ShiftwiseConv: Small Convolutional Kernel with Large Kernel Effect

Supplementary Material

Appendix A: Inference Throughput Measurement

In Table 7, we present a comparison of inference throughput. These measurements were conducted on an A100-80GB GPU, using an input resolution of 224 $\times$ 224 pixels. The experiments were carried out with PyTorch version 2.0.0 in conjunction with cuDNN version 11.8.0, employing FP32 precision. As shown in Table 7, our algorithm is notably slower than SLaK [35] and UniRepLKNet [17], with throughput on the A100 GPU at approximately 3/5 and 1/3 of the latter two, respectively.

Utilizing PyTorch's operators, which have not been specifically optimized for large kernel convolutions, our approach yields throughput comparable to SLaK. However, in this scenario, reparameterization (Rep) impedes SLaK's inference speed, reducing throughput by half. This decline is attributed to the merging of strip-like convolutions into a larger convolution, resulting in a substantial number of zero values.

Table 7. **Inference throughput comparison**. The results are reported in FP32 precision. We conducted this experiment using an A100-80GB GPU with PyTorch 2.0.0 + cuDNN 11.8.0. The symbol ★ indicates the use of efficient large-kernel convolutions provided by RepLKNet [16]. The symbol Ⓡ denotes Rep.

Method	Throughput (img/s)	IN-1K acc.
SW-tiny	378	83.4
SLaK-T Ⓡ★ [35]	638	82.5
UniRepLKNet-T [17]	1125	83.2
SLaK-T [35]	371	82.5
SLaK-T Ⓡ [35]	174	82.5
SW-small	243	83.9
SLaK-S Ⓡ★ [35]	385	83.9
UniRepLKNet-S [17]	689	83.8
SLaK-S [35]	217	83.8
SLaK-S Ⓡ [35]	108	83.8

Appendix B: Operator Optimization

Our SW operator possesses four key attributes that facilitate optimization. These characteristics are conducive to enhancing the operator's efficiency.

B.1 Computational Density

RepLKNet [16] defends the efficiency of large kernels by demonstrating that small depth-wise (DW) convolutions have a low computation *vs.* memory access cost ratio, which can render DW operations inefficient on GPUs. As kernel size increases, computational density enhances. Our SW operator, as detailed in section Sec. 3, employs group convolutions that take a single input and yield multiple outputs, matching the computational density of SLaK's strip-like convolutions. Large kernel convolutions, influenced by padding, display reduced computational density at the edges of feature maps, correlating with an arithmetic sequence related to kernel size. In contrast, the small convolutions in our algorithm are less affected by padding's impact on computational density.

B.2 Operator Optimization

In general, our SW operator capitalizes on the outputs from shared convolutions and stacks them spatially. This spatial stacking operation processes inputs and outputs from the same row or column, which minimizes data loading costs and thereby confers optimization benefits. Specifically, with E denoting the number of edges (section Sec. 3), each pixel in the input or output undergoes at most $2E + 1$ moves. Using atomicAdd for these moves on output pixels can cause delays due to synchronization. However, moving these operations to shared memory can effectively avoid such delays. Moreover, with E defaulted to 4 in this study, the shared memory size required is relatively small.

B.3 Sparsity Gain

Our SW algorithm applies coarse-grained pruning to eliminate filters, significantly reducing the computational load during convolutions. Section Sec. 5 offers an analysis of the resulting sparsity levels. Considering B.1, the operator maintains a high computation *vs.* memory access cost ratio even with sparsity, ensuring a high computational ceiling. Importantly, our approach preserves the module's structure, sidestepping the common issues associated with pruning.

B.4 Operator Fusion

The convolution and SW operator are executed sequentially, permitting further operator fusion. This can eliminate the need to transfer data to external memory, using high-speed on-chip storage to complete both operations in one step, conserving VRAM and reducing data loading times.

The version of our SW operator in use is based on atomicAdd and remains unoptimized. The development of an efficient operator that effectively utilizes these features requires the expertise of a skilled CUDA programmer.

Appendix C: Training Configurations

C.1 Phases of Config Evolution

In experiments #0 to #23, we employed the hyperparameters from SLaK, as detailed in Table 8 under SLaK-T. The top-performing model configuration #20 achieved a final

accuracy of 82.70% after training for 300 epochs. Upon observing the model's performance at 120 epochs, we suspected overfitting in the 300-epoch training regime. To address this, we selected parameters aimed at mitigating overfitting. We also observed that UniRepLKNet-T's hyperparameters closely resembled those of SLaK-T, albeit with shorter warmup phases and higher dropout rates. Guided by these insights, we combined SLaK-S's sparsity-related hyperparameters with UniRepLKNet-T's to set the configuration for SW-T, as shown in Table 8. This approach allowed our SW-T model to outperform both SLaK-T and UniRepLKNet-T, reaching an accuracy of 83.39%. Notably, with this configuration, the model's accuracy after 120 epochs of training was 82.27%, the same as #20. Leveraging this observation, we conducted an experiment (#25) removing the SE module, akin to #19 which had achieved 82.25%. However, in this instance, the accuracy decrease was more pronounced, dropping to 82.14% (Table 9).

