You Only Segment Once: Towards Real-Time Panoptic Segmentation

Jie Hu, Linyan Huang, Tianhe Ren, Shengchuan Zhang, Rongrong Ji, and Liujuan Cao*
Key Laboratory of Multimedia Trusted Perception and Efficient Computing,
Ministry of Education of China, Xiamen University

Abstract

In this paper, we propose YOSO, a real-time panoptic segmentation framework. YOSO predicts masks via dynamic convolutions between panoptic kernels and image feature maps, in which you only need to segment once for both instance and semantic segmentation tasks. To reduce the computational overhead, we design a feature pyramid aggregator for the feature map extraction, and a separable dynamic decoder for the panoptic kernel generation. The aggregator re-parameterizes interpolation-first modules in a convolution-first way, which significantly speeds up the pipeline without any additional costs. The decoder performs multi-head cross-attention via separable dynamic convolution for better efficiency and accuracy. To the best of our knowledge, YOSO is the first real-time panoptic segmentation framework that delivers competitive performance compared to state-of-the-art models. Specifically, YOSO achieves 46.4 PQ, 45.6 FPS on COCO; 52.5 PQ, 22.6 FPS on Cityscapes; 38.0 PQ, 35.4 FPS on ADE20K; and 34.1 PQ, 7.1 FPS on Mapillary Vistas. Code is available at https://github.com/hujiecpp/YOSO.

1. Introduction

Panoptic segmentation is a task that involves assigning a semantic label and an instance identity to each pixel of an input image. The semantic labels are typically classified into two types, i.e., stuff including amorphous and uncountable concepts (such as sky and road), and things consisting of countable categories (such as persons and cars). This division of label types naturally separates panoptic segmentation into two sub-tasks: semantic segmentation for stuff and instance segmentation for things. Thus, one of the major challenges for achieving real-time panoptic segmentation is the requirement for separate and computationally intensive branches to perform semantic and instance segmentation respectively. Typically, instance segmentation employs boxes or points to distinguish between different things, while semantic segmentation predicts distribution maps over semantic categories for stuff. As shown in Fig. 1, numerous efforts [9, 16, 23, 24, 31, 45] have been made to unify panoptic segmentation pipelines for improved speed and accuracy. However, achieving real-time panoptic segmentation still remains an open problem. On the one hand, heavy necks, e.g., the multi-scale feature pyramid network (FPN) used in [27, 54], and heads, e.g., the Transformer decoder used in [10, 58], are required to ensure accuracy, making real-time processing unfeasible. On the other hand, reducing the model size [16, 23, 24] leads to a decrease in model generalization. Therefore, developing a real-time panoptic segmentation framework that delivers competitive accuracy is challenging yet highly desirable.

In this paper, we present YOSO, a real-time panoptic segmentation framework. YOSO predicts panoptic kernels to convolute image feature maps, with which you only need to segment once for the masks of background stuff and foreground things. To make the process lightweight, we design a feature pyramid aggregator for extracting image feature maps, and a separable dynamic decoder for generating panoptic kernels. In the aggregator, we propose convolution-first aggregation (CFA) to re-parameterize the interpolation-first aggregation (IFA), resulting in an approximately 2.6× speedup in GPU latency without compromising performance. Specifically, we demonstrate that the order, i.e., interpolation-first or convolution-first, of applying bilinear interpolation and 1×1 convolution (w/o bias) does not affect results, but the convolution-first way provides a considerable speedup to the pipeline. In the decoder, we propose separable dynamic convolution attention (SDCA) to perform multi-head cross-attention in a weight-sharing way. SDCA achieves better accuracy (+1.0 PQ) and higher efficiency (approximately 1.2× faster GPU latency) than traditional multi-head cross-attention.

In general, YOSO has three notable advantages. First, CFA reduces computational burden without re-training the model or compromising performance. CFA can be adapted to any task that uses the combination of bilinear interpolation and 1×1 convolution operations. Second, SDCA performs multi-head cross-attention with better accuracy and...
efficiency. Third, YOSO runs faster and has competitive accuracy compared to state-of-the-art panoptic segmentation models, and its generalization is validated on four popular datasets: COCO (46.4 PQ, 45.6 FPS), Cityscapes (52.5 PQ, 22.6 FPS), ADE20K (38.0 PQ, 35.4 FPS), and Mapillary Vistas (34.1 PQ, 7.1 FPS).

2. Related Work

Real-Time Panoptic Segmentation. Panoptic segmentation aims to jointly perform semantic and instance segmentation, where each pixel in an input image is assigned both a semantic label and a unique instance identity. Many studies have been conducted for fast panoptic segmentation [8, 9, 16, 18, 23, 24, 31, 37, 41, 42, 45, 54, 55]. For instance, UPSNet [54] utilizes a deformable convolution based semantic segmentation head and a Mask R-CNN [21] style instance segmentation head. FPSNet [16] proposes a fast architecture for panoptic segmentation, avoiding instance mask prediction and merging outputs via soft attention masks. Recently, PanopticDeepLab [9], LPSNet [23], and RealTimePan [24] generate semantic masks for all categories first, then locate masks of instances via boxes or points, enabling efficient object segmentation. Meanwhile, PanopticFCN [31], K-Net [58], and MaskFormer [10, 11] attempt to predict masks for both things and stuff simultaneously through dynamic convolutions. Despite significant advances in this field, achieving real-time panoptic segmentation remains an open problem. In this paper, YOSO enables real-time panoptic segmentation with competitive accuracy by utilizing the proposed feature pyramid aggregator and separable dynamic decoder.

