Neighborhood Attention Transformer

Ali Hassani[1], Steven Walton[1], Jiachen Li[1], Shen Li[3], Humphrey Shi[1,2]
[1]SHI Labs @ U of Oregon & UIUC, [2]Picsart AI Research (PAIR), [3]Meta/Facebook AI
https://github.com/SHI-Labs/Neighborhood-Attention-Transformer

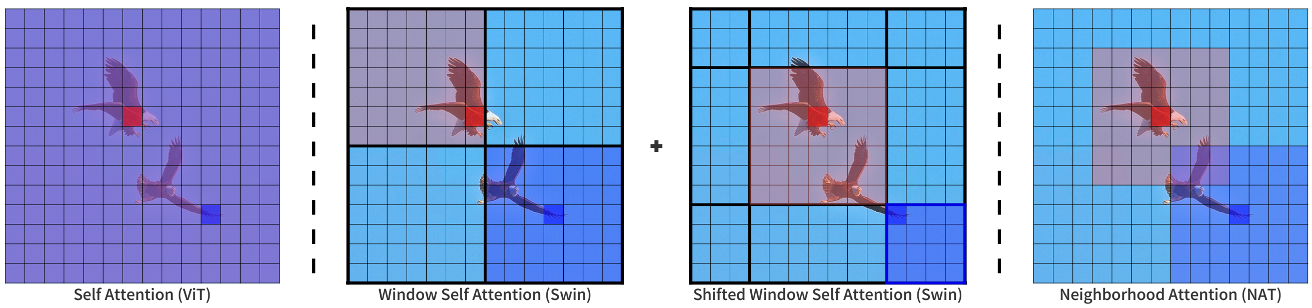

Figure 1. An illustration of attention spans in Self Attention, (Shifted) Window Self Attention, and our Neighborhood Attention. Self Attention allows each token to attend to everything. Window Self Attention divides self attention into non-overlapping sub-windows, and is followed by Shifted Window Self Attention, which allows for out-of-window interactions that are necessary to receptive field expansion. Neighborhood Attention localizes attention to a neighborhood around each token, introducing local inductive biases, maintaining translational equivariance, and allowing receptive field growth without needing extra operations.

Abstract

*We present **Neighborhood Attention (NA)**, the first efficient and scalable sliding window attention mechanism for vision. NA is a pixel-wise operation, localizing self attention (SA) to the nearest neighboring pixels, and therefore enjoys a linear time and space complexity compared to the quadratic complexity of SA. The sliding window pattern allows NA's receptive field to grow without needing extra pixel shifts, and preserves translational equivariance, unlike Swin Transformer's Window Self Attention (WSA). We develop $\mathcal{NATTEN}$ (Neighborhood Attention Extension), a Python package with efficient C++ and CUDA kernels, which allows NA to run up to 40% faster than Swin's WSA while using up to 25% less memory. We further present **Neighborhood Attention Transformer (NAT)**, a new hierarchical transformer design based on NA that boosts image classification and downstream vision performance. Experimental results on NAT are competitive; NAT-Tiny reaches 83.2% top-1 accuracy on ImageNet, 51.4% mAP on MS-COCO and 48.4% mIoU on ADE20K, which is 1.9% ImageNet accuracy, 1.0% COCO mAP, and 2.6% ADE20K mIoU improvement over a Swin model with similar size. To support more research based on sliding window attention, we open source our project and release our checkpoints.*

1. Introduction

Convolutional neural networks (CNNs) [19] have been the de facto standard architecture for computer vision models across different applications for years. AlexNet [18] showed their usefulness on ImageNet [10], and many others followed suit with architectures such as VGG [26], ResNet [17], and EfficientNet [27]. Transformers [31] on the other hand, were originally proposed as attention-based models for natural language processing (NLP), trying to exploit the sequential structure of language. They were the basis upon which BERT [11] and GPT [2, 23, 24] were built, and they continue to be the state of the art architecture in NLP.

In late 2020, Vision Transformer (ViT) [12] was proposed as an image classifier using only a Transformer Encoder operating on an embedded space of image patches, mostly for large-scale training. A number of other methods followed, attempting to increase data efficiency [13, 15, 28], eventually making such Transformer-like models the state of the art in ImageNet-1K classification (without pre-training on large-scale datasets such as JFT-300M).

These high-performing Transformer-like methods are all based on Self Attention (SA), the basic building block in the original Transformer [31]. SA has a linear complexity with respect to the embedding dimension (excluding linear projections), but a quadratic complexity with respect to the number of tokens. In the scope of vision, the number of tokens is typically in linear correlation with image resolution. As a result, higher image resolution results in a quadratic increase in complexity and memory usage in models strictly using SA, such as ViT. The quadratic complexity has prevented such models from being easily applicable to downstream vision tasks, such as object detection and segmentation, in which image resolutions are usually much larger than classification. Another problem is that convolutions benefit from inductive biases such as locality, and the 2-dimensional spatial structure, while dot-product self attention is a global 1-dimensional operation by definition. This means that some of those inductive biases have to be learned with either large sums of data [12] or advanced training techniques and augmentations [15, 28].

Local attention modules were therefore proposed to alleviate these issues. Stand-Alone Self-Attention (SASA) [25] was one of the earliest applications of local window-based attention to vision, where each pixel attends to a window around it. Its explicit sliding window pattern is identical to that of *same* convolutions, with zero paddings around and a simple 2-dimensional raster scan, therefore maintaining translational equivariance. SASA was aimed at replacing convolutions in a ResNet, and was shown to have a noticeable improvement over baselines. However, the authors noted SASA was limited in terms of speed due to the lack of an efficient implementation similar to that of convolutions. Swin [21] on the other hand was one of the first hierarchical vision transformers based on local self attention. Its design and the shifted-window self attention allowed it to be easily applicable to downstream tasks, as they made it computationally feasible, while also boosting performance through the additional biases injected. Swin's localized attention, however, first applies self attention to non-overlapping windows and then shifts the windows, the motivation of which was sliding window methods such as SASA suffering throughput bottlenecks. HaloNet [30] used a haloing mechanism that localizes self attention for blocks of pixels at a time, as opposed to pixel-wise. One of their key motivations for this was also noted to be the lack of an efficient sliding window attention.

