MORE CONVNETS IN THE 2020S: SCALING UP KERNELS BEYOND 51 × 51 USING SPARSITY

Anonymous authors
Paper under double-blind review

ABSTRACT

Transformers have quickly shined in the computer vision world since the emergence of Vision Transformers (ViTs). The dominant role of convolutional neural networks (CNNs) seems to be challenged by increasingly effective transformer-based models. Very recently, a couple of advanced convolutional models strike back with large kernels motivated by the local-window attention mechanism, showing appealing performance and efficiency. While one of them, i.e. RepLKNet, impressively manages to scale the kernel size to 31×31 with improved performance, the performance starts to saturate as the kernel size continues growing, compared to the scaling trend of advanced ViTs such as Swin Transformer. In this paper, we explore the possibility of training extreme convolutions larger than 31×31 and test whether the performance gap can be eliminated by strategically enlarging convolutions. This study ends up with a recipe for applying extremely large kernels from the perspective of sparsity, which can smoothly scale up kernels to 61×61 with better performance. Built on this recipe, we propose *Sparse Large Kernel Network* (**SLaK**), a pure CNN architecture equipped with sparse factorized 51×51 kernels that can perform on par with or better than state-of-the-art hierarchical Transformers and modern ConvNet architectures like ConvNeXt and RepLKNet, on ImageNet classification as well as a wide range of downstream tasks including semantic segmentation on ADE20K, object detection on PASCAL VOC 2007, and object detection/segmentation on MS COCO.

1 INTRODUCTION

Since invented (Fukushima & Miyake, 1982; LeCun et al., 1989; 1998), convolutional neural networks (CNNs) (Krizhevsky et al., 2012a; Simonyan & Zisserman, 2015.; He et al., 2016; Huang et al., 2017; Howard et al., 2017; Xie et al., 2017; Tan & Le, 2019) have quickly evolved as one of the most indispensable architectures of machine learning in the last decades. However, the dominance of CNNs has been significantly challenged by Transformer (Vaswani et al., 2017) over the past few years. Stemming from natural language processing, Vision Transformers (ViTs) (Dosovitskiy et al., 2021; d'Ascoli et al., 2021; Touvron et al., 2021b; Wang et al., 2021b; Liu et al., 2021e; Vaswani et al., 2021) have demonstrated strong results in various computer vision tasks including image classification (Dosovitskiy et al., 2021; Yuan et al., 2021b), object detection (Dai et al., 2021; Liu et al., 2021e), and segmentation (Xie et al., 2021; Wang et al., 2021a;c; Cheng et al., 2021). Meanwhile, works on understanding of ViTs have blossomed. Plausible reasons behind the success of ViTs are fewer inductive bias (Dosovitskiy et al., 2021), long-range dependence (Vaswani et al., 2017), advanced architecture (Yu et al., 2021), and more human-like representations (Tuli et al., 2021), etc.

Recently, there is a rising trend that attributes the supreme performance of ViTs to the ability to capture a large receptive field. In contrast to CNNs which perform convolution in a small sliding window (e.g., 3×3 and 5×5) with shared weights, global attention or local attention with larger window sizes in ViTs (Liu et al., 2021e) directly enables each layer to capture large receptive field. Inspired by this trend, some recent works on CNNs (Liu et al., 2022b; Ding et al., 2022) strike back by designing advanced pure CNN architecture and plugging large kernels into them. For instance, RepLKNet (Ding et al., 2022) successfully scales the kernel size to 31×31, while achieving comparable results to Swin Transformer (Liu et al., 2021e). However, large kernels are notoriously difficult to train. Even with the assistance of a parallel branch with small kernels, the performance of RepLKNet starts to saturate as the kernel size continues increasing, compared to the scaling trend of

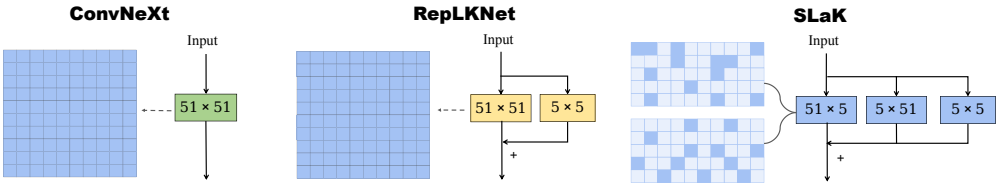


Figure 1: Large depth-wise kernel (e.g., 51×51) paradigms of ConvNeXt, RepLKNet, and SLaK. Dark blue squares refer to the dense weights in convolutional kernels. Light blue squares refer to the sparse weights in convolutional kernels.

advanced ViTs such as Swin Transformer. Therefore, it remains mysterious whether we can exceed the Transformer-based models by further scaling the kernel size beyond 31×31.

In this paper, we attempt to answer this research question by leveraging *sparsity* commonly observed in the human visual system. Sparsity has been seen as one of the most important principles in the primary visual cortex (V1) (Tong, 2003), where the incoming stimuli have been hypothesized to be sparsely coded and selected (Desimone & Duncan, 1995; Olshausen & Field, 1997; Vinje & Gallant, 2000). We extensively study the trainability of large kernels and unveil three main observations: **(i)** existing methods that either naively apply larger kernels (Liu et al., 2022b) or assist with structural re-parameterization (Ding et al., 2022) fail to scale kernel sizes beyond 31×31; **(ii)** replacing one large M×M kernel with two rectangular, parallel kernels (M×N and N×M, where N $<$ M) can smoothly scale the kernel size up to 61×61 with improved performance; **(iii)** constructing with sparse groups while expanding width significantly boosts the performance.

Built upon these observations, we propose SLaK – Sparse Large Kernel Network – a new pure CNN architecture equipped with an unprecedented kernel size of 51×51. Evaluated across a variety of tasks including ImageNet classification (Deng et al., 2009), semantic segmentation on ADE20K (Zhou et al., 2019), object detection on PASCAL VOC 2007 (Everingham et al., 2007), and object detection/segmentation on COCO (Lin et al., 2014), SLaK performs better than or on par with CNN pioneers RepLKNet and ConvNeXt (Liu et al., 2022b) as well as SOTA attention-based models e.g., Swin (Liu et al., 2021e) and Cswin (Dong et al., 2022) Transformers on ImageNet. Our analysis of effective receptive field (ERF) shows that when plugged in the recently proposed ConvNeXt, our method is able to cover a large ERF region than existing larger kernel paradigms.

2 Related Work

Large Kernel in Attention. Originally introduced for Natural Language Processing (Vaswani et al., 2017) and extended in Computer Vision by Dosovitskiy et al. (2021), self-attention can be viewed as a global depth-wise kernel that enables each layer to have a global receptive field. Swin Transformer (Liu et al., 2021e) is a ViTs variant that adopts local attention with a shifted window manner. Compared with global attention, local attention (Ramachandran et al., 2019; Vaswani et al., 2021; Chu et al., 2021; Liu et al., 2021d; Dong et al., 2022) can greatly improve the memory and computation efficiency with appealing performance. Since the size of attention windows is at least 7, it can be seen as an alternative class of large kernel. A recent work (Guo et al., 2022b) proposes a novel large kernel attention module that uses stacked depthwise, small convolution, dilated convolution as well as pointwise convolution to capture both local and global structure.

Large Kernel in Convolution. Large kernels in convolution date back to the 2010s (Krizhevsky et al., 2012b; Szegedy et al., 2015; 2017), if not earlier, where large kernel sizes such as 7×7 and 11×11 are applied. Global Convolutional Network (GCNs) (Peng et al., 2017) enlarges the kernel size to 15 by employing a combination of 1×M + M×1 and M×1 + 1×M convolutions. However, the proposed method leads to performance degradation on ImageNet. The family of Inceptions (Szegedy et al., 2016; 2017) allows for the utilization of varying convolutional kernel sizes to learn spatial patterns at different scales. With the popularity of VGG (Simonyan & Zisserman, 2014), it has been common over the past decade to use a stack of small kernels (1×1 or 3×3) to obtain a large receptive field (He et al., 2016; Howard et al., 2017; Xie et al., 2017; Huang et al., 2017). Until very recently, some works start to revive the usage of large kernels in CNNs. Li et al. (2021) propose involution with 7×7 large kernels that uses distinct weights in the spatial extent while sharing weights across channels. However, the performance improvement plateaus when further expanding the kernel size. Han et al.

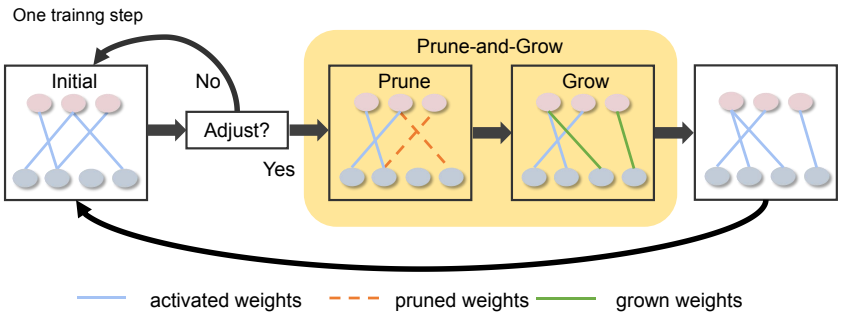


Figure 2: **Dynamic sparsity.** Dynamic sparsity allows us to construct and train initially sparse neural networks (sparse kernels) from scratch. During training, it dynamically adjusts the sparse weights by pruning the least important weights and adding new. Such dynamic procedure gradually optimizes the sparse kernels to a good pattern and hence encourages a more elaborate capture of local features.

