MaskConver: Revisiting Pure Convolution Model for Panoptic Segmentation

Abdullah Rashwan  
Google  
arashwan@google.com

Jiageng Zhang  
Google  
jiageng@google.com

Ali Taalimi  
Google  
taalimi@google.com

Fan Yang  
Google  
fyangf@google.com

Xingyi Zhou  
Google  
zhouxy@google.com

Chaochao Yan  
Google  
allenyan@google.com

Liang-Chieh Chen  
ByteDance *  
liang.chieh.chen@bytedance.com

Yeqing Li  
Google  
yeqing@google.com

Abstract

In recent years, transformer-based models have dominated panoptic segmentation, thanks to their strong modeling capabilities and their unified representation for both semantic and instance classes as global binary masks. In this paper, we revisit pure convolution model and propose a novel panoptic architecture named MaskConver. MaskConver proposes to fully unify things and stuff representation by predicting their centers. To that extent, it creates a lightweight class embedding module that can break the ties when multiple centers co-exist in the same location. Furthermore, our study shows that the decoder design is critical in ensuring that the model has sufficient context for accurate detection and segmentation. We introduce a powerful ConvNeXt-U-Net decoder that closes the performance gap between convolution- and transformer-based models. With ResNet50 backbone, our MaskConver achieves 53.6% PQ on the COCO panoptic val set, outperforming the modern convolution-based model, Panoptic FCN, by 9.3% as well as transformer-based models such as Mask2Former (+1.7% PQ) and kMaX-DeepLab (+0.6% PQ). Additionally, MaskConver with a MobileNet backbone reaches 37.2% PQ, improving over Panoptic-DeepLab by +6.4% under the same FLOPs/latency constraints. A further optimized version of MaskConver achieves 29.7% PQ, while running in real-time on mobile devices. The code and model weights will be publicly available.

1. Introduction

Panoptic segmentation [38] aims to unify instance [23] and semantic segmentation [25] in the same framework. Existing works propose to merge instance and semantic seg-

*Work done while at Google.


Figure 1. MaskConver versus existing panoptic models. MaskConver successfully bridges the gap between the modern convolution-based method, Panoptic FCN, and the transformer-based methods, Mask2Former and kMaX-DeepLab. In the efficient model setting, MaskConver outperforms Panoptic-DeepLab under the same FLOPs/latency constraints.

Efficient models

<table>
<thead>
<tr>
<th>Model</th>
<th>ResNet50 models</th>
<th>Efficient models</th>
</tr>
</thead>
<tbody>
<tr>
<td>MaskConver</td>
<td>53.6</td>
<td>37.2</td>
</tr>
<tr>
<td>PanopticFCN</td>
<td>44.3</td>
<td>30.8</td>
</tr>
<tr>
<td>kMaX-DeepLab</td>
<td>53.0</td>
<td></td>
</tr>
<tr>
<td>Mask2Former</td>
<td>51.9</td>
<td></td>
</tr>
<tr>
<td>PanopticDeepLab</td>
<td>49.0</td>
<td></td>
</tr>
</tbody>
</table>

These architectures however rely on many customized components like non-maximum suppression (NMS), and thing-stuff merging heuristics to produce panoptic outputs. Recent works [11,12,47,72,82,83] unify both segmentation tasks by producing binary masks and class scores for both things and stuff classes. Such universal architectures result in a simpler post-processing logic and make the loss closely correlated with the panoptic quality (PQ) metric. As a result, they have achieved significantly higher PQ numbers compared to traditional architectures.

Among these unified panoptic segmentation models, transformers [71] have played a critical role due to their ability to learn instance-level embeddings via a transformer decoder. DETR [6] was introduced for object detection by learning object embeddings, each of which predicts an object class and a bounding box. The idea of instance-level embeddings influenced many of the transformer-based panoptic models. Following DETR, MaX-DeepLab [72] uses a transformer decoder to learn mask embeddings [33, 67, 76] to predict a set of binary masks. The binary masks...
are then merged using a simple post-processing layer [77] to filter out duplicates, similar to NMS. Other architectures [11, 12, 47, 82, 83] follow a similar paradigm, but further improved the performance by developing modern transformer decoders [91]. The commonality between these methods is the employment of transformer blocks to learn a set of binary masks and their corresponding classes (hence the panoptic masks). On the other hand, the convolution-based methods [10, 46] lag behind in performance. It is yet unclear if using transformers justifies the quality gap compared to convolution-based models.