Real-Time Instance Segmentation. Instance segmentation aims to predict masks and classes for each instance in an image. To achieve real-time instance segmentation, various approaches have been proposed in recent literature. YOLACT [2, 3] proposes to multiply the predicted mask coefficients with prototype masks, and SipMask [5] utilizes spatial mask coefficients for more accurate segmentation. CenterMask [30] employs an efficient anchor-free framework, and DeepSnake [40] explores the use of object contours for fast segmentation of instances. OrienMask [19] designs discriminative orientation maps that recover the masks without additional foreground segmentation, and SOLO [49, 50] segments objects by locations, with a decoupled branch to speed up the framework. Recently, SparseInst [12] introduced a sparse set of instance activation maps that highlight informative regions for each object in an image, constructing a real-time instance segmentation framework. As complex operations are required to distinguish different things, solving instance segmentation efficiently is the key to real-time panoptic segmentation. In this paper, YOSO predicts unified panoptic kernels for stuff and things. Implementing bipartite matching loss [6] for fast discrimination of different things, YOSO avoids time-consuming object localization operations such as RoIAlign [21] and post-processes such as non-maximum suppression. The output masks naturally represent independent instances for the categories of things. Moreover, the experimental results in our supplementary material show that YOSO can also achieve competitive performance on real-time instance segmentation.

Real-Time Semantic Segmentation. Semantic segmentation aims to predict pixel-wise categories for input images. In recent years, many approaches have been developed to enable real-time semantic segmentation. For example, E-Net [39] proposes a lightweight architecture for high-speed segmentation, and SegNet [1] combines a small network architecture with skip connections to achieve fast segmentation. ICNet [59] uses an image cascade algorithm to speed up the pipeline, and ESPNet [35, 36] introduces an efficient spatial pyramid dilated convolution. Additionally, BiSeNet [56, 56] separates spatial details and categorical semantics to enable both high accuracy and high efficiency.
in semantic segmentation. More recently, SegFormer [53] employs Transformers with a lightweight multi-layer perceptron decoder for fast semantic segmentation. In contrast to traditional methods that predict distribution maps over classes for semantic masks, YOSO predicts kernels with corresponding categories for segmentation. This enables an efficient way to jointly solve semantic and instance segmentation for panoptic segmentation.

3. Method
3.1. Task Formulation

Unified Panoptic Segmentation. Panoptic segmentation maps each pixel of an image to a semantic class and an instance identity. We propose a unified approach that considers the set of foreground and background classes as a single entity. To achieve this, we aim to predict a binary masks $M \in \mathbb{R}^{n \times h \times w}$ with class probabilities $L \in \mathbb{R}^{n \times l}$ for an input image, where $(h, w)$ denotes the mask resolution and $l$ is the total number of categories. During training, the predictions are matched with the corresponding ground truth labels using the set prediction loss [6, 10]. At test time, the segmentation results are merged in terms of the foreground (i.e., things) and background (i.e., stuff) classes. Concretely, the masks that correspond to the same background class are merged via union operation; the masks with foreground classes are treated as independent instances; if a pixel belongs to multiple classes, the class with the highest probability is assigned to that pixel.

YOSO Framework. As shown in Fig. 1, YOSO is a compact framework designed for real-time panoptic segmentation, which consists of a feature pyramid aggregator and a separable dynamic decoder. The backbone network, such as ResNet [22], extracts multi-level feature maps from input images. The feature pyramid aggregator compresses and aggregates the multi-level feature maps into single-level. The separable dynamic decoder then generates panoptic kernels with the single-level feature maps for both mask prediction and classification.

3.2. Feature Pyramid Aggregator

Deformable Feature Pyramid. Given the multi-level feature maps $C_2, C_3, C_4, C_5$ extracted by the backbone, we utilize the deformable feature pyramid network [15, 33] to enhance the feature maps from different scales. As depicted in Fig. 2, we first apply $1 \times 1$ convolutional layers to compress the channels of the multi-level feature maps. Then, the feature maps from $C_3$ to $C_5$ are fed into deformable convolutional networks (DCNs) and upsampled level-by-level, yielding the pyramid feature maps $P_2 \in \mathbb{R}^{c_2 \times h/2 \times w/2}, P_3 \in \mathbb{R}^{c_3 \times h/4 \times w/4}$, and $P_4 \in \mathbb{R}^{c_4 \times h/8 \times w/8}$, respectively, where $c_2, c_3, c_4, c_5$ denote the channel dimensions, and $h, w$ represent the 1/4 scale of the input images. After obtaining the pyramid feature maps, we explore two aggregation methods, namely interpolation-first aggregation (IFA) and convolution-first aggregation (CFA), to merge the multi-level feature maps.

Interpolation/Convolution-First Aggregation. In IFA, the pyramid feature maps are first upsampled to the scale of $h \times w$ via bilinear interpolation. Then, the feature maps are concatenated and fused using a $1 \times 1$ convolutional layer. In CFA, the pyramid feature maps are first fed to different $1 \times 1$ convolutional layers. Then, the feature maps are bilinearly interpolated to the scale of $h \times w$ and summed.

Observation 1: The output of IFA is exactly equal to that of CFA when using $1 \times 1$ convolution without bias.

This observation is attributed to the homogeneity and additivity properties of the bilinear interpolation function $f(\cdot)$, where $f\left(\sum_i w_i v_i^{x,y}\right) = \sum_i w_i f(v_i^{x,y})$ for the constant $w_i$ and the value vector $v_i^{x,y}$ from the four positions $(x_1, y_1), (x_1, y_2), (x_2, y_1), (x_2, y_2)$ of the $i$-th feature map. Specifically, the bilinear interpolation estimates the value at $(x_0, y_0)$ using the four positions by:

$$f\left(\sum_i w_i v_i^{x,y}\right) = \frac{1}{(x_2 - x_1)(y_2 - y_1)} \begin{bmatrix} x_2 - x_0 \\ y_2 - y_0 \\ y_0 - y_1 \\ 0 \end{bmatrix}^T \begin{bmatrix} \sum_i w_i v_i^{x_1,y_1} \\ \sum_i w_i v_i^{x_1,y_2} \\ \sum_i w_i v_i^{x_2,y_1} \\ \sum_i w_i v_i^{x_2,y_2} \end{bmatrix} = \sum_i w_i f(v_i^{x,y})$$

where $w_i$ can be interpreted as a $1 \times 1$ kernel that convolutes the values of the original positions in the feature maps. Eq. 1 implies that applying $1 \times 1$ convolution (without bias) before or after bilinear interpolation does not affect the final results. Hence, it can be inferred that when using $1 \times 1$
convolution without bias in the aggregators, IFA and CFA produce identical outputs.

