Hydra Attention: Efficient Attention with Many Heads

Daniel Bolya[1,2(✉)], Cheng-Yang Fu[2], Xiaoliang Dai[2], Peizhao Zhang[2], and Judy Hoffman[1]

[1] Georgia Tech, Atlanta, USA
{dbolya,judy}@gatech.edu
[2] Meta AI, Menlo Park, USA
{chengyangfu,xiaoliangdai,stzpz}@fb.com

Abstract. While transformers have begun to dominate many tasks in vision, applying them to large images is still computationally difficult. A large reason for this is that self-attention scales quadratically with the number of tokens, which in turn, scales quadratically with the image size. On larger images (e.g., 1080p), over 60% of the total computation in the network is spent solely on creating and applying attention matrices. We take a step toward solving this issue by introducing Hydra Attention, an extremely efficient attention operation for Vision Transformers (ViTs). Paradoxically, this efficiency comes from taking multi-head attention to its extreme: by using as many attention heads as there are features, Hydra Attention is computationally linear in both tokens and features with no hidden constants, making it significantly faster than standard self-attention in an off-the-shelf ViT-B/16 by a factor of the token count. Moreover, Hydra Attention retains high accuracy on ImageNet and, in some cases, actually *improves* it.

Keywords: Vision Transformers · Attention · Token efficiency

1 Introduction

Because of their generality and high capacity to learn from large amounts of data, transformers [32] have been a dominant force in natural language processing (NLP) for the last couple of years [6,17,25]. And now, with the introduction of Vision Transformers (ViTs) [10], the same takeover is happening in vision.

Yet, unlike in NLP, the pure instantiation of transformers that can be seen in NLP with BERT [17] or in vision with ViT [10] are not the force dominating computer vision tasks. Instead, much more vision-specialized attention-based

D. Bolya—This work was done under an internship at Meta AI.

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1007/978-3-031-25082-8_3.

L. Karlinsky et al. (Eds.): ECCV 2022 Workshops, LNCS 13807, pp. 35–49, 2023.
https://doi.org/10.1007/978-3-031-25082-8_3

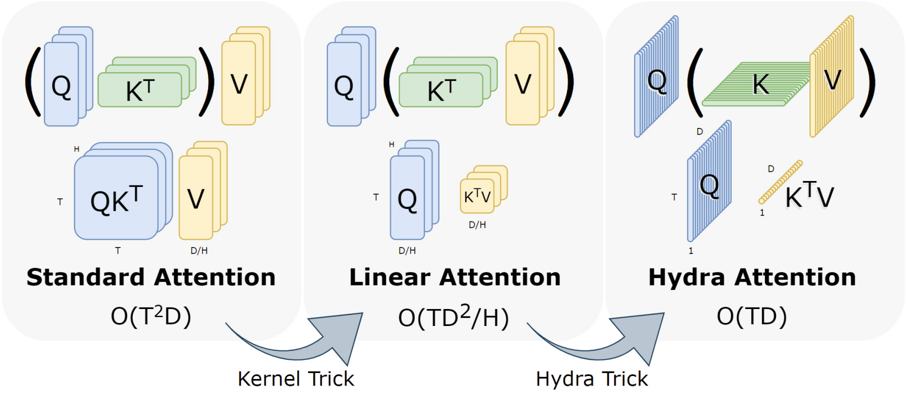

Fig. 1. Hydra Attention. Standard attention [32] scales with the square of the number of tokens T. Using a decomposable kernel, we can rearrange the order of operations as in [16] such that attention scales with the square of features D instead. Our Hydra attention goes one step further by maximizing the number of attention heads H, resulting in an $O(TD)$ operation in both space and time.

architectures such as Swin [21] or MViT [11,20] or attention-conv mixtures like LeViT [13] are being used instead.

The primary reason behind this discrepancy is efficiency: specialized vision transformers can perform better with less compute—either by adding conv layers, by using vision-specific local window attention, or by using some other way to cheaply add visual inductive bias. While pure ViTs can perform well at scale (90.45% top-1 on ImageNet [38]), the primary mechanism of a pure transformer—multihead self-attention [32]—can be an extreme bottleneck when applying a model on the large images required by several downstream tasks.

In fact, when applying an off-the-shelf ViT on 1080p images, common for benchmark tasks such as segmentation (e.g., CityScapes [7]), 60% of the total computation in the network (see Table 4) is spent simply on creating and applying attention matrices for self-attention, compared to 4% on 224×224 ImageNet [9] images. In a pure transformer, these attention matrices scale computationally with the square of tokens, which can already be prohibitively expensive (such as with long sentences in NLP). But in a ViT, the problem is compounded further by the tokens scaling with the square of the image size, meaning doubling the image size increases the computation in attention by a factor of 16.

There are already a wealth of techniques that have been explored to address this problem in the NLP space. Several works have introduced "linear" attention (in terms of tokens) either by re-arranging the order of computation using a "kernel trick" [5,16,24,28] or projecting to a token-independent low-rank space [5,24,34], some doing both. However, most of these "linear" attention methods trade computation across the tokens for computation across the features, making them rather expensive. In fact, recently, Flash Attention [8] has shown that an

IO efficient implementation of multihead self-attention can outperform most of these “linear” attention methods even with token counts in the thousands.