In this work, we revisit the pure convolution panoptic models [10, 46] and propose a novel architecture for panoptic segmentation (Fig. 2), named MaskConver, which produces segmentation masks for thing and stuff classes in a unified way. The meta architecture of MaskConver contains four main components: backbone, pixel decoder, prediction heads, and mask embedding generator. The backbone is a typical ImageNet [61] pretrained convolutional neural network (ConvNet) [40], such as ResNet [24]. We design a novel pixel-decoder, ConvNeXt-UNet, which deploys ConvNeXt blocks [54] in a manner similar to UNet decoder [60] but in an asymmetric way. Particularly, in the decoder, we discover that it is critical to stack more ConvNeXt blocks at the highest level (i.e., level 5 with stride 32), which benefits the model to effectively learn context information. The prediction heads include three predictions: Center Heatmap Prediction, Center Embedding Prediction, and Mask Feature Prediction. The Center Heatmap prediction predicts the center point heatmaps [90] for both things and stuff. We utilize the mask centers instead of box centers to represent both things and stuff. The Center Embedding Head generates the embeddings for the center points, while the Mask Feature Head produces the mask features. Finally, the Mask Embedding Generator aims to generate high-quality mask embeddings by taking into account the “instance collision”, where the center points of neighboring instances may collide, yielding the same (indistinguishable) center embeddings. To alleviate the issue, it first produces the class embeddings (via Class Embedding Lookup Table) by taking the predicted semantic classes of the center points. The output mask embeddings are then obtained by modulating the center embeddings with the class embeddings (via addition and MLP) to a different space conditioned on the semantic class of the instance. Finally, the mask embeddings are multiplied with the mask features to produce segmentation masks for things and stuff in a unified way.

We evaluate the quality of MaskConver in several settings on COCO panoptic segmentation dataset [50]. On the COCO validation set, our proposed ConvNeXt-UNet pixel decoder improves the PQ of the solid pixel decoder baseline BiFPN [66] by +3.1%, while being 18% more efficient on FLOPs. When using the ResNet50 backbone [24], MaskConver achieves 53.6% and runs at 19.6 FPS on a V100 GPU. Our model demonstrates significant gains (+9.3%) compared to the modern convolution-based method, Panoptic FCN [46], while also outperforming the transformer-based models like Mask2Former [11] (+1.7%) and kMaX-DeepLab [83] (+0.6%). When using the MobileNet backbone [29], MaskConver achieves 37.2% PQ, which is 6.4% better than Panoptic-DeepLab [10]. In addition, after further optimization via quantization, MaskConver runs at 30 FPS on Pixel 6 GPU, while achieving 29.7% PQ.

2. Related Work

Since the proposal of panoptic segmentation in [38], numerous efforts have emerged in this domain. Initiatives began with adaptations to existing networks, adding either a semantic [37] or an instance branch [10] to state-of-the-art models. These methods established a baseline using hand-crafted post-processing layers [37, 51, 78, 81] for final panoptic predictions. Following these works, researchers start to think about architectures that can solve the task in a more unified way. MaX-DeepLab [72] learns mask embeddings [33, 67, 76] in the transformer framework for both thing and stuff classes. These mask embeddings predict a set of binary masks, and a post-processing layer [77] is used to predict the final panoptic outputs. MaskFormer [12] proposes a similar paradigm, and shows how to use the same model for both semantic and panoptic segmentation by only modifying the post-processing logic. Panoptic-Segformer [47] extends Deformable DETR [91] for the panoptic segmentation task. CMT-DeepLab [82] reformulates the transformer cross-attention from the clustering perspective. Mask2Former [11] proposes masked-attention to significantly outperform MaskFormer architecture on smaller objects by masking unrelated parts of the image. kMaX-DeepLab [83] improves MaX-DeepLab by reformulating the cross attention layers to mimic k-means algorithm. Similarly, our MaskConver learns mask embeddings for both thing and stuff classes, but only uses fully convolutional layers [55] without any transformer blocks.

