MASKCONVER: A UNIVERSAL PANOPTIC AND SEMANTIC SEGMENTATION MODEL WITH PURE CONVOLUTIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

Universal panoptic segmentation models have achieved state-of-the-art quality by using transformers for predicting masks. However, in mobile applications, transformer models are not computation-friendly due to the quadratic complexity with respect to the input length. In this work, we present MaskConver, a unified panoptic and semantic segmentation model with pure convolutions, which is optimized for mobile devices. We propose a novel lightweight mask embedding decoder to predict mask embeddings. These mask embeddings are used to predict a set of binary masks for both things and stuff classes. MaskConver achieves **37.2%** panoptic quality score on COCO validation set, which is **6.4%** better than Panoptic DeepLab with the same MobileNet backbone. After mobile-specific optimizations, MaskConver runs at **30** FPS and delivers 29.7% panoptic quality score on a Pixel 6, making it a real-time model, which is $10\times$ faster than Panoptic DeepLab with similar panoptic quality.

1 INTRODUCTION

Panoptic segmentation aims to unify instance and semantic segmentation in the same framework. Existing works propose to merge instance and semantic segmentation outputs using post-processing layers (Kirillov et al., 2019b;a; Xiong et al., 2019; Liu et al., 2019). These architectures however rely on many customized components like non-maximum suppression (NMS), and things-stuff merging heuristics to produce panoptic outputs. Recent works (Wang et al., 2021; Cheng et al., 2021; Yu et al., 2022; Cheng et al., 2022) unify both segmentation tasks by producing binary masks and class scores for both instance and semantic classes. Such universal architectures result in a simpler post-processing logic and makes the loss closely correlated to the panoptic quality (PQ) metric. They have achieved significantly higher PQ numbers compared to traditional architectures.

Among these unified panoptic segmentation models, transformers played a critical role due to their ability to learn instance-level embeddings via a transformer decoder. DETR (Carion et al., 2020) was first introduced for object detection by learning n object embeddings, each of which predicts an object class and a bounding box. The object's embeddings are learned through a transformer decoder that is trained using bipartite object matching to assign each object in the ground-truth to one of the n embeddings. Following DETR, MaX-DeepLab (Wang et al., 2021) is the first to use a transformer decoder to learn mask embeddings to predict a set of binary masks. The binary masks are then merged using a simple post-processing layer (Weber et al., 2021) to filter out duplicates, similar to non-max suppression. Other architectures (Cheng et al., 2021; Yu et al., 2022; Cheng et al., 2022) also follow a similar paradigm. The commonality between all these methods is they use transformer blocks to learn a set of binary masks and hence the panoptic mask.

Despite the ability to learn powerful feature representations, transformers are challenging to be deployed in mobile devices. First, they are memory and computation intensive (Han et al., 2022), and contain mobile-inefficient operations like reshape and transpose that requires expensive memory access. Second, it is harder to quantize transformers while maintaining the quality compared to convolutions, due to the complexity of attention and layer norm. (Liu et al., 2021). As a result, most mobile friendly vision transformers are hybrid architectures of convolutions and transformers (Mehta & Rastegari, 2022; Graham et al., 2021).

In this work, we propose a novel pure convolutional architecture for panoptic segmentation, named MaskConver. It produces segmentation masks for things and stuff classes in a unified way. Specifically, we first predict a center point heatmap (center head) following CenterNet (Zhou et al., 2019). We then feed the center point features (for both things and stuff) and the image features to a mask head. The mask head produces a segmentation mask for each input feature. Importantly, we highlight two key technical components: First, we propose to decompose stuff masks into connected regions (*i.e.*, islands) and use the region centers to represent the stuff class. Second, we observe it is important to integrate class information when decoding the mask for each center. We propose a class embedding module that augments the center features with class information. Lastly, we compare design choices for mobile use cases. We use a single output layer to avoid having two stages with the NMS layer in between. We also adopt the multi-hardware MobileNet (MobileNet-MH) (Chu et al., 2021) architecture as the backbone since it is more mobile friendly compared to MobileNetV3 (Howard et al., 2019).

We evaluate our MaskConver model on the COCO panoptic dataset, and compare our results to Panoptic DeepLab (Cheng et al., 2020) when using the same backbone. MaskConver achieves **37.2%** on COCO validation set, which is **6.4%** better than Panoptic DeepLab. We also show that MaskConver runs at **21**, and **30** FPS on Pixel 6 CPU and GPU respectively, while achieving 29.7% PQ on COCO validation set. On V100 GPU, this is **10.7**$\times$ faster than Panoptic DeepLab on model-only latency under similar performance, and to the best of our knowledge the first real-time panoptic segmentation model running on mobile devices.

