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

Yuwen Xiong*[1,2] Zhiqi Li*[3,2] Yuntao Chen*[4] Feng Wang*[5]
Xizhou Zhu[5,6] Jiapeng Luo[6] Wenhai Wang[7,2] Tong Lu[3] Hongsheng Li[7]
Yu Qiao[2] Lewei Lu[6] Jie Zhou[5] Jifeng Dai[5,2]✉

[1]University of Toronto [2]OpenGVLab, Shanghai AI Laboratory
[3]Nanjing University [4]CAIR, HKISI, CAS [5]Tsinghua University
[6]SenseTime Research [7]The Chinese University of Hong Kong
https://github.com/OpenGVLab/DCNv4

Abstract

We introduce Deformable Convolution v4 (DCNv4), a highly efficient and effective operator designed for a broad spectrum of vision applications. DCNv4 addresses the limitations of its predecessor, DCNv3, with two key enhancements: 1. removing softmax normalization in spatial aggregation to enhance its dynamic property and expressive power and 2. optimizing memory access to minimize redundant operations for speedup. These improvements result in a significantly faster convergence compared to DCNv3 and a substantial increase in processing speed, with DCNv4 achieving more than three times the forward speed. DCNv4 demonstrates exceptional performance across various tasks, including image classification, instance and semantic segmentation, and notably, image generation. When integrated into generative models like U-Net in the latent diffusion model, DCNv4 outperforms its baseline, underscoring its possibility to enhance generative models. In practical applications, replacing DCNv3 with DCNv4 in the InternImage model to create FlashInternImage results in up to 80% speed increase and further performance improvement without further modifications. The advancements in speed and efficiency of DCNv4, combined with its robust performance across diverse vision tasks, show its potential as a foundational building block for future vision models.

1. Introduction

In the field of computer vision, there is an ongoing debate about whether convolutional networks (ConvNets) or Transformers offer superior performance. In recent years, Transformer models [12, 25, 44] have achieved remarkable results in large vision models with the attention mechanism, showing the potential to overtake ConvNets. However, recent works such as InternImage [38] and ConvNeXt [26] demonstrate that ConvNet-based vision models retain robust performance, efficiency, simplicity, and suitable inductive bias for various downstream tasks [15, 41]. Notably, in domains like image generation [29, 31], models containing both convolutions and transformers remain the preferred approach. This situation brings to light the enduring value of convolution-based approaches.

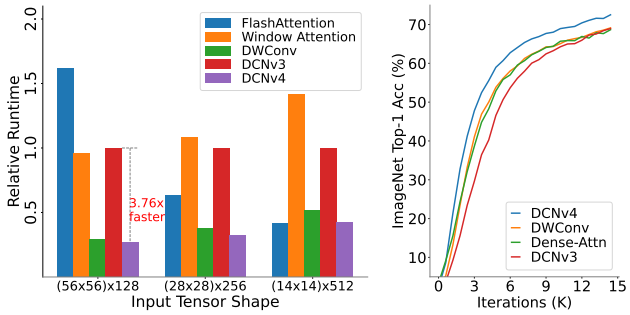

Figure 1. (a) We show relative runtime with DCNv3 as the baseline. DCNv4 shows **significant speedup** over DCNv3, and surpasses other common vision operators. (b) With the same network architecture, DCNv4 **converges faster** than other operators.

Building on convolution's strengths, Deformable Convolution v3 (DCNv3) – the core operator of the advanced ConvNet model InternImage – innovatively combines a sparse attention mechanism with convolution: it processes each output location in a sliding window manner with a small window size (*e.g.* $3 \times 3 = 9$ points) which acts as a local, sparse operator like convolution, while dynamically samples point with an adaptive range and aggregates the spatial features with input-dependent attention weights. With its

* Equal contribution
✉ Corresponding author (daijifeng@tsinghua.edu.cn)

small window size and ConvNet inductive bias, DCNv3 is expected to achieve a faster convergence rate and lower inference latency, especially when compared to dense global [12] or local window-based [25] attention methods.

Despite these advantages, DCN has not become the go-to solution for vision backbone models. This observation led us to investigate the lingering limitations of the DCN operator. The first thing we notice is the running speed. The slow speed of DCN is known to be a long-standing problem [1], as it introduces extra overhead on sampling non-nearby locations, making it not fit modern convolution algorithms. Our comparative analysis, illustrated in Fig. 1a, reveals that DCNv3 can be slower than a properly optimized dense global attention [9], highlighting the need for further optimization. Moreover, we find DCNv3 even converges slower than global attention at the initial backbone training phase, as shown in Fig. 1b, which is counter-intuitive as DCNv3 is equipped with ConvNet inductive bias.

To overcome these challenges, we propose Deformable Convolution v4 (DCNv4), an innovative advancement to optimize the sparse DCN operator for practical efficiency. DCNv4 comes with a much faster implementation and an improved operator design to enhance its performance, which we will elaborate on as follows:

First, we conduct instruction-level kernel profiling for existing implementation and find that DCNv3 is already lightweight. The compute cost is less than 1%, while memory access costs 99%. This motivates us to revisit the operator implementation and find that many memory accesses in the DCN forward process are redundant and thus can be optimized, leading to a much faster DCNv4 implementation.

Second, drawing inspiration from convolution's unbounded weight range, we find that softmax in spatial aggregation, a standard operation in dense attention, is *unnecessary* in DCNv3, as it is not a requirement for operators with a dedicated aggregation window for each location. Intuitively, softmax puts a bounded $0 \sim 1$ value range to the weight and will limit the expressive power of the aggregation weight. This insight led us to remove the softmax in DCNv4, enhancing its dynamic property and improving its performance.

As a result, DCNv4 converges significantly faster than DCNv3 and accelerates forward speed by more than $3\times$. This improvement allows DCNv4 to fully leverage its sparse property and become one of the fastest core vision operators.

We further replace DCNv3 in InternImage with DCNv4, creating FlashInternImage. Remarkably, FlashInternImage achieves a $50 \sim 80\%$ speed increase compared to InternImage without any additional modifications. This enhancement positions FlashInternImage as one of the fastest modern vision backbone networks while maintaining superior performance. With the help of DCNv4, FlashInternImage significantly improves the convergence speed in ImageNet classification [10] and transfer learning settings and further demonstrates improved performance in downstream tasks.

