KernelWarehouse: Rethinking the Design of Dynamic Convolution

Chao Li [1] **Anbang Yao** [1]

Abstract

Dynamic convolution learns a linear mixture of n static kernels weighted with their input-dependent attentions, demonstrating superior performance than normal convolution. However, it increases the number of convolutional parameters by n times, and thus is not parameter efficient. This leads to no research progress that can allow researchers to explore the setting $n > 100$ (an order of magnitude larger than the typical setting $n < 10$) for pushing forward the performance boundary of dynamic convolution while enjoying parameter efficiency. To fill this gap, in this paper, we propose *KernelWarehouse*, a more general form of dynamic convolution, which redefines the basic concepts of "kernels", "assembling kernels" and "attention function" through the lens of exploiting convolutional parameter dependencies within the same layer and across neighboring layers of a ConvNet. We testify the effectiveness of KernelWarehouse on ImageNet and MS-COCO datasets using various ConvNet architectures. Intriguingly, KernelWarehouse is also applicable to Vision Transformers, and it can even reduce the model size of a backbone while improving the model accuracy. For instance, KernelWarehouse ($n = 4$) achieves $5.61\%|3.90\%|4.38\%$ absolute top-1 accuracy gain on the ResNet18|MobileNetV2|DeiT-Tiny backbone, and KernelWarehouse ($n = 1/4$) with 65.10% model size reduction still achieves 2.29% gain on the ResNet18 backbone. The code and models are available at `https://github.com/OSVAI/KernelWarehouse`.

1. Introduction

Convolution is the key operation in convolutional neural networks (ConvNets). In a convolutional layer, normal convolution $\mathbf{y} = \mathbf{W} * \mathbf{x}$ computes the output $\mathbf{y}$ by applying the same convolutional kernel $\mathbf{W}$ defined as a set of convolutional filters to every input sample $\mathbf{x}$. *For brevity, we refer to "convolutional kernel" as "kernel" and omit the bias term throughout this paper*. Although the efficacy of normal convolution is extensively validated with various types of ConvNet architectures (Krizhevsky et al., 2012; He et al., 2016; Howard et al., 2017; Liu et al., 2022) on many computer vision tasks, recent progress in the efficient ConvNet architecture design shows that dynamic convolution, known as CondConv (Yang et al., 2019a) and DY-Conv (Chen et al., 2020), achieves large performance gains.

The basic idea of dynamic convolution is to replace the single kernel in normal convolution by a linear mixture of n same dimensioned kernels, $\mathbf{W} = \alpha_1\mathbf{W}_1 + ... + \alpha_n\mathbf{W}_n$, where $\alpha_1, ..., \alpha_n$ are scalar attentions generated by an input-dependent attention module. Benefiting from the additive property of $\mathbf{W}_1, ..., \mathbf{W}_n$ and compact attention module designs, dynamic convolution improves the feature learning ability with little extra multiply-add cost against normal convolution. However, it increases the number of convolutional parameters by n times, which leads to a huge rise in model size because the convolutional layers of a modern ConvNet occupy the vast majority of parameters. There exist few research works to alleviate this problem. DCD (Li et al., 2021b) learns a base kernel and a sparse residual to approximate dynamic convolution via matrix decomposition. This approximation abandons the basic mixture learning paradigm, and thus cannot retain the representation power of dynamic convolution when n becomes large. ODConv (Li et al., 2022) presents an improved attention module to dynamically weight static kernels along different dimensions instead of one single dimension, which can get competitive performance with a reduced number of kernels. However, under the same setting of n, ODConv has more parameters than vanilla dynamic convolution. Recently, He et al. (2023) directly used popular weight pruning strategy to compress DY-Conv via multiple pruning-and-retraining phases.

In a nutshell, existing dynamic convolution methods based on the linear mixture learning paradigm are not parameter-efficient. Restricted by this, the kernel number is typically set to $n = 8$ (Yang et al., 2019a) or $n = 4$ (Chen et al., 2020; Li et al., 2022). However, a plain fact is that the improved capacity of a ConvNet constructed with dynamic convolution

[1]Intel Labs China. Correspondence to: Anbang Yao <anbang.yao@intel.com>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).

comes from increasing the kernel number n per convolutional layer facilitated by the attention mechanism. This causes a fundamental conflict between the desired model size and capacity. In this work, we rethink the design of dynamic convolution, with the goal of reconciling such a conflict, enabling us to explore the performance boundary of dynamic convolution with the significantly larger kernel number setting $n > 100$ (an order of magnitude larger than the typical setting $n < 10$) while enjoying parameter efficiency. Note that, for existing dynamic convolution methods, $n > 100$ means that the model size will be about > 100 times larger than the backbone built with normal convolution.

To this goal, we present a more general form of dynamic convolution called *KernelWarehouse* (Figure 1 shows a schematic overview). Our work is inspired by two observations about existing dynamic convolution methods: (1) They treat all parameters in a regular convolutional layer as a static kernel, increase the kernel number from 1 to n, and use their attention modules to assemble n static kernels into a linearly mixed kernel. Though straightforward and effective, they pay no attention to parameter dependencies within the static kernel at a convolutional layer; (2) They allocate different sets of n static kernels for individual convolutional layers of a ConvNet, ignoring parameter dependencies across neighboring convolutional layers. In a sharp contrast with existing methods, the core philosophy of KernelWarehouse is to *exploit convolutional parameter dependencies within the same layer and across neighboring layers of a ConvNet*, reformulating dynamic convolution towards achieving a substantially better trade-off between parameter efficiency and representation power.

KernelWarehouse consists of three components, namely kernel partition, warehouse construction-with-sharing and contrasting-driven attention function, which are closely interdependent. Kernel partition exploits parameter dependencies within the same convolutional layer, by which we redefine "*kernels*" for the linear mixture in terms of a much smaller local scale instead of a holistic scale. Warehouse construction-with-sharing exploits parameter dependencies across neighboring convolutional layers, by which we redefine "*assembling kernels*" across all same-stage convolutional layers instead of within a single layer, and generate a large warehouse consisting of n local kernels (e.g., $n = 108$) shared for cross-layer linear mixtures. Contrasting-driven attention function is customized to address the attention optimization problem under the cross-layer linear mixture learning paradigm with the challenging settings of $n > 100$, by which we redefine "*attention function*". Given different convolutional parameter budgets (see Section. 3.3 for the definition of convolutional parameter budget), these concept shifts provide a high degree of flexibility for KernelWarehouse, allowing to well balance parameter efficiency and representation power with sufficiently large values of n.

As a drop-in replacement of normal convolution, KernelWarehouse can be easily used to various types of ConvNet architectures. We validate the effectiveness of KernelWarehouse through extensive experiments on ImageNet and MS-COCO datasets. On one hand, we show that KernelWarehouse achieves superior performance compared to existing dynamic convolution methods (e.g., the ResNet18|ResNet50|MobileNetV2|ConvNeXt-Tiny model with KernelWarehouse trained on ImageNet dataset reaches 76.05%|81.05%|75.92%|82.55% top-1 accuracy, setting new performance records for dynamic convolution research). On the other hand, we show that all three components of KernelWarehouse are essential to the performance boost in terms of model accuracy and parameter efficiency, and KernelWarehouse can even reduce the model size of a ConvNet while improving the model accuracy (e.g., our ResNet18 model with 65.10% parameter reduction to the baseline still achieves 2.29% absolute top-1 accuracy gain), and it is also applicable to Vision Transformers (e.g., our DeiT-Tiny model reaches 76.51% top-1 accuracy, bringing 4.38% absolute top-1 accuracy gain to the baseline).

2. Related Work

ConvNet Architectures. In the past decade, many notable ConvNet architectures such as AlexNet (Krizhevsky et al., 2012), VGGNet (Simonyan & Zisserman, 2015), GoogLeNet (Szegedy et al., 2015), ResNet (He et al., 2016), DenseNet (Huang et al., 2017), ResNeXt (Xie et al., 2017) and RegNet (Radosavovic et al., 2020) have been presented. Around the same time, some lightweight architectures like MobileNet (Howard et al., 2017; Sandler et al., 2018; Howard et al., 2019), ShuffleNet (Zhang et al., 2018b) and EfficientNet (Tan & Le, 2019a) have been designed for resource-constrained applications. Recently, Liu et al. (2022) presented ConvNeXt whose performance can match newly emerging vision transformers (Dosovitskiy et al., 2021; Liu et al., 2021). Our method could be used to improve their performance, as we show in the experiments.

Feature Recalibration. An effective way to enhance the capacity of a ConvNet is feature recalibration. It relies on attention mechanisms to adaptively refine the feature maps learnt by a convolutional block. Popular feature recalibration modules such as RAN (Wang et al., 2017), SE (Hu et al., 2018b), BAM (Park et al., 2018), CBAM (Woo et al., 2018), GE (Hu et al., 2018a), SRM (Lee et al., 2019) and ECA (Wang et al., 2020) focus on different design aspects: using channel attention, or spatial attention, or hybrid attention to emphasize important features and suppress unnecessary ones. Unlike these methods that retain the static kernel, dynamic convolution replaces the single static kernel of a convolutional layer by a linear mixture of n static kernels

weighted with the attention mechanism.

Dynamic Weight Networks. Many research efforts have been made on developing effective methods to generate the weights for a neural network. Jaderberg et al. (2015) proposed a Spatial Transformer module which uses a localisation network that predicts the feature transformation parameters conditioned on the learnt feature itself. Dynamic Filter Network (Jia et al., 2016) and Kernel Prediction Networks (Bako et al., 2017; Mildenhall et al., 2018) introduce two filter generation frameworks which share the same idea: using a deep neural network to generate sample-adaptive filters conditioned on the input. Based on this idea, DynamoNet (Diba et al., 2019) uses dynamically generated motion filters to boost video-based human action recognition. CARAFE (Wang et al., 2019) and Involution (Li et al., 2021a) further extend this idea by designing efficient generation modules to predict the weights for extracting informative features. By connecting this idea with SE, WeightNet (Ma et al., 2020), CGC (Lin et al., 2020) and WE (Quader et al., 2020) design different attention modules to adjust the weights in convolutional layers of a ConvNet. Hypernetwork (Ha et al., 2017) uses a small network to generate the weights for a larger recurrent network instead of a ConvNet. MetaNet (Munkhdalai & Yu, 2017) introduces a meta learning model consisting of a base learner and a meta learner, allowing the learnt network for rapid generalization across different tasks. Unlike them, this work focuses on advancing dynamic convolution research.

3. Method

3.1. Motivation and Components of KernelWarehouse

For a convolutional layer, let $\mathbf{x} \in \mathbb{R}^{h\times w\times c}$ and $\mathbf{y} \in \mathbb{R}^{h\times w\times f}$ be the input having c feature channels and the output having f feature channels respectively, where $h \times w$ denotes the channel size. Normal convolution $\mathbf{y} = \mathbf{W} * \mathbf{x}$ uses a static kernel $\mathbf{W} \in \mathbb{R}^{k\times k\times c\times f}$ consisting of f convolutional filters with the spatial size $k \times k$. Dynamic convolution (Yang et al., 2019a; Chen et al., 2020) replaces $\mathbf{W}$ in normal convolution by a linear mixture of n same dimensioned static kernels $\mathbf{W}_1, ..., \mathbf{W}_n$ weighted with $\alpha_1, ..., \alpha_n$ generated by an attention module $\phi(x)$, which is defined as

$$\mathbf{W} = \alpha_1\mathbf{W}_1 + ... + \alpha_n\mathbf{W}_n. \tag{1}$$

As we discussed earlier, the kernel number n is typically set to $n < 10$, restricted by the parameter-inefficient shortcoming. The main motivation of this work is to reformulate this linear mixture learning paradigm, enabling us to explore significantly larger settings, e.g., $n > 100$ (an order of magnitude larger than the typical setting $n < 10$), for pushing forward the performance boundary of dynamic convolution while enjoying parameter efficiency. To that end, our proposed KernelWarehouse has three key components: kernel partition, warehouse construction-with-sharing, and contrasting-driven attention function.

3.2. Kernel Partition

The main idea of kernel partition is to reduce kernel dimension via simply exploiting the parameter dependencies within the same convolutional layer. Specifically, for a regular convolutional layer, we sequentially divide the static kernel $\mathbf{W}$ along spatial and channel dimensions into m disjoint parts $\mathbf{w}_1$,...,$\mathbf{w}_m$ called "*kernel cells*" that have the same dimensions. *For brevity, here we omit to define kernel cell dimensions*. Kernel partition can be defined as

$$\begin{aligned} &\mathbf{W} = \mathbf{w}_1 \cup ... \cup \mathbf{w}_m, \\ &\text{and } \forall\, i, j \in \{1, ..., m\}, i \neq j,\ \mathbf{w}_i \cap \mathbf{w}_j = \emptyset. \end{aligned} \tag{2}$$

After kernel partition, we treat kernel cells $\mathbf{w}_1$,...,$\mathbf{w}_m$ as "*local kernels*", and define a "*warehouse*" containing n kernel cells $\mathbf{E} = \{\mathbf{e}_1, ..., \mathbf{e}_n\}$, where $\mathbf{e}_1, ..., \mathbf{e}_n$ have the same dimensions as $\mathbf{w}_1$,...,$\mathbf{w}_m$. Then, we use the warehouse $\mathbf{E} = \{\mathbf{e}_1, ..., \mathbf{e}_n\}$ to represent each of m kernel cells $\mathbf{w}_1$,...,$\mathbf{w}_m$ as a linear mixture

$$\mathbf{w}_i = \alpha_{i1}\mathbf{e}_1 + ... + \alpha_{in}\mathbf{e}_n,\ \text{and } i \in \{1, ..., m\}, \tag{3}$$

where $\alpha_{i1}, ..., \alpha_{in}$ are the input-dependent scalar attentions generated by an attention module $\phi(x)$. Finally, the static kernel $\mathbf{W}$ in a regular convolutional layer is replaced by its corresponding m linear mixtures.