Transformers have surpassed ConvNets on several vision problems beyond panoptic segmentation [32, 42] including classification [16, 18, 20, 52, 53, 75, 80], detection [6, 21, 41, 44, 84], and segmentation [17, 63, 73, 79, 88]. Recently, ConvNeXt [54], a pure convolution-based backbone, provides competitive performance compared to transformer architectures, while preserving the simplicity and efficiency of ConvNets. ConvNeXt block adopts depthwise convolutions with a large kernel size $7 \times 7$, layer norm [2], and layer scale [69]. In this work, we adopt the ConvNeXt block as a building block for our pixel decoder. The design is further improved by using Squeeze-and-Excitation [31] layer that improves the performance with little impact on
Figure 2. Illustration of MaskConver architecture. The meta architecture of MaskConver contains four components: backbone (gray), pixel decoder (pink), prediction heads (light blue), and mask embedding generator (green). The backbone is any commonly deployed neural network, e.g., ResNet50. We propose a novel ConvNeXt-UNet for the pixel decoder, which effectively captures long-range context and high-level semantics by stacking many ConvNeXt blocks at the highest level of backbone. We propose three prediction heads: Center Heatmap Head (for predicting center point heatmaps), Center Embedding Head (for predicting the embeddings for center points), and Mask Feature Head (for generating mask features). The Mask Embedding Generator first produces the class embeddings via a lookup table (Class Embedding Lookup Table module) by taking the predicted semantic classes from the top-K center points. The output mask embeddings are obtained by modulating the class embeddings with the center embeddings (via addition and MLP) to mitigate the center point collision between instances of different classes. In the end, the mask features are multiplied with the mask embeddings to generate the final binary masks. Unlike transformer-based methods, MaskConver only exploits convolutions without any self- or cross-attentions.

Panoptic FCN [46] employs the fully convolutional architecture [55] that predicts things and stuff classes in a unified way, through the convolutional kernel generator to predict things and stuff kernels. These kernels are used to predict binary masks. Although Panoptic FCN unifies things and stuff classes towards the post-processing, the kernel generator still treats things and stuff differently (particularly, they represent things as centers, but stuff as regions). Panoptic FCN’s quality is lagging behind the recent transformer-based panoptic models. To bridge the gap, MaskConver proposes to fully unify the architecture by only relying on things and stuff centers. It creates a lightweight class embedding module that can break the ties when multiple centers co-exist in the same location. MaskConver also introduces a novel pixel decoder (ConvNeXt-UNet), which provides the model with sufficient context to produce high quality centers and mask predictions.

Efficient Panoptic Segmentation models are less explored, since most architectures focus more on pushing the panoptic quality instead of having more efficient architectures. Panoptic-DeepLab [8,10] reports quality and latency numbers on a V100 GPU when using MobileNetV3 backbone [29] on an image size of 640 × 640. Hou et al. [28] propose a single-shot panoptic segmentation model that runs in real-time on a V100 GPU. In this work, we tailor MaskConver architecture for mobile devices (Pixel 6) through a set of architectural design choices.

Various lightweight model backbones, including MobileNet [29,30,62], EfficientNet [64,65], and ShuffleNet [57,86], have been proposed. They are designed for low computational power devices, like mobile CPU and GPU. A multi-hardware MobileNet (MobileNet-MH) [14], discovered by neural architecture search [4,92] and optimized for multiple hardware [5], achieves state-of-the-art latency and accuracy trade-off on a variety of mobile devices, and has been adopted as the backbone for the semantic segmentation task [74]. To evaluate MaskConver for mobile use cases, we use MobileNet-MH since it delivers similar accuracy as MobileNetV3 [29], while being more compatible with different mobile devices. Center Point Representation is used in tasks like 2D detection, tracking, action recognition, instance segmentation and panoptic segmentation [34,45,81,89,90]. In this work, we propose to utilize the center point to model both things and stuff. Additionally, we propose to utilize the mask cen-
3. Method

Fig. 2 shows the meta architecture of MaskConver, containing four main components: backbone, pixel decoder, prediction heads, and mask embedding generator.

Overview. The backbone is a typical convolutional neural network, such as ResNet [24] and MobileNet [30]. A novel pixel decoder ConvNeXt-UNet is proposed to generate the image features, on top of which prediction heads are appended. We propose three prediction heads: (1) Center Heatmap Head that predicts center point heatmaps [90] for both things and stuff, (2) Center Embedding Head that predicts embeddings for the center points, and (3) Mask Feature Head that yields mask features. The Mask Embedding Generator generates the mask embeddings by taking as input both the top-K confident predicted centers (their semantic classes and coordinates) and the center embeddings. In the end, a set of binary masks are obtained by multiplying the mask features with the mask embeddings [72]. We will first explain our design motivations before detailing the proposed modules in the following subsections.

