THU-AM-128

You Only Segment Once: Towards Real-Time Panoptic Segmentation

Jie Hu, Linyan Huang, Tianhe Ren, Shengchuan Zhang, Rongrong Ji, and Liujuan Cao

Key Laboratory of Multimedia Trusted Perception and Efficient Computing, Ministry of Education of China, Xiamen University

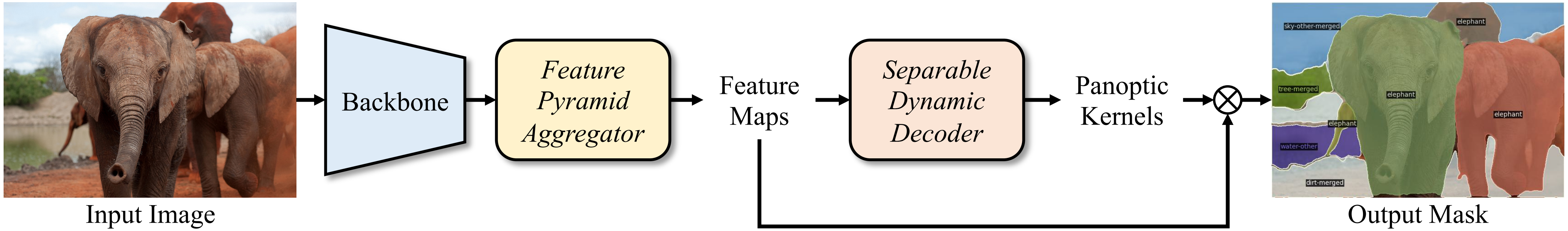
JUNE 18-22, 2023

CVPR

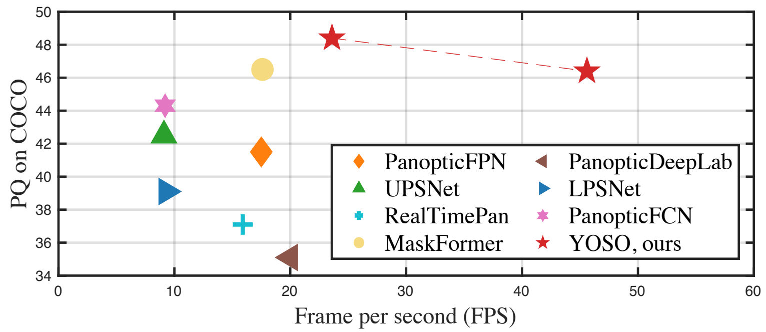
VANCOUVER, CANADA

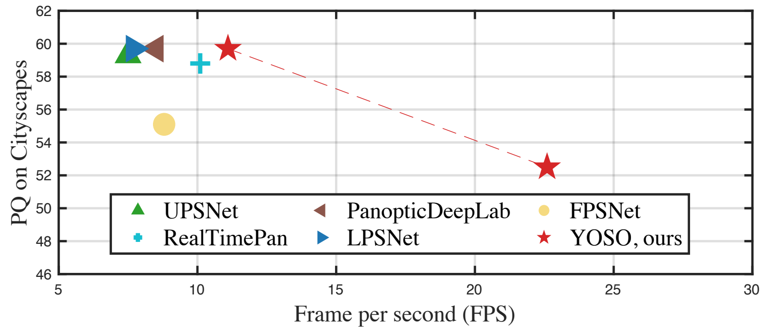
Quick Preview

- A simple, real-time framework *(YOSO)* for *panoptic segmentation*



- The proposed *feature pyramid aggregator* and *separable dynamic decoder* speed up the pipeline and obtain good accuracy





Panoptic Segmentation

- Assign each pixel with a semantic label and a unique identity
- The semantic labels are summarized into two types
  - ***stuff*** - amorphous and uncountable concepts (such as sky and road)
  - ***things*** - countable categories (such as persons and cars)

(a) image (b) semantic segmentation (c) instance segmentation (d) panoptic segmentation

Kirillov A, He K, Girshick R, Rother C, Dollár P. Panoptic segmentation. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition 2019 (pp. 9404-9413).