The incorporation of these new hyperparameters introduced some variability in model performance (#19 *vs.* #25). However, these differences did not significantly impact the overall conclusions drawn from our trials. The primary objective of our paper is to achieve the effects of large kernel convolutions using small convolutions. Consequently, in addition to the emerging variables (C.2), we continue to utilize the previous parameter configurations and network structures as informed by our experiments. Although our model's hyperparameters are not yet optimal, we have achieved our initial goal. We hope our research will inspire further studies. Detailed training commands for ImageNet-1K are included in the released code.

C.2 Emerging Variables

Our approach introduces novel variables that have not been previously considered in related research, such as the synchronization frequency of sparse masks and the number of edges in spatially stacked convolutions. These variables significantly influence model performance. For instance, adjusting the initial sparsity values (#15) has been shown to lead to substantial improvements compared to #11. The interplay among these variables requires further investigation. Sparse methods illustrate the need for a comprehensive consideration of multiple factors. In our study, experiments manipulating the initial sparsity values (#15) and the sparsity variations throughout the training process (#17) both significantly influenced the final model performance. However, these implementations employed different methodologies. This underscores the importance of effectively unifying these elements on a theoretical level. After all, the modified pruning method in our study did not utilize data from the training process but relied on the gradient status at certain steps to influence synchronized sparsity. Such a pruning approach may disproportionately affect small weights near a given threshold, potentially leading to insignificant changes and inconsistent predictions. For details on those parameter settings and their implications, please refer to the training commands on GitHub.

Table 8. We trained our model on ImageNet-1K using 8 NVIDIA GTX 4090D GPUs with the following hyperparameters. "only L" indicates whether sparsity pruning is applied only to large kernels, "u" refers to the update frequency of the sparse mask, "width factor" denotes the factor of change in the number of channels in the model, "magnitude" represents the pruning process where parameters are sorted by their absolute values and the least significant ones are removed, and "magnitude_sum" means the pruning process where filters are sorted by the sum of their absolute values and the least significant filters are eliminated, respectively.

settings	SLaK-T	SW-tiny	SW-small
input scale	224	224	224
batch size	4096	4096	4096
optimizer	AdamW	AdamW	AdamW
LR	4×10^{-3}	4×10^{-3}	4×10^{-3}
LR schedule	cosine	cosine	cosine
weight decay	0.05	0.05	0.05
warmup epochs	20	5	5
epochs	300	300	300
mixup alpha	0.8	0.8	0.8
cutmix alpha	1.0	1.0	1.0
erasing prob.	0.25	0.25	0.25
label smoothing ε	0.1	0.1	0.1
sparse init	snip	snip	snip
width factor	**1.3**	**1.0**	**1.0**
depths	[3, 3, 9, 3]	[3, 3, 18, 3]	[3, 3, 27, 3]
dims	[96, 192, 384, 768]	[80, 160, 320, 640]	[96, 192, 384, 768]
kernel size	[51, 49, 47, 13, **5**]	[51, 49, 47, 13, **3**]	[51, 49, 47, 13, **3**]
u	2000	100	100
prune	magnitude	magnitude_sum	magnitude_sum
only L	False	True	True

Table 9. We conducted analysis experiments to evaluate the model's performance before and after the hyperparameter update. In ablation studies, we analyzed the impact of ghost and sparsity on the model's performance.

	SW-tiny	IN-1K acc. (120 epochs)	IN-1K acc. (300 epochs)
#19	#20 - SE	82.25	-
#20	the best one	82.27	82.70
#24	u100	82.27	83.39
#25	#24 - SE	82.14	-
#26	#24 - ghost	-	82.99
#27	sparsity(0.3→0.2)	-	83.46

Appendix D: Structure Options

Interleaving Large and Small Convolutional Kernels Experiment #23, which interleaved large and small convolutional kernels, yielded performance comparable to our final model structure #20 and reduced the parameter count (Table 8). Despite these advantages, we opted not to adopt UniRepLKNet's such structural configuration for three reasons: (1) We aimed to design SW as a plug-and-play module. (2) The equivalent large kernel convolutions in SW consume fewer parameters, eliminating the need to balance their usage with the model's total parameters. (3) Most importantly,

Figure 7. The qualitative analysis of detection and segmentation results on the COCO dataset. Distinct categories are highlighted with unique colors. This analysis reveals that our method effectively identifies challenging objects, including densely packed entities, those with occlusions, objects in low-light conditions, scenes with significant foreground-background ambiguity, and small objects.