(2021b) find that dynamic depth-wise convolution (7×7) performs on par with the local attention mechanism if we substitute the latter with the former in Swin Transformer. Liu et al. (2022b) imitate the design elements of Swin Transformer (Liu et al., 2021e) and design ConvNeXt employed with 7×7 kernels, surpassing the performance of the former. RepLKNet (Ding et al., 2022) for the first time scales the kernel size to 31×31 by constructing a small kernel (e.g., 3×3 or 5×5) parallel to it and achieves comparable performance to the Swin Transformer. A series of work (Romero et al., 2021; 2022) of continuous convolutional kernels can be used on data of arbitrary resolutions, lengths and dimensionalities. Lately, Chen et al. (2022) reveal large kernels to be feasible and beneficial for 3D networks too. Prior works have explored the idea of paralleling (Peng et al., 2017; Guo et al., 2022a) or stacking (Szegedy et al., 2017) two complementary M×1 and 1×M kernels. However, they limit the shorter edge to 1 and do not scale the kernel size beyond 51×51. Different from those prior arts, we decompose a large kernel into two complementary non-square kernels (M×N and N×M), improving the training stability and memory scalability of large convolutions kernels.

Dynamic Sparsity. A long-standing research topic, recent attempts on sparsity (Mocanu et al., 2018; Liu et al., 2021b;c; Evci et al., 2020; Mostafa & Wang, 2019; Dettmers & Zettlemoyer, 2019; Chen et al., 2021) train intrinsically sparse neural networks from scratch using only a small proportion of parameters and FLOPs (as illustrated in Figure 2). Dynamic sparsity enables training sparse models from scratch, hence the training and inference FLOPs and memory requirements are only a small fraction of the dense models. Different from post-training pruning (Han et al., 2015; Frankle & Carbin, 2019), models built with dynamic sparsity can be trained from scratch to match their dense counterparts without involving any pre-training or dense training. Dynamic sparsity stems from Sparse Evolutionary Training (SET) (Mocanu et al., 2018; Liu et al., 2021b) which randomly initializes the sparse connectivity between layers randomly and dynamically adjusts the sparse connectivity via a parameter prune-and-grow scheme during the course of training. The parameter prune-and-grow scheme allows the model's sparse structure to gradually evolve, achieving better performance than naively training a static sparse network (Liu et al., 2021c). In this paper, our target is not to find sparse networks that can match the corresponding dense networks. Motivated by the principle of ResNeXt (Xie et al., 2017; Liu et al., 2022b) – "use more groups, expand width", we instead attempt to leverage dynamic sparsity to scale neural architectures with extreme kernels.

3 Failures of Existing Approaches to Go Beyond 31×31 Kernels

We first study the performance of extreme kernel sizes larger than 31×31 using two existing large-kernel techniques, ConvNeXt (Liu et al., 2022b) and RepLKNet (Ding et al., 2022). We take the recently-developed CNN architecture ConvNeXt on ImageNet-1K as our benchmark to conduct this study. We adopt the efficient large-kernel implementation developed by MegEngine (Meg, 2020) in this paper.

We follow recent works (Liu et al., 2022b; Bao et al., 2021; Liu et al., 2021e; Ding et al., 2022; Touvron et al., 2021b) using Mixup (Zhang et al., 2017), Cutmix (Yun et al., 2019), RandAugment (Cubuk et al., 2020), and Random Erasing (Zhong et al., 2020) as data augmentations. Stochastic Depth (Huang et al., 2016) and Label Smoothing (Szegedy et al., 2016) are applied

Table 1: Test accuracy of 120-epoch ConvNeXt-T trained with various large kernel recipes on ImageNet-1K. "Naive" refers to directly enlarging kernel size of ConvNeXt; "RepLKNet" refers to training ConvNeXt with structural re-parameterization (Ding et al., 2022). The original ConvNeXt is built with 7×7 kernels.

Kernel Size	Top-1 Acc	#Params	FLOPs	Top-1 Acc	#Params	FLOPs
	Naive			RepLKNet		
7-7-7-7	81.0	29M	4.5G			
31-29-37-13	80.5	32M	6.5G	81.5	32M	6.1G
51-49-47-13	80.4	38M	9.4G	81.3	38M	9.3G
61-59-57-13	80.3	43M	11.6G	81.3	43M	11.5G

as regularization with the same hyper-parameters as used in ConvNeXt. We train models with AdamW (Loshchilov & Hutter, 2019). **It is important to note** that all models are trained for *a reduced length of 120 epochs* in this section, just to sketch the scaling trends of large kernel sizes. Later in Section 5, we will adopt the full training recipe and train our models for 300 epochs, to enable fair comparisons with state-of-the-art models. Please refer to Appendix A for more details.

Liu et al. (2022b) show that naively increasing kernel size from 3×3 to 7×7 brings considerably performance gains. Very recently, RepLKNet (Ding et al., 2022) successfully scales convolutions up to 31×31 with structural re-parameterization (Ding et al., 2019; 2021). We further increase the kernel size to 51×51 and 61×61 and see whether larger kernels can bring more gains. Following the design in RepLKNet, we set the kernel size of each stage as [51, 49, 47, 13] and [61, 59, 57, 13], and report test accuracies in Table 1. As expected, naively enlarging kernel size from 7×7 to 31×31 decreases the performance, whereas RepLKNet can overcome this problem by improving the accuracy by 0.5%. Unfortunately, this positive trend does not continue when we further increase kernel size to 51×51.

One plausible explanation is that although the receptive field may be enlarged by using extremely large kernels, it might fail to maintain the desirable property of locality. Since the stem cell in standard ResNet (He et al., 2016) and ConvNeXt results in a 4× downsampling of the input images, extreme kernels with 51×51 are already roughly equal to the global convolution for the typical 224 × 224 ImageNet. Therefore, this observation makes sense as well-designed local attention (Liu et al., 2021e;d; Chu et al., 2021) usually outperforms global attention (Dosovitskiy et al., 2021) in a similar mechanism of ViTs. Motivated by this, we see the opportunity to address this problem by introducing the locality while preserving the ability to capture global relations.

4 A Recipe for Extremely Large Kernels beyond 31×31

In this section, we introduce a simple, two-step recipe for extremely large kernels beyond 31×31:

> **Step 1.** Decomposing a large kernel into two rectangular, parallel kernels.
> **Step 2.** Using *sparse* groups, expanding more width.

Decomposing a large kernel into two rectangular, parallel kernels smoothly scales the kernel size up to 61×61. Although using convolutions with medium sizes (e.g., 31×31) seemingly can directly avoid this problem, we want to investigate if we can further push the performance of CNNs by using (global) extreme convolutions. Our recipe here is to approximate the large M×M kernel with a combination of two parallel and rectangular convolutions whose kernel size is M×N and N×M (where N < M), respectively, as shown in Figure 1. Following Ding et al. (2022), we keep a 5×5 layer parallel to the large kernels and summed up their outputs after a batch norm layer.

This decomposition balances between capturing long-range dependencies and extracting local detail features (with its shorter edge). Moreover, existing techniques for large kernel training (Liu et al., 2022b; Ding et al., 2022) suffer from quadratic computational and memory overhead as the kernel size increases. In stark contrast, the overhead of our method increases just linearly with the kernel size (Figure 4). The performance of kernel decomposition with N = 5 (see Appendix D for the effect of N) is reported as the "Decomposed" group in Table 2. As the decomposition reduces learnable parameters and FLOPs, it is no surprise to observe our network to initially sacrifice accuracy slightly

compared to the original RepLKNet at medium kernel sizes i.e. 31×31. However, as the convolution size continues to increase, our method can *scale* kernel size up to 61×61 with improved performance.

Table 2: Test accuracy of ConvNeXt-T trained with various large kernel recipes on ImageNet-1K. All the models are trained for 120 epochs.

Kernel Size	Top-1 Acc	#Params	FLOPs	Top-1 Acc	#Params	FLOPs	Top-1 Acc	#Params	FLOPs
	Decomposed			Sparse groups			Sparse groups, expand more width		
7-7-7-7	81.0	29M	4.5G	80.0	17M	2.6G	81.1	29M	4.5G
31-29-37-13	81.3	30M	5.0G	80.4	18M	2.9G	81.5	30M	4.8G
51-49-47-13	81.5	31M	5.4G	80.5	18M	3.1G	81.6	30M	5.0G
61-59-57-13	81.4	31M	5.6G	80.4	19M	3.2G	81.5	31M	5.2G

"Use *sparse* groups, expand more width" significantly boosts the model capacity. Recently proposed ConvNeXt (Liu et al., 2022b) revisits the principle introduced in ResNeXt (Xie et al., 2017) that splits convolutional filters into small but more groups. Instead of using the standard group convolution, ConvNeXt simply employs depthwise convolutions with an increased width to achieve the goal of "use more groups, expand width". In this paper, we attempt to extend this principle from a sparsity-inspired perspective – "use *sparse* groups, expand more width".

To be specific, we first replace the dense convolutions with sparse convolutions, where the sparse kernels are randomly constructed based on the layer-wise sparsity ratio of SNIP (Lee et al., 2019)[1] due to its strong performance on large-scale models (Liu et al., 2022a). After construction, we train the sparse model with dynamic sparsity (Mocanu et al., 2018; Liu et al., 2021b), where the sparse weights are dynamically adapted during training by pruning the weights with the lowest magnitude and growing the same number of weights randomly. Doing so enables dynamic adaptation of sparse weights, leading to better local features. As kernels are sparse throughout training, the corresponding parameter count and training/inference FLOPs are only proportional to the dense models. See Appendix G for the full details of dynamic sparsity. To evaluate, we sparsify the decomposed kernels with 40% sparsity and report the performance as the "Sparse groups" column. We can observe in the middle column of Table 2 that dynamic sparsity notably reduces more than 2.0 GFLOPs, despite causing temporary performance degradation.

