Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Neural Architecture Search on Large Kernel Convolution Search Space for Computer Vision Tasks

Lbin Hou

lbhou98@163.com

PLA Information Engineering University

Linyuan Wang

PLA Information Engineering University

Tianyuan Liu

PLA Information Engineering University

Nuolin Sun

PLA Information Engineering University

Bin Yan

PLA Information Engineering University

Research Article

Keywords:

Posted Date: September 30th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-11027471/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

Neural Architecture Search on Large Kernel Convolution Search Space for Computer Vision Tasks

Libin Hou[1], Linyuan Wang[1], Tianyuan Liu[1], Nuolin Sun[1], Bin Yan[1]

[1]Henan Key Laboratory of Imaging and Intelligence Processing, PLA Information Engineering University

lbhou@stu.xidian.edu.cn, wanglinyuanwly@163.com, tytauceti@hotmail.com, 1604949605@qq.com, ybspace@hotmail.com

Abstract

Neural Architecture Search (NAS) has been evolving towards larger scales as a method for discovering effective neural networks for computer vision tasks. However, recent NAS methods have almost encountered performance bottlenecks when the model parameter reaches a larger scale. The primary reason for this limitation is that the configuration of the search space remains anchored in the 2010s. This paper introduces Neural Architecture Search on Large Kernel Convolution Search Space (LarKNAS). LarKNAS proposes a novel large kernel search space with the steps to apply phased stage architecture design blocks, consider the range of large kernels, optimize module combinations and search for sparsity in large kernel convolutions. These search space designs are represented by the architecture parameters such as kernel size, block choice, and sparsity. Then LarKNAS builds gradient estimation search method to search these architecture parameters for large-scale backbone models by the estimation of gradient mean and standard deviation, much faster than bilevel optimization methods. Experimental results indicate that LarKNAS can find larger-scale convolutional models than existing NAS, achieving performance comparable to hand-designed models in both ImageNet classification and other vision downstream tasks (ImageNet-1K ACC of 86.6%, COCO box AP of 55.2% and ADE20K mIoU of 53.4%) with 36% search cost (96 GPU days for 335M Params) of ScaleNet.

1. Introduction

Neural Architecture Search (NAS) has shown outstanding performance in designing efficient and accurate models for computer vision tasks such as image classification, object detection and semantic segmentation[1, 2, 3, 4, 5]. With the increasing popularity of large Convolutional Neural Network (ConvNet) and Transformer architectures [6, 7], researchers are increasingly inclined to apply NAS for developing powerful large-scale models, and many existing NAS studies have gradually shifted towards designing search spaces for large models and conducting search within them [8, 9, 10, 11]. However, the key modules in these spaces directly adopt existing neural networks, such as the cell-based search space [12] from ResNet [13] and the space from MobileNet [14] in BigNAS [8]. These above ConvNets were proposed in the 2010s and have been surpassed by some newer era models [7, 15, 16] on visual tasks. On the 'out-of-date' search space design, there are some NAS methods such as EfficientNet [9], FBNet [17, 18, 19], ScaleNet [20] and so on, which take more than thousands of GPU days. A large amount of search cost is invested for large-scale models but the accuracy has a gap with the current popular models [7, 15]. This observation indicates that the design of large models with NAS requires a comprehensive consideration of the search space settings.

Current NAS approaches for vision tasks [8, 9, 10] primarily focus on small kernel sizes. It is important to clarify that the term "large-scale NAS" in the aforementioned literature refers to the model parameter scale (e.g., hundreds of millions of parameters), whereas "small kernel sizes" refers to the convolution kernel dimension within the search space. These two dimensions are orthogonal: a NAS method can target large-scale models while still restricting the kernel size to 3×3 or 5×5. The research gap addressed by this work is precisely the absence of large kernel options in existing large-scale NAS search spaces. While this aligns with industrial needs for compact models, engineering applications requiring long-range feature interaction (e.g., infrastructure defect detection, remote sensing imagery) remain underserved. Large-scale models based on ConvNets are still in the early stages compared with the significant progress made in recent years by Vision Transformers (ViTs) [15, 16]. Various recent models such as ConvNeXt [7], EVA [21] and InternImage [22] have demonstrated that visual models based on ConvNets maintain robust performance, efficiency, and simplicity in various downstream tasks. This highlights the enduring value of ConvNets. The following works [22, 23] have explored the design principles that large-scale ConvNet should follow. However, these manual design rules are not concise and effective enough. RepLKNet [24]

rediscovers the significant value of the design dimension of convolution kernel size. After that, large kernels represent a concise and efficient design, which has become one of the key design paradigms for large-scale convolutional models [23, 25, 26]. This approach conserves computational resources while enhancing the visual representation capabilities of efficient structures by establishing a sufficiently effective receptive field (ERF) [27] through a limited number of large convolutional kernel layers. Additionally, this methodology is particularly advantageous for modular searches in NAS, as it effectively reduces the number of blocks in the search space and further improves the visual representation capacity of the large-scale model.

This paper explores the search for large-scale visual ConvNets and proposes Neural Architecture Search on Large Kernel Convolution Search Space (LarKNAS). The problem of searching for large-scale models in visual tasks is reconsidered inspired by several representative designs of modern ConvNets. Adopting a design with large convolutional kernels, LarKNAS provides visual representation capabilities for large-scale search spaces with large kernels (LarKSpace) while maintaining a lower complexity in the staged architecture. For the designs of LarKSpace, LarKNAS is concerned about the overall architecture of the model and the design of the large convolution kernels. The candidate subnetwork model of LarKSpace is constructed with architecture parameters such as stages, channels, kernel size variation, block selection, and sparse degree. Gradient-based iterative search is employed to assess the performance of candidate subnetworks through the estimation of the mean and standard of gradient deviation per batch. After the search process is completed, LarKNASNet is obtained through retraining. The main experimental results in Figure 1 show that our search method can efficiently discover high accuracy visual models, achieving comparable performance with hand-designed models in ImageNet classification, thereby pushing the large-scale NAS to much larger-scale efficiently.

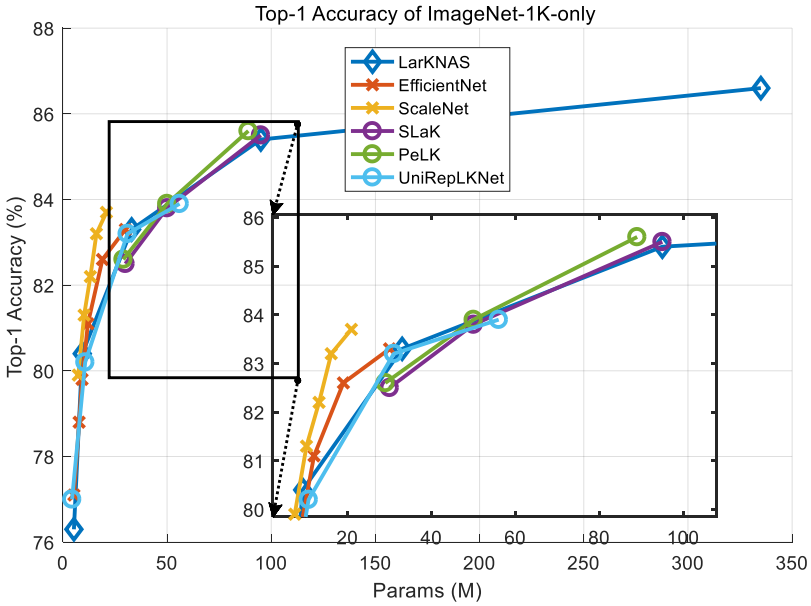


Figure 1 Comparison of LarKNAS, other large-scale NAS methods and hand-designed large kernel ConvNets on top-1 accuracy of ImageNet-1K-only.

In summary, the contributions of this work are three-fold:

1) LarKNAS is proposed to search a large-scale backbone, rethinking the design of large kernel convolution and addressing the performance limitation of visual representation ability of existing NAS.
2) LarKNAS provides a modern Large Kernel Convolution Search Space (LarKSpace) with the phased stage architecture design blocks, "small-medium-large" convolution kernel ranges, optimized module combinations and sparse large kernels. These search space designs of LarKSpace are represented by the architecture parameters for gradient estimate search.
3) LarKNAS can search for models that compare favorably to state-of-the-art hand-designed models over the various parameter scales in ImageNet classification and other downstream visual tasks. Notably, LarKNASNet-L achieved ImageNet-1K ACC of 86.6%, COCO box AP of 55.2% and ADE20K mIoU of 53.4% with search cost of 96 GPU days (only 36% of ScaleNet) for 335M Params model.

2. Related Work

Before listing the related works, we explicitly clarify why existing NAS methods cannot be directly expanded to configure a large kernel search space. First, the search spaces of current NAS methods [8, 9, 10, 20] do not include large kernel options (e.g., 21×21 to 51×51), as they were designed when small kernels dominated; simply adding large kernel candidates would cause the search space to grow exponentially, making gradient-based optimization unstable. Second, bilevel optimization methods represented by DARTS [46] require alternating gradient backpropagation between architecture and weight parameters, which leads to GPU memory overflow when large kernels are involved (as demonstrated in our experiments in Section 5.2). Third, the gradient estimation method proposed in LarKNAS eliminates the need for gradient backpropagation during architecture search, making it feasible to search large kernel configurations within a reasonable computational budget. These three aspects collectively distinguish LarKNAS from existing NAS methods and large-kernel ConvNets.