**Observation II:** CFA requires significantly fewer floating point operations (FLOPs) than IFA.

The reduction ratio of FLOPs between IFA and CFA is:
\[
\frac{4(c_5 + c_4 + c_3)hw + (c_5 + c_4 + c_3 + c_2)dhw}{\left(\frac{c_5}{10} + \frac{c_4}{10} + \frac{c_3}{4} + c_2\right)dhw + 12dhw + 3dhw} \geq 1, \tag{2}
\]
where \(d\) is the channel dimension of output feature maps. In the numerator of Eq. 2 (i.e., the FLOPs of IFA), the first term represents the number of FLOPs used in bilinear interpolation, and the second term represents the number of FLOPs used in \(1 \times 1\) convolution. In the denominator of Eq. 2 (i.e., the FLOPs of CFA), the terms represent the number of FLOPs for \(1 \times 1\) convolution, bilinear interpolation, and accumulation, respectively.

Given the above two observations, we adopt CFA in the proposed feature pyramid aggregator. It is noteworthy that the learned weights of the \(1 \times 1\) convolutional layer in IFA can be readily re-parameterized to CFA by dividing the weights into four \(1 \times 1\) convolutional layers. This can accelerate the pipeline without incurring any additional costs.

### 3.3. Separable Dynamic Decoder

In order to generate accurate kernels for segmentation, previous methods typically relied on dense predictors [31] or heavy Transformer decoders [58]. In contrast, we propose a lightweight kernel generator called the separable dynamic decoder, which speeds up kernel generation while maintaining high accuracy. The separable dynamic decoder, shown in Fig. 3, consists of three modules: a pre-attention module, a separable dynamic convolution module, and a post-attention module. Specifically, the separable dynamic convolution efficiently performs multi-head cross-attention and achieves better accuracy. We describe each module in detail below.

**Pre-Attention.** Inspired by [58], the pre-attention module selectively extracts key information from the aggregated feature maps to diversify the proposal kernels. Concretely, the aggregated feature maps \(S \in \mathbb{R}^{d \times h \times w}\) are convoluted by the learnable proposal kernels \(Q \in \mathbb{R}^{n \times d}\) to produce attention maps \(A \in \mathbb{R}^{n \times h \times w}\). A hard sigmoid function \(\sigma(\cdot)\) is applied to the attention maps to activate the input values and discretize the values to either 0 or 1 using a threshold of 0.5. Then, with the attention maps, the masked features \(V \in \mathbb{R}^{n \times d}\) can be obtained by:
\[
V = r(\sigma(S \ast Q))r(S)^T = r(\sigma(A))r(S)^T, \tag{3}
\]
where \(r(\cdot)\) reshapes \(A\) and \(S\) to the sizes of \((n, hw)\) and \((d, hw)\), respectively, \(\ast\) denotes 2D convolution operation.

**Vanilla Dynamic Convolution.** To increase the model capacity, multi-head cross-attention [46] has been widely used, which projects input features into multiple spaces to model their relationships. For instance, given \(Q, V \in \mathbb{R}^{n \times d}\) which contains \(n\) tokens with the hidden dimension of \(d\), a \(t\)-head cross-attention can be defined as follows:
\[
\text{MultiHeadCrossAtten}(Q, V) = \text{Concat}(H_1, H_2, \ldots, H_t)W^o, \tag{4}
\]
where \(W^o \in \mathbb{R}^{d \times d}\) is a linear transformation matrix. The attention operation inside each head is defined as:
\[
H_i = \text{softmax}(QW_i^q(VW_i^k)^T)(VW_i^v) = K_i(VW_i^v) = K_iV_i, \tag{5}
\]
where \(W_i^q, W_i^k, W_i^v \in \mathbb{R}^{d \times d/t}\) are the projection matrices, \(V_i \in \mathbb{R}^{n \times d/t}\) denotes the projected features, and \(K_i \in \mathbb{R}^{n \times n}\) denotes the correlation matrix.

Although increasing the model capacity enhances performance, it also results in high computational burden. Intuitively, multi-head cross-attention involves three fundamental operations: multi-head projection, cross-token interaction, and cross-dimension interaction. In Eq. 4, the cross-dimension interaction, represented by \(W^o\), learns to re-weight the importance of every hidden dimension. In Eq. 5, the multi-head projection, represented by \(VW_i^v\), maps the hidden dimensions \(d\) into \(t\) different spaces with the size of \(d/t\), and the cross-token interaction, represented by \(K_iV_i\), uses the correlation matrix to interact between tokens. These operations motivated us to perform multi-head cross-attention using 1D convolution to make the process lightweight, which is defined as:
\[
\text{Conv}_{1d}(K, V) = V \ast K, \tag{6}
\]
where \( K \in \mathbb{R}^{n \times n \times t} \) denotes the weights of kernels to convolute \( V \in \mathbb{R}^{n \times d} \). \( \ast \) denotes 1D convolution operation. The 1D convolution computes the element of the output \( O \in \mathbb{R}^{n \times d} \) at the position \( i, j \) by:

\[
O_{i,j} = \sum_{p=1}^{n} \sum_{q=1}^{t} K_{i,p,q} \cdot V_{p,j+q-1}.
\]  

