SOLO: A Simple Framework for Instance Segmentation

Xinlong Wang, Rufeng Zhang, Chunhua Shen, Tao Kong, and Lei Li

Abstract—Compared to many other dense prediction tasks, e.g., semantic segmentation, it is the arbitrary number of instances that has made instance segmentation much more challenging. In order to predict a mask for each instance, mainstream approaches either follow the "detect-then-segment" strategy (e.g., Mask R-CNN), or predict embedding vectors first then cluster pixels into individual instances. In this paper, we view the task of instance segmentation from a completely new perspective by introducing the notion of "instance categories", which assigns categories to each pixel within an instance according to the instance's location. With this notion, we propose **s**egmenting **o**bjects by **lo**cations (SOLO), a simple, direct, and fast framework for instance segmentation with strong performance. We derive a few SOLO variants (e.g., Vanilla SOLO, Decoupled SOLO, Dynamic SOLO) following the basic principle. Our method directly maps a raw input image to the desired object categories and instance masks, eliminating the need for the grouping post-processing or the bounding box detection. Our approach achieves state-of-the-art results for instance segmentation in terms of both speed and accuracy, while being considerably simpler than the existing methods. Besides instance segmentation, our method yields state-of-the-art results in object detection (from our mask byproduct) and panoptic segmentation. We further demonstrate the flexibility and high-quality segmentation of SOLO by extending it to perform one-stage instance-level image matting. Code is available at: https://git.io/AdelaiDet

Index Terms—Instance segmentation, object detection, segmenting objects by locations

1 Introduction

The goal of general object detection is to localize individual objects and recognize their categories. Bounding box stands out for its simplicity for representing the object location. Localizing objects using bounding boxes have been extensively explored, including the problem formulation [1], [2], [3], [4], [5], network architecture [6], [7], [8], post-processing [9], [10], [11] and all those focusing on optimizing and processing the bounding boxes [12], [13], [14]. 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 irregular 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.

Instance segmentation is challenging because it requires the correct separation of all objects in an image while also semantically segmenting each instance at the pixel level. To tackle this problem, recent methods can be categorized into two groups, i.e., top-down [15], [16], [17], [18] and bottom-up [19], [20], [21], [22] paradigms. The former approach, namely "detect-then-segment", first detects bounding boxes and then segments the instance mask in each bounding box. The latter approach learns an affinity relation, assigning an embedding vector to each pixel, by pushing away pixels belonging to different instances and pulling close pixels in the same instance. A grouping post-processing is then needed to separate instances. Both these two paradigms are step-wise and indirect, which either heavily rely on accurate bounding box detection or depend on per-pixel embedding learning and the grouping processing.

In contrast, we aim to directly segment instance masks, under the supervision of full instance mask annotations instead of masks in boxes or additional pixel pairwise relations. We start by rethinking a question: *What are the fundamental differences between object instances in an image?* Take the challenging MS COCO dataset [23] for example. There are in total 36,780 objects in the validation subset, 98.3% of object pairs have center distance greater than 30 pixels. As for the rest 1.7% of object pairs, 40.5% of them have size ratio greater than $1.5\times$. To conclude, in most cases, two instances in an image either have different center locations or have different object sizes. This observation makes one wonder whether we could directly distinguish instances by the center locations and object sizes?

In the closely related field, semantic segmentation, now the dominated paradigm, leverages a fully convolutional network (FCN) to output dense predictions with N channels. Each output channel is responsible for one of the semantic categories (including background). Semantic segmentation

- *Xinlong Wang is with The University of Adelaide, Adelaide, SA 5005, Australia. E-mail: wangxinlon@gmail.com.*
- *Rufeng Zhang is with Tongji University, Shanghai 200092, China. E-mail: cxrfzhang@tongji.edu.cn.*
- *Chunhua Shen is with The University of Adelaide, Adelaide, SA 5005, Australia, and also with Monash University, Clayton, VIC 3800, Australia. E-mail: chhshen@gmail.com.*
- *Tao Kong and Lei Li are with ByteDance AI Lab, Beijing 100080, China. E-mail: taokongcn@gmail.com, lileilab@bytedance.com.*

Manuscript received 8 Feb. 2021; revised 30 June 2021; accepted 26 Aug. 2021. Date of publication 13 Sept. 2021; date of current version 3 Oct. 2022.
(Corresponding authors: Chunhua Shen and Tao Kong.)
Recommended for acceptance by B. Han.
Digital Object Identifier no. 10.1109/TPAMI.2021.3111116

aims to distinguish different semantic categories. Analogously, in this work, we propose to distinguish object instances in the image by introducing the notion of "*instance categories*", i.e., the quantized center locations and object sizes, which enables to **s**egment **o**bjects by **lo**cations, thus the name of our framework, SOLO.

For center locations, an image can be divided into a grid of $S \times S$ cells, thus leading to S^2 center location classes. According to the coordinates of the object center, an object instance is assigned to one of the grid cells, as its center location category. Note that grids are used conceptually to assign a location category for each pixel. Each output channel is responsible for one of the center location categories, and the corresponding channel map should predict the instance mask of the object belonging to that location. Thus, structural geometric information is naturally preserved in the spatial matrix with dimensions of height by width. Unlike DeepMask [24] and TensorMask [18], which run in a dense sliding-window manner and segment an object in a fixed local patch, our method naturally outputs accurate masks for all scales of instances without the limitation of (anchor) box locations and scales. In essence, an instance location category approximates the location of the object center of an instance. Thus, by classification of each pixel into its instance location category, it is equivalent to predict the object center of each pixel in the latent space. The importance here of converting the location prediction task into classification is that, with classification it is much more straightforward and easier to model a varying number of instances using a fixed number of channels, at the same time not relying on post-processing like grouping or learning embeddings. For object sizes, we employ the feature pyramid network (FPN) [6] to distinguish instances with different object sizes, so as to assign objects of different sizes to different levels of feature maps. Note that FPN was designed for the purposes of detecting objects of different sizes in an image. Thus, all the object instances are separated regularly, enabling to classify objects by "instance categories".

With the proposed SOLO framework, we are able to optimize the network in an end-to-end fashion for the instance segmentation task and perform pixel-level instance segmentation out of the restrictions of local box detection and pixel grouping. To fully exploit the capabilities of this simple framework, we propose three different SOLO variants following the basic principle, namely vanilla SOLO, Decoupled SOLO and Dynamic SOLO (SOLOv2).

Besides the problem formulation, the supporting facilities of instance segmentation, e.g., post-processing, is largely unexplored compared to bounding box detection. For developing a fast and pure instance segmentation framework, we 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. Such sequential and recursive operations result in non-negligible latency. For mask NMS, this drawback is further magnified. Compared to bounding box, it consumes more time to compute the IoU of each mask pair, thus leading to huge 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 its varieties in both accuracy and speed. As a result, *Matrix NMS processes 500 masks in less than 1 ms in simple python implementation*, and outperforms the recently proposed Fast NMS [25] by 0.4% AP.

To summarize, our simple framework outperforms the state-of-the-art instance segmentation methods in both speed and accuracy. Our model with ResNet-50 backbone 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 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, yields 44.9% AP for object detection, which even surpasses many state-of-the-art, highly-engineered object detection methods. By adding the semantic segmentation branch, we easily extend our method to solve panoptic segmentation and achieve state-of-the-art results. We also extend our framework to perform image matting at instance-level. Thanks to the ability of generating high-quality object masks, our method is able to solve instance-level image matting in one shot with minimal modifications.

We believe that, with our simple, fast and sufficiently strong solution, instance segmentation can be a popular alternative to the widely used object bounding box detection, and SOLO may play an important role and predict its wide applications.

2 Related Work

We review some works that are closest to ours.

Top-Down Instance Segmentation. The methods that segment object instances in a priori bounding box fall into the typical top-down paradigm. FCIS [15] assembles the position-sensitive score maps within the region-of-interests (ROIs) generated by a region proposal network (RPN) to predict instance masks. Mask R-CNN [16] extends the Faster R-CNN detector [1] by adding a branch for segmenting the object instances within the detected bounding boxes. Based on Mask R-CNN, PANet [26] further enhances the feature representation to improve the accuracy, Mask Scoring R-CNN [27] adds a mask-IoU branch to predict the quality of the predicted mask for improving the performance. HTC [17] interweaves box and mask branches for a joint multi-stage processing. TensorMask [18] adopts the dense sliding window paradigm to segment the instance in the local window for each pixel with a predefined number of windows and scales. YOLACT [25] 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 uses the predicted boxes to crop the assembled masks. Our Dynamic SOLO directly decouples the original mask prediction into kernel learning and feature learning. No anchor box is needed. No normalization is needed. No bounding box detection is needed. Both the training and inference are much simpler. In contrast to the top-down methods above, our SOLO framework is totally

box-free thus not being restricted by (anchor) box locations and scales, and naturally benefits from the inherent advantages of FCNs.