Furthermore, DCNv4 shows potential as a universal vision operator in various architectures and tasks. We integrate DCNv4 into other modern backbone architectures, including ConvNeXt [26] and ViT [12], replacing depthwise convolution [6] and dense self-attention layers [35]. Surprisingly, without any hyperparameter adjustments, these meticulously designed networks with DCNv4 perform on par while being much faster, showing the efficacy and efficiency of the dynamic, sparse DCNv4. Moreover, we explore the potential of DCNv4 in generative models as a new application domain. Specifically, we apply it in the U-Net [30] architecture used in diffusion models [29], replacing regular convolution with DCNv4. Our experimental results show that DCNv4 can work better than the baselines in image generation, showing great potential for using DCN to improve generative models.

We have released the DCNv4 implementation to facilitate future research in the vision community.

2. Related Work

Core operators in vision models: The standard convolution [17] stands as the most prevalent and impactful operator, forming the backbone of the majority of computer vision architectures [14, 16, 32]. Nevertheless, a myriad of operators, each with unique characteristics, collectively play a crucial role in the development of computer vision. Depthwise separable convolution (DWConv) [6] separates the spatial and channel operations, and has been pivotal in developing lightweight and efficient models [26, 27]. RepLKNet [11] illustrates that a purely convolutional network, leveraging large-kernel depth-wise convolutions, can attain competitive performance in both efficiency and effectiveness. Deformable Convolution (DCN) series [7, 38, 47] significantly leaps the adaptability of convolution by adding learnable offsets to the convolutions kernels. Contrary to convolutions, attention mechanisms [35] possess the capacity to model long-range dependencies and have been successfully adopted in various computer vision tasks [3, 12, 24, 33]. Window attention [25, 36] reduces the computational complexity inherent in vanilla attention by restricting the attention operation to a fixed-size window. To mitigate the high computational complexity associated with vanilla attention, deformable attention [48] enables each query to concentrate on a select number of key sampling points, with dynamically determined locations and weights. This efficient method is widely used in the following arts perception methods [4, 19, 21, 22, 43, 45]. DynamicConv [40] and dynamic-DWNet [13] augment DWConv by incorporating dynamic weights, thereby enabling the use of instance-specific weights that adapt dynamically. For non-grid structured data, sparse operators [34, 37, 42] utilize dynamic weights obtained via bilinear interpolation or in a parametric way.

Memory access cost in vision backbones: As underscored in previous studies [18, 27], FLOPs, although a fre-

Model	5th EP	10th Ep	20th Ep	50th Ep	300th Ep
ConvNeXt	29.9	53.5	66.1	74.8	83.8
ConvNeXt	8.5	25.3	51.1	69.1	81.6
+ softmax	(-21.4)	(-28.2)	(-15.0)	(-5.7)	(-2.2)

Table 1. **ImageNet-1K accuracy at different training epochs.** Softmax on the convolution weights significantly affects the convergence speed and the final performance for the ConvNeXt model.

quently used metric to measure model complexity, do not accurately represent the model's speed or latency. In practical scenarios, the running speed of a model is influenced by multiple factors, not just FLOPs. Memory access cost plays a significant role in this context [27]. Flash-Attention [9], by reducing the number of accesses to High Bandwidth Memory (HBM), achieves a significantly faster speed in practice despite having higher FLOPs compared to vanilla attention. Although DCN operators do not exhibit a disadvantage in terms of FLOPs, their latency is considerably longer compared to DW-Conv, under the same FLOPs budget, predominantly due to substantial memory access costs. In this work, we conduct a thorough analysis and optimization of the memory access costs associated with the DCN operators, significantly accelerating the DCN's running speed.

3. Method

3.1. Rethinking the Dynamic Property in Deformable Convolution

Revisiting DCNv3: Given an input $\mathbf{x} \in \mathbb{R}^{H \times W \times C}$ with height H, width W and channel C, the DCNv3 operation with K points is defined in Eq. (2) for each point p_0:

$$\mathbf{y}_g = \sum_{k=1}^{K} \mathbf{m}_{gk} \mathbf{x}_g(p_0 + p_k + \Delta p_{gk}), \tag{1}$$

$$\mathbf{y} = \text{concat}([\mathbf{y}_1, \mathbf{y}_2, ..., \mathbf{y}_G], \text{axis=-1}), \tag{2}$$

where G denotes the number of spatial aggregation groups. For the g-th group, $\mathbf{x}_g, \mathbf{y}_g \in \mathbb{R}^{H \times W \times C'}$ represents the sliced input/output feature map with $C' = C/G$ represents the group dimension; $\mathbf{m}_{gk} \in \mathbb{R}$ denotes the spatial aggregation weights (also known as modulation scalar) of the k-th sampling point in the g-th group, conditioned on the input $\mathbf{x}$ and normalized by the softmax function along the dimension K; p_k denotes the k-th location of the pre-defined grid sampling $\{(-1,-1), (-1,0), ..., (0,+1), ..., (+1,+1)\}$ as in regular convolutions and Δp_{gk} is the offset corresponding to the grid sampling location p_k in the g-th group. A 1×1 point-wise convolution on $\mathbf{x}$ and $\mathbf{y}$ can be applied before and after the DCNv3 operator to enhance the model's expressive power, following separable convolution [6]. DCNv3 is a combination of convolution and attention: on the one hand, it processes the input data in a sliding window manner, which follows convolution and inherent its inductive bias; on the other hand, the sampling offset Δp and spatial aggregation weight $\mathbf{m}$ are dynamically predicted from the input feature,