2 Related Work

Since **panoptic segmentation** is proposed in Kirillov et al. (2019b), there has been a great amount of efforts in this area. They start from modifying existing networks, by adding a semantic branch (Kirillov et al., 2019a) or an instance branch (Cheng et al., 2020) to an existing state-of-the-art model for instance and semantic segmentation tasks. These methods quickly set up a baseline, however, hand-crafted post-processing layers are used to predict the final panoptic outputs. Following these works, researchers start to think about architectures that can solve the task in a more unified way. MaX-DeepLab (Wang et al., 2021) is the first to learn mask embeddings for both things and stuff classes. These mask embeddings predict a set of binary masks, and a post-processing layer (Weber et al., 2021) is used to predict the final panoptic outputs. MaskFormer (Cheng et al., 2021) proposes similar paradigm, and shows how to use the same model for both semantic and panoptic segmentation by only modifying the post-processing logic. Mask2Former (Cheng et al., 2022) proposes masked-attention to significantly outperform MaskFormer architecture on smaller objects by masking unrelated parts of the image. KMaX-DeepLab (Yu et al., 2022) improves MaX-DeepLab by using cross attention layers that mimics k-means algorithm, which leads to fewer parameters and higher PQ numbers. MaskConver also learns N mask embeddings for both things and stuff classes but only uses fully convolutional layers, and thus is more friendly for mobile devices.

Panoptic FCN (Li et al., 2021) is the first attempt to have fully convolutional architecture 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 N binary masks. Although Panopic FCN unifies things and stuff classes towards the post-processing, the kernel generator still treats things and stuff differently. 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. Unlike Panoptic FCN, MaskConver relies on a single output layer to learn centers, which is more suitable for mobile use cases.

Panoptic segmentation for mobile devices has not been well studied since most architectures focus more on pushing the panoptic quality instead of optimizing the architecture for mobile devices. Panoptic DeepLab (Cheng et al., 2020) reports quality and latency numbers on V100 GPU when using MobileNetV3 backbone on an image size of 640×640. Hou et al. (2020), and proposes a single-shot panoptic segmentation model that runs in real-time on V100 GPU. In this work, we tailor MaskConver architecture for mobile devices (Pixel 6) through a set of architecture design choices, and measure the latency of the architecture.

For on-device scenarios, various lightweight model backbones have been proposed, such as MobileNet (Howard et al., 2017; Sandler et al., 2018; Howard et al., 2019), EfficientNet (Tan & Le,

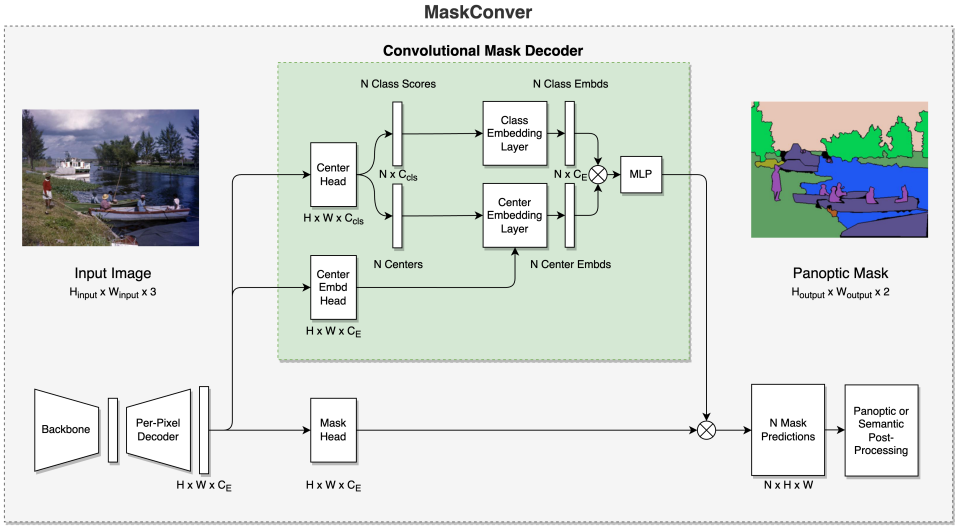

Figure 1: **Illustration of MaskConver architecture.** The input image goes through a backbone and a deconvolutional-based pixel-decoder, and produces a downsampled feature map (left). The feature map is then fed into a Center Head that produces detections for both things centers and stuff islands (Sec. 3.2) centers (middle top). We extract embedding of each center from a Center Embedding Head, and fuse this embedding feature with an external class embedding (Sec. 3.3) feature (middle center). The resulting features of each center are finally dot-producted with a dense Mask Head to produce the foreground segmentation mask of each center (middle bottom). Our framework is unified for thing and stuff classes, and only uses convolutional layers.