Bottom-Up Instance Segmentation. This category of the approaches generate instance masks by grouping the pixels into an arbitrary number of object instances presented in an image. In [19], pixels are grouped into instances using the learned associative embedding. A discriminative loss function [20] learns pixel-level instance embedding efficiently, by pushing away pixels belonging to different instances and pulling close pixels in the same instance. SGN [21] decomposes the instance segmentation problem into a sequence of sub-grouping problems. SSAP [22] learns a pixel-pair affinity pyramid, the probability that two pixels belong to the same instance, and sequentially generates instances by a cascaded graph partition. Typically, bottom-up methods lag behind in accuracy compared to top-down methods, especially on the dataset with diverse scenes. Instead of exploiting pixel pairwise relations and pixel grouping, our method directly learns with the instance mask annotations solely during training, and predicts instance masks end-to-end without grouping post-processing.

Direct Instance Segmentation. To our knowledge, no prior methods directly train with mask annotations solely, and predict instance masks and semantic categories in one shot without the need of grouping post-processing. Several recently proposed methods may be viewed as the 'semi-direct' paradigm. AdaptIS [28] first predicts point proposals, and then sequentially generates the mask for the object located at the detected point proposal. PolarMask [29] proposes to use the polar representation to encode masks and transforms per-pixel mask prediction to distance regression. They both do not need bounding boxes for training but are either being step-wise or founded on compromise, e.g., coarse parametric representation of masks. Our SOLO framework takes an image as input, directly outputs instance masks and corresponding class probabilities, in a fully convolutional, box-free and grouping-free paradigm.

Dynamic Convolutions. In traditional convolution layers, the learned convolution kernels stay fixed and are independent on the input, i.e., the weights are the same for arbitrary image and any location of the image. Some previous works [30], [31], [32] explore the idea of bringing more flexibility into the traditional convolutions. Dynamic filter [31] 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 [32] dynamically learn the sampling locations by predicting the offsets for each image location. Pixel-adaptive convolution [33] multiplies the weights of the filters and a spatially varying kernel to make the standard convolution content-adaptive. In the enhanced version SOLOv2, we bring the dynamic scheme into instance segmentation and enable learning instance segmenters by locations. Note that the work in [34] also applies dynamic convolutions for instance segmentation by extending the framework of BlendMask [35]. The dynamic scheme part is somewhat similar, but the methodology is different. CondInst [34] relies on the relative position to distinguish instances as in AdaptIS, while SOLOv2 uses absolute positions as in SOLO. It means that the former needs to encode the position information N times for N instances, while SOLOv2 performs it all at once using the global coordinates, regardless how many instances there are.

Non-Maximum Suppression. NMS is widely adopted in many computer vision tasks and becomes an essential component of object detection and instance segmentation systems. Some recent works [9], [10], [11], [25], [36] 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 [9] decreases the confidence scores of neighbors according to their overlap with higher scored predictions. In [10], the authors use KL-Divergence and reflected it in the refinement of coordinates in the NMS process. To accelerate the inference, Fast NMS [25] enables deciding the predictions to be kept or discarded 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 python implementation, which is negligible compared with the time of network evaluation, and yields 0.4% AP better than Fast NMS.

3 Our Approach

In this section, we first introduce how our SOLO framework reformulates instance segmentation as a per-pixel classification problem. Next, we provide three different variants following the basic principle. At last, we present the learning and inference strategies, as well as the Matrix NMS.

3.1 Problem Formulation

The central idea of SOLO framework is to reformulate the instance segmentation as two simultaneous category-aware prediction problems. Concretely, our system conceptually divides the input image into a uniform grid, i.e., $S \times S$. If the center of an object falls into a grid cell, that grid cell is responsible for 1) predicting the semantic category as well as 2) segmenting that object instance.

3.1.1 Semantic Category

For each grid, SOLO predicts the C-dimensional output to indicate the semantic class probabilities, where C is the number of classes. These probabilities are conditioned on the grid cell. If we divide the input image into $S \times S$ grids, the output space will be $S \times S \times C$, as shown in Fig. 1 (top). This design is based on the assumption that each cell of the $S \times S$ grid must belong to one individual instance, thus only belonging to one semantic category. During inference, the C-dimensional output indicates the class probability for each object instance.

3.1.2 Instance Mask

In parallel with the semantic category prediction, each positive grid cell will also generate the corresponding instance mask. For an input image I, if we divide it into $S \times S$ grids, there will be at most S^2 predicted masks in total. We explicitly encode these masks at the third dimension (channel) of

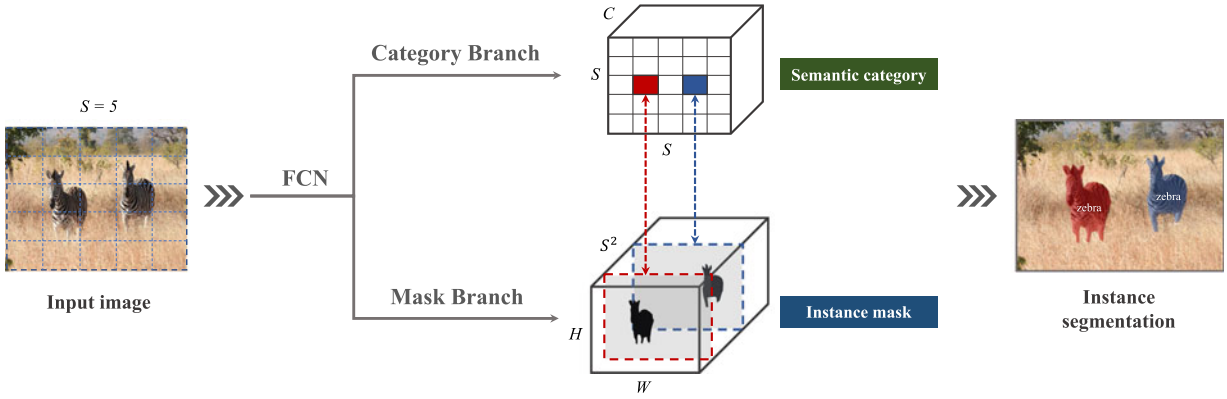


Fig. 1. *SOLO framework.* We reformulate the instance segmentation as two sub-tasks: category prediction and instance mask generation problems. An input image is divided into a uniform grids, i.e., $S \times S$. Here, we illustrate the grid with $S = 5$. If the center of an object falls into a grid cell, that grid cell is responsible for predicting the semantic category (top) and masks of instances (bottom). We do not show the feature pyramid network (FPN) here for simpler illustration.

a 3D output tensor. Specifically, the instance mask output will have $H_I \times W_I \times S^2$ dimension. The k^{th} channel will be responsible to segment instance at grid (i, j), where $k = i \cdot S + j$ (with i and j zero-based). We also show more efficient variants in Section 3.2. To this end, a one-to-one correspondence is established between the semantic category and class-agnostic mask (Fig. 1).

A direct approach to predict the instance mask is to adopt the fully convolutional networks, like FCNs in semantic segmentation [37]. However, the conventional convolutional operations are *spatially invariant* to some degree. Spatial invariance is desirable for some tasks such as semantic segmentation as it enables spatially equivalent prediction. However, here we need a model that is *spatially variant*, or in more precise words, position sensitive, since our segmentation masks are conditioned on the grid cells and must be separated by different feature channels.

Our solution is very simple: at the beginning of the network, we directly feed normalized pixel coordinates to the networks, inspired by 'CoordConv' operator [38]. Specifically, we create a tensor of the same spatial size as input that contains pixel coordinates, which are normalized to $[-1, 1]$. This tensor is then concatenated to the input features and passed to the following layers. By simply giving the convolution access to its own input coordinates, we add the spatial functionality to the conventional FCN model. It should be noted that CoordConv is not the only choice. For example the semi-convolutional operators [39] may be competent, but we employ CoordConv for its simplicity and being easy to implement. If the original feature tensor is of size $H \times W \times D$, the size of the new tensor becomes $H \times W \times (D + 2)$, in which the last two channels are x-y pixel coordinates. For more information on CoordConv, we refer readers to [38].

3.1.3 Forming Instance Segmentation

In SOLO, the category prediction and the corresponding mask are naturally associated by their reference grid cell, i.e., $k = i \cdot S + j$. Based on this, we can directly form the final instance segmentation result for each grid. The raw instance segmentation results are generated by gathering all grid results. Finally, mask non-maximum-suppression (NMS) is used to obtain the final instance segmentation results. No other post-processing operations are needed.

3.2 Network Architecture

SOLO attaches to a convolutional backbone, e.g., ResNet [40]. We use FPN [6], which generates a pyramid of feature maps with different sizes with a fixed number of channels for each level. These feature maps are used as input for the following parallel branches to generate the final predictions, i.e., semantic categories and instance masks.

To demonstrate the generality and effectiveness of our framework, we introduce the vanilla SOLO and a few efficient variants, e.g., Decoupled SOLO and Dynamic SOLO (SOLOv2). We show the overall comparison in Fig. 2. We note that our instance segmentation heads have a straightforward structure. More complex designs have the potential to improve performance, but are not the focus of this work.