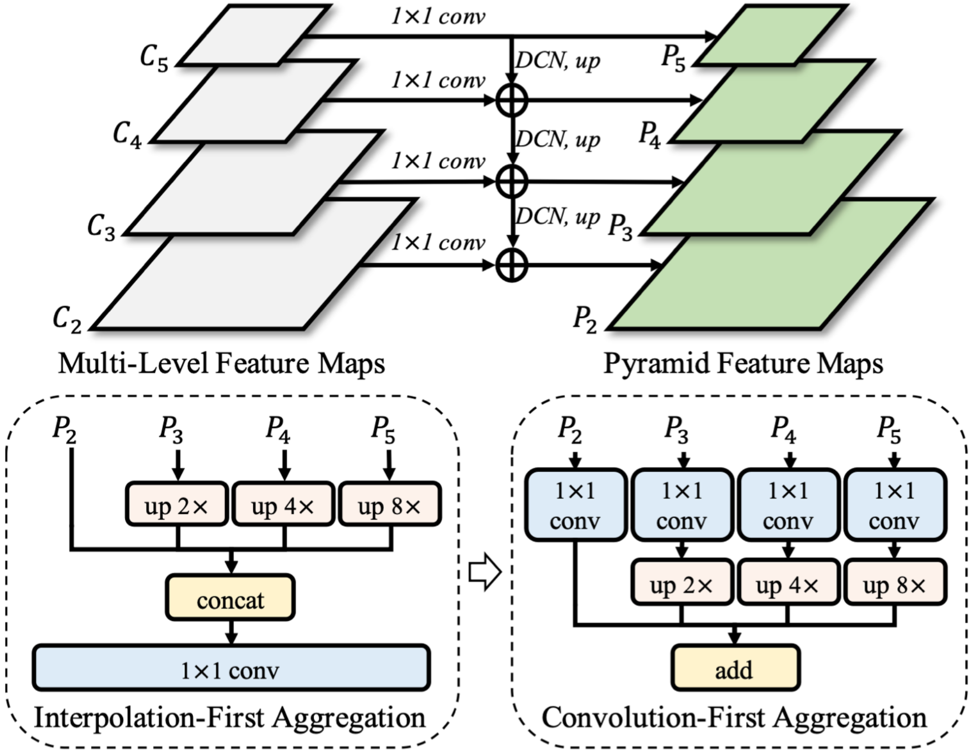
YOSO

- Unify the two types of classes for *stuff* and *things*
  - You only need to segment once for semantic and instance masks

- Task formulation:
  - Predict n binary masks and corresponding class probabilities
  - Masks with the same background class *(stuff)* are merged via union operation
  - Masks with foreground classes *(things)* are treated as independent instances

Feature Pyramid Aggregator

- Key idea: switch the order of interpolation and convolution



Observation I: *The output of IFA is exactly equal to that of CFA when using 1×1 convolution without bias.*

$$f\Big(\sum_i w^i \boldsymbol{v}^i_{x,y}\Big) = \frac{1}{(x_2-x_1)(y_2-y_1)} \begin{bmatrix} x_2 - x_0 \\ x_0 - x_1 \end{bmatrix}^\top \begin{bmatrix} \sum_i w^i v^i_{x_1,y_1}, & \sum_i w^i v^i_{x_1,y_2} \\ \sum_i w^i v^i_{x_2,y_1}, & \sum_i w^i v^i_{x_2,y_2} \end{bmatrix} \begin{bmatrix} y_2 - y_0 \\ y_0 - y_1 \end{bmatrix} = \sum_i w^i f(\boldsymbol{v}^i_{x,y})$$

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

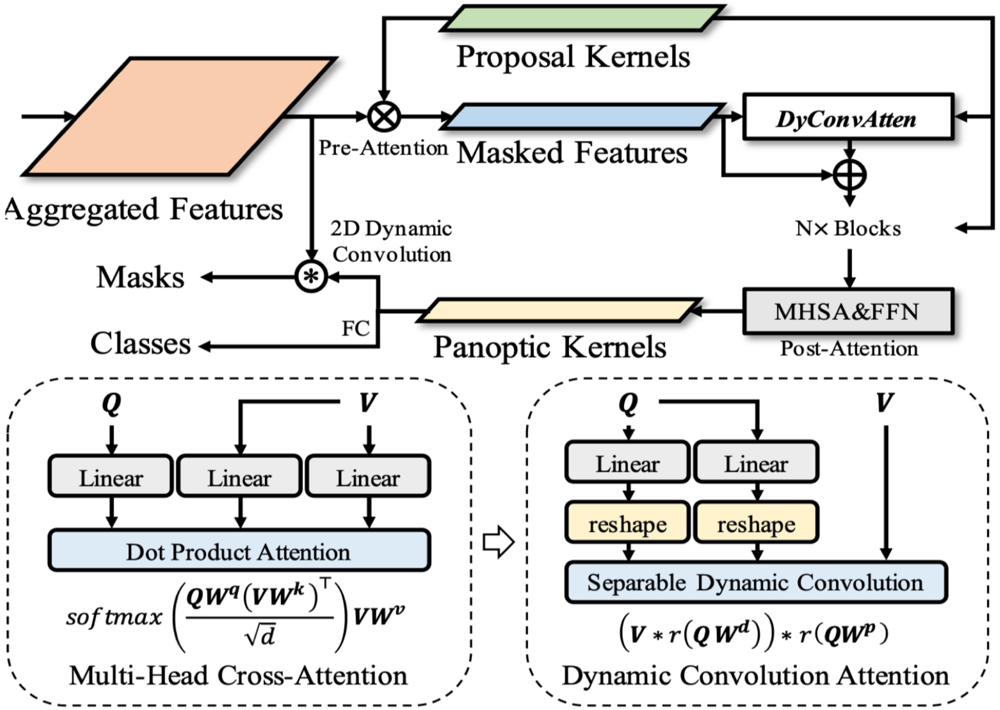
Aggregator	PQ	SQ	RQ	PQ^t	PQ^s	FLOPs	Latency (μs)↓	FPS↑
IFA	47.5	82.2	56.9	52.7	39.7	16.6G	4871±11	23.3
CFA	47.0	81.4	56.2	52.3	39.0	2.1G	1877±52	29.2