We next show that the above high efficiency of dynamic sparsity can be effectively transferred to model scalability. Dynamic sparsity allows us to computation-friendly scale the model size up. For example, using the same sparsity (40%), we can expand the model width by 1.3× while keeping the parameter count and FLOPs roughly the same as the dense model. This brings us significant performance gains, increasing the performance from 81.3% to 81.6% with extreme 51×51 kernels. Impressively, equipped with 61×61 kernels, our method outperforms the previous state of the arts (Liu et al., 2022b; Ding et al., 2022) while saving 55% FLOPs.

Large Kernels Generalize Better than Small Kernels with Our Recipe. To demonstrate that the benefits of large kernels, we also report the impact of each step for the small 7×7 kernel in Table 2. We can clearly see that the performance consistently increases with kernel size, up to 51×51. Applying each part of our proposed recipe to 7×7 kernels leads to either no gain or marginal gains compared to our 51x51 kernels. This break-down experiment justifies our claim: large kernel is the root of power, and our proposed recipe helps unleash such power from large kernels.

4.1 Building the Sparse Large Kernel Network (SLaK)

So far, we have discovered our recipe which can successfully scale up kernel size to 51×51 without backfiring performance. Built on this recipe, we next construct our own Sparse Large Kernel Network (SLaK), a pure CNN architecture employed with extreme 51×51 kernels. SLaK is built based on the architecture of ConvNeXt. The design of the stage compute ratio and the stem cell are inherited from ConvNeXt. The number of blocks in each stage is [3, 3, 9, 3] for SLaK-T and [3, 3, 27, 3] for SLaK-S/B. The stem cell is simply a convolution layer with 4×4 kernels and 4 strides.

We first directly increase the kernel size of ConvNeXt to [51, 49, 47, 13] for each stage, and replace each M×M kernel with a combination of M×5 and 5×M kernels as illustrated in Figure 1. We find

[1]SNIP ratio is obtained by globally selecting the important weights across layers with the highest connection sensitivity score $|g \odot w|$, where w and g is the network weight and gradient, respectively.

that adding a BatchNorm layer directly after each decomposed kernel is crucial before summing the output up. Following the guideline of "use *sparse* groups, expand more width", we further sparsify the whole network and expand the width of stages by 1.3×, ending up with SLaK. Even there could be a large room to improve SLaK performance by tuning the trade-off between model width and sparsity (as shown in Appendix C), we keep one set of hyperparameters (1.3× width and 40% sparsity) for all experiments, so SLaK works simply "out of the box" with no ad-hoc tuning at all.

5 Evaluation of SLaK

To comprehensively verify the effectiveness of SLaK, we compare it with various state-of-the-art baselines on a large variety of tasks, including: ImageNet-1K classification (Deng et al., 2009), semantic segmentation on ADE20K (Zhou et al., 2019), object detection on PASCAL VOC 2007 (Everingham et al., 2007), and object detection/segmentation on COCO.

Table 3: **Classification accuracy on ImageNet-1K.** Our Models are trained with AdamW for 300 epochs following ConvNeXt and Swin Transformer.

Model	Image Size	#Param.	FLOPs	Top-1 Acc
ResNet-50 (He et al., 2016)	224×224	26M	4.1G	76.5
ResNeXt-50-32×4d (Xie et al., 2017)	224×224	25M	4.3G	77.6
ResMLP-24 (Touvron et al., 2021a)	224×224	30M	6.0G	79.4
DeiT-S (Touvron et al., 2021b)	224×224	22M	4.6G	79.8
Swin-T (Liu et al., 2021e)	224×224	28M	4.5G	81.3
TNT-S (Han et al., 2021a)	224×224	24M	5.2G	81.3
T2T-ViT$_t$-14 (Yuan et al., 2021a)	224×224	22M	6.1G	81.7
ConvNeXt-T (Liu et al., 2022b)	224×224	29M	4.5G	82.1
SLaK-T	224×224	30M	5.0G	**82.5**
Mixer-B/16 (Tolstikhin et al., 2021)	224×224	59M	11.6G	76.4
ResNet-101 (He et al., 2016)	224×224	45M	7.9G	77.4
ResNeXt101-32x4d (Xie et al., 2017)	224×224	44M	8.0G	78.8
PVT-Large (Wang et al., 2021b)	224×224	61M	9.8G	81.7
T2T-ViT$_t$-19 (Yuan et al., 2021a)	224×224	39M	9.8G	82.4
Swin-S (Liu et al., 2021e)	224×224	50M	8.7G	83.0
ConvNeXt-S (Liu et al., 2022b)	224×224	50M	8.7G	83.1
SLaK-S	224×224	55M	9.8G	**83.8**
DeiT-Base/16 (Touvron et al., 2021b)	224×224	87M	17.6G	81.8
RepLKNet-31B (Ding et al., 2022)	224×224	79M	15.3G	83.5
Swin-B (Liu et al., 2021e)	224×224	88M	15.4G	83.5
ConvNeXt-B (Liu et al., 2022b)	224×224	89M	15.4G	83.8
SLaK-B	224×224	95M	17.1G	**84.0**
ViT-Base/16 (Dosovitskiy et al., 2021)	384×384	87M	55.4G	77.9
DeiT-B/16 (Touvron et al., 2021b)	384×384	86M	55.4G	83.1
Swin-B (Liu et al., 2021e)	384×384	88M	47.1G	84.5
RepLKNet-31B (Ding et al., 2022)	384×384	79M	45.1G	84.8
ConvNeXt-B (Liu et al., 2022b)	384×384	89M	45.0G	85.1
SLaK-B	384×384	95M	50.3G	**85.5**

5.1 Evaluation on ImageNet-1K

ImageNet-1K contains 1,281,167 training images, 50,000 validation images. We use exactly the same training configurations in Section 4, except now training for the full 300 epochs. We observed that models with BatchNorm layers and EMA see poor performance when trained over 300 epochs (also pointed by Liu et al. (2022b)), and resolved this by running one additional pass over the training data, the same as used in Garipov et al. (2018); Izmailov et al. (2018). Please refer to Appendix A for more details about the training configurations.

We compare the performance of SLaK on ImageNet-1K with various the state-of-the-arts in Table 3. With similar model sizes and FLOPs, SLaK outperforms the existing convolutional models such as ResNe(X)t (He et al., 2016; Xie et al., 2017), RepLKNet (Ding et al., 2022), and ConvNeXt (Liu

et al., 2022b). Without using any complex attention modules and patch embedding, SLaK is able to achieve higher accuracy than the state-of-the-art transformers, e.g., Swin Transformer (Liu et al., 2021e) and Pyramid Vision Transformer (Wang et al., 2021b; 2022). Perhaps more interestingly, directly replacing the 7×7 of ConvNeXt-S to 51×51 is able to improve the accuracy over the latter by 0.7%. Moreover, Table 3 shows that our model benefits more from larger input sizes: the performance improvement of SLaK-B over ConvNeXt-B on 384×384 is twice that on 224×224 input, highlighting the advantages of using large kernels on high-resolution training (Liu et al., 2021d).

Moreover, we also examine if SLaK can rival stronger Transformer models – CSwin Transformer (Dong et al., 2022), a carefully designed hybrid architecture with transformers and convolutions. CSwin performs self-attention in both horizontal and vertical stripes. As shown in Appendix B, SLaK consistently performs on par with CSwin Transformers, and is the first pure ConvNet model that can achieve so without any bells and whistles, to our best knowledge.

5.2 Evaluation on ADE20K

For semantic segmentation, we choose the widely-used ADE20K, a large-scale dataset which contains 20K images of 150 categories for training and 2K images for validation. To draw a solid conclusion, we conduct experiments with both short and long training procedures. The backbones are pre-trained on ImageNet-1K with 224×224 input for 120/300 epochs and then are finetuned with UperNet (Xiao et al., 2018) for 80K/160K iterations, respectively. We report the mean Intersection over Union (mIoU) with single-scale testing in Table 4 for comparison. The upper part of Table 4 demonstrates a very clear trend that the performance increases as the kernel size. Specifically, RepLKNet scales the kernel size of ConvNeXt-T to 31×31 and achieves 0.4% higher mIoU. Notably, SLaK-T with larger kernels (51×51) further brings 1.2% mIoU improvement over ConvNeXt-T (RepLKNet), surpassing the performance of ConvNeXt-S. With longer training procedures, SLaK also consistently outperforms its small-kernel baseline (ConvNeXt) by a good margin, reaffirming the advantages of large kernels on downstream dense vision tasks.

Table 4: **Semantic segmentation on ADE20K.** Models are pre-trained on ImageNet-1K with 224×224 input and finetuned on with UperNet. Following Ding et al. (2022), we report mIoU results with single-scale testing. FLOPs are based on input sizes of (2048, 512). Methods marked with * are implemented/reproduced by us.