2019), and ShuffleNet (Zhang et al., 2018), etc. They are designed for low computational power devices, like mobile CPU and GPU. A multi-hardware MobileNet (MobileNet-MH) (Chu et al., 2021) is proposed as a discovered by neural architecture search and optimized for multiple hardware. It 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 (Wang & Howard, 2021). MaskConver uses MobileNet-MH since it delivers similar accuracy as MobileNetV3 (Howard et al., 2019), while being more compatible with different mobile devices.

Center point-based representation is first proposed in CenterNet (Zhou et al., 2019) for 2D object detection. Researchers have applied the center-point detection to various vision tasks, including tracking (Zhou et al., 2020), instance segmentation (Wang et al., 2020), and action recognition (Li et al., 2020), etc. Most of such extensions use a center point to model instances, while it is yet unclear how the center-point representation can help modeling stuff. To the best of our knowledge, our idea of decomposing stuff into non-overlapping islands is the first one to apply center point-based representations in panoptic segmentation.

3 MaskConver

The MaskConver framework is illustrated in Fig. 1. It consist of a backbone, per-pixel decoder, a mask decoder, and a mask head. While the architecture is similar to MaskFormer (Cheng et al., 2021), we use a fully convolutional mask decoder that proposes object centers, instead of using a transformer-based mask decoder that uses learned queries (Cheng et al., 2021). Such difference is one of the key factors making our MaskConver more mobile friendly. We introduce our convolutional mask decoder in details below.

3.1 Convolutional Mask Decoder

The convolutional mask decoder aims at producing N mask embeddings for an instance, and representing the center and class of the instance, where the instance is either thing or stuff. The input

(a) Sample image. (b) Tree mask.

(c) Global stuff center. (d) Island centers.

Figure 2: **Illustration of different center representation for stuff class.** We show (a) the original RGB image; (b) the mask for stuff class "tree"; (c) a naive center-based representation that takes the global center-of-mass point; (d) our novel island center representation. Our island center representation aligns better with the actual stuff locations.

to the convolutional mask decoder is the output from the per-pixel decoder, and the outputs are N mask embeddings and corresponding N classification scores.

Our convolutional decoder models both thing classes and stuff classes as their center points by a center head, following CenterNet. Unlike CenterNet that uses bounding box centers to represent objects, we resort to *masks* to construct the things centers. The center head uses DeepLabV3+ (Chen et al., 2018) decoder style to upsample the feature maps from the last stage of the backbone, and then fuses it with lower level feature maps through either concatenation or summation. For things, we construct the bounding boxes from the panoptic instance mask. For each instance mask, we generate an axis-aligned bounding box tightly enclosing the object. The center of the bounding box represents the center of each instance. For stuff, we have explored different ways to model its centers and we will discuss that in Section 3.2. We use the standard heatmap focal loss (Lin et al., 2017; Zhou et al., 2019; Law & Deng, 2018) to train the center heatmap for both thing and stuff classes.

Besides the center heatmap head, we create a center embedding head from the per-pixel decoder. The center embedding head learns to generate an $H \times W \times C_E$ feature map, where at each spatial location there is a C_E-dimension embedding. We also use DeepLabV3+ decoder to upsample the feature maps. We extract embeddings from the embedding head at corresponding center peak locations, *i.e.*, the center embedding for a center i, j is obtained by retrieving the vector at location i, j from the center embedding head. The convolutional decoder obtains class embedding for each center as explained in Section 3.3. The center and class embeddings are then fused with an element-wise multiplication. Lastly, a multi-layer perceptron (MLP) is used to predict the final mask embedding for each center.

Following MaskFormer, we use the mask embeddings and output of the mask head to predict a binary mask for each mask embedding. Similar to the center embedding head, the mask head generates an $H \times W \times C_E$ feature map and uses DeepLabV3+ decoder to upsample it. The binary masks are obtained by dot-product between center embeddings and mask head outputs, followed by sigmoid activation.

3.2 Stuff Centers

Modeling stuff centers is very challenging since it requires defining the stuff class as an instance or instances with one or multiple centers. We consider three different techniques to define stuff centers.