(7)

Correspondingly, the basic operations of multi-head cross-attention are also performed in 1D convolution in a weight-sharing manner. For the multi-head projection, the sliding window in 1D convolution densely splits the hidden dimensions into \( d \) groups of size \( t \), and the \( t \) successive hidden dimensions in each group are projected with shared kernels. For the cross-token interaction, the first accumulation term in Eq. 7 interacts \( n \) tokens by \( n \) different kernels. For the cross-dimension interaction, the second accumulation term in Eq. 7 locally incorporates the information from \( t \) successive hidden dimensions, instead of using all hidden dimensions globally in multi-head cross-attention. Furthermore, inspired by [13, 26, 29, 48, 51, 57], we employ a dynamic approach to generate the kernel \( K \) conditioned on \( Q \), which introduces the cross-attention mechanism into 1D convolution and defines a dynamic convolution attention as follows:

\[
DyConvAtten(Q, V) = V \ast r(QW),
\]  

(8)

where \( W \in \mathbb{R}^{d \times n \times t} \) is the projection matrix for generating the dynamic kernels.

**Separable Dynamic Convolution.** The standard convolution can be further decomposed into a depthwise convolution and a pointwise convolution as in [25]. Following this approach, we propose a separable form for vanilla dynamic convolution as follows:

\[
SepDyConvAtten(Q, V) = (V \ast r(QW^d)) \ast r(QW^p),
\]  

(9)

where \( W^d \in \mathbb{R}^{d \times t} \), \( W^p \in \mathbb{R}^{d \times n} \) project \( Q \) to generate the kernels for depthwise convolution and pointwise convolution. The kernels are then reshaped to \( (n, 1, t) \) and \( (n, n, 1) \), respectively. In Eq. 9, the depthwise convolution models the cross-dimension interaction, while the pointwise convolution models the cross-token interaction. Compared to multi-head cross-attention, the reduction in FLOPs is:

\[
\frac{4nd^2 + 2n^2d}{2ndt + 2n^2d} = \frac{2d + n}{k + n} > 1.
\]  

(10)

In the numerator of Eq. 10 (i.e., the FLOPs of MHCA), the first term denotes the number of FLOPs used in multi-head projection, while the second term represents the number of FLOPs used in cross-attention. In the denominator of Eq. 10 (i.e., the FLOPs of SDCA), the terms correspond to the number of FLOPs for 1D convolution operation and linear projection, respectively.

**Post-Attention.** In the post-attention module, we use a multi-head self-attention layer and a feed-forward network to generate the panoptic kernels. Subsequently, we produce the masks by means of 2D convolution and predict the associated classes using additional feed-forward networks. Inspired by [4, 7], we utilize the panoptic kernels to update the proposal kernels iteratively for improved accuracy.

### 4. Experiments

#### 4.1. Datasets and Evaluation Metrics

**Datasets.** We evaluated the effectiveness and efficiency of YOSO on four widely used panoptic segmentation datasets: the COCO dataset [34], the Cityscapes dataset [14], the ADE20K [60] dataset, and the Mapillary Vistas [38] dataset. The COCO dataset gathers images of complex everyday scenes with common objects, containing 80 things categories and 53 stuff categories in 118k images for training and 5k images for validation. The Cityscapes dataset contains images of urban street-view scenes, which has 8 things categories and 11 stuff categories in 2.9k images for training and 0.5k images for validation. The ADE20K dataset is annotated in an open-vocabulary setting with 50 things categories and 100 stuff categories, including 20k images for training and 2k images for validation. The Mapillary Vistas dataset is a large-scale urban street-view dataset with 37 things categories and 28 stuff categories in 18k and 2k images for training and validation.

**Evaluation Metrics.** The panoptic segmentation results are assessed using the panoptic quality (PQ) metric [28], which can be further decomposed to the segmentation quality (SQ) and the recognition quality (RQ). The evaluation outcomes for stuff and things are represented by the superscripts \( s \) and \( t \), respectively. The frame rate, i.e., frames per second (FPS), of the YOSO models is evaluated on a single V100 GPU for the main results and a single 3090 GPU for the ablation study.

#### 4.2. Implementation Details

For the COCO dataset, we used a batch size of 16 and set the learning rate to 0.0001. The models were trained for 370k iterations with large-scale jitter augmentation [20]. For the Cityscapes and Mapillary datasets, we trained the models with a batch size of 16, the learning rate set to 0.0001, and the training schedule set to 180k iterations. For the ADE20K dataset, we set the batch size to 16, the learning rate to 0.0001, and the models were trained for 30k iterations. The ResNet50 [22] (denoted as R50) pre-trained on the ImageNet [17] dataset is employed as our backbone for the four datasets. We used ResNet50 [22] (denoted as
Table 1. Panoptic segmentation on the COCO validation set.