Model	Kernel Size	mIoU (↑)	#Param	FLOPs
pre-trained for 120 epochs, finetuned for 80K iteration				
ConvNeXt-T (Liu et al., 2022b)	7-7-7-7	44.6	60M	939G
ConvNeXt-S (Liu et al., 2022b)	7-7-7-7	45.9	82M	1027G
ConvNeXt-T (RepLKNet)* (Ding et al., 2022)	31-29-27-13	45.0	64M	973G
SLaK-T	51-49-47-13	**46.2**	65M	936G
pre-trained for 300 epochs, finetuned for 160K iteration				
ConvNeXt-T (Liu et al., 2022b)	7-7-7-7	46.0	60M	939G
SLaK-T	51-49-47-13	**47.6**	65M	936G
ConvNeXt-S (Liu et al., 2022b)	7-7-7-7	48.7	82M	1027G
SLaK-S	51-49-47-13	**49.4**	91M	1028G
ConvNeXt-B (Liu et al., 2022b)	7-7-7-7	49.1	122M	1170G
SLaK-B	51-49-47-13	**50.2**	135M	1172G

5.3 Evaluation on PASCAL VOC 2007

PASCAL VOC 2007 is another popular benchmark for object detection and semantic segmentation. In this section, we evaluate our model on object detection. Models are on pre-trained on ImageNet-1K for 300 epochs and finetuned with Faster-RCNN (Ren et al., 2015). Table 5 shows the results comparing SLaK-T, ConvNeXt-T, ConvNet-T (RepLKNet), and traditional convolutional networks, i.e., ResNet. Again, large kernels lead to better performance. Specifically, ConvNeXt-T with 31×31 kernels achieves 0.7% higher mean Average Precision (mAP) than the 7×7 kernels and SLaK-T with 51 kernel sizes further brings 1.4% mAP improvement, highlighting the crucial role of extremely large kernels on downstream vision tasks.

Table 5: **Object detection on PASCAL VOC 2007.** Faster RCNN is equipped with various backbone networks that are pre-trained for 120 epochs on ImageNet-1K. The pre-trained ConvNeXt-T is obtained from its GitHub repository. FLOPs are based on input sizes of (1280, 800). Methods marked with * are implemented by us.

Model	Kernel Size	mAP (%) (↑)	#Param	FLOPs
ResNet-50 (He et al., 2016)	3-3-3-3	74.0	-	-
ResNet-101 (He et al., 2016)	3-3-3-3	74.3	-	-
ConvNeXt-T (Liu et al., 2022b)	7-7-7-7	80.6	45M	208G
ConvNeXt-T (RepLKNet)* (Ding et al., 2022)	31-29-27-13	81.3	55M	207G
SLaK-T	51-49-47-13	**82.7**	49M	205G

5.4 Evaluation on COCO

Furthermore, we finetune Cascade Mask R-CNN (Cai & Vasconcelos, 2018) on MS COCO (Lin et al., 2014) with our SLaK backbones. Following ConvNeXt, we use the multi-scale setting and train models with AdamW. Please refer to Appendix A.3 for more implementation details. Again, we see an encouraging trend: the performance consistently improves with the increase of kernel size and our 51×51 kernel SLaK outperforms smaller-kernel models.

Table 6: **Object detection and segmentation on MS COCO.** Models are pre-trained on ImageNet-1K and finetuned using Cascade Mask-RCNN. Methods marked with * are implemented by us.

Model	Kernel Size	AP^{box}	AP^{box}_{50}	AP^{box}_{75}	AP^{mask}	AP^{mask}_{50}	AP^{mask}_{75}
pre-trained for 120 epochs, finetuned for 1× (12 epochs)							
ConvNeXt-T (Liu et al., 2022b)	7-7-7-7	47.3	65.9	51.5	41.1	63.2	44.4
ConvNeXt-T (RepLKNET)* (Ding et al., 2022)	31-29-27-13	47.8	66.7	52.0	41.4	63.9	44.7
SLaK-T	51-49-47-13	**48.4**	**67.2**	**52.5**	**41.8**	**64.4**	**45.2**
pre-trained for 300 epochs, finetuned for 3× (36 epochs)							
ConvNeXt-T (Liu et al., 2022b)	7-7-7-7	50.4	69.1	54.8	43.7	66.5	47.3
SLaK-T	51-49-47-13	**51.3**	**70.0**	**55.7**	**44.3**	**67.2**	**48.1**

6 Analysis of SLaK

6.1 Effective Receptive Field (ERF)

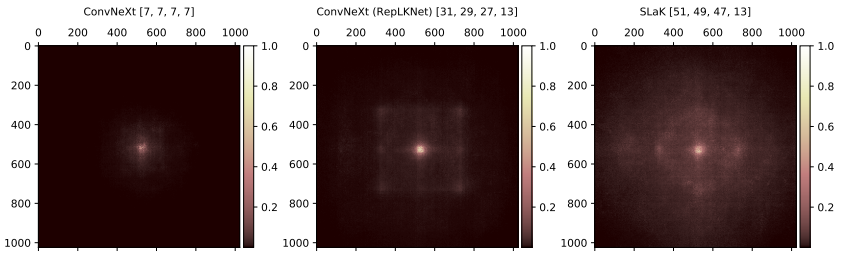


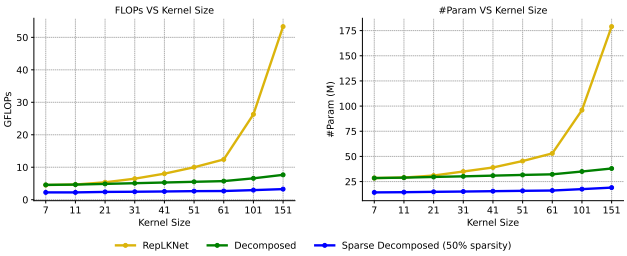
Figure 3: **Effective receptive field (ERF) of models with various kernel sizes**. SLaK is not only able to capture long-range dependence but also the local context features.

The concept of receptive field (Luo et al., 2016; Araujo et al., 2019) is important for deep CNNs: anywhere in an input image outside the receptive field of a unit does not affect its output value. Ding et al. (2022) scale kernels up to 31×31 and show enlarged ERF and also higher accuracy over small-kernel models (He et al., 2016). As shown in the ERF theory (Luo et al., 2016), ERF is proportion to $\mathcal{O}(k\sqrt{n})$, where k and n refers to the kernel size and the network depth, respectively. Therefore, the hypothesis behind the kernel decomposition in SLaK is that the two decomposed M×N and N×M kernels can well maintain the ability of large kernels in terms of capturing large ERF, while also focusing on fine-grained local features with the shorter edge (N).

To evaluate this hypothesis, we compare the ERFs captured by SLaK and RepLKNet. Following Kim et al. (2021); Ding et al. (2022), we sample and resize 50 images from the validation set to 1024×1024,

and measure the contribution of the pixel on input images to the central point of the feature map generated in the last layer. The contribution scores are further accumulated and projected to a 1024×1024 matrix, as visualized in Figure 3. In the left sub-figure, although the original ConvNeXt already improves the kernel size to 7×7, its high-contribution pixels concentrate in the center of the input. Even the 31×31 kernels used by RepLKNet are not sufficient for ConvNeXt to cover the whole input. In comparison, high-contribution pixels of SLaK spread in a much larger ERF, and some high-contribution pixels emerge in non-center areas. This observation is in line with our hypothesis that SLaK balances between capturing long-range dependencies and focusing on the local details.

6.2 Kernel Scaling Efficiency



Resolution R	Dense 51×51	Sparse 51×51	Sparse 51×5 + 5×51
16×16	13.81	13.63	3.92
32×32	58.66	58.66	14.31
64×64	243.15	243.14	58.14
128×128	990.19	990.20	234.32

Figure 4: **Left: Scaling efficiency.** The number of GFLOPs (left) and parameters (right) as the kernel size in ConvNeXt-T scales up. **Right: Real inference time latency (ms).** Real inference time latency of depth-size convolutions with different kernel size. The results are obtained on a single A100 GPU with PyTorch 1.10.0 + cuDNN 8.2.0, in FP32 precision, using no special hardware accelerator.

As Section 4 mentioned, the two components of SLaK, kernel decomposition and sparse group, substantially improve the scaling efficiency of kernel sizes. To further support this, we report the overhead required by various large kernel training methods in Figure 4-left. We simply replace all the kernels in stages of ConvNeXt-T with a set of kernel sizes from 7 to 151 and report the required GFLOPs and the number of parameters. One can clearly see the big gap between full-kernel scaling (yellow lines) and kernel decomposition (green lines) as the kernel size increases beyond 31×31. Even using the ultra-large 151×151 kernels, using our methods would require fewer FLOPs and parameters, compared to full-kernel scaling with 51×51 kernels. Growing evidence has demonstrated that training with high resolution is a performance booster for classification (Tan & Le, 2019; Liu et al., 2021d) and object detection (He et al., 2017; Lin et al., 2017). We believe that large kernels will benefit more in this scenario.

We also report the real inference time latency (ms) of different large-kernel recipes in Figure 4-right using one-layer depth-wise convolutions. The results are obtained on a single A100 GPU with PyTorch 1.10.0 + cuDNN 8.2.0, in FP32 precision, using ***no*** *dedicated sparsity-friendly hardware accelerator*. The input shape is (64, 384, R, R). In general, without special hardware support, while vanilla sparse large kernels cost similar run time to dense large kernels of the same size (sparse kernels receive limited support in common hardware), the sparse decomposed kernels yield more than 4× real inference speed acceleration than directly using vanilla large kernels.