Thanks to kernel partition, the dimensions of the kernel cell $\mathbf{w_i}$ could be much smaller than the dimensions of the static kernel $\mathbf{W}$. For example, when $m = 16$, the number of convolutional parameters in the kernel cell $\mathbf{w_i}$ is 1/16 to that of the static kernel $\mathbf{W}$. Under a desired convolutional parameter budget b (we will define it in Section. 3.3), this allows a warehouse can easily have a significantly larger value of n (e.g., $n = 64$), in comparison to existing dynamic convolution methods that define the linear mixture in terms of n (e.g., $n = 4$) "*holistic kernels*".

3.3. Warehouse Construction-with-Sharing

The main idea of warehouse construction-with-sharing is to further improve the warehouse-based linear mixture learning formulation through simply exploiting parameter dependencies across neighboring convolutional layers. Specifically, for l same-stage convolutional layers of a ConvNet, we construct a shared warehouse $\mathbf{E} = \{\mathbf{e}_1, ..., \mathbf{e}_n\}$ by using the same kernel cell dimensions for kernel partition. This allows the shared warehouse not only can have a larger value of n (e.g., $n = 188$) compared to the layer-specific warehouse (e.g., $n = 36$), but also can have improved representation power (see Table 6). Thanks to the modular design

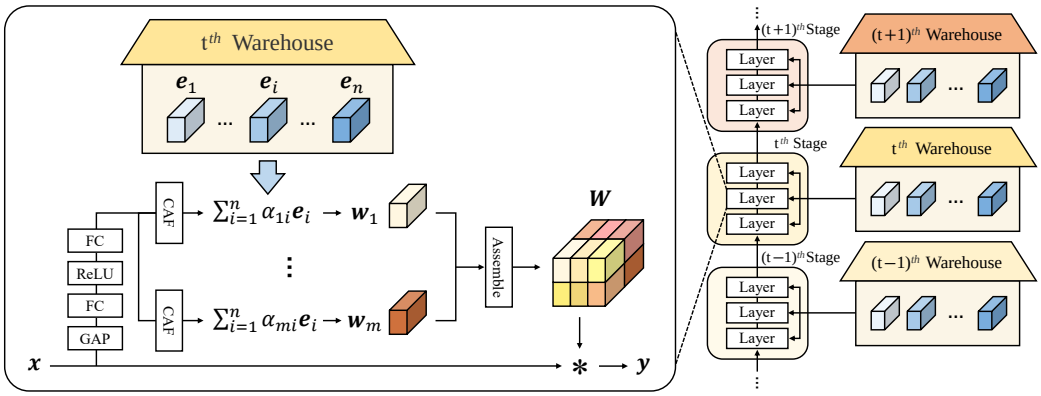


Figure 1: A schematic overview of *KernelWarehouse* to a ConvNet. As a more general form of dynamic convolution, KernelWarehouse consists of three interdependent components, namely kernel partition, warehouse construction-with-sharing and contrasting-driven attention function (CAF), which redefine the basic concepts of "kernels", "assembling kernels" and "attention function" in the perspective of exploiting convolutional parameter dependencies within the same layer and across neighboring layers of a ConvNet, enabling to use significantly large kernel number settings (e.g., $n > 100$) while enjoying improved model accuracy and parameter efficiency. Please see the Method section for the detailed formulation.

mechanism of modern ConvNets, we simply use common dimension divisors for all l same-stage static kernels as the uniform kernel cell dimensions to perform kernel partition, which naturally determines the kernel cell number m for each same-stage convolutional layer, as well as n for the shared warehouse when a desired convolutional parameter budget b is given. Figure 2 illustrates the processes of kernel partition and warehouse construction-with-sharing.

Convolutional Parameter Budget. For vanilla dynamic convolution (Yang et al., 2019a; Chen et al., 2020), the convolutional parameter budget b relative to normal convolution is always equal to the kernel number. That is, $b == n$, and $n >= 1$. When setting a large value of n, e.g., $n = 188$, existing dynamic convolution methods get $b = 188$, leading to about 188 times larger model size for a ConvNet backbone. For KernelWarehouse, such drawbacks are resolved. Let m_t be the total number of kernel cells in l same-stage convolutional layers ($m_t = m$, when $l = 1$) of a ConvNet. Then, the convolutional parameter budget of KernelWarehouse relative to normal convolution can be defined as $b = n/m_t$. In implementation, we use the same value of b to all convolutional layers of a ConvNet, so that KernelWarehouse can easily scale up or scale down the model size of a ConvNet by changing b. Compared to normal convolution: (1) When $b < 1$, KernelWarehouse tends to get the reduced model size; (2) When $b = 1$, KernelWarehouse tends to get the similar model size; (3) When $b > 1$, KernelWarehouse tends to get the increased model size.

Parameter Efficiency and Representation Power. Intriguingly, given a desired parameter budget b, a proper and large value of n can be obtained by simply changing m_t (controlled by kernel partition and warehouse construction-with-sharing), providing a representation power guarantee for KernelWarehouse. Thanks to this flexibility, KernelWarehouse can strike a favorable trade-off between parameter efficiency and representation power, under different convolutional parameter budgets, as well validated by trained model exemplifications KW $(1/2\times, 3/4\times, 1\times, 4\times)$ in Table 1, Table 2 and Table 4.

3.4. Contrasting-driven Attention Function

With the above formulation, the optimization of KernelWarehouse differs with existing dynamic convolution methods in three aspects: (1) The linear mixture is used to a dense local kernel cell scale instead of a holistic kernel scale; (2) The number of kernel cells in a warehouse is significantly larger ($n > 100$ vs. $n < 10$); (3) A warehouse is not only shared to represent each of m kernel cells for a specific convolutional layer of a ConvNet, but also is shared to represent every kernel cell for the other $l-1$ same-stage convolutional layers. However, for KernelWarehouse with these optimization properties, we empirically find that popular attention functions lose their effectiveness (see Table 8). We present contrasting-driven attention function (CAF), a customized design, to solve the optimization of KernelWarehouse. For i^{th} kernel cell in the static kernel $\mathbf{W}$, let $z_{i1}, ..., z_{in}$ be the feature logits generated by the second fully-connected layer of a compact SE-typed attention module $\phi(x)$ (*its structure is clarified in Appendix*), then CAF is defined as

$$\alpha_{ij} = \tau\beta_{ij} + (1-\tau)\frac{z_{ij}}{\sum_{p=1}^{n}|z_{ip}|}, \text{ and } j \in \{1, ..., n\}, \quad (4)$$

where τ is a temperature linearly reducing from 1 to 0 in the early training stage; β_{ij} is a binary value (0 or 1) for initializing attentions; $\frac{z_{ij}}{\sum_{p=1}^{n}|z_{ip}|}$ is a normalization function.

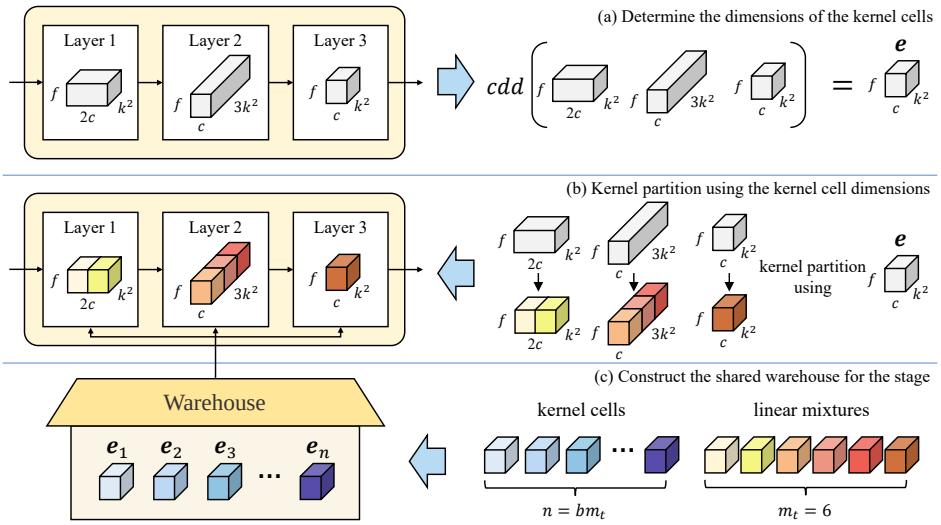


Figure 2: An illustration of *kernel partition* and *warehouse construction-with-sharing* across three same-stage convolutional layers of a ConvNet. cdd denotes common kernel dimension divisors, and b is the desired convolutional parameter budget.

Our CAF relies on two smart design principles: (1) The first term ensures that the initial valid kernel cells ($\beta_{ij} = 1$) in a shared warehouse are uniformly allocated to represent different linear mixtures at all l same-stage convolutional layers of a ConvNet when starting the training; (2) The second term enables the existence of both negative and positive attentions, unlike popular attention functions that always generate positive attentions. This encourages the optimization process to learn contrasting and diverse attention distributions among all linear mixtures at l same-stage convolutional layers sharing the same warehouse (as illustrated in Figure 3), guaranteeing to improving model performance.

At CAF initialization, the setting of β_{ij} at l same-stage convolutional layers should assure the shared warehouse can assign: (1) At least one specified kernel cell ($\beta_{ij} = 1$) to every linear mixture, given $b \geq 1$; (2) At most one specific kernel cell ($\beta_{ij} = 1$) to every linear mixture, given $b < 1$. We adopt a simple strategy to assign one of the total n kernel cells in a shared warehouse to each of m_t linear mixtures at l same-stage convolutional layers without repetition. When $n < m_t$, we let the remaining linear mixtures always have $\beta_{ij} = 0$ once n kernel cells are used up. *In the Appendix*, we provide visualization examples to illustrate this strategy.

3.5. Discussion

Note that the split-and-merge strategy with multi-branch group convolution has been widely used in many ConvNet architectures (Szegedy et al., 2015; Xie et al., 2017; Sandler et al., 2018; Li et al., 2019; Tan & Le, 2019b; Yang et al., 2019b; Tan & Le, 2019b; Li et al., 2020; Liu et al., 2022). Although KernelWarehouse also uses the parameter splitting idea in kernel partition, its focus and motivation we have clarified above are clearly different from them. Besides, KernelWarehouse could be also used to improve their performance as they use normal convolution. We validate this on MobileNetV2 and ConvNeXt (see Table 2 and Table 4).

According to its formulation, KernelWarehouse will degenerate into vanilla dynamic convolution (Yang et al., 2019a; Chen et al., 2020) when uniformly setting $m = 1$ in kernel partition (i.e., all kernel cells in each warehouse have the same dimensions as the static kernel $\mathbf{W}$ in normal convolution) and $l = 1$ in warehouse sharing (i.e., each warehouse is used to its specific convolutional layer). Therefore, KernelWarehouse is a more general form of dynamic convolution.

In formulation, the three key components of KernelWarehouse are closely interdependent, and their joint regularizing effect leads to significantly improved performance in terms of both model accuracy and parameter efficiency, as validated by multiple ablations in the Experiments section.

4. Experiments

In this section, we conduct comprehensive experiments on ImageNet dataset (Russakovsky et al., 2015) and MS-COCO dataset (Lin et al., 2014) to evaluate the effectiveness of our proposed KernelWarehouse ("**KW**" for short, in Tables).

4.1. Image Classification on ImageNet Dataset

Our basic experiments are conducted on ImageNet dataset.

ConvNet Backbones. We select five ConvNet backbones from MobileNetV2 (Sandler et al., 2018), ResNet (He et al., 2016) and ConvNeXt (Liu et al., 2022) families for experiments, including both lightweight and larger architectures.