Motivations. Convolution-based models [10, 46] lag in performance, compared to transformer-based models [11, 83]. We carefully look into the mask transformer frameworks [11, 72, 83], and discover that the advantages of employing transformer blocks [71] are twofold: the attention mechanism [3] effectively enriches the pixel decoder with long-range information (thus generates high quality masks) and the object queries produce thing and stuff segmentation masks in a unified way. The observation motivates us to revisit the existing convolution-based methods [10, 46] by designing a better pixel decoder (Sec. 3.1), prediction heads (Sec. 3.2), and mask embedding generator (Sec. 3.3).

3.1. Pixel Decoder: ConvNeXt-UNet

To bridge the gap with transformer-based methods, we first design a novel pixel decoder ConvNeXt-UNet, as shown in Fig. 3, consisting of the modern ConvNeXt blocks [54] deployed in a manner similar to UNet [60] to generate image features. Notably, ConvNeXt-UNet deploys more ConvNeXt blocks at the highest level \( L_5 \) of backbone (stride 32). Thanks to the large kernel design, stacking more ConvNeXt blocks at level \( L_5 \) effectively captures long-range context information and high-level semantics. Specifically, the decoder architecture is defined by two hyper-parameters: number of repeats, \( N = [N_5, N_4, N_3] \), and channel sizes, \( D = [D_5, D_4, D_3] \), determining the UNet structure from high level \( L_5 \) (stride 32) to low level \( L_3 \) (stride 8). For example, setting \( N_5 = 18 \) and \( D_5 = 384 \) means the deployment of 18 ConvNeXt blocks with 384 channels at the level \( L_5 \). Additionally, we empirically find it effective to add another Squeeze-and-Excitation [31] layer in the ConvNeXt block (called ConvNeXt-SE), as shown in Fig. 4, which improves the model capacity at the cost of extra marginal parameters and negligible FLOPs.

3.2. Prediction Heads

On top of the image features generated by the proposed pixel decoder, we build three prediction heads for center heatmaps, class embeddings, and mask features. Below, we first explain the structure of our prediction heads.

Light Structure of Head. Unlike existing methods [7, 49, 90] that commonly employ \( 3 \times 3 \) convolutions in the prediction heads and introduce heavy computations on low level features (i.e., stride 8 or even stride 4 features), MaskConver, following the design principle of ConvNeXt [54], adopts depthwise convolutions with a large kernel size \( 7 \times 7 \) (along with layer normalization [2] and GeLU activation function [26]). This design reduces the FLOPs significantly with no degradation in the panoptic quality.
**Center Heatmap Head.** Extending object detection methods [68, 90], we propose to use center point representation for both things and stuff. We empirically discover that mask center is a better representation than bounding box center. The Center Heatmap Head produces a feature map with shape $H_i \times W_i \times C_{\text{class}}$, where $H_i$ and $W_i$ are the height and width of $i$-th level feature map in the feature pyramid [48], and $C_{\text{class}}$ is the number of semantic classes. We will feed the top-K most confident predicted center points (their predicted semantic classes and coordinates) to the Mask Embedding Generator.

**Center Embedding Head.** The Center Embedding Head generates the embeddings for center points with shape $H_i \times W_i \times C_{\text{embd}}$, where $C_{\text{embd}}$ is the channel size of embeddings. Its output is fed into the Mask Embedding Generator to gather K center embeddings for the top-K most confident predicted center points (based on their coordinates).

**Mask Feature Head.** The Mask Feature Head combines the decoder features from L_5 to L_3 to create the mask features. This is done by resizing all the decoder features to the same size (stride 4) and summing them together, before feeding to the light prediction head. The resulting mask features have shape $H/4 \times W/4 \times C_{\text{embd}}$, where $H$ and $W$ are the height and width of input image, respectively. The mask features, multiplied with the mask embeddings (from the Mask Embedding Generator, detailed in Sec. 3.3), generate the final output: a set of K binary masks.

### 3.3. Mask Embedding Generator

The Mask Embedding Generator is one of the crucial designs in MaskConv, aiming to generate high quality mask embeddings. It takes as input both the top-K most confident predicted centers from the Center Heatmap Head (their semantic classes and coordinates) and center embeddings from the Center Embedding Head.

An na"ive design would be the simple gathering of the K center embeddings based on the top-K center coordinates (i.e., directly use the K center embeddings as mask embeddings). However, it results in an inferior performance, as we observe the confusion caused by neighboring instances, especially when their centers collide, leading to exactly the same embedding vector being gathered from the output of Center Embedding Head.

