FastInst: A Simple Query-Based Model for Real-Time Instance Segmentation

Junjie He, Pengyu Li, Yifeng Geng, Xuansong Xie
DAMO Academy, Alibaba Group
hejunjie.hjj@alibaba-inc.com, lipengyu007@gmail.com, cangyu.gyf@alibaba-inc.com
xingtong.xxs@taobao.com

Abstract

Recent attention in instance segmentation has focused on query-based models. Despite being non-maximum suppression (NMS)-free and end-to-end, the superiority of these models on high-accuracy real-time benchmarks has not been well demonstrated. In this paper, we show the strong potential of query-based models on efficient instance segmentation algorithm designs. We present FastInst, a simple, effective query-based framework for real-time instance segmentation. FastInst can execute at a real-time speed (i.e., *32.5 FPS) while yielding an AP of more than 40* (i.e., *40.5 AP) on COCO* `test-dev` *without bells and whistles. Specifically, FastInst follows the meta-architecture of recently introduced Mask2Former. Its key designs include instance activation-guided queries, dual-path update strategy, and ground truth mask-guided learning, which enable us to use lighter pixel decoders, fewer Transformer decoder layers, while achieving better performance. The experiments show that FastInst outperforms most state-of-the-art real-time counterparts, including strong fully convolutional baselines, in both speed and accuracy. Code can be found at* `https://github.com/junjiehe96/FastInst`.

1. Introduction

Instance segmentation aims to segment all objects of interest in an image. The mainstream methods like Mask R-CNN [5, 15, 19, 28] follow the design of detection-then-segmentation. Despite being simple and intuitive, those methods generate a lot of duplicate region proposals that introduce redundant computations. To improve efficiency, many single-stage methods [2, 8, 23, 42] built upon Fully Convolutional Networks (FCNs) [29] appear. They segment objects end-to-end without region proposals. The inference speed of such methods is appealing, especially in real-time scenes. However, due to the dense predictions, the classical single-stage methods still rely on manually-designed post-processing steps like non-maximum suppression (NMS).

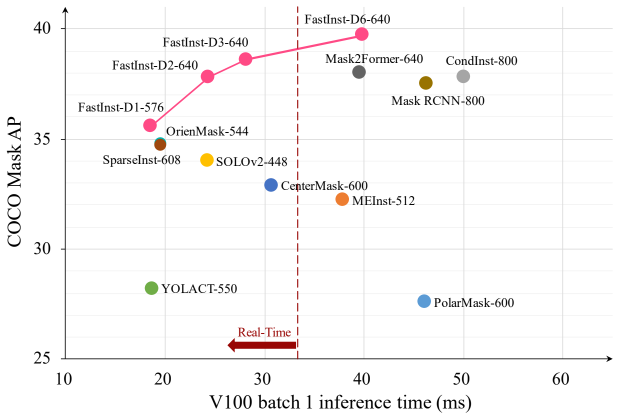

Figure 1. **Speed-performance trade-off on COCO `test-dev`.** All models employ ResNet-50 [16] as the backbone except OrienMask with DarkNet-53 [33]. Our FastInst surpasses most state-of-the-art real-time instance segmentation algorithms in both speed and accuracy. To keep the speed and accuracy in a similar order, Mask2Former here takes the pyramid pooling module-based [48] FPN as the pixel decoder, the same as FastInst and SparseInst.

Recently, with the success of DETR [4] in object detection, query-based single-stage instance segmentation methods [9, 10, 25, 43] have emerged. Instead of convolution, they exploit the versatile and powerful attention mechanism [39] combined with a sequence of learnable queries to infer the object class and segmentation mask. For example, Mask2Former [9] simplifies the workflow of instance segmentation by adding a pixel decoder and a masked-attention Transformer decoder on top of a backbone. Unlike previous methods [15, 42], Mask2Former does not require additional handcrafted components, such as training target assignment and NMS post-processing. While being simple, Mask2Former has its own issues: (1) it requires a large number of decoder layers to decode the object queries since its queries are learned static and need a lengthy process to refine; (2) It relies upon a heavy pixel decoder, *e.g.*, multi-scale deformable attention Transformer (MSDeformAttn) [50], because its object segmentation mask straightforwardly depends on the output of the pixel decoder, which is used as a per-pixel embedding feature for distinguishing different objects; (3) masked attention restricts the receptive field of each query, which may cause the Transformer decoder to fall into a suboptimal query update process. Although Mask2Former achieves outstanding performance, its superiority on fast, efficient instance segmentation has not been well demonstrated, which yet is critical for many real-world applications such as self-driving cars and robotics. In fact, due to the lack of prior knowledge and the high computational complexity of the attention mechanism, the efficiency of query-based models is generally unsatisfactory [9, 18, 25]. The efficient real-time instance segmentation benchmarks are still dominated by classical convolution-based models [11, 42].

In this paper, we fill this gap by proposing FastInst, a concise and effective query-based framework for real-time instance segmentation. We demonstrate that the query-based model can achieve outstanding performance on the instance segmentation task while maintaining a fast speed, showing great potential in efficient instance segmentation algorithm design. As an example, our designed fastest query-based model with ResNet-50 [16] backbone achieves 35.6 AP at 53.8 FPS (frames-per-second) on the COCO [27] `test-dev`, evaluated on a single V100 GPU (see Figure 1); moreover, our best trade-off model can execute at a real-time speed, *i.e*., 32.5 FPS, while yielding an AP of more than 40, *i.e*., 40.5 AP, which to the best of our knowledge, has not yet been achieved in previous methods.

Specifically, our model follows the meta-architecture of Mask2Former [9]. To achieve efficient real-time instance segmentation, we have proposed three key techniques. First, we use instance activation-guided queries, which dynamically pick the pixel embeddings with high semantics from the underlying feature map as the initial queries for the Transformer decoder. Compared with static zero [4] or learnable [9, 10] queries, these picked queries contain rich embedding information about potential objects and reduce the iteration update burden of the Transformer decoder. Second, we adopt a dual-path architecture in the Transformer decoder where the query features and the pixel features are updated alternately. Such a design enhances the representational ability of pixel features and saves us from the heavy pixel decoder design. Moreover, it makes a direct communication between query features and pixel features, which speeds up the iterative update convergence and effectively reduces the dependence on the number of decoder layers. Third, to prevent the masked attention from falling into a suboptimal query update process, we introduce ground truth mask-guided learning. We replace the mask used in the standard masked attention with the last-layer bipartite matched ground truth mask to forward the Transformer decoder again and use a fixed matching assignment to supervise the outputs. This guidance allows each query to see the whole region of its target predicted object during training and helps masked attention attend within a more appropriate foreground region.