In this work, we revisit explicit sliding window attention mechanisms, and propose Neighborhood Attention (NA). NA localizes SA to each pixel's nearest neighbors, which is not necessarily a fixed window around the pixel. This change in definition allows all pixels to maintain an identical attention span, which would otherwise be reduced for corner pixels in zero-padded alternatives (SASA). NA also approaches SA as its neighborhood size grows, and is equivalent to SA at maximum neighborhood. Additionally, NA has the added advantage of maintaining translational equivariance [30], unlike blocked and window self attention. We

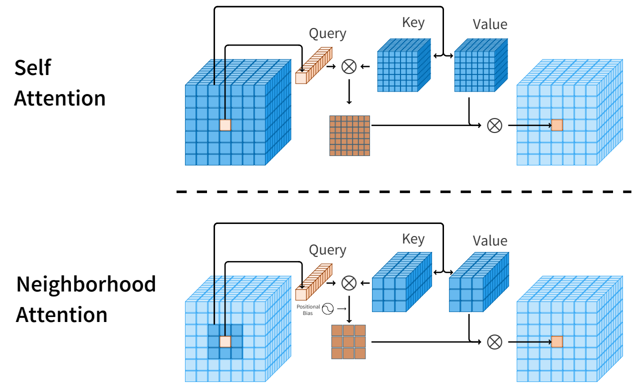

Figure 2. Illustration of the query-key-value structure of Neighborhood Attention (NA) vs Self Attention (SA) for a single pixel. SA allows each pixel to attend to every other pixel, whereas NA localizes attention for each pixel to a neighborhood around itself. Therefore, each pixel's attention span is usually different from the next.

develop $\mathcal{N}ATTEN$, a Python package with efficient C++ and CUDA kernels that allow NA to run even faster than Swin's WSA in practice, while using less memory. We build Neighborhood Attention Transformer (NAT), which achieves competitive results across vision tasks.

To summarize, our main contributions are:

1. Proposing **Neighborhood Attention (NA)**: A simple and flexible explicit sliding window attention mechanism that localizes each pixel's attention span to its nearest neighborhood, approaches self attention as its span grows, and maintains translational equivariance. We compare NA in terms of complexity and memory usage to self attention, window self attention, and convolutions.

2. Developing efficient C++ and CUDA kernels for NA, including the **tiled NA** algorithm, which allow NA to run up to 40% faster than Swin's WSA while using up to 25% less memory. We release them under a new Python package for explicit sliding window attention mechanisms, $\boldsymbol{\mathcal{N}}$**ATTEN**, to provide easy-to-use modules with autograd support that can be plugged into any existing PyTorch pipeline.

3. Introducing **Neighborhood Attention Transformer (NAT)**, a new efficient, accurate, and scalable hierarchical transformer based on NA. We demonstrate its effectiveness on both classification and downstream tasks. For instance, NAT-Tiny reaches 83.2% top-1 accuracy on ImageNet with only 4.3 GFLOPs and 28M parameters, and 51.4% box mAP on MS-COCO and 48.4% mIoU on ADE20K, significantly outperforming both Swin Transformer and ConvNeXt [22].

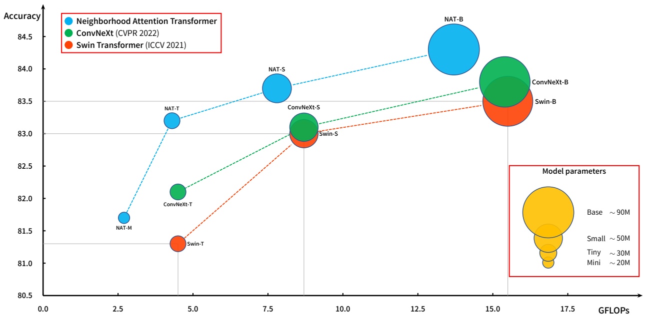

Figure 3. ImageNet-1K classification performance versus compute, with bubble size representing the number of parameters. NAT outperfoms both Swin Transformer and ConvNeXt in classification with fewer FLOPs, and a similar number of parameters.

2. Related Works

In this section, we briefly review the original Self Attention (SA) [31], some of the notable vision transformers and Transformer-like architectures [12, 28], some of the notable local attention-based vision transformers [21, 30], and a recent CNN which provides an up-to-date baseline for attention-based models.

2.1. Self Attention

Scaled dot-product attention was defined by Vaswani et al. [31] as an operation on a query and a set of key-value pairs. The dot product of query and key is computed and scaled. Softmax is applied to the output in order to normalize attention weights, and is then applied to the values. It can be expressed as follows:

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d}}\right)V, \quad (1)$$

where d is embedding dimension. Self attention applies dot-product attention over linear projections of the same input as both the query and key-value pairs. In Transformers, the multi-headed variants of attention and self attention are typically applied. Multi-headed attention applies dot-product attention multiple times over different embeddings, hence forming attention heads. Given an input $X \in \mathbb{R}^{n \times d}$, where n is the number of tokens and d is the embedding dimension, this operation has a complexity of $\mathcal{O}(n^2 d)$ and a space complexity of $\mathcal{O}(n^2)$ for the attention weights.