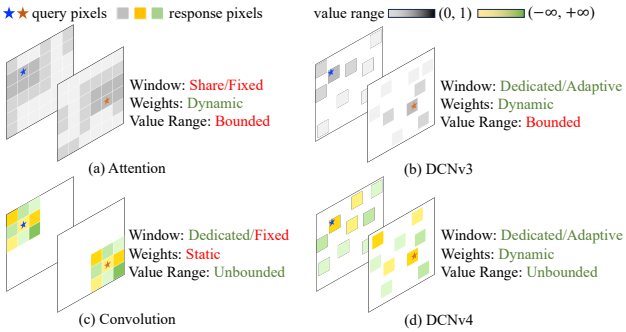

Figure 2. **Comparisons of core operators in spatial aggregation for query pixels on different locations within the same channel.** (a) Attention and (b) DCNv3 use bounded (range from $0 \sim 1$) dynamic weights to aggregate spatial features, while the window (sampling point set) for attention is the same, and DCNv3 uses a dedicated window for each location. (c) Convolution has a more flexible unbounded value range for aggregation weights and uses a dedicated sliding window for each location, but the window shape and aggregation weights are input-independent. (d) DCNv4 combines their advantages, using an adaptive aggregation window and dynamic aggregation weights with an unbounded value range.

showing its dynamic property and making it more like an attention mechanism. We compare different operators where each has its own property, as illustrated in Fig. 2

Softmax normalization: A key difference between convolution and DCNv3 is that DCNv3 *normalizes* $\mathbf{m}$, the spatial aggregation weights, with a softmax function, following the convention of scaled dot-product self-attention. Conversely, convolution does not employ softmax over its weights and still works well. The reason why attention needs a softmax is straightforward: scaled dot-product self-attention with $Q, K, V \in \mathbb{R}^{N \times d}$ is defined with a formulation:

$$\text{softmax}(\frac{1}{\sqrt{d}} QK^\top)V, \tag{3}$$

where N is the number of points in the same attention window (can be either global [12] or local [25]), d is the hidden dimension, Q, K, V are the query, key, and value matrices computed from the input. Softmax operation is required in Eq. (3) for attention; without softmax, $K^\top V \in \mathbb{R}^{d \times d}$ can be calculated first, and it degrades to a linear projection for all queries in the same attention window, resulting in degenerated performance. However, for convolutional operators like depthwise convolution and DCNv3 where each point has its own dedicated aggregation window and the values in each aggregation window are already different and there is no "key" concept, such degradation issue no longer exists, and the normalization becomes unnecessary. In fact, normalizing convolution weights within a fixed 0-1 range using softmax could impose a significant limitation on the operator's expressive power and make the learning slower.

To confirm this hypothesis, we train a ConvNeXt model and apply softmax to the 7×7 window of the depthwise

convolution weights before convolution forward. We observe a remarkable decline in model performance as well as convergence speed from results in Tab. 1. This suggests that for operators with a dedicated aggregation window on each location like convolution or DCN, aggregation weights with an unbounded range offer better expressive power than softmax-normalized, bounded-range weights.

Enhancing dynamic property: Motivated by this observation, we remove the softmax normalization in DCNv3, transforming the modulation scalars ranging from 0 to 1 to unbounded dynamic weights similar to convolution. As shown in Fig. 2, this alteration further amplifies the dynamic property of DCN, where other operators have certain limits, such as bounded value range (attention/DCNv3) or fixed aggregation window with input-independent aggregation weights (convolution). Fig. 1b shows that by making this change, DCNv4 converges significantly faster than DCNv3 and other common operators, including convolution and attention. Results in Sec. 4 further showcase that DCNv4 works well in both pre-training and transfer learning settings.

3.2. Speeding up DCN

Theoretically, DCN, as a sparse operator with 3×3 window, should act faster than other common operators that employ larger window sizes, like dense attention or 7×7 depthwise convolution. However, we find that this is not the case, as shown in Fig. 1a. In this subsection, we first conduct a theoretical analysis of GPU efficiency, showing a large variance in memory access cost depending on how we read the memory. We further perform optimization based on our observations, significantly improving the speed of DCN by saving additional memory instruction and bringing the speed advantage of being a sparse operator into reality.

Theoretical analysis of GPU efficiency Our study begins with a theoretical examination of the DCNv3 operator's computational behavior. We employ the roofline model to evaluate its performance, focusing on theoretical FLOPs and memory access cost. For an input and output tensor of shape (H, W, C), the DCNv3 operator requires $36HWC$ FLOPs, where 3×3 represents the convolution kernel's spatial dimensions and the factor of 4 accounts for the bilinear interpolation at each sampling point.

Following the framework outlined in [27], DCNv3's memory access cost is calculated as $2HWC+27HWG$. The first term corresponds to the input/output feature map size and the second to the DCNv3's offset and aggregation weights with G groups. We approximate G as $C/16$ assuming a group dimsion of 16, resulting in approximately $3.7HWC$ memory access cost. However, this assumes an ideal scenario of infinite cache and a single memory read for each value, which is often unrealistic in parallel computing environments where concurrent thread execution necessitates simultaneous data access.

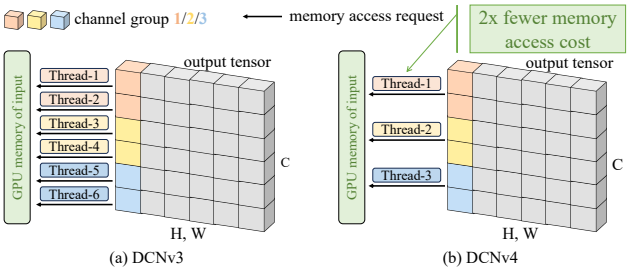

Figure 3. **Illustration of our optimization.** In DCNv4, we use one thread to process multiple channels in the same group that shares sampling offset and aggregation weights. Workloads like memory reading and bilinear interpolation coefficient computation can be reduced, and multiple memory access instructions can be merged.

To estimate the maximum memory access requirement, we consider a scenario devoid of cache, where each output location requires fresh memory reads and involves 36 reads for bilinear interpolation, 27 for offset/aggregation weights, and one write operation, resulting in a memory access cost of $64HWC$. This is 17 times larger than the ideal case.

This analysis reveals a substantial gap in the ratio of computation-to-memory access (ranging from 0.6 to 9.7), highlighting the significant potential for memory access optimization. Notably, despite DCNv3's use of input-dependent, dynamic offsets causing non-deterministic memory access, one deterministic thing is that channels within the same group share offset values. This leads us to propose a specific optimization strategy for enhancing DCNv3's speed.