1. Fix the stuff centers to be the center of the image. This does not work well since all stuff centers collide in the center of the image, resulting in poor discriminative ability of stuff embeddings.
2. Treat the stuff mask as a single instance, and compute the center of the stuff class from the enclosing bounding box, similar to our solution to thing classes. This option works fairly well, however, we find some inconsistencies between train centers and predicted centers, when the stuff class is split into multiple regions in the image.
3. Split stuff into islands and compute the island centers from the enclosing bounding box for each island. Before computing the stuff islands, we merge small islands by using max

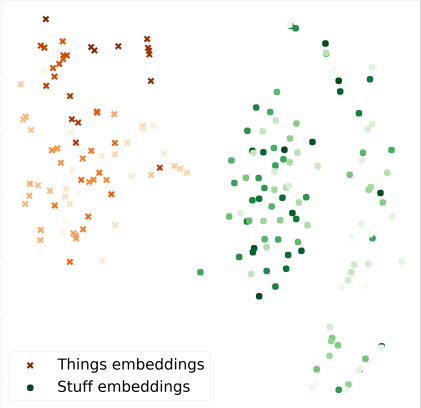

Figure 3: **tSNE (Van der Maaten & Hinton, 2008) plot of learned class embeddings split by things and stuff.** Orange crosses are thing classes and green dots are stuff classes. Color transparencies encode class IDs in the dataset which are grouped by super-classes (Lin et al., 2014). Our learned class embedding separates things and stuff classes automatically.

pooling on the stuff mask. We compute the islands by finding connected components, and select the largest k islands, where k is a hyper-parameter. In our experiments, we use $k = 3$.

The second technique works better to some extent. However, there is inconsistency between groundtruth labels, and what the model is trying to predict. Fig. 2a shows an example image, where the tree class is split into disjoint regions. Even though it is trained with the second technique (Fig. 2c), the model still predicts two centers. With the third technique, the groundtruth labels (Fig. 2d) are better aligned with the model predictions. We will show in the experiments that using stuff islands provides decent performance gains especially for stuff PQ metric.

3.3 Class Embeddings

The problem with center embeddings is that instance centers may collide and leads to exactly the same embedding vector being pooled from the center embedding head. To fix this issue, we propose to use class embeddings, which is a lookup embedding layer of $N \times C_E$ embeddings. For each class c, we learn a C_E-dimensional vector. To learn a unique embedding vector per location per class, we multiply the class embedding and center embedding using element-wise multiplication. We find that class embedding is crucial to the quality of the predicted masks.

Fig. 3 shows the learned class embeddings for things and stuff. There are two well-separated clusters: one for things and another for stuff. Our interpretation is that many collisions happen between things and stuff classes, as a result the center embeddings will be projected to different spaces conditioned on the type of the center (*i.e.*, thing or stuff). We will also show in experiments that class embedding layer provides a decent performance improvements.

3.4 Optimizations for Mobile

To tailor MaskConver for mobile use cases, we simplify the architecture by having only a single feature map scale. It is crucial the backbone can localize objects accurately using a single output scale. Hence, segmentation backbones work better for MaskConver than detection or classification backbones. The backbone and the per-pixel decoder aim at accurately predicting object centers, and learning localized mask embeddings.

We use the multi-hardware MobileNet (MobileNet-MH) Chu et al. (2021) as the backbone since it delivers similar accuracy as MobileNetV3 Large, while being more mobile friendly. We adopt the Atrous Spatial Pyramid Pooling (ASPP) decoder (Chen et al., 2017) and separable convolutions in the decoder. For all the three heads (center, center embedding, and mask heads), we use DeepLabv3+ decoder with separable convolutions for feature fusion to merge outputs from the backbone and decoder, followed by two convolutional blocks. Each block contains a 3×3 separable convolution and a 1×1 convolution. We set the hidden size of both decoder and heads to 256.

For efficiency, we avoid any reshape or transpose operations, and replace sigmoid by hard-sigmoid 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 $2\times$.

3.5 Training and Evaluation

During training, the center head is trained using targets computed from target assignments, and the center embeddings are pooled using the groundtruth centers. We have tried pooling the best N centers from center head predictions, but it does not improve the accuracy. Then, we retrieve the class embeddings using the class assignments of the groundtruth centers to produce N binary masks. Binary cross entropy loss and dice loss are used to optimize the binary masks. We combine the losses by weighted sum: $Loss_{total} = \lambda_{centers} Loss_{centers} + \lambda_{BCE} Loss_{BCE} + \lambda_{dice} Loss_{dice}$. Across all experiments, we fix the weighting factors, which are $\lambda_{centers} = 1$, $\lambda_{BCE} = 5$, and $\lambda_{dice} = 5$.

Inference is done differently since the groundtruth centers and classes are not available. To predict N centers, we use a simple NMS layer by applying a max pooling layer to the center head output to predict N local maxima and filter out adjacent pixels. The resulting locations and corresponding class labels from these N local maxima are subsequently used to obtain the center and class embeddings, as well as to produce N binary masks. We then follow the post-processing logic of MaskFormer to finally generate panoptic and semantic segmentation outputs. Empirically, we set the pool size to 3 for the max pooling layer.