We evaluate FastInst on the challenging MS COCO dataset [27]. As shown in Figure 1, FastInst obtains strong performance on the COCO benchmark while staying fast, surpassing most of the previous state-of-the-art methods. We hope FastInst can serve as a new baseline for real-time instance segmentation and advance the development of query-based instance segmentation models.

2. Related Work

Existing instance segmentation techniques can be grouped into three classes, *i.e*., region-based methods, instance activation-based methods, and query-based methods.

Region-based methods first detect object bounding boxes and then apply RoI operations such as RoI-Pooling [34] or RoI-Align [15] to extract region features for object classification and mask generation. As a pioneering work, Mask R-CNN [15] adds a mask branch on top of Faster R-CNN [34] to predict the segmentation mask for each object. Follow-up methods either focus on improving the precision of detected bounding boxes [3, 5] or address the low-quality segmentation mask arising in Mask R-CNN [12, 21, 36]. Although the performance has been advanced on several benchmarks, these region-based methods suffer from a lot of duplicated region proposals that hurt the model's efficiency.

Instance activation-based methods employ some meaningful pixels to represent the object and train the features of these pixels to be activated for the segmentation during the prediction. A typical class of such methods is based on the center activation [2, 37, 42, 47], which forces the center pixels of the object to correspond to the segmentation and classification. For example, SOLO [41, 42] exploits the center features of the object to predict a mask kernel for the segmentation. MEInst [46] and CondInst [37] build the model upon the center-activation-based detector FCOS [38] with an additional branch of predicting mask embedding vectors for dynamic convolution. Recently, SparseInst [11] learns a weighted pixel combination to represent the object. The proposed FastInst exploits the pixels located in the object region with the high class semantics as the representation of the object and extracts their features as the queries.

Query-based methods have emerged with DETR [4] and show that a convolutional backbone with an end-to-end set prediction-based Transformer encoder-decoder [39] can achieve good performance on the instance segmentation task. SOLQ [13] and ISTR [18] exploit the learned object queries to infer mask embeddings for instance segmentation. Panoptic SegFormer [25] adds a location decoder to provide object position information. Mask2Former [9, 10] introduces masked attention for improved performance and faster convergence. Mask DINO [24] unifies object detection and image segmentation tasks, obtaining great results on instance segmentation. Despite the outstanding performance, query-based models are usually too computationally expensive to be applied in the real world. Compared with convolutional networks [11, 42], their advantages on fast, efficient instance segmentation have not been well demonstrated. Our goal is to leverage the powerful modeling capabilities of the Transformer while designing an efficient, concise, and real-time instance segmentation scheme to promote the application of query-based segmentation methods. In addition, many works [40, 45] also utilize the dual-path Transformer architecture in image segmentation tasks. However, their designs are generally complex and hard to deploy. We build our dual-path architecture simply upon plain Transformer layers for improved efficiency.

3. Methods

3.1. Overall architecture

As illustrated in Figure 2 , FastInst consists of three modules: backbone, pixel decoder, and Transformer decoder.

Our model feeds an input image $\mathbf{I} \in \mathbb{R}^{H\times W\times 3}$ to the backbone and obtains three feature maps C_3, C_4, and C_5, of which the resolutions are $1/8$, $1/16$, and $1/32$ of the input image, respectively. We project these three feature maps to the ones with 256 channels by a 1×1 convolutional layer and feed them into the pixel decoder. The pixel decoder aggregates the contextual information and outputs the enhanced multi-scale feature maps E_3, E_4, and E_5. After that, we pick N_a instance activation-guided queries from the feature map E_4, concatenated with N_b auxiliary learnable queries to obtain the total queries $\mathbf{Q} \in \mathbb{R}^{N\times 256}$, where $N = N_a + N_b$. The Transformer decoder takes as input the total queries $\mathbf{Q}$ as well as the flattened high-resolution pixel feature E_3, denoted as $\mathbf{X} \in \mathbb{R}^{L\times 256}$, where $L = H/8 \times W/8$. Then in the Transformer decoder, we update the pixel features $\mathbf{X}$ and the queries $\mathbf{Q}$ in a dual-path way and predict the object class and segmentation mask at each decoder layer.

We now discuss each component in detail.

3.2. Lightweight pixel decoder

Multi-scale contextual feature maps are essential for image segmentation [6, 20, 42]. However, using a complicated multi-scale feature pyramid network increases the computational burden. Unlike previous methods [9, 10], which directly employ the underlying feature maps from the pixel decoder, we use the refined pixel features in the Transformer decoder to produce segmentation masks. This setup reduces the requirement of the pixel decoder for heavy context aggregation. We thus can use a lightweight pixel decoder module. For a better trade-off between accuracy and speed, we use a variant called PPM-FPN [11] instead of vanilla FPN [26], which adopts a pyramid pooling module [48] after C_5 to enlarge receptive fields for improved performance.

3.3. Instance activation-guided queries

Object queries play a crucial role in Transformer architecture [4]. One of the reasons for the slow convergence of DETR is that its object queries are zero-initialized. Although learnable queries [9] mitigate this issue, they are still image-independent and require many Transformer decoder layers to refine. Inspired by Deformable DETR [50], which selects the query bounding boxes from pyramidal features for object detection, we propose instance activation-guided queries that straightforwardly pick the queries with high semantics from underlying multi-scale feature maps. Specifically, given the output feature maps of the pixel decoder, we add an auxiliary classification head, followed by a softmax activation, on top of the feature map E_4 to yield the class probability prediction $\mathbf{p}_i \in \Delta^{K+1}$ for each pixel, where Δ^{K+1} is the $(K+1)$-dimensional probability simplex, K is the number of classes, added by one for “no object” ($\varnothing$), i is the pixel index, and the auxiliary classification head is composed of two convolutional layers with 3×3 and 1×1 kernel sizes, respectively. Through $\mathbf{p}_i$ we obtain the foreground probability p_{i,k_i}, $k_i = \mathrm{argmax}_k\{p_{i,k}|p_{i,k} \in \mathbf{p}_i, k \in \{1, \cdots, K\}\}$ for each pixel. Then we select N_a pixel embeddings from the feature map E_4 with high foreground probabilities as the object queries. Here we first select the ones with p_{i,k_i} that is the *local maximum* in the corresponding class plane (*i.e.*, $p_{i,k_i} \geq p_{n,k_i}$, $n \in \delta(i)$, where $\delta(i)$ is the spatially 8-neighboring index set of i) and then pick the ones with the top foreground probabilities in $\{p_{i,k_i}\}_i$. Note that a pixel with a non-local-maximum probability in the corresponding class plane means there exists a pixel in its 8-neighborhood which has a higher probability score of that class. With locations so close, we naturally prefer to pick its neighboring pixel rather than it as the object query.