2.1. Convolution design for Large Kernels

The advent of large kernel convolutional networks can be traced back to classic architectures such as AlexNet [26] and Inception [27], which employed 7×7 or 11×11 kernels in their lower layers. However, following the rise of VGGNet [28] and ResNet [13], which favored 3×3 and 5×5 kernels, the large kernel designs fell out of favor due to challenges related to computational efficiency and performance degradation in GCN [29] and LR-Net [30]. In the 2020s, the kernel concept has transcended spatial convolutions exemplified by Swin Transformer [16], MetaFormer [31], GFNet [32], thereby reflecting the possible advantages of large kernels. RepLKNet [24] employs 31×31 convolutions and implements optimizations that establish modern large convolutional kernels as a scalable backbone model.

Modern large convolution kernels can be primarily categorized by the optimization methods into Re-parameterization ConvNets (RC) and Sparse ConvNets (SC). In terms of RC, RepLKNet [24] scales the kernel size to 31×31 in parallel with small kernels (e.g., 5×5) through re-parameterization techniques of RepVGG [33]. UniRepLKNet [23] further explores the kernel size and branch for reparameterization, utilizing a 17×17 kernel through a manual trial method to optimize the kernel size at each stage. This inspired us to consider that the kernel size should not be excessively large and to pursue further optimizations. However, re-parameterized branches may pose challenges in NAS due to GPU memory constraints. In terms of SC, SLaK [25] combines factorized convolutions with dynamic sparsity to extend the kernel size to 51×51 using strip convolutions. PeLK [26] employs 101×101 dense blocks with parameter sharing based on focus and blur mechanisms, which we also view as a sparse technique with a parameter sharing mechanism. These methods aim to achieve a large effective receptive field with a small number of parameters. While sparsity is an effective design of large kernels, the appropriate kernel range needs further exploration.

In contrast to existing techniques, LarKNAS considers the variation in kernel size. It determines which stage to utilize large kernels convolution and searches for optimal variation in kernel size. Additionally, LarKNAS employs sparse large kernel convolution, and their sparsity is searched to balance performance and computational cost.

2.2. Model architecture for Large Kernels

It is common practice to scale it up for improved performance when utilizing a small model, making the overall architecture crucial for achieving an optimal accuracy-efficiency trade-off. For convolutional neural networks, existing scaling approaches generally focus on factors such as model depth, width, input resolution, bottleneck ratio, and group width, drawing insights from prior NAS developments like EfficientNet [9] and MobileNet [14], as well as hand-crafted design experience. ConvNeXt [7] evolves from ResNet [13] and modernizes architecture design after manual optimization. Following this, large kernel models such as RepLKNet [24] and SLaK [25] align with the principles of ConvNeXt. UniRepLKNet [23] utilizes efficient structures for both inter-channel communication and spatial aggregation to enhance model depth. The concept of block-wise search spaces has gained significant attention in NAS [34, 35, 36, 37]. BlockQNN [38] employs Block-wisely and Q-learning reinforcement method to construct a comprehensive convolutional Neural Network. Building on the block supervision framework proposed by [37], BossNAS [34] investigates the search space of the Transformer architecture. MobileNetV4 [39] precisely identifies the optimal number of blocks, thereby ensuring a configuration that maximizes inference performance.

In contrast to above search spaces, LarKNAS considers that when the proposed model becomes large-scale, how to design a candidate space with small search cost and efficient modules for ConvNets. A phased stage architecture is tailored for large-scale models. ULarK Block integrates with forming four modern structures to efficiently reduce the search cost and improve the representation capability.

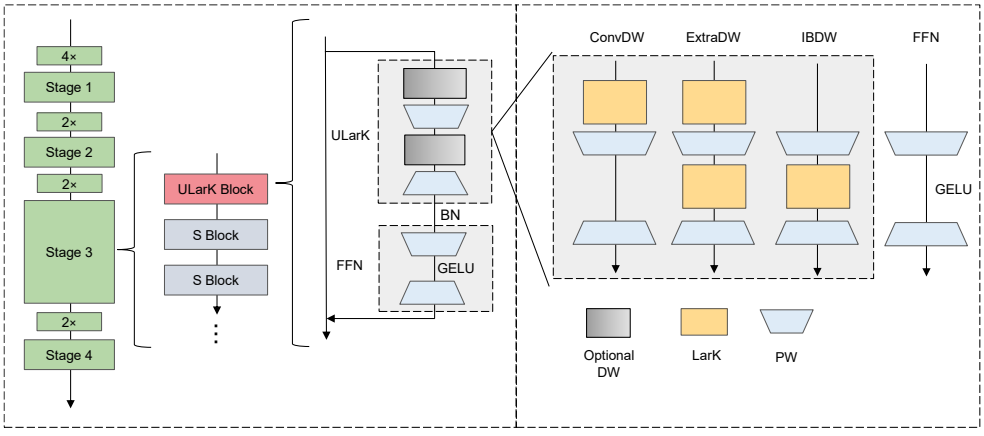
3. Method

The method of LarKNAS is outlined in this section divided into two main components. The first component (Sections 3.1 and 3.2) is a large convolution kernel search space, referred to as LarKSpace. The second component (Section 3.3) produces the algorithm pipeline of LarKNAS. The following issues are considered during the construction process for LarKSpace:

Figure 2 The graphic abstract of our proposed LarKNAS.
(a) The architecture design of LarKSpace. It features ULarK Blocks (Universe Large Kernel Blocks), comprising a ULarK Convolution, a Feed Forward Network (FFN), and Batch Normalization (BN) layers. The main difference between Small Kernel Blocks (S Blocks) and ULarK Blocks is that S Blocks use a depth-wise 3×3 convolution layer, while ULarK Blocks use Dilated Convolution. Stages connect via down-sampling blocks.
(b) ULarK Blocks Design. It comprises optional Depth wise (DW) and Pointwise (PW) convolutions, forming four structures based on Optional DW variations: ConvDW (ConvNeXt), ExtraDW, IBDW (Inverted Bottleneck), and FFN, balancing simplicity and efficiency in convolution design.

Q1: How to design the overall model architecture?
Q2: How to design the large convolution?

In response to Q1, Section 3.1 presents a phased architectural design for blocks and convolution kernels; in response to Q2, Section 3.2 discusses the search settings for sparse large convolution kernels. Subsequently in Section 3.3, The pipeline of LarKNAS is summarized in three steps and the visual backbone LarKNASNet family is searched through the pipeline.



3.1. Model Architecture of LarKSpace

This section introduces the design of the model architecture of LarKSpace, focusing on two primary considerations: the phased architecture design and the optimization of module types. First, a phased stage architecture design with "small-large-medium" convolution kernel is employed to achieve architectural scalability and preserve the representational capacity of

large ConvNets. Furthermore, a unified large convolution kernel representation module, referred to as the ULarK Block, is proposed to enhance the search efficiency of the convolution modules. Additionally, the ConvFFN module is utilized to increase the depth of model and improve the capability in processing features at various levels.

Phased stage architecture design. The staged design offers distinct advantages, as it effectively accommodates varying levels of feature processing requirements. In the design of early convolutional neural networks, the block-based search space typically adhered to the design principles of EfficientNet [9], which demonstrates superior performance when the model size is small. However, as the model size increases, the differences in feature processing capabilities among various modules become more pronounced, rendering a single modular design inadequate. Consequently, a phased design has become a necessary choice to adapt to the increasing model size and the diversification of feature processing needs. LarKSpace incorporates large-scale convolutional design concepts from the 2020s [7, 25] and adopts a four-stage configuration. The basic architecture of LarKSpace, illustrated in Figure. 2(a), divides the model into four stages, each comprising ULarK blocks and S blocks. Each stage is interconnected by a down-sampling block with a stride of 2, which reduces the input image feature size by half and doubles the number of channels compared to the previous stage (2×). The staged design of LarKSpace aims to address varying feature levels and facilitate hierarchical feature transfer through down-sampling blocks.

Range of Large Kernels. The variation range of the convolution kernel in four stages is considered from the maximum value and the staged setting variation. As illustrated in Table 1, the architecture of LarKSpace is divided into four stages: from N1 to N4. The range of kernel size for each stage is determined by following considerations:

First, establish the maximum size of the convolution kernel. The expansion of receptive fields is crucial for improving model performance. However, the larger convolution kernels lead to limited additional gains in performance. A plausible explanation is that, while using very large kernels can expand the receptive field, it may not be able to maintain desirable locality. Since the stem layers of standard ResNet [13] and ConvNeXt [7] cause a 4× down sampling of the input image, for typical 224×224 ImageNet images, a kernel size of 51×51 is already close to the size of global convolution. Although PeLK [26] attempted 101×101 kernels and achieved good classification results, this was actually achieved through the splicing of four parts, without gaining a larger effective receptive field. For the ImageNet classification task, larger kernels may not have a greater impact on feature processing. Therefore, the search for convolution kernel sizes from N1 to N4 is conducted between 3×3 and 51×51.