2.2. Vision Transformer

Dosovitskiy et al. [12] proposed a Transformer-based image classifier that merely consists of a Transformer encoder [31] and an image tokenizer, named **V**ision **T**ransformer (**ViT**). Previous works, such as DETR [4], explored CNN-Transformer hybrids for object detection. ViT on the other hand proposed a model that would only rely on a single non-overlapping convolutional layer (patching and embedding). ViT was pre-trained primarily on the private JFT-300M dataset, and was shown to outperform state-of-the-art CNNs on many benchmarks. However, it was also added that when ViT is pre-trained on medium-scale datasets, such as ImageNet-1K and ImageNet-21K, it no longer achieves competitive results. This was attributed to the lack of inductive biases that are inherent to CNNs, which the authors argued is trumped by large-scale training. While this effectively proved ViT inferior in medium-scale training, it provided empirical evidence that Transformer-based models outperform CNNs in larger scales. ViT paved the way for many more vision transformers, and attention-based models in general, that followed and transferred it to medium-scale learning [28], and even small-scale learning on much smaller datasets [15]. Touvron et al. [28] extended the study of Vision Transformers by exploring data efficiency. Their **D**ata-**e**fficient **i**mage **T**ransformer (**DeiT**) model pushed ViT ahead with minimal architectural changes, and through the use of advanced augmentations and training techniques. Their efforts highlighted the true potential of a Transformer-based image classifier in medium-sized data regimes, and inspired many more to adopt their training techniques [21, 29].

2.3. Local Attention

Stand Alone Self Attention (SASA) [25], is one of the earliest sliding window self attention patterns, aimed to replace convolutions in existing CNNs. It operates similarly to a convolution with zero padding, and extracts key-value pairs by striding the feature map. The authors reported a noticeable accuracy improvement, but observed that the implementation suffered high latency despite the lower theoretical cost. This attention pattern was also adopted in language processing in Longformer [1] (sliding window attention), and later adopted in Vision Longformer (ViL) [38]. While Longformer and ViL's implementations were different from SASA, they were still not able to scale to larger windows and models as a result of both computational overhead. Additionally, the reduced receptive field in corner cases caused by padding was not addressed. Window and **S**hifted **Win**dow (**Swin**) Attention [21] were introduced by Liu et al. as non-sliding window-based self attention mechanisms that partition feature maps and apply self attention to each partition separately. This operation has a similar theoretical complexity to SASA, but it can be easily parallelized through batched matrix multiplication. The shifted variant follows the regular, and as the name suggests shifts the partitioning to allow out-of-window interactions, which are necessary for receptive field growth. Their proposed model, **Swin Transformer**, is one of the earliest hierarchical vision transformers. It produces pyramid-like feature maps, reducing spatial dimensionality while increasing depth. This structure has been commonly used in CNNs over the years, and is why Swin can be easily integrated with other networks for application to downstream tasks, such as detection and segmentation. Swin outperformed DeiT, which uses a convolutional teacher, at ImageNet-1K classification. Moreover, Swin Transformer became the state-of-the-art method in object detection on MS-COCO and in semantic segmentation on ADE20K. Vaswani et al. [30] proposed **HaloNet**, which aimed to avoid SASA's speed issue by replacing it with a new blocked attention pattern. They noted that while this change relaxes translational equivariance, it can provide a reasonable trade-off with speed and memory. HaloNet's attention mechanism consists of 3 stages: blocking, haloing, and attention. Input feature maps are blocked into non-overlapping subsets, which will serve as queries. Followed by that, "haloed" neighboring blocks are extracted, which will serve as keys and values. Attention is then applied to the extracted queries and key-value pairs. HaloNet was shown to be effective at both reducing cost (compared to SASA) and improving performance, especially when used in conjunction with convolutional layers in the network. Many works followed Swin in adopting WSA, such as RegionViT [6], in which a regional token is inserted into every local self attention layer for the purpose of introducing global context. This work and HaloNet highlight that the research community has lost interest in sliding window attention patterns in part because they are thought to be inefficient. We aim to change that by introducing $\mathcal{N}ATTEN$.

2.4. New Convolutional Baselines

Liu et al. [22] proposed a new CNN architecture influenced by models such as Swin, dubbed ConvNeXt. These models are not attention-based, and manage to outperform Swin across different vision tasks. This work has since served as a new CNN baseline for fair comparison of convolutional models and attention-based models.

We propose Neighborhood Attention, which by design localizes the receptive field to a window around each query, and therefore would not require additional techniques such as the cyclic shift used by Swin. Additionally, Neighborhood Attention maintains translational equivariance, which is traded off for efficiency in methods such as HaloNet and Swin. We demonstrate that Neighborhood Attention can run even faster than methods such as Swin, while using less memory, with our $\mathcal{N}ATTEN$ python package. We introduce a hierarchical transformer-like model with this attention mechanism, dubbed Neighborhood Attention Transformer, and demonstrate its performance compared to Swin on image classification, object detection, and semantic segmentation.

3. Method

In this section, we introduce Neighborhood Attention, a localization of self attention (see Eq. (1)) considering the structure of visual data. This not only reduces computational cost compared to self attention, but also introduces local inductive biases, similar to that of convolutions. We show that NA is better alternative to the previously proposed SASA [25] in terms of restricting self attention, while being equivalent in theoretical cost. We then introduce our Python package, $\mathcal{N}ATTEN$, which provides efficient implementations of NA for both CPU and GPU acceleration. We discuss the novelties in the extension and how it manages to exceed the speed of Swin's WSA and SWSA, while using less memory. We finally introduce our model, **N**eighborhood **A**ttention **T**ransformer (**NAT**), which uses this new mechanism instead of self attention. In addition, NAT utilizes a multi-level hierarchical design, similar to Swin [21], meaning that feature maps are downsampled between levels, as opposed to all at once. Unlike Swin, NAT uses overlapping convolutions to downsample feature maps, as opposed to non-overlapping (patched) ones, which have been shown to improve model performance by introducing useful inductive biases [15, 34].