Eliminating redundant workload: In previous CUDA implementations of DCN kernel, for input with shape (H, W, C)[1], offset $(H, W, G, K^2 \times 2)$ and aggregation weight (H, W, G, K^2), we will create $H \times W \times C$ threads in total to maximize parallelism, where each thread processes one channel for one output location. Notably, the $D = C/G$ channels within each group share the same sampling offset and aggregation weight values for each output location. Using multiple threads to process these D channels on the same output location is wasteful as different threads will read the same sampling offset and aggregation weight values from GPU memory multiple times, which is critical for a memory-bound operator. Processing multiple channels within the same group on each output location with one thread can eliminate these redundant memory read requests, greatly reducing memory bandwidth usage. As the sampling locations are the same, we can also only calculate the bilinear interpolation coefficient used in DCN once. Specifically, if each thread processes D' channels, the memory access cost for reading offset and aggregation weight, as well as the computation cost for calculating bilinear interpolation coefficient, can be reduced D' times.

Eliminating redundant memory instructions: In practice, solely reusing threads for multiple channels will not

[1]We assume the batch size is 1 and memory layout is channel-last.

Operator	Runtime (ms)				
	$56 \times 56 \times 128$	$28 \times 28 \times 256$	$14 \times 14 \times 512$	$7 \times 7 \times 1024$	$14 \times 14 \times 768$
Attention (torch)	30.8 / 19.3	3.35 / 2.12	0.539 / 0.448	0.446 / 0.121	0.779 / 0.654
FlashAttention-2	N/A / 2.46	N/A / 0.451	N/A / **0.123**	N/A / 0.0901	N/A / 0.163
Window Attn (7×7)	4.05 / 1.46	2.07 / 0.770	1.08 / 0.422	0.577 / 0.239	1.58 / 0.604
DWConv (7×7, torch)	2.02 / 1.98	1.03 / 1.00	0.515 / 0.523	0.269 / 0.261	0.779 / 0.773
DWConv (7×7, cuDNN)	0.981 / 0.438	0.522 / 0.267	0.287 / 0.153	0.199 / 0.102	0.413 / 0.210
DCNv3	1.45 / 1.52	0.688 / 0.711	0.294 / 0.298	0.125 / 0.126	0.528 / 0.548
DCNv4	**0.606 / 0.404**	**0.303 / 0.230**	**0.145 / 0.123**	**0.0730 / 0.0680**	**0.224 / 0.147**

Table 2. **Op-level benchmark on standard input shape with various downsample rates.** FP32/FP16 results are reported when the implementation is available. Our new DCNv4 can surpass all other commonly used operators under different input resolutions.

see speed improvement. The reason is that when D' increases, we create fewer threads and the workload of each thread now increases D' times. This essentially reduces the degree of parallelism for the CUDA kernel. Luckily, the DCN kernel is now computationally lightweight as the bilinear interpolation only needs to be performed once for all D' channels, and most of the workload is the memory instructions reading input values from different channels. When the memory layout is channel-last, and all D' channel values are contiguous, we can leverage vectorized load: for example, to read four 32-bit float values from memory, instead of reading one 32-bit float value four times with four instructions, we can use a single instruction to load a 128-bit packed value at once, thus reducing the number of instructions and execution time of each thread. We can apply similar technique when writing the final results to GPU memory, minimizing the memory access time and increasing memory bandwidth utilization. Moreover, the modern half-precision data format (float16/bfloat16) halves the bytes that need to be loaded, which means the memory efficiency can be twice as much under the same memory bandwidth when using the half-precision format. However, we do not see speed improvement with half-precision data in the original DCNv3 implementation, possibly due to too much overhead on data access and computation, while in our new implementation, the speedup is significant. It is worth noting that the aforementioned optimization techniques can also be applied to DCNv1/v2 and deformable attention [48], as they share a similar performance bottleneck and issue.

Micro design in DCN module: DCNv3 module introduces multiple micro designs; as the core kernel is optimized, their impact on the speed becomes non-negligible. We identify two points in DCNv3 designs that could be further optimized: first, after removing the softmax and transforming the modulation scalar into dynamic aggregation weights as mentioned in the previous paragraph. The linear layers for computing offset and dynamic weights can actually be combined into one linear layer. This reduces network fragmentation and eliminates extra overheads, such as kernel launching and synchronization, enhancing run-time efficiency on the GPU; second, in the original DCNv3 module design, a complex sub-network that consists of depthwise 3×3 conv, layer normalization (LN), GELU, and linear layer is used to compute offsets and dynamic weights. Following the design in Xception [6], we remove the additional LN-GELU layers and use the original separable convolution structure, further reducing running time. We empirically find that if latency is a higher priority, the depthwise convolution can also be removed with only a minor performance sacrifice.

4. Experiments

In this section, we verify the effectiveness of our proposed DCNv4 module from both speed as well as performance perspective. We benchmark the operator-level speed and integrate DCNv4 into the backbone to test the system-level performance. All speed test results are obtained with an NVIDIA A100 80G SXM GPU. We include additional experimental results and implementation details in supp.

4.1. Speed Benchmark for Operators

Settings: We conduct the op-level benchmark by solely measuring the running time of several representative operators building state-of-the-art vision backbone models, including full attention [35] implemented with PyTorch as well as the advanced FlashAttention-2 [8] implementation, window-based attention with window size 7×7 [25], depthwise convolution with 7×7 window, implemented by cuDNN [5] and ATen library from PyTorch [28], respectively. We only benchmark the core operation for spatial aggregation, and additional linear layers like qkv projection and output projection layers are excluded and not included in the runtime measurement. Please refer to supp. for a more comprehensive module-level comparison. We first consider a feature map shape generated from the standard 224×224 input resolution for image classification with 4, 8, 16, $32\times$ downsample ratio as used by common hierarchical ConvNet/transformer backbones; we also add a feature map shape from isotropic backbone like ViT with a downsampling ratio 16 and larger hidden dimension. We further consider high-resolution inputs often used in downstream tasks like object detection. We set the input shape to be 800×1280 and 1024×1024 for the hierarchical feature map and isotropic feature map, respectively, as they are the common practice in object detection [15, 20]. Batch size is 64 and 1 for these two input sets, respectively. For operators with a head/group concept, we

