**Fully Convolutional Networks for Panoptic Segmentation**

Yanwei Li\(^1\)  Hengshuang Zhao\(^2\)  Xiaojuan Qi\(^3\)  Liwei Wang\(^1\)
Zeming Li\(^4\)  Jian Sun\(^4\)  Jiaya Jia\(^{1,5}\)

The Chinese University of Hong Kong\(^1\)  University of Oxford\(^2\)
The University of Hong Kong\(^3\)  MEGVII Technology\(^4\)  SmartMore\(^5\)

**Abstract**

In this paper, we present a conceptually simple, strong, and efficient framework for panoptic segmentation, called Panoptic FCN. Our approach aims to represent and predict foreground things and background stuff in a unified fully convolutional pipeline. In particular, Panoptic FCN encodes each object instance or stuff category into a specific kernel weight with the proposed kernel generator and produces the prediction by convolving the high-resolution feature directly. With this approach, instance-aware and semantically consistent properties for things and stuff can be respectively satisfied in a simple generate-kernel-then-segment workflow. Without extra boxes for localization or instance separation, the proposed approach outperforms previous box-based and -free models with high efficiency on COCO, Cityscapes, and Mapillary Vistas datasets with single scale input. Our code is made publicly available at https://github.com/Jia-Research-Lab/PanopticFCN.\(^1\)

**1. Introduction**

Panoptic segmentation, aiming to assign each pixel with a semantic label and unique identity, is regarded as a challenging task. In panoptic segmentation\(^{19}\), countable and uncountable instances (i.e., things and stuff) are expected to be represented and resolved in a unified workflow. One main difficulty impeding unified representation comes from conflicting properties requested by things and stuff. Specifically, to distinguish among various identities, countable things usually rely on instance-aware features, which vary with objects. In contrast, uncountable stuff would prefer semantically consistent characters, which ensures consistent predictions for pixels with the same semantic meaning. An example is given in Fig. 1, where embedding of individuals should be diverse for inter-class variations, while characters of grass should be similar for intra-class consistency.

\(^{1}\)Part of the work was done in MEGVII Research.

For conflict at feature level, specific modules are usually tailored for things and stuff separately, as presented in Fig. 1(a). In particular, instance-aware demand of things is satisfied mainly from two streams, namely box-based\(^{18, 50, 25}\) and box-free\(^{51, 10, 6}\) methods. Meanwhile, the semantic-consistency of stuff is met in a pixel-by-pixel manner\(^{33}\), where similar semantic features would bring identical predictions. A classic case is Panoptic FPN\(^{18}\), which utilizes Mask R-CNN\(^{12}\) and FCN\(^{33}\) in separated branches to respectively classify things and stuff, similar to that of Fig. 1(a). Although attempt\(^{51, 10, 6}\) has been made to predict things without boxes, extra predictions (e.g., affinities\(^{10}\), and offsets\(^{51}\)) together with post-process procedures are still needed to distinguish among instances, which slow down the whole system and hinder it from being fully convolutional. Consequently, a unified representation is required to bridge this gap.

In this paper, we propose a fully convolutional framework for unified representation, called Panoptic FCN. In particular, Panoptic FCN encodes each instance into a specific kernel and generates the prediction by convolutions directly. Thus, both things and stuff can be predicted together...
with a same resolution. In this way, instance-aware and semantically consistent properties for things and stuff can be respectively satisfied in a unified workflow, which is briefly illustrated in Fig. 1(b). To sum up, the key idea of Panoptic FCN is to represent and predict things and stuff uniformly with generated kernels in a fully convolutional pipeline.

To this end, kernel generator and feature encoder are respectively designed for kernel weights generation and shared feature encoding. Specifically, in kernel generator, we draw inspirations from point-based object detectors [20, 55] and utilize the position head to locate as well as classify foreground objects and background stuff by object centers and stuff regions, respectively. Then, we select kernel weights [17] with the same positions from the kernel head to represent corresponding instances. For the instance-awareness and semantic-consistency described above, a kernel-level operation, called kernel fusion, is further proposed, which merges kernel weights that are predicted to have the same identity or semantic category. With a naive feature encoder, which preserves the high-resolution feature with details, each prediction of things and stuff can be produced by convolving with generated kernels directly.

In general, the proposed method can be distinguished from two aspects. Firstly, different from previous work for things generation [12, 4, 45], which outputs dense predictions and then utilizes NMS for overlaps removal, the designed framework generates instance-aware kernels and produces each specific instance directly. Moreover, compared with traditional FCN-based methods for stuff prediction [53, 3, 9], which select the most likely category in a pixel-by-pixel manner, our approach aggregates global context into semantically consistent kernels and presents results of existing semantic classes in a whole-instance manner.

The overall approach, named Panoptic FCN, can be easily instantiated for panoptic segmentation, which will be fully elaborated in Sec. 3. To demonstrate its superiority, we give extensive ablation studies in Sec. 4.2. Furthermore, experimental results are reported on COCO [29], Cityscapes [8], and Mapillary Vistas [35] datasets. Without bells-and-whistles, Panoptic FCN outperforms previous methods with efficiency, and respectively attains 44.3% PQ and 47.5% PQ on COCO val and test-dev set. Meanwhile, it surpasses all similar box-free methods by large margins and achieves leading performance on Cityscapes and Mapillary Vistas val set with 61.4% PQ and 36.9% PQ, respectively.

2. Related Work

Panoptic segmentation. Traditional approaches mainly conduct segmentation for things and stuff separately. The benchmark for panoptic segmentation [19] directly combines predictions of things and stuff from different models, causing heavy computational overhead. To solve this problem, methods have been proposed by dealing with things and stuff in one model but in separate branches, including Panoptic FPN [18], AUNet [25], and UPSNet [50]. From the view of instance representation, previous work mainly formats things and stuff from different perspectives. Foreground things are usually separated and represented with boxes [18, 52, 5, 24] or aggregated according to center offsets [51], while background stuff is often predicted with a parallel FCN [33] branch. Although methods of [23, 10] represent things and stuff uniformly, the inherent ambiguity cannot be resolved well merely with the pixel-level affinity, which yields the performance drop in complex scenarios. In contrast, the proposed Panoptic FCN represents things and stuff in a uniform and fully convolutional framework with decent performance and efficiency.