Next, determine the range of convolution kernel sizes for each stage. The necessity and rationale for using large kernels at each stage are evaluated. Typically, early stages learn low-level features such as edges and angles [40], hence large kernels like 31×31 or 51×51 may be unnecessary [23]. Additionally, it was observed that variations in multiples of the convolution kernel size more significantly impact the convolution effect than numerical differences [26]. For instance, the distinction between 13×13 and 17×17 kernels are minimal. In contrast, the difference between 3×3 and 7×7 kernels can yield substantial improvements. This phenomenon has been corroborated by numerous large convolutional models [24, 25, 26]. To facilitate the search, the kernels are categorized into small, medium, and large based on their receptive fields for different stages of LarKSpace. *Small kernels in first stage (N1) employ 3×3 convolutions. Medium kernels in final stage (N4) utilize 7×7 and 13×13 convolutions. Large kernels in middle stage (N2 and N3) apply 21×21, 31×31, and 51×51 convolutions.*

Table 1 Kernel size of LarKSpace in four stages after the design of Phased stage architecture design and Range of Large Kernels.

Stage	Kernel size
N1	3
N2	21,31,51
N3	21,31,51
N4	7,13

ULarK Block. In the context of module type optimization, a significant trade-off exists between the complexity and effectiveness of the search space. A limited search space arises with only a single type of convolutional layer, such as the MBConv search space [10]. Alternatively, incorporating ConvNeXt blocks, inverse bottleneck (IB) blocks, and other designs directly into the search space results in excessive size, thereby increasing search complexity. A unified module adaptable for search is required to express the above designs. Consequently, the Universal Large Kernel Block (ULarK Block) is proposed to integrate prevalent optimization designs from contemporary convolutional models, drawing inspiration from MobileNetv4 [39]. ULarK Block decomposes four modules (ConvNeXt, ExtraDW, IBDW and FFN) to the combine of pointwise (PW) and depth-wise (DW) convolution, as illustrated in Figure 2(b). The structure comprises two ULarK convolutional layers, each with two optional DWs. The first ULarK layer offers ConvNeXt, ExtraDW, and IBDW options. The second ULarK layer defaults to nullifying all DWs, functioning as a feedforward network (FFN). The ULarK Block offers an adjustable and flexible structure, suitable for efficient network design without significantly increasing search complexity.

ConvFFN. In addition to providing sufficient receptive fields, representational capability is closely linked to the depth of the network. To enhance nonlinear computation and inter-channel computation capabilities, we employ S blocks to augment network depth. The S block primarily uses 3×3 convolutional layers, with the remaining structure consistent with that of the ULarK Block. Following the convolution operation, a feature processing module ConvFFN is included, with a design similar to RepLKNet [24] and InternImage [22], comprising convolutional layers and a feedforward network (FFN) [7, 25] with GRN[41] units. The FFN is widely utilized in Transformers and MLPs, and we incorporate a similar convolutional FFN block as the feature processing module, which includes BN layers [42], 1×1 PW layers, and GELU activation functions. ConvNeXt integrates LayerNorm(LN) [43] from Transformers into convolutional layers, which enhances performance; however, experiments conducted by RepLKNet demonstrate the effectiveness of BatchNorm (BN) as well. In this approach, we apply BN after the FFN, allowing for effective integration into the second layer of the FFN.

Based on the above designs, a series of models with varying parameter counts are instantiated: a) Baseline architecture LarKNASNet-T; b) Lightweight architecture LarKNASNet-A/S achieved by reducing channels and depth; c) Deep architecture LarKNASNet-B developed by increasing depth; d) Wide architecture LarKNASNet-L created by extending width based on LarKNASNet-B.

The specific designs for the various parameter quantity models of LarKNASNet are presented in Table 2, with the number of blocks in the four stages denoted by N (N1, N2, N3, N4). Following the methodology of ConvNeXt-T, the baseline architecture of LarKNASNet-T has a channel count of C=96, and the block ratio is set as N= (3,3,9,3). To summarize, each stage is defined by four architectural hyperparameters: the number of blocks N, the channel dimension C, the kernel size K, and the configuration of the ULarK Block T. The network structure of LarKSpace is represented as [N, C, K, T], where N and C are fixed, while K and T are searchable parameters.

Table 2 The architecture hyperparameter settings of LarKNASNet include the number of blocks N1, N2, N3, N4 in the four stages and the number of channels (C) in the first stage ranged in Atom(A), Small(S), Tiny(T), Base(B) and Large(L). F refers to FLOPs_max(G), the constraint on the maximum computational load when searching for this model. The first stage uses only S Block, while the second and fourth stages use only ULarK Block.

Model Size	N1	N2	N3	N4	C	F
LarKNASNet-A	2	2	6	2	32	1
LarKNASNet-S	2	2	6	2	64	3
LarKNASNet-T	3	3	9 + 9	3	96	5
LarKNASNet-B	3	3	9 + 18	3	128	20
LarKNASNet-L	3	3	9 + 18	3	256	100

3.2. Sparse large convolution kernel

The large convolutional kernel captures long-distance dependencies for the image, which is essential for obtaining global contextual information. However, as the size of the convolution kernel increases, both computational and storage costs escalate quadratically, leading to decreased efficiency. When expanding the convolution kernel to improve the performance of convolutional models, it is crucial to consider parameter usage efficiency and GPU utilization. Four convolution types designed to expand parameters are summarized as shown in Figure 3.

Utilization of dense convolution as Figure 3(a), kernel sizes increase, the number of parameters grows quadratically, but improvements in model accuracy remain minimal. Dilated convolution as Figure 3(b) enhances effective receptive field by introducing gaps within the kernel. However, significantly increasing the dilated convolution kernel size (e.g., beyond 31×31) yields limited performance gains. Sole reliance on the dilation rate may result in the model overlooking critical regions within the input image. Modern convolutional architectures introduce two strategies to optimize the allocation of parameters: re-

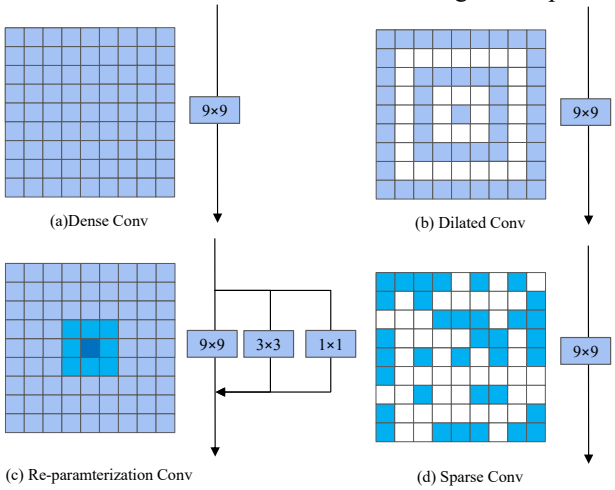


Figure 2 Two classic convolutional designs as a), b) and two modern convolutional designs as c), d).

parameterized kernels and sparse kernels. These modern kernel types facilitate parameter reallocation from large convolution kernels. The objective is to assign greater weights to influential areas, such as centers or corners. The key distinction lies in the presence of branches for re-parameterizing large kernels as Figure 3(c) during the search phase, which may lead to GPU memory overflow during extensive searches. In contrast, sparse convolution as Figure 3(d) reduces GPU usage and improves computational efficiency by executing a limited number of convolution operations on the input image. This strategy is particularly relevant for large kernels that inherently require more GPU memory resources. A sparse large kernel design is employed to enhance both parameter efficiency and GPU memory usage during the search phase.

Algorithm 1 Pseudo code of search sparse large convolution kernel

Input: θ Dense network parameters, ΔT Adaptation frequency
Constraints: Initialize sparse matrix M_0, GPU memory and FLOPs limitations
Output: s* Optimal sparse degree
1: Initialize M_0 with SNIP
2: Initialize $s_0 = 0.1$
3: for each s in [0.1, 0.2, 0.4, 0.6, 0.8] do
4: $\theta s = M_0 * \theta$
5: Train θs with AdamW
6: for each ΔT do
7: *//Sparse Weight Adaptation*
8: Pruning θs by p%
9: Update p with cosine decay
10: Evaluate θs
11: end for
12: end for
13: Apply s* to final model

The sparsity of the sparse large kernel is selected as shown in Algorithm 1, which searches for the optimal sparsity with SNIP [44] among the predefined five values by dynamically adjusting the sparsity to find the best neural network performance within the computational limits. It is first executed to obtain preliminary optimization results with s = [0.4, 0.6] in LarKNASNet-B/L models, and these results are used as hyperparameters for the next search step.

Implementation details of Algorithm 1 are elaborated as follows. (1) Sparse mask update: after SNIP [44] initialization, the sparse mask M is updated by retaining the top-k elements based on gradient magnitude, where k is determined by the current sparse degree s. The mask is kept fixed within each adaptation cycle and refreshed at the beginning of the next cycle. (2) Pruning percentage p is initialized to a small value and decayed via a cosine schedule over the adaptation frequency F , ensuring gradual sparsification. (3) Stage-wise sparsity degree s is searched independently for each stage, as different stages exhibit distinct feature processing requirements (Section 3.1). Stages N1 and N4, which use small/medium kernels, adopt a default dense configuration (s = 0.1), while stages N2 and N3, which employ large kernels, are searched over the full range [0.1, 0.2, 0.4, 0.6, 0.8]. (4) Final sparsity selection: when multiple stages and kernel sizes are involved, the final sparse degree s* for each stage is selected as the value that maximizes the gradient estimation score (Equation 4) under the FLOPs and GPU memory constraints. The overall sparsity configuration is the combination of per-stage optimal s* values, which is then applied to the final model for retraining.