During training, we apply matching-based Hungarian loss [4, 35] to supervise the auxiliary classification head. Unlike [50], which employs prior anchor boxes and binary classification scores for the matching problem, we simply use the class predictions with a *location cost* $\mathcal{L}_{\text{loc}}$ to compute the assignment costs. The location cost $\mathcal{L}_{\text{loc}}$ is defined as an indicator function that is 0 when the pixel is located in the region of that object; otherwise, it is 1. The intuition behind this cost is that only pixels that fall inside an object can have a reason to infer the class and mask embedding of that object. Also, the location cost reduces the bipartite matching space and speeds up training convergence.

We term the queries generated from the above strategy as instance activation-guided (IA-guided) queries. Compared to the zero [4] or learnable queries [9], IA-guided queries hold rich information about potential objects at the initial and improve the efficiency of query iterations in the Transformer decoder. Note that we can also select the queries

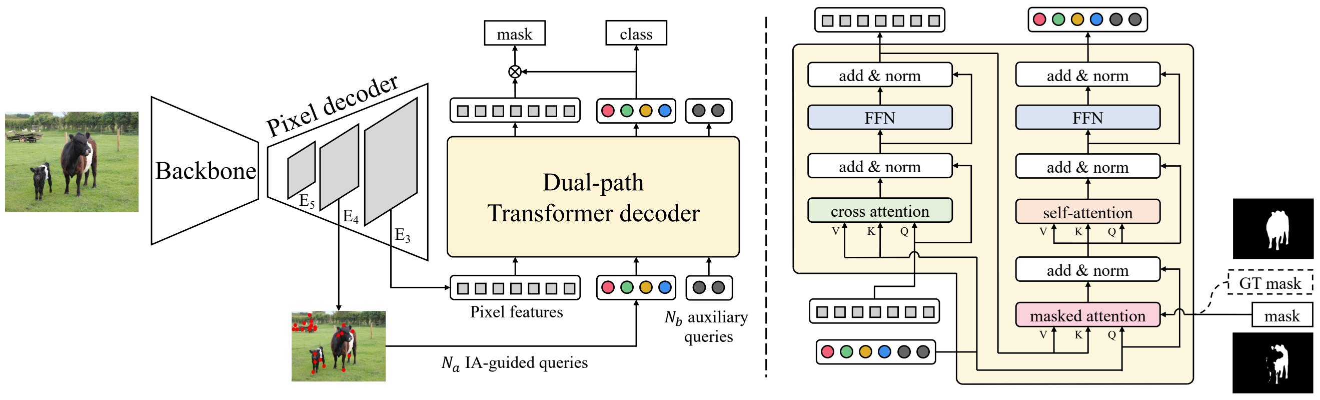

Figure 2. **Model overview.** FastInst consists of three modules: backbone, pixel decoder, and Transformer decoder. The backbone and pixel decoder extract and refine multi-scale features (Section 3.2). The Transformer decoder selects N_a instance activation-guided queries (IA-guided queries) from the feature map E_4 (Section 3.3) and concatenates them with N_b auxiliary learnable queries as initial queries. Then taking the initial queries and the flattened feature map E_3 as input, the Transformer decoder performs the object classification and segmentation at each layer with a dual-path update strategy (Section 3.4). During training, we introduce ground truth (GT) mask-guided learning to improve the performance of masked attention (Section 3.5). For readability, we omit positional embeddings in this figure.

from feature maps E_3 or E_5. Larger feature maps contain richer instance clues but suffer heavier computational burdens. We use the middle-size feature map E_4 for a trade-off.

3.4. Dual-path Transformer decoder

After selecting N_a IA-guided queries from the underlying feature map, we concatenate them with N_b auxiliary learnable queries to obtain the total queries $\mathbf{Q}$, where auxiliary learnable queries are used to facilitate grouping background pixel features and provide general image-independent information in the subsequent dual update process. Then the total queries $\mathbf{Q}$ combined with the flattened $1/8$ high-resolution pixel features $\mathbf{X}$ are fed into the Transformer decoder. In the Transformer decoder, we add positional embeddings for queries $\mathbf{Q}$ and pixel features $\mathbf{X}$, followed by successive Transformer decoder layers to update them. One Transformer decoder layer *contains* one pixel feature update and one query update. The whole process is like an EM (Expectation–Maximization) clustering algorithm. E step: update pixel features according to the centers (queries) they belong to; M step: update cluster centers (queries). Compared with the single-path update strategy [9], the dual-path update strategy co-optimizes both pixel features and queries, reducing the dependence on heavy pixel decoders and acquiring more fine-grained feature embeddings. Finally, we use the refined pixel features and queries to predict the object classes and segmentation masks at each layer.

Positional embeddings. Location information is critical in distinguishing different instances with similar semantics, especially for objects with the same class [37, 41, 42]. Instead of non-parametric sinusoidal positional embeddings [9], we use the learnable positional embeddings, which we find can improve the model inference speed without compromising the performance. Specifically, we employ a fixed-size learnable spatial positional embedding $\mathbf{P} \in \mathbb{R}^{S\times S\times 256}$, where S is the spatial size and we empirically set it to the rounded square root of the IA-guided query number N_a. During forwarding, we interpolate $\mathbf{P}$ to two different sizes. One is with the same size as E_3, which is then flattened as positional embeddings for pixel features $\mathbf{X}$; the other is with the same size as E_4, from which we select the positional embeddings for IA-guided queries according to their locations $\{(x_i, y_i)\}_{i=1}^{N_a}$ in the feature map E_4. The auxiliary learnable queries employ additional N_b learnable positional embeddings.

Pixel feature update. We first update the pixel features. Given the flattened pixel features $\mathbf{X}$ and the queries $\mathbf{Q}$, the pipeline of pixel feature update consists of a cross-attention layer and a feedforward layer, as illustrated in the right side of Figure 2. The positional embeddings are added to queries and keys at every cross-attention layer [4]. For the update of pixel features, we do not use self-attention, which will introduce a massive computation and memory cost due to the long sequence length of pixel features. The global features can be aggregated through cross-attention on queries.

Query update. Asymmetrically, we use masked attention followed by self-attention and feedforward network for the query update, as in Mask2Former [9]. Masked attention restricts the attention of each query to only attend within the foreground region of the predicted mask from the previous layer, and the context information is hypothesized to be gathered through following self-attention. Such a design has significantly improved query-based model performance in image segmentation tasks [9]. Here the positional embeddings are also added to queries and keys at every masked- and self-attention layer.