Operator	Runtime (ms)				
	$200 \times 320 \times 128$	$100 \times 160 \times 256$	$50 \times 80 \times 512$	$25 \times 40 \times 1024$	$64 \times 64 \times 768$
Attention (torch)	OOM / OOM	25.4 / 12.9	2.88 / 1.89	0.490 / 0.309	4.17 / 2.57
FlashAttention-2	N/A / 13.2	N/A / 1.74	N/A / 0.285	N/A / 0.0797	N/A / 0.437
Window Attn (7×7)	1.33 / 0.509	0.728 / 0.291	0.426 / 0.186	0.279 / 0.165	0.673 / 0.272
DWConv (7×7, torch)	0.634 / 0.608	0.313 / 0.315	0.167 / 0.158	0.0943 / 0.0894	0.260 / 0.253
DWConv (7×7, cuDNN)	0.331 / 0.282	0.188 / 0.168	0.114 / 0.115	0.0817 / 0.0881	0.161 / 0.156
DCNv3	0.472 / 0.493	0.244 / 0.253	0.128 / 0.132	0.0737 / 0.0767	0.194 / 0.199
DCNv4	**0.210 / 0.136**	**0.124 / 0.0895**	**0.0707 / 0.0589**	**0.0452 / 0.0426**	**0.103 / 0.0672**

Table 3. **Op-level benchmark on high-resolution input shape with various downsample rates.** DCNv4 performs well as a sparse operator, surpassing all other baselines, while dense global attention is slow under this scenario.

set the dimension of each head/group to 32 and change the number of heads/groups when the hidden dimension varies.

Results: We show the benchmark results on standard resolution and high-resolution input in Tab. 2 and Tab. 3. We report results with both FP32 and FP16 data formats unless the FP32 implementation is not available. Dense global attention implemented with PyTorch performs significantly slower when the input resolution is large and even out of memory. FlashAttention significantly improves the speed of attention and can be even faster than 7×7 window attention in certain cases, indicating the importance of proper optimization. However, it does not change the quadratic complexity of attention; when the input resolution is high, it still falls behind local/sparse operators like window attention or convolution. While DCNv3 can be faster than DWConv with plain implementation, it is slower than the heavily optimized cuDNN version. Instead, our DCNv4 can provide more than $3\times$ speedup compared to DCNv3, greatly saving the running time. Moreover, DCNv4 can leverage the advantage of using a 3×3 sparse window to perform significantly faster than other baselines under different settings.

4.2. Image Classification

Settings: We evaluate the effectiveness of DCNv4 on ImageNet classification. We start from InternImage [38] as it shows state-of-the-art performance among ConvNet-based models. We replace the DCNv3 in InternImage with DCNv4 and create FlashInternImage. Other implementation details, including network architecture and hyperparameters, are kept the same as [38]. We compare Swin-Transformer and ConvNeXt which are two representative baselines in Transformer and ConvNet models. We follow the common practice [25, 26, 38] of training protocols, including data augmentation, preprocessing, optimizer and learning rate schedule, and train FlashInternImage-Tiny/Small/Base on ImageNet-1K for 300 epochs. FlashInternImage-Large is trained on ImageNet-22K for 90 epochs and then fine-tuned on ImageNet-1K for 20 epochs. Other baselines share the same setting for a fair comparison.

Results: Tab. 4 shows the results of models at various scales. Besides the model size and training/inference resolution, we also report each model's overall throughput (number of images per second) in FP32/FP16 data format. We use timm [39] implementation of ConvNeXt and Swin Transformer, which is faster than the original implementation. Equipped with DCNv4, FlashInternImage significantly improves the throughput of the InternImage counterpart over $50\% \sim 80\%$ and slightly improves the model performance. FlashInternImage now matches the speed of ConvNeXt with higher accuracy. It is noteworthy that FlashInternImage-S can outperform ConvNeXt-B (84.4% *vs*. 83.8%) while being faster than it, showing a better speed-accuracy trade-off. Moreover, the FlashInternImage-L can surpass ConvNeXt-XL and InternImage-XL and being $30\% \sim 130\%$ (401 *vs*. 174) faster, showing the effectiveness of our DCNv4 module.

Model	Size	Scale	Acc	Throughput
Swin-T	29M	224^2	81.3	1989 / 3619
ConvNeXt-T	29M	224^2	82.1	2485 / 4305
InternImage-T	30M	224^2	83.5	1409 / 1746
FlashInternImage-T	30M	224^2	**83.6**	2316 / 3154 (+64%/ + 80%)
Swin-S	50M	224^2	83.0	1167 / 2000
ConvNeXt-S	50M	224^2	83.1	1645 / 2538
InternImage-S	50M	224^2	84.2	1044 / 1321
FlashInternImage-S	50M	224^2	**84.4**	1625 / 2396 (+56%/ + 81%)
Swin-B	88M	224^2	83.5	934 / 1741
ConvNeXt-B	89M	224^2	83.8	1241 / 1888
InternImage-B	97M	224^2	**84.9**	779 / 1030
FlashInternImage-B	97M	224^2	**84.9**	1174 / 1816 (+51%/ + 76%)
Swin-L	197M	384^2	87.3	206 / 301
ConvNeXt-L	198M	384^2	87.5	252 / 436
InternImage-L	223M	384^2	87.7	158 / 214
ConvNeXt-XL	350M	384^2	87.8	170 / 299
InternImage-XL	335M	384^2	88.0	125 / 174
FlashInternImage-L	223M	384^2	**88.1**	248 / 401 (+57%/ + 87%)

Table 4. **Image classification performance on ImageNet-1K.** We show relative speedup between FlashInternImage w/ DCNv4 and its InternImage counterparts. DCNv4 significantly improves the speed while shows state-of-the-art performance.

