Removing Rows and Columns of Tokens in Vision Transformer enables Faster Dense Prediction without Retraining Supplementary Document

Diwei Su[1], Cheng Fei[1], and Jianxu Luo[1](✉)

East China University of Science and Technology, Shanghai 200237, China
dvsu@mail.ecust.edu.cn, cfei_jarvis@163.com, jxluo@ecust.edu.cn

1 Implementation Details

1.1 Hyper-parameter Details

We summarize the detailed hyper-parameter configurations and results of our method applied in the Segmenter and MaskDINO in Tabs. 2a and 2b, respectively.

Tab. 2a presents the hyper-parameters used when applying Token Adapter to the Segmenter, along with the corresponding evaluation metrics associated with Tab.1 of the main paper. These hyper-parameters and metrics include the positions ($\mathcal{L}_b$ and $\mathcal{L}_b + \mathcal{L}_m$), where token injectors and token ejectors are inserted, the reduction ratios of rows and columns in tokens (r_h and r_w), the proportions of representative tokens in rows and columns (rp_{hr} and rp_{wr}), mIoU, and FPS (im/s).

Tab. 2b presents the hyper-parameters used when applying Token Adapter to different stages of MaskIDNO, along with the corresponding evaluation metrics associated with Tab.2 of the main paper. These hyper-parameters and metrics include the positions ($\mathcal{L}_b$ and $\mathcal{L}_b + \mathcal{L}_m$), where token injectors and token ejectors are inserted, the reduction ratios of rows and columns in tokens (r_h and r_w), the proportions of representative tokens in rows and columns (rp_{hr} and rp_{wr}), PQ (Panoptic Quality), Mask AP (Average Precision), Box AP (Bounding Box Average Precision), mIoU, and FPS (im/s).

1.2 Evaluation Details

In Tab. 1, we present the evaluation details for measuring the computational complexity and inference efficiency of different methods. We calculate the evaluation metrics based on the input resolutions specified in the official configurations of each method. Consistent with the official implementation of Expediting [2], we compute the overall complexity and latency for Segmenter while only calculating the complexity and latency of backbone for MaskDINO.

Table 1: Hyper-parameter settings for computing model inference speed and complexity.

Model	**Dataset**	Input Resolution
Segmenter [3]	ADE20K	640×640
	PASCAL	480×480
	Cityscapes	768×768
MaskDINO [1]	COCO	1152×1152

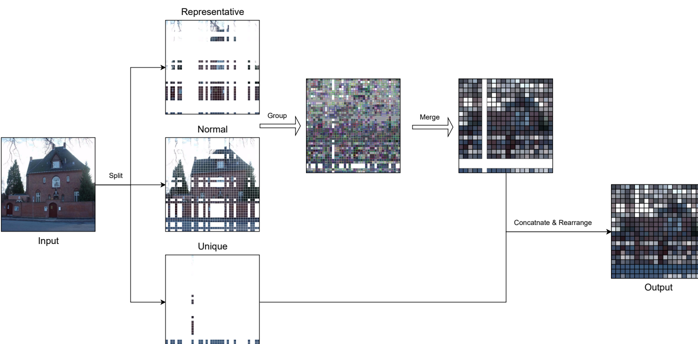

Fig. 1: Work flow of token injector. We begin by categorizing all tokens into representative tokens, normal tokens, and unique tokens. Subsequently, representative tokens and normal tokens are grouped and matched, with the normal tokens being injected into the representative tokens. Finally, the injected results and unique tokens are placed in their original order, resulting in reduced tokens.

1.3 Method Implementation

We present the PyTorch implementation of token adapter in Sec. 1.3, with the omitted parts detailed in Sec.3.2 of the main paper.

2 More Visualization

We visualize more sample images to explain the working flow of our method.