3.2.1 Vanilla SOLO

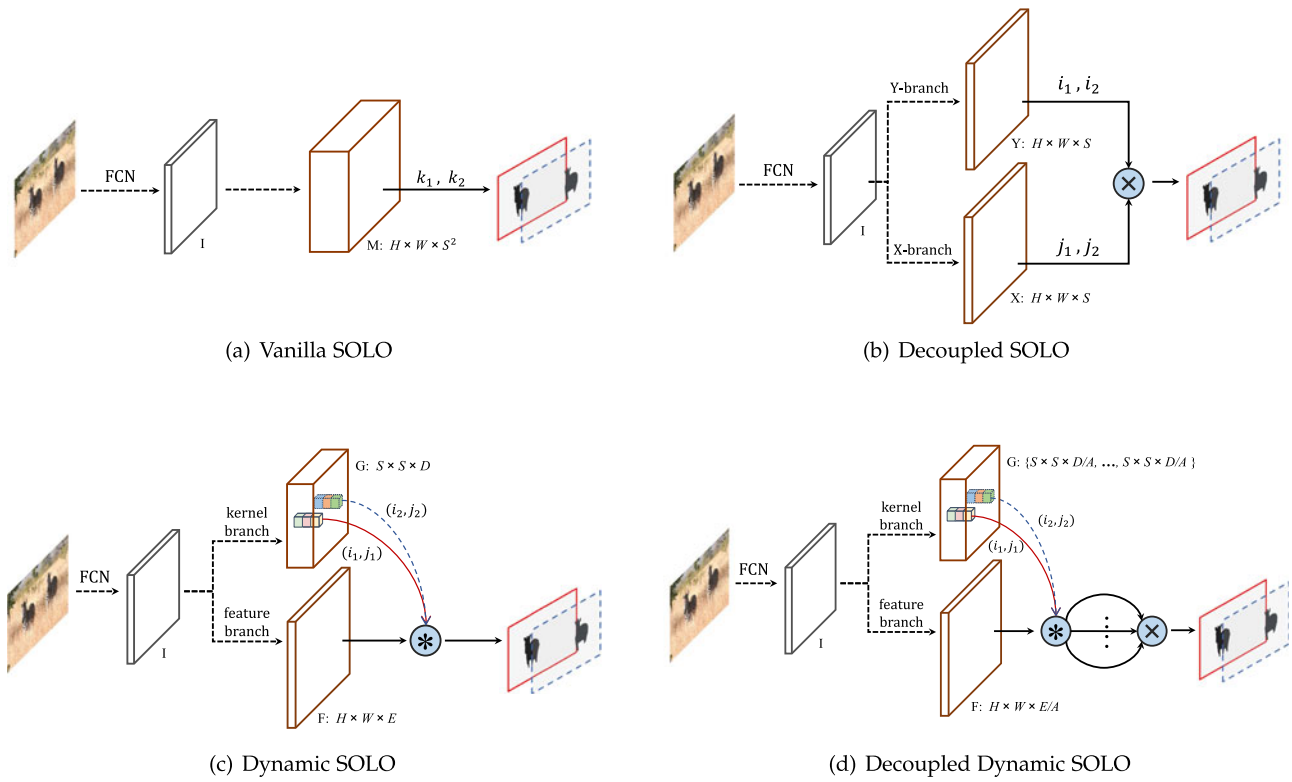
As shown in Fig. 2a, vanilla SOLO has the simplest architecture among all three variants. It directly maps the input image to the output tensor M. The k_1^{th} and k_2^{th} instance mask could be obtained from the k_1^{th} and k_2^{th} channel of M.

We instantiate vanilla SOLO with multiple architectures. The differences include: (a) the *backbone* architecture used for feature extraction, (b) the network *head* for computing the instance segmentation results, and (c) training *loss function* used to optimize the model. Most of the experiments are based on the *head* architecture as shown in Fig. 3.

3.2.2 Decoupled SOLO

Given an predefined grid number, e.g., $S = 20$, vanilla SOLO head outputs $S^2 = 400$ channel maps. However, the prediction is somewhat redundant, as in most cases the objects are located sparsely in the image. In this section, we further introduce an equivalent and significantly more efficient variant of the vanilla SOLO, termed Decoupled SOLO, shown in Fig. 2b.

In Decoupled SOLO, the original output tensor $M \in \mathbb{R}^{H \times W \times S^2}$ is replaced with two output tensors $X \in \mathbb{R}^{H \times W \times S}$ and $Y \in \mathbb{R}^{H \times W \times S}$, corresponding two axes respectively. Thus, the output space is decreased from $H \times W \times S^2$ to $H \times W \times 2S$. For an object located at grid location (i, j), the



(a) Vanilla SOLO (b) Decoupled SOLO

(c) Dynamic SOLO (d) Decoupled Dynamic SOLO

Fig. 2. *SOLO variants.* I is the input feature after FCN-backbone representation extraction. Black dashed arrows denote convolutions. $k = i \cdot S + j$. "$\otimes$" denotes element-wise multiplication. "$\circledast$" denotes the dynamic convolution operation.

mask prediction of that object is defined as the element-wise multiplication of two channel maps

$$\mathbf{m}_k = \mathbf{x}_j \otimes \mathbf{y}_i, \tag{1}$$

where $\mathbf{x}_j$ and $\mathbf{y}_i$ are the j^{th} and i^{th} channel map of X and Y after `sigmoid` operation. The motivation behind this is that *the probability of a pixel belonging to location category (i, j) is the joint probability of belonging to i^{th} row and j^{th} column*, as the horizontal and vertical location categories are independent.

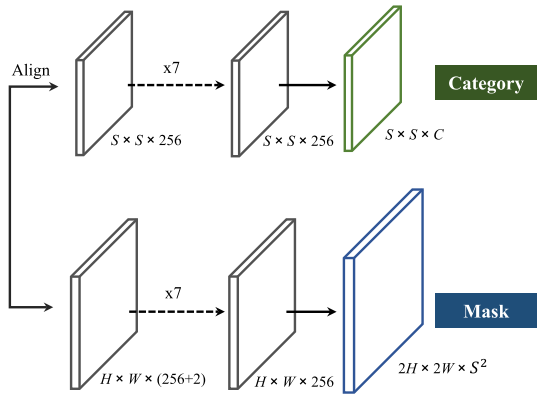


Fig. 3. *Vanilla Head architecture.* At each FPN feature level, we attach two sibling sub-networks, one for instance category prediction (top) and one for instance mask segmentation (bottom). In the mask branch, we concatenate the x, y coordinates and the original features to encode spatial information. Here, numbers denote spatial resolution and channels. In this figure, we assume 256 channels as an example. Arrows denote either convolution or interpolation. 'Align' means bilinear interpolation. During inference, the mask branch outputs are further upsampled to the original image size.

3.2.3 Dynamic SOLO

In vanilla SOLO, to generate the instance mask of S^2 channels corresponding to $S \times S$ grids, the last layer takes one level of pyramid features $F \in \mathbb{R}^{H \times W \times E}$ as input and at last applies a convolution layer with S^2 output channels. The operation can be written as

$$M_{i,j} = G_{i,j} \circledast F, \tag{2}$$

where $G_{i,j} \in \mathbb{R}^{1 \times 1 \times E}$ is the convolution kernel, and $M_{i,j} \in \mathbb{R}^{H \times W}$ is the final mask containing only one instance whose center is at location (i, j).

In other words, we need two input F and G to generate the final mask M. From another perspective, if we separately learn F and G, the final M could be directly generated using the both components. In this way, we can simply pick the valid ones from predicted S^2 kernels and perform the convolution dynamically. The number of model parameters also decreases. Furthermore, as the predicted kernel is generated dynamically conditioned on the input, it benefits from the flexibility and adaptive nature. Additionally, each of the S^2 kernels 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.

Mask Kernel G. Given the backbone and FPN, we predict the mask kernel G at each pyramid level. We first resize the input feature $F_I \in \mathbb{R}^{H_I \times W_I \times C}$ into the shape of $S \times S \times C$. Then 4×convs and a final $3 \times 3 \times D$ conv are employed to generate the kernel G. We add the spatial functionality to F_I by giving the first convolution access to the normalized coordinates following CoordConv [38], i.e., concatenating

two additional input channels which contains pixel coordinates normalized to $[-1, 1]$. Weights for the head are shared across different feature map levels. 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 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.

Mask Feature F. Since the mask feature and mask kernel are decoupled and separately predicted, there are two ways to construct the mask feature. We can put it into the head, along with the kernel branch. It means that we predict the mask features for each FPN level. Or, to predict a unified mask feature representation for all FPN levels. We have compared the two implementations in Section 4.2.2 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 [41]. After repeated stages of 3×3 conv., group norm [42], 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. 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.

3.2.4 Decoupled Dynamic SOLO

We further develop a more unified algorithm by combining decoupled SOLO [43] and dynamic SOLO [44]. Specifically, based on dynamic SOLO, we further decouple the mask representation inspired by decoupled SOLO. We divide the predicted mask kernel weights $G \in \mathbb{R}^{S \times S \times D}$ into A groups, i.e., $G_1, \ldots, G_A$. Each group is in the shape of $S \times S \times {}^D/_A$. The mask feature F is reduced from E channels to ${}^E/_A$ channels. For an object, its mask is predicted as

$$\begin{aligned} M &= M_1 \otimes \ldots \otimes M_A \\ &= \texttt{sigmoid}(G_1 \circledast F) \otimes \ldots \otimes \texttt{sigmoid}(G_A \circledast F). \end{aligned} \tag{3}$$

Decoupled mask representation enables to maintain high performance with a compact mask feature. As such, we adopt mask feature F with reduced output space, e.g., from $H \times W \times 256$ to $H \times W \times 64$ for $A = 4$. The frameworks of the proposed SOLO variants are illustrated and compared in Fig. 2.