Prediction. We apply two separate 3-layer MLPs on top of refined IA-guided queries at each decoder layer to predict object classes and mask embeddings, respectively. Each IA-guided query needs to predict the probability of all object classes, including the "no object" ($\varnothing$) class. A linear projection is added to the refined pixel features to obtain mask features. Then mask embeddings are multiplied with mask features to obtain the segmentation masks for each query. Here the parameters of MLPs and linear projection at each Transformer decoder layer are not shared, since queries and pixel features are updated alternately and their features can be in different representation spaces at different decoder layers. In addition, instance segmentation requires a confidence score for each prediction for evaluation. We follow previous work [9] and multiply the class probability score with the mask score (*i.e.*, the average of mask probabilities in the foreground region) as the confidence score.

3.5. Ground truth mask-guided learning

Although masked attention introduces prior sparse attention knowledge that accelerates model convergence and improves the performance, it restricts the receptive field of each query and may cause the Transformer decoder to fall into a suboptimal query update process. To mitigate this issue, we introduce ground truth (GT) mask-guided learning. Firstly, we use the last layer's bipartite matched ground truth mask to replace the predicted mask used in l-th layer's masked attention. For the queries that do not match any instance in the last layer (including auxiliary learnable queries), we use the standard cross attention, *i.e.*,

$$\mathbf{M}_i^l = \begin{cases} \mathbf{M}_j^{gt} & \text{if } (i,j) \in \sigma \\ \varnothing & \text{otherwise} \end{cases} . \tag{1}$$

where $\mathbf{M}_i^l$ is the attention mask for the i-th query in the l-th layer, $\sigma = \{(i,j) | i \in \{1, \cdots, N_a\}, j \in \{1, \cdots, N_{obj}\}\}$ is the matching of the last decoder layer, and $\mathbf{M}_j^{gt}$ is the matched ground truth mask for the i-th query in the last layer. Here N_{obj} denotes the number of ground truth targets. Then we use the replaced attention mask $\mathbf{M}^l$ combined with original output queries and pixel features of l-th layer, which are refined and for better guidance, as input to forward the l-th Transformer decoder layer again. The new output is supervised according to the fixed matching σ, consistent with the last layer's bipartite matching results. This fixed matching ensures the consistency of the predictions at each Transformer decoder layer and saves the matching computation cost during training. By such guided learning, we allow each query to see the whole region of its target predicted object during training, which helps the masked attention attend within a more appropriate foreground region.

3.6. Loss function

The overall loss function for FastInst can be written as:

$$\mathcal{L} = \mathcal{L}_{\text{IA-q}} + \mathcal{L}_{\text{pred}} + \mathcal{L}'_{\text{pred}} \tag{2}$$

where $\mathcal{L}_{\text{IA-q}}$ is the instance activation loss of the auxiliary classification head for IA-guided queries, $\mathcal{L}_{\text{pred}}$ and $\mathcal{L}'_{\text{pred}}$ are prediction loss and GT mask-guided loss, respectively.

Instance activation loss. The $\mathcal{L}_{\text{IA-q}}$ is defined as:

$$\mathcal{L}_{\text{IA-q}} = \lambda_{\text{cls-q}} \mathcal{L}_{\text{cls-q}} \tag{3}$$

where $\lambda_{\text{cls-q}}$ is a hyperparameter and $\mathcal{L}_{\text{cls-q}}$ is the cross-entropy loss with a weight $1/T$ for "no object" class. Here $T = (H/16) \times (W/16)$ is the spatial size of E_4 from which IA-guided queries are selected. We use the Hungarian algorithm [22] to search for the optimal bipartite matching between the prediction and ground truth sets. For the matching cost, we add an additional location cost $\mathcal{L}_{\text{loc}}$ of a weight λ_{loc} to the above classification cost, as illustrated in Section 3.3.

Prediction loss. Following the prior work [9], the prediction loss $\mathcal{L}_{\text{pred}}$ for the Transformer decoder is defined as:

$$\mathcal{L}_{\text{pred}} = \sum_{i=0}^{D} (\lambda_{\text{ce}} \mathcal{L}_{\text{ce}}^i + \lambda_{\text{dice}} \mathcal{L}_{\text{dice}}^i) + \lambda_{\text{cls}} \mathcal{L}_{\text{cls}}^i \tag{4}$$

where D denotes the number of Transformer decoder layers and $i = 0$ represents the prediction loss for IA-guided queries before being fed into the Transformer decoder, $\mathcal{L}_{\text{ce}}^i$ and $\mathcal{L}_{\text{dice}}^i$ denote the binary cross-entropy loss and dice loss [31] for segmentation masks, respectively, and $\mathcal{L}_{\text{cls}}$ is the cross-entropy loss for object classification with a "no object" weight of 0.1. λ_{ce}, λ_{dice}, and λ_{cls} are the hyperparameters to balance three losses. Similarly, we exploit the Hungarian algorithm to search for the best bipartite matching for target assignments. And for the matching cost, we add an additional location cost $\lambda_{\text{loc}} \mathcal{L}_{\text{loc}}$ for each query.

GT mask-guided loss. The GT mask-guided loss $\mathcal{L}'_{\text{pred}}$ is similar to Equation (4). The only differences are that it does not count the 0-th layer's loss and uses a fixed target assignment strategy, which is consistent with the bipartite matching result of the last Transformer decoder layer.

4. Experiments

In this section, we evaluate FastInst on COCO [27] and compare it with several state-of-the-art methods. We also conduct detailed ablation studies to verify the effectiveness of each proposed component.

4.1. Implementation details

Our model is implemented using Detectron2 [44]. We use the AdamW [30] optimizer with a step learning rate schedule. The initial learning rate is 0.0001, and the weight decay is 0.05. We apply a learning rate multiplier of 0.1 to the backbone, which is ImageNet-pretrained, and decay the learning rate by 10 at fractions 0.9 and 0.95 of the total number of training iterations. Following [9], we train our model for 50 epochs with a batch size of 16. For data

