Efficient Deformable ConvNets: Rethinking Dynamic and Sparse Operator for Vision Applications

Supplementary Material

A. Implementation Details

Environment: We use an A100 80GB SXM GPU to benchmark throughput on all experiments. The software environment is PyTorch 1.13, CUDA 11.7, cuDNN 8.5. When testing Flash Attention [2], we use Flash Attention 2.3.1. When testing Window Attention in Swin Transformer, we use the PyTorch implementation from timm 0.9.7 [3].

2D object detection on COCO: To validate the effectiveness of our method in 2D object detection, we employed two object detection methods: Mask R-CNN and Cascade Mask R-CNN, primarily referring to the settings of Internimage. Following common practices. We used two schedules: 1x (12 epochs) and 3x (36 epochs) to respectively assess the convergence speed and final performance of our model. For the $1\times$ schedule, images are resized such that the shorter side is 800 pixels, with the longer side not exceeding 1,333 pixels. During the evaluation phase, the shorter side of input images is consistently set to 800 pixels. For the $3\times$ schedule, the shorter side is resized to a range between 480 and 800 pixels, while the longer side remains capped at 1,333 pixels. The base learning rate is set at 1e-4 for a batch size of 16. We employ the AdamW optimizer, incorporating a weight decay of 0.05. The initialization of the backbone is the pre-trained classification weights.

2D semantic detection on ADE20K: We employed the UperNet to validate the effectiveness of our method in 2D semantic segmentation on the ADE20K dataset. Our experimental setup is primarily based on InternImage. For FlashInternImage-T/S/B and FlashInternImage-L, we use the AdamW optimizer with learning rates of 6e-5 and 2e-5, respectively. The crop size for FlashInternimage T/S/B is set to 512, while for FlashInternImage-L, it is set to 640. We train all our models using a batch size of 16 for 160k iterations to ensure a fair comparison with previous methods. The initialization of the backbone is also the pre-trained classification weights.

3D object detection on nuScenes: We employed BEVFormerV2 to validate the effectiveness of our method in 3D object detection on nuScenes. Adhering to the settings of BEVFormerV2, the backbone initialization is pretrained on the COCO dataset. In alignment with BEVFormerV2, we utilized data spanning a total of 8 seconds, encompassing both past and future information. We use the AdamW optimizer with a batch size of 16 and a learning rate of 4e-4. We train the model for 24 epochs.

Model	#param	FPS	DINO			
			AP	AP^S	AP^M	AP^L
Swin-T	48.2M	37 / 50	51.1	34.3	53.9	66.1
ConvNext-T	48.5M	39 / 54	51.0	33.5	53.9	65.8
InternImage-T	48.7M	33 / 37	53.9	37.7	57.7	68.7
FlashInternImage-T	48.6M	39 / 48	**54.7**	**37.9**	**58.6**	**69.8**
Swin-B	108M	24 / 32	53.1	36.8	56.7	68.9
ConvNeXt-B	109M	28 / 36	53.1	35.5	56.6	68.5
InternImage-B	116M	24 / 26	54.8	38.2	58.6	70.3
FlashInternImage-S	68.8M	33 / 41	55.3	39.0	59.2	71.1
FlashInternImage-B	116M	28 / 35	**56.0**	**41.2**	**59.8**	**71.2**
Swin-L	218M	17 / 24	56.1	39.6	59.8	71.6
ConvNeXt-L	219M	20 / 27	55.9	39.8	59.5	71.1
InternImage-L	241M	17 / 18	57.6	**44.1**	61.5	73.4
FlashInternImage-L	241M	20 / 26	**58.8**	43.1	**62.6**	**74.6**

Table 1. **Object detection performance with DINO on COCO val2017.** AP^S, AP^M, and AP^L indicate the results for small, middle, and large bounding boxes. FPS metrics are reported based on single-scale testing. FlashInternImage, when integrated with DCNv4, not only achieves superior performance but also maintains a competitive inference speed. All experimental results were obtained with our codebase.

B. Additional Experimental Results

Downstream results with advanced headers: We employed the more advanced DINO [4] and Mask2Former [1] to further validate the effectiveness of our FlashInternImage, as shown in Tab. 1 and Tab. 3. For 2D object detection on COCO with DINO head, we train our model with 12 epochs. During the training phase, we adopt the same multi-scale training strategy that introduced above. Other settings, including optimizer, weight decay, and learning rate, are also the same as those used in Mask-RCNN. For 2D semantic segmentation on ADE20K, we set the learning rate to be 1e-4 with a batch size of 16. For Base and Large scale, we use a crop size of 640. Other settings are the same as those used in UperNet.

Under the application of more advanced task heads, our method still maintains a significant advantage in accuracy while also offering competitive inference speed.

