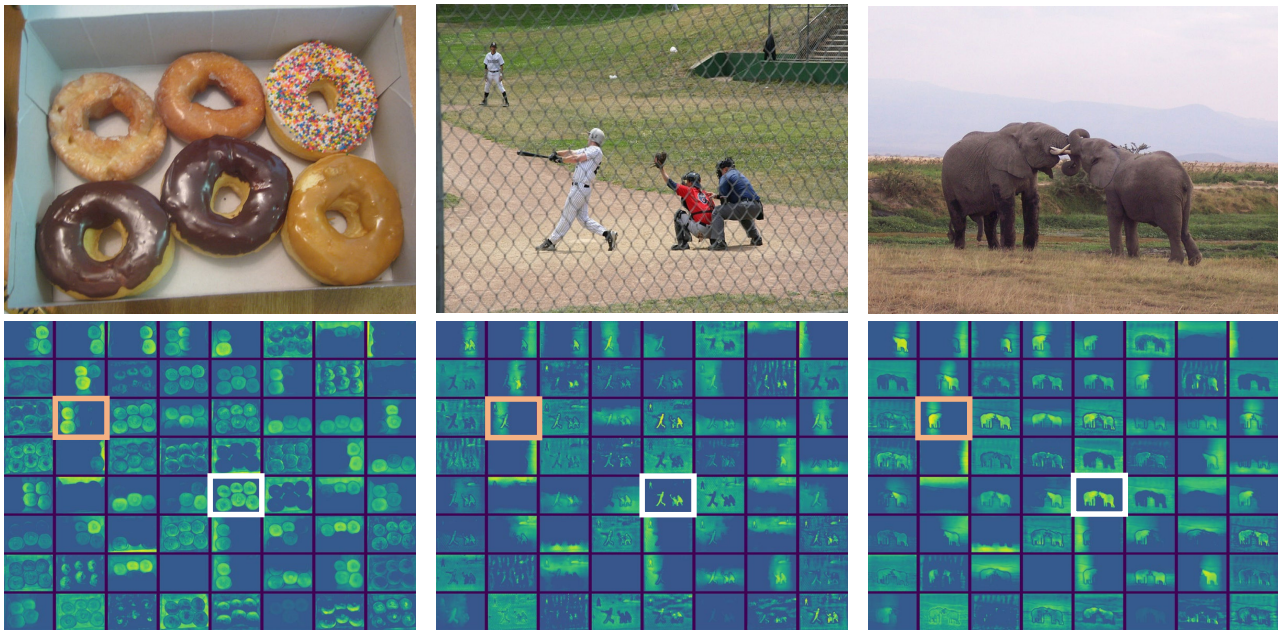


Figure 5 – SOLOv2 **Mask Feature Behavior.** Each plotted subfigure corresponds to one of the 64 channels of the last feature map prior to mask prediction. The mask features appear to be position-sensitive (orange box), while a few mask features are position-agnostic and activated on all instances (white box). Best viewed on screens.

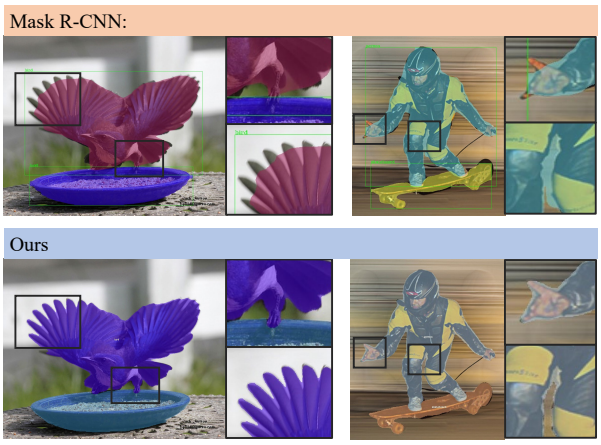


Figure 6 – **Comparison of mask details.** We compare our model (39.7% AP) with Mask R-CNN (37.8% AP) on predicted object boundary details. Mask-RCNN's mask head is typically restricted to 28×28 resolution, leading to inferior prediction at object boundaries.