method	backbone	epochs	size	FPS	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
MEInst [46]	R50	36	512	26.4	32.2	53.9	33.0	13.9	34.4	48.7
CenterMask [23]	R50	48	600	32.6	32.9	-	-	12.9	34.7	48.7
SOLOv2 [42]	R50	36	448	41.3	34.0	54.0	36.1	10.3	36.3	54.4
OrienMask [14]	D53	100	544	51.0	34.8	56.7	36.4	16.0	38.2	47.8
SparseInst [11]	R50	144	608	51.2	34.7	55.3	36.6	14.3	36.2	50.7
YOLACT [1]	R50	54	550	53.5	28.2	46.6	29.2	9.2	29.3	44.8
FastInst-D1 (ours)	R50	50	576	**53.8**	35.6	56.7	37.2	8.8	53.0	72.8
CondInst [37]	R50	36	800	20.0	37.8	59.1	40.5	**21.0**	40.3	48.7
Mask2Former†	R50	50	640	25.3	38.0	**60.3**	39.8	10.8	54.9	74.3
FastInst-D3 (ours)	R50	50	640	35.5	**38.6**	60.2	**40.6**	10.8	**56.2**	**75.2**
YOLACT [1]	R101	54	700	33.5	31.2	50.6	32.8	12.1	33.3	47.1
FastInst-D1 (ours)	R101	50	640	**35.3**	38.3	60.2	40.5	10.7	55.8	74.8
SOLOv2 [42]	R101	36	800	15.7	39.7	60.7	**42.9**	17.3	42.9	57.4
CondInst [37]	R101	36	800	16.4	39.1	60.9	42.0	**21.5**	41.7	50.9
Mask2Former†	R101	50	640	21.1	39.5	**61.7**	41.6	11.2	56.3	75.8
FastInst-D3 (ours)	R101	50	640	28.0	**39.9**	61.5	42.3	11.4	**57.1**	**76.6**
SOLOv2 [42]	R50-DCN	36	512	32.0	37.1	57.7	39.7	12.9	40.0	57.4
YOLACT++ [2]	R50-DCN	54	550	39.4	34.1	53.3	36.2	11.7	36.1	53.6
SparseInst [11]	R50-d-DCN	144	608	46.5	37.9	59.2	40.2	**15.7**	39.4	56.9
FastInst-D1 (ours)	R50-d-DCN	50	576	**47.8**	38.0	59.7	39.9	10.0	54.9	74.5
FastInst-D3 (ours)	R50-d-DCN	50	640	32.5	**40.5**	**62.6**	**42.9**	11.9	**57.9**	**76.7**

Table 1. **Instance segmentation on COCO `test-dev`.** FastInst outperforms most previous real-time instance segmentation algorithms in both accuracy and speed. Mask2Former† denotes a light version of Mask2Former [9] that exploits PPM-FPN as the pixel decoder, as in FastInst and SparseInst [11]. FastInst-Dα represents FastInst with α Transformer decoder layers. "R50-d-DCN" means ResNet-50-d [17] backbone with deformable convolutions [49]. For a fair comparison, all entries are *single-scale* results.

augmentation, we use the same scale jittering and random cropping as in [11]. For example, the shorter edge varies from 416 to 640 pixels, and the longer edge is no more than 864 pixels. We set the loss weights λ_{cls}, λ_{ce}, and λ_{dice} to 2.0, 5.0, and 5.0, respectively, as in [9]. $\lambda_{\text{cls-q}}$ and λ_{loc} are set to 20.0 and 1000.0, respectively. We use 100 IA-guided queries and 8 auxiliary learnable queries by default. We report the AP performance as well as the FLOPs and FPS. FLOPs are averaged using 100 validation images. FPS is measured on a V100 GPU with a batch size of 1 using the entire validation set. Unless specified, we use a shorter edge of 640 pixels with a longer edge not exceeding 864 pixels to test and benchmark models.

4.2. Main results

We compare FastInst with state-of-the-art methods on the COCO dataset in Table 1. Since the goal of FastInst is for an efficient real-time instance segmenter, we mainly compare it with state-of-the-art real-time instance segmentation algorithms in terms of accuracy and inference speed. The evaluation is conducted on COCO `test-dev`. We provide FastInst with different backbones and different numbers of Transformer decoder layers to achieve a trade-off between speed and accuracy. The results show that FastInst outperforms most previous state-of-the-art real-time instance segmentation methods with better performance and faster speed. For example, with a ResNet-50 [16] backbone, the designed FastInst-D1 model outperforms a strong convolutional baseline SparseInst [11] by 0.9 AP while using fewer training epochs and less inference time. We also compare FastInst with the query-based model Mask2Former [9]. To keep the speed and accuracy in a similar order, we replace the MSDeformAttn [50] pixel decoder in Mask2Former with the PPM-FPN-based one, which is the same as FastInst as well as SparseInst [11]. Meanwhile, for a fair comparison, the training setting of Mask2Former, including data augmentation, is replaced with the same as FastInst. As expected, Mask2Former relies on a strong pixel decoder and performs worse than FastInst in both accuracy and speed with a lighter pixel decoder (even if it has 9 decoder layers), showing less efficiency in the real-time benchmark. Besides, with ResNet-50-d-DCN [17, 49] backbone, our algorithm achieves 32.5 FPS and 40.5 AP, the only algorithm in Table 1 with AP$>$40 while maintaining a real-time speed ($\geq$30 FPS). Figure 1 illustrates the speed-accuracy trade-off curve, which also demonstrates the superiority of our method.

4.3. Ablation studies

We now perform a series of ablation studies to analyze FastInst. We first verify the effectiveness of three proposed key components, *i.e.*, IA-guided queries, dual-path update strategy, and GT mask-guided learning, and then study the effect of some other designs about FastInst. Unless specified otherwise, we conduct experiments on FastInst-D3 with ResNet-50 [16] backbone. All ablation results are evaluated on the COCO `val2017` set.

IA-guided queries. As shown in Table 2, our IA-guided

	D	$\text{AP}^{\texttt{val}}$	AP_S	AP_M	AP_L	FLOPs	FPS
zero [4]	1	31.5	10.8	33.6	52.6	58.4G	50.0
learnable [9]	1	34.6	13.5	37.5	55.4	58.4G	50.0
resize [32]	1	34.9	13.7	37.9	56.2	58.4G	49.7
IA-guided	1	**35.6**	**14.3**	**38.8**	**56.6**	59.6G	48.8
zero [4]	3	37.2	15.4	40.3	58.6	74.3G	36.1
learnable [9]	3	37.5	15.0	40.6	59.0	74.3G	36.1
resize [32]	3	37.6	15.6	40.4	59.7	74.3G	36.0
IA-guided	3	**37.9**	**16.0**	**40.7**	**60.1**	75.5G	35.5

Table 2. **IA-guided queries.** Our IA-guided queries perform better than other methods, especially when the Transformer decoder layer number (*i.e.*, D) is small.