Therefore, we propose to additionally exploit the class embeddings, which learn to embed each semantic class to the vector of size $C_{\text{embd}}$. The class embeddings are used to modulate (via addition and a MLP) the center embeddings, mitigating the center collisions caused by instances of different semantic classes. Specifically, we design a “Class Embedding Lookup Table” module, which stores the learned embeddings for semantic classes. For the top-K centers, we infer their most likely semantic classes, and obtain their corresponding class embeddings from the module.

We then add the obtained class embeddings and the center embeddings, and pass them to a MLP module (two fully-connected layers) to generate the final mask embeddings. We note that exploiting the class embeddings is critical to the quality of the predicted mask embeddings. It ensures that each instance will have a unique embedding vector, avoiding the problem of instance center collision.

Fig. 5 visualizes the learned class embeddings for things and stuff, using tSNE [70]. As shown in the figure, there are two well-separated clusters: one for things and the other for stuff. As a result, the center collision between things and stuff is alleviated by adding the class embeddings and center embeddings, forming better mask embeddings to generate high quality masks. We will show in experiments that class embeddings provide a decent performance improvements.

### 4. Experimental Results

We evaluate the effectiveness of the proposed MaskConv on the challenging COCO dataset [50]. COCO has 118,287 training images and 5,000 validation images. It has 80 thing categories and 53 stuff categories. The code and model weights will be publicly available.

### 4.1. Implementation Details

**Training Strategy.** The Center Heatmap Head is supervised by the Focal loss [49], while the final mask prediction is supervised by binary cross entropy loss and Dice loss [58]. The total loss is thus defined as: $\text{Loss}_{\text{total}} = \lambda_{\text{centers}} \text{Loss}_{\text{centers}} + \lambda_{\text{bce}} \text{Loss}_{\text{bce}} + \lambda_{\text{dice}} \text{Loss}_{\text{dice}}$. Across all experiments and unless stated otherwise, we fix the weighting factors as $\lambda_{\text{centers}} = 1$, $\lambda_{\text{bce}} = 10$, and $\lambda_{\text{dice}} = 10$. We use Adam weight decay optimizer (AdamW) [36, 56] with learning rate of 0.001, and weight decay of 0.05. Exponential Moving Average (EMA) [59] optimizer is used with average decay of 0.99996. We train the models for 270k it-
<table>
<thead>
<tr>
<th>Architecture</th>
<th>Backbone</th>
<th>Params</th>
<th>FLOPs</th>
<th>PQ</th>
<th>PQ_{thing}</th>
<th>PQ_{stuff}</th>
<th>FPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>DETR [6]</td>
<td>ResNet50 [24]</td>
<td>-</td>
<td>-</td>
<td>43.4</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>K-Net [85]</td>
<td>ResNet50 [24]</td>
<td>-</td>
<td>-</td>
<td>47.1</td>
<td>51.7</td>
<td>40.3</td>
<td>-</td>
</tr>
<tr>
<td>CMT-DeepLab [82]</td>
<td>ResNet50 [24]</td>
<td>-</td>
<td>-</td>
<td>48.5</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Panoptic SegFormer [47]</td>
<td>ResNet50 [24]</td>
<td>51M</td>
<td>214B</td>
<td>49.6</td>
<td>54.4</td>
<td>42.4</td>
<td>7.8</td>
</tr>
<tr>
<td>kMaX-DeepLab [83]</td>
<td>ResNet50 [24]</td>
<td>57M</td>
<td>168B</td>
<td>53.0</td>
<td>58.3</td>
<td>44.9</td>
<td>22.8</td>
</tr>
<tr>
<td>Transformer Based</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Real-time Panoptic [10]†</td>
<td>ResNet50 [28]</td>
<td>-</td>
<td>-</td>
<td>37.1</td>
<td>41.0</td>
<td>31.3</td>
<td>16</td>
</tr>
<tr>
<td>SOLO-V2 [76]</td>
<td>ResNet50 [24]</td>
<td>-</td>
<td>-</td>
<td>42.1</td>
<td>49.6</td>
<td>30.7</td>
<td>-</td>
</tr>
<tr>
<td>Unifying [43]</td>
<td>ResNet50 [24]</td>
<td>-</td>
<td>-</td>
<td>43.3</td>
<td>48.6</td>
<td>35.5</td>
<td>-</td>
</tr>
<tr>
<td>Panoptic FCN [46]</td>
<td>ResNet50 [24]</td>
<td>-</td>
<td>-</td>
<td>44.3</td>
<td>50.0</td>
<td>35.6</td>
<td>9.2</td>
</tr>
<tr>
<td>MaskConver (ours)</td>
<td>ResNet50 [24]</td>
<td>57M</td>
<td>171B</td>
<td>53.6</td>
<td>58.9</td>
<td>45.6</td>
<td>19.6</td>
</tr>
<tr>
<td>Convolutional Based</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| MaskConver w/ ResNet50 Backbone. We provide more details for MaskConver architecture, when using the ResNet50 backbone [24]. For the proposed ConvNeXt-UNet pixel decoder, we set N = [18, 1, 1] and D = [384, 384, 384], i.e., stacking 18 ConvNeXt-SE blocks with 384 channels at the L_5 level, and only one ConvNeXt-SE block at L_4 and L_3 levels. The Center Heatmap Head and Center Embedding Head are attached from levels L_3 to L_7, where extra strided 7 x 7 depthwise convolutions are applied to the backbone to get L_6 and L_7. **MaskConver w/ Efficient Backbone.** We make some changes, when deploying MaskConver with efficient backbones [14, 29]. The Center Heatmap Head and Center Embedding Head are only appended to a single scale, i.e., L_3 feature map in the feature pyramid. We experiment with MobileNetV3-Large [29] and multi-hardware MobileNet (MobileNet-MH) [14] as the backbone. The ConvNeXt-UNet pixel decoder is replaced with the simpler DeepLabv3+ decoder [9]. For efficiency, we use hard-sigmoid [15], since sigmoid is costly on mobile devices. The model is converted to TFLite models and benchmarked on mobile devices to obtain the latency. We also apply weight-only post-training quantization to further speed up the model by 2x.