3.3 Learning and Inference

3.3.1 Label Assignment

For the category prediction branch, the network needs to give the object category probability for each of $S \times S$ grid. Specifically, grid (i, j) is considered as a positive sample if it falls into the *center region* of any ground truth mask, Otherwise it is a negative sample. Center sampling is effective in recent works of object detection [5], [45], and here we also utilize a similar technique for mask category classification. Given the mass center (c_x, c_y), width w, and height h of the ground truth mask, the center region is controlled by constant scale factors ϵ: $(c_x, c_y, \epsilon w, \epsilon h)$. We set $\epsilon = 0.2$ and there are on average 3 positive samples for each ground truth mask.

Besides the label for instance category, we also have a binary segmentation mask for each positive sample. Since there are S^2 grids, we also have S^2 output masks for each image. For each positive samples, the corresponding target binary mask will be annotated. One may be concerned that the order of masks will impact the mask prediction branch, however, we show that the most simple row-major order works well for our method.

3.3.2 Loss Function

We define our training loss function as follows:

$$L = L_{cate} + \lambda L_{mask}, \tag{4}$$

where L_{cate} is the conventional Focal Loss [2] for semantic category classification. L_{mask} is the loss for mask prediction

$$L_{mask} = \frac{1}{N_{pos}} \sum_k \mathbb{1}_{\{\mathbf{p}^*_{i,j} > 0\}} d_{mask}(\mathbf{m}_k, \mathbf{m}^*_k), \tag{5}$$

Here indices $i = \lfloor k/S \rfloor, j = k \bmod S$, if we index the grid cells (instance category labels) from left to right and top to down. N_{pos} denotes the number of positive samples, $\mathbf{p}^*$ and $\mathbf{m}^*$ represent category and mask target respectively. $\mathbb{1}$ is the indicator function, being 1 if $\mathbf{p}^*_{i,j} > 0$ and 0 otherwise.

We have compared different implementations of $d_{mask}(\cdot, \cdot)$: Binary Cross Entropy (BCE), Focal Loss [2] and Dice Loss [46]. Finally, we employ Dice Loss for its effectiveness and stability in training. λ in Equation (4) is set to 3. The Dice Loss is defined as

$$L_{Dice} = 1 - D(\mathbf{p}, \mathbf{q}), \tag{6}$$

where D is the dice coefficient, which is defined as

$$D(\mathbf{p}, \mathbf{q}) = \frac{2 \sum_{x,y} (\mathbf{p}_{x,y} \cdot \mathbf{q}_{x,y})}{\sum_{x,y} \mathbf{p}^2_{x,y} + \sum_{x,y} \mathbf{q}^2_{x,y}}. \tag{7}$$

Here $\mathbf{p}_{x,y}$ and $\mathbf{q}_{x,y}$ refer to the value of pixel located at (x, y) in predicted soft mask $\mathbf{p}$ and ground truth mask $\mathbf{q}$.

3.3.3 Inference Pipeline

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. For vanilla and decoupled SOLO, we select the top 500 scoring masks and feed them into the NMS operation. For dynamic SOLO, the corresponding predicted mask kernels are used to perform convolution on the mask feature to generate the soft masks. 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.

Maskness. We calculate maskness for each predicted mask, which represents the quality and confidence of mask

prediction $\texttt{maskness} = \frac{1}{N_f}\sum_i^{N_f} \mathbf{p}_i$. Here N_f is the number of foreground pixels of the predicted soft mask $\mathbf{p}$, i.e., the pixels that have values greater than threshold 0.5. The classification score for each prediction is multiplied by the maskness as the final confidence score.

3.3.4 Matrix NMS

Motivation. Our Matrix NMS is motivated by Soft-NMS [9]. 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 ($s_i > s_j$), where s_i and s_j are the confidence scores; 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

$$f(\texttt{iou}_{\cdot,i}) = \min_{\forall s_k > s_i} f(\texttt{iou}_{k,i}). \tag{8}$$

To this end, the final decay factor becomes

$$decay_j = \min_{\forall s_i > s_j} \frac{f(\texttt{iou}_{i,j})}{f(\texttt{iou}_{\cdot,i})}, \tag{9}$$

and the updated score is computed by $s_j = s_j \cdot decay_j$. We consider two most simple decremented functions, denoted as $\texttt{linear} f(\texttt{iou}_{i,j}) = 1 - \texttt{iou}_{i,j}$, and $\texttt{Gaussian}$ $f(\texttt{iou}_{i,j}) = \exp(-\frac{\texttt{iou}_{i,j}^2}{\sigma})$.

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 (Equation (9)). Finally, the scores are updated by the decay factors. For usage, we only need thresholding and selecting top-k scoring masks as the final predictions.