	D	$\text{AP}^{\texttt{val}}$	AP_S	AP_M	AP_L	FLOPs	FPS
single pixel feat. update	6	32.5	13.9	36.3	50.5	85.4G	35.5
single query update [9]	6	36.9	15.0	39.6	59.8	63.3G	35.0
dual query-then-pixel	3	37.8	**16.0**	40.6	60.0	75.5G	35.5
dual pixel-then-query	3	**37.9**	**16.0**	**40.7**	**60.1**	75.5G	35.5

Table 3. **Dual-path update strategy.** Our dual pixel-then-query update strategy consistently outperforms single-path update strategies. We double the Transformer decoder layers (*i.e.*, D) for the single-path update strategies for a fair comparison.

	backbone	$\text{AP}^{\texttt{val}}$	AP_S	AP_M	AP_L	FLOPs	FPS
w/o GT mask guidance	R50	37.4	15.2	40.6	59.6	75.5G	35.5
w/ GT mask guidance	R50	**37.9**	**16.0**	**40.7**	**60.1**	75.5G	35.5
w/o GT mask guidance	R50-d-DCN	39.7	17.5	43.1	61.9	77.9G	32.5
w/ GT mask guidance	R50-d-DCN	**40.1**	**17.7**	**43.2**	**62.4**	77.9G	32.5

Table 4. **GT mask-guided learning.** Our GT mask-guided learning improves the performance across different backbones.

	$\text{AP}^{\texttt{val}}$	AP_S	AP_M	AP_L	FLOPs	FPS
FPN [26]	37.4	15.5	40.3	59.4	75.4G	35.7
Transformer-Encoder [4]	38.9	17.3	41.6	61.4	78.5G	29.5
MSDeformAttn [50]	**40.0**	**17.5**	**43.5**	**62.2**	114.7G	21.2
PPM-FPN [11]	37.9	16.0	40.7	60.1	75.5G	35.5

Table 5. **Pixel decoder.** Stronger pixel decoders lead to higher performance but consume more computation. PPM-FPN obtains a better trade-off between accuracy and speed.

queries achieve better results than zero [4] or learning-based ones [9]. Recent work [32] proposes to use resized multi-scale features as instance queries. However, such a fix-position query selection strategy is hard to extract representative embeddings for all potential objects and thus obtains lower performance. Note that IA-guide queries achieve a more significant result when the model is equipped with only one Transformer decoder layer, which shows their great efficiency in lightweight model design.

Dual-path update strategy. Table 3 shows the effectiveness of our dual-path update strategy. Thanks to the co-optimization of query and pixel features, our dual-path update strategy performs better than the conventional single query update strategy [9] in our lightweight pixel decoder setting. The update order of query and pixel features does not matter much in our experiments.

GT mask-guided learning. As shown in Table 4, GT

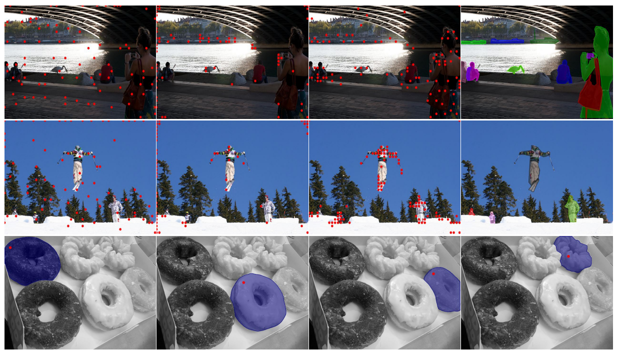

Figure 3. **Visualization of IA-guided queries.** The **first and second rows** show the distributions of IA-guided queries with different supervision losses for the auxiliary classification head. First column: dense pixel-wise semantic classification loss. Second column: bipartite matching-based Hungarian loss without the location cost. Third column: bipartite matching-based Hungarian loss with the location cost (ours). Fourth column: ground truth. The query points under our designed loss (third column) are more concentrated on the region of each foreground object. The **last row** shows four predicted masks (**blue**) with corresponding IA-guided query locations (**red**).

mask-guided learning improves the model performance by up to 0.5 AP, indicating that this technique indeed helps the Transformer decoder learn how to update queries for better object embeddings under masked attention mechanism. Table 4 also demonstrates the generality of GT mask-guided learning for different backbones.

The above ablations demonstrate the effectiveness of our proposed three key techniques. We refer interested readers to the Appendix for more ablations about them, *e.g.*, the changes and the corresponding improvements based on the original Mask2Former. We then explore the effect of some other designs about FastInst.

Pixel decoder. FastInst is compatible with any existing pixel decoders. Table 5 shows the performance of FastInst with different pixel decoders. Stronger pixel decoders produce better contextual features and lead to higher results but consume more computation. For fast real-time instance segmentation, PPM-FPN [11] is a good trade-off choice.

Transformer decoder layer number. As shown in Table 6a, increasing the number of Transformer decoder layers contributes to the segmentation performance in FastInst. In particular, the mask performance achieves 30.5 AP without using the Transformer decoder. This is mainly attributed to the effectiveness of IA-guided queries, which carry rich embedding information about potential objects at the initial. In addition, our segmentation performance is saturated at around the sixth layer. Continuing to increase decoder layers only marginally improve it. Also note that FastInst can obtain good performance with only a few Transformer decoder layers, which is advantageous in real-time.

D	AP^{val}	AP_S	AP_M	AP_L	FLOPs	FPS
0	30.5	12.1	34.5	48.6	51.6G	60.2
1	35.6	14.3	38.8	56.6	59.6G	48.8
2	37.1	15.8	39.7	58.8	67.5G	41.0
3	37.9	16.0	40.7	60.1	75.5G	35.5
6	38.7	**16.6**	41.7	61.1	99.3G	25.1
9	**39.1**	16.1	**42.3**	**62.1**	123.2G	19.2

(a) **Transformer decoder layer number.** FastInst benefits from more Transformer decoder layers.

N_a	AP^{val}	AP_S	AP_M	AP_L	FLOPs	FPS
10	31.2	10.2	32.9	54.2	71.9G	39.8
50	36.9	14.5	39.6	58.5	73.5G	37.7
100	37.9	16.0	40.7	60.1	75.5G	35.5
200	38.5	16.2	41.3	**60.8**	79.5G	31.6
300	**38.8**	**17.5**	**42.0**	60.2	83.5G	28.6

(b) **Effect of IA-guided query number on AP.** Increasing IA-guided query number contributes to AP performance.

N_a	AR^{val}	AR_S	AR_M	AR_L
10	38.2	14.4	40.5	62.8
50	49.8	26.2	53.3	72.6
100	52.0	28.9	55.3	74.2
200	53.4	30.5	57.4	74.9
300	**54.0**	**31.3**	**57.6**	**75.3**

(c) **Effect of IA-guided query number on AR@100.** Larger IA-guided query number improves object recalls, especially for small objects.