<table>
<thead>
<tr>
<th>Method</th>
<th>Backbone</th>
<th>Scale</th>
<th>PQ</th>
<th>PQ$^1$</th>
<th>PQ$^2$</th>
<th>FPS$^1$</th>
<th>FPS$^2$</th>
<th>GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>BGRNet [52]</td>
<td>R50-FPN</td>
<td>800,1333</td>
<td>43.2</td>
<td>49.8</td>
<td>33.4</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>K-Net [58]</td>
<td>R50-FPN</td>
<td>800,1333</td>
<td>47.1</td>
<td>51.7</td>
<td>40.3</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>PanSegFormer [32]</td>
<td>R50</td>
<td>800,1333</td>
<td>49.6</td>
<td>54.9</td>
<td>42.4</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Max-DeepLab [47]</td>
<td>Max-S</td>
<td>800,1333</td>
<td>48.4</td>
<td>53.0</td>
<td>41.5</td>
<td>7.6</td>
<td>V100</td>
<td>-</td>
</tr>
<tr>
<td>Mask2Former [10]</td>
<td>R50</td>
<td>800,1333</td>
<td>51.9</td>
<td>57.7</td>
<td>43.0</td>
<td>8.6</td>
<td>V100</td>
<td>-</td>
</tr>
<tr>
<td>UPSNet [54]</td>
<td>R50-FPN</td>
<td>800,1333</td>
<td>42.5</td>
<td>48.5</td>
<td>33.4</td>
<td>9.1</td>
<td>V100</td>
<td>-</td>
</tr>
<tr>
<td>PanopticFCN [31]</td>
<td>R50-FPN</td>
<td>800,1333</td>
<td>44.3</td>
<td>50.0</td>
<td>35.6</td>
<td>9.2</td>
<td>V100</td>
<td>-</td>
</tr>
<tr>
<td>LPSNet [23]</td>
<td>R50-FPN</td>
<td>800,1333</td>
<td>39.1</td>
<td>43.9</td>
<td>30.1</td>
<td>9.3</td>
<td>V100</td>
<td>-</td>
</tr>
<tr>
<td>RealTimePan [24]</td>
<td>R50-FPN</td>
<td>800,1333</td>
<td>37.1</td>
<td>41.0</td>
<td>30.7</td>
<td>15.9</td>
<td>V100</td>
<td>-</td>
</tr>
<tr>
<td>PanopticFPN [27]</td>
<td>R50-FPN</td>
<td>800,1333</td>
<td>41.5</td>
<td>48.3</td>
<td>31.2</td>
<td>17.5</td>
<td>V100</td>
<td>-</td>
</tr>
<tr>
<td>MaskFormer [11]</td>
<td>R50</td>
<td>800,1333</td>
<td>46.5</td>
<td>51.0</td>
<td>39.8</td>
<td>17.6</td>
<td>V100</td>
<td>-</td>
</tr>
<tr>
<td>PanopticDeepLab [9]</td>
<td>R50</td>
<td>641,641</td>
<td>35.1</td>
<td>-</td>
<td>-</td>
<td>20.0</td>
<td>V100</td>
<td>-</td>
</tr>
<tr>
<td><strong>YOSO, ours</strong></td>
<td>R50</td>
<td>800,1333</td>
<td>48.4</td>
<td>53.5</td>
<td>40.8</td>
<td>23.6</td>
<td>V100</td>
<td>-</td>
</tr>
<tr>
<td><strong>YOSO, ours</strong></td>
<td>R50</td>
<td>512,800</td>
<td>46.4</td>
<td>50.7</td>
<td>40.0</td>
<td>45.6</td>
<td>V100</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 2. Panoptic segmentation on the Cityscapes validation set.

<table>
<thead>
<tr>
<th>Method</th>
<th>Backbone</th>
<th>Scale</th>
<th>PQ</th>
<th>PQ$^1$</th>
<th>PQ$^2$</th>
<th>FPS$^1$</th>
<th>FPS$^2$</th>
<th>GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>PanopticFPN [27]</td>
<td>R50-FPN</td>
<td>1024,2048</td>
<td>57.7</td>
<td>51.6</td>
<td>62.2</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Seamless [43]</td>
<td>R50</td>
<td>1024,2048</td>
<td>59.8</td>
<td>54.6</td>
<td>63.6</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>PanopticFCN [31]</td>
<td>R50-FPN</td>
<td>1024,2048</td>
<td>61.4</td>
<td>54.8</td>
<td>66.6</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Mask2Former [10]</td>
<td>R50</td>
<td>1024,2048</td>
<td>62.1</td>
<td>54.9</td>
<td>67.3</td>
<td>4.1</td>
<td>V100</td>
<td>-</td>
</tr>
<tr>
<td>UPSNet [54]</td>
<td>R50-FPN</td>
<td>1024,2048</td>
<td>59.3</td>
<td>54.6</td>
<td>62.7</td>
<td>7.5</td>
<td>V100</td>
<td>-</td>
</tr>
<tr>
<td>LPSNet [23]</td>
<td>R50-FPN</td>
<td>1024,2048</td>
<td>59.7</td>
<td>54.0</td>
<td>63.9</td>
<td>7.7</td>
<td>V100</td>
<td>-</td>
</tr>
<tr>
<td>PanopticDeepLab [9]</td>
<td>R50-FPN</td>
<td>1024,2048</td>
<td>59.7</td>
<td>-</td>
<td>-</td>
<td>8.5</td>
<td>V100</td>
<td>-</td>
</tr>
<tr>
<td>FPSNet [16]</td>
<td>R50-FPN</td>
<td>1024,2048</td>
<td>55.1</td>
<td>-</td>
<td>-</td>
<td>8.8</td>
<td>Titan</td>
<td>-</td>
</tr>
<tr>
<td>RealTimePan [24]</td>
<td>R50-FPN</td>
<td>1024,2048</td>
<td>58.8</td>
<td>52.1</td>
<td>63.7</td>
<td>10.1</td>
<td>V100</td>
<td>-</td>
</tr>
<tr>
<td><strong>YOSO, ours</strong></td>
<td>R50</td>
<td>1024,2048</td>
<td>59.7</td>
<td>51.0</td>
<td>66.1</td>
<td>11.1</td>
<td>V100</td>
<td>-</td>
</tr>
<tr>
<td><strong>YOSO, ours</strong></td>
<td>R50</td>
<td>512,1024</td>
<td>52.5</td>
<td>43.5</td>
<td>59.1</td>
<td>22.6</td>
<td>V100</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 3. Panoptic segmentation on the ADE20K validation set.