The pseudo-code of Matrix NMS is provided in Fig. 4. In our code base, Matrix NMS is 9× faster than traditional NMS and being more accurate (Table 5(e)). We show that Matrix NMS serves as a superior alternative of traditional NMS in both 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, Equation (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
```

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

4 Experiments

To evaluate the proposed methods, we conduct experiments on three basic tasks, instance segmentation, object detection, and panoptic segmentation on MS COCO [23]. Specifically, we report the results of a few variants of SOLO on COCO instance segmentation benchmark and adopt our best variant SOLOv2 in other experiments for its simplicity and high performance.[1] We also present experimental results on Cityscapes [47] and LVIS [48]. Unlike COCO, Cityscapes specifically focuses on urban street scenes. LVIS has more than 1K categories and thus is considerably more challenging compared with COCO. We conduct extensive ablation experiments, as well as intuitive visualization, to show how each component contributes and how SOLO framework works.

4.1 Main Results

4.1.1 MS COCO Instance Segmentation

For our main results, we report MS COCO mask AP on the `test-dev` split, which has no public labels and is evaluated on the evaluation server.

Training Details. SOLO 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×) 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 [52] pre-trained weights. We use scale jittering, where the shorter image side is randomly sampled from 640 to 800 pixels.

We compare our methods to the state-of-the-art methods in instance segmentation on MS COCO `test-dev` in Table 1. Our SOLOv2 with ResNet-101 achieves a mask AP of 39.7%, which is much better than other state-of-the-art instance segmentation methods. Our method shows its superiority especially 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 segmentation algorithms (Fig. 5a). We show our models with ResNet-50, ResNet-101, ResNet-DCN-101 and two light-weight

1. The default SOLOv2 has similar performance with the dynamic version while being conceptually simpler.

TABLE 1
Instance Segmentation Mask AP (%) on COCO `test-dev`

	backbone	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
box-based:							
Mask R-CNN [16]	R101-FPN	35.7	58.0	37.8	15.5	38.1	52.4
Mask R-CNN*	R101-FPN	37.8	59.8	40.7	20.5	40.4	49.3
MaskLab+ [49]	R101-C4	37.3	59.8	39.6	16.9	39.9	53.5
TensorMask [18]	R101-FPN	37.1	59.3	39.4	17.4	39.1	51.6
YOLACT [25]	R101-FPN	31.2	50.6	32.8	12.1	33.3	47.1
MEInst [50]	R101-FPN	33.9	56.2	35.4	19.8	36.1	42.3
CenterMask [51]	Hourglass-104	34.5	56.1	36.3	16.3	37.4	48.4
BlendMask [35]	R101-FPN	38.4	60.7	41.3	18.2	41.5	53.3
CondInst [34]	R101-FPN	39.1	60.9	42.0	21.5	41.7	50.9
box-free:							
PolarMask [29]	R101-FPN	32.1	53.7	33.1	14.7	33.8	45.3
SOLO	R50-FPN	36.8	58.6	39.0	15.9	39.5	52.1
SOLO	R101-FPN	37.8	59.5	40.4	16.4	40.6	54.2
D-SOLO	R50-FPN	37.4	58.5	40.0	16.2	40.3	52.9
D-SOLO	R101-FPN	38.4	59.6	41.1	16.8	41.5	54.6
SOLOv2	R50-FPN	38.8	59.9	41.7	16.5	41.7	56.2
SOLOv2	R101-FPN	39.7	60.7	42.9	17.3	42.9	57.4
SOLOv2	R-dcn101-FPN	41.7	63.2	45.1	18.0	45.0	61.6
SOLOv2$^+$	R-dcn101-FPN	42.8	64.6	46.4	19.9	45.7	61.9

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. SOLOv2$^+$ is our improved version with auxiliary semantic loss and denser grid, as detailed in Section 4.3.*

versions described in Section 4.2.2. 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. We download the code and pre-trained models to test inference time for each model on the same machine. Further, as described in Fig. 5b, SOLOv2 predicts much finer masks than Mask R-CNN.

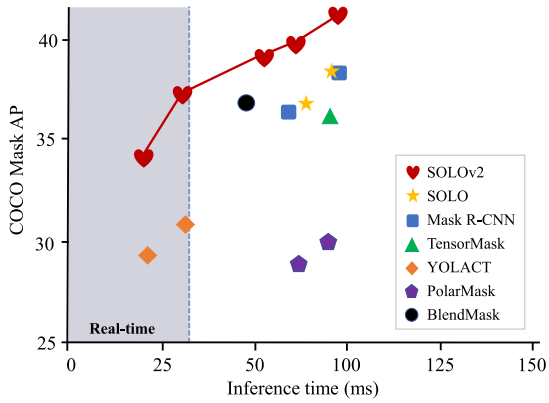
4.1.2 Instance Segmentation on LVIS

LVIS [48] is a recently proposed dataset for long-tail object segmentation, which has more than 1,000 object categories. In LVIS, each object instance is segmented with a high-quality mask that surpasses the annotation quality of the relevant COCO dataset.

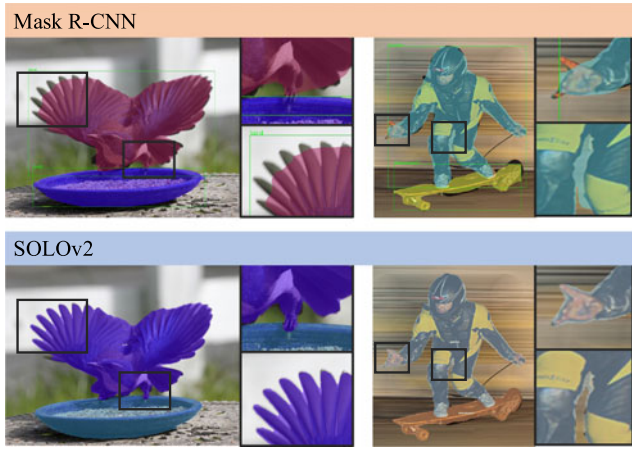
Table 2 reports the performances on the rare (1∼10 images), common (11∼100), and frequent (>100) subsets, as well as the overall AP. Both the reported Mask R-CNN and SOLOv2 use data resampling training strategy, following [48]. Our SOLOv2 outperforms the baseline method by about 1% AP. For large-size objects (AP_L), our SOLOv2 achieves 6.7% AP improvement, which is consistent with the results on the COCO dataset.

4.1.3 Instance Segmentation on Cityscapes

Cityscapes [47] is another popular instance segmentation benchmark which only focuses on urban street scenes. The



(a) Accuracy vs. Speed



(b) Segmentation Detail Comparison

Fig. 5. Comparison of instance segmentation performance of SOLO and other methods on the COCO `test-dev`. (a) The proposed method outperforms a range of state-of-the-art algorithms. All methods are evaluated using one Tesla V100 GPU. (b) SOLOv2 obtains higher-quality masks compared with Mask R-CNN. Mask R-CNN's mask head is typically restricted to 28×28 resolution, leading to inferior prediction at object boundaries.

size of images in Cityscapes is much larger than that in MS COCO. We double the grid numbers for the experiments on Cityscapes. As shown in Table 3, SOLOv2 outperforms Mask R-CNN by 0.9% AP, which demonstrates the effectiveness of our method.

4.2 Ablation Study

4.2.1 SOLO Ablation

Grid Number. We compare the impacts of grid number on the performance with single output feature map as shown in Table 4(a). The feature is generated by merging C3, C4, and C5 outputs in ResNet (stride: 8). To our surprise, $S =$

TABLE 2
Instance Segmentation Results on the LVISv0.5 Validation Dataset

	backbone	AP_r	AP_c	AP_f	AP_S	AP_M	AP_L	AP
Mask R-CNN [48]	Res-50-FPN	14.5	24.3	28.4	-	-	-	24.4
Mask R-CNN*-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

∗ *means re-implementation.*

TABLE 3
Instance Segmentation Results on the Cityscapes `val`

	backbone	AP
Mask R-CNN [16]	Res-50-FPN	36.4
Mask R-CNN*	Res-50-FPN	36.5
SOLOv2	Res-50-FPN	37.4
SOLOv2	Res-101-FPN	38.0

$*$ *means re-implementation.*

12 can already achieve 27.2% AP on the challenging MS COCO dataset. SOLO achieves 29% AP when improving the grid number to 24. This results indicate that our single-scale SOLO can be applicable to some scenarios where object scales do not vary much.

Multi-Level Prediction. From Table 4(a) we can see that our single-scale SOLO could already get 29.0% AP on MS COCO dataset. In this ablation, we show that the performance could be further improved via multi-level prediction using FPN [6]. We use five pyramids to segment objects of different scales (details in supplementary), which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPAMI.2021.3111116. Scales of ground-truth masks are explicitly used to assign them to the levels of the pyramid. From P2 to P6, the corresponding grid numbers are $[40, 36, 24, 16, 12]$ respectively. Based on our multi-level prediction, we further achieve 35.8% AP. As expected, the performance over all the metrics has been largely improved.

CoordConv. Another important component that facilitates our SOLO paradigm is the *spatially variant* convolution (CoordConv [38]). As shown in Table 4(b), the standard convolution can already have spatial variant property to some extent, which is in accordance with the observation in [38]. As also revealed in [53], CNNs can implicitly learn the absolute position information from the commonly used zero-padding operation. 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 3.6% absolute AP gains. Two or more CoordConvs do not bring noticeable improvement. It suggests that a single CoordConv already enables the predictions to be well spatially variant/position sensitive.

Loss Function. Table 4(c) compares different loss functions for our mask optimization branch. The methods include conventional Binary Cross Entropy (BCE), Focal Loss (FL), and Dice Loss (DL). To obtain improved performance, for Binary Cross Entropy, we set a mask loss weight of 10 and a pixel weight of 2 for positive samples. The mask loss weight of Focal Loss is set to 20. As shown, the Focal Loss works much better than ordinary Binary Cross Entropy loss. It is because that the majority of pixels of an instance mask are in background, and the Focal Loss is designed to mitigate the sample imbalance problem by decreasing the loss of well-classified samples. However, the Dice Loss achieves the best results without the need of manually adjusting the loss hyper-parameters. Dice Loss views the pixels as a whole object and could establish the right balance between foreground and background pixels automatically. Note that with carefully tuning the balance hyper-parameters and introducing other training tricks, the results of Binary Cross Entropy and Focal Loss may be considerably improved. However, the point here is that with the Dice Loss, training typically becomes much more stable and more likely to attain good results without using much heuristics. To make a fair comparison, we also show the results of Mask R-CNN with Dice loss. In Mask R-CNN, replacing the original BCE loss with Dice loss gives a -0.9% AP drop.

TABLE 4
Ablation Experiments for Vanilla SOLO

(a) The impact of **grid number and FPN**. FPN significantly improves the performance thanks to its ability to deal with varying sizes of objects.

grid number	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
12	27.2	44.9	27.6	8.7	27.6	44.5
24	29.0	47.3	29.9	10.0	30.1	45.8
36	28.6	46.3	29.7	9.5	29.5	45.2
Pyramid	35.8	57.1	37.8	15.0	38.7	53.6

(b) **Conv vs. CoordConv.** CoordConv can considerably improve AP upon standard convolution. Two or more layers of CoordConv are not necessary.

#CoordConv	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
0	32.2	52.6	33.7	11.5	34.3	51.6
1	35.8	57.1	37.8	15.0	38.7	53.6