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

- Accelerate the pipeline *with no cost*

Separable Dynamic Decoder

- Key idea: replace matrix multiplication by 1D convolution



Attention	PQ	SQ	RQ	PQ^t	PQ^s	FLOPs	Latency (μs)↓	FPS↑
MHCA	46.0	81.9	55.1	51.5	37.7	31.5M	2608±210	27.3
SDCA	47.0	81.4	56.2	52.3	39.0	5.4M	2183±279	29.2
DCA	46.9	82.0	55.8	51.9	38.7	15.5M	1701±186	30.0
PDCA	43.7	81.3	52.3	49.3	35.3	5.2M	1450±183	30.2
DDCA	46.6	82.3	55.9	52.3	38.4	0.3M	1242±101	30.3

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

- Accelerate the pipeline and improve the performance

Main Results

Method	Backbone	Scale	PQ	PQ^t	PQ^s	FPS↑	GPU
BGRNet [52]	R50-FPN	800,1333	43.2	49.8	33.4	-	-
K-Net [58]	R50-FPN	800,1333	47.1	51.7	40.3	-	-
PanSegFormer [32]	R50	800,1333	49.6	54.4	42.4	-	-
Max-DeepLab [47]	Max-S	800,1333	48.4	53.0	41.5	7.6	V100
Mask2Former [10]	R50	800,1333	51.9	57.7	43.0	8.6	V100
UPSNet [54]	R50-FPN	800,1333	42.5	48.5	33.4	9.1	V100
PanopticFCN [31]	R50-FPN	800,1333	44.3	50.0	35.6	9.2	V100
LPSNet [23]	R50-FPN	800,1333	39.1	43.9	30.1	9.3	V100
RealTimePan [24]	R50-FPN	800,1333	37.1	41.0	30.7	15.9	V100
PanopticFPN [27]	R50-FPN	800,1333	41.5	48.3	31.2	17.5	V100
MaskFormer [11]	R50	800,1333	46.5	51.0	39.8	17.6	V100
PanopticDeepLab [9]	R50	641,641	35.1	-	-	20.0	V100
YOSO, *ours*	R50	800,1333	48.4	53.5	40.8	23.6	V100
YOSO, *ours*	R50	512,800	46.4	50.7	40.0	45.6	V100

Table 1. Panoptic segmentation on the **COCO** *validation* set.

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

Table 2. Panoptic segmentation on the **Cityscapes** *validation* set.

Method	Backbone	Scale	PQ	PQ^t	PQ^s	FPS↑	GPU
BGRNet [52]	R50-FPN	-	31.8	34.1	27.3	-	-
PanSegFormer [32]	R50	-	36.4	35.3	38.6	-	-
MaskFormer [11]	R50	640,2560	34.7	32.2	39.7	-	-
Mask2Former [10]	R50	640,2560	39.7	39.0	40.9	11.1	V100
YOSO, *ours*	R50	640,2560	38.0	37.3	39.4	35.4	V100

Table 3. Panoptic segmentation on the **ADE20K** *validation* set.

Method	Backbone	Scale	PQ	PQ^t	PQ^s	FPS↑	GPU
AdaptIS [44]	R50	-	32.0	39.1	26.6	-	-
Seamless [43]	R50	-	36.2	33.6	40.0	-	-
LPSNet [23]	R50-FPN	-	36.5	33.2	41.0	-	-
PanopticFCN [31]	R50-FPN	-	36.9	32.9	42.3	-	-
PanopticDeepLab [9]	R50	2176,2176	33.3	-	-	3.5	V100
Mask2Former [10]	R50	2048,2048	36.3	-	-	3.2	A100
YOSO, *ours*	R50	2048,2048	34.1	24.3	47.2	7.1	A100

Table 4. Panoptic segmentation on the **Mapillary** *validation* set.

Main Results