### 4.2. Main Results

In Tab. 1, we compare the proposed MaskConver with other methods in three categories: convolution-based, transformer-based, and efficient models.
Convolution-Based Models. In the category of convolution-based Models (i.e., middle group in Tab. 1), MaskConver consistently outperforms all the other convolution-based methods in terms of both performance (PQ) and speed (FPS). Particularly, when comparing with the state-of-the-art Panoptic FCN [46], MaskConver is +9.3% PQ better and running 2.13 times faster.

Transformer-Based Models. When compared with transformer-based models (i.e., top group in Tab. 1), MaskConver achieves better PQ when using similar FLOPs/parameters. In particular, MaskConver is +1.7% better than Mask2Former [11], while also being faster on a V100 GPU. MaskConver is also +0.6% better than kMaX-DeepLab [83] with a slightly higher number of flops. These results suggest that with a better designed pixel decoder, prediction heads, and mask embedding generator, MaskConver can successfully bridge the gap between transformer- and convolution-based models.

Efficient Models. For efficient models [14, 29] (i.e., bottom group in Tab. 1), we compared MaskConver with Panoptic-DeepLab [10]. We employ Panoptic-DeepLab with the same MobileNet-MH backbone [14] and input size $640 \times 640$ to have a fair comparison. Our model with $640$ input image achieves +6.4% better PQ compared to Panoptic-DeepLab, while also being $1.42 \times$ times faster on a V100 GPU. Furthermore, if we change the input size to $256 \times 256$, our MaskConver-256 achieves a similar PQ to Panoptic-DeepLab ($29.7\%$ vs. $30.8\%$ PQ), while running $5.07 \times$ times faster. Our MaskConver-256 runs real-time on Pixel 6 GPU with 33 FPS.

4.3. Ablation Studies

We conduct the systematic ablation studies on the MaskConver architecture, using the ResNet50 backbone.