Table 1: Results comparison on ImageNet with the ResNet18 backbone using the traditional training strategy. *Best results are bolded.*

Models	Params	Top-1 Acc (%)	Top-5 Acc (%)
ResNet18	11.69M	70.25	89.38
+ SE	11.78M	70.98 (↑0.73)	90.03 (↑0.65)
+ CBAM	11.78M	71.01 (↑0.76)	89.85 (↑0.47)
+ ECA	11.69M	70.60 (↑0.35)	89.68 (↑0.30)
+ CGC	11.69M	71.60 (↑1.35)	90.35 (↑0.97)
+ WeightNet	11.93M	71.56 (↑1.31)	90.38 (↑1.00)
+ DCD	14.70M	72.33 (↑2.08)	90.65 (↑1.27)
+ CondConv ($8\times$)	81.35M	71.99 (↑1.74)	90.27 (↑0.89)
+ DY-Conv ($4\times$)	45.47M	72.76 (↑2.51)	90.79 (↑1.41)
+ ODConv ($4\times$)	44.90M	73.97 (↑3.72)	91.35 (↑1.97)
+ KW ($1/2\times$)	7.43M	72.81 (↑2.56)	90.66 (↑1.28)
+ KW ($1\times$)	11.93M	73.67 (↑3.42)	91.17 (↑1.79)
+ KW ($2\times$)	23.24M	74.03 (↑3.78)	91.37 (↑1.99)
+ KW ($4\times$)	45.86M	**74.16 (↑3.91)**	**91.42 (↑2.04)**

Experimental Setup. In the experiments, we make diverse comparisons of our method with related methods to demonstrate its effectiveness. Firstly, on the ResNet18 backbone, we compare our method with various state-of-the-art attention based methods, including: (1) SE (Hu et al., 2018b), CBAM (Woo et al., 2018) and ECA (Wang et al., 2020), which focus on feature recalibration; (2) CGC (Lin et al., 2020) and WeightNet (Ma et al., 2020), which focus on adjusting convolutional weights; (3) CondConv (Yang et al., 2019a), DY-Conv (Chen et al., 2020), DCD (Li et al., 2021b) and ODConv (Li et al., 2022), which focus on dynamic convolution. Secondly, we select DY-Conv (Chen et al., 2020) and ODConv (Li et al., 2022) as our key reference methods, since they are top-performing dynamic convolution methods which are most closely related to our method. We compare our KernelWarehouse with them on all the other ConvNet backbones except ConvNeXt-Tiny (since there is no publicly available implementation of them on ConvNeXt). To make fair comparisons, all the methods are implemented using the public codes with the same settings for training and testing. In the experiments, we use $b\times$ to denote the convolutional parameter budget of each dynamic convolution method relative to normal convolution, the values of n and m in KernelWarehouse and the experimental details for each ConvNet backbone are provided *in the Appendix*.

Results Comparison with Traditional Training Strategy. We first use the traditional training strategy adopted by lots of previous studies, training the ResNet18 backbone for 100 epochs. The results are shown in Table 1. It can be observed that dynamic convolution methods (CondConv, DY-Conv and ODConv), which introduce obviously more extra parameters, tend to have better performance compared with the other reference methods (SE, CBAM, ECA, CGC, WeightNet and DCD). Note that our KW ($1/2\times$), which has 36.45% parameters less than the baseline, can even outperform all the above attention based methods (except ODConv ($4\times$), but including CondConv ($8\times$) and DY-Conv ($4\times$) which increase the model size to 6.96|3.89 times). Comparatively, our KW ($4\times$) achieves the best results.

Table 2: Results comparison on ImageNet with the ResNet18, ResNet50 and ConvNeXt-Tiny backbones using the advanced training strategy (Liu et al., 2022).

Models	Params	Top-1 Acc (%)	Top-5 Acc (%)
ResNet18	11.69M	70.44	89.72
+ DY-Conv ($4\times$)	45.47M	73.82 (↑3.38)	91.48 (↑1.76)
+ ODConv ($4\times$)	44.90M	74.45 (↑4.01)	91.67 (↑1.95)
+ KW ($1/4\times$)	4.08M	72.73 (↑2.29)	90.83 (↑1.11)
+ KW ($1/2\times$)	7.43M	73.33 (↑2.89)	91.42 (↑1.70)
+ KW ($1\times$)	11.93M	74.77 (↑4.33)	92.13 (↑2.41)
+ KW ($2\times$)	23.24M	75.19 (↑4.75)	92.18 (↑2.46)
+ KW ($4\times$)	45.86M	**76.05 (↑5.61)**	**92.68 (↑2.96)**
ResNet50	25.56M	78.44	94.24
+ DY-Conv ($4\times$)	100.88M	79.00 (↑0.56)	94.27 (↑0.03)
+ ODConv ($4\times$)	90.67M	80.62 (↑2.18)	95.16 (↑0.92)
+ KW ($1/2\times$)	17.64M	79.30 (↑0.86)	94.71 (↑0.47)
+ KW ($1\times$)	28.05M	80.38 (↑1.94)	95.19 (↑0.95)
+ KW ($4\times$)	102.02M	**81.05 (↑2.61)**	**95.21 (↑0.97)**
ConvNeXt-Tiny	28.59M	82.07	95.86
+ KW ($3/4\times$)	24.53M	82.23 (↑0.16)	95.88 (↑0.02)
+ KW ($1\times$)	32.99M	**82.55 (↑0.48)**	**96.08 (↑0.22)**

Results Comparison with Advanced Training Strategy. To better explore the potential of our method, we further adopt the advanced training strategy recently proposed in ConvNeXt (Liu et al., 2022), with a longer training schedule of 300 epochs and aggressive augmentations, for comparisons on the ResNet18, ResNet50 and ConvNeXt-Tiny backbones. From the results summarized in Table 2, we can observe: (1) KW ($4\times$) gets the best results on the ResNet18 backbone, bringing a significant top-1 gain of 5.61%. Even with 36.45%|65.10% parameter reduction, KW ($1/2\times$)|KW($1/4\times$) brings 2.89%|2.29% top-1 accuracy gain to the baseline model; (2) On the larger ResNet50 backbone, vanilla dynamic convolution DY-Conv ($4\times$) brings small top-1 gain to the baseline model, but KW ($1/2\times, 1\times, 4\times$) gets promising top-1 gains. Even with 30.99% parameter reduction against the baseline model, KW ($1/2\times$) attains a top-1 gain of 0.86%. KW ($4\times$) outperforms DY-Conv ($4\times$) and ODConv ($4\times$) by 2.05% and 0.43% top-1 gain, respecrtively. Besides, we also apply our method to the ConvNeXt-Tiny backbone to investigate its performance on the state-of-the-art ConvNet architecture. Results show that our method generalizes well on ConvNeXt-Tiny. Surprisingly, KW($3/4\times$) with 14.20% parameter reduction to the baseline model gets 0.16% top-1 gain.

Results Comparison on MobileNets. We further apply our method to MobileNetV2 ($1.0\times$, $0.5\times$) to validate its effectiveness on lightweight ConvNet architectures. Since the lightweight MobileNetV2 backbones have lower model capacity compared to ResNet and ConvNeXt backbones, we don't use aggressive augmentations. The results are shown in Table 4. We can see that our method can strike a

Table 3: Results comparison on MS-COCO using the pre-trained backbone models. *Best results are bolded.*

Detectors	Backbone Models	Object Detection						Instance Segmentation					
		AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Mask R-CNN	ResNet50	39.6	61.6	43.3	24.4	43.7	50.0	36.4	58.7	38.6	20.4	40.4	48.4
	+ DY-Conv ($4\times$)	39.6	62.1	43.1	24.7	43.3	50.5	36.6	59.1	38.6	20.9	40.2	49.1
	+ ODConv ($4\times$)	42.1	65.1	46.1	**27.2**	46.1	53.9	38.6	61.6	41.4	**23.1**	42.3	52.0
	+ KW ($1\times$)	41.8	64.5	45.9	26.6	45.5	53.0	38.4	61.4	41.2	22.2	42.0	51.6
	+ KW ($4\times$)	**42.4**	**65.4**	**46.3**	**27.2**	**46.2**	**54.6**	**38.9**	**62.0**	**41.5**	22.7	**42.6**	**53.1**
	MobileNetV2 ($1.0\times$)	33.8	55.2	35.8	19.7	36.5	44.4	31.7	52.4	33.3	16.4	34.4	43.7
	+ DY-Conv ($4\times$)	37.0	58.6	40.3	21.9	40.1	47.9	34.1	55.7	36.1	18.6	37.1	46.3
	+ ODConv ($4\times$)	37.2	59.4	39.9	22.6	40.0	48.0	34.5	56.4	36.3	19.3	37.3	46.8
	+ KW ($1\times$)	36.4	58.3	39.2	22.0	39.6	47.0	33.7	55.1	35.7	18.9	36.7	45.6
	+ KW ($4\times$)	**38.0**	**60.0**	**40.8**	**23.1**	**40.7**	**50.0**	**34.9**	**56.6**	**37.0**	**19.4**	**37.9**	**47.8**
	ConvNeXt-Tiny	43.4	65.8	47.7	27.6	46.8	55.9	39.7	62.6	42.4	23.1	43.1	53.7
	+ KW ($3/4\times$)	44.1	66.8	48.4	29.7	47.4	56.7	40.2	63.6	43.0	**24.8**	43.6	54.3
	+ KW ($1\times$)	**44.8**	**67.7**	**48.9**	**29.8**	**48.3**	**57.3**	**40.6**	**64.4**	**43.4**	24.7	**44.1**	**54.8**

Table 4: Results comparison on ImageNet with the MobileNetV2 backbones trained for 150 epochs.

Models	Params	Top-1 Acc (%)	Top-5 Acc (%)
MobileNetV2 ($0.5\times$)	1.97M	64.30	85.21
+ DY-Conv ($4\times$)	4.57M	69.05 (↑4.75)	88.37 (↑3.16)
+ ODConv ($4\times$)	4.44M	70.01 (↑5.71)	89.01 (↑3.80)
+ KW ($1/2\times$)	1.47M	65.19 (↑0.89)	85.98 (↑0.77)
+ KW ($1\times$)	2.85M	68.29 (↑3.99)	87.93 (↑2.72)
+ KW ($4\times$)	4.65M	**70.26** (↑**5.96**)	**89.19** (↑**3.98**)
MobileNetV2 ($1.0\times$)	3.50M	72.02	90.43
+ DY-Conv ($4\times$)	12.40M	74.94 (↑2.92)	91.83 (↑1.40)
+ ODConv ($4\times$)	11.52M	75.42 (↑3.40)	92.18 (↑1.75)
+ KW ($1/2\times$)	2.65M	72.59 (↑0.57)	90.71 (↑0.28)
+ KW ($1\times$)	5.17M	74.68 (↑2.66)	91.90 (↑1.47)
+ KW ($4\times$)	11.38M	**75.92** (↑**3.90**)	**92.22** (↑**1.79**)

favorable trade-off between parameter efficiency and representation power for lightweight ConvNets as well as larger ones. Even on the lightweight MobileNetV2 ($1.0\times, 0.5\times$) with 3.50M|1.97M parameters, KW ($1/2\times$) can reduce the model size by 24.29%|25.38% while bringing top-1 gain of 0.57%|0.89%. Similar to the results on the ResNet18 and ResNet50 backbones, KW ($4\times$) also obtains the best results on both MobileNetV2 ($1.0\times$) and MobileNetV2 ($0.5\times$).

4.2. Detection and Segmentation on MS-COCO Dataset

Next, to evaluate the generalization ability of the classification backbone models trained by our method to downstream object detection and instance segmentation tasks, we conduct comparative experiments on MS-COCO dataset.

Experimental Setup. We adopt Mask R-CNN (He et al., 2017) as the detection framework, ResNet50 and MobileNetV2 ($1.0\times$) built with different dynamic convolution methods as the backbones which are pre-trained on ImageNet dataset. Then, all the models are trained with standard $1\times$ schedule on MS-COCO dataset. For fair comparisons, we adopt the same settings including data processing pipeline and hyperparameters for all the models. Experimental details are described *in the Appendix*.

Results Comparison. Table 3 summarizes all results. For Mask R-CNN with the ResNet50 backbone, we observe a similar trend to the main experiments on ImageNet dataset: KW ($4\times$) outperforms DY-Conv ($4\times$) and ODConv ($4\times$) on both object detection and instance segmentation tasks. Our KW ($1\times$) brings an AP improvement of 2.2%|2.0% on object detection and instance segmentation tasks, which is even on par with ODConv ($4\times$). On the MobileNetV2 ($1.0\times$) backbone, our method yields consistent high accuracy improvements to the baseline, and KW ($4\times$) achieves the best results. With the ConvNeXt-Tiny backbone, the performance gains of KW ($1\times$) and KW ($3/4\times$) to the baseline model become more pronounced on MS-COCO dataset compared to those on ImageNet dataset, showing good transfer learning ability of our method.

Table 5: Effect of kernel partition.

Models	Kernel Partition	Params	Top-1 Acc (%)	Top-5 Acc (%)
ResNet18	-	11.69M	70.44	89.72
+ KW ($1\times$)	✓	11.93M	**74.77** (↑**4.33**)	**92.13** (↑**2.41**)
	✗	11.78M	70.49 (↑0.05)	89.84 (↑0.12)

4.3. Ablation Studies

For a better understanding of our method, we further conduct a lot of ablative experiments on ImageNet dataset, using the advanced training strategy proposed in ConvNeXt (Liu et al., 2022).

Effect of Kernel Partition. Thanks to the kernel partition component, our method can apply dense kernel assembling with a large number of kernel cells. In Table 5, we provide the ablative experiments on the ResNet18 backbone with KW ($1\times$) to study the efficacy of kernel partition. We can see, when removing kernel partition, the top-1 gain of our method to the baseline model sharply decreases from 4.33% to 0.05%, demonstrating its great importance to our method.

Effect of Warehouse Sharing Range in terms of Layer. To validate the effectiveness of the warehouse construction-with-sharing component, we first study warehouse sharing range in terms of layer by performing ablative experiments on the ResNet18 backbone with KW ($1\times$). From the results

Table 6: Effect of warehouse sharing range in terms of layer.

Models	Warehouse Sharing Range	Params	Top-1 Acc (%)	Top-5 Acc (%)
ResNet18	-	11.69M	70.44	89.72
+ KW (1×)	Within each stage	11.93M	**74.77** (↑**4.33**)	**92.13** (↑**2.41**)
	Within each layer	11.81M	74.34 (↑3.90)	91.82 (↑2.10)
	Without sharing	11.78M	72.49 (↑2.05)	90.81 (↑1.09)

Table 7: Effect of warehouse sharing range in terms of static kernel dimensions.