3.3. Pipeline of LarKNAS

LarKNAS is outlined, including construct initial architecture, gradient estimation search and retrain. The candidate subnetwork model of LarKSpace is constructed based on architectural parameters derived from sections 3.1 and 3.2. Then the high-performance candidate subnetwork is searched by taking gradient estimation as the optimization objective for the iterative of a batch. After the search process is completed, LarKNASNet is acquired via retraining. The detailed pseudocode provided in Algorithm 2. Detailed steps are described following.

Construct initial architecture. Initially, LarKSpace is constructed, which includes architecture parameters defined by [N, C, K, T, s]. The stages N, channels C are fixed as Table 2 and the variation ranges of the kernel sizes K, block selection T, sparse degree s are as designed in Section 3.1 and 3.2. Both architecture and weight parameters are randomly initialized, resulting in the construction of a sub-network within LarKSpace. Subsequently, a commonly utilized warm-up training technique [45] is employed to preliminarily train the weights, enhancing the accuracy for the search method in the next step. For each architecture in LarKSpace, determine the average FLOPs.

Gradient estimation search. After the construction of LarKSpace, a gradient estimation search for large-scale convolutional models is developed as follows. Gradient-based search strategies [46, 47, 48] are frequently employed due to their efficiency. The optimal architecture for general gradient-based search can be formulated as

$$S^* = \arg\min_S L_{val}(w^*(\alpha), \alpha), \qquad (1)$$

where S^* is the best candidate of the sub-networks S of LarKSpace, and $L_{val}(w^*(\alpha), \alpha)$ is the validation set loss including architecture parameters α and weight parameters $w^*(\alpha)$. $w^*(\alpha)$ changes with α , and gradient-based methods represented by DARTS [46] optimize w and α simultaneously through approximate bi-level optimization. The gradient iteration process can be expressed as:

$$\nabla_\alpha L_{val}\left(w^*(\alpha), \alpha\right) \approx \nabla_\alpha L_{val}\left(w - \xi \nabla_w L_{train}(w, \alpha), \alpha\right), \quad (2)$$

where $\nabla_\alpha L_{val}$ is iteration of α in the validation set, $\nabla_w L_{train}$ is iteration of w in the training set and ξ is learning rate.

The approximation in Equation 2 requires alternating iterations and gradient backpropagation, which remain time-consuming for large-scale models. Furthermore, softmax normalization cannot be effectively implemented when multiple architecture parameters [K, T, s] are present.

The gradient estimation search method is proposed to address the problem described in Equation 1. High-performance models can be achieved by estimating the performance of candidate architectures within a batch gradient. This improvement is inspired by an approximation within a gradient iteration in Equation 2. The gradient estimation search method eliminates the need for gradient backpropagation, thereby enhancing efficiency. To evaluate and select high-performance subnetworks, the absolute mean value and standard deviation of gradients are introduced as key indices. In recent NAS studies [49, 50], it has been suggested that a high-performance subnetworks with high training convergence speed and strong generalization capability tends to exhibit high absolute mean values of the gradient for input batches. Meanwhile, low standard deviation values of the gradients of its parameters are observed across diverse batches. For the m-th subnetwork, the mean $\mu_\nabla^{(m)}$ and standard deviation $\sigma_\nabla^{(m)}$ of the absolute value of its gradient are defined as follows:

$$\mu_{\nabla}^{(m)} = \frac{1}{N}\sum_{i=1}^{N}|\nabla_{\omega} L(\mathrm{S}_m;\mathrm{D}_i)|,$$
$$\sigma_{\nabla}^{(m)} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(|\nabla_{\omega} L(\mathrm{S}_m;\mathrm{D}_i)| - \mu_{\nabla}^{(m)}\right)^2}, \quad (3)$$

where $L(\mathrm{S}_m;\mathrm{D}_i)$ represents the loss function of the subnetwork for the i-th batch D_i.

By maximizing $\mu_{\nabla}^{(m)}$ and minimizing $\sigma_{\nabla}^{(m)}$ in Equation 3, we can effectively filter out subnetworks that exhibit both high performance and stability during training. The objective is to optimize subnetwork selection while adhering to computational resource constraints. Specifically, we aim to maximize the expected performance of the subnetwork, subject to limits on computational complexity (measured in FLOPs) and GPU memory usage, both of which must remain within a predefined budget. The optimization objective can be formalized as

$$\begin{aligned} \mathrm{S}^* &= \arg\max_{\mathrm{S}} \lambda_1 \cdot \mu_{\nabla}^{(\mathrm{S})} - \lambda_2 \cdot \sigma_{\nabla}^{(\mathrm{S})} \\ \text{subject to} \quad & \mathrm{FLOPs}(\mathrm{S}) \le \mathrm{FLOPs}_{\mathrm{budget}}, \\ & \mathrm{Memory}(\mathrm{S}) \le \mathrm{Memory}_{\mathrm{budget}}, \end{aligned} \quad (4)$$

where λ_1 and λ_2 are hyperparameters employed to balance the contributions of the gradient mean and standard deviation. The constraints $\mathrm{FLOPs}(\mathrm{S})$ and $\mathrm{Memory}(\mathrm{S})$ ensure that the computational complexity and memory usage of the selected subnetworks remain within the specified budget. Then the architecture parameters are iteratively updated and the optimal architecture is selected by comparing S^*.

Retrain. The architecture parameters denoted by [K, T, s] are applied to instantiate the model LarKNASNet, which is then retrained to fine-tune the weight parameter.

Algorithm 2 Pseudo code of LarKNAS

Input: LarKSpace [N, C, K, T, s]
fix N, C as Table 2 and search K, T, s,
warm epoch n_1 and search epoch n_2
Output: Optimized [N, C, K, T, s] and weights w
// construct initial architecture
1: Initialize [K, T, s] and w
2: if N<n_1
3: Train and update w of best Acc
4: end if
5: Calculate FLOPs_avg
//gradient estimation search
6: if N<n_2
7: Calculate S^* via Equation 4
8: Update K, T, s of best S^*
9: end if
//retrain
10: Retrain and update w of best Acc
11: Return K, T, s, and weights w

4. Experiments

In this section, LarKNASNet-A/S/T is first compared with other NAS methods in terms of accuracy and search cost. Then, experiments on various essential vision tasks are conducted to evaluate LarKNASNet-T/B/L against state-of-the-art baseline models, as common NAS methods often encounter performance-time bottlenecks with further increases in model parameters.

4.1. Comparison with NAS methods

The search results for the small-scale models LarKNASNet-A/S/T are trained and evaluated on the ImageNet-1K classification task as shown in Table 3.

LarKNASNet-A/S/T all achieved high accuracy ranging from 76.3% to 82.5% of different parameters. LarKNASNet-A exhibits search cost comparable to DARTS while achieving an accuracy that is 3% higher. This performance is significantly higher than NASNet [36] by 2.3% and slightly lower than EfficientNet-B0 family [9] with multiplier level reduced search cost. To be specific, it reduces search time by 100× to 1000× in contrast to other NAS based on evolutionary algorithms or reinforcement learning approaches like NASNet [36], BigNAS-XL[8] and others, thereby demonstrating the efficiency of LarKNAS.

It is worth further discussing the cost-benefit tradeoff of LarKNAS. The 96 GPU days search cost for LarKNASNet-L (335M parameters) is a one-time investment: once the architecture is searched, it can be directly reused and transferred to downstream tasks without repeated search. Compared with ScaleNet [20] (270+ GPU days for a similar scale model), LarKNAS reduces the search cost to 36%. Furthermore, compared with one-shot NAS methods such as OFA [51] (1486 GPU days) and BigNAS [8] (1228 GPU days), LarKNAS achieves a significantly lower search cost while exploring a fundamentally different search space (large kernels vs. small kernels). The primary value of LarKNAS lies not in surpassing hand-designed models by a large margin, but in demonstrating that automated NAS can discover large-kernel architectures comparable to expert-designed ones (e.g., RepLKNet, SLaK, PeLK), thereby validating the effectiveness of the large kernel search space as a new research direction for NAS.

Table 3 **NAS methods on ImageNet-1k Classification**. The small models LarKNASNet-A/S/T, are compared with other NAS method divided by the Params. In Table 3 and subsequent tables, the metrics highlighted in bold indicate superior performance compared to all methods presented.

Method	Params (M)	FLOPs (G)	Acc (%)	Search Cost (GPU days)
NASNet[36]	5.3	0.56	74.0	2000
EfficientNet-B0[9]	5.3	0.39	77.1	1714
DARTS[46]	4.7	0.52	73.3	0.4
LarKNASNet-A	5.3	0.92	76.3	**0.4**
OFA-L[51]	9.1	0.60	80.0	1486
BigNAS-XL[8]	9.5	1.04	80.9	1228
LarKNASNet-S	9.3	2.8	80.4	**2**
FBNetV2[17]	29.2	21	77.2	1633
EfficientNet-B4[9]	19	4.2	82.6	1756
LarKNASNet-T	33	4.6	82.5	**13**

4.2. Comparison with the visual backbone models

To thoroughly assess the effectiveness of LarKNAS, we compare it with several state-of-the-art baselines across a diverse range of tasks, including ImageNet-1K classification [52], object detection on COCO [53], and semantic segmentation on ADE20K [54].