3.1. Neighborhood Attention

Swin's WSA can be considered one of the fastest existing methods to restrict self attention for the purpose of cutting down the quadratic attention cost. It simply partitions inputs and applies self attention to each partition separately. WSA requires to be paired with the shifted variant, SWSA, which shifts those partition lines to allow out-of-window interactions. This is crucial to expanding its receptive field. Nevertheless, the most direct way to restrict self attention locally, is to allow each pixel to attend to its neighboring pixels. This results in most pixels having a dynamically-shifted window around them, which expands receptive field, and would therefore not need a manual shifted variant. Additionally, different from Swin and similar to convolutions, such dynamic forms of restricted self attention can preserve translational equivariance [30] (we analyze translational equivariance in different methods including our own in Appendix C.) Inspired by this, we introduce **Neighborhood Attention (NA)**. Given an input $X \in \mathbb{R}^{n \times d}$, which is a matrix whose rows are d-dimensional token vectors, and X's linear projections, Q, K, and V, and relative positional biases $B(i, j)$, we define attention weights for the i-th input with neighborhood size k, $\mathbf{A}_i^k$, as the dot product of the i-th input's query projection, and its k nearest neighboring key projections:

$$\mathbf{A}_i^k = \begin{bmatrix} Q_i K_{\rho_1(i)}^T + B_{(i,\rho_1(i))} \\ Q_i K_{\rho_2(i)}^T + B_{(i,\rho_2(i))} \\ \vdots \\ Q_i K_{\rho_k(i)}^T + B_{(i,\rho_k(i))} \end{bmatrix}, \tag{2}$$

where $\rho_j(i)$ denotes i's j-th nearest neighbor. We then define neighboring values, $\mathbf{V}_i^k$, as a matrix whose rows are the i-th input's k nearest neighboring value projections:

$$\mathbf{V}_i^k = \begin{bmatrix} V_{\rho_1(i)}^T & V_{\rho_2(i)}^T & \dots & V_{\rho_k(i)}^T \end{bmatrix}^T. \tag{3}$$

Neighborhood Attention for the i-th token with neighborhood size k is then defined as:

$$\text{NA}_k(i) = softmax\left(\frac{\mathbf{A}_i^k}{\sqrt{d}}\right)\mathbf{V}_i^k, \tag{4}$$

where $\sqrt{d}$ is the scaling parameter. This operation is repeated for every pixel in the feature map. Illustrations of this operation are presented in Figs. 2 and VIII.

From this definition, it is easy to see that as k grows, $\mathbf{A}_i^k$ approaches self attention weights, and $\mathbf{V}_i^k$ approaches V_i itself, therefore Neighborhood Attention approaches self attention. This is the key difference between NA and SASA [25], where each pixel attends to a window around it with padding around the input to handle edge cases. It is thanks to this difference that NA approaches self attention as window size grows, which does not hold true in SASA, due to the zero padding around the input.

3.2. Tiled NA and $\mathcal{N}$ATTEN

Restricting self attention in a pixel-wise manner has not been well-explored in the past, primarily because it was considered a costly operation [21, 25, 30] that would require lower-level reimplementation. That is because self attention itself is broken down to matrix multiplication, an operation that is easily parallelizable on accelerators, and has a myriad of efficient algorithms defined for different use cases in computational software (to name a few: LAPACK, cuBLAS, CUTLASS). Additionally, most deep learning platforms, such as PyTorch, are written on top of such software, and additional packages (such as cuDNN). This is very helpful to researchers, as it allows them to use abstractions of operations such as matrix multiplication or convolutions, while the backend decides which algorithm to run based on their hardware, software, and use case. They also typically handle automatic gradient computation, which makes designing and training deep neural networks very straightforward. Because of the pixel-wise structure of NA (and other pixel-wise attention mechanisms, such as SASA [25]), and also the novelty of the definition of neighborhoods in NA, the only way to implement NA with these

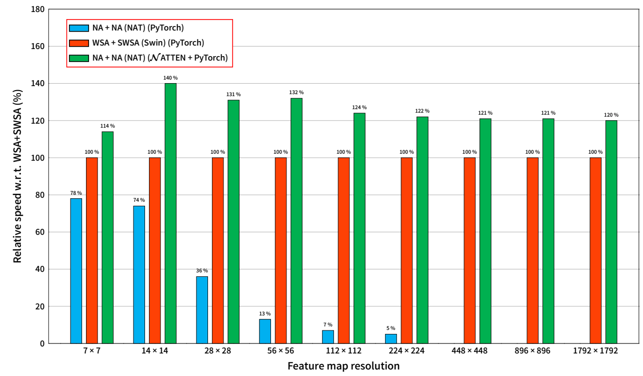

Figure 4. NAT's layer-wise relative speed with respect to Swin. Two NA layers with kernel size 7^2, are up to 40% faster than a pair of WSA and SWSA layers with the same kernel size. Latency is measured on a single A100 GPU. PyTorch implementation of NA runs out of memory at resolutions 448^2 and higher.

platforms is to stack a number of highly inefficient operations to extract the neighborhoods, store them as an intermediary tensor, and then compute attention. This results in a significantly slow operation, with an exponentially growing memory usage. To tackle these challenges, we developed a set of efficient CPU and CUDA kernels and packaged them as a Python package, **Neighborhood Attention Extension ($\mathcal{N}$ATTEN)**. $\mathcal{N}ATTEN$ includes half precision support, support for both 1D and 2D data, and autograd-compatible integration with PyTorch. This means that users can simply import NA as a PyTorch module and integrate it into existing pipelines. We also add that SASA can also be easily implemented with this package with no change in the underlying kernels (simply by padding inputs with zeros), as it is a special case of NA. The reverse does not hold true. It also includes our **tiled NA algorithm**, which computes neighborhood attention weights by loading non-overlapping query tiles into shared memory to minimize global memory reads. Compared to the naive implementation, tiled NA can decrease latency up to an order of magnitude (see Appendix A for technical details), and it allows NA-based models to run up to 40% faster than similar Swin counterparts (see Fig. 4.) $\mathcal{N}ATTEN$ is open-sourced at: `https://github.com/SHI-Labs/NATTEN`.