4 Experiments

4.1 Mobile Models

4.1.1 Panoptic Segmentation

Training Setup We use SGD optimizer with momentum of 0.9, and a cosine learning rate schedule with initial learning rate of 0.04 and a warm-up step of 2000. The model is trained for 270 epochs with batch size of 64. We apply a weight decay of $1e^{-5}$ to all of the convolutional layers. We use scale augmentation of $[0.1, 1.9]$, random horizontal flipping and AutoAugment (Cubuk et al., 2019) for data augmentation. All experiments start from a model checkpoint pretrained on ImageNet and the COCO segmentation task.

Comparison to State-of-the-Arts We compare MaskConver to Panoptic DeepLab using MobileNetV3 Large and MobileNet-MH backbones. As shown in Table 1, MaskConver achieves **7.02%** improvement over the original Panoptic DeepLab using MobileNetV3 Large, and **6.4%** improvement over the variant using MobileNet-MH. At the same time, MaskConver reduces latency by 38.9% with image size of 640×640 and by a notable **91.6%** with image size of 256×256 on Pixel 6 CPU. This makes MaskConver a real-time model on mobile devices and $10\times$ faster than Panoptic DeepLab, while achieving comparable quality. Note that post-processing (Weber et al., 2021) of the Panoptic DeepLab model is not supported on mobile devices so it is excluded from latency measurement for fair comparison.

Inference Speed vs. Model Quality We mainly study two important factors regarding the trade-off between quality and latency of MaskConver: input size of the model, and the number of proposals. In particular, we study the effect on latency and quality of the model by reducing input image size, and varying number of proposals in post-processing. Since the number of proposals is a key factor affecting post-processing due to argmax and resizing operations, such study helps us evaluate the cost of post-processing.

Table 2 shows the effect of reducing input image size on the inference speed (latency) and model quality. All numbers include post-processing. First, we notice 7.5% PQ drop and 31% latency reduction on V100 GPU, when decreasing the input image size from 640×640 to 256×256 but upsampling it to a fixed size of 640×640 during post-processing. When such upsampling is removed to produce the final output of the same size as input, we observe 58% latency reduction. On mobile devices, we achieve more significant latency reduction, which is 87% on Pixel 6 CPU, and 81% on

Table 1: **Comparison of MaskConver to existing mobile models on COCO panoptic segmentation validation set.** First row: The original Panoptic DeepLab. Numbers quoted from the paper (Cheng et al., 2020); Second row: our retrained Panoptic DeepLab with the same backbone; Third row: MaskConver under the same setting with Panoptic DeepLab; Forth row: MaskConver with a smaller input size. Latency is measured on Pixel 6 CPU without post-processing. MaskConver outperforms Panoptic DeepLab by 6.4% PQ under the same setting while running faster.

Architecture	Backbone	Image Size	Params	FLOPs	PQ	Latency
Panoptic DeepLab	MobileNetV3L	641	-	12.24B	30.0	359.6 ms
Panoptic DeepLab	MobileNet-MH	641	3.89M	13.93B	30.8	400.1 ms
MaskConver (ours)	MobileNet-MH	640	3.99M	9.53B	**37.2**	244.5 ms
MaskConver (ours)	MobileNet-MH	256	3.99M	1.54B	29.7	33.49 ms

Table 2: **Inference speed (latency) and model quality under different input sizes.** We report PQ on different devices (V100 GPU, Pixel 6 CPU, and Pixel 6 GPU) on COCO validation set with MobileNet-MH backbone. The runtime includes all post-processing. **IS** means post-processing is set to output images of the same size as the input. With input size 256×256, MaskConver runs in real-time on Pixel 6 GPU with 29.7 PQ.

Backbone	Input	Params	FLOPs	PQ	Latency/Output Size			
					V100/ 640	V100/ IS	P6 CPU/IS	P6 GPU/IS
MobileNet-MH	256	3.99M	1.54B	29.7	17.12 ms	10.48 ms	**46.9 ms**	**33.2 ms**
MobileNet-MH	384	3.99M	3.37B	33.8	18.14 ms	14.62 ms	117.1 ms	66.8 ms
MobileNet-MH	640	3.99M	9.58B	37.2	24.93 ms	24.93 ms	347.9 ms	172.1 ms

Pixel 6 GPU, when we keep the same size for input and output. By using a 256×256 input image size, we are able to run MaskConver nearly in real-time on Pixel 6 CPU, and real-time on Pixel 6 GPU. This is more than $10\times$ faster than Panoptic DeepLab with comparable quality. Moreover, we expect an even lower latency by more aggressive quantization for the cases where computational power is highly constrained.