A few works have attempted efficient attention in the vision space, too, but none have been explored on their own in a traditional ViT shell. PolyNL [2] treats attention as an efficient third-order polynomial, but this hasn’t yet been explored in a ViT architecture. Attention Free Transformer [37] has an AFT-Simple variant that is similarly efficient, but it performs poorly in a pure ViT and requires extra support from convs and position encodings. We test both of these methods in a standard DeiT [31] shell (see Table 1), and find that both methods, while efficient, result in a significant accuracy drop. Thus, there is room in the literature for a truly efficient, accurate, and general replacement for multihead self-attention.

To that extent, we introduce Hydra Attention (see Fig. 1). Our method results from a somewhat paradoxical behavior in linear attention: with standard multihead self-attention, adding more heads to the model keeps the amount of computation the same. However, after changing the order of operations in linear attention, adding more heads actually *reduces* the compute cost of the layer. We take this observation to its extreme by setting the number of heads in the model to be equal to the number of features, thereby creating an attention module that’s computationally linear with respect to both tokens and features.

Not only is Hydra Attention a more general formulation of previous efficient attention works (see Sect. 3.5), but when using the right kernel, it can be significantly more accurate (see Table 1). In fact, when mixed with standard multi-head attention, Hydra Attention can actually *increase* the accuracy of a baseline DeiT-B model while being faster (see Fig. 4). And by being derived from multihead attention, our method retains several of attention’s nice properties, such as explainability (see Fig. 3) and generality to different tasks.

However, while Hydra Attention is general and efficient for large images, in this paper we focus solely on ImageNet [9] classification using DeiT-B [31], which traditionally uses smaller 224 × 224 and 384 × 384 images. While the efficiency gains aren’t as much here (10–27% based on image size), other efficient attention methods (e.g., [2,37]) already suffer from huge accuracy drops in this regime (see Table 1), whereas Hydra Attention does not. We hope Hydra Attention can become a stepping stone for general, pure transformers with large numbers of tokens in the future.

Our contributions are as follows: we perform a study to validate how many heads a transformer can have (Fig. 2) and find that 12 is the limit for softmax attention, but with the right kernel, any number is feasible. Then we use that observation to introduce Hydra Attention (Sect. 3) for pure transformers by increasing the number of heads in multihead self-attention. We then analyze the action of Hydra Attention mathematically (Sect. 3.4) and introduce a method to visualize its focus (Fig. 3). Finally, we find that by replacing specific attention layers with Hydra Attention (Fig. 4), we can either *improve* accuracy by 1% or match the accuracy of the baseline, while producing a strictly faster model using DeiT-B [31] on ImageNet-1k [9].

2 Related Work

In this paper, our goal is to speed up the inference time of a transformer by removing the token squared computation bottleneck in multihead self-attention.

Efficient Attention. Multihead Self-Attention [32] is a notoriously slow operation, and there have been plenty of works trying to address its computational shortcomings in different domains.

In NLP, several works approximate attention with a decomposable kernel function [5,16,24,28]. This "kernel trick" allows them to reorder the matrix multiplications to be quadratic in terms of features instead of tokens. Some of these methods go further and reduce the dimensionality of this matrix multiplication through a projection to a low rank space [5,24,34]. However, these "linear" attention methods trade computation across the tokens for computation across the features, which can make them expensive. In fact, in the domain of this paper (ImageNet classification), there aren't enough tokens to justify these approaches and most of them produce a *slower* model. And even with thousands of tokens, Flash Attention [8] has shown that an IO-aware implementation of multihead self-attention can actually outspeed even the fastest of these methods.

But reordering operations isn't the only way to speed up attention. In fact, the most common way to "linearize" attention in vision is by using local window attention (e.g., [3,19,21]). This is indeed computationally linear with respect to the number of tokens, but local window attention can be difficult to compute (especially in the case of Swin [21]) and this is only possible with dense, spatially ordered modalities such as images and videos.

Our goal is instead to produce a linear attention method that is efficient, fast to compute, and general across several different modalities.

Efficient Transformers. Replacing the attention module is not the only way to speed up the inference time of a transformer. In fact, depending on the task and the number of tokens, other efficient transformer methods can be more desirable. For instance, attention only accounts for 4% of the total network computation on ImageNet [9] classification, meaning 4% is the maximum obtainable speed-up if only attention is modified.

There are several efficient vision transformers that mix convs and attention together to create a more efficient end product, such as LeViT [13], MobileViT [22], Mobile-Former [4], and LVT [35]. All of these are a valid strategy for images, and we view them as adjacent techniques. Other vision-specific attention papers such as [2,37] use convolutions in addition to their efficient attention, making it difficult to discern whether the improvement comes from the attention method or the introduction of convolution.

In this paper, we make no modifications to the underlying ViT architecture except to swap multihead self-attention for Hydra Attention in order to clearly isolate its impact on performance.