2	35.7	57.0	37.7	14.9	38.7	53.3
3	35.8	57.4	37.7	15.7	39.0	53.0

(c) **Different loss functions** may be employed in the mask branch. The Dice loss (DL) leads to best AP and is more stable to train.

mask loss	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
BCE	30.0	50.4	31.0	10.1	32.5	47.7
FL	31.6	51.1	33.3	9.9	34.9	49.8
DL	35.8	57.1	37.8	15.0	38.7	53.6

(d) **Head depth.** We use depth being 7 in other vanilla SOLO experiments.

head depth	4	5	6	7	8
AP	34.6	35.2	35.5	35.8	35.8

All models are trained on MS COCO `train2017` and tested on `val2017` unless noted.

Alignment in the Category Branch. In the category prediction branch, we must match the convolutional features with spatial size $H \times W$ to $S \times S$. Here, we compare three common implementations: interpolation, adaptive-pool, and region-grid-interpolation. (a) Interpolation: directly bilinear interpolating to the target grid size; (b) Adaptive-pool: applying a 2D adaptive max-pool over $H \times W$ to $S \times S$; (c) Region-grid-interpolation: for each grid cell, we use bilinear interpolation conditioned on dense sample points, and aggregate the results with average. From our observation, there is no noticeable performance gap between these variants ($\pm 0.1\%$ AP), indicating that the alignment process does not have a significant impact on the final accuracy.

Different Head Depth. In SOLO, instance segmentation is formulated as a pixel-to-pixel task, and we exploit the spatial layout of masks by using an FCN. In Table 4(d), we compare different head depth used in our work. Changing the head depth from 4 to 7 gives 1.2% AP gains. The results show that when the depth grows beyond 7, the performance becomes stable. In this paper, we use depth being 7 in other experiments of vanilla SOLO.

Vanilla versus Decoupled. As shown in Table 1, decoupled SOLO performs slightly better than vanilla SOLO (37.4% versus 36.8% AP). Here, we qualitatively and quantitatively analyze why decoupled SOLO has better performance. For mask prediction, vanilla SOLO defines S^2 location categories. Give an object in the input image, the model needs to figure out which location category this object belongs to. When S is large, e.g., 40, it is challenging to distinguish S^2 location categories. By contrast, decoupled SOLO decouples the S^2 location categories into $2S$ location categories, i.e., S horizontal categories and S vertical categories. For each pixel of an input object, the model only needs to classify it into $2S$ categories, which is considerably simpler than that in vanilla SOLO.

To show a more intuitive comparison, we plot the training curves of vanilla SOLO and decoupled SOLO. As shown in Figure S7 in the supplementary, available online, the loss curves of category prediction are almost the same. By contrast, the decoupled SOLO shows better convergence in mask loss compared to vanilla SOLO. The comparison of the mask loss curve indicates the superiority of the decoupled mask representation.

4.2.2 SOLOv2 Ablation

Kernel Shape. We consider the kernel shape from two aspects: number of input channels and kernel size. The comparisons are shown in Table 5(a). 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 5(b), 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 [53]. 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 SOLO and use seven 3×3 convolutions to predict the mask features. For the latter one, we fuse the FPN's features in a simple way and obtain the unified mask representations. The detailed implementation is in supplementary material, available online. We compare these two modes in Table 5 (c). As shown, the unified representation achieves better results, especially for the medium and large objects. This is easy to understand: In a separate way, the large-size objects are assigned to high-level feature maps of low spatial resolutions, leading to coarse boundary prediction.

Dynamic versus Decoupled. The dynamic head and decoupled head both serve as the efficient varieties of the SOLO head. We compare the results in Table 5(d). All the settings are the same except the head type, which means that for the dynamic head we use the separate features as

TABLE 5
Ablation Experiments for SOLOv2

(a) **Kernel shape.** The performance is stable when the shape goes beyond $1 \times 1 \times 256$.

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

(b) **Explicit coordinates.** Precise coordinates input can considerably improve the results.

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

(c) **Mask feature representation.** We compare the separate mask feature representation in parallel heads and the unified representation.

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

(d) **Dynamic head vs. Decoupled head.** Dynamic* indicates that the dynamic head here is applied on separate features outputted by 7 convs in parallel.

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**

(e) **Matrix NMS.** Matrix NMS outperforms other methods in both speed and accuracy.

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

(f) **Real-time SOLOv2**. The speed is reported on a single V100 GPU by averaging 5 runs (on COCO `test-dev`).

Model	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L	fps
SOLOv2-448	34.0	54.0	36.1	10.3	36.3	54.4	46.5
SOLOv2-512	37.1	57.7	39.7	12.9	40.0	57.4	31.3

(g) **Training schedule.** 1× means 12 epochs using single-scale training. 3× means 36 epochs with multi-scale training.

Schedule	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
1×	34.8	54.8	36.8	13.1	38.0	53.8
3×	37.8	58.5	40.4	15.6	41.3	56.8

All models are trained on MS COCO `train2017` and tested on `val2017` unless noted.

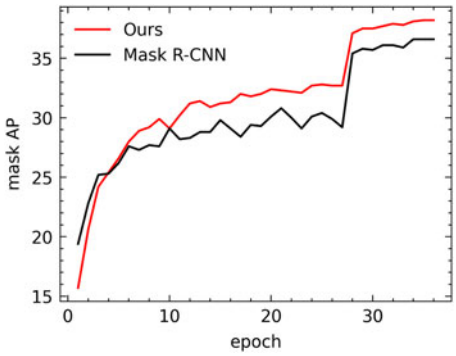


Fig. 6. Instance segmentation results of different numbers of training epochs. Ours is the SOLOv2 model with denser grid. The models are trained on MS COCO `train2017` with 36 epochs, and tested on `val2017`.

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 5(e) presents the speed and accuracy comparison 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 (e.g., 22 ms). Our Matrix NMS only needs less than 1 ms and is almost cost free. Here we also show the performance of Fast NMS, which also 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 [32] is used in the backbone and the last layer of prediction head. We train both models with the $3\times$ schedule, with the shorter side randomly sampled from [352, 512]. Results are shown in Table 5(f). SOLOv2 can not only push state-of-the-art, but has also been ready for real-time applications.

Training Schedule. We report the results of different training schedules in Table 5(g), including 12 epochs using single-scale training ($1\times$) and 36 epochs with multi-scale training ($3\times$). To further show how training time affects our method, we have plotted the AP curve and compared it with the previous dominant Mask R-CNN. As shown in Fig. 6, Mask R-CNN shows a slight advantage in the first 5 epochs. As the training time increases, our method consistently outperforms it by a large margin.

4.3 Other Improvements

In this section, we further explore some potential techniques to improve our method, including auxiliary semantic loss, denser grid, decoupled mask representation, and multi-scale testing. The ablation experiments are conducted on our SOLOv2 model with ResNet-101 backbone. All models are trained for 36 epochs (i.e., $3\times$ schedule) on COCO `train2017`, and evaluated on `val2017`. With auxiliary semantic loss and denser grid, we boost the results of our best single model from 41.7% AP to 42.8% AP on MS COCO `test-dev`, as reported in Table 1.

TABLE 6
Other Improvements on SOLOv2 with ResNet-101 Backbone

(a) **Auxiliary semantic loss.** The additional semantic segmentation branch improves the baseline by 0.3% AP.

aux	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
	39.1	59.8	41.9	16.4	43.2	58.8
✓	39.4	60.1	42.3	16.4	43.4	59.6

(b) **Denser grid.** The performance on small objects (AP_S) is largely improved.

denser	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
	39.1	59.8	41.9	16.4	43.2	58.8
✓	39.8	60.7	42.6	18.6	43.5	58.2

(c) **Decoupled mask representation.** Decoupled dynamic SOLO enables to maintain high performance with a compact mask feature, *i.e.*, $F \in \mathbb{R}^{H\times W\times 64}$.

decoupled	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
	39.1	59.8	41.9	16.4	43.2	58.8
✓	39.3	59.8	42.3	17.1	43.4	58.7

(d) **Multi-scale testing.** Two additional scales bring 0.4% AP gains.

ms-test	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
	39.1	59.8	41.9	16.4	43.2	58.8
✓	39.5	60.4	42.4	17.0	43.4	59.1

The models are trained on MS COCO `train2017` and tested on `val2017`.

Auxiliary Semantic Loss. Inspired by [25], [35], we apply the auxiliary semantic loss during the training. Specifically, a semantic segmentation branch is added to the FPN features P2, which consists of three 3×3 convs. The number of output channels is C, which is the number of object classes, i.e., 80 for COCO dataset. The labels of semantic segmentation are generated from the instance segmentation annotations by merging the object masks of the same class. As shown in Table 6(a), the auxiliary semantic loss brings 0.3% AP gains.

Denser Grid. In our method, a pyramid of number of grid is used during training and inference. The density of the grid decides how many objects the model can detect. To improve the capability of our model, we increase the number of grid from $[40, 36, 24, 16, 12]$ to $[80, 64, 32, 24, 12]$. The denser grid gives 0.7% AP gains over the strong baseline as shown in Table 6(b).