We have extensively studied MaskConver with mobile backbones on COCO panoptic dataset, and our results show that MaskConver has a significant improvement over existing mobile baselines and achieves real-time speed on mobile devices. To the best of our knowledge, this is the first time a panoptic segmentation model is optimized to run real-time on mobile devices with competitive panoptic quality.

In Table 3, we show the effect on model latency and quality as we vary the number of output proposals. Reducing the number of proposals effectively reduces model latency while preserving reasonable panoptic quality. For example, we can reduce the number of proposals from 100 to 15 to achieve nearly $2\times$ speed-up, while only introducing 0.7% PQ drop. This study verifies that we can optimize the number of proposals to gain significant latency improvements without sacrificing the quality.

4.1.2 Semantic Segmentation

We also evaluate MaskConver for semantic segmentation task on the Cityscapes and Pascal VOC datasets to verify modeling stuff centers indeed works well for semantic classes.

Training Setup We use the SGD optimizer with 0.9 momentum and the same model checkpoint used for panoptic segmentation. We adjust the weight decay to $1e{-}4$ to all convolutional layers, and apply the same set of data augmentations. For scale augmentation, we use $[0.1, 1.9]$ on the Cityscapes dataset and a narrower range $[0.5, 2.0]$ on the Pascal VOC dataset. On Cityscapes dataset, we use cosine learning rate schedule with initial learning rate of 0.1 and warm-up step of 925, and train for 2150 epochs with a batch size of 64. On Pascal VOC dataset, we use a polynomial learning rate schedule with initial learning rate of 0.007 and warm-up step of 660, and train for 100 epochs with a batch size of 32.

Table 3: **Inference speed (latency) versus model quality trade-offs under different number of proposals.** We report PQ with MobileNet-MH backbone on COCO validation set. MaskConver retains high PQ with as few as 20 proposals.

Proposals	FLOPs	PQ	Latency/Output Size			
			V100/640	V100/256	P6 CPU/256	P6 GPU/256
100	1.57B	29.9	22.37 ms	11.53 ms	66.8 ms	46.9 ms
50	1.55B	29.7	17.12 ms	10.48 ms	46.9 ms	33.2 ms
20	1.54B	29.6	12.78 ms	9.41 ms	40.7 ms	31.2 ms
15	1.53B	29.2	11.73 ms	8.9 ms	39.9 ms	29.3 ms
10	1.53B	27.1	10.96 ms	8.61 ms	36.6 ms	27.9 ms

Table 4: **Comparison of MaskConver to DeepLabV3 models on the CityScapes and Pascal VOC validation sets**. We report semantic segmentation performance in mean IOU. Under the same setting, MaskConver outperform DeepLabV3+ by 2.7% points on the Cityscapes dataset and 1.1% points on the VOC dataset.

Architecture	Backbone	Image Size	Params	FLOPs	mean IOU
CityScapes					
DeepLabV3+	MobileNet-MH	1024x2048	2.2M	23B	72.77
MaskConver	MobileNet-MH	1024x2048	2.3M	25B	**75.54**
Pascal VOC					
DeepLabV3	MobileNetV2	513	4.52M	5.69B	75.7
DeepLabV3+	MobileNet-MH	513	2.84M	3.64B	75.29
MaskConver	MobileNet-MH	512	3.48M	4.28B	**76.4**

Comparison to State-of-the-Arts We compare MaskConver to DeepLabV3 and DeepLabV3+ architectures. All models are using the same mobile backbone. Although we are using the same architecture as DeepLabV3, MaksConver is clearly better than DeepLabV3+ as shown in Table 4. Specifically, MaskConver improves over DeepLabV3+ by 2.7% on the Cityscapes dataset and 1.1% on the Pascal VOC dataset in terms of mean IoU, and only slightly increases number of parameters and FLOPs. These results show that our center-based segmentation masks are more effective compared to standard per-class segmentation models.

4.2 Large Models

4.2.1 Panoptic Segmentation

Comparison to State-of-the-Arts We train large models using the same training setup as that for mobile models and compare MaskConver to existing SoTA models of similar scale. Similar to MaskFormer (Cheng et al., 2021), we only consider models that use single scale feature maps to build mask embeddings. We use the SpineNet-Seg backbone (Rashwan et al., 2021) and apply 2 architecture changes following Bello et al. (2021). First, we use 3 3×3 convolutional layers instead of one 7×7 convolutional layer (He et al., 2019). Second, we add Squeeze-and-Excitation (Hu et al., 2018) layer to the bottleneck block in the backbone. These changes, as shown in Table 5, have significant increase in PQ numbers with minor increase in model latency. We refer to the improved backbone by SpineNet-Seg+. Different from Panoptic FCN, we do not use multi-scale feature maps to build mask embeddings since it would greatly slow down the model on mobile devices.