3.3. Neighborhood Attention Transformer

NAT embeds inputs using 2 consecutive 3×3 convolutions with 2×2 strides, resulting in a spatial size $1/4$th the size of the input. This is similar to using a patch and embedding layer with 4×4 patches, but it utilizes overlapping convolutions instead of non-overlapping ones to introduce useful inductive biases [15, 34]. On the other hand, using overlapping convolutions would increase cost, and two convolutions incurs more parameters. However, we handle that by re-configuring the model, which results in a better

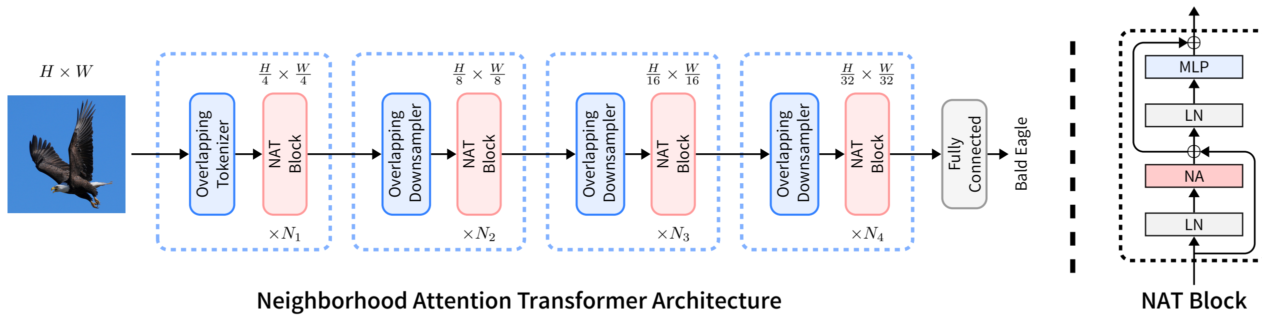

Figure 5. An overview of our model, NAT, with its hierarchical design. The model starts off with a convolutional downsampler, then moves on to 4 sequential levels, each consisting of multiple NAT Blocks, which are transformer-like encoder layers. Each layer is comprised of a multi-headed neighborhood attention (NA), a multi-layered perceptron (MLP), Layer Norm (LN) before each module, and skip connections. Between the levels, feature maps are downsampled to half their spatial size, while their depth is doubled. This allows for easier transfer to downstream tasks through feature pyramids.

Variant	Layers	Dim × Heads	MLP ratio	# of Params	FLOPs
∘ **NAT-Mini**	3, 4, 6, 5	32 × 2	3	20 M	2.7 G
∘ **NAT-Tiny**	3, 4, 18, 5	32 × 2	3	28 M	4.3 G
∘ **NAT-Small**	3, 4, 18, 5	32 × 3	2	51 M	7.8 G
∘ **NAT-Base**	3, 4, 18, 5	32 × 4	2	90 M	13.7 G

Table 1. Comparison of NAT Variants.

trade-off. NAT consists of 4 levels, each followed by a downsampler (except the last). Downsamplers cut spatial size in half, while doubling the number of channels. We use 3×3 convolutions with 2×2 strides, instead of 2×2 non-overlapping convolutions that Swin uses (patch merge). Since the tokenizer downsamples by a factor of 4, our model produces feature maps of sizes $\frac{h}{4} \times \frac{w}{4}$, $\frac{h}{8} \times \frac{w}{8}$, $\frac{h}{16} \times \frac{w}{16}$, and $\frac{h}{32} \times \frac{w}{32}$. This change is motivated by previous successful CNN structures, and followed by other hierarchical attention-based methods, such as PVT [32], ViL [38], and Swin Transformer [21]. Additionally, we use LayerScale [29] for stability in larger variants. An illustration of the overall network architecture is presented in Fig. 5. We present a summary of different NAT variants in Tab. 1.

3.4. Complexity Analysis

We present a complexity and memory usage analysis in this subsection, which compares SA, WSA, NA, and convolutions in Tab. 2. For simplicity, we exclude attention heads. Given input feature maps of shape $h \times w \times d$, where d is the number of channels, and h and w are feature map height and width respectively, the QKV linear projections are $3hwd^2$ FLOPs, which is the same for all three attention patterns. SA has a quadratic complexity, as both computing attention weights and output are h^2w^2d FLOPs, and attention weights are of shape $hw \times hw$. Swin's WSA divides the queries, keys, and values into $\frac{h}{k} \times \frac{w}{k}$ windows of shape $k \times k$, then applies self attention with each window, which is $hwdk^2$ FLOPs. WSA's memory consumption, given that its attention weights are of shape $\frac{h}{k} \times \frac{w}{k} \times k^2 \times k^2$, is therefore $hwdk^2$. In NA, $\mathbf{A}_i^k$ is of size $h \times w \times k^2$, and the cost to compute it is $hwdk^2$. $\mathbf{V}_i^k$ is of shape $h \times w \times k^2 \times d$, and therefore the cost of applying attention weights to it would be $hwdk^2$. As for convolutions, computational cost is hwd^2k^2, and memory usage would be only d^2k^2. The summary in Tab. 2 clarifies that Swin's WSA and NA have identical computational cost and memory usage in theory.

Module	FLOPs	Memory
∘ **Self Attn (SA)**	$3hwd^2 + 2h^2w^2d$	$3d^2 + h^2w^2$
∘ **Window Self Attn (WSA)**	$3hwd^2 + 2hwdk^2$	$3d^2 + hwk^2$
∘ **Neighborhood Attn (NA)**	$3hwd^2 + 2hwdk^2$	$3d^2 + hwk^2$
• **Convolution**	hwd^2k^2	d^2k^2

Table 2. Computational cost and memory usage in different attention patterns and convolutions. SA has a quadratic complexity with respect to resolution, while WSA, NA, and convolutions have a linear complexity.

4. Experiments

We demonstrate NAT's applicability and effectiveness by conducting experiments across different vision tasks, such as image classification, object detection, and semantic segmentation. We also present ablations on different attention patterns, as well as our NAT design. Additional experiments, including saliency analysis can be found in Appendix B.