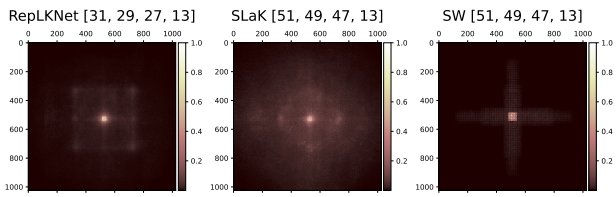

Figure 8. **Effective receptive fields (ERFs) comparison.**

the parameters saved were insufficient to significantly increase the model's depth or width.

Sparsity and Ghost-like Structures These are key features of the SW architecture, but their necessity is under scrutiny. Both are closely related to our foundational experiments with SLaK. Experiments #24 and #26 confirm that ghost structures enhance model performance (Table 8). Sparsity, however, is more complex. Early experiments #15 and #16 suggest that its role is similar to adjusting the initial sparsity levels across model layers. Additionally, comparing experiments #24 and #27 reveals that reducing sparsity can further improve overall model performance (Table 8). Given the inherited settings from SLaK and the complexities within sparsity that require further analysis, we have retained sparsity in our experiments for this study.

While these structures indicate that some parameters have not been fully optimized, they have satisfied our initial exploration into convolutional kernel sizes. The pursuit of hyper-parameters to saturate the accuracy of our proposed structure is beyond the scope and resources of this paper.

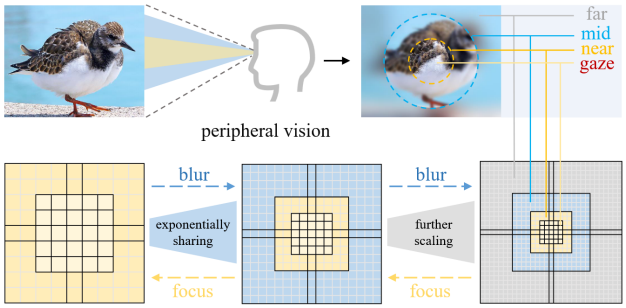

Figure 9. PeLK's [4] focus and blur mechanism, which emulates human peripheral vision. Although our conceptual approach differs from PeLK's, our ERF coincidentally aligns closely with PeLK's mechanisms.

Appendix E: Scaling Issues

Our study refrains from testing models with larger parameter and computational demands for three primary reasons: (1) We have inherited hyperparameters from SLaK [35] and UniRepLKNet [17], yet our model is not saturated. There is potential for performance gains through hyperparameter tuning (#27 *vs.* #24), which underscores the urgency for increased computational resources. (2) Custom hyperparameters are required for models of varying sizes, and given our method involves more hyperparameters and unoptimized operators, the costs are prohibitive. (3) Our focus is on the scientific exploration of utilizing small convolutions as substitutes for large ones. We present performance benchmarks for models that possess parameter and FLOPs comparable to those of ResNet50 and ResNet101.

Appendix F: Analysis

F.1 Qualitative Analysis

We present qualitative analysis results, as shown in Fig. 7, providing an intuitive understanding of our model's performance on COCO. We selected a subset of challenging images from the COCO dataset and draw the predicted detection boxes and segmentation results on them. It is evident that our model effectively recognizes objects under various lighting conditions, across different object sizes, amidst occlusions, and against complex backgrounds. The segmentation results feature well-defined visual boundaries, like mobile phones and skateboards. The SW operator demonstrates robust marginal detection capabilities in these complex tasks. Such performance lays a solid foundation for the future integration of this module into self-supervised learning or sophisticated generative tasks.

F.2 Visualization of ERFs.

In accordance with the method used by RepLKNet [16] and SLaK [35] for generating Effective Receptive Fields (ERFs), we have obtained a 1024×1024 matrix, as illustrated in Fig. 8. For further details on the generation process, refer to SLaK [35]. Our ERFs contrasts with theirs by displaying robust correlations within the immediate neighborhood that diminish progressively with distance. Our ERFs pattern is more predictable, which better complements the sliding window mechanism of convolutions. Despite our conceptual divergence from PeLK [4] and the resulting disparity in convolutional design, our ERFs closely mirrors PeLK's focus and blur mechanisms (Fig. 9).