ImageNet-1K classification. Training experiments were conducted on ImageNet-1K to evaluate the fundamental visual capabilities of the model. The training configurations adhere to common practices [7, 23]. The training utilized the Timm library [45], while the recently developed EfficientTrain++ [57] was employed to optimize GPU memory usage for an input resolution of 384 and 448. More experimental settings are in the Supplementary Material. Consistent with ConvNeXt [7], the searched model LarKNASNet was retrained using a widely adopted method consisting of 300 training epochs on ImageNet-1K. Table 4 presents the top-1 accuracy on the ImageNet-1K validation set. Results were organized into multiple sections based on model size for enhanced clarity.

Table 4 **Classification accuracy on ImageNet-1k-only**. Type： T/C/RC/SC denote Transformer/ConvNet/Sparse ConvNet/ Re-parameterization ConvNet.

Model	Type	Input size	Params (M)	FLOPs (G)	Acc (%)
Swin-T[16]	T	224^2	28	4.5	81.3
ConvNeXt-T[7]	C	224^2	29	5.6	82.6
SLaK-T[25]	SC	224^2	30	5.0	82.5
PeLK-T[26]	SC	224^2	29	5.6	82.6
UniRepLKNet-T[23]	RC	224^2	31	4.9	83.2
LarKNASNet-T	SC	224^2	33	4.6	**83.3**
DeiT-B/16 [55]	T	224^2	87	17.6	81.8

Swin-B[16]	T	224^2	88	15.4	83.5
RepLKNet-31B[24]	RC	224^2	79	15.3	83.5
ConvNeXt-B[7]	C	224^2	89	15.4	83.8
SLaK-B[25]	SC	224^2	95	17.1	84.0
PeLK-B[26]	SC	224^2	89	18.3	84.2
LarKNASNet-B	SC	224^2	95	19.8	**84.2**
ViT-B/16[15]	T	384^2	87	55.5	77.9
DeiT-B/16[55]	T	384^2	87	55.4	83.1
Swin-B [16]	T	384^2	88	47.1	84.5
RepLKNet-31B[24]	RC	384^2	79	45.1	84.8
ConvNeXt-B[7]	C	384^2	89	45.0	85.1
SLaK-B[25]	SC	384^2	95	50.3	85.5
PeLK-B[26]	SC	384^2	89	54.0	**85.6**
LarKNASNet-B	SC	384^2	95	58.9	85.4
MAE-ViT-H[56]	T	448^2	656	-	**87.8**
LarKNASNet-L	SC	448^2	335	208	86.6

LarKNASNet demonstrates superior performance by comparable or modest top-1 accuracy across various parameter scales in comparison to recent representative vision backbone networks such as ViT [15], Swin-Transformer [16], UniRepLKNet [23], SLaK [25], PeLK [26] and so on shown in Table 4. 1) LarKNASNet-L achieves an accuracy of 86.6% on ImageNet-1K-only, outperforming the results reported by ConvNeXt [7] and Swin-B [16] by about 1%, while lower than that of MAE-ViT-H [56], yet it utilizes only half the number of parameters. 2) LarKNASNet-B achieves greater accuracy than state-of-the-art Transformers, such as Swin-Transformer [16] by 0.9% and DeiT-B [54] by 2.3%, and significantly outperforms ViT-B/16 [15] by 7.5% on 384×384 inputs, all without the use of complex attention modules or patch embedding. 3) LarKNASNet-T is 0.7% higher than ConvNeXt-T (7×7 kernel size) [7] and LarKNASNet-B is 0.3% higher than ConvNeXt-B, emphasizing the advantages of employing large kernels during the image classification task. 4) LarKNASNet demonstrates superior or comparable performance compared to Sparse Large ConvNet (SC) represented by SLaK [25] and Re-parameterization Large ConvNet (RC) as RepLKNet [24]. Improved accuracy (0.2%~0.8%) is achieved through the search of kernel and module optimizations at these parameter scales.

Regarding the input resolution difference, LarKNASNet-L adopts a 448×448 input resolution because large convolution kernels require higher resolution to fully exploit their effective receptive field (ERF) advantage, as analyzed in Section 5.3. Under the same 448×448 resolution, LarKNASNet-L (335M parameters) achieves 86.6% accuracy, which is competitive with MAE-ViT-H (656M parameters, 87.8% accuracy) while using only 51% of the parameters, demonstrating superior parameter efficiency. The resolution choice is consistent with common practice for large-kernel ConvNets [24, 25, 26].

Large kernel ConvNets should be evaluated on downstream tasks, as accuracy derived solely from ImageNet may not fully reflect their potential [24]. Therefore, in addition to the ImageNet-1K accuracy after 300 training epochs, the trained model will be transferred to ADE20K [54] using UPerNet [58] to assess its performance in semantic segmentation, and the single-scale mIoU will be reported after the standard fine-tuning process (160k iterations) [59].

COCO Object Detection. LarKNASNet will be utilized as the backbone for the Cascade Mask R-CNN [60, 61], employing a standard training configuration of 3x (36 epochs) with MMDetection [62]. The batch size for all models is set to 16, and the AdamW [63] optimizer will be employed, with an initial learning rate set to 1×10^{-4}.

Table 5 **COCO Validation Set Object Detection**. The FLOPs are measured using an input of 1280×800.
* Denotes pretrained on ImageNet-22K.

Model	Params (M)	FLOPs (G)	AP^{box}	AP^{mask}
Swin-T[16]	86	745	50.5	43.7
ConvNeXt-T[7]	86	741	50.4	43.7
SLaK-T[25]	-	-	51.3	44.3
PeLK-T[26]	86	770	51.4	44.6
LarKNASNet-T	89	785	**51.9**	**44.8**
Swin-B[16]	145	982	51.9	45.0
RepLKNet-B[24]	137	965	52.2	45.2
SLaK-B[25]	152	1001	52.5	45.5
ConvNeXt-B[7]	146	964	52.7	45.6
PeLK-B[26]	147	1028	52.9	45.9
LarKNASNet-B	152	1138	**53.1**	**46.1**
Swin-L*[16]	253	1382	53.9	46.7
ConvNeXt-L*[7]	255	1354	54.8	47.6
LarKNASNet-L	386	1675	**55.2**	**48.3**

As shown in Table 5, 1) Our proposed LarKNASNet exceeds previous state of-the-art models with remarkable

improvements, demonstrating the generalization of the model searched out by LarKNAS. 2) The COCO detection model of LarKNASNet-L contains more parameters than Swin-L due to its transfer from the ImageNet-1K pre-trained model. Notably, LarKNASNet-L demonstrates strong scalability, as the APbox and APmask of LarKNASNet-L, without ImageNet-22K pre-training, surpass those of Swin-L pre-trained on ImageNet-22K by 1.3% and 1.6%, respectively.

Regarding parameter efficiency in downstream tasks, LarKNASNet-L contains more parameters than Swin-L (386M vs. 253M) because the backbone itself is designed at a larger scale to leverage large kernels. However, the APbox and APmask improvements (1.3% and 1.6% over Swin-L with ImageNet-22K pretraining) indicate that the additional parameters translate to meaningful performance gains rather than redundancy. The parameter increase primarily comes from the large kernel layers, which are essential for capturing long-range dependencies in detection and segmentation tasks.

ADE20K Semantic Segmentation. The pre-trained LarKNASNet will serve as the backbone for UPerNet [58] on ADE20K [54], following a standard training scheme of 160k iterations with MMSegmentation [59]. Initialization will use pre-trained classification weights. The initial learning rate is set to 1×10^{-4} and is decreased according to a polynomial decay schedule with an exponent of 1.0. In accordance with previous studies [7, 16], a cropping size of 512 is used for ImageNet-1K pre-trained models, while a size of 640 is used for ImageNet-22K pre-trained models. The batch size for all segmentation models is 16, and training will be conducted for 160k iterations.

Table 6 reports the mIoU on the validation set. 1) Our proposed LarKNASNet exhibits strong generalization performance on the semantic segmentation task, achieving a commendable mean Intersection over Union (mIoU). It demonstrates a larger advantage at 640×2560 resolution compared to 512×2048. LarKNASNet-L without additional pretraining, outperforms Swin-L [16] by 1.3% in mIoU and matches the performance of ConvNeXt-L [7], both of which are pretrained on ImageNet-22K. 2) LarKNASNet with large receptive fields might be more effective for semantic segmentation tasks than Swin-Transformer [16], and this trend is even more pronounced at higher resolutions. In terms of computational efficiency, LarKNASNet-B is larger parameters than Swin-B with comparable FLOPs on the segmentation task. 3) Additionally, LarKNASNet-B optimizes the size of convolution kernels at different stages, achieving results comparable to a 101×101 convolution size PeLK-B [26].

Table 6 **ADE20K Validation Set Semantic Segmentation**. The FLOPs are measured using inputs of 512×2048 or 640×2560 based on the crop size of single scale testing.
* Denotes pretrained on ImageNet-22K.