4.3. Downstream Tasks with High-Resolution Input

We evaluate the performance of DCNv4 on representative downstream perception tasks with high-resolution input, in-

Model	#param	FPS	Mask R-CNN 1× AP^b	1× AP^m	3×+MS AP^b	3×+MS AP^m
Swin-T	48M	66 / 106	42.7	39.3	46.0	41.6
ConvNeXt-T	48M	78 / 113	44.2	40.1	46.2	41.7
InternImage-T	49M	54 / 69	47.2	42.5	49.1	43.7
FlashInternImage-T	49M	72 / 102	**48.0**	**43.1**	**49.5**	**44.0**
Swin-S	69M	45 / 77	44.8	40.9	48.2	43.2
ConvNeXt-S	70M	54 / 83	45.4	41.8	47.9	42.9
InternImage-S	69M	44 / 56	47.8	43.3	49.7	44.5
FlashInternImage-S	69M	57 / 83	**49.2**	**44.0**	**50.5**	**44.9**
Swin-B	107M	33 / 59	46.9	42.3	48.6	43.3
ConvNeXt-B	108M	43 / 70	47.0	42.7	48.5	43.5
InternImage-B	115M	33 / 43	48.8	44.0	50.3	44.8
FlashInternImage-B	115M	44 / 67	**50.1**	**44.5**	**50.6**	**45.4**

Model	#param	FPS	Cascade Mask R-CNN 1× AP^b	1× AP^m	3×+MS AP^b	3×+MS AP^m
Swin-L	253M	20 / 26	51.8	44.9	53.9	46.7
ConvNeXt-L	255M	26 / 40	53.5	46.4	54.8	47.6
InternImage-L	277M	20 / 26	54.9	47.7	56.1	48.5
ConvNeXt-XL	407M	21 / 32	53.6	46.5	55.2	47.7
InternImage-XL	387M	16 / 23	55.3	48.1	56.2	48.8
FlashInternImage-L	277M	26 / 39	**55.6**	**48.2**	**56.7**	**48.9**

Table 5. **Object detection and instance segmentation performance on COCO val2017.** AP^b and AP^m denotes box AP and mask AP, respectively. "MS" means multi-scale training. FlashInternImage w/ DCNv4 models converge faster, clearly outperform other baselines with 1× training schedule, and still maintain a leading position when training 3× longer while being significantly faster than InternImage.

cluding instance segmentation, semantic segmentation and 3D object detection. We keep all implementation details the same as InternImage and only change the backbone model. The backbone models are initialized from the ImageNet pretrained weights when training the downstream models.

Instance Segmentation: We train FlashInternImage with two representative instance segmentation frameworks, Mask R-CNN [15] and Cascade Mask-RCNN [2], on COCO dataset [23] at 1× (12 epochs) and 3× (36 epochs) training schedules. The results are shown in Tab. 5. We also report FPS with batch size 16 in FP32/FP16 data format. FlashInternImage shows superior results on all model scales and training schedules, achieving a higher speed-accuracy tradeoff. For example, FlashInternImage-T/S surpasses all other models with the same scale and is on par with a larger InternImage-S/B while being $80\% - 90\%$ faster.

Semantic Segmentation: We train FlashInternImage with UperNet [41] on ADE20K [46] dataset for 160K iterations. We can draw a similar conclusion as instance segmentation from the results in Tab. 6, with FPS reported with batch size 16 in FP32/FP16. FlashInternImage w/ DCNv4 can achieve significantly faster speed and further improve the performance of InternImage across different model scales, resulting in a new state-of-the-art.

Model	crop size	#param	FPS	mIoU (SS)	mIoU (MS)
Swin-T	512^2	60M	107 / 168	44.5	45.8
ConvNeXt-T	512^2	60M	120 / 184	46.0	46.7
InternImage-T	512^2	59M	100 / 139	47.9	48.1
FlashInternImage-T	512^2	59M	119 / 206	**49.3**	**50.3**
Swin-S	512^2	81M	89 / 142	47.6	49.5
ConvNeXt-S	512^2	82M	107 / 164	48.7	49.6
InternImage-S	512^2	80M	89 / 123	50.1	50.9
FlashInternImage-S	512^2	80M	107 / 182	**50.6**	**51.6**
Swin-B	512^2	121M	77 / 126	48.1	49.7
ConvNeXt-B	512^2	122M	95 / 147	49.1	49.9
InternImage-B	512^2	128M	77 / 104	50.8	51.3
FlashInternImage-B	512^2	128M	94 / 157	**52.0**	**52.6**
Swin-L	640^2	234M	59 / 99	52.1	53.5
ConvNeXt-L	640^2	235M	73 / 117	53.2	53.7
InternImage-L	640^2	256M	56 / 78	53.9	54.1
ConvNeXt-XL	640^2	391M	53 / 75	53.6	54.0
InternImage-XL	640^2	368M	47 / 67	55.0	55.3
FlashInternImage-L	640^2	256M	71 / 122	**55.6**	**56.0**

Table 6. **Semantic segmentation performance on the ADE20K validation set.** All models are trained with UperNet. "SS" and "MS" denote single-scale and multi-scale testing, respectively. FPS is reported with single-scale testing. FlashInternImage w/ DCNv4 achieves superior performance with competitive speed.

Model	NDS	mAP	FPS†	FPS
InternImage-B	62.0	54.0	8.0	2.7
InternImage-XL	63.4	55.6	4.0	2.0
FlashInternImage-S	61.7	55.5	16.8	4.1
FlashInternImage-B	63.1	57.4	12.1	3.8

Table 7. **3D detection performance of BEVFormer v2 on nuScenes *test* set.** We report backbone FPS (denoted with †), overall FPS results with *underoptimized* head implementation are also added for reference. With on-par NDS and higher mAP results, FlashInternImage can be $50 - 90\%$ faster than InternImage baselines or $200\% - 300\%$ when only considering the backbone.

3D Detection: We further test DCNv4 on the camera-based 3D object detection task in the autonomous driving scenario, We train BEVFormer v2 [43], a state-of-the-art multi-camera 3D object detector, with FlashInternImage-Small and Base backbone models on the nuScenes dataset for 24 epochs. We report results on the nuScenes test set in Tab. 7 with FPS for each model. We note that the header parts, such as the BEV encoder and object decoder in BEVFormer v2, are *underoptimized* and take more than 50% of the running time (and even more with a faster backbone); thus, we also report the FPS for the backbone for a clearer illustration. Our results show that when only considering the backbone, FlashInternImage can be twice or even three times faster than the InternImage backbone with an on-par performance, greatly increasing the model efficiency.