Instance segmentation. Instance segmentation aims to discriminate objects in the pixel level, which is a finer representation compared with detected boxes. For instance-awareness, previous works can be roughly divided into two streams, i.e., box-based methods and box-free approaches. Box-based methods usually utilize detected boxes to locate or separate objects [12, 32, 1, 21, 38]. Meanwhile, box-free approaches are designed to generate instances without assistance of object boxes [10, 4, 45, 46]. Recently, AdaptIS [40] and CondInst [42] are proposed to utilize point-proposal for instance segmentation. However, the instance aggregation or object-level removal is still needed for results. In this paper, we represent objects in a box-free pipeline, which generates the kernel for each object and produces results by convolving the detail-rich feature directly, with no need for object-level duplicates removal [15, 37].

Semantic segmentation. Semantic segmentation assigns each pixel with a semantic category, without considering diverse object identities. In recent years, rapid progress has been made on top of FCN [33]. Due to the semantically consistent property, several attempts have been made to capture contextual cues from wider perception fields [53, 2, 3] or establish pixel-wise relationship for long-range dependencies [54, 16, 41]. There is also work to design network architectures for semantic segmentation automatically [30, 26], which is beyond the scope of this paper. Our proposed Panoptic FCN adopts a similar method to represent things and stuff, which aggregates global context into a specific kernel to predict corresponding semantic category.

3. Panoptic FCN

Panoptic FCN is conceptually simple: kernel generator is introduced to generate kernel weights for things and stuff with different categories; kernel fusion is designed to merge kernel weights with the same identity from multiple stages; and feature encoder is utilized to encode the high-resolution feature. In this section, we elaborate on the above components as well as the training and inference scheme.
3.1. Kernel Generator

Given a single stage feature $X_i$ from the $i$-th stage in FPN [27], the proposed kernel generator aims at generating the kernel weight map $G_i$ with positions for things $L_i^{th}$ and stuff $L_i^{st}$, as depicted in Fig. 2. To this end, position head is utilized for instance localization and classification, while kernel head is designed for kernel weight generation.

**Position head.** With the input $X_i \in \mathbb{R}^{C_i \times W_i \times H_i}$, we simply adopt stacks of convolutions to encode the feature map and generate $X'_i$, as presented in Fig. 2. Then we need to locate and classify each instance from the shared feature map $X'_i$. However, according to the definition [19], things can be distinguished by object centers, while stuff is uncountable. Thus, we adopt object centers and stuff regions to respectively represent position of each individual and stuff category. It means background regions with the same semantic meaning are viewed as one instance. In particular, object map $L_i^{th} \in \mathbb{R}^{N_{th} \times W_i \times H_i}$ and stuff map $L_i^{st} \in \mathbb{R}^{N_{st} \times W_i \times H_i}$ can be generated by convolutions directly with the shared feature map $X'_i$, where $N_{th}$ and $N_{st}$ denote the number of semantic category for things and stuff, respectively.

To better optimize $L_i^{th}$ and $L_i^{st}$, different strategies are adopted to generate the ground truth. For the $k$-th object in class $c$, we split positive key-points onto the $c$-th channel of the heatmap $Y_i^{th} \in [0, 1]^{N_{th} \times W_i \times H_i}$ with Gaussian kernel, similar to that in [20, 55]. With respect to stuff, we produce the ground truth $Y_i^{st} \in [0, 1]^{N_{st} \times W_i \times H_i}$ by bilinear interpolating the one-hot semantic label to corresponding sizes. Hence, the position head can be optimized with $L_{pos}$ and $L_{pos}^{th}$ for object centers and stuff regions, respectively.

$$L_{pos} = \sum_i FL(L_i^{th}, X_i^{th})/N_{th},$$

$$L_{pos}^{st} = \sum_i FL(L_i^{st}, Y_i^{st})/W_iH_i,$$

$$L_{pos} = L_{pos}^{th} + L_{pos}^{st},$$

where $FL(\cdot, \cdot)$ represents the Focal Loss [28] for optimization. For inference, $D_i^{th} = \{(x, y) : 1(L_i^{th}_{c,x,y}) = 1\}$ and $D_i^{st} = \{(x, y) : 1(L_i^{st}_{c,x,y}) = 1\}$ are selected to respectively represent the existence of object centers and stuff regions in corresponding positions with predicted categories $O_i$. This process will be further explained in Sec. 3.4.

**Kernel head.** In kernel head, we first capture spatial cues by directly concatenating relative coordinates to the feature $X_i$, which is similar with that in CoordConv [31]. With the concatenated feature map $X'_i' \in \mathbb{R}^{(C_i + 2) \times W_i \times H_i}$, stacks of convolutions are adopted to generate the kernel weight map $G_i \in \mathbb{R}^{C_i \times W_i \times H_i}$, as presented in Fig. 2. Given predictions $D_i^{th}$ and $D_i^{st}$ from the position head, kernel weights with the same coordinates in $G_i$ are chosen to represent corresponding instances. For example, assuming candidate $(x_c, y_c) \in D_i^{th}$, kernel weight $G_i_{c,x_c,y_c} \in \mathbb{R}^{C_i \times 1 \times 1}$ is selected to generate the result with predicted category $c$. The same is true for $D_i^{st}$. We represent the selected kernel weights in $i$-th stage for things and stuff as $G_i^{th}$ and $G_i^{st}$, respectively. Thus, the kernel weight $G_i^{th}$ and $G_i^{st}$ together with predicted categories $O_i$ in the $i$-th stage can be produced with the proposed kernel generator.
3.2. Kernel Fusion