Models	Warehouse sharing range	Params	Top-1 Acc (%)	Top-5 Acc (%)
ResNet50	-	25.56M	78.44	94.24
+ KW (1×)	Different dimensioned kernels	28.05M	**80.38** (↑**1.94**)	**95.27** (↑**1.03**)
	Same dimensioned kernels	26.95M	79.80 (↑1.36)	95.01 (↑0.77)

Table 8: Effect of different attention functions.

Models	Attention Functions	Params	Top-1 Acc (%)	Top-5 Acc (%)
ResNet18	-	11.69M	70.44	89.72
+ KW (1×)	$z_{ij}/\sum_{p=1}^{n}\lvert z_{ip}\rvert$ (ours)	11.93M	**74.77** (↑**4.33**)	**92.13** (↑**2.41**)
	Softmax	11.93M	72.67 (↑2.23)	90.82 (↑1.10)
	Sigmoid	11.93M	72.09 (↑1.65)	90.70 (↑0.98)
	$max(z_{ij},0)/\sum_{p=1}^{n}\lvert z_{ip}\rvert$	11.93M	72.74 (↑2.30)	90.86 (↑1.14)

shown in Table 6, we can see that when sharing warehouse in a wider range, our method brings larger performance improvement to the baseline model. This clearly indicates that explicitly enhancing convolutional parameter dependencies either within the same or across the neighboring layers can strengthen the capacity of a ConvNet.

Effect of Warehouse Sharing Range in terms of Static Kernel Dimensions. For modern ConvNet backbones, a convolutional block in the same-stage mostly contains several static kernels having different dimensions ($k \times k \times c \times f$). Next, we perform ablative experiments on the ResNet50 backbone to study the effect of warehouse sharing range in terms of static kernel dimensions in the same-stage convolutional layers of a ConvNet. Results are summarized in Table 7, showing that warehouse sharing across convolutional layers having different dimensioned kernels performs better than only sharing warehouse across convolutional layers having the same dimensioned kernels. Combining the results in Table 6 and Table 7, we can conclude that enhancing the warehouse sharing between more kernel cells tends to achieve better performance for our method.

Effect of Different Attention Functions. Recall that our method relies on the proposed contrasting-driven attention function (CAF). To explore its role, we also conduct ablative experiments on the ResNet18 backbone to compare the performance of KernelWarehouse using different attention functions. According to the results in Table 8, the top-1 gain of our CAF $z_{ij}/\sum_{p=1}^{n}|z_{ip}|$ against popular attention function Softmax|Sigmoid reaches 2.10%|2.68%, and our CAF also outperforms another counterpart $max(z_{ij},0)/\sum_{p=1}^{n}|z_{ip}|$ by 2.03% top-1 gain. These experiments well validate the efficacy and importance of two design principles (clarified in the Method section) of our CAF.

Effect of Attentions Initialization Strategy. To help the optimization of KernelWarehouse in the early training stage, our CAF uses binary β_{ij} with temperature γ to initialize the scalar attentions. Next, we perform experiments on the ResNet18 backbone to study its role. As shown in Table 9, this attentions initialization strategy is essential to our method in learning contrasting and diverse attention relationships between linear mixtures and kernel cells, bringing 1.38% top-1 gain to the baseline model with KW (1×).

Table 9: Effect of attentions initialization strategy.

Models	Attentions Initialization Strategy	Params	Top-1 Acc (%)	Top-5 Acc (%)
ResNet18	-	11.69M	70.44	89.72
+ KW (1×)	✓	11.93M	**74.77** (↑**4.33**)	**92.13** (↑**2.41**)
	✗	11.93M	73.39 (↑2.95)	91.24 (↑1.52)

Table 10: Applying CAF to other dynamic convolution methods.

Models	Params	Attention Function	Top-1 Acc (%)	Top-5 Acc (%)
ResNet18	11.69M	-	70.44	89.72
+ DY-Conv (4×)	45.47M	Softmax	**73.82** (↑**3.38**)	**91.48** (↑**1.76**)
		Our CAF	73.74 (↑3.30)	91.45 (↑1.73)
+ ODConv (4×)	44.90M	Softmax	**74.45** (↑**4.01**)	**91.67** (↑**1.95**)
		Our CAF	74.27 (↑3.83)	91.62 (↑1.90)
+ KW (1×)	11.93M	Softmax	72.67 (↑2.23)	90.82 (↑1.10)
		Our CAF	**74.77** (↑**4.33**)	**92.13** (↑**2.41**)
+ KW (4×)	45.86M	Softmax	74.31 (↑3.87)	91.75 (↑2.03)
		Our CAF	**76.05** (↑**5.61**)	**92.68** (↑**2.96**)

Table 11: Applying KW to Vision Transformer backbones.

Models	Params	Top-1 Acc (%)	Top-5 Acc (%)
DeiT-Small	22.06M	79.78	94.99
+ KW (3/4×)	19.23M	79.94 (↑0.16)	95.05 (↑0.06)
+ KW (1×)	24.36M	**80.63** (↑**0.85**)	**95.24** (↑**0.25**)
DeiT-Tiny	5.72M	72.13	91.32
+ KW (1×)	6.39M	73.56 (↑1.43)	91.82 (↑0.50)
+ KW (4×)	20.44M	**76.51** (↑**4.38**)	**93.05** (↑**1.73**)

Applying CAF to Other Dynamic Convolution Methods. In Table 10, we provide experimental results for applying the proposed attention function CAF to existing top-performing dynamic convolution methods DY-Conv and ODConv. We can see that CAF gets slight model accuracy drop compared to the original Softmax function. This is because CAF is customized to fit three unique optimization properties of KernelWarehouse, as we discussed in the Method section.

Applying KW to Vision Transformer Backbones. One interesting question is whether our method can be generalized to Vision Transformer backbones. We study it by applying our method to two popular DeiT (Touvron et al., 2021) backbones. In the experiments, each of partitioned cells of weight matrices for "value and MLP" layers of each DeiT backbone is represented as a linear mixture of kernel warehouse shared across multiple multi-head self-attention blocks and MLP blocks, except the "query" and "key" matrices which are used to compute self-attention. From the results shown in Table 11, it can be seen that: (1) With a small parameter budget, e.g., $b = 3/4$, KW gets slightly improved model accuracy while reducing model

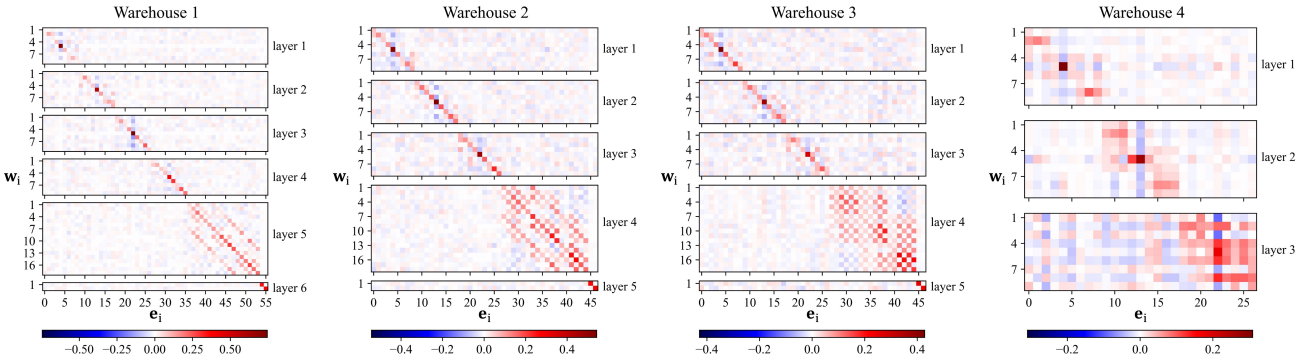


Figure 3: Visualization of statistical mean values of learnt attention α_{ij} in each warehouse. The results are obtained from the pre-trained ResNet18 model with KW ($1\times$) using the whole ImageNet validation set. Best viewed with zoom-in.

Table 12: Runtime model speed (frames per second (FPS)) comparison. All models are tested on an NVIDIA TITAN X GPU (with batch size 100 and input image size 224×224).

Models	Params	Top-1 Acc (%)	Runtime Speed (FPS)
ResNet50	25.56M	78.44	647.0
+ DY-Conv ($4\times$)	100.88M	79.00 (↑0.56)	322.7
+ ODConv ($4\times$)	90.67M	80.62 (↑2.18)	142.3
+ KW ($1/2\times$)	17.64M	79.30 (↑0.86)	227.8
+ KW ($1\times$)	28.05M	80.38 (↑1.94)	265.4
+ KW ($4\times$)	102.02M	81.05 (↑2.61)	191.1
MobileNetV2 ($1.0\times$)	3.50M	72.02	1410.8
+ DY-Conv ($4\times$)	12.40M	74.94 (↑2.92)	862.4
+ ODConv ($4\times$)	11.52M	75.42 (↑3.40)	536.5
+ KW ($1/2\times$)	2.65M	72.59 (↑0.57)	926.0
+ KW ($1\times$)	5.17M	74.68 (↑2.66)	798.7
+ KW ($4\times$)	11.38M	75.92 (↑3.90)	786.9

size of DeiT-Small; (2) With a larger parameter budget, e.g., $b = 4$, KW can significantly improve model accuracy, bringing 4.38% top-1 accuracy gain to DeiT-Tiny; (3) These performance trends are similar to our results on ConvNet backbones (see Table 2 and Table 4), demonstrating the appealing generalization ability of our method to different neural network architectures.

Visualization of Learnt Attentions. In Figure 3, we provide visualization results to analyze the statistical mean values of learnt attention α_{ij}, for better understanding our method. The results are obtained from the pre-trained ResNet18 model with KW ($1\times$). We can observe: (1) Each linear mixture can learn its own distribution of scalar attentions for different kernel cells; (2) In each warehouse, the maximum value of α_{ij} in each row mostly appears in the diagonal line throughout the whole warehouse. This indicates that our attentions initialization strategy can help our method to build rich one-to-one relationships between linear mixtures and kernel cells; (3) The attentions α_{ij} with higher absolute values for linear mixtures in the same layer have more overlaps than linear mixtures across different layers. This indicates that parameter dependencies within the same kernel are stronger than those across neighboring layers, which can be learned by our method.

Runtime Model Speed and Discussion. The above experimental results demonstrate that KernelWarehouse strikes a favorable trade-off between parameter efficiency and representation power. Table 12 further compares the runtime model speeds of different dynamic convolution methods. We can see, under the same convolutional parameter budget b, the runtime model speed of KernelWarehouse: (1) is slower than DY-Conv (vanilla dynamic convolution) on larger backbones like ResNet50; (2) is at a similar level to DY-Conv on lightweight backbones like MobileNetV2; (3) is always faster than ODConv (existing best-performing dynamic convolution method). The model speed gap of KernelWarehouse to DY-Conv is primarily due to the dense attentive mixture and assembling operations at the same-stage convolutional layers having a shared warehouse. Thanks to the parallel property of these operations, there are some potential optimization strategies to alleviate this gap. Given a ConvNet model pre-trained with KernelWarehouse, a customized optimization strategy is to adaptively allocate available Tensor Cores and CUDA Cores for a better trade-off of memory-intensive and compute-intensive operations in KernelWarehouse at different convolutional layers. Another potential optimization strategy is to reduce the number of kernel cells in a shared warehouse or reduce the warehouse sharing range, under the condition that the desired model size and accuracy can be still reached in real applications.

More experiments and visualizations are in the Appendix.

5. Conclusion

In this paper, we rethink the design of dynamic convolution and present KernelWarehouse. As a more general form of dynamic convolution, KernelWarehouse can improve the performance of modern ConvNets while enjoying parameter efficiency. Experiments on ImageNet and MS-COCO datasets show its great potential. We hope our work would inspire future research in dynamic convolution.

Acknowledgements

This work was done when Chao Li was an intern at Intel Labs China, supervised by Anbang Yao who led the project and the paper writing. We thank Intel Data Center & AI group's great support of their DGX-2 and DGX-A100 servers for training large models in this project.

Impact Statement

This paper presents KernelWarehouse, a more general form of dynamic convolution, which advances dynamic convolution research towards substantially better parameter efficiency and representation power. As far as we know, our work does not have any ethical impacts and potential societal consequences.

References

Bako, S., Vogels, T., McWilliams, B., Meyer, M., NovDollák, J., Harvill, A., Sen, P., DeRose, T., and Rousselle, F. Kernel-predicting convolutional networks for denoising monte carlo renderings. In *Siggraph*, 2017.

Chen, Y., Dai, X., Liu, M., Chen, D., Yuan, L., and Liu, Z. Dynamic convolution: Attention over convolution kernels. In *CVPR*, 2020.

Cubuk, E. D., Zoph, B., Shlens, J., and V, L. Q. Randaugment: Practical automated data augmentation with a reduced search space. In *CVPR Workshops*, 2020.

Diba, A., Sharma, V., Van Gool, L., and Stiefelhagen, R. Dynamonet: Dynamic action and motion network. In *ICCV*, 2019.

Dosovitskiy, A., Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N. An image is worth 16x16 words: Transformers for image recognition at scale. In *ICLR*, 2021.