Pixel Decoder. In Tab. 2, we ablote on the design choices of pixel decoder. We start with the popular pixel decoder choice: FPN (feature pyramid network) [48] as our baseline, which attains the performance of 43.1% PQ. We then use the more advanced feature pyramid architecture, BiFPN (bi-directional feature pyramid network) [66], which improves the performance to 47.7% PQ with a reasonable increase in both FLOPs and model parameters, serving as another solid baseline. After setting the solid baselines, we explore the structure of the proposed ConvNeXt-UNet with two hyper-parameters: number of repeats of ConvNeXt blocks, $N = [N_5, N_4, N_3]$, and channel sizes, $D = [D_5, D_4, D_3]$, which determine the UNet structure from high level $L_5$ (stride 32) to low level $L_3$ (stride 8). We first set $N = [3, 9, 3]$ and $D = [768, 384, 192]$, which corresponds to the inverting ConvNeXt-Tiny [54] structure. This simple design surprisingly improves over the strong baseline BiFPN by +1.4% PQ, while also being slightly more efficient in FLOPs. Motivated by the prior works [7, 87] that employ the multi-scale context module at the highest level (i.e., stride 32) of backbone, we explore stacking more ConvNeXt blocks to level 5 to capture more long-range information and high-level semantics. Hence, we move most of the blocks to level 5 by setting $N = [11, 1, 1]$ and $D = [512, 383, 384]$, in order to keep the similar number of parameters. This structure further improves the performance by +1% PQ. To further push the envelope, we explore stacking more blocks at level 5 by using $N = [18, 1, 1]$ and $D = [384, 384, 384]$, which yields additional +0.3% improvement. Finally, employing the proposed ConvNeXt-SE block (i.e., adding another Squeeze-and-Excitation [31] layer in ConvNeXt block) improves the PQ by +0.4% with minor effect on the model FLOPs. Overall, we observe a +3.1% improvement in PQ compared to the solid baseline BiFPN [66] with 19% lower FLOPs, and +7.7% improvement in PQ compared to FPN.

Light Structure of Head. In this study, we evaluate the effectiveness of the adopted light structure of prediction heads (in Sec. 3.2). As shown in Tab. 3, the light prediction head using the $7 \times 7$ depthwise convolution slightly improves over the regular $3 \times 3$ convolution by +0.3% PQ with a significant reduction of 75% in FLOPs. This reduction is mainly because the convolution layers that process low-level features (e.g., stride 8 features) are very expensive. Replacing these regular convolution layers with depthwise convolution is more efficient for both accuracy and FLOPs.

Class Embeddings. The proposed class embeddings (generated by Class Embedding Lookup Table in Sec. 3.3) modulates the center embeddings (via addition and a MLP) to mitigate the instance collision. In this study, we evaluate its significance, using both ResNet50 and MobileNet-MH backbone, in Tab. 4. As shown in the table, using the class embedding (see column ‘Cls-Embd’) shows +1.5% and +2.3% improvement for ResNet50 and MobileNet-MH, respectively. These results suggest that using class embeddings helps break the tie, when two centers of different classes are present at the same location. Additionally, the improvement for MobileNet-MH is more significant, since MaskConver with MobileNet-MH uses only a single output scale ($L_3$) to predict the centers (for the purpose of efficiency), in which case we expect more center collisions and hence the class embeddings become more important.

Mask Centers vs. Box Centers. Unlike CenterNet [90], MaskConver uses mask centers, instead of bounding box centers. We ablate this design choice in Tab. 5, which shows +0.6% PQ improvement, when using the mask centers. As a result, the mask generates a better center representation than the bounding box.

Effect of Training Strategy. In this study, we evaluate the effect of several training techniques that are used in our framework. As shown in Tab. 6, training the model longer for 270k iterations gives 0.3% improvement over 150k it-
Table 2. **Effect of pixel decoder designs on COCO val set.** Our final design of ConvNeXt-UNet pixel decoder (last row) stacks many ConvNeXt blocks at the level 5 (stride 32), effectively capturing long-range information and high-level semantics.

<table>
<thead>
<tr>
<th>Pixel Decoder</th>
<th>Design</th>
<th>FLOPs</th>
<th>Params</th>
<th>PQ</th>
</tr>
</thead>
<tbody>
<tr>
<td>FPN [48]</td>
<td>L₃-L₇</td>
<td>155B</td>
<td>31M</td>
<td>43.1</td>
</tr>
<tr>
<td>BiFPN [66]</td>
<td>6 BiFPN layers, L₃-L₇</td>
<td>210B</td>
<td>55M</td>
<td>47.7</td>
</tr>
<tr>
<td>ConvNeXt-UNet</td>
<td>N = [3, 9, 3], D = [768, 384, 192]</td>
<td>195B</td>
<td>54M</td>
<td>49.1</td>
</tr>
<tr>
<td>ConvNeXt-UNet</td>
<td>N = [11, 1, 1], D = [512, 384, 384]</td>
<td>173B</td>
<td>53M</td>
<td>50.1</td>
</tr>
<tr>
<td>ConvNeXt-UNet</td>
<td>N = [18, 1, 1], D = [384, 384, 384]</td>
<td>171B</td>
<td>51M</td>
<td>50.4</td>
</tr>
<tr>
<td>ConvNeXt-UNet</td>
<td>N = [18, 1, 1], D = [384, 384, 384] + SE [31]</td>
<td>171B</td>
<td>57M</td>
<td>50.8</td>
</tr>
</tbody>
</table>

Table 3. **Effect of using efficient heads on COCO val set.** In the prediction heads, using 7 × 7 depthwise convolution is more efficient than 3 × 3 convolution in both accuracy and FLOPs.