In Fig. 2, we visualize the results of applying our method to Segmenter with different reduction ratios on ADE20K validation dataset. We visualize the input image, matching group results, compressed results, segmentation results, and original segmentation results. Different matching groups are represented by randomly colored bounding boxes, and the colors inside the boxes are filled with the average color of all tokens within that group. The visualized compressed results are obtained by removing the tokens that have been reduced from the matching group results. Please note that the hyper-parameters used for visualization are consistent with Fig.5 of the main paper, with only the reduction ratio being varied. Specifically, we visualize the results with reduction

Table 2: Hyper-parameter settings of all experiments for evaluating task-specific metrics. Here, rp_{hr} and rp_{wr} respectively represent the proportions of representative tokens in the vertical and horizontal directions.

Model	Backbone	Dataset	Settings							Metrics	
			$\mathcal{L}_b$	$\mathcal{L}_m$	$\mathcal{L}_a$	r_h	r_w	rp_{hr}	rp_{wr}	mIoU	im/s
Segmenter [3]	ViT-L/16	ADE20K	13	11	0	0.3	0.3	0.9	0.95	52.02	15.09
			0	24	0					45.73	24.41
		PASCAL-Context	15	9	0	0.4	0.55	0.6	0.65	57.87	34.53
			0	24	0					47.80	63.34
		Cityscapes	14	10	0	0.3	0.35	0.95	0.9	78.48	10.20
			0	24	0					72.08	19.38

(a)

Model	Dataset	Settings								Metrics				
		stage	$\mathcal{L}_b$	$\mathcal{L}_m$	$\mathcal{L}_a$	r_h	r_w	rp_{hr}	rp_{wr}	PQ	Mask AP	Box AP	mIoU	im/s
MaskDINO [1] + Swin-L	COCO (instance seg.)	1	1	1	0	0.3	0.3	0.9	1.0	-	49.23	55.10	-	6.31
			0	2	0					-	40.66	45.43	-	6.61
		2	1	1	0					-	49.05	54.74	-	6.22
			0	2	0					-	42.86	47.61	-	6.41
		3	15	3	0					-	51.42	57.48	-	6.96
			0	18	0					-	31.34	34.43	-	9.78
		4	1	1	0					-	52.34	58.74	-	6.19
			0	2	0					-	52.26	58.72	-	6.32
	COCO (panoptic seg.)	1	1	1	0	0.3	0.3	0.9	1.0	55.91	47.66	52.83	65.31	6.31
			0	2	0					46.99	39.02	43.10	56.34	6.61
		2	1	1	0					56.03	47.59	52.61	66.05	6.22
			0	2	0					51.12	41.76	45.71	60.95	6.41
		3	15	3	0					57.57	49.69	55.04	66.92	6.96
			0	18	0					39.94	29.44	31.81	47.67	9.78
		4	1	1	0					58.42	50.54	58.74	67.32	6.19
			0	2	0					58.17	50.50	56.19	67.33	6.32

(b)

ratios of ($r_h = 0.3, r_w = 0.3$), ($r_h = 0.5, r_w = 0.5$), ($r_h = 0, r_w = 0.6$), and ($r_h = 0.6, r_w = 0$). To facilitate comprehension of our approach, we have additionally visualized the workflow of the token injector in Fig. 1.