Ha, D., Dai, A. M., and Le, Q. V. Hypernetworks. In *ICLR*, 2017.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *CVPR*, 2016.

He, K., Gkioxari, G., Dollár, P., and Girshick, R. Mask r-cnn. In *ICCV*, 2017.

He, S., Jiang, C., Dong, D., and Ding, L. Sd-conv: Towards the parameter-effciency of dynamic convolution. In *WACV*, 2023.

Howard, A., Sandler, M., Chu, G., Chen, L.-C., Chen, B., Tan, M., Wang, W., Zhu, Y., Pang, R., Vasudevan, V., Le, Q. V., and Adam, H. Searching for mobilenetv3. In *ICCV*, 2019.

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

Hu, J., Shen, L., Albanie, S., Sun, G., and Vedaldi, A. Gather-excite: Exploiting feature context in convolutional neural networks. In *NeurIPS*, 2018a.

Hu, J., Shen, L., and Sun, G. Squeeze-and-excitation networks. In *CVPR*, 2018b.

Huang, G., Liu, Z., van der Maaten, L., and Weinberger, K. Q. Densely connected convolutional networks. In *CVPR*, 2017.

Jaderberg, M., Simonyan, K., Zisserman, A., and Kavukcuoglu, K. Spatial transformer networks. In *NIPS*, 2015.

Jia, X., Brabandere, B. D., Tuytelaars, T., and Gool, L. V. Dynamic filter networks. In *NIPS*, 2016.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

Lee, H., Kim, H.-E., and Nam, H. Srm: A style-based recalibration module for convolutional neural networks. In *ICCV*, 2019.

Li, C., Zhou, A., and Yao, A. Omni-dimensional dynamic convolution. In *ICLR*, 2022.

Li, D., Yao, A., and Chen, Q. Psconv: Squeezing feature pyramid into one compact poly-scale convolutional layer. In *ECCV*, 2020.

Li, D., Hu, J., Wang, C., Li, X., She, Q., Zhu, L., Zhang, T., and Chen, Q. Involution: Inverting the inherence of convolution for visual recognition. In *CVPR*, 2021a.

Li, X., Wang, W., Hu, X., and Yang, J. Selective kernel networks. In *CVPR*, 2019.

Li, Y., Chen, Y., Dai, X., Liu, M., Chen, D., Yu, Y., Lu, Y., Liu, Z., Chen, M., and Vasconcelos, N. Revisiting dynamic convolution via matrix decomposition. In *ICLR*, 2021b.

Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., and Zitnick, C. L. Microsoft coco: Common objects in context. In *ECCV*, 2014.

Lin, X., Ma, L., Liu, W., and Chang, S.-F. Context-gated convolution. In *ECCV*, 2020.

Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., Lin, S., and Guo, B. Swin transformer: Hierarchical vision transformer using shifted windows. In *ICCV*, 2021.

Liu, Z., Mao, H., Wu, C.-Y., Feichtenhofer, C., Darrell, T., and Xie, S. A convnet for the 2020s. In *CVPR*, 2022.

Ma, N., Zhang, X., Huang, J., and Sun, J. Weightnet: Revisiting the design space of weight networks. In *ECCV*, 2020.

Mildenhall, B., Barron, J. T., Chen, J., Sharlet, D., Ng, R., and Carroll, R. Burst denoising with kernel prediction networks. In *CVPR*, 2018.

Munkhdalai, T. and Yu, H. Meta networks. In *ICML*, 2017.

Park, J., Woo, S., Lee, J.-Y., and Kweon, I. S. Bam: Bottleneck attention module. In *BMVC*, 2018.

Quader, N., Bhuiyan, M. M. I., Lu, J., Dai, P., and Li, W. Weight excitation: Built-in attention mechanisms in convolutional neural networks. In *ECCV*, 2020.

Radosavovic, I., Kosaraju, R. P., Girshick, R., He, K., and Dollár, P. Designing network design spaces. In *CVPR*, 2020.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Li, F.-F. Imagenet large scale visual recognition challenge. *IJCV*, 2015.

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. Mobilenetv2: Inverted residuals and linear bottlenecks. In *CVPR*, 2018.

Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. Going deeper with convolutions. In *CVPR*, 2015.

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. Rethinking the inception architecture for computer vision. In *CVPR*, 2016.

Tan, M. and Le, Q. Efficientnet: Rethinking model scaling for convolutional neural networks. In *ICML*, 2019a.

Tan, M. and Le, Q. V. Mixconv: Mixed depthwise convolutional kernels. In *BMVC*, 2019b.

Touvron, H., Cord, M., Douze, M., Massa, F., Sablayrolles, A., and Jégou, H. Training data-efficient image transformers & distillation through attention. In *ICML*, 2021.

Wang, F., Jiang, M., Qian, C., Yang, S., Li, C., Zhang, H., Wang, X., and Tang, X. Residual attention network for image classification. In *CVPR*, 2017.

Wang, J., Chen, K., Xu, R., Liu, Z., Chen, C. L., and Lin, D. Carafe: Content-aware reassembly of features. In *ICCV*, 2019.

Wang, Q., Wu, B., Zhu, P., Li, P., Zuo, W., and Hu, Q. Eca-net: Efficient channel attention for deep convolutional neural networks. In *CVPR*, 2020.

Woo, S., Park, J., Lee, J.-Y., and Kweon, I. S. Cbam: Convolutional block attention module. In *ECCV*, 2018.

Xie, S., Girshick, R., Dollár, P., Tu, Z., and He, K. Aggregated residual transformations for deep neural networks. In *CVPR*, 2017.

Yang, B., Bender, G., Le, Q. V., and Ngiam, J. Condconv: Conditionally parameterized convolutions for efficient inference. In *NeurIPS*, 2019a.

Yang, Z., Yunhe, W., Chen, H., Liu, C., Shi, B., Xu, C., Xu, C., and Xu, C. Legonet: Efficient convolutional neural networks with lego filters. In *ICML*, 2019b.

Yun, S., , Han, D., Oh, S. J., Chun, S., Choe, J., and Yoo, Y. Cutmix: Regularization strategy to train strong classifiers with localizable features. In *ICCV*, 2019.

Zhang, H., Cisse, M., Dauphin, Y. N., and Lopez-Paz, D. mixup: Beyond empirical risk minimization. In *ICLR*, 2018a.

Zhang, X., Zhou, X., Lin, M., and Sun, J. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *CVPR*, 2018b.

Zhong, Z., Zheng, L., Kang, G., Li, S., and Yang, Y. Random erasing data augmentation. In *AAAI*, 2020.

A. Appendix

A.1. Datasets and Implementation Details

A.1.1. Image Classification on ImageNet

Recall that we use ResNet (He et al., 2016), MobileNetV2 (Sandler et al., 2018) and ConvNeXt (Liu et al., 2022) families for the main experiments on ImageNet dataset (Russakovsky et al., 2015), which consists of over 1.2 million training images and 50,000 validation images with 1,000 object categories. We use an input image resolution of 224×224 for both training and evaluation. All the input images are standardized with mean and standard deviation per channel. For evaluation, we report top-1 and top-5 recognition rates of a single 224×224 center crop on the ImageNet validation set. All the experiments are performed on the servers having 8 GPUs. Specifically, the models of ResNet18, MobileNetV2 ($1.0\times$), MobileNetV2 ($0.5\times$) are trained on the servers with 8 NVIDIA Titan X GPUs. The models of ResNet50, ConvNeXt-Tiny are trained on the servers with 8 NVIDIA Tesla V100-SXM3 or A100 GPUs. The training setups for different models are as follows.

Training setup for ResNet models with the traditional training strategy. All the models are trained by the stochastic gradient descent (SGD) optimizer for 100 epochs, with a batch size of 256, a momentum of 0.9 and a weight decay of 0.0001. The initial learning rate is set to 0.1 and decayed by a factor of 10 for every 30 epoch. Horizontal flipping and random resized cropping are used for data augmentation. For KernelWarehouse, the temperature τ linearly reduces from 1 to 0 in the first 10 epochs.

Training setup for ResNet and ConvNeXt models with the advanced training strategy. Following the settings of ConvNeXt (Liu et al., 2022), all the models are trained by the AdamW optimizer with $\beta_1 = 0.9, \beta_2 = 0.999$ for 300 epochs, with a batch size of 4096, a momentum of 0.9 and a weight decay of 0.05. The initial learning rate is set to 0.004 and annealed down to zero following a cosine schedule. Randaugment (Cubuk et al., 2020), mixup (Zhang et al., 2018a), cutmix (Yun et al., 2019), random erasing (Zhong et al., 2020) and label smoothing (Szegedy et al., 2016) are used for augmentation. For KernelWarehouse, the temperature τ linearly reduces from 1 to 0 in the first 20 epochs.

Training setup for MobileNetV2 models. All the models are trained by the SGD optimizer for 150 epochs, with a batch size of 256, a momentum of 0.9 and a weight decay of 0.00004. The initial learning rate is set to 0.1 and annealed down to zero following a cosine schedule. Horizontal flipping and random resized cropping are used for data augmentation. For KernelWarehouse, the temperature τ linearly reduces from 1 to 0 in the first 10 epochs.

A.1.2. Object Detection and Instance Segmentation on MS-COCO

Recall that we conduct comparative experiments for object detection and instance segmentation on the MS-COCO 2017 dataset (Lin et al., 2014), which contains 118,000 training images and 5,000 validation images with 80 object categories. We adopt Mask R-CNN as the detection framework, ResNet50 and MobileNetV2 ($1.0\times$) built with different dynamic convolution methods as the backbones which are pre-trained on ImageNet dataset. All the models are trained with a batch size of 16 and standard $1\times$ schedule on the MS-COCO dataset using multi-scale training. The learning rate is decreased by a factor of 10 at the 8th and the 11th epoch of total 12 epochs. For a fair comparison, we adopt the same settings including data processing pipeline and hyperparameters for all the models. All the experiments are performed on the servers with 8 NVIDIA Tesla V100 GPUs. The attentions initialization strategy is not used for KernelWarehouse during fine-tuning to avoid disrupting the learnt relationships of the pre-trained models between kernel cells and linear mixtures. For evaluation, we report both bounding box Average Precision (AP) and mask AP on the MS-COCO 2017 validation set, including AP_{50}, AP_{75} (AP at different IoU thresholds) and AP_S, AP_M, AP_L (AP at different scales).

A.2. Visualization Examples of Attentions Initialization Strategy

Recall that we adopt an attentions initialization strategy for KernelWarehouse using τ and β_{ij}. It forces the scalar attentions to be one-hot in the early training stage for building one-to-one relationships between kernel cells and linear mixtures. To give a better understanding of this strategy, we provide visualization examples for KW ($1\times$), KW ($2\times$) and KW ($1/2\times$), respectively. We also provide a set of ablative experiments to compare our proposed strategy with other alternatives.

Attentions Initialization for KW ($1\times$). A visualization example of attentions initialization strategy for KW ($1\times$) is shown in Figure 4. In this example, a warehouse $\mathbf{E} = \{\mathbf{e}_1, \ldots, \mathbf{e}_6, \mathbf{e}_z\}$ is shared to 3 neighboring convolutional layers with kernel dimensions of $k \times k \times 2c \times f$, $k \times 3k \times c \times f$ and $k \times k \times c \times f$, respectively. The kernel dimensions are selected for simple illustration. The kernel cells have the same dimensions of $k \times k \times c \times f$. Note that the kernel cell $\mathbf{e}_z$ doesn't really

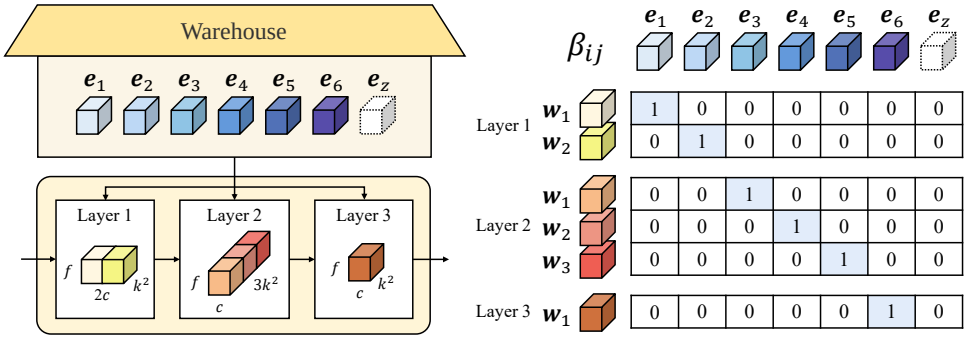


Figure 4: A visualization example of attentions initialization strategy for KW ($1\times$), where both n and m_t equal to 6. It helps the ConvNet to build one-to-one relationships between kernel cells and linear mixtures in the early training stage according to our setting of β_{ij}. $\mathbf{e}_z$ is a kernel cell that doesn't really exist and it keeps as a zero matrix constantly. In the beginning of the training process when temperature τ is 1, a ConvNet built with KW ($1\times$) can be roughly seen as a ConvNet with standard convolutions.