<table>
<thead>
<tr>
<th>Prediction Head Structure</th>
<th>FLOPs</th>
<th>Params</th>
<th>PQ</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 × 3 Convs</td>
<td>696B</td>
<td>68M</td>
<td>53.3</td>
</tr>
<tr>
<td>7 × 7 Depthwise Convs</td>
<td>171B</td>
<td>57M</td>
<td>53.6</td>
</tr>
</tbody>
</table>

Table 4. **Significance of class embeddings on COCO val set.** Using class embeddings (Cls-Emb) improves both things classes (PQ⁸₉) and stuff classes (PQmaf) for both ResNet50 and MobileNet-MH.

<table>
<thead>
<tr>
<th>Backbone</th>
<th>Cls-Emb</th>
<th>PQ</th>
<th>PQ⁸₉</th>
<th>PQmaf</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet-50</td>
<td>×</td>
<td>52.1</td>
<td>57.2</td>
<td>44.1</td>
</tr>
<tr>
<td>ResNet-50</td>
<td>✓</td>
<td>53.6</td>
<td>58.9</td>
<td>45.6</td>
</tr>
<tr>
<td>MobileNet-MH</td>
<td>×</td>
<td>34.9</td>
<td>37.5</td>
<td>31.0</td>
</tr>
<tr>
<td>MobileNet-MH</td>
<td>✓</td>
<td>37.2</td>
<td>39.8</td>
<td>33.1</td>
</tr>
</tbody>
</table>

Table 5. **Effect of mask centers on COCO panoptic val set.** Using mask centers improves the PQ compared to box centers.

<table>
<thead>
<tr>
<th>Centers</th>
<th>FLOPs</th>
<th>Params</th>
<th>PQ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Box Centers</td>
<td>171B</td>
<td>57M</td>
<td>53.0</td>
</tr>
<tr>
<td>Mask Centers</td>
<td>171B</td>
<td>57M</td>
<td>53.6</td>
</tr>
</tbody>
</table>

Table 6. **Effect of training techniques on COCO val set.** Our final setting employs longer training iterations (270k), strong scale augmentation ([0.05, 2.9]), and panoptic copy-paste.

<table>
<thead>
<tr>
<th>Longer Training</th>
<th>Strong Aug.</th>
<th>Copy-Paste</th>
<th>PQ</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>50.8</td>
</tr>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>51.1</td>
</tr>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>51.9</td>
</tr>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>53.6</td>
</tr>
</tbody>
</table>

Table 7. **Region-based vs. Center-based stuff representations using ResNet-50.** The table highlights the efficiency gains of the center-based approach in terms of latency (FPS), while showcasing a minimal compromise in PQmaf.

<table>
<thead>
<tr>
<th>Stuff</th>
<th>Backbone</th>
<th>PQmaf</th>
<th>FPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Center</td>
<td>Resnet-50</td>
<td>45.6</td>
<td>19.6</td>
</tr>
<tr>
<td>Region [46]</td>
<td>Resnet-50</td>
<td>45.8</td>
<td>15.4</td>
</tr>
</tbody>
</table>

5. Conclusion

In this work, we have presented MaskConver, revisiting pure convolution for panoptic segmentation. MaskConver simplifies the convolution-based panoptic models by unifying things and stuff modeling. Specifically, MaskConver uses centers to represent both thing and stuff regions, and employs the light class embedding module to predict unique embedding vectors for multiple instances that are present at the same locations. MaskConver also adopted the ConvNeXt-UNet pixel decoder that provides the prediction heads with long-range context and high-level semantics. With simplified architecture and the ConvNeXt-UNet, MaskConver closes the gap with the transformer-based models on COCO dataset. Finally, MaskConver excelled in the mobile domain, thanks to the simplicity and efficiency of convolutions.
References


[12] Bowen Cheng, Alex Schwing, and Alexander Kirillov. Per-pixel classification is not all you need for semantic segmentation. *NeurIPS*, 2021. 1, 2, 6


[20] Haoqi Fan, Bo Xiong, Karttikeya Mangalam, Yanghao Li, Zhicheng Yan, Jitendra Malik, and Christoph Feichtenhofer. Multiscale vision transformers. In *ICCV*, 2021. 2