Head type	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Decoupled	36.6	57.2	39.0	**16.0**	40.3	53.5
Dynamic*	**37.3**	**58.2**	**40.0**	15.7	**40.8**	**55.5**

Table 6 – Dynamic head vs. Decoupled head. The dynamic head here is applied on separate features outputted by 7 convs in parallel.

Method	Iterative?	Time (ms)	AP
Hard-NMS	✓	9	36.3
Soft-NMS	✓	22	36.5
Fast NMS	✗	<1	36.2
Our Matrix NMS	✗	<1	**36.6**

Table 7 – Matrix NMS. Our Matrix NMS outperforms the other methods in both speed and accuracy.

Fast NMS, which utilizes matrix operations but with performance penalty. To conclude, our Matrix NMS shows its advantages on both speed and accuracy.

Real-time setting We design two light-weight models for different purposes. 1) `Speed priority`, the number of convolution layers in the prediction head is reduced to two and the input shorter side is 448. 2) `Accuracy priority`, the number of convolution layers in the prediction head is reduced to three and the input shorter side is 512. Moreover, deformable convolution [8] is used in the backbone and the last layer of prediction head. We train both models with the $3\times$ schedule, with shorter side randomly sampled from [352, 512]. Results are shown in Table 8. SOLO can not only push state-of-the-art, but has also been ready for real-time applications.

Model	backbone	AP	AP_{50}	AP_{75}	fps
SOLO-448	ResNet-50-FPN	34.0	54.0	36.1	46.5
SOLO-512	ResNet-50-FPN	37.1	57.7	39.7	31.3

Table 8 – Real-time SOLO. The speed is reported on a single V100 GPU by averaging 5 runs. All models are evaluated on COCO `test-dev`.

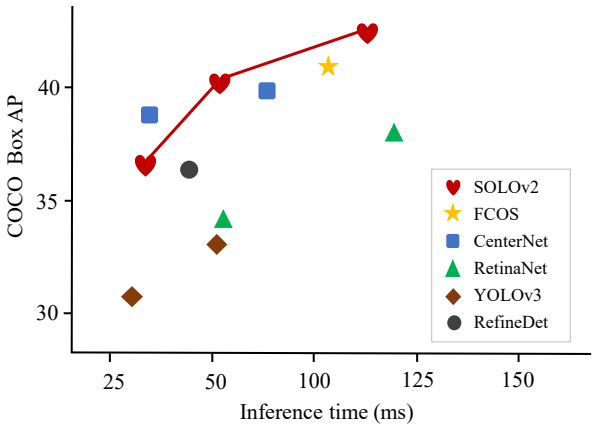


Figure 7 – Speed-accuracy trade-off of bounding-box object detection on the COCO `test-dev`. Inference time of all methods is evaluated on a TitanX-GPU machine.

5.2. Bounding-box Object Detection

Although our instance segmentation solution removes the dependence of bounding box prediction, we are able to produce the 4D object bounding box from each instance mask. In Table 2, we compare the generated box detection performance with other object detection methods on COCO. All models are trained on the `train2017` subset and tested on `test-dev`.

As shown in Table 2, our detection results outperform most methods, especially for objects of large scales, demonstrating the effectiveness of SOLOv2 in object box detection. Similar to instance segmentation, we also plot the speed/accuracy trade-off curve for different methods in Figure 7. We show our models with ResNet-101 and two light-weight versions described above. The plot reveals that the bounding box performance of SOLOv2 beats most recent object detection methods in both accuracy and speed. Here we emphasis that our results are directly generated from the off-the-shelf instance mask, without any box based supervised training or engineering.

An observation from Figure 7 is as follows. If one does not care much about the cost difference between mask annotation and bounding box annotation, it appears to us that there is no reason to use box detectors for downstream applications, considering the fact that our SOLOv2 beats most modern detectors in both accuracy and speed.

5.3. Panoptic Segmentation

The proposed SOLOv2 can be easily extended to panoptic segmentation by adding the semantic segmentation branch, analogue to the mask feature branch. We use annotations of COCO 2018 panoptic segmentaiton task. All models are trained on `train2017` subset and tested on `val2017`. We use the same strategy as in Panoptic-FPN to combine instance and semantic results. As shown in Table 10, our method achieves state-of-the-art results and outperforms other recent box-free methods by a large margin. All methods listed use the same backbone (ResNet50-FPN) except SSAP (ResNet101) and Panoptic-DeepLab (Xception-71).