4.1. Classification

We trained our variants on ImageNet-1K [10] in order to compare to other transformer-based and convolutional image classifiers. This dataset continues to be one

Model	# of Params	FLOPs	Thru. (imgs/sec)	Memory (GB)	Top-1 (%)
○ **NAT-M**	20 M	2.7 G	2135	2.4	81.8
○ **Swin-T**	28 M	4.5 G	1730	4.8	81.3
● **ConvNeXt-T**	28 M	4.5 G	2491	3.4	82.1
○ **NAT-T**	28 M	4.3 G	1541	2.5	**83.2**
○ **Swin-S**	50 M	8.7 G	1059	5.0	83.0
● **ConvNeXt-S**	50 M	8.7 G	1549	3.5	83.1
○ **NAT-S**	51 M	7.8 G	1051	3.7	**83.7**
○ **Swin-B**	88 M	15.4 G	776	6.7	83.5
● **ConvNeXt-B**	89 M	15.4 G	1107	4.8	83.8
○ **NAT-B**	90 M	13.7 G	783	5.0	**84.3**

Table 3. ImageNet-1K classification performance. All models run at 224×224 resolution with no extra data or pre-training. Peak memory usage and throughput are measured with a batch size of 256 on a single NVIDIA A100 GPU.

of the few benchmarks for medium-scale image classification, containing roughly 1.28M training, 50K validation, and 100K test images, categorized into 1000 classes. We train NAT with the commonly used `timm` [33] (Apache License v2), and use the conventional augmentations (CutMix [36], Mixup [37], RandAugment [8], and Random Erasing [39]) and training techniques used in methods we compare to [21, 22]. We follow Swin's [21] training configuration (learning rate, iteration-wise cosine schedule, and other hyperparameters). Following convention, we train for 300 epochs, 20 of which warm up the learning rate, while the rest decay according to the scheduler, and do 10 additional cooldown epochs [28]. Results are presented in Tab. 3, and visualized in Fig. 3. We observe that NAT-Mini outperforms Swin-Tiny by a margin of 0.5%, with fewer parameters, higher throughput and lower memory usage. As for the other three variants, we observe they consistently outperform both Swin and ConvNeXt counterparts with similar number of parameters and FLOPs. While our Small variant is slightly slower than its Swin counterpart due to the difference in architecture, our Base variant catches up to being faster than Swin-Base.

4.2. Object Detection and Instance Segmentation

We trained Mask [16] and Cascade Mask R-CNN [3] on MS-COCO [20], with NAT backbones, which were pre-trained on ImageNet. We followed Swin [21]'s training settings, using `mmdetection` [5] (Apache License v2), and trained with the same accelerated 3× LR schedule. Results are presented in Tab. 4. NAT-Mini outperforms Swin-Tiny with Mask R-CNN, while falling slightly short to it with Cascade Mask R-CNN, all while having significantly fewer FLOPs. NAT-Tiny outperforms both its Swin and ConvNeXt counterparts, with both Mask and Cascade Mask R-CNN, while having a slightly lower throughput compared to its Swin counterpart. NAT-Small reaches a competitive

Backbone	# of Params	FLOPs	Thru. (FPS)	AP^b	AP^b_{50}	AP^b_{75}	AP^m	AP^m_{50}	AP^m_{75}
Mask R-CNN - 3x schedule									
○ **NAT-M**	40 M	225 G	54.1	46.5	68.1	51.3	41.7	65.2	44.7
○ **Swin-T**	48 M	267 G	45.1	46.0	68.1	50.3	41.6	65.1	44.9
● **ConvNeXt-T**	48 M	262 G	52.0	46.2	67.0	50.8	41.7	65.0	44.9
○ **NAT-T**	48 M	258 G	44.5	47.7	69.0	52.6	42.6	66.1	45.9
○ **Swin-S**	69 M	359 G	31.7	48.5	70.2	53.5	43.3	67.3	46.6
○ **NAT-S**	70 M	330 G	34.8	48.4	69.8	53.2	43.2	66.9	46.5
Cascade Mask R-CNN - 3x schedule									
○ **NAT-M**	77 M	704 G	27.8	50.3	68.9	54.9	43.6	66.4	47.2
○ **Swin-T**	86 M	745 G	25.1	50.4	69.2	54.7	43.7	66.6	47.3
● **ConvNeXt-T**	86 M	741 G	27.3	50.4	69.1	54.8	43.7	66.5	47.3
○ **NAT-T**	85 M	737 G	24.9	51.4	70.0	55.9	44.5	67.6	47.9
○ **Swin-S**	107 M	838 G	20.3	51.9	70.7	56.3	45.0	68.2	48.8
● **ConvNeXt-S**	108 M	827 G	23.0	51.9	70.8	56.5	45.0	68.4	49.1
○ **NAT-S**	108 M	809 G	21.7	52.0	70.4	56.3	44.9	68.1	48.6
○ **Swin-B**	145 M	982 G	17.3	51.9	70.5	56.4	45.0	68.1	48.9
● **ConvNeXt-B**	146 M	964 G	19.5	52.7	71.3	57.2	45.6	68.9	49.5
○ **NAT-B**	147 M	931 G	18.6	52.5	71.1	57.1	45.2	68.6	49.0

Table 4. COCO object detection and instance segmentation performance. FLOPS are with respect to an input resolution of (1280, 800). Throughput is measured at the same resolution on a single NVIDIA A100 GPU.

Backbone	# of Params	FLOPs	Thru. (FPS)	mIoU single scale	mIoU multi scale
○ **NAT-M**	50 M	900 G	24.5	45.1	46.4
○ **Swin-T**	60 M	946 G	21.3	44.5	45.8
● **ConvNeXt-T**	60 M	939 G	23.3	46.0	46.7
○ **NAT-T**	58 M	934 G	21.4	47.1	48.4
○ **Swin-S**	81 M	1040 G	17.0	47.6	49.5
● **ConvNeXt-S**	82 M	1027 G	19.1	48.7	49.6
○ **NAT-S**	82 M	1010 G	17.9	48.0	49.5
○ **Swin-B**	121 M	1188 G	14.6	48.1	49.7
● **ConvNeXt-B**	122 M	1170 G	16.4	49.1	49.9
○ **NAT-B**	123 M	1137 G	15.6	48.5	49.7

Table 5. ADE20K semantic segmentation performance. FLOPS are with respect to an input resolution of (2048, 512). Throughput is measured at the same resolution on a single NVIDIA A100 GPU.

performance compared to its Swin counterpart, while being faster. NAT-Base can even outperform its Swin counterpart, while also enjoying a higher throughput.