Previous work [39, 12, 45] utilized NMS to remove duplicate boxes or instances in the post-processing stage. Different from them, the designed kernel fusion operation merges repetitive kernel weights from multiple FPN stages before final instance generation, which guarantees instance-awareness and semantic-consistency for things and stuff, respectively. In particular, given aggregated kernel weights \(G^\text{th}\) and \(G^\text{st}\) from all the stages, the \(j\)-th kernel weight \(K_j \in \mathbb{R}^{C_s \times 1 \times 1}\) is achieved by

\[
K_j = \text{AvgCluster}(G^j),
\]

where \text{AvgCluster} denotes average-clustering operation, and the candidate set \(G^j = \{G_m : \text{ID}(G_m) = \text{ID}(G_j)\}\) includes all the kernel weights, which are predicted to have the same identity ID with \(G_j\). For object centers, kernel weight \(G^\text{th}_m\) is viewed as identical with \(G^\text{th}_j\) if the cosine similarity between them surpasses a given threshold \(\text{thres}\), which will be further investigated in Table 3. For stuff regions, all kernel weights in \(G^\text{st}\), which share a same category with \(G^\text{st}_j\), are marked as one identity ID.

With the proposed approach, each kernel weight \(K^\text{th}_k\) in \(K^\text{th} = \{K^\text{th}_1, ..., K^\text{th}_M\} \in \mathbb{R}^{M \times C_s \times 1 \times 1}\) can be viewed as an embedding for single object, where the total number of objects is \(M\). Therefore, kernels with the same identity are merged as a single embedding for things, and each kernel in \(K^\text{th}\) represents an individual object, which satisfies the instance-awareness for things. Meanwhile, kernel weight \(K^\text{st}_j\) in \(K^\text{st} = \{K^\text{st}_1, ..., K^\text{st}_N\} \in \mathbb{R}^{N \times C_s \times 1 \times 1}\) represents the embedding for all \(j\)-th class pixels, where the existing number of stuff is \(N\). With this method, kernels with the same semantic category are fused into a single embedding, which guarantees the semantic-consistency for stuff. Thus, both properties requested by things and stuff can be fulfilled with the proposed kernel fusion operation.

3.3. Feature Encoder

To preserve details for instance representation, high-resolution feature \(F^\text{th} \in \mathbb{R}^{C_s \times W/4 \times H/4}\) is utilized for feature encoding. Feature \(F^\text{th}\) can be generated from FPN in several ways, e.g., P2 stage feature, summed features from all stages, and features from semantic FPN [18]. These methods are compared in Table 6. Given the feature \(F^\text{th}\), a similar strategy with that in kernel head is applied to encode positional cues and generate the encoded feature \(F^e = \mathbb{R}^{C_s \times W/4 \times H/4}\), as depicted in Fig. 2. Thus, given \(M\) and \(N\) kernel weights for things \(K^\text{th}\) and stuff \(K^\text{st}\) from the kernel fusion, each instance is produced by \(P_j = K_j \otimes F^e\). Here, \(P_j\) denotes the \(j\)-th prediction, and \(\otimes\) indicates the convolutional operation. That means \(M + N\) kernel weights generate \(M + N\) instance predictions with resolution \(W/4 \times H/4\) for the whole image. Consequently, the panoptic result can be produced with a simple process [18].

3.4. Training and Inference

**Training scheme.** In the training stage, the central point in each object and all the points in stuff regions are utilized to generate kernel weights for things and stuff, respectively. Here, Dice Loss [34] is adopted to optimize the predicted segmentation, which is formulated as

\[
\mathcal{L}_{\text{seg}} = \sum_j \text{Dice}(P_j, Y^\text{seg}_j)/(M + N),
\]

where \(Y^\text{seg}_j\) denotes ground truth for the \(j\)-th prediction \(P_j\). To further release the potential of kernel generator, multiple positives inside each object are sampled to represent the instance. In particular, we select \(k\) positions with top predicted scores \(s\) inside each object in \(L^\text{th}_i\), resulting in \(k \times M\) kernels as well as instances in total. This will be explored in Table 7. As for stuff regions, the factor \(k\) is set to 1, which means all the points in same category are equally treated. Then, we replace the original loss with a weighted version

\[
WDice(P_j, Y^\text{seg}_j) = \frac{\sum_k w_k \text{Dice}(P_{j,k}, Y^\text{seg}_j)}{\sum_k w_k},
\]

where \(w_k\) denotes the \(k\)-th weighted score with \(w_k = s_k/\sum_i s_i\). According to Eqs. (1) and (3), optimized target \(\mathcal{L}\) is defined with the weighted Dice Loss \(\mathcal{L}_{\text{seg}}\) as

\[
\mathcal{L}_{\text{seg}} = \sum_j WDice(P_j, Y^\text{seg}_j)/(M + N),
\]

**Inference scheme.** In the inference stage, Panoptic FCN follows a simple generate-kernel-then-segment pipeline. Specifically, we first aggregate positions \(D^\text{th}_i\), \(D^\text{st}_i\) and corresponding categories \(O_i\) from the \(i\)-th position head, as illustrated in the Sec. 3.1. For object centers, we preserve the peak points in \(\text{MaxPool}(L^\text{th}_i)\) utilizing a similar method with that in [55]. Thus, the indicator for things \(I(L^\text{th}_{i,c,x,y})\) is marked as positive if point \((x, y)\) in the \(c\)-th channel is preserved as the peak point. Similarly, the indicator for stuff regions \(I(L^\text{st}_{i,c,x,y})\) is viewed as positive if point \((x, y)\) with category \(c\) is kept. With the designed kernel fusion and the feature encoder, the prediction \(P\) can be easily produced. Specifically, we keep the top 100 scoring kernels of objects and all the kernels of stuff after kernel fusion for instance generation. The threshold 0.4 is utilized to convert predicted soft masks to binary results. It should be noted that both the heuristic process or direct \(\text{argmax}\) could be used to generate non-overlap panoptic results. The \(\text{argmax}\) could accelerate the inference but bring performance drop (1.4% PQ). For fair comparison both from speed and accuracy, the heuristic procedure [18] is adopted in experiments.
4. Experiments

In this section, we first introduce experimental settings for Panoptic FCN. Then we conduct abundant studies on the COCO [29] val set to reveal the effect of each component. Finally, comparison with previous methods on COCO [29], Cityscapes [8], and Mapillary Vistas [8] dataset is reported.