N_b	N_a	AP	AP_S	AP_M	AP_L	FLOPs	FPS
0	100	37.7	15.7	40.4	**60.3**	75.2G	35.6
0	108	37.6	15.6	40.7	59.9	75.6G	35.3
8	100	**37.9**	16.0	40.7	60.1	75.5G	35.5
16	100	37.8	**16.1**	**40.9**	59.9	75.7G	35.1

(d) **Auxiliary learnable query number.** Adding a few (*i.e.*, 8) auxiliary learnable queries performs better than setting all queries as IA-guided ones.

	AP	AP_S	AP_M	AP_L	FLOPs	FPS
E_5	37.8	15.5	40.7	60.2	74.6G	35.6
E_4	37.9	16.0	40.7	60.1	75.5G	35.5
E_3	**38.0**	**16.3**	**41.2**	**60.6**	79.1G	34.7

(e) **Query selection source.** Selecting IA-guided queries from larger feature maps leads to better results, but the gain is limited. E_4 is a trade-off choice between accuracy and speed.

	AP	AP_S	AP_M	AP_L	FLOPs	FPS
Baseline	**37.9**	**16.0**	**40.7**	**60.1**	75.5G	35.5
− bi. matching	35.7	14.2	38.1	57.5	75.5G	35.5
− loc. cost	37.1	14.9	40.0	59.3	75.5G	35.5

(f) **Instance activation loss.** We remove one component at a time. When removing the bipartite matching strategy, we use a fixed target assignment for each pixel according to their semantic class labels.

Table 6. **Several ablations for FastInst**. Results are evaluated on COCO `val2017`.

	AP^{val}	AP_S	AP_M	AP_L	FLOPs	FPS
FastInst-D3	37.9	16.0	40.7	60.1	75.5G	35.5
− learnable pos. embeddings	37.9	16.4	40.6	**60.3**	75.5G	32.9

Table 7. **Positional embeddings.** When removing learnable positional embeddings, we use the non-parametric sinusoidal positional embeddings, as in [4, 9]

IA-guided query number. In Mask2Former, increasing the query number to more than 100 will slightly degrade the instance segmentation performance. In Table 6b, the results indicate that increasing the number of IA-guided queries will contribute to the segmentation performance in FastInst. We attribute this to the improved object recall (see Table 6c) and increased object embedding information for decoding. On the other hand, growing IA-guided queries will affect the model inference speed. Note that even with 10 IA-guided queries, our model can obtain 31.2 AP on COCO dataset, which has 7.7 instances per image on average [27]. This indicates the effectiveness of IA-guided queries again.

Auxiliary learnable query number. Auxiliary queries aim to gather background and image-independent information for pixel feature updates and query updates. They do not participate in object predictions. Table 6d shows that adding a few auxiliary learnable queries helps improve the performance, better than setting all queries as IA-guided queries.

Query selection source. As shown in Table 6e, selecting IA-guided queries from larger feature maps leads to better results. E_4 is a good trade-off choice between accuracy and speed. Nevertheless, the contribution of the selection source to the model performance is limited.

Instance activation loss. We study the effect of two components in instance activation loss. As shown in Table 6f, the bipartite matching-based target assignment strategy leads to a significant gain, which provides a sparse pixel embedding activation for IA-guided query selection. Here when removing the bipartite matching strategy, we use the semantic class label as the target of each pixel, as in common semantic segmentation tasks [6, 7]. The location cost also plays a vital role in the matching loss, which reduces matching space and accelerates model convergence. Figure 3 visualizes the distributions of IA-guided queries, which also shows the superiority of our designed loss.

Positional embeddings. Table 7 demonstrates that using learnable positional embeddings instead of non-parametric sinusoidal positional embeddings can improve the model inference speed without compromising the performance.

5. Conclusion

We propose FastInst for real-time instance segmentation. Built on a query-based segmentation framework [9] and three designed efficient components, *i.e.*, instance activation-guided queries, dual-path update strategy, and ground truth mask-guided learning, FastInst achieves excellent performance on the popular COCO dataset while maintaining a fast inference speed. Extensive experiments demonstrate the effectiveness of core ideas and the superiority of FastInst over previous state-of-the-art real-time counterparts. We hope this work can serve as a new baseline for real-time instance segmentation and promote the development of query-based image segmentation algorithms.

Limitations. (1) Like general query-based models [4,9,25], FastInst is not good at small targets. Even though using stronger pixel decoders or larger feature maps improves it, it introduces heavier computational burdens, and the result is still unsatisfactory. We look forward to an essential solution to handle this problem. (2) although GT mask-guided learning improves the performance of masked attention, it increases training costs. We hope a more elegant method can be proposed to replace it.

References

[1] Daniel Bolya, Chong Zhou, Fanyi Xiao, and Yong Jae Lee. Yolact: Real-time instance segmentation. In *ICCV*, 2019. 6

[2] Daniel Bolya, Chong Zhou, Fanyi Xiao, and Yong Jae Lee. Yolact++: Better real-time instance segmentation. *PAMI*, 2020. 1, 2, 6

[3] Zhaowei Cai and Nuno Vasconcelos. Cascade R-CNN: Delving into high quality object detection. In *CVPR*, 2018. 2

[4] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *ECCV*, 2020. 1, 2, 3, 4, 7, 8

[5] Kai Chen, Jiangmiao Pang, Jiaqi Wang, Yu Xiong, Xiaoxiao Li, Shuyang Sun, Wansen Feng, Ziwei Liu, Jianping Shi, Wanli Ouyang, et al. Hybrid task cascade for instance segmentation. In *CVPR*, 2019. 1, 2

[6] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. DeepLab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected CRFs. *PAMI*, 2018. 3, 8

[7] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. In *ECCV*, 2018. 8

[8] Xinlei Chen, Ross Girshick, Kaiming He, and Piotr Dollár. TensorMask: A foundation for dense object segmentation. In *ICCV*, 2019. 1

[9] Bowen Cheng, Ishan Misra, Alexander G. Schwing, Alexander Kirillov, and Rohit Girdhar. Masked-attention mask transformer for universal image segmentation. In *CVPR*, 2022. 1, 2, 3, 4, 5, 6, 7, 8

[10] Bowen Cheng, Alexander G. Schwing, and Alexander Kirillov. Per-pixel classification is not all you need for semantic segmentation. In *NeurIPS*, 2021. 1, 2, 3

[11] Tianheng Cheng, Xinggang Wang, Shaoyu Chen, Wenqiang Zhang, Qian Zhang, Chang Huang, Zhaoxiang Zhang, and Wenyu Liu. Sparse instance activation for real-time instance segmentation. In *CVPR*, 2022. 2, 3, 6, 7