7 Conclusion

Recent works on modern ConvNets defend the essential roles of convolution in computer vision by designing advanced architectures and plugging large kernels. However, the largest kernel size is limited to 31×31 and the performance starts to saturate as the kernel size continues growing. In this paper, we investigate the training of ConvNets with extremely large kernels that are beyond 31×31 and consequently provide a recipe for applying extremely large kernels inspired by sparsity. Based on this recipe, we build a pure ConvNet model that smoothly scales up the kernel size beyond 51×51, while achieving better performance than Swin Transformer and ConvNeXt. Our strong results suggest that sparsity, as the "old friend" of deep learning, can make a promising tool to boost network scaling.

REFERENCES

Megengine:a fast, scalable and easy-to-use deep learning framework. https://github.com/MegEngine/MegEngine, 2020.

GitHub repository: Convnext. https://github.com/facebookresearch/ConvNeXt, 2021.

André Araujo, Wade Norris, and Jack Sim. Computing receptive fields of convolutional neural networks. *Distill*, 4(11):e21, 2019.

Hangbo Bao, Li Dong, and Furu Wei. Beit: Bert pre-training of image transformers. *arXiv preprint arXiv:2106.08254*, 2021.

Guillaume Bellec, David Kappel, Wolfgang Maass, and Robert Legenstein. Deep rewiring: Training very sparse deep networks. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=BJ_wN01C-.

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*, pp. 6154–6162, 2018.

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*, pp. 213–229. Springer, 2020.

Kai Chen, Jiaqi Wang, Jiangmiao Pang, Yuhang Cao, Yu Xiong, Xiaoxiao Li, Shuyang Sun, Wansen Feng, Ziwei Liu, Jiarui Xu, Zheng Zhang, Dazhi Cheng, Chenchen Zhu, Tianheng Cheng, Qijie Zhao, Buyu Li, Xin Lu, Rui Zhu, Yue Wu, Jifeng Dai, Jingdong Wang, Jianping Shi, Wanli Ouyang, Chen Change Loy, and Dahua Lin. MMDetection: Open mmlab detection toolbox and benchmark. *arXiv:1906.07155*, 2019.

Tianlong Chen, Yu Cheng, Zhe Gan, Lu Yuan, Lei Zhang, and Zhangyang Wang. Chasing sparsity in vision transformers: An end-to-end exploration. *Advances in Neural Information Processing Systems*, 34, 2021.

Yukang Chen, Jianhui Liu, Xiaojuan Qi, Xiangyu Zhang, Jian Sun, and Jiaya Jia. Scaling up kernels in 3d cnns. *arXiv preprint arXiv:2206.10555*, 2022.

Bowen Cheng, Alex Schwing, and Alexander Kirillov. Per-pixel classification is not all you need for semantic segmentation. *Advances in Neural Information Processing Systems*, 34, 2021.

Xiangxiang Chu, Zhi Tian, Yuqing Wang, Bo Zhang, Haibing Ren, Xiaolin Wei, Huaxia Xia, and Chunhua Shen. Twins: Revisiting the design of spatial attention in vision transformers. *Advances in Neural Information Processing Systems*, 34, 2021.

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

Ekin D Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V Le. Randaugment: Practical automated data augmentation with a reduced search space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pp. 702–703, 2020.

Xiyang Dai, Yinpeng Chen, Bin Xiao, Dongdong Chen, Mengchen Liu, Lu Yuan, and Lei Zhang. Dynamic head: Unifying object detection heads with attentions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7373–7382, 2021.

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*, pp. 248–255. Ieee, 2009.

Robert Desimone and John Duncan. Neural mechanisms of selective visual attention. *Annual review of neuroscience*, 18(1):193–222, 1995.

Tim Dettmers and Luke Zettlemoyer. Sparse networks from scratch: Faster training without losing performance. *arXiv preprint arXiv:1907.04840*, 2019.

Xiaohan Ding, Yuchen Guo, Guiguang Ding, and Jungong Han. Acnet: Strengthening the kernel skeletons for powerful cnn via asymmetric convolution blocks. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 1911–1920, 2019.

Xiaohan Ding, Honghao Chen, Xiangyu Zhang, Jungong Han, and Guiguang Ding. Repmlpnet: Hierarchical vision mlp with re-parameterized locality. *arXiv preprint arXiv:2112.11081*, 2021.

Xiaohan Ding, Xiangyu Zhang, Yizhuang Zhou, Jungong Han, Guiguang Ding, and Jian Sun. Scaling up your kernels to 31x31: Revisiting large kernel design in cnns. *arXiv preprint arXiv:2203.06717*, 2022.

Xiaoyi Dong, Jianmin Bao, Dongdong Chen, Weiming Zhang, Nenghai Yu, Lu Yuan, Dong Chen, and Baining Guo. Cswin transformer: A general vision transformer backbone with cross-shaped windows. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 12124–12134, 2022.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=YicbFdNTTy.

Stéphane d'Ascoli, Hugo Touvron, Matthew L Leavitt, Ari S Morcos, Giulio Biroli, and Levent Sagun. Convit: Improving vision transformers with soft convolutional inductive biases. In *International Conference on Machine Learning*, pp. 2286–2296. PMLR, 2021.

Erich Elsen, Marat Dukhan, Trevor Gale, and Karen Simonyan. Fast sparse convnets. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 14629–14638, 2020.

Paul Erdős and Alfréd Rényi. On random graphs i. *Publicationes Mathematicae (Debrecen)*, 6: 290–297, 1959.

Utku Evci, Trevor Gale, Jacob Menick, Pablo Samuel Castro, and Erich Elsen. Rigging the lottery: Making all tickets winners. In *International Conference on Machine Learning*, pp. 2943–2952. PMLR, 2020.

M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results. http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html, 2007.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=rJl-b3RcF7.

Kunihiko Fukushima and Sei Miyake. Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In *Competition and cooperation in neural nets*, pp. 267–285. Springer, 1982.

Trevor Gale, Matei Zaharia, Cliff Young, and Erich Elsen. Sparse gpu kernels for deep learning. *arXiv preprint arXiv:2006.10901*, 2020.

Timur Garipov, Pavel Izmailov, Dmitrii Podoprikhin, Dmitry P Vetrov, and Andrew G Wilson. Loss surfaces, mode connectivity, and fast ensembling of dnns. *Advances in neural information processing systems*, 31, 2018.

Meng-Hao Guo, Cheng-Ze Lu, Qibin Hou, Zhengning Liu, Ming-Ming Cheng, and Shi-Min Hu. Segnext: Rethinking convolutional attention design for semantic segmentation, 2022a. URL https://arxiv.org/abs/2209.08575.

Meng-Hao Guo, Cheng-Ze Lu, Zheng-Ning Liu, Ming-Ming Cheng, and Shi-Min Hu. Visual attention network. *arXiv preprint arXiv:2202.09741*, 2022b.

Kai Han, An Xiao, Enhua Wu, Jianyuan Guo, Chunjing Xu, and Yunhe Wang. Transformer in transformer. *Advances in Neural Information Processing Systems*, 34, 2021a.

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. In *International Conference on Learning Representations*, 2021b.

Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *International Conference on Learning Representations*, 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016. doi: 10.1109/CVPR.2016.90.

Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pp. 2961–2969, 2017.

Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q Weinberger. Deep networks with stochastic depth. In *European conference on computer vision*, pp. 646–661. Springer, 2016.

Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4700–4708, 2017.

Itay Hubara, Brian Chmiel, Moshe Island, Ron Banner, Seffi Naor, and Daniel Soudry. Accelerated sparse neural training: A provable and efficient method to find n: M transposable masks. *arXiv preprint arXiv:2102.08124*, 2021.

Pavel Izmailov, Dmitrii Podoprikhin, Timur Garipov, Dmitry Vetrov, and Andrew Gordon Wilson. Averaging weights leads to wider optima and better generalization. *arXiv preprint arXiv:1803.05407*, 2018.

Siddhant Jayakumar, Razvan Pascanu, Jack Rae, Simon Osindero, and Erich Elsen. Top-kast: Top-k always sparse training. *Advances in Neural Information Processing Systems*, 33, 2020.

Bum Jun Kim, Hyeyeon Choi, Hyeonah Jang, Dong Gu Lee, Wonseok Jeong, and Sang Woo Kim. Dead pixel test using effective receptive field. *arXiv preprint arXiv:2108.13576*, 2021.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger (eds.), *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012a. URL https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012b.

Yann LeCun, Bernhard Boser, John Denker, Donnie Henderson, Richard Howard, Wayne Hubbard, and Lawrence Jackel. Handwritten digit recognition with a back-propagation network. *Advances in neural information processing systems*, 2, 1989.

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.

Namhoon Lee, Thalaiyasingam Ajanthan, and Philip Torr. SNIP: SINGLE-SHOT NETWORK PRUNING BASED ON CONNECTION SENSITIVITY. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=B1VZqjAcYX.

Duo Li, Jie Hu, Changhu Wang, Xiangtai Li, Qi She, Lei Zhu, Tong Zhang, and Qifeng Chen. Involution: Inverting the inherence of convolution for visual recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 12321–12330, 2021.

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*, pp. 740–755. Springer, 2014.

Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2117–2125, 2017.

Shiwei Liu, Tianlong Chen, Xiaohan Chen, Zahra Atashgahi, Lu Yin, Huanyu Kou, Li Shen, Mykola Pechenizkiy, Zhangyang Wang, and Decebal Constantin Mocanu. Sparse training via boosting pruning plasticity with neuroregeneration. *Advances in Neural Information Processing Systems*, 34:9908–9922, 2021a.