Multihead Attention. Hydra Attention relies on increasing the number of heads used in multihead attention. Interestingly enough, since its introduction in [32], the number of heads used for multihead attention has not been explored in much depth. Some studies have been done on pruning attention heads [23,33], however all studies have been in the direction in reducing the number of heads. In fact, even with ViT-G, the largest ViT models explored in [38], the authors only use 16 attention heads. Thus, we conduct this study ourselves in Fig. 2.

3 Hydra Attention

Standard multihead self-attention [32] scales quadratically with the number of tokens in an image. More concretely, if T is the number of tokens and D is the number of feature dimensions, then creating and applying an attention matrix are both $O(T^2D)$. This poses a problem, then, when T is large (as it is the case with large images), as this operation can quickly become computationally infeasible.

3.1 The Kernel Trick

As discussed in Sect. 2, many works [5,16,24,28] have already attempted to address this by introducing "linear" attention. Given queries Q, keys K, and values V in $\mathbb{R}^{T\times D}$, standard softmax self-attention is computed as

$$A(Q,K,V) = \text{softmax}\left(\frac{QK^T}{\sqrt{D}}\right)V \tag{1}$$

Computing QK^T is $O(T^2D)$ and creates a $T \times T$ matrix, which scales poorly with T. As in [16], we can generalize this operation by treating $\text{softmax}(\cdot)$ as a pairwise similarity between Q and K. That is, for some similarity function $\text{sim}(\cdot)$, we can write

$$A(Q,K,V) = \text{sim}(Q,K)V \tag{2}$$

If we then choose a decomposable kernel with feature representation $\phi(\cdot)$ such that $\text{sim}(x,y) = \phi(x)\phi(y)^T$, we can obtain

$$A(Q,K,V;\phi) = \left(\phi(Q)\phi(K)^T\right)V \tag{3}$$

Then by associativity, we can change the order of computation such that

$$A(Q,K,V;\phi) = \phi(Q)\left(\phi(K)^TV\right) \tag{4}$$

This allows us to compute $\phi(K)^TV$ first, leading to an operation that is $O(TD^2)$ and that creates a D^2 matrix instead of a T^2 one. Note this formulation differs slightly from [16], in that we leave the normalization to the similarity function rather than make it explicit.

3.2 Multi-head Attention

Despite being linear with respect to T, the result in Eq. 4 is still undesirable: D is typically large ($\geq$768) and so creating a $D \times D$ matrix and performing $O(TD^2)$ operations can still be quite costly. However, Eq. 1 through Eq. 4 assume that we create one attention matrix, and thus have one "head".

In practice, most vision transformers use H heads (typically between 6 and 16), where each head creates and applies its own attention matrix. Following [32], each of heads operate on their own D/H subset of features from Q, K, and V. Thus Eq. 1 becomes

$$A(Q_h, K_h, V_h) = \text{softmax}\left(\frac{Q_h K_h^T}{\sqrt{D}}\right) V_h \qquad \forall h \in \{1, \ldots, H\} \tag{5}$$

where $Q_h, K_h, V_h \in \mathbb{R}^{T \times \frac{D}{H}}$. This keeps the total number of operations the same:

$$O(HT^2 D/H) = O(T^2 D) \tag{6}$$

The same is not true, however, for linear attention. Equation 4 becomes

$$A(Q_h, K_h, V_h; \phi) = \phi(Q_h)\left(\phi(K_h)^T V_h\right) \qquad \forall h \in \{1, \ldots, H\} \tag{7}$$

By computing attention in this way, adding heads actually *decreases* the number of operations:

$$O(HT(D/H)^2) = O(TD^2/H) \tag{8}$$

3.3 Adding Heads

Given Eq. 8, the more heads we add to the network, the faster multihead linear attention becomes. That begs the question, how many heads can we reasonably add, anyway? Most transformers in the wild use between 6 and 16 heads [10,17,32,38] depending on the number of features D, but what happens if you increase the number of heads beyond that?

To find out, we train DeiT-B [31] on ImageNet-1k [9] and vary the number of heads H using either standard multi-head self-attention (Eq. 5, MSA) with softmax or multi-head linear attention (Eq. 7, MLA) with cosine similarity, plotting the results in Fig. 2. In terms of memory usage, MSA runs out of memory when $H > 96$ and MLA runs out of memory when $H < 3$.

In terms of performance, while the accuracy for MSA tanks for $H > 12$, the accuracy for MLA with cosine similarity stays quite consistent all the way up to $H = 768$. Amazingly, at this number of heads, H is equal to D, meaning each head has only a scalar features to work with!