4.1. Experimental Setting

Architecture. From the perspective of network architecture, ResNet [13] with FPN [27] are utilized for backbone instantiation. P3 to P7 stages in FPN are used to provide single stage feature $X_i$ for the kernel generator that is shared across all stages. Meanwhile, P2 to P5 stages are adopted to generate the high-resolution feature $F^h$, which will be further investigated in Table 6. All convolutions in kernel generator are equipped with GroupNorm [47] and ReLU activation. Moreover, a naive convolution is adopted at the end of each head in kernel generator for feature projection.

Datasets. COCO dataset [29] is a widely used benchmark, which contains 80 thing classes and 53 stuff classes. It involves 118K, 5K, and 20K images for training, validation, and testing, respectively. Cityscapes dataset [8] consists of 5,000 street-view fine annotations with size $1024 \times 2048$, which are divided into 2,975, 500, and 1,525 images for training, validation, and testing, respectively. Mapillary Vistas [8] is a traffic-related dataset with resolutions ranging from $1024 \times 768$ to more than $4000 \times 6000$. It includes 37 thing classes and 28 stuff classes with 18K, 2K, and 5K images for training, validation, and testing, respectively.

Optimization. Network optimization is conducted using SGD with weight decay $10^{-4}$ and momentum 0.9. And $\lambda_{pol}$ schedule with power 0.9 is adopted. Experimentally, $\lambda_{pol}$ is set to a constant 1, and $\lambda_{seg}$ are respectively set to 3, 4, and 3 for COCO, Cityscapes, and Mapillary Vistas datasets. For COCO, we set initial rate to 0.01 and follow the $1 \times$ strategy in Detectron2 [48] by default. We randomly flip and rescale the shorter edge from 640 to 800 pixels with 90K iterations. Herein, annotated object centers with instance scale range $\{(1,64), (32,128), (64,256), (128,512), (256,2048)\}$ are assigned to P3-P7 stages, respectively. For Cityscapes, we optimize the network for 65K iterations with an initial rate 0.02 and construct each mini-batch with 32 random $512 \times 1024$ crops from images that are randomly rescaled from 0.5 to 2.0×. For Mapillary Vistas, the network is optimized for 150K iterations with an initial rate 0.02. In each iteration, we randomly resize images from 1024 to 2048 pixels at the shorted side and build 32 crops with the size $1024 \times 1024$. Due to the variation in scale distribution, we modify the assigning strategy to $\{(1,128), (64,256), (128,512), (256,1024), (512,2048)\}$ for Cityscapes and Mapillary Vistas datasets.

4.2. Component-wise Analysis

Kernel generator. Kernel generator plays a vital role in Panoptic FCN. Here, we compare several settings inside kernel generator to improve the kernel expressiveness in each stage. As presented in Table 1, with the number of convolutions in each head increasing, the network performance improves steadily and achieves the peak PQ with 3 stacked Conv3 × 3 whose channel number is 256. Simi-

<table>
<thead>
<tr>
<th>deform</th>
<th>conv num</th>
<th>PQ</th>
<th>PQ$^{th}$</th>
<th>PQ$^{st}$</th>
<th>AP</th>
<th>mIoU</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td>1</td>
<td>38.4</td>
<td>43.4</td>
<td>31.0</td>
<td>28.3</td>
<td>39.9</td>
</tr>
<tr>
<td>✓</td>
<td>2</td>
<td>38.9</td>
<td>44.1</td>
<td>31.1</td>
<td>28.9</td>
<td>40.1</td>
</tr>
<tr>
<td>✓</td>
<td>3</td>
<td>39.2</td>
<td>44.7</td>
<td>31.0</td>
<td>29.6</td>
<td>40.2</td>
</tr>
<tr>
<td>✓</td>
<td>4</td>
<td>39.2</td>
<td>44.9</td>
<td>30.8</td>
<td>29.4</td>
<td>39.9</td>
</tr>
<tr>
<td>✓</td>
<td>3</td>
<td>39.9</td>
<td>45.0</td>
<td>32.4</td>
<td>29.9</td>
<td>41.2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>coord$_w$</th>
<th>coord$_l$</th>
<th>PQ</th>
<th>PQ$^{th}$</th>
<th>PQ$^{st}$</th>
<th>AP</th>
<th>mIoU</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td>✓</td>
<td>39.9</td>
<td>45.0</td>
<td>32.4</td>
<td>29.9</td>
<td>41.2</td>
</tr>
<tr>
<td>✓</td>
<td>✓</td>
<td>40.2</td>
<td>45.3</td>
<td>32.5</td>
<td>30.4</td>
<td>41.6</td>
</tr>
<tr>
<td>✓</td>
<td>✓</td>
<td>41.3</td>
<td>46.9</td>
<td>32.9</td>
<td>32.1</td>
<td>41.7</td>
</tr>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>class-aware</th>
<th>thres</th>
<th>PQ</th>
<th>PQ$^{th}$</th>
<th>PQ$^{st}$</th>
<th>AP</th>
<th>mIoU</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td>0.80</td>
<td>39.7</td>
<td>44.3</td>
<td>32.9</td>
<td>29.9</td>
<td>41.7</td>
</tr>
<tr>
<td>✓</td>
<td>0.85</td>
<td>40.8</td>
<td>46.1</td>
<td>32.9</td>
<td>31.5</td>
<td>41.7</td>
</tr>
<tr>
<td>✓</td>
<td>0.90</td>
<td>41.3</td>
<td>46.9</td>
<td>32.9</td>
<td>32.1</td>
<td>41.7</td>
</tr>
<tr>
<td>✓</td>
<td>0.95</td>
<td>41.3</td>
<td>47.0</td>
<td>32.9</td>
<td>31.1</td>
<td>41.7</td>
</tr>
<tr>
<td>✓</td>
<td>1.00</td>
<td>38.7</td>
<td>42.6</td>
<td>32.9</td>
<td>25.4</td>
<td>41.7</td>
</tr>
<tr>
<td>✓</td>
<td>0.90</td>
<td>41.2</td>
<td>46.7</td>
<td>32.9</td>
<td>30.9</td>
<td>41.7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>kernel-fusion</th>
<th>nms</th>
<th>PQ</th>
<th>PQ$^{th}$</th>
<th>PQ$^{st}$</th>
<th>AP</th>
<th>mIoU</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td>✗</td>
<td>38.7</td>
<td>42.6</td>
<td>32.9</td>
<td>25.4</td>
<td>41.7</td>
</tr>
<tr>
<td>✓</td>
<td>✓</td>
<td>38.7</td>
<td>42.6</td>
<td>32.9</td>
<td>27.8</td>
<td>41.7</td>
</tr>
<tr>
<td>✓</td>
<td>✓</td>
<td>41.3</td>
<td>46.9</td>
<td>32.9</td>
<td>32.1</td>
<td>41.7</td>
</tr>
<tr>
<td>✓</td>
<td>✓</td>
<td>41.3</td>
<td>46.9</td>
<td>32.8</td>
<td>32.3</td>
<td>41.7</td>
</tr>
</tbody>
</table>