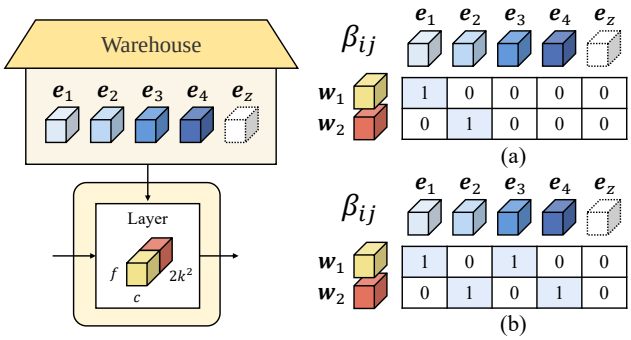


Figure 5: Visualization examples of attentions initialization strategies for KW ($2\times$), where $n = 4$ and $m_t = 2$. (a) our proposed strategy builds one-to-one relationships between kernel cells and linear mixtures; (b) an alternative strategy which builds two-to-one relationships between kernel cells and linear mixtures.

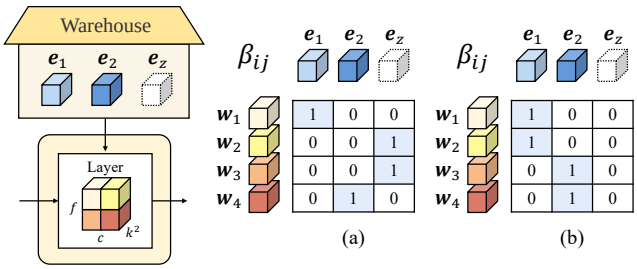


Figure 6: Visualization examples of attentions initialization strategies for KW ($1/2\times$), where $n = 2$ and $m_t = 4$. (a) our proposed strategy builds one-to-one relationships between kernel cells and linear mixtures; (b) an alternative strategy which builds one-to-two relationships between kernel cells and linear mixtures.

exist and it keeps as a zero matrix constantly. It is only used for attentions normalization but not assembling kernels. This kernel is mainly designed for attentions initialization when $b < 1$ and not counted in the number of kernel cells n. In the early training stage, we adopt a strategy to explicitly force every linear mixture to build relationship with one specified kernel cell according to our setting of β_{ij}. As shown in Figure 4, we assign one of $\mathbf{e}_1, \ldots, \mathbf{e}_6$ in the warehouse to each of the 6 linear mixtures at the 3 convolutional layers without repetition. So that in the beginning of the training process when temperature τ is 1, a ConvNet built with KW ($1\times$) can be roughly seen as a ConvNet with standard convolutions. The results of Table 9 in the main manuscript validate the effectiveness of our proposed attentions initialization strategy. Here, we compare it with another alternative. In this alternative strategy, we force every linear mixture to build relationships with all the kernel cells equally by setting all the β_{ij} to be 1. The results are shown in Table 13. The all-to-one strategy demonstrates similar performance with KernelWarehouse without using any attentions initialization strategy, while our proposed strategy outperforms it by 1.41% top-1 gain.

Attentions Initialization for KW ($2\times$). For KernelWarehouse with $b > 1$, we adopt the same strategy for initializing attentions used in KW ($1\times$). Figure 5(a) provides a visualization example of attentions initialization strategy for KW ($2\times$). For building one-to-one relationships, we assign $\mathbf{e}_1$ to $\mathbf{w}_1$ and $\mathbf{e}_2$ to $\mathbf{w}_2$, respectively. When $b > 1$, another reasonable strategy is to assign multiple kernel cells to every linear mixture without repetition, which is shown in Figure 5(b). We use the ResNet18 backbone based on KW ($4\times$) to compare the two strategies. From the results in Table 13, we can see that our

Algorithm 1: Implementation of KernelWarehouse

Part-A: Kernel Partition and Warehouse Construction-with-Sharing

Require : network $\mathbf{M}$ consisting of S convolutional stages, parameter budget b

for $s \leftarrow 1$ **to** S **do**
  $\{\mathbf{W}_i \in \mathbb{R}^{k_i \times k_i \times c_i \times f_i}\}_{i=1}^{l}$= static kernels in the s-th stage of $\mathbf{M}$
  $k_e, c_e, f_e \leftarrow$**cdd**$(\{k_i\}_{i=1}^{l})$,**cdd**$(\{c_i\}_{i=1}^{l})$,**cdd**$(\{f_i\}_{i=1}^{l})$
  $m_t \leftarrow 0$
  for $i \leftarrow 1$ **to** l **do**
    $\{\mathbf{w}_j \in \mathbb{R}^{k_e \times k_e \times c_e \times f_e}\}_{j=1}^{m} \leftarrow$**kernel_partition**$(\mathbf{W}_i, k_e, c_e, f_e)$
    $\mathbf{W}_i \leftarrow \mathbf{w}_1 \cup \cdots \cup \mathbf{w}_m$, and $\forall i, j \in \{1, \ldots, m\}, i \neq j, w_i \cap w_j = \varnothing$
    $m \leftarrow k_i k_i c_i f_i / k_e k_e c_e f_e$
    $m_t \leftarrow m + m_t$
  end
  $n \leftarrow b m_t$
  $\mathbf{E}_s \leftarrow \{\mathbf{e}_i \in \mathbb{R}^{k_e \times k_e \times c_e \times f_e}\}_{i=1}^{n}$
end
$\mathbf{E} \leftarrow \{\mathbf{E}_1, \ldots, \mathbf{E}_S\}$

Return : network $\mathbf{M}$ with partitioned kernels, set $\mathbf{E}$ consisting of S warehouses

Part-B: Kernel Assembling for Single Same-Stage Convolutional Layer

Require : input $\mathbf{x}$, attention module ϕ, warehouse $\mathbf{E} = \{\mathbf{e}_i\}_{i=1}^{n}$, linear mixtures $\{\mathbf{w}_i\}_{i=1}^{m}$

$\boldsymbol{\alpha} \leftarrow \phi(\mathbf{x})$
for $i \leftarrow 1$ **to** m **do**
  $\mathbf{w_i} \leftarrow \alpha_{i1}\mathbf{e}_1 + \cdots + \alpha_{in}\mathbf{e}_n$
end
$\mathbf{W} \leftarrow \mathbf{w}_1 \cup \cdots \cup \mathbf{w}_m$, and $\forall i, j \in \{1, \ldots, m\}, i \neq j, w_i \cap w_j = \varnothing$

Return : assembled kernel $\mathbf{W}$

Table 13: Ablation of KernelWarehouse with different attentions initialization strategies.

Models	Attentions Initialization Strategies	Params	Top-1 Acc (%)	Top-5 Acc (%)
ResNet18	-	11.69M	70.44	89.72
+ KW (1×)	1 kernel cell to 1 linear mixture	11.93M	**74.77** (↑**4.33**)	**92.13** (↑**2.41**)
	all the kernel cells to 1 linear mixture	11.93M	73.36 (↑2.92)	91.41 (↑1.69)
	without attentions initialization	11.93M	73.39 (↑2.95)	91.24 (↑1.52)
+ KW (4×)	1 kernel cell to 1 linear mixture	45.86M	**76.05** (↑**5.61**)	**92.68** (↑**2.96**)
	4 kernel cells to 1 linear mixture	45.86M	76.03 (↑5.59)	92.53 (↑2.81)
+ KW (1/2×)	1 kernel cell to 1 linear mixture	7.43M	**73.33** (↑**2.89**)	**91.42** (↑**1.70**)
	1 kernel cell to 2 linear mixtures	7.43M	72.89 (↑2.45)	91.34 (↑1.62)

one-to-one strategy performs better.

Attentions Initialization for KW $(1/2\times)$**.** For KernelWarehouse with $b < 1$, the number of kernel cells is less than that of linear mixtures, meaning that we cannot adopt the same strategy used for $b \geq 1$. Therefore, we only assign one of the total n kernel cells in the warehouse to n linear mixtures respectively without repetition. And we assign $\mathbf{e}_z$ to all of the remaining linear mixtures. The visualization example for KW $(1/2\times)$ is shown in Figure 6(a). When temperature τ is 1, a ConvNet built with KW $(1/2\times)$ can be roughly seen as a ConvNet with group convolutions (groups=2). We also provide comparison results between our proposed strategy and another alternative strategy which assigns one of the n kernel cells to every 2 linear mixtures without repetition. As shown in Table 13, our one-to-one strategy achieves better result again, showing that introducing an extra kernel $\mathbf{e}_z$ for $b < 1$ can help the ConvNet learn more appropriate relationships between kernel cells and linear mixtures. When assigning one kernel cell to multiple linear mixtures, a ConvNet could not balance the relationships between them well.

A.3. Design Details of KernelWarehouse

In this section, we describe the design details of our KernelWarehouse. The corresponding values of m and n for each of our trained models are provided in the Table 14. Note that the values of m and n are naturally determined according to our setting of the dimensions of the kernel cells, the layers to share warehouses and b. Algorithm 1 shows the implementation of KernelWarehouse, given a ConvNet backbone and the desired convolutional parameter budget b.

Design details of Attention Module of KernelWarehouse. Following existing dynamic convolution methods, KernelWare-

Table 14: The values of m and n for the ResNet18, ResNet50, ConvNeXt-Tiny, MobileNetV2 (1.0×) and MobileNetV2 (0.5×) backbones based on KernelWarehouse.

Backbones	b	m	n
ResNet18	1/4	224, 188, 188, 108	56, 47, 47, 27
	1/2	224, 188, 188, 108	112, 94, 94, 54
	1	56, 47, 47, 27	56, 47, 47, 27
	2	56, 47, 47, 27	112, 94, 94, 54
	4	56, 47, 47, 27	224, 188, 188, 108
ResNet50	1/2	348, 416, 552, 188	174, 208, 276, 94
	1	87, 104, 138, 47	87, 104, 138, 47
	4	87, 104, 138, 47	348, 416, 552, 188
ConvNeXt-Tiny	1	16,4,4,4,147,24,147,24,147,24,147,24,147,24,147,24	16,4,4,4,147,24,147,24,147,24,147,24,147,24,147,24
	3/4	16,4,4,4,147,24,147,24,147,24,147,24,147,96,147,96	16,4,4,4,147,24,147,24,147,24,147,24,147,48,147,48
MobileNetV2 (1.0×) MobileNetV2 (0.5×)	1/2	9, 36, 18, 27, 36, 27, 12, 27, 80, 40	9, 36, 18, 27, 36, 27, 6, 27, 40, 20
	1	9, 36, 34, 78, 18, 42, 27, 102, 36, 120, 27, 58, 27	9, 36, 34, 78, 18, 42, 27, 102, 36, 120, 27, 58, 27
	4	9, 36, 11, 1, 2, 18, 7, 3, 27, 4, 4, 36, 9, 3, 27, 11, 3, 27, 20	36, 144, 44, 4, 8, 72, 28, 12, 108, 16, 16, 144, 36, 12, 108, 44, 12, 108, 80

Table 15: The example of warehouse sharing for the ResNet18 backbone based on KW (1×) according to the original stages and reassigned stages.

Dimensions of Kernel Cells	Original Stages	Layers	Reassigned Stages	Dimensions of Kernel Cells
1×1×64×64	1	3×3×64×64	1	1×1×64×64
		3×3×64×64		
		3×3×64×64		
		3×3×64×64		
1×1×64×128	2	3×3×64×128		
		3×3×128×128	2	1×1×128×128
		3×3×128×128		
		3×3×128×128		
1×1×128×256	3	3×3×128×256		
		3×3×256×256	3	1×1×256×256
		3×3×256×256		
		3×3×256×256		
1×1×256×512	4	3×3×256×512		
		3×3×512×512	4	1×1×512×512
		3×3×512×512		
		3×3×512×512		

house also adopts a compact SE-typed structure as the attention module $\phi(x)$ (illustrated in Figure 1) to generate attentions for weighting kernel cells in a warehouse. For any convolutional layer with a static kernel $\mathbf{W}$, it starts with a channel-wise global average pooling (GAP) operation that maps the input $\mathbf{x}$ into a feature vector, followed by a fully connected (FC) layer, a rectified linear unit (ReLU), another FC layer, and a contrasting-driven attention function (CAF). The first FC layer reduces the length of the feature vector by 16, and the second FC layer generates m sets of n feature logits in parallel which are finally normalized by our CAF set by set.

Design details of KernelWarehouse on ResNet18. Recall that in KernelWarehouse, a warehouse is shared to all same-stage convolutional layers. While the layers are originally divided into different stages according to the resolutions of their input feature maps, the layers are divided into different stages according to their kernel dimensions in our KernelWarehouse. In our implementation, we usually reassign the first layer (or the first two layers) in each stage to the previous stage. An example for the ResNet18 backbone based on KW (1×) is given in Table 15. By reassigning the layers, we can avoid the condition that all the other layers have to be partitioned according to a single layer because of the greatest common dimension divisors. For the ResNet18 backbone, we apply KernelWarehouse to all the convolutional layers except the first one. In each stage, the corresponding warehouse is shared to all of its convolutional layers. For KW (1×), KW (2×) and KW (4×), we use the greatest common dimension divisors for static kernels as the uniform kernel cell dimensions for kernel partition. For KW ($1/2\times$) and KW ($1/4\times$), we use half of the greatest common dimension divisors.

Design details of KernelWarehouse on ResNet50. For the ResNet50 backbone, we apply KernelWarehouse to all the convolutional layers except the first two layers. In each stage, the corresponding warehouse is shared to all of its convolutional layers. For KW (1×) and KW (4×), we use the greatest common dimension divisors for static kernels as the uniform kernel cell dimensions for kernel partition. For KW ($1/2\times$), we use half of the greatest common dimension divisors.