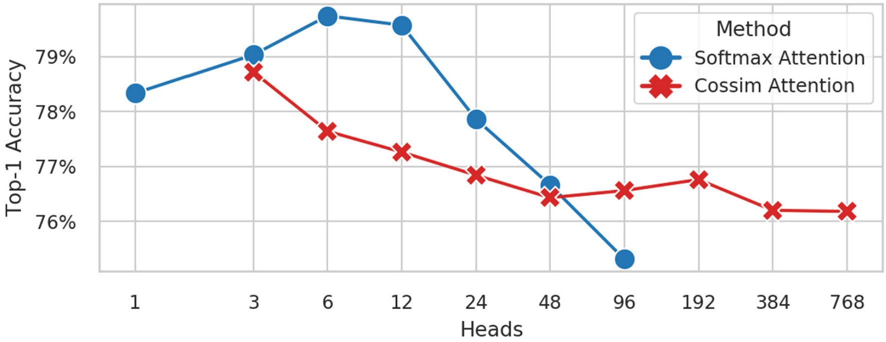

Fig. 2. Varying Heads. We train a DeiT-B model on ImageNet-1k with different numbers of heads using either standard self-attention (blue) using softmax or multi-head linear attention (red) using cosine similarity. Results for standard self-attention ran out of memory for $H > 96$ and multi-head linear attention for $H < 3$. Softmax attention seems to crash in accuracy as we add more heads, while multi-head linear attention stays consistent. Note that H must divide $D = 768$.

3.4 The Hydra Trick

As shown in Fig. 2, it's feasible to scale H up arbitrarily as long as the similarity function $\text{sim}(x, y)$ is not softmax. To exploit this, we introduce the "hydra trick", where we set $H = D$:

$$A(Q_h, K_h, V_h; \phi) = \phi(Q_h) \left(\phi(K_h)^T V_h\right) \qquad \forall h \in \{1, \ldots, D\} \tag{9}$$

In this case, each Q_h, K_h, V_h is a column vector in $\mathbb{R}^{T \times 1}$. If we then vectorize the operation across the heads, we end up with

$$\text{Hydra}(Q, K, V; \phi) = \phi(Q) \odot \sum_{t=1}^{T} \phi(K)^t \odot V^t \tag{10}$$

where $\odot$ denotes element-wise multiplication. Note there is a subtle difference between this vectorization and Eq. 9: ϕ is applied to the entirety of Q and K, rather than to individual column vectors Q_h and K_h. This is important because for each token, Q_h and K_h are scalars, and taking the similarity between two scalars is very restrictive (e.g., cosine similarity can only output -1, 0, or +1).

Also, while the derivation of Eq. 10 comes from multihead attention, it actually ends up performing something quite different: it first creates a global feature vector $\sum_{t=1}^{T} \phi(K)^t \odot V^t$ that aggregates information across all the tokens in the image. Then each $\phi(Q)$ gates the importance of this global feature for each output token. Thus, Hydra Attention mixes information through a global bottleneck, rather than doing explicit token-to-token mixing as in standard self-attention.

This results in a computational complexity of

$$O(TD(D/H)) = O(TD) \tag{11}$$

leaving us with an efficient token mixing module that is linear with both the number of tokens and features in the model, and with no extra constants as in other linear attention methods (such as [5,16,34]). Note that the space complexity of this technique is also $O(TD)$, which is important for real-world speed, where many operations are IO-bound (see [8]).

3.5 Relation to Other Works

There are a few other $O(TD)$ attention candidates in the literature: Attention-Free Transformer [37] (specifically AFT-Simple) and PolyNL [2]. In this section, we explore how Hydra Attention as described in Eq. 10 relates to each.

AFT-Simple [37] is described as

$$\text{AFT-Simple}(Q, K, V) = \sigma(Q) \odot \sum_{t=1}^{T} \text{softmax}(K)^t \odot V^t \tag{12}$$

where $\sigma(\cdot)$ denotes sigmoid. If we allow ϕ to vary between Q and K, this is a direct specilization of Eq. 10 with $\phi(Q) = \sigma(Q)$ and $\phi(K) = \text{softmax}(K)$.

PolyNL [2], on the other hand, is described as

$$\text{PolyNL}(X; W_1, W_2, W_3) = \left(X \odot \frac{1}{T} \sum_{t=1}^{T} X W_1 \odot X W_2 \right) W_3 \tag{13}$$

If we denote $K = XW_1$ and $V = XW_2$, and let $\phi_{\text{mean}}(x) = x/\sqrt{T}$, we can write

$$\text{PolyNL}(X; W_1, W_2, W_3) = \text{Hydra}(X, K, V; \phi_{\text{mean}}) W_3 \tag{14}$$

Thus, Hydra attention can be seen as a more general form of other $O(TD)$ attention methods.

4 Experiments

For all experiments, unless otherwise noted, we use DeiT-B [31] with default settings trained on ImageNet-1k [9] reported as Top-1 accuracy on the validation set. When not specified, the function used for $\phi(\cdot)$ in Eq. 10 is L2 normalization such that $\text{sim}(\cdot, \cdot)$ is cosine similarity. To compute throughput, we sweep over several batch sizes and report the highest average throughput on 30 batches after 10 discarded warm-up iterations.

4.1 The Choice of Kernel

In most of our experiments, following [16] we use cosine similarity as our kernel function for Eq. 10. In Table 1, we explore other possible kernels, including those used by other candidate attention replacement methods as discussed in Sect. 3.5. Yet, no kernel we test outperforms simple cosine similarity.