From this figure, it is evident that even when compressing only one direction of tokens, the resulting segmentation outcome is still satisfactory. This phenomenon also demonstrates a higher ceiling for dynamic token compression methods, while our approach can be extended to dynamic form through cross-attention modules. Allowing the model to autonomously determine how to reduce tokens would yield superior outcomes.

```
def token_adapter(x: torch.Tensor, r_ratios: tuple, rp_ratios: tuple):
    """
    Args:
        x (torch.Tensor): input tokens with the shape of [B, H, W, C]
        r_ratios (tuple): token reduction ratios with the content (r_h, r_w)
        rp_ratios (tuple):
         - representative token ratios with the content (rp_hr, rp_wr)
    """
    r_h, r_w = r_ratios      #
    b, c, h, w = x.shape     #
    # compute rows' importance scores
    x_H = compute_scores(x, mode='H')     # [B, H, W, C] -> [B, H, 1]
    x_W = compute_scores(x, mode='W')     # [B, H, W, C] -> [B, 1, W]
    # sort by scores to obtain the indices
    H_idx = x_H.argsort(dim=1, descending=True)
    W_idx = x_W.argsort(dim=2, descending=True)
    # reserved tokens' numbers
    resH = h - round(h * r_h)
    resW = w - round(h * r_w)
    # compute the number of representative tokens, unique
    # tokens and normal tokens, respectively
    rp_hr, rp_wr = rp_ratios
    rp_h = round(h * rp_hr)
    rp_w = round(w * rp_wr)
    u_h = resH - rp_h
    u_w = resW - rp_w
    # create indice matrix with each value filled 0/1/2
    x_labs = torch.zeros((b, h, w), device=x.device)     # [B, H, W]
    x_idx = torch.arange(h*w, device=x.device).reshape(1, h, w).expand(b, -1,
                                        -1)   # [B, H, W]
    # unique tokens' indices
    uh_idx = H_idx[:, rp_h:rp_h+u_h]
    uw_idx = W_idx[:, :, rp_w:rp_w+u_w]
    # normal tokens' indices
    nh_idx = H_idx[:, rp_h+u_h:]
    nw_idx = W_idx[:, :, rp_w+u_w:]
    src_lab = torch.ones(size=(1,1,1), dtype=x_labs.dtype, device=x_labs.
                                       device)
    # unique tokens
    x_labs.scatter_(dim=1, index=uh_idx.expand(-1, -1, w),
                    src=src_lab.expand(b, uh_idx.shape[1], w))
    x_labs.scatter_(dim=2, index=uw_idx.expand(-1, h, -1),
                    src=src_lab.expand(b, h, uw_idx.shape[2]))
    # normal tokens
    x_labs.scatter_(dim=1, index=nh_idx.expand(-1, -1, w),
                    src=2*src_lab.expand(b, nh_idx.shape[1], w))
    x_labs.scatter_(dim=2, index=nw_idx.expand(-1, h, -1),
                    src=2*src_lab.expand(b, h, nw_idx.shape[2]))
    def token_injector(x: torch.Tensor):
        # x: [B, N, C]
        x_labs_ = x_labs.flatten(1)      # [B, N]
        x_p = x[x_labs_==0].view(b, -1, c)
        x_u = x[x_labs_==1].view(b, -1, c)
        x_n = x[x_labs_==2].view(b, -1, c)
        # normal tokens are injected into representative tokens
        values, counts = inject_q_kv(q=x_p, kv=x_n)
        value_ = (x_p + values) / (counts + 1)     # average
        value = torch.cat((value_, x_u), dim=1)
        # for ejector
        restore_dist = compute_distance(q=x, kv=value, mode='cosine')   # [B,
                                             N, np+nu]
        return value, restore_dist
    def token_ejector(x_r: torch.Tensor, dist: torch.Tensor):
        # restore the original resolution
        # x_r: [B, np+nu, C]   dist: [B, N, np+nu]
        x_restore, counts = ejector_q_kv(x_r=x_r, dist=dist, threshold_value=
                                            0.5)
        return x_restore / (counts + 1e-10)
    return token_injector, token_ejector
```