Table 1. Comparison with different settings of the kernel generator on the COCO val set. deform and conv num respectively denote deformable convolutions for position head and number of convolutions in both heads of the kernel generator.

Table 2. Comparison with different positional settings on the COCO val set. coord$\_w$ and coord$\_l$ denote combining coordinates for the kernel head, and feature encoder, respectively.

Table 3. Comparison with different similarity thresholds of kernel fusion on the COCO val set. class-aware denotes only merging kernel weights with the same predicted class. And thres indicates the cosine similarity threshold thres for kernel fusion in Sec. 3.2.

Table 4. Comparison with different methods of removing repetitive predictions. kernel-fusion and nms indicates the proposed kernel-level fusion method and Matrix NMS [46], respectively.
Latency (ms)

Table 5. Comparison with different channel numbers of the feature encoder on the COCO val set. channel num represents the channel number C_i of the feature encoder.

<table>
<thead>
<tr>
<th>channel num</th>
<th>PQ</th>
<th>PQ^{th}</th>
<th>PQ^st</th>
<th>AP</th>
<th>mIoU</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>39.9</td>
<td>45.0</td>
<td>32.1</td>
<td>30.8</td>
<td>41.3</td>
</tr>
<tr>
<td>32</td>
<td>40.8</td>
<td>46.3</td>
<td>32.5</td>
<td>31.7</td>
<td>41.6</td>
</tr>
<tr>
<td>64</td>
<td>41.3</td>
<td>46.9</td>
<td>32.9</td>
<td>32.1</td>
<td>41.7</td>
</tr>
<tr>
<td>128</td>
<td>41.3</td>
<td>47.0</td>
<td>32.6</td>
<td>32.6</td>
<td>41.7</td>
</tr>
</tbody>
</table>

Table 6. Comparison with different feature types for the feature encoder on the COCO val set. feature type denotes the method to generate high-resolution feature F_i in Sec. 3.3.

<table>
<thead>
<tr>
<th>feature type</th>
<th>PQ</th>
<th>PQ^{th}</th>
<th>PQ^st</th>
<th>AP</th>
<th>mIoU</th>
</tr>
</thead>
<tbody>
<tr>
<td>FPN-P2</td>
<td>40.6</td>
<td>46.0</td>
<td>32.4</td>
<td>31.6</td>
<td>41.3</td>
</tr>
<tr>
<td>FPN-Summed</td>
<td>40.5</td>
<td>46.0</td>
<td>32.1</td>
<td>31.7</td>
<td>41.1</td>
</tr>
<tr>
<td>Semantic FPN [18]</td>
<td>41.3</td>
<td>46.9</td>
<td>32.2</td>
<td>32.1</td>
<td>41.7</td>
</tr>
</tbody>
</table>

Table 7. Comparison with different settings of weighted dice loss on the COCO val set. weighted and k denote weighted dice loss and the number of sampled points in Sec. 3.4, respectively.

<table>
<thead>
<tr>
<th>weighted</th>
<th>k</th>
<th>PQ</th>
<th>PQ^{th}</th>
<th>PQ^st</th>
<th>AP</th>
<th>mIoU</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td></td>
<td>40.2</td>
<td>45.5</td>
<td>32.4</td>
<td>31.0</td>
<td>41.3</td>
</tr>
<tr>
<td>✓</td>
<td>1</td>
<td>40.0</td>
<td>45.1</td>
<td>32.4</td>
<td>30.9</td>
<td>41.4</td>
</tr>
<tr>
<td>✓</td>
<td>3</td>
<td>41.0</td>
<td>46.4</td>
<td>32.7</td>
<td>31.6</td>
<td>41.4</td>
</tr>
<tr>
<td>✓</td>
<td>5</td>
<td>41.0</td>
<td>46.5</td>
<td>32.9</td>
<td>32.1</td>
<td>41.7</td>
</tr>
<tr>
<td>✓</td>
<td>7</td>
<td>41.3</td>
<td>46.9</td>
<td>32.9</td>
<td>32.1</td>
<td>41.7</td>
</tr>
<tr>
<td>✓</td>
<td>9</td>
<td>41.3</td>
<td>46.8</td>
<td>32.9</td>
<td>32.1</td>
<td>41.8</td>
</tr>
</tbody>
</table>

Table 8. Comparison with different training schedules on the COCO val set. 1×, 2×, and 3× schedule denote the 90K, 180K, and 270K training iterations in Detectron2 [48], respectively.