Table 1. Kernel Choice. Here we vary the choice of kernel function through its feature representation $\phi(\cdot)$ in Eq. 10. We also compare against AFT and PolyNL here as mentioned in Sect. 3.5. Note that some kernels can be asymmetric, with different $\phi(Q)$ and $\phi(K)$. See the appendix for more kernels.

Method	Kernel	$\phi(Q)$	$\phi(K)$	Accuracy
Hydra	Cosine Similarity	$x/\|\|x\|\|_2$		**76.37**
Hydra	Mean	$x/\sqrt{T}$		75.95
Hydra	Tanh Softmax	$\tanh(x)$	$\text{softmax}(x)$	74.18
AFT-Simple [37]	Sigmoid Softmax	$\sigma(x)$	$\text{softmax}(x)$	74.02
PolyNL [2]	Mean	$x/\sqrt{T}$		73.96

This might be because cosine similarity changes the nature of attention. With MSA (Eq. 5), attention exclusively mixes information contained in V, as the mixing weights $\text{sim}(Q, K)$ must sum to 1. That's not the case when using cosine similarity or other unrestricted dot-product kernels like mean. And it turns out, these weights summing to 1 might not be a desirable property in the first place: AFT-Simple [37] as described in Eq. 12 sets $\phi(Q) = \sigma(Q)$ and $\phi(K) = \text{softmax}(K)$, which is closer to a strict mixing of V, but the performance suffers as a result (see Table 1).

We also test using $\tanh(Q)$ instead of $\sigma(Q)$ to see if cosine similarity allowing the result to be negative was the reason, but that performs only slightly better than AFT-Simple. Thus, in this computationally constrained environment, it seems that leaving the kernel to be as unrestricted as possible while normalizing it in some way is important. We test several other kernels and note them in the appendix, but none outperform this simple technique.

4.2 Visualizing Hydra Attention

One of the most desirable qualities of self-attention is its explainability: visualizing the focus of an attention-based model (e.g. with attention rollout [1]) is typically straightforward. The same is less true for Hydra attention.

In order to visualize the focus of a Hydra attention module, we could construct attention matrices $\phi(Q)_h \phi(K)_h^T$ for $h \in \{1, \ldots, D\}$, but each would be rank 1 and it isn't clear how to combine D different attention matrices when each is responsible for a different feature dimension. Simply averaging the heads together produces a meaningless result because each feature dimension encodes different information.

Instead, let's look at the information that each token contributes to the output for the class token. If we sample just the class token c's output from Eq. 10, we get

$$\phi(Q)^c \odot \sum_{t=1}^{T} \phi(K)^t \odot V^t = \sum_{t=1}^{T} \phi(Q)^c \odot \phi(K)^t \odot V^t \tag{15}$$

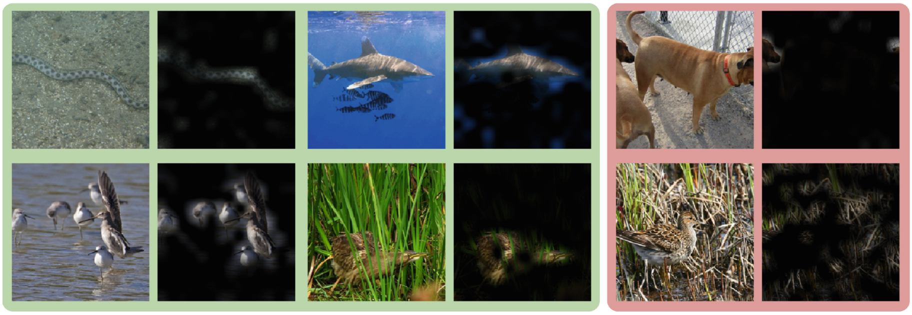

Fig. 3. Hydra Attention Visualization. Visualization of the class token's Hydra attention in the last layer as specified in Sect. 4.2. The 4 images on the left are predicted correctly, while the two examples on the right are misclassified. In the top right image, the network focuses on the head of the wrong dog, guessing the wrong breed. Then on the bottom right, the network misses the bird completely. See the Appendix for more examples.

Thus, each token t has a contribution to the output of the class token c given by

$$\phi(Q)^c \odot \phi(K)^t \odot V^t \tag{16}$$

To tell how this relates to the final prediction, we can use a method similar to Grad-CAM [27]: set the loss to be the logit for the predicted class, then obtain the gradient g with respect to the output of the Hydra attention layer. Then the contribution of each token along the direction of that gradient is

$$(\phi(Q)^c \odot \phi(K)^t \odot V^t)^T g \tag{17}$$

We plot this quantity for several different images in Fig. 3 and the Appendix. For these visualizations, we normalize Eq. 17 along the tokens and show the positive values. These focus maps show that while the math might be different, Hydra attention is performing much the same function as standard self-attention.

4.3 Which Layers Can We Replace?

As discussed in Sect. 3.4 and Sect. 4.1, Hydra Attention with a cosine similarity kernel mixes information between tokens in a different way to standard MSA [32]. Thus, it is perhaps unreasonable to replace every attention layer in the network with Hydra attention. In fact, Hydra attention creates a global feature from the tokens and applies that to each token weighted by Q. Because this is a global operation, it would make more sense in the later layers of the network, as at that point information has already been mixed locally. We test this in Fig. 4, where we progressively replace the MSA attention layers in DeiT-B with Hydra attention following different strategies.