Decoupled Dynamic SOLO. We propose a more unified algorithm by combining decoupled SOLO and dynamic SOLO. Specifically, based on dynamic SOLO, we further decouple the mask representation inspired by decoupled SOLO. In this way, we only need a compact mask feature F with reduced output space, e.g., from $H \times W \times 256$ to $H \times W \times 64$ for $A = 4$. As shown in Table 6(c), we achieve similar performance using a compact mask feature $F \in \mathbb{R}^{H\times W\times 64}$, compared to the $F \in \mathbb{R}^{H\times W\times 256}$ in SOLOv2 without decoupled mask representation. It should be attributed to the improved capacity of the

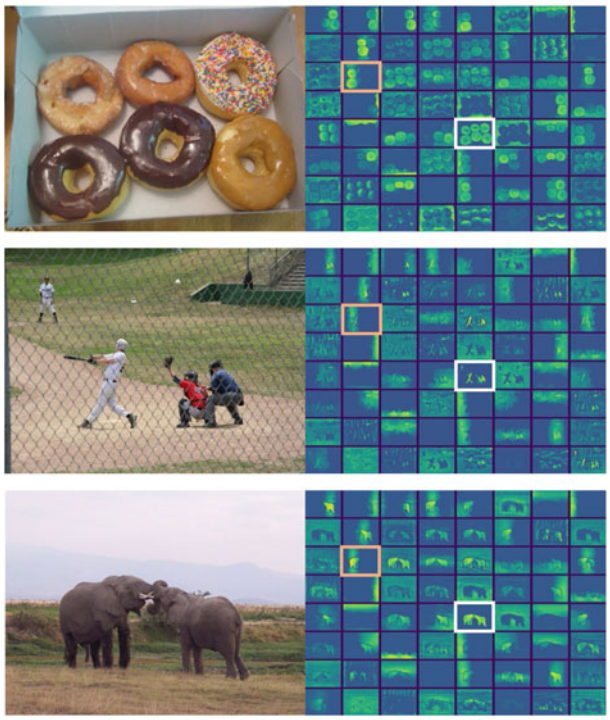


Fig. 7. *Mask feature behavior.* Each plotted subfigure corresponds to one of the 64 channels of the last feature map prior to mask prediction. The mask features appear to be position-sensitive (orange box on the left), while a few mask features are position-agnostic and activated on all instances (white box on the right). Best viewed on screens.

mask representation. The default dynamic scheme in SOLOv2 acts as the linear combination of mask feature maps. The decoupled mask representation improves the capacity through ensembling multiple sub-masks. It enables to maintain high performance with a compact mask feature.

Multi-Scale Testing. In Table 6(d), we report the results of multi-scale testing. During the inference, we feed images with different scales to the model and fuse the outputs as the final predictions. Specifically, we use three scales (the shorter size being 600, 800 and 1000) to get the corresponding outputs and apply Matrix NMS again on these outputs.

4.4 Qualitative Analysis

We visualize what our method has learned 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 Fig. 7.

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. 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 S6 in the supplementary, available online. Different objects are in different colors. Our method shows promising results in diverse scenes. It is worth noting that the details at the boundaries are segmented well, especially for large objects.

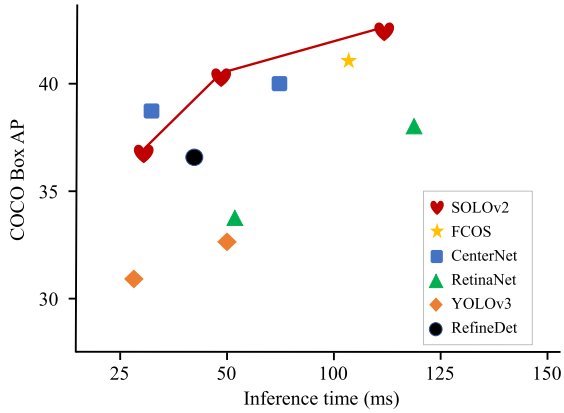


Fig. 8. SOLOv2 for *object detection*. Speed-accuracy trade-off of bounding-box detection on the COCO `test-dev`.

4.5 Extension: 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 S5 in the supplementary, available online, we compare the generated box detection performance with other object detection methods on MS COCO. All models are trained on the `train2017` subset and tested on `test-dev`.

As shown in Table S5 in the supplementary, available online, our detection results outperform most methods, especially for objects of large sizes, demonstrating the effectiveness of SOLOv2 in bounding-box object detection. We also plot the speed/accuracy trade-off curve for different methods in Fig. 8. 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 emphasize that our results are directly generated from the off-the-shelf instance mask, without any box based training or engineering.

An observation from Fig. 8 is as follows. If one does not care much about the annotation 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.

4.6 Extension: Panoptic Segmentation

We also demonstrate the effectiveness of SOLOv2 on the problem of panoptic segmentation [54]. 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 segmentation task. All models are trained on `train2017` subset and tested on `val2017`. We use the same strategy as in Panoptic-FPN [41] to combine instance and semantic results. As shown in Table 7, 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 Pano-DeepLab (Xception-71). Note that UPSNet has used deformable convolutions [32] for better performance.

TABLE 7
SOLOv2 for *Panoptic Segmentation* – Results on MS COCO val2017

	PQ	PQ^{Th}	PQ^{St}
box-based:			
AUNet [55]	39.6	49.1	25.2
UPSNet [56]	42.5	48.5	33.4
Panoptic-FPN [41]	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 [28]	35.9	40.3	29.3
SSAP [22]	36.5	–	–
Pano-DeepLab [57]	39.7	43.9	33.2
SOLOv2	42.1	49.6	30.7

∗ *means our re-implementation.*

4.7 Extension: Instance-Level Image Matting

Image matting is a fundamental problem in computer vision and graphics and has attracted much research attention [58], [59], [60]. Given an image, image matting demands for accurate foreground estimation, which is typically formulated as the alpha map prediction, i.e., to output the soft transition between foreground and background. Most matting methods require an extra trimap input, which indicates the regions of absolute foreground, absolute background and unknown. Few methods [61], [62], [63] explore the trimap-free solution. There is an obvious ambiguity: without trimap it is very challenging to tell which object is the target foreground object given multiple objects in an image. However, it can be solved from another perspective: *to perform image matting at the instance level.* Thus, we will be able to deal with arbitrary foreground objects and enable much more flexible, automated image editing.

To further demonstrate the flexibility of the proposed framework, we extend the proposed instance segmentation framework to perform image matting at instance-level. Thanks to the ability of generating high-quality object masks, our method is able to solve instance-level image matting problem with minimal modifications.

In this setting, instead of the binary pixel-wise mask, for each instance, we want to attain the soft transitions between objects and the background, i.e., soft matte, which is critically important for photo-realistic image manipulation.

4.7.1 Method

The modifications lie in the mask feature branch and the loss function. In mask feature branch, we use the raw input to enhance the feature representation in image details. For the loss function, we add a mean average error term computed between the predicted soft matte and the ground-truth matte. The model is initialized by the weights pre-trained on COCO instance segmentation. During inference, we obtain the soft mattes after sigmoid operation and no thresholding is needed.

In parallel with the prediction head, the mask feature branch takes the feature maps as input and fuses them into a $^1/_4$ scale feature map. Specifically, we modify the mask feature branch by introducing the Details Refinement module. It takes $^1/_4$ scale fused features and $^1/_1$ scale raw image as input and outputs the $^1/_1$ scale low-level features. Specifically, after 1×1 conv the 4× bilinear upsampling, the input features are concatenated with the raw RGB input. The output features after three 3×3 convs are expanded to 256 channels through a 1 × 1 conv. The hidden feature maps all have a small number of channels, e.g., 32. The output and the upsampled original features are fused together through an element-wise summation as the final mask features, which will be convolved by the predicted convolution kernels to generate individual soft mattes for all the objects.

4.7.2 Dataset

To our knowledge, no existing public datasets provide separate alpha matte annotations for each instance. Thus, we construct a training set consists of human matting data with 880 alpha mattes and 520 images with person category from the LVIS training set [48].

4.7.3 Results

We apply the well-trained model to the images collected from the Internet. As shown in Figure S4 in the supplementary, available online, our model is able to generate high-quality alpha matte. The predicted alpha matte enables us to perform some image editing applications for instance-level image editing and compositing. The accurate predictions at object boundaries make the composite image look very natural. In order to analyze the effect of details refinement, we visualize the feature maps at its input and output. As shown in Figure S5 in the supplementary, available online, the input feature maps show coarse activation at boundaries and details. The details are recovered after the detail refinement.

Note that the model trained with human matting data has reasonable matting results for objects of other categories, e.g., the dog in Figure S6. This capability comes from two aspects. 1) As mentioned in Section 4.7.1, the model is initialized by the weights pre-trained on COCO instance segmentation. During the training, the weights of the object category branch are frozen for maintaining the strong object recognition ability. 2) The matting capability is transferred from human to other categories. Because in our framework, the matte prediction is class-agnostic. Basically, the mask branch predicts the soft masks for all the potential objects. The object classes are determined by the category branch. Thus, when the model learns with human matting data, the enhanced matte features (Figure S5 in the supplementary), available online, also benefit objects of other categories. More results and details of the instance-level image matting are in the supplementary, available online.

5 Conclusion

In this work, we have developed a simple and direct instance segmentation framework, termed SOLO. To fully exploit the capabilities of this framework, we propose a few variants following the same principle (i.e., segmenting objects by locations). Besides the state-of-the-art performance in instance segmentation, we demonstrate the flexibility and effectiveness of SOLO by extending it with minimal modifications to

solve object detection, panoptic segmentation and instance-level image matting problems. Given the simplicity, efficiency, and strong performance of SOLO, we hope that our method can serve as a cornerstone for many instance-level recognition tasks.

REFERENCES

[1] S. Ren, K. He, R. B. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in *Proc. Proc. Int. Conf. Neural Inf. Process. Syst.*, 2015, pp. 91–99.

[2] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *Proc. IEEE Int. Conf. Comp. Vis.*, 2017, pp. 2999–3007.

[3] X. Zhou, J. Zhuo, and P. Krahenbuhl, "Bottom-up object detection by grouping extreme and center points," in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2019, pp. 850–859.