4.3. Semantic Segmentation

To demonstrate NAT's performance on semantic segmentation, we trained UPerNet [35] with NAT backbones on ADE20K [40]. We followed Swin's configuration for training ADE20K, and used `mmsegmentation` [7] (Apache License v2). Additionally, and following standard practice, input images are randomly resized and cropped at 512 × 512 when training. Results are presented in Tab. 5. It is noticeable that NAT-Mini outperforms Swin-Tiny, and also comes very close to ConvNeXt-Tiny. NAT-Tiny outperforms ConvNeXt-Tiny significantly, while also slightly more efficient. NAT-Small outperforms Swin-Small on single-scale performance, while matching the multi-scale performance. NAT-Base similarly performs on-par with Swin-Base, while falling slightly short of ConvNeXt-Base. Note that both NAT-Small and NAT-Base bear fewer FLOPs with them compared to their Swin and ConvNeXt counterparts, while their performance is within the same region. It is also noteworthy that Swin especially suffers from more FLOPs even beyond the original difference due to the fact that the image resolution input in this task specifically (512×512) will not result in feature maps that are divisible by 7×7, Swin's window size, which forces the model to pad input feature maps with zeros to resolve that issue, prior to every attention operation. NAT on the other hand supports feature maps of arbitrary size.

4.4. Ablation Study

We compare Swin's attention pattern (WSA+SASA) to sliding window patterns, namely SASA [25] (implemented with our $\mathcal{N}ATTEN$ package and therefore enjoys approximately the same throughput and identical memory usage as NA), and our NA. We simply replace the attention blocks in Swin-Tiny, and run the model on ImageNet-1K classification, MSCOCO object detection and instance segmentation, and ADE20K segmentation. Results are presented in Tab. 6.

Separately, we investigate the effects of our NAT design (convolutional downsampling and deeper-thinner architecture), by performing an ablation with Swin-Tiny as baseline. We slowly transform the model into NAT-Tiny, and present the results in Tab. 7. We start by replacing the patched embedding and patched merge with the overlapping convolution design used in NAT. This results in almost 0.5% improvement in accuracy. After taking the second step to reduce the model size and compute, by making it deeper and thinner, we notice the model sees approximately an improvement in accuracy of 0.9% over the first step. We then try swapping the WSA and SWSA attention patterns in Swin with SASA [25], and see a slight drop in accuracy. However, swapping WSA and SWSA with our NA shows a further 0.5% improvement in accuracy.

We also present a kernel size experiment in Tab. 8, in which we try kernel sizes ranging from 3× 3 to 9 × 9, in an effort to analyze its affect on our model's performance.

5. Conclusion

In this paper, we present Neighborhood Attention (NA), the first efficient and scalable sliding window attention mechanism for vision. NA is a pixel-wise operation which localizes self attention for each pixel to its nearest neighborhood, and therefore enjoys linear complexity. It also introduces local inductive biases and maintains translational equivariance, unlike blocked (HaloNet) and window self attention (Swin). Different from SASA, NA approaches self attention as its window size grows, and does not limit attention span at corner cases. We challenge the common notion that explicit sliding window attention patterns are not efficient or parallelizable [21] by developing $\mathcal{N}ATTEN$. Through using $\mathcal{N}ATTEN$, NA-based models can run even faster than existing alternatives, despite the latter running primarily on highly optimized deep learning libraries built on top of lower-level computational packages. We also propose Neighborhood Attention Transformer (NAT) and show the power of such models: NAT outperforms both Swin Transformer and ConvNeXt in image classification, and outperforms or competes with both in downstream vision tasks. We open-source our entire project to encourage more research in this direction.

Acknowledgments. This work was in part supported by the Intelligence Advanced Research Projects Activity (IARPA) under Contract No. 2022-21102100004. We thank Picsart AI Research (PAIR) and Meta for their generous support that made this work possible.

Attention	ImageNet Top-1	MSCOCO AP^B	MSCOCO AP^m	ADE20K mIoU	# of Params	FLOPs	Thru. (imgs/sec)	Memory (GB)
○ SWSA	81.3%	46.0	41.6	45.8	28.28 M	4.51 G	1730	4.8
○ SASA	81.6%	46.0	41.4	46.4	28.27 M	4.51 G	2021	4.0
○ NA	81.8%	46.2	41.5	46.4	28.28 M	4.51 G	2021	4.0

Table 6. Performance comparison of different attention mechanisms. All models are presented are identical in architecture to Swin-T, with only the attention mechanisms replaced. SASA is implemented with our $\mathcal{N}ATTEN$, and therefore enjoys the same speed and memory efficiency as NA.

Attention	Down--sampler	# of Layers	# of Heads	MLP Ratio	Top-1 (%)	# of Params	FLOPs (G)	Thru. (imgs/sec)	Memory (GB)
○ SWSA	Patch	2, 2, 6, 2	3	4	81.29	28.3 M	4.5	1730	4.8
○ SWSA	Conv	2, 2, 6, 2	3	4	81.78	30.3 M	4.9	1692	4.8
○ SWSA	Conv	3, 4, 18, 5	2	3	82.72	27.9 M	4.3	1320	3.0
○ SASA	Conv	3, 4, 18, 5	2	3	82.54	27.9 M	4.3	1541	2.5
○ NA	Conv	3, 4, 18, 5	2	3	83.20	27.9 M	4.3	1541	2.5

Table 7. Ablation study on NAT, with Swin-T as the baseline. Through overlapping convolutions (row 2), and our NAT configuration (row 3), we boost Swin classification accuracy (row 1) significantly. Swapping SWSA with NA (row 5) results in an improvement of almost 0.5% in accuracy, while swapping it with SASA (row 4) results in a slight decrease in accuracy. SASA is implemented with our $\mathcal{N}ATTEN$, and therefore enjoys the same speed and memory efficiency as NA.