5.4. Results on the LVIS dataset

LVIS [11] is a recently proposed dataset for long-tail object segmentation, which has more than 1000 object categories. In LVIS, each object instance is segmented with a high-quality mask that surpasses the annotation quality of the relevant COCO dataset. Since LVIS is new, only the results of Mask R-CNN are publicly available. Therefore we only compare SOLOv2 against the Mask R-CNN baseline.

Table 9 reports the performances on the rare (1$\sim$10 images), common (11$\sim$100), and frequent ($>$ 100) subsets, as well as the overall AP. Our SOLOv2 outperforms the baseline method by about 1% AP. For large-size objects (AP_L), our SOLOv2 achieves 6.7% AP improvement, which is consistent with the results on the COCO dataset.

6. Conclusion

In this work, we have significantly improved SOLOv1 instance segmentation from three aspects.

- We have proposed to learn adaptive, dynamic convolutional kernels for the mask prediction head, conditioned on the location, leading to a much more compact yet more powerful head design, and achieving better results with reduced FLOPs.
- We have re-designed the mask features as shown in Figure 3, which predicts more accurate boundaries. Especially for the medium-size and large-size objects, better mask APs are attained compared against SOLOv1.
- Moreover, unlike box NMS as in object detection, for instance segmentation a bottleneck in inference efficiency is the NMS of masks. Previous works either use box NMS as a surrogate, or speed it up via approximation which results in harming mask AP. We have designed a simple and much faster NMS strategy, termed Matrix NMS, for NMS processing of masks, without sacrificing mask AP.

	backbone	AP_r	AP_c	AP_f	AP_S	AP_M	AP_L	AP
Mask-RCNN [11]	Res-50-FPN	14.5	24.3	28.4	-	-	-	24.4
Mask-RCNN*-3×	Res-50-FPN	12.1	25.8	28.1	**18.7**	31.2	38.2	24.6
SOLOv2	Res-50-FPN	13.4	26.6	28.9	15.9	34.6	44.9	25.5
SOLOv2	Res-101-FPN	**16.3**	**27.6**	**30.1**	16.8	**35.8**	**47.0**	**26.8**

Table 9 – Instance segmentation results on the LVISv0.5 validation dataset.

	PQ	PQ^{Th}	PQ^{St}
box-based:			
AUNet [19]	39.6	49.1	25.2
UPSNet [36]	**42.5**	48.5	**33.4**
Panoptic-FPN [17]	39.0	45.9	28.7
Panoptic-FPN*-1×	38.7	45.9	27.8
Panoptic-FPN*-3×	40.8	48.3	29.4
box-free:			
AdaptIS [31]	35.9	40.3	29.3
SSAP (Res101) [10]	36.5	–	–
Pano-DeepLab (Xcept71) [7]	39.7	43.9	33.2
SOLOv2	42.1	**49.6**	30.7

Table 10 – Panoptic results on COCO val2017. Here Panoptic-FPN* is our re-implemented version in `mmdetection` [4] with 12 and 36 training epochs (1× and 3×) and multi-scale training. All model's backbones are Res50, except SSAP and Pano-DeepLab. Note that UPSNet has used deformable convolution [8] for better performance,

Our experiments on the Microsoft COCO and LVIS datasets demonstrate the superior performance in terms of both accuracy and speed of the proposed SOLOv2. Being versatile for instance-level recognition tasks, we show that without any modification to the framework, SOLOv2 performs competitively for panoptic segmentation. Thanks to its simplicity (being proposal free, anchor free, FCN-like), strong performance in both accuracy and speed, and potentially being capable of solve many instance-level tasks, we hope that SOLOv2 can be a strong baseline approach to instance recognition, and inspires future work such that its full potential can be exploited as we believe that there is still much room for improvement.

References

[1] Navaneeth Bodla, Bharat Singh, Rama Chellappa, and Larry Davis. Soft-NMS: improving object detection with one line of code. In *Proc. IEEE Int. Conf. Comp. Vis.*, 2017.