<table>
<thead>
<tr>
<th>schedule</th>
<th>PQ</th>
<th>PQ^{th}</th>
<th>PQ^st</th>
<th>AP</th>
<th>mIoU</th>
</tr>
</thead>
<tbody>
<tr>
<td>1×</td>
<td>41.3</td>
<td>46.9</td>
<td>32.9</td>
<td>32.1</td>
<td>41.7</td>
</tr>
<tr>
<td>2×</td>
<td>43.2</td>
<td>48.8</td>
<td>34.7</td>
<td>34.3</td>
<td>43.4</td>
</tr>
<tr>
<td>3×</td>
<td>43.6</td>
<td>49.3</td>
<td>35.0</td>
<td>34.5</td>
<td>43.8</td>
</tr>
</tbody>
</table>

Table 9. Comparison with different settings of the feature encoder on the COCO val set. deform and channel num represent deformable convolutions and the channel number C_i, respectively.

<table>
<thead>
<tr>
<th>deform</th>
<th>channel num</th>
<th>PQ</th>
<th>PQ^{th}</th>
<th>PQ^st</th>
<th>AP</th>
<th>mIoU</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td>9</td>
<td>41.3</td>
<td>46.9</td>
<td>32.9</td>
<td>32.1</td>
<td>41.7</td>
</tr>
<tr>
<td>✓</td>
<td>3</td>
<td>41.0</td>
<td>46.4</td>
<td>32.7</td>
<td>31.6</td>
<td>41.4</td>
</tr>
<tr>
<td>✓</td>
<td>5</td>
<td>41.0</td>
<td>46.5</td>
<td>32.9</td>
<td>32.1</td>
<td>41.7</td>
</tr>
<tr>
<td>✓</td>
<td>7</td>
<td>41.3</td>
<td>46.9</td>
<td>32.9</td>
<td>32.1</td>
<td>41.7</td>
</tr>
<tr>
<td>✓</td>
<td>9</td>
<td>41.3</td>
<td>46.8</td>
<td>32.9</td>
<td>32.1</td>
<td>41.8</td>
</tr>
</tbody>
</table>

Table 10. Upper-bound analysis on the COCO val set. gt position and gt class denote utilizing the ground-truth position G_i and class O_i in each position head for kernel generation, respectively.

<table>
<thead>
<tr>
<th>gt position</th>
<th>gt class</th>
<th>PQ</th>
<th>PQ^{th}</th>
<th>PQ^st</th>
<th>AP</th>
<th>mIoU</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td>✓</td>
<td>43.6</td>
<td>49.3</td>
<td>35.0</td>
<td>34.5</td>
<td>43.8</td>
</tr>
<tr>
<td>✓</td>
<td>✗</td>
<td>49.8</td>
<td>52.2</td>
<td>46.1</td>
<td>38.2</td>
<td>54.6</td>
</tr>
<tr>
<td>✓</td>
<td>✓</td>
<td>65.9</td>
<td>64.1</td>
<td>68.7</td>
<td>45.5</td>
<td>86.6</td>
</tr>
</tbody>
</table>

Figure 3. Speed-Accuracy trade-off curve on the COCO val set. All results are compared with Res50 except DeeperLab [51] based on Xception-71 [7]. The latency is measured end-to-end from single input to panoptic result. Details are given in Table 11.

Kernel fusion. Kernel fusion is a core operation in the proposed method, which guarantees the required properties for things and stuff, as elaborated in Sec. 3.2. We investigate the fusion type class-aware and similarity thresholds thres in Table 3. As shown in the table, the network attains the best performance with thres 0.90. And the class-agnostic manner could dismiss some similar instances with different categories, which yields drop in AP. Furthermore, we compare kernel fusion with Matrix NMS [46] which is utilized for pixel-level removal. As presented in Table 4, the performance saturates with the simple kernel-level fusion method, and extra NMS brings no more gain.

Feature encoder. To enhance expressiveness of the encoded feature F_e, we further explore the channel number and feature type used in feature encoder. As illustrated in Table 5, the network achieves 41.3% PQ with 64 channels, and extra channels contribute little improvement. For effi-
Table 11. Comparisons with previous methods on the COCO val set. Panoptic FCN-400, 512, and 600 denotes utilizing smaller input instead of the default setting. All our results are achieved on the same device with single input and no flipping. FPS is measured end-to-end from single input to panoptic result with an average speed over 1,000 images, which could be further improved with more optimizations. The simple enhanced version is marked with *. The model testing by ourselves according to released codes is denoted as †.