4.4. DCNv4 as a Universal Operator

Drop-in replacement in other vision backbones : We verify whether DCNv4 can still work well in architectures de-

Model	IN-1K Acc	Throughput
ConvNeXt-B	83.8	1241 / 1888
ConvNeXt-B + DCNv4	84.0	1495 / 2513 (+20%/ + 33%)
ViT-B	81.8	1683 / 2781†
ViT-B + DCNv4	81.9	2092 / 3261 (+24%/ + 17%)

Table 8. **DCNv4 in other architecture.** We show supervised learning results on ImageNet-1K and throughput. DCNv4 achieves higher throughput with comparable accuracy. † denotes testing with the advanced FlashAttention-2 implementation.

Model	#param	FID	FPS
U-Net	860M	2.94	4.82
U-Net + DCNv4	566M	2.44	4.92

Table 9. **Class conditional generation on ImageNet 256x256.** Latent diffusion models are trained from scratch with U-Net. We replace the convolution layer in the models with DCNv4. DCNv4 can achieve better FID results without any hyperparameter tuning.

signed with other operators, such as ConvNeXt and ViT. To achieve that, we replace the attention module and depthwise convolution layer with DCNv4 in ViT and ConvNeXt and perform supervised learning on ImageNet-1K without changing any other architecture and hyperparameters, similar to FlashInternImage and InternImage. The results are shown in Tab. 8. We can see that on these architectures, which are carefully tuned for the specific operators, our DCNv4 can perform equally well. Thanks to the fast speed of DCNv4, the new model can even achieve better throughput, showcasing the superior performance of DCNv4.

Drop-in replacement in diffusion model: DCN has been recognized to be an effective operator for perception tasks. As generative models become a fundamental tool for AI-generated content (AIGC), we are also curious if DCNv4 can work well on generation tasks with diffusion-based generative models. Specifically, we choose the U-Net [30] used in Stable Diffusion [29] as our baselines and replace the attention module and regular 3×3 convolution in U-Net. We use U-ViT's codebase, follow its training schedule, and train a latent diffusion model based on the image latent extracted from an image autoencoder provided by Stable Diffusion. We show the results in Tab. 9. We can see that DCNv4 also works well in generative modeling, achieving better results in terms of FID/Throughput with fewer parameters compared to regular convolution in U-Net. Notice that the architecture/hyperparameters may not be optimal for DCNv4, and it is possible that re-designing the models or searching for new hyperparameters for DCNv4 will give better results.

4.5. Ablation Study

We conduct ablation studies in our optimization choice described in Sec. 3.2. The results are shown in Tab. 10. The time in the table is obtained with $56 \times 56 \times 128$ input with batch size 64 and 4 groups (32 channels per group). We first remove the softmax operation and improve the micro design, which means we merge the two linear layers into one and remove costly layer norm and GELU activation in offset/aggregation weight computing, simplifying the overall modules and increasing the speed. We then start modifying the kernel implementation. First, we change the parallel execution pattern and let each thread process 8 channels instead of 1 channel, thus, unnecessary memory access on loading sampling offset and aggregation weight values from the GPU memory can be saved. As we expected, solely applying this change will not increase the speed as the degree of parallelism decreases, and each thread's workload increases 8 times now. The latency is increased instead. Eliminating redundant computation by reusing the bilinear interpolation coefficient (4th row) can save some time but is insignificant. Removing the redundant memory instruction via vectorized load/store can greatly reduce the workload of each thread and largely accelerate the GPU kernel speed (5th row). Using a half-precision datatype, which halves the number of bytes the kernel needs to read/write, further increases the data throughput, as shown in the 6th row. In the end, we reach the final DCNv4 design, which is three times more efficient than the original implementation.

Implementation variant	Module	Kernel
Original DCNv3	3.28	1.45
- micro design	2.12	1.45
- redundant memory access	2.20	1.53
- redundant computation	2.18	1.51
- redundant memory instr.	1.28	0.606
- half-precision format	0.873	0.404

Table 10. **Ablation studies of DCN's runtime (ms).** We show how to achieve DCNv4 (gray row) from the original DCNv3 implementation and how different design choices affect the speed.

5. Conclusion

We present Deformable Convolution v4 (DCNv4), an efficient dynamic and sparse operator. By rethinking the dynamic property in deformable convolution and streamlining memory access, DCNv4 emerges as a much faster and more effective operator than its predecessor DCNv3. DCNv4-equipped FlashInternImage backbone not only enhances speed but also improves performance across various vision tasks. We further show DCNv4's versatility and effectiveness as a universal operator by integrating it into state-of-the-art architecture like ConvNeXt and ViT with improved throughput and accuracy; and it also works well in latent diffusion model, showing its potential to enhance generative models.

Acknowledgement

The work is supported by the National Key R&D Program of China (NO. 2022ZD0161300), by the National Natural Science Foundation of China (62376134).

References

[1] Saehyun Ahn, Jung-Woo Chang, and Suk-Ju Kang. An efficient accelerator design methodology for deformable convolutional networks. In *2020 IEEE International Conference on Image Processing (ICIP)*, pages 3075–3079. IEEE, 2020. 2

[2] Zhaowei Cai and Nuno Vasconcelos. Cascade r-cnn: Delving into high quality object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6154–6162, 2018. 7

[3] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *European conference on computer vision*, pages 213–229. Springer, 2020. 2

[4] 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. 2

[5] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. cudnn: Efficient primitives for deep learning. *arXiv preprint arXiv:1410.0759*, 2014. 5

[6] François Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1251–1258, 2017. 2, 3, 5

[7] Jifeng Dai, Haozhi Qi, Yuwen Xiong, Yi Li, Guodong Zhang, Han Hu, and Yichen Wei. Deformable convolutional networks. In *Proceedings of the IEEE international conference on computer vision*, pages 764–773, 2017. 2

[8] Tri Dao. Flashattention-2: Faster attention with better parallelism and work partitioning. *arXiv preprint arXiv:2307.08691*, 2023. 5