[2] Daniel Bolya, Chong Zhou, Fanyi Xiao, and Yong Jae Lee. YOLACT: Real-time instance segmentation. In *Proc. IEEE Int. Conf. Comp. Vis.*, 2019.

[3] Hao Chen, Kunyang Sun, Zhi Tian, Chunhua Shen, Yongming Huang, and Youliang Yan. BlendMask: Top-down meets bottom-up for instance segmentation. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2020.

[4] Kai Chen, Jiaqi Wang, Jiangmiao Pang, Yuhang Cao, Yu Xiong, Xiaoxiao Li, Shuyang Sun, Wansen Feng, Ziwei Liu, Jiarui Xu, Zheng Zhang, Dazhi Cheng, Chenchen Zhu, Tianheng Cheng, Qijie Zhao, Buyu Li, Xin Lu, Rui Zhu, Yue Wu, Jifeng Dai, Jingdong Wang, Jianping Shi, Wanli Ouyang, Chen Change Loy, and Dahua Lin. MMDetection: Open mmlab detection toolbox and benchmark. *arXiv:1906.07155*, 2019.

[5] Liang-Chieh Chen, Alexander Hermans, George Papandreou, Florian Schroff, Peng Wang, and Hartwig Adam. Masklab: Instance segmentation by refining object detection with semantic and direction features. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2018.

[6] Xinlei Chen, Ross Girshick, Kaiming He, and Piotr Dollar. TensorMask: A foundation for dense object segmentation. In *Proc. IEEE Int. Conf. Comp. Vis.*, 2019.

[7] Bowen Cheng, Maxwell D Collins, Yukun Zhu, Ting Liu, Thomas S Huang, Hartwig Adam, and Liang-Chieh Chen. Panoptic-deeplab. *arXiv:1910.04751*, 2019.

[8] Jifeng Dai, Haozhi Qi, Yuwen Xiong, Yi Li, Guodong Zhang, Han Hu, and Yichen Wei. Deformable convolutional networks. In *Proc. IEEE Int. Conf. Comp. Vis.*, 2017.

[9] Bert De Brabandere, Davy Neven, and Luc Van Gool. Semantic instance segmentation with a discriminative loss function. *arXiv:1708.02551*, 2017.

[10] Naiyu Gao, Yanhu Shan, Yupei Wang, Xin Zhao, Yinan Yu, Ming Yang, and Kaiqi Huang. SSAP: Single-shot instance segmentation with affinity pyramid. In *Proc. IEEE Int. Conf. Comp. Vis.*, 2019.

[11] Agrim Gupta, Piotr Dollar, and Ross Girshick. LVIS: A dataset for large vocabulary instance segmentation. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2019.

[12] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross B. Girshick. Mask R-CNN. In *Proc. IEEE Int. Conf. Comp. Vis.*, 2017.

[13] Zhaojin Huang, Lichao Huang, Yongchao Gong, Chang Huang, and Xinggang Wang. Mask scoring R-CNN. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2019.

[14] Md Amirul Islam, Sen Jia, and Neil D. B. Bruce. How much position information do convolutional neural networks encode? In *Proc. Int. Conf. Learn. Representations*, 2020.

[15] Max Jaderberg, Karen Simonyan, Andrew Zisserman, and Koray Kavukcuoglu. Spatial transformer networks. In *Proc. Advances in Neural Inf. Process. Syst.*, 2015.

[16] Xu Jia, Bert De Brabandere, Tinne Tuytelaars, and Luc V Gool. Dynamic filter networks. In *Proc. Advances in Neural Inf. Process. Syst.*, 2016.

[17] Alexander Kirillov, Ross Girshick, Kaiming He, and Piotr Dollár. Panoptic feature pyramid networks. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2019.

[18] Tao Kong, Fuchun Sun, Huaping Liu, Yuning Jiang, and Jianbo Shi. Foveabox: Beyond anchor-based object detector. *arXiv:1904.03797*, 2019.

[19] Yanwei Li, Xinze Chen, Zheng Zhu, Lingxi Xie, Guan Huang, Dalong Du, and Xingang Wang. Attention-guided unified network for panoptic segmentation. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2019.