Downstream results for other backbones: As demonstrated in Tab. 4, we evaluated the performance of "ConvNext+DCNv4" and "ViT+DCNv4" on downstream tasks,

Operator	Runtime (ms)				
	$56 \times 56 \times 128$	$28 \times 28 \times 256$	$14 \times 14 \times 512$	$7 \times 7 \times 1024$	$14 \times 14 \times 768$
Conv (3×3)	0.833 / 0.596	0.708 / 0.464	0.653 / 0.384	0.687 / 0.459	1.31 / 0.754
Attention (torch)	32.0 / 20.3	4.06 / 2.66	1.01 / 0.801	0.789 / 0.357	2.28 / 1.44
FlashAttention-2	N/A / 2.81	N/A / 0.641	N/A / 0.256	N/A / 0.209	N/A / 0.451
Window Attn (7×7)	4.48 / 1.87	2.39 / 1.00	1.35 / 0.581	0.824 / 0.371	2.12 / 0.911
DCNv3	3.28 / 2.98	1.62 / 1.42	0.846 / 0.748	0.526 / 0.546	1.37 / 1.10
DCNv4 (lightweight)	0.762 / 0.547	0.419 / 0.313	0.375 / 0.192	0.306 / 0.130	0.389 / 0.249
DCNv4	1.28 / 0.873	0.738 / 0.483	0.452 / 0.324	0.334 / 0.265	0.787 / 0.463

Table 2. **Module-level operator benchmark on standard input shape with various downsample rates.**

Model	crop size	#param	FPS	mIoU (SS)
Swin-T	512^2	47.4M	96 / 146	49.6
ConvNeXt-T	512^2	47.7M	111 / 155	49.6
InternImage-T	512^2	48.6M	91 / 123	50.6
FlashInternImage-T	512^2	48.5M	105 / 145	**51.3**
Swin-B	640^2	110M	48 / 71	51.9
ConvNeXt-B	640^2	111M	54 / 77	50.7
InternImage-B	640^2	118M	45 / 58	52.1
FlashInternImage-S	640^2	71.5M	60 / 80	52.6
FlashInternImage-B	640^2	118M	55 / 75	**53.4**
Swin-L	640^2	218M	37 / 55	56.1
ConvNeXt-L	640^2	220M	41 / 60	55.7
InternImage-L*	640^2	242M	33 / 45	55.5
FlashInternImage-L	640^2	242M	41 / 59	**56.7**

Table 3. **Semantic segmentation performance on the ADE20K validation set.** All models are trained with Mask2Former. "SS" denotes single-scale testing. FPS is reported with single-scale testing. FlashInternImage w/ DCNv4 exhibits a significant advantage in terms of performance while also maintaining competitive inference speed. *: In this experiment, we observed a significant overfitting phenomenon.

specifically focusing on semantic segmentation tasks using the UperNet head. Our observations indicate that substituting the previously used DWConv or Attention with our DCNv4 leads to an increase in inference speed. For ConvNext, using DCNv4 rather than DWConv also achieves higher performance.

Visualization for image generation: We show qualitative results of our latent diffusion model with DCNv4 in Fig. 1 for a better illustration. DCNv4 also can work well in this generation task.

Module-level speed benchmark: We show module-level speed benchmark results in Tab. 2 and 5, where additional linear projection layers in each operator are also considered. We also include the regular convolution with a 3×3 window here as it mixes channel information. For DCNv4, we show two variants: where the first implementation removes the input/output projection layers inside the module (denoted as lightweight). We use this implementation in the "ConvNeXt + DCNv4" experiments as it shares similar properties (only performs spatial aggregation) and the amount of computation/parameters as the original depthwise convolution in ConvNeXt. The second implementation includes the input/output linear projections and is used in the rest of the models described in the paper.

Model	mIoU	FPS
ConvNeXt-B	49.1	95 / 147
ConvNeXt-B + DCNv4	49.9	96 / 164
ViT-B	48.8	51 / 74
ViT-B + DCNv4	48.8	67 / 92

Table 4. **DCNv4 in other architecture.** All models are trained with UperNet on ADE20K. By replacing ConvNext's DWConv with DCNv4, we not only achieved better inference speeds but also obtained improved results. Replacing the Attention in ViT with DCNv4 resulted in faster inference speeds while maintaining the same level of performance.