[9] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359, 2022. 2, 3

[10] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009. 2

[11] Xiaohan Ding, Xiangyu Zhang, Jungong Han, and Guiguang Ding. Scaling up your kernels to 31x31: Revisiting large kernel design in cnns. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11963–11975, 2022. 2

[12] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2020. 1, 2, 3

[13] Qi Han, Zejia Fan, Qi Dai, Lei Sun, Ming-Ming Cheng, Jiaying Liu, and Jingdong Wang. On the connection between local attention and dynamic depth-wise convolution. *arXiv preprint arXiv:2106.04263*, 2021. 2

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 2

[15] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, 2017. 1, 5, 7

[16] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 2012. 2

[17] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 2

[18] Youngwan Lee, Joong-won Hwang, Sangrok Lee, Yuseok Bae, and Jongyoul Park. An energy and gpu-computation efficient backbone network for real-time object detection. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*, pages 0–0, 2019. 2

[19] Feng Li, Hao Zhang, Huaizhe Xu, Shilong Liu, Lei Zhang, Lionel M Ni, and Heung-Yeung Shum. 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*, pages 3041–3050, 2023. 2

[20] Yanghao Li, Hanzi Mao, Ross Girshick, and Kaiming He. Exploring plain vision transformer backbones for object detection. In *European Conference on Computer Vision*, pages 280–296. Springer, 2022. 5

[21] Zhiqi Li, Wenhai Wang, Hongyang Li, Enze Xie, Chonghao Sima, Tong Lu, Yu Qiao, and Jifeng Dai. Bevformer: Learning bird's-eye-view representation from multi-camera images via spatiotemporal transformers. In *European conference on computer vision*, pages 1–18. Springer, 2022. 2

[22] Zhiqi Li, Wenhai Wang, Enze Xie, Zhiding Yu, Anima Anandkumar, Jose M Alvarez, Ping Luo, and Tong Lu. Panoptic segformer: Delving deeper into panoptic segmentation with transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1280–1289, 2022. 2

[23] 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 *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*, pages 740–755. Springer, 2014. 7

[24] Yingfei Liu, Tiancai Wang, Xiangyu Zhang, and Jian Sun. Petr: Position embedding transformation for multi-view 3d object detection. In *European Conference on Computer Vision*, pages 531–548. Springer, 2022. 2

[25] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 10012–10022, 2021. 1, 2, 3, 5, 6

[26] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11976–11986, 2022. 1, 2, 6

[27] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *ECCV*, 2018. 2, 3, 4

[28] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019. 5

[29] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10684–10695, 2022. 1, 2, 8

[30] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*, pages 234–241. Springer, 2015. 2, 8

[31] Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily L Denton, Kamyar Ghasemipour, Raphael Gontijo Lopes, Burcu Karagol Ayan, Tim Salimans, et al. Photorealistic text-to-image diffusion models with deep language understanding. *Advances in Neural Information Processing Systems*, 35:36479–36494, 2022. 1

[32] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 2

[33] Robin Strudel, Ricardo Garcia, Ivan Laptev, and Cordelia Schmid. Segmenter: Transformer for semantic segmentation. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 7262–7272, 2021. 2

[34] Hugues Thomas, Charles R Qi, Jean-Emmanuel Deschaud, Beatriz Marcotegui, François Goulette, and Leonidas J Guibas. Kpconv: Flexible and deformable convolution for point clouds. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 6411–6420, 2019. 2

[35] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017. 2, 5

[36] Ashish Vaswani, Prajit Ramachandran, Aravind Srinivas, Niki Parmar, Blake Hechtman, and Jonathon Shlens. Scaling local self-attention for parameter efficient visual backbones. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12894–12904, 2021. 2

[37] Shenlong Wang, Simon Suo, Wei-Chiu Ma, Andrei Pokrovsky, and Raquel Urtasun. Deep parametric continuous convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2589–2597, 2018. 2

[38] Wenhai Wang, Jifeng Dai, Zhe Chen, Zhenhang Huang, Zhiqi Li, Xizhou Zhu, Xiaowei Hu, Tong Lu, Lewei Lu, Hongsheng Li, et al. Internimage: Exploring large-scale vision foundation models with deformable convolutions. In *CVPR*, 2023. 1, 2, 6

[39] Ross Wightman. Pytorch image models. `https://github.com/rwightman/pytorch-image-models`, 2019. 6

[40] Felix Wu, Angela Fan, Alexei Baevski, Yann N Dauphin, and Michael Auli. Pay less attention with lightweight and dynamic convolutions. *arXiv preprint arXiv:1901.10430*, 2019. 2

[41] Tete Xiao, Yingcheng Liu, Bolei Zhou, Yuning Jiang, and Jian Sun. Unified perceptual parsing for scene understanding. In *Proceedings of the European conference on computer vision (ECCV)*, pages 418–434, 2018. 1, 7

[42] Yuwen Xiong, Mengye Ren, Renjie Liao, Kelvin Wong, and Raquel Urtasun. Deformable filter convolution for point cloud reasoning. *arXiv preprint arXiv:1907.13079*, 2019. 2

[43] Chenyu Yang, Yuntao Chen, Hao Tian, Chenxin Tao, Xizhou Zhu, Zhaoxiang Zhang, Gao Huang, Hongyang Li, Yu Qiao, Lewei Lu, et al. Bevformer v2: Adapting modern image backbones to bird's-eye-view recognition via perspective supervision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 17830–17839, 2023. 2, 7

[44] Xiaohua Zhai, Alexander Kolesnikov, Neil Houlsby, and Lucas Beyer. Scaling vision transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12104–12113, 2022. 1

[45] 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. 2

[46] Bolei Zhou, Hang Zhao, Xavier Puig, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Scene parsing through ade20k dataset. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 633–641, 2017. 7

[47] Xizhou Zhu, Han Hu, Stephen Lin, and Jifeng Dai. Deformable convnets v2: More deformable, better results. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019. 2

[48] Xizhou Zhu, Weijie Su, Lewei Lu, Bin Li, Xiaogang Wang, and Jifeng Dai. Deformable detr: Deformable transformers for end-to-end object detection. In *International Conference on Learning Representations*, 2020. 2, 5