[4] Z. Yang, S. Liu, H. Hu, L. Wang, and S. Lin, "RepPoints: Point set representation for object detection," in *Proc. IEEE Int. Conf. Comp. Vis.*, 2019, pp. 9656–9665.

[5] Z. Tian, C. Shen, H. Chen, and T. He, "FCOS: Fully convolutional one-stage object detection," in *Proc. IEEE Int. Conf. Comp. Vis.*, 2019, pp. 9626–9635.

[6] T. Lin, P. Dollár, R. B. Girshick, K. He, B. Hariharan, and S. J. Belongie, "Feature pyramid networks for object detection," in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2017, pp. 936–944.

[7] S. Zhang, L. Wen, X. Bian, Z. Lei, and S. Z. Li, "Single-shot refinement neural network for object detection," in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2018, pp. 4203–4212.

[8] M. Tan, R. Pang, and Q. V. Le, "Efficientdet: Scalable and efficient object detection," in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2020, pp. 10778–10787.

[9] N. Bodla, B. Singh, R. Chellappa, and L. Davis, "Soft-NMS: Improving object detection with one line of code," in *Proc. IEEE Int. Conf. Comp. Vis.*, 2017, pp. 5562–5570.

[10] Y. He, C. Zhu, J. Wang, M. Savvides, and X. Zhang, "Bounding box regression with uncertainty for accurate object detection," in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2019, pp. 2883–2892.

[11] L. Cai *et al.*, "Maxpoolnms: Getting rid of NMS bottlenecks in two-stage object detectors," in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2019, pp. 9348–9356.

[12] J. Yu, Y. Jiang, Z. Wang, Z. Cao, and T. Huang, "Unitbox: An advanced object detection network," in *Proc. ACM Int. Conf. Multimedia*, 2016, pp. 516–520.

[13] H. Rezatofighi, N. Tsoi, J. Gwak, A. Sadeghian, I. Reid, and S. Savarese, "Generalized intersection over union: A metric and a loss for bounding box regression," in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2019, pp. 658–666.

[14] H. Hu, J. Gu, Z. Zhang, J. Dai, and Y. Wei, "Relation networks for object detection," in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2018, pp. 3588–3597.

[15] Y. Li, H. Qi, J. Dai, X. Ji, and Y. Wei, "Fully convolutional instance-aware semantic segmentation," in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2017, pp. 4438–4446.

[16] K. He, G. Gkioxari, P. Dollár, and R. B. Girshick, "Mask R-CNN," in *Proc. IEEE Int. Conf. Comp. Vis.*, 2017, pp. 2980–2988.

[17] K. Chen *et al.*, "Hybrid task cascade for instance segmentation," in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2019, pp. 4969–4978.

[18] X. Chen, R. Girshick, K. He, and P. Dollar, "Tensormask: A foundation for dense object segmentation," in *Proc. IEEE Int. Conf. Comp. Vis.*, 2019, pp. 2061–2069.

[19] A. Newell, Z. Huang, and J. Deng, "Associative embedding: End-to-end learning for joint detection and grouping," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 2274–2284.

[20] B. De Brabandere, D. Neven, and L. Van Gool, "Semantic instance segmentation with a discriminative loss function," 2017, pp. 478–480.

[21] S. Liu, J. Jia, S. Fidler, and R. Urtasun, "Sequential grouping networks for instance segmentation," in *Proc. IEEE Int. Conf. Comp. Vis.*, 2017, pp. 3516–3524.

[22] N. Gao *et al.*, "SSAP: Single-shot instance segmentation with affinity pyramid," in *Proc. IEEE Int. Conf. Comp. Vis.*, 2019, pp. 642–651.

[23] T. Lin *et al.*, "Microsoft COCO: Common objects in context," in *Proc. Eur. Conf. Comp. Vis.*, 2014, pp. 740–755.

[24] P. H. O. Pinheiro, R. Collobert, and P. Dollár, "Learning to segment object candidates," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2015, pp. 1990–1998.

[25] D. Bolya, C. Zhou, F. Xiao, and Y. J. Lee, "YOLACT: Real-time instance segmentation," in *Proc. IEEE Int. Conf. Comp. Vis.*, 2019, pp. 9156–9165.

[26] S. Liu, L. Qi, H. Qin, J. Shi, and J. Jia, "Path aggregation network for instance segmentation," in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2018, pp. 8759–8768.

[27] Z. Huang, L. Huang, Y. Gong, C. Huang, and X. Wang, "Mask scoring R-CNN," in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2019, pp. 6402–6411.

[28] K. Sofiiuk, O. Barinova, and A. Konushin, "Adaptis: Adaptive instance selection network," in *Proc. IEEE Int. Conf. Comp. Vis.*, 2019, pp. 7354–7362.

[29] E. Xie *et al.*, "PolarMask: Single shot instance segmentation with polar representation," in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2020, pp. 12190–12199.

[30] M. Jaderberg, K. Simonyan, A. Zisserman, and K. Kavukcuoglu, "Spatial transformer networks," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2015, pp. 2017–2025.

[31] X. Jia, B. De Brabandere , T. Tuytelaars, and L. V. Gool, "Dynamic filter networks," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2016, pp. 667–675.

[32] J. Dai *et al.*, "Deformable convolutional networks," in *Proc. IEEE Int. Conf. Comp. Vis.*, 2017, pp. 764–773.

[33] H. Su, V. Jampani, D. Sun, O. Gallo, E. G. L.-Miller, and J. Kautz, "Pixel-adaptive convolutional neural networks," in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2019, pp. 11158–11167.

[34] Z. Tian, C. Shen, and H. Chen, "Conditional convolutions for instance segmentation," in *Proc. Eur. Conf. Comp. Vis.*, 2020, pp. 282–298.

[35] H. Chen, K. Sun, Z. Tian, C. Shen, Y. Huang, and Y. Yan, "BlendMask: Top-down meets bottom-up for instance segmentation," in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2020, pp. 8570–8578.

[36] S. Liu, D. Huang, and Y. Wang, "Adaptive NMS: Refining pedestrian detection in a crowd," in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2019, pp. 6452–6461.

[37] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2015, pp. 3431–3440.

[38] R. Liu *et al.*, "An intriguing failing of convolutional neural networks and the CoordConv solution," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2018, pp. 9628–9639.

[39] D. Novotný, S. Albanie, D. Larlus, and A. Vedaldi, "Semi-convolutional operators for instance segmentation," in *Proc. Eur. Conf. Comp. Vis.*, 2018, pp. 89–105.

[40] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2016, pp. 770–778, .

[41] A. Kirillov, R. Girshick, K. He, and P. Dollár, "Panoptic feature pyramid networks," in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2019, pp. 6392–6401.

[42] Y. Wu and K. He, "Group normalization," in *Proc. Eur. Conf. Comp. Vis.*, 2018, pp. 3–19

[43] X. Wang, T. Kong, C. Shen, Y. Jiang, and L. Li, "SOLO: Segmenting objects by locations," in *Proc. Eur. Conf. Comp. Vis.*, 2020, pp. 649–665.

[44] X. Wang, R. Zhang, T. Kong, L. Li, and C. Shen, "SOLOv2: Dynamic, faster and stronger," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2020.

[45] T. Kong, F. Sun, H. Liu, Y. Jiang, L. Li, and J. Shi, "Foveabox: Beyond anchor-based object detector," *IEEE Trans. Image Process.*, vol. 29, pp. 7389–7398, Jun. 23, 2020, doi: 10.1109/TIP.2020.3002345.

[46] F. Milletari, N. Navab, and S.-A. Ahmadi, "V-net: Fully convolutional neural networks for volumetric medical image segmentation," in *Proc. Int. Conf. 3D Vision*, 2016, pp. 565–571.

[47] M. Cordts *et al.*, "The cityscapes dataset for semantic urban scene understanding," in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2016, pp. 3213–3223.

[48] A. Gupta, P. Dollar, and R. Girshick, "LVIS: A dataset for large vocabulary instance segmentation," in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2019, pp. 5351–5359.

[49] L.-C. Chen, A. Hermans, G. Papandreou, F. Schroff, P. Wang, and H. Adam, "MaskLab: Instance segmentation by refining object detection with semantic and direction features," in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2018, pp. 4013–4022.

[50] R. Zhang, Z. Tian, C. Shen, M. You, and Y. Yan, "Mask encoding for single shot instance segmentation," in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2020, pp. 10223–10232.

[51] Y. Wang, Z. Xu, H. Shen, B. Cheng, and L. Yang, "CenterMask: Single shot instance segmentation with point representation," in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2020, pp. 9310–9318.
[52] O. Russakovsky *et al.*, "Imagenet large scale visual recognition challenge," *Int. J. Comput. Vision*, vol. 115, no. 3, pp. 211–252, 2015.
[53] M. A. Islam, S. Jia, and N. D. B. Bruce, "How much position information do convolutional neural networks encode?," in *Proc. Int. Conf. Learn. Representations*, 2020.
[54] A. Kirillov, K. He, R. B. Girshick, C. Rother, and P. Dollár, "Panoptic segmentation," in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2019, pp. 9396–9405.
[55] Y. Li *et al.*, "Attention-guided unified network for panoptic segmentation," in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2019, pp. 7019–7028.
[56] Y. Xiong *et al.*, "UPSNet: A unified panoptic segmentation network," in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2019, pp. 8810–8818.
[57] B. Cheng *et al.*, "Panoptic-deeplab: A simple, strong, and fast baseline for bottom-up panoptic segmentation," in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2020, pp. 12472–12482.
[58] A. Levin, D. Lischinski, and Y. Weiss, "A closed-form solution to natural image matting," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 30, no. 2, pp. 228–242, Feb. 2008.
[59] N. Xu, B. Price, S. Cohen, and T. Huang, "Deep image matting," in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2017, pp. 311–320.
[60] H. Lu, Y. Dai, C. Shen, and S. Xu, "Indices matter: Learning to index for deep image matting," in *Proc. IEEE Int. Conf. Comp. Vis.*, 2019, pp. 3265–3274.
[61] Q. Chen, T. Ge, Y. Xu, Z. Zhang, X. Yang, and K. Gai, "Semantic human matting," in *Proc. ACM Int. Conf. Multimedia*, 2018, pp. 618–626.
[62] Y. Zhang *et al.*, "A late fusion CNN for digital matting," in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2019, pp. 7461–7470.
[63] J. Liu *et al.*, "Boosting semantic human matting with coarse annotations," in *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2020, pp. 8560–8569.