Shiwei Liu, Decebal Constantin Mocanu, Amarsagar Reddy Ramapuram Matavalam, Yulong Pei, and Mykola Pechenizkiy. Sparse evolutionary deep learning with over one million artificial neurons on commodity hardware. *Neural Computing and Applications*, 33(7):2589–2604, 2021b.

Shiwei Liu, Lu Yin, Decebal Constantin Mocanu, and Mykola Pechenizkiy. Do we actually need dense over-parameterization? in-time over-parameterization in sparse training. In *Proceedings of the 39th International Conference on Machine Learning*, pp. 6989–7000. PMLR, 2021c.

Shiwei Liu, Tianlong Chen, Xiaohan Chen, Li Shen, Decebal Constantin Mocanu, Zhangyang Wang, and Mykola Pechenizkiy. The unreasonable effectiveness of random pruning: Return of the most naive baseline for sparse training. In *International Conference on Learning Representations*, 2022a. URL https://openreview.net/forum?id=VBZJ_3tz-t.

Ze Liu, Han Hu, Yutong Lin, Zhuliang Yao, Zhenda Xie, Yixuan Wei, Jia Ning, Yue Cao, Zheng Zhang, Li Dong, et al. Swin transformer v2: Scaling up capacity and resolution. *arXiv preprint arXiv:2111.09883*, 2021d.

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*, pp. 10012–10022, 2021e.

Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s. *arXiv preprint arXiv:2201.03545*, 2022b.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

Wenjie Luo, Yujia Li, Raquel Urtasun, and Richard Zemel. Understanding the effective receptive field in deep convolutional neural networks. *Advances in neural information processing systems*, 29, 2016.

Decebal Constantin Mocanu, Elena Mocanu, Peter Stone, Phuong H Nguyen, Madeleine Gibescu, and Antonio Liotta. Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. *Nature communications*, 9(1):1–12, 2018.

Hesham Mostafa and Xin Wang. Parameter efficient training of deep convolutional neural networks by dynamic sparse reparameterization. In *International Conference on Machine Learning*, pp. 4646–4655. PMLR, 2019.

Nvidia. Nvidia a100 tensor core gpu architecture. *https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/nvidia-ampere-architecture-whitepaper.pdf*, 2020.

Bruno A Olshausen and David J Field. Sparse coding with an overcomplete basis set: A strategy employed by v1? *Vision research*, 37(23):3311–3325, 1997.

Chao Peng, Xiangyu Zhang, Gang Yu, Guiming Luo, and Jian Sun. Large kernel matters–improve semantic segmentation by global convolutional network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4353–4361, 2017.

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*, 32, 2019.

Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28, 2015.

David W Romero, Anna Kuzina, Erik J Bekkers, Jakub M Tomczak, and Mark Hoogendoorn. Ckconv: Continuous kernel convolution for sequential data. *arXiv preprint arXiv:2102.02611*, 2021.

David W Romero, David M Knigge, Albert Gu, Erik J Bekkers, Efstratios Gavves, Jakub M Tomczak, and Mark Hoogendoorn. Towards a general purpose cnn for long range dependencies in nd. *arXiv preprint arXiv:2206.03398*, 2022.

K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

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

Peize Sun, Rufeng Zhang, Yi Jiang, Tao Kong, Chenfeng Xu, Wei Zhan, Masayoshi Tomizuka, Lei Li, Zehuan Yuan, Changhu Wang, et al. Sparse r-cnn: End-to-end object detection with learnable proposals. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 14454–14463, 2021.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2818–2826, 2016.

Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Thirty-first AAAI conference on artificial intelligence*, 2017.

Mingxing Tan and Quoc Le. EfficientNet: Rethinking model scaling for convolutional neural networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 6105–6114. PMLR, 09–15 Jun 2019.

Ilya O Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Andreas Steiner, Daniel Keysers, Jakob Uszkoreit, et al. Mlp-mixer: An all-mlp architecture for vision. *Advances in Neural Information Processing Systems*, 34, 2021.

Frank Tong. Primary visual cortex and visual awareness. *Nature Reviews Neuroscience*, 4(3):219–229, 2003.

Hugo Touvron, Piotr Bojanowski, Mathilde Caron, Matthieu Cord, Alaaeldin El-Nouby, Edouard Grave, Gautier Izacard, Armand Joulin, Gabriel Synnaeve, Jakob Verbeek, et al. Resmlp: Feedforward networks for image classification with data-efficient training. *arXiv preprint arXiv:2105.03404*, 2021a.

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*, pp. 10347–10357. PMLR, 2021b.

Hugo Touvron, Matthieu Cord, Alexandre Sablayrolles, Gabriel Synnaeve, and Hervé Jégou. Going deeper with image transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 32–42, 2021c.

Shikhar Tuli, Ishita Dasgupta, Erin Grant, and Thomas L Griffiths. Are convolutional neural networks or transformers more like human vision? *arXiv preprint arXiv:2105.07197*, 2021.

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.

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*, pp. 12894–12904, 2021.

William E Vinje and Jack L Gallant. Sparse coding and decorrelation in primary visual cortex during natural vision. *Science*, 287(5456):1273–1276, 2000.

Huiyu Wang, Yukun Zhu, Hartwig Adam, Alan Yuille, and Liang-Chieh Chen. Max-deeplab: End-to-end panoptic segmentation with mask transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 5463–5474, 2021a.

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 *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 568–578, 2021b.

Wenhai Wang, Enze Xie, Xiang Li, Deng-Ping Fan, Kaitao Song, Ding Liang, Tong Lu, Ping Luo, and Ling Shao. Pvt v2: Improved baselines with pyramid vision transformer. *Computational Visual Media*, pp. 1–10, 2022.

Yuqing Wang, Zhaoliang Xu, Xinlong Wang, Chunhua Shen, Baoshan Cheng, Hao Shen, and Huaxia Xia. End-to-end video instance segmentation with transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8741–8750, 2021c.

Ross Wightman. GitHub repository: Pytorch image models. https://github.com/rwightman/pytorch-image-models, 2019.

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)*, pp. 418–434, 2018.

Enze Xie, Wenhai Wang, Zhiding Yu, Anima Anandkumar, Jose M Alvarez, and Ping Luo. Segformer: Simple and efficient design for semantic segmentation with transformers. *Advances in Neural Information Processing Systems*, 34, 2021.

Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1492–1500, 2017.

Weihao Yu, Mi Luo, Pan Zhou, Chenyang Si, Yichen Zhou, Xinchao Wang, Jiashi Feng, and Shuicheng Yan. Metaformer is actually what you need for vision. *arXiv preprint arXiv:2111.11418*, 2021.

Li Yuan, Yunpeng Chen, Tao Wang, Weihao Yu, Yujun Shi, Zi-Hang Jiang, Francis EH Tay, Jiashi Feng, and Shuicheng Yan. Tokens-to-token vit: Training vision transformers from scratch on imagenet. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 558–567, 2021a.

Li Yuan, Qibin Hou, Zihang Jiang, Jiashi Feng, and Shuicheng Yan. Volo: Vision outlooker for visual recognition. *arXiv preprint arXiv:2106.13112*, 2021b.

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 *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 6023–6032, 2019.

Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*, 2017.

Zhun Zhong, Liang Zheng, Guoliang Kang, Shaozi Li, and Yi Yang. Random erasing data augmentation. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pp. 13001–13008, 2020.

Aojun Zhou, Yukun Ma, Junnan Zhu, Jianbo Liu, Zhijie Zhang, Kun Yuan, Wenxiu Sun, and Hongsheng Li. Learning n:m fine-grained structured sparse neural networks from scratch. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=K9bw7vqp_s.

Bolei Zhou, Hang Zhao, Xavier Puig, Tete Xiao, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Semantic understanding of scenes through the ade20k dataset. *International Journal of Computer Vision*, 127(3):302–321, 2019.

A EXPERIMENTAL SETTINGS

A.1 IMAGENET-1K

We share the (pre-)training settings of SLaK on ImageNet-1K in this section. We train SLaK for 300 epochs (Section 5.1) and 120 epochs (Section 4) using AdamW (Loshchilov & Hutter, 2019) with a batch size of 4096, and a weight decay of 0.05. The only differnce between models training for 300 epochs and 120 epochs is the training time. The learning rate is 4e-3 with a 20-epoch linear warmup followed by a cosine decaying schedule. For data augmentation, we use the default setting of RandAugment (Cubuk et al., 2020) in Timm (Wightman, 2019) – "rand-m9-mstd0.5-inc1", Label Smoothing (Szegedy et al., 2016) coefficient of 0.1, Mixup (Zhang et al., 2017) with $\alpha = 0.8$, Cutmix (Yun et al., 2019) with $\alpha = 1.0$, Random Erasing (Zhong et al., 2020) with $p = 0.25$, Stochastic Depth with drop rate of 0.1 for SLaK-T, 0.4 for SLaK-S, and 0.5 for SLaK-B, Layer Scale (Touvron et al., 2021c) of initial value of 1e-6, and EMA with a decay factor of 0.9999. We train SLaK-T with NVIDIA A100 GPUs and the rest of models are trained with NVIDIA V100.

A.2 SEMANTIC SEGMENTATION ON ADE20K

We follow the training setting used in Ding et al. (2022); Liu et al. (2022b) using UperNet (Xiao et al., 2018) implemented by MMSegmentation (Contributors, 2020) with the 80K/160K-iteration training schedule. We conduct experiments with both short and long training procedures. The backbones are pre-trained on ImageNet-1K with 224×224 input for 120/300 epochs and then are finetuned with UperNet (Xiao et al., 2018) for 80K/160K iterations, respectively. We report the mean Intersection over Union (mIoU) with single-scale. All the hyperparameters are the exactly the same as the ones used in the official GitHub repository of ConvNeXt (con, 2021).