References

[1] Bowen Cheng, Ishan Misra, Alexander G Schwing, Alexander Kirillov, and Rohit Girdhar. Masked-attention mask transformer for universal image segmentation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 1290–1299, 2022. 1

[2] Tri Dao. Flashattention-2: Faster attention with better parallelism and work partitioning. *arXiv preprint arXiv:2307.08691*, 2023. 1

[3] Ross Wightman. Pytorch image models. https://github.com/rwightman/pytorch-image-models, 2019. 1

[4] Hao Zhang, Feng Li, Shilong Liu, Lei Zhang, Hang Su, Jun Zhu, Lionel M Ni, and Heung-Yeung Shum. Dino: Detr with improved denoising anchor boxes for end-to-end object detection. *arXiv preprint arXiv:2203.03605*, 2022. 1

Operator	Runtime (ms)				
	$200 \times 320 \times 128$	$100 \times 160 \times 256$	$50 \times 80 \times 512$	$25 \times 40 \times 1024$	$64 \times 64 \times 768$
Conv (3×3)	0.602 / 0.234	0.623 / 0.199	0.725 / 0.214	0.422 / 0.281	0.532 / 0.318
Attention (torch)	OOM / OOM	26.0 / 13.1	3.05 / 1.99	0.599 / 0.433	4.47 / 2.69
FlashAttention-2	N/A / 13.4	N/A / 1.81	N/A / 0.354	N/A / 0.158	N/A / 0.531
Window Attn (7×7)	1.49 / 0.657	0.867 / 0.377	0.539 / 0.274	0.409 / 0.276	0.920 / 0.410
DCNv3	1.06 / 0.964	0.576 / 0.560	0.451 / 0.459	0.387 / 0.409	0.534 / 0.543
DCNv4 (lightweight)	0.283 / 0.207	0.187 / 0.143	0.134 / 0.110	0.105 / 0.0912	0.180 / 0.125
DCNv4	0.446 / 0.346	0.313 / 0.272	0.267 / 0.270	0.268 / 0.253	0.325 / 0.271

Table 5. **Module-level operator benchmark on high-resolution input shape with various downsample rates.**

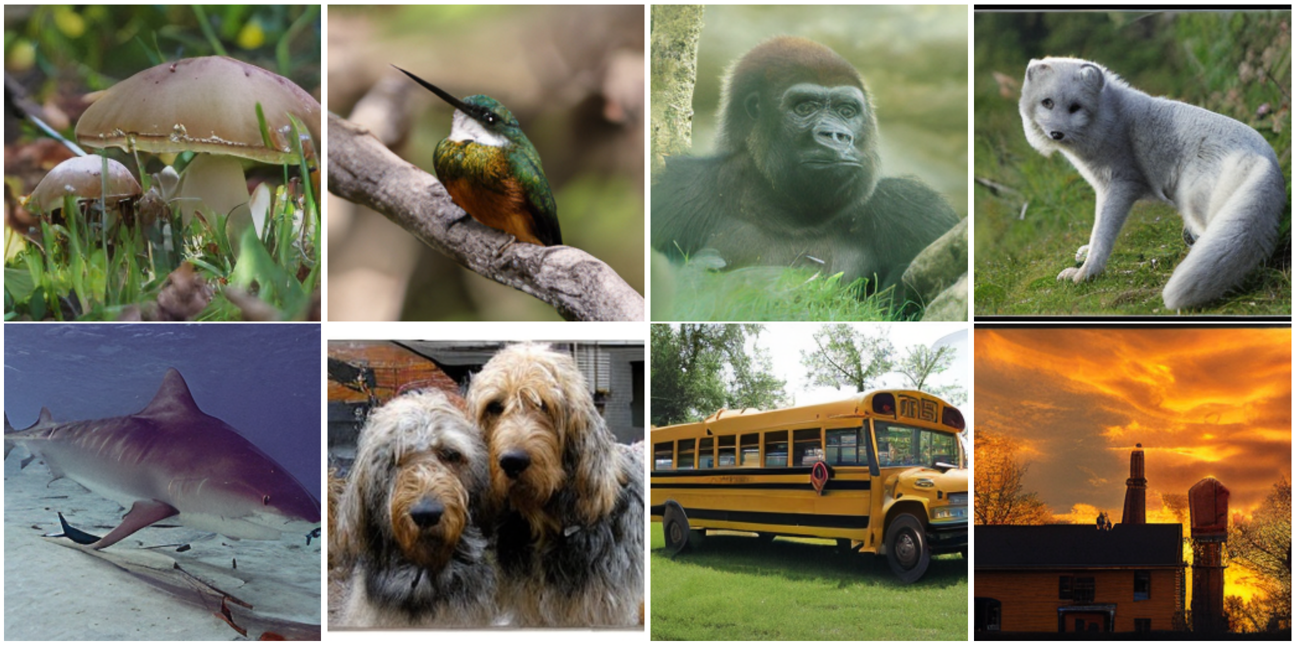

Figure 1. ImageNet 256×256 generation results of U-Net + DCNv4 latent diffusion model.