In this experiment, we observe that if we start replacing from the first layer of the network, the performance of the model quickly degrades. However, as it

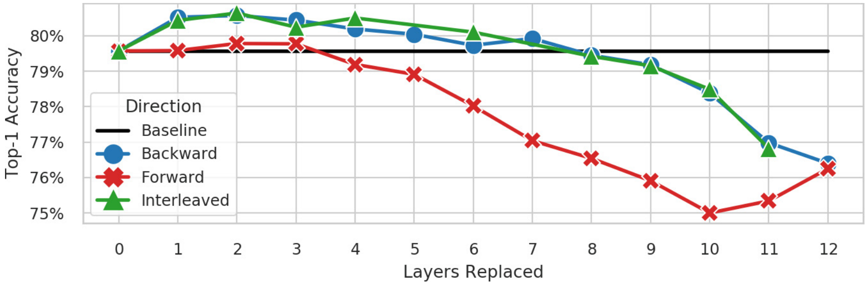

Fig. 4. Which layers can we replace? Replacing softmax self-attention with Hydra attention using different replacement strategies: from the front, from the back, or by interleaving the layers. In all cases, 0 indicates no layers replaced (the baseline), and 12 indicates that all layers were replaced. Surprisingly, with the right layer replacement strategy, Hydra attention can actually *improve* accuracy on ImageNet by 1%, while being faster. Alternatively, we can replace up to 8 layers with no accuracy drop.

turns out, if we replace the layers in reverse starting with the last layer, we can actually *improve* the accuracy of the model. And this improvement is so great that we can replace the last 8 layers of the network and still match the accuracy of the baseline DeiT-B model.

Then, if Hydra attention can be complementary with standard softmax attention, perhaps the best way to combine the two is to interleave them. In Fig. 4, we also attempt to alternate MSA and Hydra layers following the principle that Hydra attention layers should follow MSA layers. However, we don't observe much tangible benefit to this interleaving strategy over starting at the back, suggesting that the number, not necessary the placement, of Hydra layers is what's important.

Note that other efficient attention methods such as AFT [37] and UFO-ViT [29] add conv layers instead of interspersing regular attention layers. Adding these convs serves much the same purpose as using self-attention to perform local mixing, but it's not clear whether the benefit of these prior methods come from the conv layers or their proposed attention layer. In this case, we've clearly isolated that Hydra attention can not only benefit the speed of the model, but also its performance. Future work may be interested in using convs instead.

4.4 Results

We present our final accuracy and FLOP count using Hydra attention in Table 2 compared to standard $O(T^2D)$ attention and other $O(TD)$ methods on ImageNet-1k. Hydra attention achieves 2.4% higher accuracy compared to other $O(TD)$ methods when replacing all layers. And when replacing fewer layers, Hydra attention can strictly outperform the baseline standard attention model: with 2 layers, accuracy increases by 1.1% at 0.7% reduced FLOPs and 2.3% increase in throughput, and with 8 layers, accuracy stays the same with 2.7% reduced FLOPs and 6.3% faster throughput. Interestingly enough, the actual

Table 2. Results. Results for different attention methods in a DeiT-B [31] shell on ImageNet-1k [9] val trained on 224px images along with throughput measured on a V100. Hydra attention results in less accuracy drop than other $O(TD)$ attention methods (AFT-Simple [37] and PolyNL [2]). Moreover, if we don't replace every attention layer in the network, Hydra attention can improve accuracy or keep it the same while still reducing FLOPs and increasing throughput.

Method	Accuracy (%)		FLOPs (G)		Speed (im/s)	
Standard Attention [32]	79.57		17.58		314.8	
AFT-Simple [37]	74.02	(-5.55)	**16.87**	(-4.0%)	346.1	(+9.9%)
PolyNL [2]	73.96	(-5.61)	**16.87**	(-4.0%)	353.8	(+12.4%)
Hydra (2 layers)	**80.64**	(+1.1)	17.46	(-0.7%)	321.9	(+2.3%)
Hydra (8 layers)	79.45	(-0.12)	17.11	(-2.7%)	334.8	(+6.3%)
Hydra (12 layers)	76.40	(-3.17)	**16.87**	(-4.0%)	346.8	(+10.2%)

Table 3. 384px Fine-Tuning. Results for the models in Table 2 fine-tuned with 384px images for 30 epochs. Even with more tokens, Hydra attention can still improve the accuracy over the baseline by 0.59% with 2 layers and increase throughput by 15.4% with 7 layers while matching the baseline's accuracy.

Method	Accuracy (%)		FLOPs (G)		Speed (im/s)	
Standard Attention [32]	81.33		55.54		92.5	
Hydra (2 layers)	**81.92**	(+0.59)	54.52	(-1.8%)	96.3	(+4.1%)
Hydra (7 layers)	81.26	(-0.07)	51.96	(-6.4%)	106.8	(+15.4%)
Hydra (12 layers)	77.85	(-3.48)	**49.40**	(-11.0%)	117.6	(+27.1%)

throughput increase outpaces the flops reduction substantially. This could be due to the observation in [8] that attention is memory-bound and because Hydra Attention uses less memory than standard attention.