<table>
<thead>
<tr>
<th>Method</th>
<th>Backbone</th>
<th>Scale</th>
<th>PQ</th>
<th>PQ$^1$</th>
<th>PQ$^2$</th>
<th>FPS$^1$</th>
<th>FPS$^2$</th>
<th>GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>BGRNet [52]</td>
<td>R50-FPN</td>
<td>-</td>
<td>31.8</td>
<td>34.1</td>
<td>27.3</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>PanSegFormer [32]</td>
<td>R50</td>
<td>-</td>
<td>36.4</td>
<td>35.3</td>
<td>38.6</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>MaskFormer [11]</td>
<td>R50</td>
<td>640,2560</td>
<td>34.7</td>
<td>32.2</td>
<td>39.7</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Mask2Former [10]</td>
<td>R50</td>
<td>640,2560</td>
<td>39.7</td>
<td>39.0</td>
<td>40.9</td>
<td>11.1</td>
<td>V100</td>
<td>-</td>
</tr>
<tr>
<td><strong>YOSO, ours</strong></td>
<td>R50</td>
<td>640,2560</td>
<td>38.0</td>
<td>37.3</td>
<td>39.4</td>
<td>35.4</td>
<td>V100</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 4. Panoptic segmentation on the Mapillary validation set.

R50) pre-trained on ImageNet [17] as the backbone for all four datasets, and set the hidden dimension \(d\) to 256 through experimentation. Since the COCO dataset contains images from various scenes, ranging from indoor to outdoor, we conducted ablation studies on this dataset. In the ablation studies, the models were trained with 270k iterations.

4.3. Main Results

The results of panoptic segmentation on the COCO dataset are presented in Tab. 1. We have the following findings. First, YOSO is significantly faster than predominant efficient panoptic segmentation models such as PanopticFPN [27] and RealTimePan [24]. Specifically, YOSO achieves a PQ of 48.4 and an FPS of 23.6 with an input image scale of \((800, 1333)\). This PQ is 11.3 higher than that of RealTimePan, while the speed is approximately \(1.5\times\) faster. Furthermore, when scaling the input image to \((512, 800)\), YOSO is around \(2.3\times\) faster than the previous fastest model, PanopticDeepLab, while achieving an approximately 11.0-point higher PQ. Second, YOSO achieves comparable accuracy with state-of-the-art models such as MaskFormer [11], Mask2Former [10], Max-DeepLab [47], and K-Net [58]. For example, YOSO outperforms MaskFormer and K-Net by 1.9 and 1.3 PQ, respectively, and achieves the same PQ performance as Max-DeepLab. Although the PQ of YOSO is 3.5 points lower than that of Mask2Former, YOSO is \(2.7\times\) faster than Mask2Former.

The results of panoptic segmentation on the Cityscapes dataset are presented in Tab. 2. YOSO is the fastest model with competitive accuracy among state-of-the-art approaches. For example, YOSO achieves 59.7 PQ and 11.1 FPS with the input image size of \((1024, 2048)\), which is 4.7 points higher than FPSNet. Moreover, when reducing the input image scale to \((512, 1024)\), YOSO achieves an accuracy of 52.5 PQ with 22.6 FPS.

In Tab. 3 and Tab. 4, we show the panoptic segmentation results on the ADE20K and the Mapillary Vistas datasets, respectively, to evaluate the model generalization of YOSO.
<table>
<thead>
<tr>
<th>Aggregator</th>
<th>PQ</th>
<th>SQ</th>
<th>RQ</th>
<th>PQ'</th>
<th>PQ''</th>
<th>FLOPs</th>
<th>Latency (µs)</th>
<th>FPS$^*$</th>
</tr>
</thead>
<tbody>
<tr>
<td>IFA</td>
<td>47.5</td>
<td>82.2</td>
<td>56.9</td>
<td>52.7</td>
<td>39.7</td>
<td>16.6G</td>
<td>4871±11</td>
<td>23.3</td>
</tr>
<tr>
<td>CFA</td>
<td>47.0</td>
<td>81.4</td>
<td>56.2</td>
<td>52.3</td>
<td>39.0</td>
<td>2.1G</td>
<td>1877±52</td>
<td>29.2</td>
</tr>
</tbody>
</table>

Table 5. Comparison of different aggregators. The FLOPs and GPU latency were obtained from single modules with the setting of $d=256$, $c_2=128$, $c_3=256$, $c_4=512$, $c_5=1024$, and $h=512$.

<table>
<thead>
<tr>
<th>Attention</th>
<th>PQ</th>
<th>SQ</th>
<th>RQ</th>
<th>PQ'</th>
<th>PQ''</th>
<th>FLOPs</th>
<th>Latency (µs)</th>
<th>FPS$^*$</th>
</tr>
</thead>
<tbody>
<tr>
<td>MHCA</td>
<td>46.0</td>
<td>81.9</td>
<td>55.1</td>
<td>51.5</td>
<td>37.7</td>
<td>3.15M</td>
<td>2688±210</td>
<td>27.3</td>
</tr>
<tr>
<td>SDCA</td>
<td>47.0</td>
<td>81.4</td>
<td>56.2</td>
<td>52.3</td>
<td>39.0</td>
<td>5.4M</td>
<td>2183±279</td>
<td>29.2</td>
</tr>
<tr>
<td>DCA</td>
<td>46.9</td>
<td>82.0</td>
<td>55.8</td>
<td>51.9</td>
<td>38.7</td>
<td>15.5M</td>
<td>1701±186</td>
<td>30.0</td>
</tr>
<tr>
<td>PDCA</td>
<td>43.7</td>
<td>81.3</td>
<td>52.3</td>
<td>49.3</td>
<td>35.3</td>
<td>5.2M</td>
<td>1450±183</td>
<td>30.2</td>
</tr>
<tr>
<td>DDCA</td>
<td>46.6</td>
<td>82.3</td>
<td>55.9</td>
<td>52.3</td>
<td>38.4</td>
<td>0.3M</td>
<td>1242±101</td>
<td>30.3</td>
</tr>
</tbody>
</table>

Table 6. Comparison of different attention modules. The FLOPs and GPU latency were obtained from single modules with the setting of $n=100$, $d=256$, and $t=3$. The modules tested included MHCA (Multi-Head Cross-Attention), DCA (Dynamic Convolution Attention), SDCA (Separable Dynamic Convolution Attention), PDCA (Pointwise Dynamic Convolution Attention), and DDCA (Depthwise Dynamic Convolution Attention).