Model	Kernel size	Params (M)	FLOPs (G)	mIoU (%)
Crop size=512×2048				
Swin-T[16]	N/A	60	945	44.5
ConvNeXt-T[7]	7-7-7-7	60	939	46.0
SLaK-T[25]	51-49-47-13	64	957	47.6
PeLK-T[26]	51-49-47-13	62	970	48.1
UniRepLKNet-T[23]	3-13-13-13	61	946	48.6
LarKNASNet-T	3-31-31-7	63	942	**48.7**
Swin-B[16]	N/A	121	1188	48.1
ConvNeXt-B[7]	7-7-7-7	122	1170	49.1
RepLKNet-B[24]	31-29-27-13	112	1170	49.9
SLaK-B[25]	51-49-47-13	131	1210	50.2
PeLK-B[26]	51-49-47-13	126	1237	50.4
PeLK-B-101[26]	101-69-67-13	126	1339	**50.6**
LarKNASNet-B	3-31-51-13	131	1342	50.5
Crop size=640×2560				
Swin-B*[16]	N/A	121	1841	50.0
ConvNeXt-B*[7]	7-7-7-7	122	1828	52.6
Swin-L*[16]	N/A	234	2468	52.1
ConvNeXt-L*[7]	7-7-7-7	235	2458	53.2
LarKNASNet-L	3-51-51-13	364	2345	**53.4**

5. Discussions

This section conducts ablation experiments on architectural design and large convolution kernels for Q1 and Q2 proposed in Section 3, separately. Additionally, it analyzes the enhancement of visual properties in LarKNAS.

5.1. Ablation on architectural design

ULarK Block. Ablation experiments on the ULarK Block design demonstrate that the ULarK Block reduces search time. The search cost with ('w/') and without ('w/o') ULarK Block is tested in different sizes of LarKNASNet and reported in Figure 4. ULarK Block decreases the search time from different model sizes, despite achieving only a 4.76% reduction rate for small-scale models like LarKNASNet-A. However, the reduction rate of 23.81% is substantial for LarKNASNet-L from 126 GPU days to 96 GPU days.

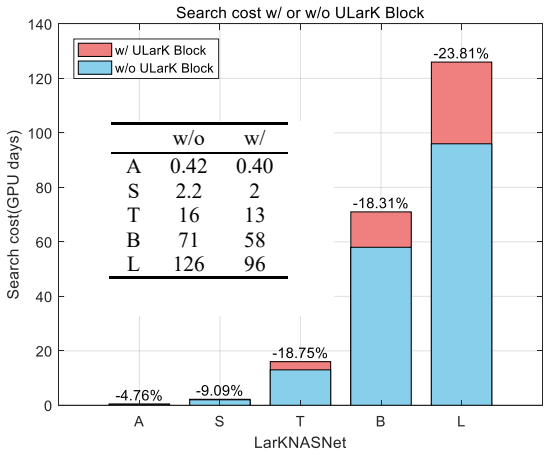


Figure 3 The search cost (GPU days) and reduction rate of 'w/o' and 'w/' ULarK Block with different Model Size of LarKNASNet. The specific search cost is shown in the inner table.

ConvFFN. Ablation experiments on the ConvFFN design indicate that, with the same number of parameters, the configuration using GELU and BN achieves the highest accuracy. Table 7 presents the precise numerical changes in LarKNASNet-T after incorporating various modules of increasing depth. The accuracy of both the Bottleneck and SE-Block modules is lower, as these components possess a diminished capacity for multi-channel feature processing when compared to FFN and ConvFFN. Although ConvFFN demonstrates similar accuracy, it has fewer parameters than FFN, attributing this efficiency to the ability to fuse Batch Normalization (BN) into the subsequent FFN.

Table 7 LarKNASNet-T models with different effective additional structures to increase depth. We report ImageNet accuracy (Acc), number of parameters, and FLOPs are reported.

Depth structures	Acc (%)	Params(M)	FLOPs(G)
Bottleneck	81.9	29	4.97
SE-Block	82.5	28	4.57
FFN(GELU+LN)	83.2	39	4.63
ConvFFN(GELU+BN)	83.3	33	4.67

Stages for kernel sizes. The ablation study on the design stages of convolution kernel size indicates that the "small-medium-large" design effectively increases inference throughput of images on a large scale ConvNet. Throughput is a critical factor to consider for inference speed when implementing large convolution kernel models [23]. Utilizing optimizations of PyTorch library [24], the previous works [25, 26] have scaled to parameters of Tiny model. The real inference throughput are we measured with an A100 GPU in FP32 precision as shown in Table 8. LarKNASNet-T achieves the highest throughput with an input resolution of 224×224. Then, with elevated parameters for the Base model at an input resolution of 384×384, it is indicated that the throughput optimization of SLaK-B [25] becomes less pronounced. Consequently, the "small-medium-large" design of the kernel size as LarKNASNet-B is more effective than SLaK-B for the inference ability of large-scale convolution models.

Table 8 Inference throughput comparison on ImageNet-1K.

Models	Input Size	Kernel Size	Throughput (img/s)
SLaK-T	224^2	51-49-47-13	754
PeLK-T	224^2	51-49-47-13	1138
PeLK-101-T	224^2	101-69-67-13	1077
LarKNASNet-T	224^2	3-31-31-7	**1385**
SLaK-B	384^2	51-49-47-13	567

LarKNASNet-B	384^2	3-31-51-13	**684**

5.2. Ablation on large convolution kernels

Re-parameterization large ConvNet (RC) [33] is another crucial approach for optimizing large kernels, although it may not be suitable for NAS, as discussed in the previous section. Here, we present analytical experiments conducted on the LarKNASNet-B search, with results summarized in Table 9. 1) Regardless of whether Sparse large ConvNet (SC) are applied, RC can significantly increase peak GPU memory usage, potentially interrupting the search process. This disruption may occur due to the multiple branches created during both the search and training phases, which consume GPU memory, while the architectural optimizations associated with re-parameterization primarily occur during the inference phase. 2) Without applying SC, the accuracy on ImageNet decreases by 0.8%, and the mean mIoU drops significantly by 2.7%. This decrease may be attributed to the tendency to favor smaller kernels or feedforward networks (FFN) after employing dense convolutions, which reduces effective receptive field. Additionally, both search cost and peak GPU memory usage increase, highlighting the effectiveness of SC in managing GPU memory and streamlining the search process.

Table 9 Ablation on Sparsity and Re-parameterization. We compare the change of LarKNASNet-B after removing or adding the technique. RC/SC denote Re-parameterization ConvNet/ Sparse ConvNet. OOM denotes out of GPU memory (24G).

SC	RC	ACC (%)	mIoU (%)	Search Cost (GPU days)	Peak GPU memory (GB)
w/	w/o	84.2	50.5	58	17.4
w/	w/	-	-	-	OOM
w/o	w/	-	-	-	OOM
w/o	w/o	83.4	47.8	78	23.5

5.3. Ablation on search components

Component-wise ablation. To isolate the contribution of each architecture parameter, additional ablation experiments are conducted on LarKNASNet-T, as presented in Table 12. Searching only the kernel size K (with block type T and sparsity s fixed) yields 82.9% accuracy, contributing the majority of the performance gain. Searching only the block type T yields 82.7%, and searching only the sparsity s yields 82.6%. The full search over K, T, and s jointly achieves 83.3%, demonstrating that the three parameters are complementary and their combination leads to the optimal architecture. The kernel size K is the most impactful single factor, consistent with the observation that large kernels are the key design dimension for modern ConvNets [24, 25, 26].

Comparison with random and grid search. To demonstrate the necessity of the proposed gradient estimation search, we compare it with random search and grid search under the same search budget (13 GPU days for LarKNASNet-T). The results are summarized in Table 12. Random search samples 50 architectures uniformly from LarKSpace and trains each for a short period; grid search enumerates kernel size combinations on a coarse grid. The gradient estimation search achieves 82.5% accuracy, outperforming random search (81.4%) by 1.1% and grid search (81.8%) by 0.7%. This confirms that the gradient-based selection criterion effectively identifies high-performance subnetworks that random sampling cannot discover within the same budget. Additionally, we apply the gradient estimation search on a conventional small-kernel search space (3×3 and 5×5 only), which yields 81.9% accuracy, lower than the 82.5% achieved on LarKSpace, validating that the large kernel search space itself contributes to the performance improvement.

Hyperparameter sensitivity. The sensitivity of the balancing hyperparameters lambda1 and lambda2 in Equation 4 is analyzed. Experiments are conducted on LarKNASNet-B with lambda1, lambda2 sampled from {0.25, 0.5, 1.0, 2.0}. The results show that the searched architecture is robust to lambda1 and lambda2 within the range [0.5, 2.0], with accuracy variation less than 0.3%. When lambda1 is too small (0.25), the search tends to favor subnetworks with larger gradients but lower stability, leading to a 0.5% accuracy drop. The default values lambda1 = lambda2 = 1.0 are used throughout the experiments, providing a balanced trade-off between gradient mean and standard deviation.

5.4. Improvement of the visual properties

LarKNAS not only achieves high performance and search efficiency, but also improves the visual properties of the models searched by NAS: 1) The effective receptive field (ERF) of LarKNASNet is larger, which brings better long-range modeling ability to the NAS method. 2) LarKNAS can search the visual backbone model with shape bias that is closer to human vision.