Design details of KernelWarehouse on ConvNeXt-Tiny. For the ConvNeXt backbone, we apply KernelWarehouse to all

Table 16: Comparison of memory requirements of DY-Conv, ODConv and KernelWarehouse for training and inference. For ResNet50, we set batch size to 128|100 for each gpu during training|inference; for MobileNetV2($1.0\times$), we set batch size to 32|100 for each gpu during training|inference.

Models	Params	Training Memory (batch size=128)	Inference Memory (batch size=100)
ResNet50	25.56M	11,084 MB	1,249 MB
+ DY-Conv ($4\times$)	100.88M	24,552 MB	2,062 MB
+ ODConv ($4\times$)	90.67M	31,892 MB	5,405 MB
+ KW ($1/2\times$)	17.64M	23,323 MB	2,121 MB
+ KW ($1\times$)	28.05M	23,231 MB	2,200 MB
+ KW ($4\times$)	102.02M	24,905 MB	2,762 MB

Models	Params	Training Memory (batch size=32)	Inference Memory (batch size=100)
MobileNetV2 ($1.0\times$)	3.50M	2,486 MB	1,083 MB
+ DY-Conv ($4\times$)	12.40M	2,924 MB	1,151 MB
+ ODConv ($4\times$)	11.52M	4,212 MB	1,323 MB
+ KW ($1/2\times$)	2.65M	3,002 MB	1,076 MB
+ KW ($1\times$)	5.17M	2,823 MB	1,096 MB
+ KW ($4\times$)	11.38M	2,916 MB	1,144 MB

the convolutional layers. We partition the 9 blocks in the third stage of the ConvNeXt-Tiny backbone into three stages with the equal number of blocks. In each stage, the corresponding three warehouses are shared to the point-wise convolutional layers, the depth-wise convolutional layers and the downsampling layer, respectively. For KW ($1\times$), we use the greatest common dimension divisors for static kernels as the uniform kernel cell dimensions for kernel partition. For KW ($3/4\times$), we apply KW ($1/2\times$) to the point-wise convolutional layers in the last two stages of ConvNeXt backbone using half of the greatest common dimension divisors. And we apply KW ($1\times$) to the other layers using the greatest common dimension divisors.

Design details of KernelWarehouse on MobileNetV2. For the MobileNetV2 ($1.0\times$) and MobileNetV2 ($0.5\times$) backbones based on KW ($1\times$) and KW ($4\times$), we apply KernelWarehouse to all the convolutional layers. For MobileNetV2 ($1.0\times$, $0.5\times$) based on KW ($1\times$), the corresponding two warehouses are shared to the point-wise convolutional layers and the depth-wise convolutional layers in each stage, respectively. For MobileNetV2 ($1.0\times$, $0.5\times$) based on KW ($4\times$), the corresponding three warehouses are shared to the depth-wise convolutional layers, the point-wise convolutional layers for channel expansion and the point-wise convolutional layers for channel reduction in each stage, respectively. We use the greatest common dimension divisors for static kernels as the uniform kernel cell dimensions for kernel partition. For the MobileNetV2 ($1.0\times$) and MobileNetV2 ($0.5\times$) backbones based on KW ($1/2\times$), we take the parameters in the attention modules and classifier layer into consideration in order to reduce the total number of parameters. We apply KernelWarehouse to all the depth-wise convolutional layers, the point-wise convolutional layers in the last two stages and the classifier layer. We set $b = 1$ for the point-wise convolutional layers and $b = 1/2$ for the other layers. For the depth-wise convolutional layers, we use the greatest common dimension divisors for static kernels as the uniform kernel cell dimensions for kernel partition. For the point-wise convolutional layers, we use half of the greatest common dimension divisors. For the classifier layer, we use the kernel cell dimensions of 1000×32.

A.4. More Experiments for Studying Other Potentials of KernelWarehouse

In this section, we provide a lot of extra experiments conducted for studying other potentials of KernelWarehouse.

Comparison of Memory Requirements. From the table 16, we can observe that, for both training and inference, the memory requirements of our method are very similar to those of DY-Conv, and are much smaller than those for ODConv (that generates attention weights along all four dimensions including the input channel number, the output channel number, the spatial kernel size and the kernel number, rather than one single dimension as DY-Conv and KernelWarehouse), showing that our method does not have a potential limitation on memory requirements compared to existing top-performing dynamic convolution methods. The reason is: although KernelWarehouse introduces dense attentive mixturing and assembling operations at the same-stage convolutional layers having a shared warehouse, the memory requirement for these operations is significantly smaller than that for convolutional feature maps and the memory requirement for attention weights are also significantly smaller than that for convolutional weights, under the same convolutional parameter budget b.

Combining KernelWarehouse with ODConv. The improvement of KernelWarehouse to ODConv could be further boosted by a simple combination of KernelWarehouse and ODConv to compute attention weights for KernelWarehouse along the aforementioned four dimensions instead of one single dimension. We add experiments to explore this potential, and the results are summarized in the Table 17. We can see that, on the ImageNet dataset with MobileNetV2 ($1.0\times$) backbone, combining ODConv with KernelWarehouse ($4\times$) further brings 1.12% absolute top-1 improvement to ODConv ($4\times$) while retaining the similar model size.

Table 17: Effect of combining KernelWarehouse and ODConv, where KW* denotes attention function which combines KernelWarehouse and ODConv.

Models	Params	Top-1 Acc (%)	Top-5 Acc (%)
MobileNetV2 ($1.0\times$)	3.50M	72.02	90.43
+ ODConv ($4\times$)	11.52M	75.42 (↑3.40)	92.18 (↑1.75)
+ KW ($4\times$)	11.38M	75.92 (↑3.90)	92.22 (↑1.79)
+ KW*($4\times$)	12.51M	**76.54** (↑**4.52**)	**92.35** (↑**1.92**)

A.5. More Visualization Results for Learnt Attentions of KernelWarehouse

In the main manuscript, we provide visualization results of learnt attention values α_{ij} for the ResNet18 backbone based on KW ($1\times$) (see Figure 3 in the main manuscript). For a better understanding of KernelWarehouse, we provide more visualization results in this section, covering different alternative attention functions, alternative initialization strategies and values of b. For all the results, the statistical mean values of learnt attention α_{ij} are obtained using all of the 50,000 images on the ImageNet validation dataset.

Visualization Results for KernelWarehouse with Different Attention Functions. The visualization results for KernelWarehouse with different attention functions are shown in Figure 7, which are corresponding to the comparison results of Table 8 in the main manuscript. From which we can observe that: (1) for all of the attention functions, the maximum value of α_{ij} in each row mostly appears in the diagonal line throughout the whole warehouse. It indicates that our proposed attentions initialization strategy also works for the other three attention functions, which helps our KernelWarehouse to build one-to-one relationships between kernel cells and linear mixtures; (2) with different attention functions, the scalar attentions learnt by KernelWarehouse are obviously different, showing that the attention function plays an important role in our design; (3) compared to the other three functions, the maximum value of α_{ij} in each row tends to be relatively lower for our design (shown in Figure 7(a)). It indicates that the introduction of negative values for scalar attentions can help the ConvNet to enhance warehouse sharing, where each linear mixture not only focuses on the kernel cell assigned to it.

Visualization Results for KernelWarehouse with Attentions Initialization Strategies. The visualization results for KernelWarehouse with different attentions initialization strategies are shown in Figure 8, Figure 9 and Figure 10, which are corresponding to the comparison results of Table 13. From which we can observe that: (1) with all-to-one strategy or without initialization strategy, the distribution of scalar attentions learnt by KernelWarehouse seems to be disordered, while our proposed strategy can help the ConvNet learn more appropriate relationships between kernel cells and linear mixtures; (2) for KW ($4\times$) and KW ($1/2\times$), it's hard to directly determine which strategy is better only according to the visualization results. While the results demonstrate that the learnt attentions of KernelWarehouse are highly related to our setting of α_{ij}; (3) for KW ($1\times$), KW ($4\times$) and KW ($1/2\times$) with our proposed initialization strategy, some similar patterns of the value distributions can be found. For example, the maximum value of α_{ij} in each row mostly appears in the diagonal line throughout the whole warehouse. It indicates that our proposed strategy can help the ConvNet learn stable relationships between kernel cells and linear mixtures.

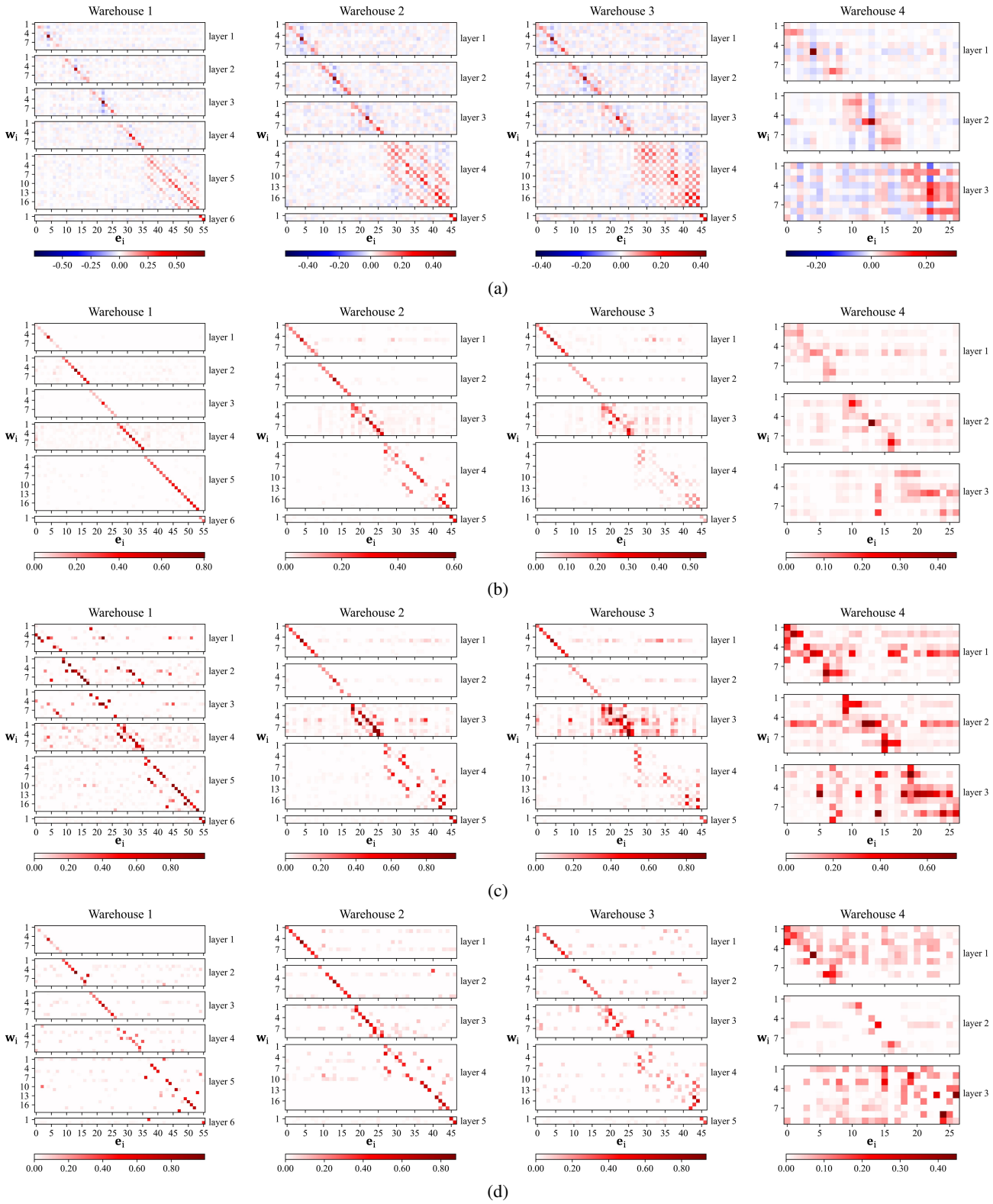


Figure 7: Visualization of statistical mean values of learnt attention α_{ij} in each warehouse for KernelWarehouse with different attention functions. The results are obtained from the pre-trained ResNet18 backbone with KW ($1\times$) for all of the 50,000 images on the ImageNet validation set. Best viewed with zoom-in. The attention functions for the groups of visualization results are as follows: (a) $z_{ij}/\sum_{p=1}^{n}|z_{ip}|$ (our design); (b) softmax; (c) sigmoid; (d) $max(z_{ij},0)/\sum_{p=1}^{n}|z_{ip}|$.