[20] Yi Li, Haozhi Qi, Jifeng Dai, Xiangyang Ji, and Yichen Wei. Fully convolutional instance-aware semantic segmentation. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2017.

[21] Tsung-Yi Lin, Piotr Dollár, Ross B. Girshick, Kaiming He, Bharath Hariharan, and Serge J. Belongie. Feature pyramid networks for object detection. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2017.

[22] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: common objects in context. In *Proc. Eur. Conf. Comp. Vis.*, 2014.

[23] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proc. IEEE Int. Conf. Comp. Vis.*, 2017.

[24] Rosanne Liu, Joel Lehman, Piero Molino, Felipe Petroski Such, Eric Frank, Alex Sergeev, and Jason Yosinski. An intriguing failing of convolutional neural networks and the coordconv solution. In *Proc. Advances in Neural Inf. Process. Syst.*, 2018.

[25] Songtao Liu, Di Huang, and Yunhong Wang. Adaptive NMS: Refining pedestrian detection in a crowd. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2019.

[26] Shu Liu, Jiaya Jia, Sanja Fidler, and Raquel Urtasun. Sequential grouping networks for instance segmentation. In *Proc. IEEE Int. Conf. Comp. Vis.*, 2017.

[27] Shu Liu, Lu Qi, Haifang Qin, Jianping Shi, and Jiaya Jia. Path aggregation network for instance segmentation. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2018.

[28] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *Proc. Eur. Conf. Comp. Vis.*, 2016.

[29] Alejandro Newell, Zhiao Huang, and Jia Deng. Associative embedding: End-to-end learning for joint detection and grouping. In *Proc. Advances in Neural Inf. Process. Syst.*, 2017.

[30] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv:1804.02767*, 2018.

[31] Konstantin Sofiiuk, Olga Barinova, and Anton Konushin. AdaptIS: Adaptive instance selection network. In *Proc. IEEE Int. Conf. Comp. Vis.*, 2019.

[32] Zhi Tian, Chunhua Shen, Hao Chen, and Tong He. FCOS: Fully convolutional one-stage object detection. In *Proc. IEEE Int. Conf. Comp. Vis.*, 2019.

[33] Xinlong Wang, Tao Kong, Chunhua Shen, Yuning Jiang, and Lei Li. SOLO: Segmenting objects by locations. *arXiv preprint arXiv:1912.04488*, 2019.

[34] Yuxin Wu and Kaiming He. Group normalization. In *Proc. Eur. Conf. Comp. Vis.*, 2018.

[35] Enze Xie, Peize Sun, Xiaoge Song, Wenhai Wang, Xuebo Liu, Ding Liang, Chunhua Shen, and Ping Luo. PolarMask: Single shot instance segmentation with polar representation. 2020.

[36] Yuwen Xiong, Renjie Liao, Hengshuang Zhao, Rui Hu, Min Bai, Ersin Yumer, and Raquel Urtasun. UPSNet: A unified panoptic segmentation network. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2019.

[37] Ze Yang, Shaohui Liu, Han Hu, Liwei Wang, and Stephen Lin. Reppoints: Point set representation for object detection. In *Proc. IEEE Int. Conf. Comp. Vis.*, 2019.

[38] Rufeng Zhang, Zhi Tian, Chunhua Shen, Mingyu You, and Youliang Yan. Mask encoding for single shot instance segmentation. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2020.

[39] Shifeng Zhang, Longyin Wen, Xiao Bian, Zhen Lei, and Stan Z Li. Single-shot refinement neural network for object detection. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2018.

[40] Xingyi Zhou, Dequan Wang, and Philipp Krähenbühl. Objects as points. *arXiv:1904.07850*, 2019.

Acknowledgements and declaration of conflicting interests We would like to thank ByteDance AI Lab for the use of GPU computing resources. Chunhua Shen and his employer received no financial support for the research, authorship, and/or publication of this article.

Figure 8 – **Visualization of instance segmentation results** using the Res-101-FPN backbone. The model is trained on the COCO `train2017` dataset, achieving a mask AP of 39.7% on the COCO `test-dev`.