A.3 OBJECT DETECTION AND SEGMENTATION ON COCO

For COCO experiments, we follow the training settings used in BEiT, Swin, and ConvNeXt using MMDetection (Chen et al., 2019) and MMSegmentation (Contributors, 2020) toolboxes. The final model weights are adopted (instead of EMA weights) from ImageNet-1K pre-training with 224×224 input. We also conduct experiments with both short and long training procedures. The backbones are pre-trained on ImageNet-1K with 224×224 input for 120/300 epochs and then are finetuned with Cascade Mask R-CNN (Cai & Vasconcelos, 2018) for 12/36 epochs, respectively. All the hyperparameters are the exactly the same as the ones used in the official GitHub repository of ConvNeXt (con, 2021).

A.4 OBJECT DETECTION ON PASCAL VOC 2007

We follow (Liu et al., 2021e) and finetune Faster-RCNN on PASCAL VOC dataset with SLaK-T as the backbone. We use multi-scale setting (Carion et al., 2020; Sun et al., 2021) which leads to the length of the shorter side between 480 and 800 and the ones of the longer side at most 1333. The model is trained with AdamW for 36 epochs with a learning rate of 0.0001, a weight decay of 0.05, and a batch size of 16.

B COMPARISON BETWEEN CSWIN TRANSFORMER AND SLAK

In this appendix, we examine if SLaK can rival CSwin Transformer (Dong et al., 2022), a carefully designed hybrid architecture with transformers and convolutions. It performs self-attention in both horizontal and vertical stripes. To produce a hierarchical representation, convolution layers are also used in CSwin Transformer to expand the channel dimension. As shown in Table 7, besides clearly outperforming the peer ConvNets backbones (ConvNeXt and RepLKNet), SLaK models can also perform on par with CSwin Transformers, with a pure ConvNet for the first time, without any bells and whistles.

Table 7: **Comparison with the CSwin Transformer.** All models are trained with AdamW for 300 epochs.

Model	Image Size	#Param.	FLOPs	Top-1 Accuracy (%)
CSWin-T (Dong et al., 2022)	224×224	23M	4.3G	**82.7**
CSWin-S (Dong et al., 2022)	224×224	35M	6.9G	83.6
CSWin-B (Dong et al., 2022)	224×224	78M	15.0G	**84.2**
CSWin-B (Dong et al., 2022)	384×384	78M	47.0G	85.4
SLaK-T	224×224	30M	5.0G	82.5
SLaK-S	224×224	55M	9.8G	**83.8**
SLaK-B	224×224	95M	17.1G	84.0
SLaK-B	384×384	95M	50.3G	**85.5**

C Trade-off between Sparsity and Width

The principle of "use *sparse* groups, expand more" in Observation #3 essentially sacrifices network sparsity for network width. Therefore, there is a trade-off between model sparsity and width. To have a better understanding of this trade-off, we choose 5 combinations of (Sparsity, Width factor), i.e., (0.20, 1.1×), (0.40, 1.3×), (0.55, 1.5×), (0.70, 1.9×) and (0.82, 2.5×). All the settings have roughly 5.0M FLOPs but with different network widths. The experiments are conducted on SLaK-T. As we expected, the model's performance keeps increasing as model width increases until the width factor reaches 1.5×, after which increasing width further starts to hurt the performance apparently due to the training difficulties associated with highly sparse neural networks.

Table 8: **Trade-off between sparsity and width.** The experiments are conducted on SLaK-T trained for 120 epoch on ImageNet-1K.

Model	(Sparsity, Width)	#Param	FLOPs	Top-1 Accuracy (%)
SLaK-S	(0.20, 1.1×)	29M	5.0G	81.3
	(0.40, 1.3×)	30M	5.0G	81.6
	(0.55, 1.5×)	30M	4.9G	**81.7**
	(0.70, 1.9×)	32M	5.0G	81.6
	(0.82, 2.5×)	32M	5.1G	81.3

D Effect of the Shorter Edge N on SLaK

In Table 9, we report the effect of the shorter edge on the performance of SLaK. We vary the shorter edge $\mathrm{N} \in [3, 5, 7]$ and report the accuracy. All models were trained with AdamW on ImageNet-1K for 120 epochs. We empirically find that N=5 give us the best results, whereas $\mathrm{N} = 3$ and $\mathrm{N} = 7$ has slightly lower accuracy. We hence choose $\mathrm{N} = 5$ as our default option.

Table 9: **Effect of the shorter edge N on SLaK.** The experiments are conducted on SLaK-T trained for 120 epoch on ImageNet-1K.

Model	N	Top-1 Accuracy (%)
SLaK-T	3	81.5
SLaK-T	5	81.6
SLaK-T	7	81.4

E ERF Quantitation of Models with Different Kernel Sizes

In this appendix, we quantify ERF of various models by reporting the high-contribution area ratio r in Table 10 following Ding et al. (2022). r refers to the ratio of a minimum rectangle to the overall input area that can cover the contribution scores over a given threshold t. Assuming an area of A×A at the

Table 10: **Quantitative analysis on the ERF with the high-contribution area ratio** r. A larger value suggests a smoother distribution of high-contribution pixels, hence larger ERF.

	Kernel Size	$t = 20\%$	$t = 30\%$	$t = 50\%$	$t = 99\%$
ResNet-101	3-3-3-3	0.9%	1.5%	3.2%	22.4%
ResNet-152	3-3-3-3	1.1%	1.8%	3.9%	34.4%
ConvNeXt-T	7-7-7-7	2.0%	3.6%	7.7%	55.5%
ConvNeXt-T (RepLKNet)	31-29-27-13	4.0%	9.1%	19.1%	97.5%
SLaK-T	51-49-47-13	**6.9%**	**11.5%**	**23.4%**	**97.5%**

center can cover $t = 30\%$ contribution scores of an 1024×1024 input, then the area ratio of $t = 30\%$ is $r = (A/1024)^2$. Larger r suggests a smoother distribution of high-contribution pixels. We can see that with global kernels, SLaK naturally considers a larger range of pixels to make decisions than ConvNeXt and RepLKNet.

F Learning Curve

We here share the learning curve of different architectures on ImageNet-1K classification in Figure 5. Again, we can observe that large kernels provide significant training loss gains (yellow lines and blue lines). Increasing kernel size from 31×31 to 51×51 further decreases the training loss with improved test accuracy. It is worth noting that even though SLaK's kernel size is as large as 51×51, it enjoys a very promising convergence speed compared to the models with smaller kernel sizes.

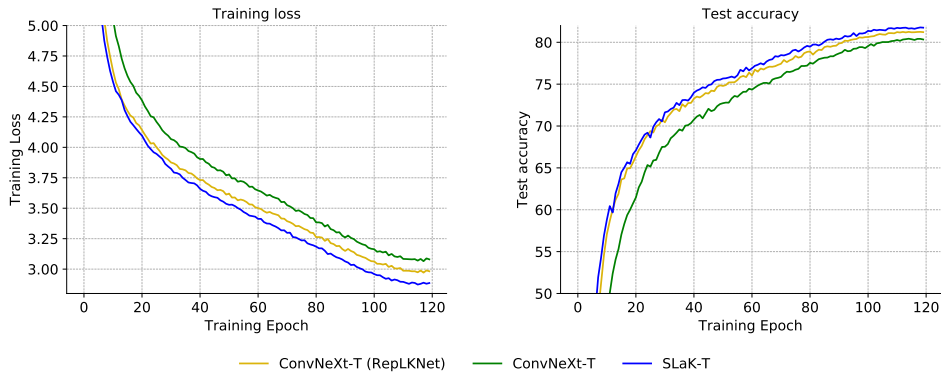


Figure 5: **Learning Curve.** Learning curves in terms of training loss and test accuracy of various models on ImageNet-1K classification.

G Dynamic Sparsity

Dynamic Sparsity (Mocanu et al., 2018; Liu et al., 2021b;c; Evci et al., 2020; Mostafa & Wang, 2019; Dettmers & Zettlemoyer, 2019; Chen et al., 2021) is a class of methods that enable the end-to-end training of intrinsically sparse neural networks (also termed as sparse training). Dynamic Sparsity starts from a sparse network and jointly optimizes the sparse connectivity and model weights during training. Without loss of generality, we provide the general pseudocode of Dynamic Sparsity that can cover most of the existing methods in Algorithm 1.

While there is an upsurge in increasingly efficient ways for sparse training (Bellec et al., 2018; Mostafa & Wang, 2019; Dettmers & Zettlemoyer, 2019; Evci et al., 2020; Liu et al., 2021c; Jayakumar et al., 2020; Chen et al., 2021), most dynamic sparsity methods contain three key components: sparse model initialization, model weight optimization, and sparse weight adaptation. We explain them in detail below.