Visualization of effective receptive field. The concept of the effective receptive field (ERF) is crucial for deep convolutional neural networks [27, 64, 65]. The ERF is proportional to $O(k\sqrt{n})$, where k refers to the kernel size and n denotes the network depth according to [27], and any portion of an input image outside the receptive field does not influence its output value. To evaluate this hypothesis of our model, we compare the ERFs captured by BigNAS-XL [8] and ResNet-151 [13]. Following the methodologies of [24], we sample and resize 50 images from the validation set to 1024×1024 pixels and measure the contribution of each pixel in the input images to the central point of the feature map generated in the last layer. The contribution scores are then accumulated and projected onto a 1024×1024 matrix, as visualized in Figure 5. Despite the deep network architecture of ResNet-151 [13], its high-contribution pixels are primarily concentrated in the center of the input, and the ERF is not significantly improved. Although BigNAS-XL [8] employs kernels larger than 5×5 and increases the number of parameters, it still fails to extend the ERF to cover the entirety of the input. In contrast, the high-contributing pixels of LarKNASNet-L are distributed across a larger ERF, with some located in non-central regions. This observation indicates that LarKNAS effectively enhances the ERF compared to previous NAS methods, thereby improving the ability of model to capture long-range dependencies.

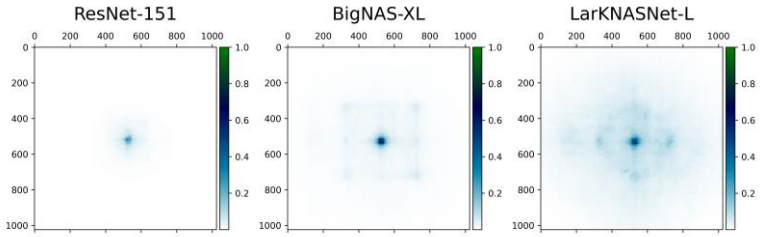


Figure 5 Effective receptive field (ERF) comparison between LarKNASNet-L, BigNAS-XL and ResNet-151. LarKNAS can bring a large ERF to large-scale visual model.

Analysis of shape bias. Shape properties are more heavily relied upon by humans for object discrimination [66]. Previous models, such as ResNet [13], have predominantly emphasized texture attributes. A higher shape bias signifies that a model depends more on the shape than texture of the object. This alignment with human cognition may enhance transfer performance in downstream tasks.

The shape bias of LarKNAS is analyzed in the experiments utilizing the model-vs-human toolbox [67], as depicted in Figure 6. LarKNAS, which employs only supervised training, demonstrates a 3% lower shape bias compared to the specially designed SimCLR [68] that utilizes a contrastive learning approach grounded in visual representation, including shape attributes. However, the performance across all categories for LarKNAS significantly exceeds that of ResNet [13] and AlexNet [26]. Furthermore, LarKNAS consistently outperforms BigNAS [8] across nearly all categories, achieving an improvement of approximately 8% in the proportion of decisions made based on the overall shape of objects. This suggests that LarKNAS can identify a visual backbone model with a shape bias more closely aligned with human visual perception.

Additionally, the impact of variations is investigated in convolution kernel size on visual bias. For SLaK [25] and LarKNAS, the output is derived from the last layer of four stages. As shown in Figure 7, a detailed analysis of the visual bias at each stage reveals that shape bias remains minimal in the shallow layers. Despite SLaK employing a large shallow convolution kernel, the shape bias does not increase, corroborating the argument presented in Section 3.1, which states that "there is no need to use large kernels in the early stages." Shape bias increases in the second and third stages, but does not show significant growth in the fourth stage, indicating that shape feature processing predominantly occurs in the second and third stages. The larger convolution kernels in these stages are better suited to handling shape features owing to their enhanced long-range modeling capabilities, reinforcing the necessity of employing larger convolution kernels during these stages, as discussed in Section 3.1.

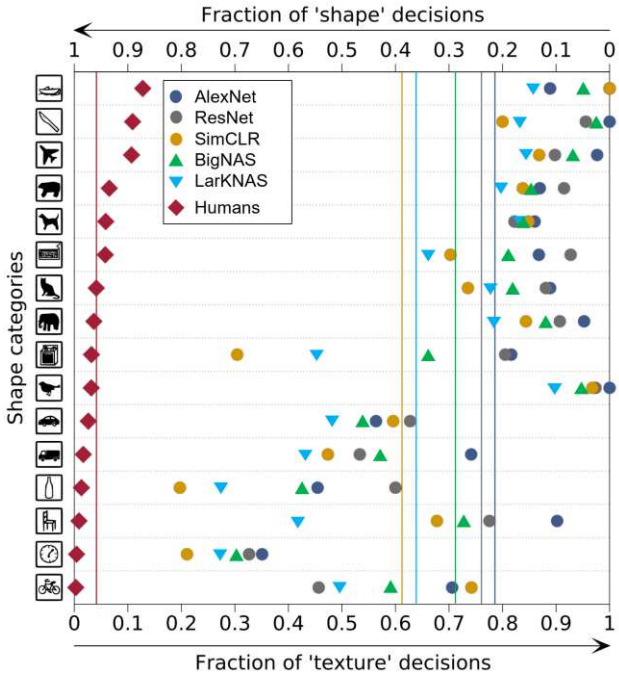


Figure 6 The shape bias of LarKNAS, BigNAS, SimCLR and other models. The scatter points represent the shape bias across 16 categories, while the vertical lines denote the averages for these categories. The line represents the average of these categories. A position further to the left indicates a closer alignment to the human visual baseline.

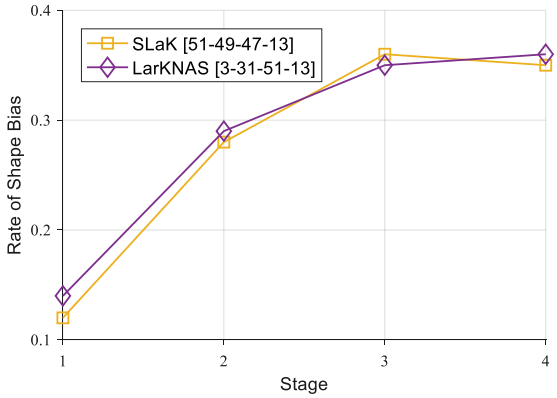


Figure 7 The average rates of shape bias across different stages for SLaK and LarKNAS.

6. Conclusion

This paper presents LarKNAS, Neural Architecture Search on Large Kernel convolution search space. LarKNAS provides a modern large kernel search space with the consideration of phased stage architecture and large sparse convolution kernel, and then develops a gradient estimation search method. Experiments show that LarKNAS addresses performance limitations of existing NAS and achieves performance comparable to hand-designed models (ImageNet-1K ACC of 86.6%, COCO box AP of 55.2% and ADE20K mIoU of 53.4%) with search cost of 96 GPU days for 335M Params, only 36% of ScaleNet. These findings indicate that our proposed methodology not only enhances the efficiency and effectiveness of large-scale NAS, but also provides a promising direction for future advancements in the design of convolutional neural networks.

References

[1] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search: A survey," *The Journal of Machine Learning Research*, vol. 20, no. 1, pp. 1997–2017, 2019.

[2] X. He, K. Zhao, and X. Chu, "AutoML: A Survey of the State-of-the-Art," *Knowledge-Based Systems*, vol. 212, p. 106622, 2021.

[3] X. Dong, D. J. Kedziora, K. Musial, and B. Gabrys, "Automated Deep Learning: Neural Architecture Search Is Not the End," *ACM Computing Surveys (CSUR)*, vol. 1, no. 1, 2021.

[4] P. Ren *et al.*, "A Comprehensive Survey of Neural Architecture Search: Challenges and Solutions," *ACM Computing Surveys (CSUR)*, vol. 54, no. 4, May 2021.

[5] K. T. Chitty-Venkata and A. K. Somani, "Neural Architecture Search Survey: A Hardware Perspective," *ACM Computing Surveys (CSUR)*, vol. 55, no. 4, Nov. 2022.

[6] S. Khan, M. Naseer, M. Hayat, S. W. Zamir, F. S. Khan, and M. Shah, "Transformers in vision: A survey," *ACM computing surveys (CSUR)*, vol. 54, no. 10s, pp. 1–41, 2022.

[7] Z. Liu, H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Darrell, and S. Xie, "A ConvNet for the 2020s," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[8] J. Yu, "BigNAS: Scaling up Neural Architecture Search with Big Single-Stage Models," in *Computer Vision – ECCV 2020*, vol. 12352, A. Vedaldi, H. Bischof, T. Brox, and J.-M. Frahm, Eds., Cham: Springer International Publishing, 2020, pp. 702–717.

[9] M. Tan and Q. Le, "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks," in *International Conference on Machine Learning*, 2019, pp. 6105–6114.

[10] H. Andrew *et al.*, "Searching for mobilenetv3," *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1314–1324, 2019.

[11] K. T. Chitty-Venkata, M. Emani, V. Vishwanath, and A. K. Somani, "Neural Architecture Search for Transformers: A Survey," *IEEE Access*, 2022.

[12] M. Tan, "Mnasnet: Platform-aware neural architecture search for mobile," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019.

[13] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[14] A. G. Howard *et al.*, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," *arXiv preprint arXiv:1704.04861*, 2017.

[15] A. Dosovitskiy, "An image is worth 16x16 words: Transformers for image recognition at scale," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.