[12] Tianheng Cheng, Xinggang Wang, Lichao Huang, and Wenyu Liu. Boundary-preserving mask r-cnn. In *ECCV*, 2020. 2

[13] Bin Dong, Fangao Zeng, Tiancai Wang, Xiangyu Zhang, and Yichen Wei. Solq: Segmenting objects by learning queries. In *NeurIPS*, 2021. 2

[14] Wentao Du, Zhiyu Xiang, Shuya Chen, Chengyu Qiao, Yiman Chen, and Tingming Bai. Real-time instance segmentation with discriminative orientation maps. In *ICCV*, 2021. 6

[15] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask R-CNN. In *ICCV*, 2017. 1, 2

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 1, 2, 6

[17] Tong He, Zhi Zhang, Hang Zhang, Zhongyue Zhang, Junyuan Xie, and Mu Li. Bag of tricks for image classification with convolutional neural networks. In *CVPR*, 2019. 6

[18] Jie Hu, Liujuan Cao, Yao Lu, ShengChuan Zhang, Yan Wang, Ke Li, Feiyue Huang, Ling Shao, and Rongrong Ji. Istr: End-to-end instance segmentation with transformers. *arXiv:2105.00637*, 2021. 2

[19] Zhaojin Huang, Lichao Huang, Yongchao Gong, Chang Huang, and Xinggang Wang. Mask scoring R-CNN. In *CVPR*, 2019. 1

[20] Alexander Kirillov, Ross Girshick, Kaiming He, and Piotr Dollár. Panoptic feature pyramid networks. In *CVPR*, 2019. 3

[21] Alexander Kirillov, Yuxin Wu, Kaiming He, and Ross Girshick. PointRend: Image segmentation as rendering. In *CVPR*, 2020. 2

[22] Harold W Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 1955. 5

[23] Youngwan Lee and Jongyoul Park. Centermask: Real-time anchor-free instance segmentation. In *CVPR*, 2020. 1, 6

[24] Feng Li, Hao Zhang, Huaizhe xu, Shilong Liu, Lei Zhang, Lionel M. Ni, and Heung-Yeung Shum. Mask dino: Towards a unified transformer-based framework for object detection and segmentation. *arXiv:2206.02777*, 2022. 2

[25] Zhiqi Li, Wenhai Wang, Enze Xie, Zhiding Yu, Anima Anandkumar, Jose M. Alvarez, Ping Luo, and Tong Lu. Panoptic segformer: Delving deeper into panoptic segmentation with transformers. In *CVPR*, 2022. 1, 2, 8

[26] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *CVPR*, 2017. 3, 7

[27] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft COCO: Common objects in context. In *ECCV*, 2014. 2, 5, 8

[28] Shu Liu, Lu Qi, Haifang Qin, Jianping Shi, and Jiaya Jia. Path aggregation network for instance segmentation. In *CVPR*, 2018. 1

[29] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *CVPR*, 2015. 1

[30] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *ICLR*, 2019. 5

[31] Fausto Milletari, Nassir Navab, and Seyed-Ahmad Ahmadi. V-Net: Fully convolutional neural networks for volumetric medical image segmentation. In *3DV*, 2016. 5

[32] Jialun Pei, Tianyang Cheng, Deng-Ping Fan, He Tang, Chuanbo Chen, and Luc Van Gool. Osformer: One-stage camouflaged instance segmentation with transformers. In *ECCV*, 2022. 7

[33] Joseph Redmon and Ali Farhadi. YOLOv3: An Incremental Improvement. *arXiv:1804.02767*, 2018. 1

[34] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *NeurIPS*, 2015. 2

[35] Russell Stewart and Mykhaylo Andriluka. End-to-end people detection in crowded scenes. In *CVPR*, 2016. 3

[36] Chufeng Tang, Hang Chen, Xiao Li, Jianmin Li, Zhaoxiang Zhang, and Xiaolin Hu. Look closer to segment better: Boundary patch refinement for instance segmentation. In *CVPR*, 2021. 2

[37] Zhi Tian, Chunhua Shen, and Hao Chen. Conditional convolutions for instance segmentation. In *ECCV*, 2020. 2, 4, 6

[38] Zhi Tian, Chunhua Shen, Hao Chen, and Tong He. FCOS: A simple and strong anchor-free object detector. *PAMI*, 2021. 2

[39] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, 2017. 1, 2

[40] Huiyu Wang, Yukun Zhu, Hartwig Adam, Alan Yuille, and Liang-Chieh Chen. MaX-DeepLab: End-to-end panoptic segmentation with mask transformers. In *CVPR*, 2021. 3

[41] Xinlong Wang, Tao Kong, Chunhua Shen, Yuning Jiang, and Lei Li. SOLO: Segmenting objects by locations. In *ECCV*, 2020. 2, 4

[42] Xinlong Wang, Rufeng Zhang, Tao Kong, Lei Li, and Chunhua Shen. SOLOv2: Dynamic and fast instance segmentation. *NeurIPS*, 2020. 1, 2, 3, 4, 6

[43] Jialian Wu, Sudhir Yarram, Hui Liang, Tian Lan, Junsong Yuan, Jayan Eledath, and Gerard Medioni. Efficient video instance segmentation via tracklet query and proposal. In *CVPR*, 2022. 1

[44] Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick. Detectron2. `https://github.com/facebookresearch/detectron2`, 2019. 5

[45] Qihang Yu, Huiyu Wang, Dahun Kim, Siyuan Qiao, Maxwell Collins, Yukun Zhu, Hartwig Adam, Alan Yuille, and Liang-Chieh Chen. Cmt-deeplab: Clustering mask transformers for panoptic segmentation. In *CVPR*, 2022. 3

[46] Rufeng Zhang, Zhi Tian, Chunhua Shen, Mingyu You, and Youliang Yan. Mask encoding for single shot instance segmentation. In *CVPR*, 2020. 2, 6

[47] Tao Zhang, Shiqing Wei, and Shunping Ji. E2ec: An end-to-end contour-based method for high-quality high-speed instance segmentation. In *CVPR*, 2022. 2

[48] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network. In *CVPR*, 2017. 1, 3

[49] Xizhou Zhu, Han Hu, Stephen Lin, and Jifeng Dai. Deformable convnets v2: More deformable, better results. In *CVPR*, 2019. 6

[50] Xizhou Zhu, Weijie Su, Lewei Lu, Bin Li, Xiaogang Wang, and Jifeng Dai. Deformable detr: Deformable transformers for end-to-end object detection. In *ICLR*, 2021. 1, 3, 6, 7