Larger Images. To explore whether Hydra Attention retains these gains with more tokens, in Table 3 we fine-tune the backwards replacement models from Fig. 4 at a 384px resolution for 30 epochs using the hyperparameters suggested in [31]. This results in a model with almost 3 times the number of tokens, which should both accentuate the difference between $O(TD)$ and $O(T^2D)$ attention and indicate whether the global information propogation strategy of Hydra Attention is effective at these higher token counts. And indeed, in Table 3, we see the same trend as with 224px images: Hydra Attention can increase accuracy by 0.59% and throughput by 4.1% with 2 layers or keep accuracy the same and increase throughput by 15.4% with 7 layers this time.

Limitations. Okay, but Hydra attention is 197x faster than standard attention (with $T = 197$), so why is the maximum FLOP count reduction only 4%? Well, it turns out that with ViT-B/16 224 × 224 images ($T = 197, D = 768$), only 4.10% of total model FLOPs reside in creating and applying attention matrices.

Table 4. FLOP Count vs Image Size. FLOP count scaling of a ViT-B/16 model across different attention methods as image size increases. We also list the percent of total computation taken by creating and applying attention matrices. While Hydra attention significantly improves the FLOP count of the model at large image sizes, so does local window attention, which has already been shown effective on large images [19]. A limitation of Hydra attention is that it can only be 4% faster than local window attention, though it's more general and can lead to proportionally higher throughputs.

Image Size	T	Baseline		Hydra		Local Window	
		GFLOPs	Attn	GFLOPs	Attn	GFLOPs	Attn
224	197	17.6	4.10%	16.8	0.02%	17.6	4.10%
384	577	55.1	11.13%	49.0	0.02%	51.1	4.10%
448	785	78.0	14.56%	66.7	0.02%	69.5	4.10%
1024	4097	657.3	47.06%	348.1	0.02%	362.8	4.10%
1280	6401	1298.9	58.14%	543.8	0.02%	566.9	4.10%

With Hydra attention, this is reduced down to 0.02%, essentially eliminating the cost of attention in the model. While this does result in a raw throughput increase of up to 10.2% (see Table 2), we can clearly do better.

Of course, the story changes as you increase the image size: in Table 4, we repeat this computation for different image sizes, and the computation of standard attention balloons all the way up to 58% with 1280px images, while Hydra attention remains negligible at 0.02%. We test 384px images ourselves in Table 3, and the speed-up for Hydra Attention is much more pronounced (up to a 27.1% throughput increase). However, further work needs to be done to validate Hydra Attention on tasks that use more tokens (e.g. instance segmentation [15]). Though in those tasks, we'd be comparing against the local window attention used in ViTDet [19], which has already been shown to be effective for large token regimes in images. Compared to local window attention, Hydra attention uses only 4% fewer FLOPs at any image size, though its throughput would likely be proportionally higher (due to less memory usage).

In general, the usefulness of Hydra attention lies in its generality. Local window attention is a powerful solution for dense image prediction, but quickly becomes cumbersome with token sparsity (e.g., with masked pretraining [12,14,30] or token pruning [18,26,36]). We leave this for future work to explore.

5 Conclusion and Future Directions

In this paper, we introduce Hydra Attention, an efficient attention module with many heads. We show that Hydra Attention outperforms other $O(TD)$ attention methods in Table 1 and can even work in tandem with traditional multihead self-attention to improve the accuracy of a baseline DeiT-B model in Fig. 4. However, while Hydra attention works well on ImageNet classification (Table 2, Table 3), its real potential for speed-up lies in larger images (Table 4).

We've taken the first step in showing that Hydra attention can work at all and hope that future work can explore its use in other, more token-intensive domains such as detection, segmentation, or video. Moreover, Hydra attention is a general technique that doesn't make any assumptions about the relationships between tokens, so it can be applied to further improve the speed of token-sparse applications such as masked pretraining [12,14,30] or token pruning [18,26,36]. We hope Hydra attention can be used as a step toward more powerful, efficient, and general transformers in the future.