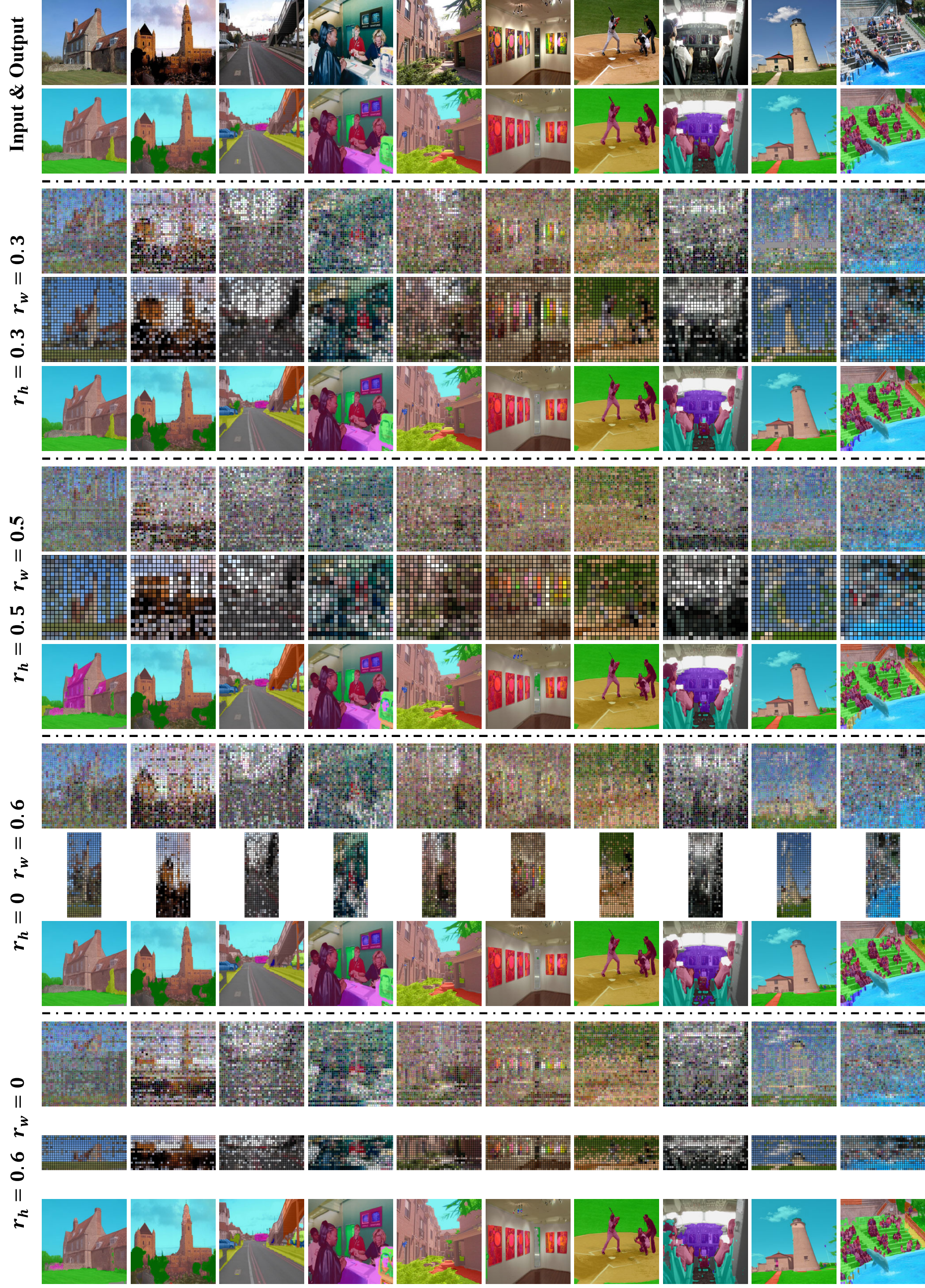

Fig. 2: More Visualizations on ADE20K validation dataset using Segmenter + ViT-L/16 equiped with Token Adapter.

References

1. Li, F., Zhang, H., Xu, H., Liu, S., Zhang, L., Ni, L.M., Shum, H.Y.: Mask dino: Towards a unified transformer-based framework for object detection and segmentation. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 3041–3050 (2023)
2. Liang, W., Yuan, Y., Ding, H., Luo, X., Lin, W., Jia, D., Zhang, Z., Zhang, C., Hu, H.: Expediting large-scale vision transformer for dense prediction without finetuning. Advances in Neural Information Processing Systems **35**, 35462–35477 (2022)
3. Strudel, R., Garcia, R., Laptev, I., Schmid, C.: Segmenter: Transformer for semantic segmentation. In: Proceedings of the IEEE/CVF international conference on computer vision. pp. 7262–7272 (2021)