As shown in Table 5, MaskConver outperforms all convolution based models in terms of both PQ and inference speed (latency). In particular, MaskConver obtains **2.48%** better PQ compared to the improved version of Panoptic FCN (Li et al., 2021), while also being clearly faster. MaskConver also reaches on-par quality compared to transformer based models. For example, MaskConver with SpineNet-Seg49+ backbone achieves 47.0% PQ on COCO validation set using 199G FLOPs, while MaskFormer with ResNet50 backbone achieves 46.5% PQ on the same dataset using 181G Flops.

Table 5: **Comparison of MaskConver model to existing regular models on COCO panoptic validation set.** Latency numbers are computed on V100 GPU. MaskConver performs on-par with transformer models under similar FLOPs, and outperforms all other fully-convolutional models.

Architecture	Backbone	Image Size	Params	FLOPs	PQ	Latency
Transformer Based						
DETR	ResNet50	800x1333	-	-	43.4	-
MaskFormer	ResNet50	800x1333	45M	181B	46.5	56.8 ms
MaX-DeepLab	MaX-S	-	62M	324B	48.4	-
Convolutional Based						
Panoptic DeepLab	Xception	641	-	109B	38.9	-
Panoptic FCN	ResNet50	800x1333	-	-	43.6	80.0 ms
Panoptic FCN*	ResNet50	800x1333	-	-	44.3	108.7 ms
MaskConver (ours)	SpineNet-Seg49	640	75M	197B	45.9	73.0 ms
MaskConver (ours)	SpineNet-Seg49+	640	97M	199B	**47.0**	78.1 ms

From these experiments, we have demonstrated that MaskConver achieves on-par quality with transformer-based models at similar FLOPs, and outperforms all existing convolutional based models in both PQ and inference speed.

4.3 Ablation Study

Stuff Islands We first study the effect of stuff centers on the panoptic quality using SpineNet-Seg49 backbone (Rashwan et al., 2021). In Table 6, we show the effect of using stuff islands versus using the center of the whole stuff mask. Using stuff islands improves the stuff PQ by 1.53%, and overall PQ by 0.43% on COCO panoptic validation set.

Table 6: **Effect of using stuff centers on COCO panoptic validation dataset.** Global means the stuff mask is treated as a single instance, and the center for the stuff class is computed from a single bounding box. Island center means that the stuff class is split into islands using connected component analysis, and the stuff centers are computed using each island. Our island center representation outperforms global centers especially on stuff classes, with a 1.53% PQ gain.

Stuff Centers	PQ all	PQ Things	PQ Stuff
Global Centers	45.11	49.41	38.61
Island Centers	**45.54**	49.11	**40.14**

Class Embeddings We also study the effect of using class embedding layer on the panoptic quality. Using SpineNet-Seg49 backbone, we show that using class embedding layer improves the PQ significantly on COCO panoptic validation set by 1.81% (see Table 7).

Table 7: **Significance of class embedding layer on COCO PQ metric.** It improves both things classes and stuff classes, with an overall 1.81% PQ improvement.

Class Embedding	PQ all	PQ Things	PQ Stuff
No	43.73	47.06	38.70
YES	**45.54**	49.11	40.14

5 Conclusions

In this work, we have presented MaskConver, a universal panoptic segmentation model with pure convolutions. MaskConver relies on things and stuff centers to encode their mask embeddings, splits stuff regions into islands, and uses the center of each island to encode the stuff class embeddings. MaskConver also uses class embeddings to allow for unique mask embeddings even with colliding centers. Extensive experiments have demonstrated that MaskConver achieves outstanding panoptic segmentation quality compared to existing panoptic segmentation modls. In particular, by using pure convolutions, MaskConver achieves real-time inference speed on mobile devices.

REFERENCES

Irwan Bello, William Fedus, Xianzhi Du, Ekin Dogus Cubuk, Aravind Srinivas, Tsung-Yi Lin, Jonathon Shlens, and Barret Zoph. Revisiting resnets: Improved training and scaling strategies. *NeurIPS*, 2021.

Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *ECCV*, 2020.

Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *TPAMI*, 2017.

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

Bowen Cheng, Maxwell D Collins, Yukun Zhu, Ting Liu, Thomas S Huang, Hartwig Adam, and Liang-Chieh Chen. Panoptic-deeplab: A simple, strong, and fast baseline for bottom-up panoptic segmentation. In *CVPR*, 2020.

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

Bowen Cheng, Ishan Misra, Alexander G Schwing, Alexander Kirillov, and Rohit Girdhar. Masked-attention mask transformer for universal image segmentation. In *CVPR*, 2022.