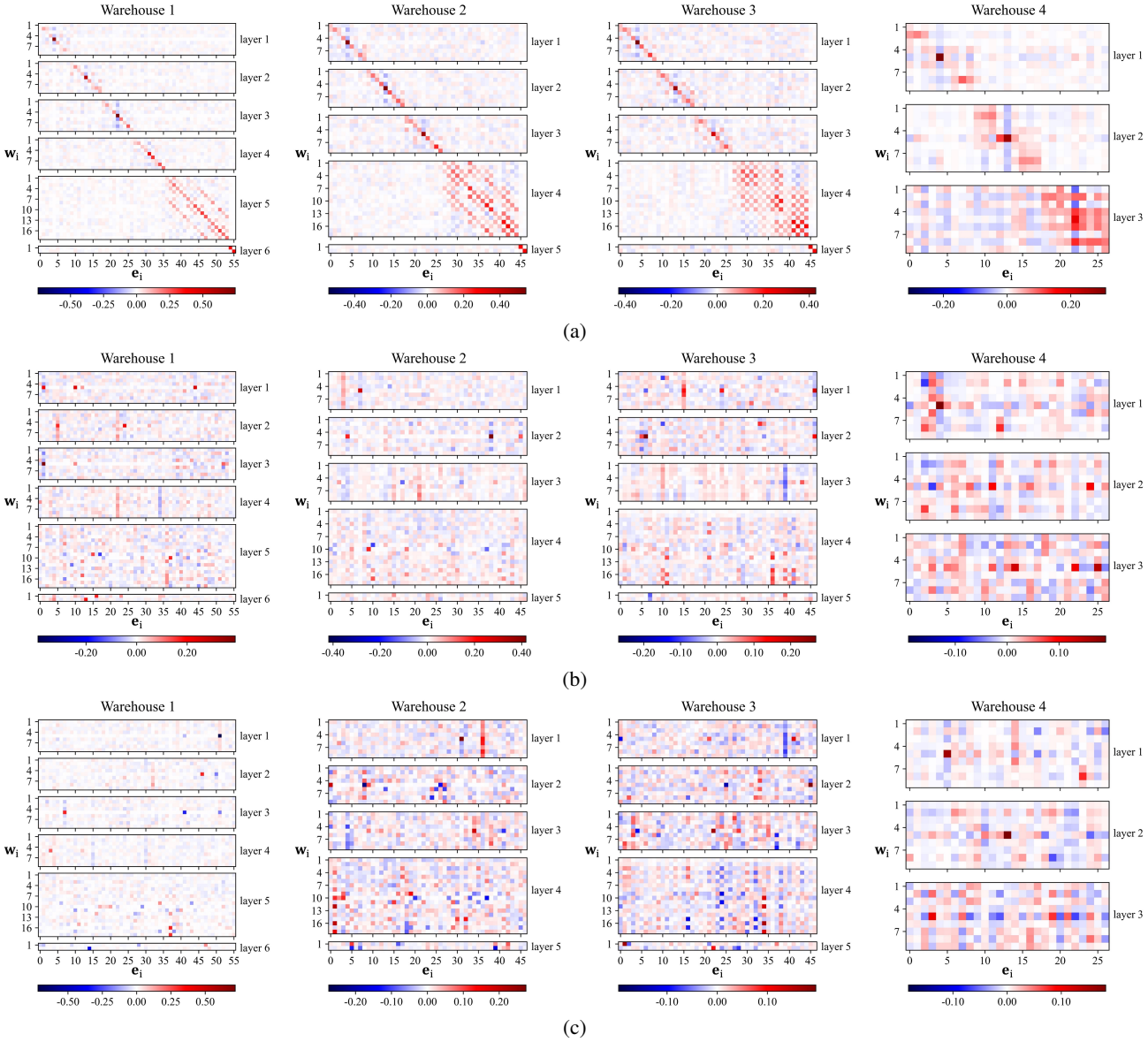


Figure 8: Visualization of statistical mean values of learnt attention α_{ij} in each warehouse for KernelWarehouse with different attentions initialization strategies. The results are obtained from the pre-trained ResNet18 backbone with KW ($1\times$) for all of the 50,000 images on the ImageNet validation set. Best viewed with zoom-in. The attentions initialization strategies for the groups of visualization results are as follows: (a) building one-to-one relationships between kernel cells and linear mixtures; (b) building all-to-one relationships between kernel cells and linear mixtures; (c) without initialization.

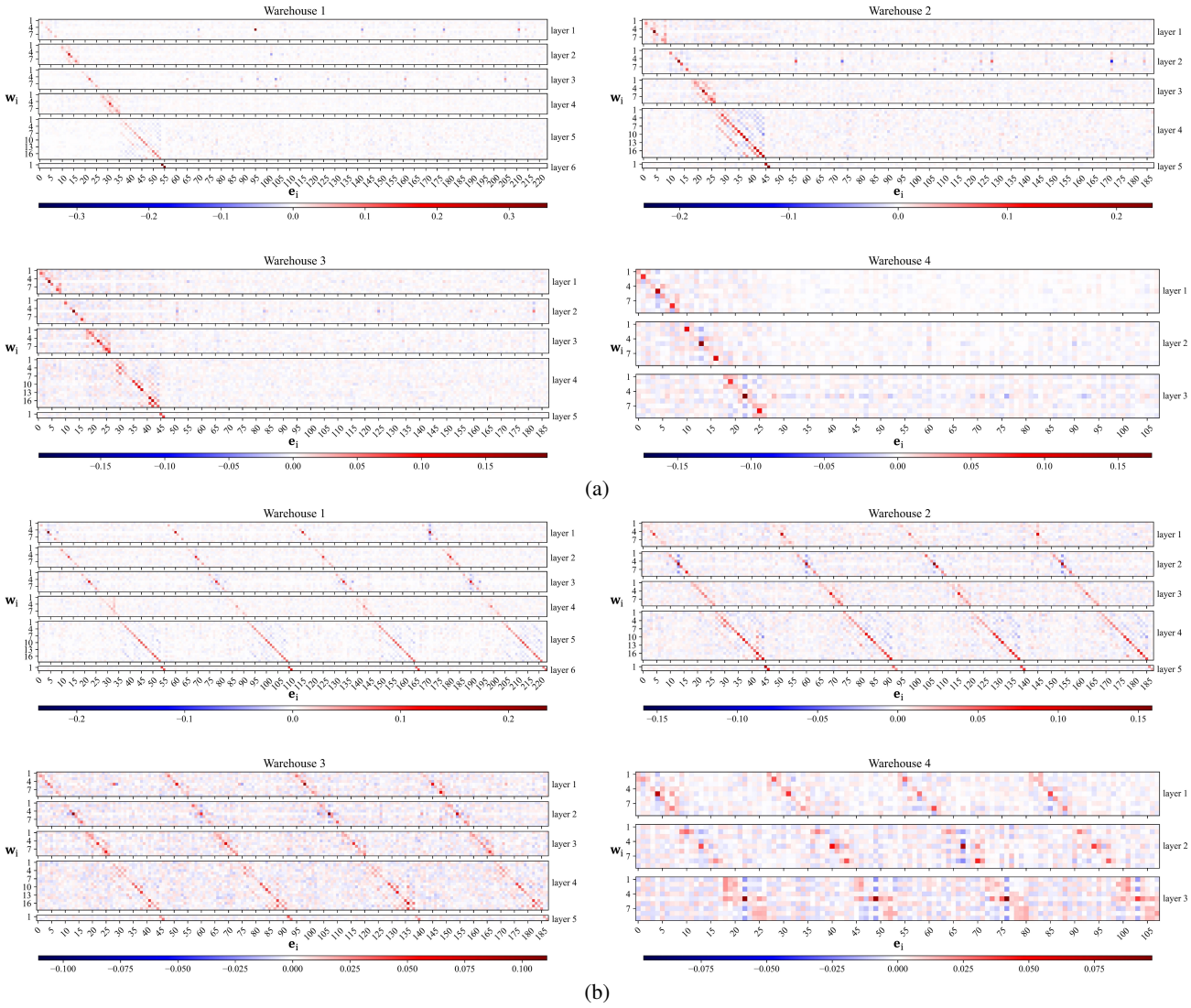
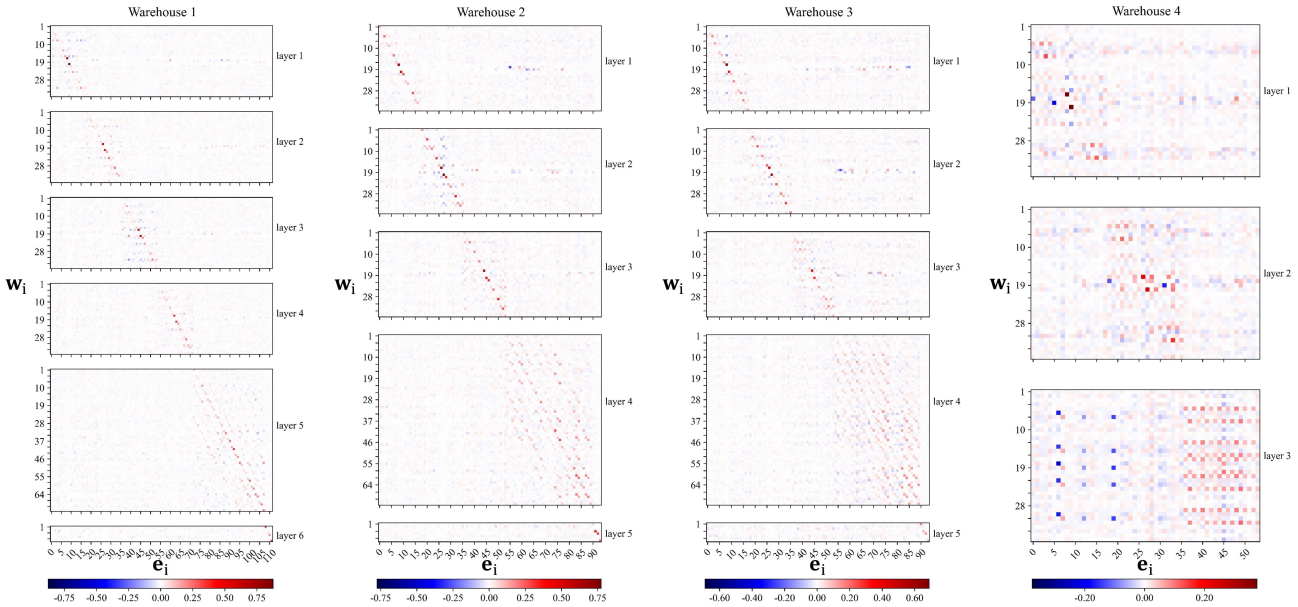
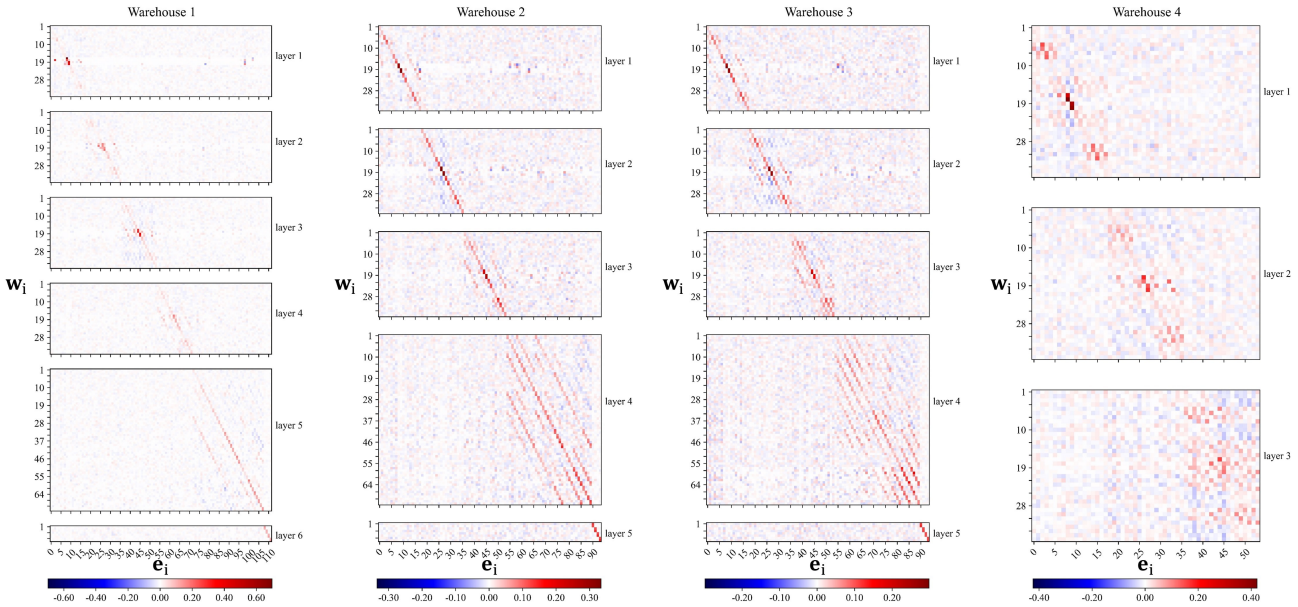


Figure 9: Visualization of statistical mean values of learnt attention α_{ij} in each warehouse for KernelWarehouse with different attentions initialization strategies. The results are obtained from the pre-trained ResNet18 backbone with KW ($4\times$) for all of the 50,000 images on the ImageNet validation set. Best viewed with zoom-in. The attentions initialization strategies for the groups of visualization results are as follows: (a) building one-to-one relationships between kernel cells and linear mixtures; (b) building four-to-one relationships between kernel cells and linear mixtures.



(a)



(b)

Figure 10: Visualization of statistical mean values of learnt attention α_{ij} in each warehouse for KernelWarehouse with different attentions initialization strategies. The results are obtained from the pre-trained ResNet18 backbone with KW $(1/2\times)$ for all of the 50,000 images on the ImageNet validation set. Best viewed with zoom-in. The attentions initialization strategies for the groups of visualization results are as follows: (a) building one-to-one relationships between kernel cells and linear mixtures; (b) building one-to-two relationships between kernel cells and linear mixtures.