[16] Z. Liu, "Swin transformer: Hierarchical vision transformer using shifted windows," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 10012–10022.

[17] J. E. G. Alvin Wan Xiaoliang Dai, Peizhao Zhang, Zijian He, Yuandong Tian, Saining Xie, Bichen Wu, Matthew Yu, Tao Xu, Kan Chen, Peter Vajda, "FBNetV2: Differentiable Neural Architecture Search for Spatial and Channel Dimensions," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[18] X. Dai, "FBNetV3: Joint architecture-recipe search using predictor pretraining," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR*, 2021, pp. 16276–16285.

[19] B. Wu *et al.*, "FBNetV5: Neural Architecture Search for Multiple Tasks in One Run," *CoRR*, vol. abs/2111.10007, 2021.

[20] J. Xie, X. Su, S. You, Z. Ma, F. Wang, and C. Qian, "ScaleNet: Searching for the Model to Scale," in *Computer Vision - ECCV 2022 - 17th European Conference, Tel Aviv, Israel, October 23-27, 2022, Proceedings, Part XXI*, S. Avidan, G. J. Brostow, M. Cissé, G. M. Farinella, and T. Hassner, Eds., in Lecture Notes in Computer Science, vol. 13681. Springer, 2022, pp. 104–120.

[21] Y. Fang *et al.*, "EVA: Exploring the Limits of Masked Visual Representation Learning at Scale," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2023, Vancouver, BC, Canada, June 17-24, 2023*, IEEE, 2023, pp. 19358–19369.

[22] W. Wang *et al.*, "Internimage: Exploring large-scale vision foundation models with deformable convolutions," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 14408–14419.

[23] X. Ding *et al.*, "UniRepLKNet: A Universal Perception Large-Kernel ConvNet for Audio, Video, Point Cloud, Time-Series and Image Recognition," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2024.

[24] X. Ding, X. Zhang, J. Han, and G. Ding, "Scaling up your kernels to 31x31: Revisiting large kernel design in CNNs," in *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022, pp. 11963–11975.

[25] S. Liu *et al.*, "More ConvNets in the 2020s: Scaling up Kernels Beyond 51x51 using Sparsity," in *The Eleventh International Conference on Learning Representations*, 2023.
[26] H. Chen, X. Chu, Y. Ren, X. Zhao, and K. Huang, "PeLK: Parameter-efficient Large Kernel ConvNets with Peripheral Convolution," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2024, pp. 5557–5567.
[27] W. Luo, Y. Li, R. Urtasun, and R. S. Zemel, "Understanding the Effective Receptive Field in Deep Convolutional Neural Networks," in *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, D. D. Lee, M. Sugiyama, U. von Luxburg, I. Guyon, and R. Garnett, Eds., 2016, pp. 4898–4906.
[28] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *ICLR*, 2015.
[29] G. Li, M. Muller, A. Thabet, and B. Ghanem, "Deepgcns: Can gcns go as deep as cnns?," in *ICCV*, 2019, pp. 9267–9276.
[30] H. Hu, Z. Zhang, Z. Xie, and S. Lin, "Local relation networks for image recognition," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 3464–3473.
[31] W. Yu *et al.*, "Metaformer is actually what you need for vision," in *CVPR*, 2022, pp. 10819–10829.
[32] Y. Rao, W. Zhao, Z. Zhu, J. Lu, and J. Zhou, "Global filter networks for image classification," *arXiv preprint arXiv:2107.00645*, 2021.
[33] X. Ding, X. Zhang, N. Ma, J. Han, G. Ding, and J. Sun, "Repvgg: Making vgg-style convnets great again," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 13733–13742.
[34] C. Li *et al.*, "Bossnas: Exploring hybrid cnn-transformers with block-wisely self-supervised neural architecture search," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 12281–12291.
[35] H. Cai, L. Zhu, and S. Han, "ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware," in *International Conference on Learning Representations*, 2019.
[36] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 8697–8710.
[37] C. Li *et al.*, "Block-wisely supervised neural architecture search with knowledge distillation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 1989–1998.
[38] Z. Zhong, J. Yan, W. Wu, J. Shao, and C.-L. Liu, "Practical Block-Wise Neural Network Architecture Generation," in *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, Computer Vision Foundation / IEEE Computer Society, 2018, pp. 2423–2432.
[39] D. Qin *et al.*, "MobileNetV4 - Universal Models for the Mobile Ecosystem," *CoRR*, vol. abs/2404.10518, 2024.
[40] L.-C. Chen, G. Papandreou, F. Schroff, and H. Adam, "Rethinking Atrous Convolution for Semantic Image Segmentation," *CoRR*, vol. abs/1706.05587, 2017.
[41] S. Woo *et al.*, "Convnext v2: Co-designing and scaling convnets with masked autoencoders," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 16133–16142.
[42] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International Conference on Machine Learning*, 2015, pp. 448–456.
[43] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer Normalization," *arXiv preprint arXiv:1607.06450*, 2016.
[44] N. Lee, T. Ajanthan, and P. Torr, "SNIP: Single-shot Network Pruning based on Connection Sensitivity," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2019.
[45] R. Wightman, H. Touvron, and H. Jegou, "ResNet strikes back: An improved training procedure in timm," in *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2021.
[46] H. Liu, K. Simonyan, and Y. Yang, "DARTS: Differentiable Architecture Search," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2019.
[47] X. Chu, T. Zhou, B. Zhang, and J. Li, "Fair darts: Eliminating unfair advantages in differentiable architecture search," in *European conference on computer vision*, Springer, 2020, pp. 465–480.
[48] P. Y. and B. L. and Y. L. and T. C. and J. F. and W. Ouyang, "β-DARTS: Beta-Decay Regularization for Differentiable Architecture Search," *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.
[49] G. Li, Y. Yang, K. Bhardwaj, and R. Marculescu, "ZiCo: Zero-shot NAS via inverse Coefficient of Variation on Gradients," in *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*, OpenReview.net, 2023.
[50] H. Zheng, K.-H. Liu, I. Fedorov, X. Zhang, W.-Y. Chen, and W. Wen, "SiGeo: Sub-One-Shot NAS via Geometry of Loss Landscape," in *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, KDD 2024, Barcelona, Spain, August 25-29, 2024*, R. Baeza-Yates and F. Bonchi, Eds., ACM, 2024, pp. 4536–4547.

[51] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han, "Once-for-All: Train One Network and Specialize it for Efficient Deployment," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2020.
[52] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, IEEE, 2009, pp. 248–255.
[53] T.-Y. Lin *et al.*, "Microsoft coco: Common objects in context," in *European conference on computer vision*, Springer, 2014, pp. 740–755.
[54] B. Zhou *et al.*, "Semantic understanding of scenes through the ade20k dataset," *International Journal of Computer Vision*, vol. 127, no. 3, pp. 302–321, 2019.
[55] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jégou, "Training data-efficient image transformers & distillation through attention," in *International Conference on Machine Learning*, PMLR, Jul. 2021, pp. 10347–10357.
[56] K. He, X. Chen, S. Xie, Y. Li, P. Dollár, and R. Girshick, "Masked Autoencoders Are Scalable Vision Learners," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2022, pp. 16000–16009.
[57] Y. Wang, Y. Yue, R. Lu, Y. Han, S. Song, and G. Huang, "EfficientTrain++: Generalized Curriculum Learning for Efficient Visual Backbone Training," *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2024.
[58] T. Xiao, Y. Liu, B. Zhou, Y. Jiang, and J. Sun, "Unified perceptual parsing for scene understanding," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 418–434.
[59] MMSegmentation Contributors, "MMSegmentation: OpenMMLab Semantic Segmentation Toolbox and Benchmark." 2020.
[60] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask r-cnn," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2961–2969.
[61] Z. Cai and N. Vasconcelos, "Cascade r-cnn: High quality object detection and instance segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2019.
[62] K. Chen *et al.*, "MMDetection: Open MMLab Detection Toolbox and Benchmark," *CoRR*, vol. abs/1906.07155, 2019.
[63] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," *arXiv preprint arXiv:1711.05101*, 2017.
[64] L. Xiao, Y. Bahri, J. Sohl-Dickstein, S. Schoenholz, and J. Pennington, "Dynamical Isometry and a Mean Field Theory of CNNs: How to Train 10,000-Layer Vanilla Convolutional Neural Networks," in *International Conference on Machine Learning*, 2018, pp. 5393–5402.
[65] B. J. Kim, H. Choi, H. Jang, D. G. Lee, W. Jeong, and S. W. Kim, "Dead Pixel Test Using Effective Receptive Field," *arXiv preprint arXiv:2108.13576*, 2021.
[66] R. Geirhos, P. Rubisch, C. Michaelis, M. Bethge, F. A. Wichmann, and W. Brendel, "ImageNet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness," *arXiv preprint arXiv:1811.12231*, 2018.
[67] R. Geirhos *et al.*, "Partial success in closing the gap between human and machine vision," in *Advances in Neural Information Processing Systems 34*, 2021.
[68] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A simple framework for contrastive learning of visual representations," in *Proceedings of the International Conference on Machine Learning (ICML)*, PMLR, 2020, pp. 1597–1607.

Supplementary Files

This is a list of supplementary files associated with this preprint. Click to download.

- SupplementaryMaterial.docx