Grace Chu, Okan Arikan, Gabriel Bender, Weijun Wang, Achille Brighton, Pieter-Jan Kindermans, Hanxiao Liu, Berkin Akin, Suyog Gupta, and Andrew Howard. Discovering multi-hardware mobile models via architecture search. In *CVPR*, 2021.

Ekin D Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V Le. Autoaugment: Learning augmentation policies from data. *CVPR*, 2019.

Benjamin Graham, Alaaeldin El-Nouby, Hugo Touvron, Pierre Stock, Armand Joulin, Hervé Jégou, and Matthijs Douze. Levit: a vision transformer in convnet's clothing for faster inference. In *ICCV*, 2021.

Kai Han, Yunhe Wang, Hanting Chen, Xinghao Chen, Jianyuan Guo, Zhenhua Liu, Yehui Tang, An Xiao, Chunjing Xu, Yixing Xu, et al. A survey on vision transformer. *TPAMI*, 2022.

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

Rui Hou, Jie Li, Arjun Bhargava, Allan Raventos, Vitor Guizilini, Chao Fang, Jerome Lynch, and Adrien Gaidon. Real-time panoptic segmentation from dense detections. In *CVPR*, 2020.

Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. In *ICCV*, 2019.

Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *CVPR*, 2018.

Alexander Kirillov, Ross Girshick, Kaiming He, and Piotr Dollár. Panoptic feature pyramid networks. In *CVPR*, 2019a.

Alexander Kirillov, Kaiming He, Ross Girshick, Carsten Rother, and Piotr Dollár. Panoptic segmentation. In *CVPR*, 2019b.

Hei Law and Jia Deng. Cornernet: Detecting objects as paired keypoints. In *ECCV*, 2018.

Yanwei Li, Hengshuang Zhao, Xiaojuan Qi, Liwei Wang, Zeming Li, Jian Sun, and Jiaya Jia. Fully convolutional networks for panoptic segmentation. In *CVPR*, 2021.

Yixuan Li, Zixu Wang, Limin Wang, and Gangshan Wu. Actions as moving points. In *ECCV*, 2020.

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

Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *ICCV*, 2017.

Huanyu Liu, Chao Peng, Changqian Yu, Jingbo Wang, Xu Liu, Gang Yu, and Wei Jiang. An end-to-end network for panoptic segmentation. In *CVPR*, 2019.

Zhenhua Liu, Yunhe Wang, Kai Han, Wei Zhang, Siwei Ma, and Wen Gao. Post-training quantization for vision transformer. *NeurIPS*, 2021.

Sachin Mehta and Mohammad Rastegari. Mobilevit: light-weight, general-purpose, and mobile-friendly vision transformer. *ICLR*, 2022.

Abdullah Rashwan, Xianzhi Du, Xiaoqi Yin, and Jing Li. Dilated spinenet for semantic segmentation. *arXiv preprint arXiv:2103.12270*, 2021.

Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *CVPR*, 2018.

Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *ICML*. PMLR, 2019.

Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 2008.

Huiyu Wang, Yukun Zhu, Hartwig Adam, Alan Yuille, and Liang-Chieh Chen. Max-deeplab: End-to-end panoptic segmentation with mask transformers. In *CVPR*, 2021.

Weijun Wang and Andrew Howard. Mosaic: Mobile segmentation via decoding aggregated information and encoded context. *arXiv preprint arXiv:2112.11623*, 2021.

Yuqing Wang, Zhaoliang Xu, Hao Shen, Baoshan Cheng, and Lirong Yang. Centermask: single shot instance segmentation with point representation. In *CVPR*, 2020.

Mark Weber, Huiyu Wang, Siyuan Qiao, Jun Xie, Maxwell D. Collins, Yukun Zhu, Liangzhe Yuan, Dahun Kim, Qihang Yu, Daniel Cremers, Laura Leal-Taixe, Alan L. Yuille, Florian Schroff, Hartwig Adam, and Liang-Chieh Chen. DeepLab2: A TensorFlow Library for Deep Labeling. *arXiv: 2106.09748*, 2021.

Yuwen Xiong, Renjie Liao, Hengshuang Zhao, Rui Hu, Min Bai, Ersin Yumer, and Raquel Urtasun. Upsnet: A unified panoptic segmentation network. In *CVPR*, 2019.

Qihang Yu, Huiyu Wang, Siyuan Qiao, Maxwell Collins, Yukun Zhu, Hartwig Adam, Alan Yuille, and Liang-Chieh Chen. k-means mask transformer. In *ECCV*, 2022.

Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *CVPR*, 2018.

Xingyi Zhou, Dequan Wang, and Philipp Krähenbühl. Objects as points. *arXiv preprint arXiv:1904.07850*, 2019.

Xingyi Zhou, Vladlen Koltun, and Philipp Krähenbühl. Tracking objects as points. In *ECCV*, 2020.