Kernel size	ImageNet Top-1 (%)	ImageNet Thru.	MSCOCO AP^b	MSCOCO AP^m	MSCOCO Thru.	ADE20K mIoU	ADE20K Thru.
3×3	81.4	2015 imgs/sec	46.1	41.4	46.8 fps	46.0	23.6 fps
5×5	81.6	1810 imgs/sec	46.8	42.0	45.5 fps	46.3	22.9 fps
7×7	83.2	1537 imgs/sec	47.7	42.6	44.5 fps	48.4	21.4 fps
9×9	83.1	1253 imgs/sec	48.5	43.3	39.4 fps	48.1	20.2 fps

Table 8. NAT-Tiny performance with different kernel sizes.

References

[1] Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020. 3

[2] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020. 1

[3] Zhaowei Cai and Nuno Vasconcelos. Cascade r-cnn: Delving into high quality object detection. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. 7

[4] 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 (ECCV)*, 2020. 3

[5] Kai Chen, Jiaqi Wang, Jiangmiao Pang, Yuhang Cao, Yu Xiong, Xiaoxiao Li, Shuyang Sun, Wansen Feng, Ziwei Liu, Jiarui Xu, et al. Mmdetection: Open mmlab detection toolbox and benchmark. *arXiv preprint arXiv:1906.07155*, 2019. 7

[6] Richard Chen, Rameswar Panda, and Quanfu Fan. Regionvit: Regional-to-local attention for vision transformers. In *International Conference on Learning Representations (ICLR)*, 2022. 4

[7] MMSegmentation Contributors. MMSegmentation: Openmmlab semantic segmentation toolbox and benchmark. `https://github.com/open-mmlab/mmsegmentation`, 2020. 7

[8] Ekin D Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V Le. Randaugment: Practical automated data augmentation with a reduced search space. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, 2020. 7

[9] Jifeng Dai, Haozhi Qi, Yuwen Xiong, Yi Li, Guodong Zhang, Han Hu, and Yichen Wei. Deformable convolutional networks. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2017. 11

[10] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009. 1, 6

[11] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 2019. 1

[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 (ICLR)*, 2020. 1, 2, 3

[13] Alaaeldin El-Nouby, Hugo Touvron, Mathilde Caron, Piotr Bojanowski, Matthijs Douze, Armand Joulin, Ivan Laptev, Natalia Neverova, Gabriel Synnaeve, Jakob Verbeek, et al. Xcit: Cross-covariance image transformers. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021. 1

[14] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`. 14

[15] Ali Hassani, Steven Walton, Nikhil Shah, Abulikemu Abuduweili, Jiachen Li, and Humphrey Shi. Escaping the big data paradigm with compact transformers. *arXiv preprint arXiv:2104.05704*, 2021. 1, 2, 3, 4, 5

[16] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2017. 7

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 1

[18] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2012. 1

[19] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4), 1989. 1

[20] 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 *European Conference on Computer Vision (ECCV)*, 2014. 7

[21] 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 *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021. 2, 3, 4, 5, 6, 7, 8

[22] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s. *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022. 2, 4, 7

[23] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training, 2018. 1

[24] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners, 2019. 1

[25] Prajit Ramachandran, Niki Parmar, Ashish Vaswani, Irwan Bello, Anselm Levskaya, and Jon Shlens. Stand-alone self-attention in vision models. *Advances in Neural Information Processing Systems (NeurIPS)*, 2019. 2, 3, 4, 5, 8, 14

[26] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 1

[27] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning (ICML)*, 2019. 1

[28] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training

data-efficient image transformers & distillation through attention. In *International Conference on Machine Learning (ICML)*, 2021. 1, 2, 3, 7

[29] Hugo Touvron, Matthieu Cord, Alexandre Sablayrolles, Gabriel Synnaeve, and Hervé Jégou. Going deeper with image transformers. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021. 3, 6

[30] Ashish Vaswani, Prajit Ramachandran, Aravind Srinivas, Niki Parmar, Blake Hechtman, and Jonathon Shlens. Scaling local self-attention for parameter efficient visual backbones. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021. 2, 3, 4, 5, 14

[31] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017. 1, 3

[32] Wenhai Wang, Enze Xie, Xiang Li, Deng-Ping Fan, Kaitao Song, Ding Liang, Tong Lu, Ping Luo, and Ling Shao. Pyramid vision transformer: A versatile backbone for dense prediction without convolutions. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021. 6

[33] Ross Wightman. Pytorch image models. https://github.com/rwightman/pytorch-image-models, 2019. 7

[34] Haiping Wu, Bin Xiao, Noel Codella, Mengchen Liu, Xiyang Dai, Lu Yuan, and Lei Zhang. Cvt: Introducing convolutions to vision transformers. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021. 4, 5

[35] Tete Xiao, Yingcheng Liu, Bolei Zhou, Yuning Jiang, and Jian Sun. Unified perceptual parsing for scene understanding. In *European Conference on Computer Vision (ECCV)*, 2018. 7

[36] Sangdoo Yun, Dongyoon Han, Seong Joon Oh, Sanghyuk Chun, Junsuk Choe, and Youngjoon Yoo. Cutmix: Regularization strategy to train strong classifiers with localizable features. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019. 7

[37] Hongyi Zhang, Moustapha Cisse, Yann N. Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. In *International Conference on Learning Representations (ICLR)*, 2018. 7

[38] Pengchuan Zhang, Xiyang Dai, Jianwei Yang, Bin Xiao, Lu Yuan, Lei Zhang, and Jianfeng Gao. Multi-scale vision longformer: A new vision transformer for high-resolution image encoding. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021. 3, 6

[39] Zhun Zhong, Liang Zheng, Guoliang Kang, Shaozi Li, and Yi Yang. Random erasing data augmentation. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2020. 7

[40] Bolei Zhou, Hang Zhao, Xavier Puig, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Scene parsing through ade20k dataset. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 7

[41] 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 (ICLR)*, 2020. 11