Algorithm 1 Dynamic Sparsity

Require: Dense neural network $\boldsymbol{\theta}$, sparse Neural Network $\boldsymbol{\theta}_s$ dataset $\{x_i, y_i\}_{i=1}^{\mathrm{N}}$, sparsity s, binary masks: $\mathbf{M} = \{\mathbf{M}^1, \ldots, \mathbf{M}^{\mathrm{L}}\}$ where $\{\mathbf{M}^1, \ldots, \mathbf{M}^{\mathrm{L}}\}$ refer to the binary mask from layer 1 to L, Adaptation Frequency $\Delta\mathrm{T}$, Adaptation Rate p

```
# Sparse Model Initialization
M ← SNIP(x; θ; s)                          ▷ Using SNIP to generate masks.
θ_s ← θ ⊙ M                                ▷ Applying masks.
for each training step t do
    # Model Weight Optimization
    θ_s ← AdamW(x; θ_s)
    θ_s ← θ_s ⊙ M                          ▷ Applying masks.
    if (t mod ΔT) = 0 then
        # Sparse Weight Adaptation
        Pruning p percentage of parameters using magnitude pruning
        Growing p percentage of parameters randomly
        Update the adaptation rate p with a cosine decay schedule
    end if
end for
```

G.1 Sparse Model Initialization

Sparse model initialization determines how to distribute the limited number of parameters across layers, i.e., layer-wise sparsity ratio. Layer-wise sparsity ratio plays an important role for sparse training. Mocanu et al. (2018) first introduced *Erdős-Rényi* (ER) (Erdős & Rényi, 1959) from graph theory to the field of neural networks, achieving better performance than the uniform sparsity ratio. Evci et al. (2020) further extended ER to CNN and brings significant gains to sparse CNN training with the *Erdős-Rényi-Kernel* (ERK) ratio. Liu et al. (2022a) discover that the layer-wise sparsity ratio learned by SNIP (Lee et al., 2019) outperforms the ER based sparsity ratios with large scale models on ImageNet. We confirm that SNIP ratio achieves a better trade-off between accuracy and FLOPs with SLaK than ERK. Concretely, we first calculate the SNIP scores $|g \odot w|$ for each weight, where w and g is the network weight and gradient, respectively. Then, we globally sort all the scores across layers and force the weights with the smallest scores to be zero according to the target sparsity.

G.2 Model Weight Optimization

After initializing the sparse model, we need to optimize the model weights to minimize the loss function. Dynamic sparsity is compatible with the most widely used optimizers. In this paper, we directly inherit the default training configurations and optimizers (AdamW) from ConvNeXt (Liu et al., 2022b) without any changes. After each update, we multiply the weights with the obtained binary masks to enforce the sparse structure.

G.3 Sparse Weight Adaptation

Sparse weight adaptation is the vital factor for dynamic sparsity to outperform static sparsity (Liu et al., 2021c). Sparse weight adaptation controls how we adapt the sparse weights for better sparse structures during training. One simple yet effective heuristic used here is prune-and-grow. Removing p percentage of the existing weights and regrowing the same number of new weights allows dynamic sparsity gradually optimizing the sparse connectivity.

Weight Prune: Although there exists various pruning criteria in pruning literature, we choose the most common way for SLaK, that is, removing the weights with the smallest magnitude (Mocanu et al., 2018).

Weight Grow: The most common ways to grow new weights are random-based growth (Mocanu et al., 2018) and gradient-based growth (Dettmers & Zettlemoyer, 2019; Evci et al., 2020). We simply grow new weights by randomly sampling following (Mocanu et al., 2018).

Following Liu et al. (2021c), we specifically tune two factors for SLaK-T that control the strength of weight adaptation, adaptation frequency f and adaptation rate p. Adaptation frequency determines after how many training iterations we adjust the sparse weights, and the latter controls the ratio of the weight that we adjust at each adaptation. We share the results in Figure 6. We empirically find that $f = 100$ and $p = 0.3$ works best for SLak-T and choose the same set of hyperparameters for SLak-S/B, without further tuning.

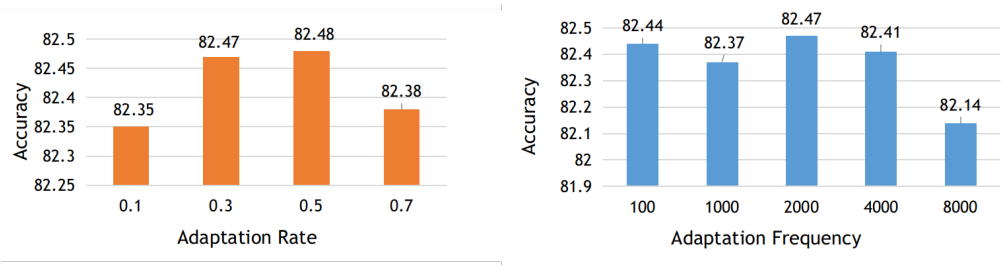


Figure 6: **Left:** Effect of adaptation rate p on the performance of SLaK-T. f is set as 100. **Right:** Effect of the adaptation frequency f on the performance of SLaK-T. p is set as 0.3.

H INFERENCE THROUGHPUT MEASUREMENT

Currently, the promising benefits of dynamic sparsity in our paper can not be fully translated into real speedup, due to the limited support of unstructured sparsity in commonly used hardware.

To provide a full picture of this aspect, we provide a comparison of inference throughput among SLaK-T, Swin-T, and ConvNeXt-T using an A100 GPU with PyTorch 1.10.0 + cuDNN 8.2.0, in FP32 precision. The results are reported in Table 11. On common GPUs without any sparsity-aware accelerators, the throughput of Swin-T and ConvNeXt-T is around 1.8x and 2.5x larger than SLaK-T, which is rather acceptable given the 1.3× wider model size and >7× larger kernel size, thanks to the kernel decomposition trick. Without using decomposition, naively scaling the kernel size of ConvNeXt to 51×51 will be 8.5× slower than the original 7×7 kernel.

Table 11: **Inference throughput comparisons among ConvNeXt-T, Swin-T, and SLaK-T on ImageNet-1K.** The results are obtained on an A100 GPU with PyTorch 1.10.0 + cuDNN 8.2.0, in FP32 precision, using no sparsity-aware accelerator.

Models	Image Size	Kernel Size	Throughput ↑
ConvNeXt-T	224×224	7-7-7-7	1779
Swin-T	224×224	-	1305
SLaK-T	224×224	51-19-17-13	709

To better understand different components of our training recipe on real inference speed (without sparsity-aware accelerators), we break down our training recipe and report the corresponding throughput in Table 12. In general, sparse large kernels cost similar run time to dense large kernels of the same size (neither acceleration nor overhead), but the decomposed kernels yield 3× real inference speed acceleration than directly using vanilla large kernels. After increasing the model width to 1.3×, our model is still 2× faster than vanilla large kernels.

Table 12: **Inference throughput of different large-kernel recipes on ImageNet-1K.** The results are obtained on an A100 GPU with PyTorch 1.10.0 + cuDNN 8.2.0, in FP32 precision, using no sparsity-aware accelerator.

Model	Kernel Size	Naive	Decomposed	Sparse Groups	Sparse Groups + 1.3× width
ConvNeXt-B	51-49-47-13	96	283	282	198

I COMPARISONS WITH OTHER CHOICES FOR LARGE-KERNEL DECOMPOSITION

To further study the effectiveness of our choice for large-kernel decomposition, we conduct a comparison among the following four other baselines:

- Naively stacking 3 3x3 kernels.
- Sequential convolutions with 51×5 and 5×51 kernels.
- Parallel convolutions with 51×5 and 5×51 kernels (same as SLaK).
- Dilation convolutions with a similar receptive field to SLaK, i.e., we set the kernel size of each stage as [19, 17, 17, 5] with a dilation factor of 3.

To eliminate the effect of other components, we remove the dynamic sparsity and keep a parallel 5x5 kernel for all models. We can observe from Table 13 that naively stacking multiple small kernels and dilation kernels achieve much lower accuracy than the two decomposed 51x51 methods. And the sequential decomposition slightly underperforms the parallel one by 0.1%. The superior performance of kernel decomposition (either sequential or parallel) over other choices also support the effectiveness of our recipe.

Table 13: **Comparisons with other choices for large-kernel decomposition.** All models are trained for 120 epochs.

Model	Stacking 3 3x3	Dilation Convolutions	Sequential 51×5 + 5×51	Parallel 51×5 + 5×51
ConvNeXt-T	80.2	81.1	81.4	81.5

J LIMITATIONS AND DISCUSSION OF BROADER IMPACT

The main limitation of this work is that the sparse architecture is implemented with binary masks due to the limited support of sparse neural networks by the commonly used hardware such as GPU and TPU. Therefore, the inference FLOPs reported in the main paper are the theoretical values. Traditional works on structured sparsity (Han et al., 2015; Frankle & Carbin, 2019; Gale et al., 2020; Liu et al., 2021a) mainly focus on finding sparse subnetworks that can match the performance of their dense counterparts. The promising results in our paper go one step further and demonstrate the large potential of unstructured sparsity for scaling modern neural architectures. Once this great potential is supported in the future, it can have a significant positive impact on our planet by saving a huge amount of energy and reducing overall total carbon emissions. Although not the focus of this current work, it would be interesting for future work to examine the speedup of sparse large kernels, using such specialized hardware accelerators, as we see much improvement room of promise here. For example, at high unstructured sparsity levels, XNNPACK (Elsen et al., 2020) has already shown significant speedups over dense baselines on smartphone processors. Furthermore, an increasing number of companies and researchers start considering including unstructured sparsity support in their hardware (Liu et al., 2021b; Gale et al., 2020; Nvidia, 2020; Zhou et al., 2021; Hubara et al., 2021). We hope our work can provide more motivation for such advances.

While our paper is solely scientific, others could build software with a negative impact using our technique, like privacy leakage and fairness issues. Moreover, the promising results of sparsity could bring benefits as cost saving, but could also possibly result in a debate between dense NN hardware and sparse NN hardware, might leading to resource waste.