<table>
<thead>
<tr>
<th>Method</th>
<th>Backbone</th>
<th>PQ</th>
<th>SQ</th>
<th>RQ</th>
<th>PQ&lt;sup&gt;th&lt;/sup&gt;</th>
<th>SQ&lt;sup&gt;th&lt;/sup&gt;</th>
<th>RQ&lt;sup&gt;th&lt;/sup&gt;</th>
<th>PQ&lt;sup&gt;st&lt;/sup&gt;</th>
<th>SQ&lt;sup&gt;st&lt;/sup&gt;</th>
<th>RQ&lt;sup&gt;st&lt;/sup&gt;</th>
<th>Device FPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Panoptic FPN</td>
<td>Res50-FPN</td>
<td>39.0</td>
<td>-</td>
<td>-</td>
<td>45.9</td>
<td>-</td>
<td>-</td>
<td>28.7</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Panoptic FPN&lt;sup&gt;3×&lt;/sup&gt;</td>
<td>Res50-FPN</td>
<td>39.4</td>
<td>77.8</td>
<td>48.3</td>
<td>45.9</td>
<td>80.9</td>
<td>55.3</td>
<td>29.6</td>
<td>73.3</td>
<td>37.7</td>
<td>V100 17.5</td>
</tr>
<tr>
<td>Panoptic FPN&lt;sup&gt;3×&lt;/sup&gt;</td>
<td>Res50-FPN</td>
<td>41.5</td>
<td>79.1</td>
<td>50.5</td>
<td>48.3</td>
<td>82.2</td>
<td>57.9</td>
<td>31.2</td>
<td>74.4</td>
<td>39.5</td>
<td>V100 17.5</td>
</tr>
<tr>
<td>AU-Net [25]</td>
<td>Res50-FPN</td>
<td>39.6</td>
<td>-</td>
<td>-</td>
<td>49.1</td>
<td>-</td>
<td>-</td>
<td>25.2</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>C1AE [11]</td>
<td>Res50-FPN</td>
<td>40.2</td>
<td>-</td>
<td>-</td>
<td>45.3</td>
<td>-</td>
<td>-</td>
<td>32.3</td>
<td>-</td>
<td>-</td>
<td>2080Ti 12.5</td>
</tr>
<tr>
<td>UPS-Net&lt;sup&gt;†&lt;/sup&gt; [50]</td>
<td>Res50-FPN</td>
<td>42.5</td>
<td>78.0</td>
<td>52.5</td>
<td>48.6</td>
<td>79.4</td>
<td>59.6</td>
<td>33.4</td>
<td>75.9</td>
<td>41.7</td>
<td>V100 9.1</td>
</tr>
<tr>
<td>Unifying [24]</td>
<td>Res50-FPN</td>
<td>43.4</td>
<td>79.6</td>
<td>53.0</td>
<td>48.6</td>
<td>-</td>
<td>-</td>
<td>35.5</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Panoptic FCN-400</td>
<td>Res50-FPN</td>
<td>40.7</td>
<td>80.5</td>
<td>49.3</td>
<td>44.9</td>
<td>82.0</td>
<td>54.0</td>
<td>34.3</td>
<td>78.1</td>
<td>42.1</td>
<td>V100 10.6</td>
</tr>
<tr>
<td>Panoptic FCN-512</td>
<td>Res50-FPN</td>
<td>42.3</td>
<td>80.9</td>
<td>51.2</td>
<td>47.4</td>
<td>82.1</td>
<td>56.9</td>
<td>34.7</td>
<td>79.1</td>
<td>42.7</td>
<td>V100 18.9</td>
</tr>
<tr>
<td>Panoptic FCN-600</td>
<td>Res50-FPN</td>
<td>42.8</td>
<td>80.6</td>
<td>51.6</td>
<td>47.9</td>
<td>82.6</td>
<td>57.2</td>
<td>35.1</td>
<td>77.4</td>
<td>43.1</td>
<td>V100 16.8</td>
</tr>
<tr>
<td>Panoptic FCN</td>
<td>Res50-FPN</td>
<td>43.6</td>
<td>80.6</td>
<td>52.6</td>
<td>49.3</td>
<td>82.6</td>
<td>58.9</td>
<td>35.0</td>
<td>77.6</td>
<td>42.9</td>
<td>V100 12.5</td>
</tr>
<tr>
<td>Panoptic FCN*</td>
<td>Res50-FPN</td>
<td>44.3</td>
<td>80.7</td>
<td>53.0</td>
<td>50.0</td>
<td>83.4</td>
<td>59.3</td>
<td>35.6</td>
<td>76.7</td>
<td>43.5</td>
<td>V100 9.2</td>
</tr>
</tbody>
</table>

**Upper-bound analysis.** In Table 10, we give analysis to the upper-bound of generate-kernel-then-segment fashion with Res50-FPN backbone on the COCO val set. As illustrated in the table, given ground truth positions of object centers \( L_i^{th} \) and stuff regions \( L_i^{st} \), the network yields 6.2% PQ from more precise locations. And it will bring extra boost (16.1% PQ) to the network if we assign ground truth categories to the position head. Compared with the baseline method, there still remains huge potential to be explored (22.3% PQ in total), especially for stuff regions which could have even up to 33.7% PQ and 42.8% mIoU gains.

**Speed-accuracy.** To verify the network efficiency, we plot the end-to-end speed-accuracy trade-off curve on the COCO val set. As presented in Fig. 3, the proposed Panoptic FCN surpasses all previous box-free models by large margins on both performance and efficiency. Even compared with the well-optimized Panoptic FPN [18] from Detectron2 [48], our approach still attains a better speed-accuracy balance with different image scales. Details about these data points are included in Table 11.

### 4.3. Main Results

We further conduct experiments on different scenarios, namely COCO dataset for common context, Cityscapes and Mapillary Vistas datasets for traffic-related environments.
In Table 12, Experiments on the COCO test-dev set. All our results are achieved with single scale input and no flipping. The simple enhanced version and val set for training are marked with * and \].