References

1. Abnar, S., Zuidema, W.: Quantifying attention flow in transformers. arXiv:2005.00928 [cs.LG] (2020)
2. Babiloni, F., Marras, I., Kokkinos, F., Deng, J., Chrysos, G., Zafeiriou, S.: Poly-NL: linear complexity non-local layers with 3rd order polynomials. In: ICCV (2021)
3. Chen, B., Wang, R., Ming, D., Feng, X.: ViT-P: rethinking data-efficient vision transformers from locality. arXiv:2203.02358 [cs.CV] (2022)
4. Chen, Y., et al.: Mobile-former: bridging mobilenet and transformer. In: CVPR (2022)
5. Choromanski, K., et al.: Rethinking attention with performers. arXiv:2009.14794 [cs.LG] (2020)
6. Chowdhery, A., et al.: PaLM: scaling language modeling with pathways. arXiv:2204.02311 [cs.CL] (2022)
7. Cordts, M., et al.: The cityscapes dataset for semantic urban scene understanding. In: CVPR (2016)
8. Dao, T., Fu, D.Y., Ermon, S., Rudra, A., Ré, C.: FlashAttention: fast and memory-efficient exact attention with IO-awareness. arXiv:2205.14135 [cs.LG] (2022)
9. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: ImageNet: a large-scale hierarchical image database. In: CVPR (2009)
10. Dosovitskiy, A., et al.: An image is worth 16 × 16 words: transformers for image recognition at scale. In: ICLR (2020)
11. Fan, H., et al.: Multiscale vision transformers. In: ICCV (2021)
12. Feichtenhofer, C., Fan, H., Li, Y., He, K.: Masked autoencoders as spatiotemporal learners. arXiv:2205.09113 [cs.CV] (2022)
13. Graham, B., et al.: LeViT: a vision transformer in convnet's clothing for faster inference. In: ICCV (2021)
14. He, K., Chen, X., Xie, S., Li, Y., Dollár, P., Girshick, R.: Masked autoencoders are scalable vision learners. In: CVPR (2022)
15. He, K., Gkioxari, G., Dollár, P., Girshick, R.: Mask R-CNN. In: ICCV (2017)
16. Katharopoulos, A., Vyas, A., Pappas, N., Fleuret, F.: Transformers are RNNs: fast autoregressive transformers with linear attention. In: ICML (2020)
17. Kenton, J.D.M.W.C., Toutanova, L.K.: BERT: pre-training of deep bidirectional transformers for language understanding. In: NAACL-HLT (2019)
18. Kong, Z., et al.: SPViT: enabling faster vision transformers via soft token pruning. arXiv:2112.13890 [cs.CV] (2021)
19. Li, Y., Mao, H., Girshick, R., He, K.: Exploring plain vision transformer backbones for object detection. arXiv:2203.16527 [cs.CV] (2022)

20. Li, Y., et al.: MViTv2: improved multiscale vision transformers for classification and detection. In: CVPR (2022)
21. Liu, Z., et al.: Swin transformer: hierarchical vision transformer using shifted windows. In: ICCV (2021)
22. Mehta, S., Rastegari, M.: MobileViT: light-weight, general-purpose, and mobile-friendly vision transformer. arXiv:2110.02178 [cs.CV] (2021)
23. Michel, P., Levy, O., Neubig, G.: Are sixteen heads really better than one? In: NeurIPS (2019)
24. Peng, H., Pappas, N., Yogatama, D., Schwartz, R., Smith, N.A., Kong, L.: Random feature attention. arXiv:2103.02143 [cs.CL] (2021)
25. Radford, A., et al.: Language models are unsupervised multitask learners. OpenAI Blog (2019)
26. Rao, Y., Zhao, W., Liu, B., Lu, J., Zhou, J., Hsieh, C.J.: DynamicViT: efficient vision transformers with dynamic token sparsification. In: NeurIPS (2021)
27. Selvaraju, R.R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., Batra, D.: Grad-CAM: visual explanations from deep networks via gradient-based localization. In: ICCV (2017)
28. Shen, Z., Zhang, M., Zhao, H., Yi, S., Li, H.: Efficient attention: attention with linear complexities. In: WACV (2021)
29. Song, J.: UFO-ViT: high performance linear vision transformer without softmax. arXiv:2109.14382 [cs.CV] (2021)
30. Tong, Z., Song, Y., Wang, J., Wang, L.: VideoMAE: masked autoencoders are data-efficient learners for self-supervised video pre-training. arXiv:2203.12602 [cs.CV] (2022)
31. Touvron, H., Cord, M., Douze, M., Massa, F., Sablayrolles, A., Jégou, H.: Training data-efficient image transformers & distillation through attention. arXiv:2012.12877 [cs.CV] (2020)
32. Vaswani, A., et al.: Attention is all you need. In: NeurIPS (2017)
33. Voita, E., Talbot, D., Moiseev, F., Sennrich, R., Titov, I.: Analyzing multi-head self-attention: specialized heads do the heavy lifting, the rest can be pruned. arXiv:1905.09418 [cs.CL] (2019)
34. Wang, S., Li, B.Z., Khabsa, M., Fang, H., Ma, H.: Linformer: self-attention with linear complexity. arXiv:2006.04768 [cs.LG] (2020)
35. Yang, C., et al.: Lite vision transformer with enhanced self-attention. In: CVPR (2022)
36. Yin, H., Vahdat, A., Alvarez, J., Mallya, A., Kautz, J., Molchanov, P.: AdaViT: adaptive tokens for efficient vision transformer. In: CVPR (2022)
37. Zhai, S., et al.: An attention free transformer. arXiv:2105.14103 [cs.LG] (2021)
38. Zhai, X., Kolesnikov, A., Houlsby, N., Beyer, L.: Scaling vision transformers. arXiv:2106.04560 [cs.CV] (2021)