On the ADE20K dataset, YOSO outperforms most previous methods such as PanSegFormer [32] and MaskFormer [11] in terms of both speed and accuracy. On the Mapillary Vistas dataset, although YOSO has a good PQ, the performance of PQ' lags behind that of state-of-the-art models. This suggests that YOSO still has the potential to be improved on the Mapillary Vistas dataset.

Additionally, we plot the PQ w.r.t. FPS results on the COCO and the Cityscapes datasets in Fig. 4, which shows that YOSO runs faster and achieves competitive accuracy among state-of-the-art models. In summary, the main results on the four datasets validate the good generalization and well-balanced speed-accuracy of YOSO.

### 4.4. Ablation Study

In order to examine the impact of different components on the speed and accuracy of YOSO, we conducted several ablation studies focusing on the feature pyramid aggregator and the separable dynamic decoder. Specifically, we evaluated the effectiveness of the aggregation modules and the attention modules, which resulted in several interesting findings. Moreover, we analyzed how variations in the number of attention blocks, kernel size, iteration stages, and proposal kernels affected the performance. The details of our investigations are discussed below.

**Comparison of different aggregators.** The results of using different aggregators are presented in Tab. 5. Specifically, we trained YOSO with IFA and CFA, respectively. The PQ results indicate that IFA achieves higher accuracy than CFA, with PQ values of 47.5 and 47.0, respectively. However, IFA has much larger FLOPs than CFA, with 16.6G compared to 2.1G, and a slower GPU latency, with 4871µs compared to 1877µs. Considering that the learned parameters of IFA can be directly re-parameterized to CFA, we can train YOSO with IFA and infer with the re-parameterized CFA for better speed and accuracy.

In Fig. 5 (left), we investigate the reduction ratio of FLOPs between CFA and IFA. Specifically, we set the channel dimensions of the input feature maps $c_5$, $c_4$, $c_3$, and $c_2$ to be equal to $c$, and analyze how the input channel dimension $c$ and the output hidden dimension $d$ affect the FLOPs reduction ratio in Eq. 2. Our results indicate that increasing the input channel dimension and decreasing the output hidden dimension lead to an increase in the reduction ratio. This implies that CFA will be more efficient when the dimension of the input channel is large.

**Comparison of different attention modules.** We present a comparison of the effectiveness of different attention modules in Tab. 6. In terms of PQ performance, we have made two interesting findings. First, we were surprised to find that the MHCA module did not perform better than DCA and SDCA. The reason for this may be the difference between the basic operations in these two types of attention modules: DCA densely splits the hidden dimension into $d$ groups, while MHCA sparsely splits it into $t$ groups. Second, PDCA showed inferior performance compared to the other modules, with a PQ of 43.7. This is likely due to the fact that PDCA is the only module that does not apply cross-dimension interaction, which implies that interactions between hidden dimensions are significant for the attention module. This observation is supported by the performance of DDCA, which only performs cross-token interaction and achieved a PQ of 46.7. These two results suggest that the cross-dimension interaction may be more significant than the cross-token interaction in panoptic kernel generation, as the panoptic kernels are expected to be independent to represent different stuff or things.

In terms of FLOPs, we observed that the attention modules based on dynamic convolution require fewer FLOPs. Specifically, DDCA exhibits the lowest computational cost, requiring only 0.3M FLOPs. In terms of GPU latency, we made two interesting observations. First, although the
Table 7. Influence of attention block number. The PQ accuracy increases with more blocks, while the FPS decreases.

<table>
<thead>
<tr>
<th>#Blocks</th>
<th>PQ</th>
<th>SQ</th>
<th>RQ</th>
<th>PQ'</th>
<th>SQ'</th>
<th>RQ'</th>
<th>PQ''</th>
<th>SQ''</th>
<th>RQ''</th>
<th>FPS↑</th>
</tr>
</thead>
<tbody>
<tr>
<td>N=1</td>
<td>46.4</td>
<td>81.7</td>
<td>55.7</td>
<td>52.1</td>
<td>82.3</td>
<td>62.1</td>
<td>37.9</td>
<td>79.9</td>
<td>46.0</td>
<td>30.1</td>
</tr>
<tr>
<td>N=2</td>
<td>47.0</td>
<td>81.4</td>
<td>56.2</td>
<td>52.3</td>
<td>83.1</td>
<td>62.2</td>
<td>39.0</td>
<td>78.8</td>
<td>47.1</td>
<td>29.2</td>
</tr>
<tr>
<td>N=3</td>
<td>47.5</td>
<td>82.1</td>
<td>56.8</td>
<td>52.6</td>
<td>83.2</td>
<td>62.6</td>
<td>39.7</td>
<td>80.4</td>
<td>48.3</td>
<td>28.9</td>
</tr>
<tr>
<td>N=4</td>
<td>47.2</td>
<td>81.8</td>
<td>56.7</td>
<td>52.5</td>
<td>82.9</td>
<td>62.6</td>
<td>39.3</td>
<td>80.1</td>
<td>47.6</td>
<td>28.7</td>
</tr>
</tbody>
</table>

Table 8. Results of using different convolution kernel sizes. The PQ performance increases with more iterations, while the FPS decreases when $t=5$.