<table>
<thead>
<tr>
<th>Method</th>
<th>Backbone</th>
<th>PQ</th>
<th>PQ\textsuperscript{th}</th>
<th>PQ\textsuperscript{st}</th>
</tr>
</thead>
<tbody>
<tr>
<td>box-based</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Panoptic FPN [18]</td>
<td>Res101-FPN</td>
<td>40.9</td>
<td>48.3</td>
<td>29.7</td>
</tr>
<tr>
<td>CIAE [11]</td>
<td>DCN101-FPN</td>
<td>44.5</td>
<td>49.7</td>
<td>36.8</td>
</tr>
<tr>
<td>AUNet [25]</td>
<td>ResNetX152-FPN</td>
<td>46.5</td>
<td>\textbf{55.8}</td>
<td>32.5</td>
</tr>
<tr>
<td>UPSNet [50]</td>
<td>DCN101-FPN</td>
<td>46.6</td>
<td>53.2</td>
<td>36.7</td>
</tr>
<tr>
<td>Unifying\textsuperscript{1} [24]</td>
<td>DCN101-FPN</td>
<td>47.2</td>
<td>53.5</td>
<td>37.7</td>
</tr>
<tr>
<td>BANet [5]</td>
<td>DCN101-FPN</td>
<td>47.3</td>
<td>54.9</td>
<td>35.9</td>
</tr>
<tr>
<td>box-free</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DeeperLab [51]</td>
<td>Xception-71</td>
<td>34.3</td>
<td>37.5</td>
<td>29.6</td>
</tr>
<tr>
<td>SSAP [10]</td>
<td>Res101-FPN</td>
<td>36.9</td>
<td>40.1</td>
<td>32.0</td>
</tr>
<tr>
<td>PCV [43]</td>
<td>Res50-FPN</td>
<td>37.7</td>
<td>40.7</td>
<td>33.1</td>
</tr>
<tr>
<td>Panoptic-DeepLab [6]</td>
<td>Xception-71</td>
<td>39.7</td>
<td>43.9</td>
<td>33.2</td>
</tr>
<tr>
<td>AdaptIS [40]</td>
<td>ResNeXt-101</td>
<td>42.8</td>
<td>53.2</td>
<td>36.7</td>
</tr>
<tr>
<td>Axial-DeepLab\textsuperscript{4} [44]</td>
<td>Axial-ResNet-L</td>
<td>43.6</td>
<td>48.9</td>
<td>35.6</td>
</tr>
<tr>
<td>Panoptic FCN</td>
<td>Res101-FPN</td>
<td>45.5</td>
<td>51.4</td>
<td>36.4</td>
</tr>
<tr>
<td>Panoptic FCN\textsuperscript{*}</td>
<td>DCN101-FPN</td>
<td>47.0</td>
<td>53.0</td>
<td>37.8</td>
</tr>
<tr>
<td>Panoptic FCN\textsuperscript{**}</td>
<td>DCN101-FPN</td>
<td>47.5</td>
<td>53.7</td>
<td>\textbf{38.2}</td>
</tr>
</tbody>
</table>

Table 13. Experiments on the Cityscape val set. All our results are achieved with single scale input and no flipping. The simple enhanced version is marked with *.

<table>
<thead>
<tr>
<th>Method</th>
<th>Backbone</th>
<th>PQ</th>
<th>PQ\textsuperscript{th}</th>
<th>PQ\textsuperscript{st}</th>
</tr>
</thead>
<tbody>
<tr>
<td>box-based</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Panoptic FPN [18]</td>
<td>Res101-FPN</td>
<td>58.1</td>
<td>52.0</td>
<td>62.5</td>
</tr>
<tr>
<td>AUNet [25]</td>
<td>Res101-FPN</td>
<td>59.0</td>
<td>54.8</td>
<td>62.1</td>
</tr>
<tr>
<td>UPSNet [50]</td>
<td>Res50-FPN</td>
<td>59.3</td>
<td>54.6</td>
<td>62.7</td>
</tr>
<tr>
<td>SOGNet [52]</td>
<td>Res50-FPN</td>
<td>60.0</td>
<td>\textbf{56.7}</td>
<td>62.5</td>
</tr>
<tr>
<td>Seamless [36]</td>
<td>Res50-FPN</td>
<td>60.2</td>
<td>55.6</td>
<td>63.6</td>
</tr>
<tr>
<td>Unifying [24]</td>
<td>Res50-FPN</td>
<td>61.4</td>
<td>54.7</td>
<td>66.3</td>
</tr>
<tr>
<td>box-free</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PCV [43]</td>
<td>Res50-FPN</td>
<td>54.2</td>
<td>47.8</td>
<td>58.9</td>
</tr>
<tr>
<td>DeeperLab [51]</td>
<td>Xception-71</td>
<td>56.5</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>SSAP [10]</td>
<td>Res50-FPN</td>
<td>58.4</td>
<td>50.6</td>
<td>-</td>
</tr>
<tr>
<td>AdaptIS [40]</td>
<td>Res50</td>
<td>59.0</td>
<td>55.8</td>
<td>61.3</td>
</tr>
<tr>
<td>Panoptic-DeepLab [6]</td>
<td>Res50</td>
<td>59.7</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Panoptic FCN</td>
<td>Res50-FPN</td>
<td>59.6</td>
<td>52.1</td>
<td>65.1</td>
</tr>
<tr>
<td>Panoptic FCN\textsuperscript{*}</td>
<td>Res50-FPN</td>
<td>\textbf{61.4}</td>
<td>54.8</td>
<td>\textbf{66.6}</td>
</tr>
</tbody>
</table>

47.5% PQ with single scale inputs. Compared with the similar box-free fashion, our method improves 1.9% PQ over Axial-DeepLab [44] which adopts stronger backbone.

Cityscapes. Furthermore, we carry out experiments on Cityscapes val set in Table 13. Panoptic FCN exceeds the top box-free model [6] with 1.7% PQ and attains 61.4% PQ. Even compared with the leading box-based model [24], which utilizes Lovasz loss for further optimization, the proposed method still achieves comparable performance.

Mapillary Vistas. In Table 14, we compare with other state-of-the-art models on the large-scale Mapillary Vistas val set with Res50-FPN backbone. As presented in the table, the proposed Panoptic FCN surpasses the leading box-free methods by a large margin in both things and stuff. Specifically, Panoptic FCN surpasses the leading box-based [36] and box-free [6] models with 0.7% and 3.6% PQ, and attains 36.9% PQ with simple enhancement in the feature encoder.

5. Conclusion

We have presented the Panoptic FCN, a conceptually simple yet effective framework for panoptic segmentation. The key difference from prior works lies on that we represent and predict things and stuff in a fully convolutional manner. To this end, kernel generator and kernel fusion are proposed to generate the unique kernel weight for each object instance or semantic category. With the high-resolution feature produced by feature encoder, prediction is achieved by convolutions directly. Meanwhile, instance-awareness and semantic-consistency for things and stuff are respectively satisfied with the designed workflow.

6. Acknowledgment

This research was partially supported by National Key R&D Program of China (No. 2017YFA0700800), and Beijing Academy of Artificial Intelligence (BAAI).