<table>
<thead>
<tr>
<th>Kernel Size</th>
<th>PQ</th>
<th>SQ</th>
<th>RQ</th>
<th>PQ'</th>
<th>SQ'</th>
<th>RQ'</th>
<th>PQ''</th>
<th>SQ''</th>
<th>RQ''</th>
<th>FPS↑</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t=1$</td>
<td>46.8</td>
<td>82.3</td>
<td>55.9</td>
<td>52.4</td>
<td>83.4</td>
<td>62.3</td>
<td>38.5</td>
<td>80.6</td>
<td>46.4</td>
<td>30.6</td>
</tr>
<tr>
<td>$t=3$</td>
<td>47.0</td>
<td>81.4</td>
<td>56.2</td>
<td>52.3</td>
<td>83.1</td>
<td>62.2</td>
<td>39.0</td>
<td>78.8</td>
<td>47.1</td>
<td>29.2</td>
</tr>
<tr>
<td>$t=5$</td>
<td>47.3</td>
<td>81.3</td>
<td>56.8</td>
<td>52.2</td>
<td>82.0</td>
<td>62.2</td>
<td>40.0</td>
<td>80.3</td>
<td>48.6</td>
<td>28.3</td>
</tr>
<tr>
<td>$t=7$</td>
<td>47.1</td>
<td>81.2</td>
<td>56.3</td>
<td>52.4</td>
<td>83.2</td>
<td>62.3</td>
<td>39.8</td>
<td>80.2</td>
<td>48.4</td>
<td>27.6</td>
</tr>
</tbody>
</table>

Table 9. Impact of iteration stages. The PQ accuracy increases with more iteration stages, while the FPS decreases.

<table>
<thead>
<tr>
<th>#Stages</th>
<th>PQ</th>
<th>SQ</th>
<th>RQ</th>
<th>PQ'</th>
<th>SQ'</th>
<th>RQ'</th>
<th>PQ''</th>
<th>SQ''</th>
<th>RQ''</th>
<th>FPS↑</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T=1$</td>
<td>45.7</td>
<td>80.4</td>
<td>53.6</td>
<td>50.3</td>
<td>82.1</td>
<td>59.6</td>
<td>36.5</td>
<td>77.3</td>
<td>45.2</td>
<td>30.1</td>
</tr>
<tr>
<td>$T=2$</td>
<td>47.0</td>
<td>81.4</td>
<td>56.2</td>
<td>52.3</td>
<td>83.1</td>
<td>62.2</td>
<td>39.0</td>
<td>78.8</td>
<td>47.1</td>
<td>29.2</td>
</tr>
<tr>
<td>$T=3$</td>
<td>47.5</td>
<td>81.9</td>
<td>56.9</td>
<td>52.8</td>
<td>82.9</td>
<td>63.0</td>
<td>39.4</td>
<td>80.4</td>
<td>47.7</td>
<td>28.9</td>
</tr>
</tbody>
</table>

Table 10. Different number of proposals. The PQ accuracy increases with more proposals and saturates when $n=150$.

<table>
<thead>
<tr>
<th>#Proposals</th>
<th>PQ</th>
<th>SQ</th>
<th>RQ</th>
<th>PQ'</th>
<th>SQ'</th>
<th>RQ'</th>
<th>PQ''</th>
<th>SQ''</th>
<th>RQ''</th>
<th>FPS↑</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n=50$</td>
<td>44.2</td>
<td>80.1</td>
<td>52.6</td>
<td>49.8</td>
<td>81.2</td>
<td>59.2</td>
<td>36.1</td>
<td>78.4</td>
<td>44.1</td>
<td>33.7</td>
</tr>
<tr>
<td>$n=100$</td>
<td>47.0</td>
<td>81.4</td>
<td>56.2</td>
<td>52.3</td>
<td>83.1</td>
<td>62.2</td>
<td>39.0</td>
<td>78.8</td>
<td>47.1</td>
<td>29.2</td>
</tr>
<tr>
<td>$n=150$</td>
<td>46.9</td>
<td>81.0</td>
<td>55.7</td>
<td>52.0</td>
<td>82.7</td>
<td>61.5</td>
<td>38.9</td>
<td>78.6</td>
<td>46.8</td>
<td>26.3</td>
</tr>
<tr>
<td>$n=200$</td>
<td>46.7</td>
<td>81.1</td>
<td>55.2</td>
<td>51.5</td>
<td>82.5</td>
<td>61.2</td>
<td>38.7</td>
<td>78.5</td>
<td>46.2</td>
<td>24.3</td>
</tr>
</tbody>
</table>

5. Conclusion

In this paper, we propose a real-time panoptic segmentation framework, termed YOSO. With YOSO, you only need to segment once for the masks of both foreground things and background stuff. YOSO includes a feature pyramid aggregator and a separable dynamic detector to accelerate the pipeline. The CFA module in the feature pyramid aggregator re-parameters the IFA module, reducing FLOPs without extra costs. The SDC module in the separable dynamic decoder performs weight-sharing multi-head cross-attention, enhancing both speed and accuracy. Our extensive experiments demonstrate that YOSO is significantly faster than other prominent panoptic segmentation methods while maintaining competitive PQ performance. Given its effectiveness and simplicity, we hope YOSO can serve as a strong baseline and bring fresh insights for future research on real-time panoptic segmentation.

Acknowledgements. This work was supported by National Key R&D Program of China (2022ZD0118202), National Science Fund for Distinguished Young Scholars (No.62025603), National Natural Science Foundation of China (No.U21B2037, No.U22B2051, No.62176222, No.62176223, No.62176225, No.62072386, No.62072387, No.62072389, No.62002305 and No.62272401), and Natural Science Foundation of Fujian Province of China (No.2021J01002, No.2022J06001).
References


[54] Yuwen Xiong, Renjie Liao, Hengshuang Zhao, Rui Hu, Min Bai, Ersin Yumer, and Raquel Urtasun. Upsnet: A


