RepNeXt: A Re-parameterizable ConvNet with High Performance and Efficiency

Anonymous CVPR submission

Paper ID 4947

Abstract

*As computer vision evolves, vision transformers (ViTs) and ConvNets continue to outperform each other in performance. However, there is always a speed-accuracy trade-off regarding practical applications. We propose a ConvNet called **RepNeXt** that takes full advantage of **structural re-parameterization** to guarantee superb performance and efficiency, making it less necessary to find trade-offs. Concretely, we choose structures that are proven to be high-performance, more efficient, and suitable for structural re-parameterization to frame our network. A multi-branch and multi-scale structure RepUnit is designed for feature extraction, improving the network's performance by extracting more texture bias. RepUnit can be re-parameterized into a single-branch structure to accelerate inference after training. Shortcut branches in the network are re-parameterized generically to achieve a faster inference speed by improving memory access. RepNeXt runs **5.8 folds faster** than the SOTA model ConvNeXt on average with the same level of accuracy on ImageNet-1K and shows robust adaptability to fine-grained and low-resolution tasks.*

1. Introduction

With the continuous development of computer vision research, vision transformers (ViTs) tend to show performance beyond ConvNets on various tasks, including image classification [11, 28, 45], object detection [2, 13], image generation [51], etc. By analyzing the reasons for the success of ViTs and borrowing some innovative ideas from them, ConvNets can in turn outperform ViTs [9, 29]. In the virtuous competition between ViTs and ConvNets, a set of design paradigms for high-performance computer vision models can be summarized: factors such as large receptive fields, layer normalization, patch embedding, and attention mechanisms are important in improving the performance of a model [1, 9, 44, 46].

However, there is always a speed-accuracy trade-off when it comes to practical applications. To apply models in scenarios requiring real-time inference, techniques such as lightweight model design [19, 32, 54], network pruning [27, 30], and network quantization [36, 48] have been widely investigated. Still, they all inevitably sacrifice a portion of the model's performance. Application scenarios exist where both accuracy and inference speed are required, such as autonomous driving [16], because incorrect recognition and untimely decisions may cause severe damage. Therefore it is imperative to design recognition models with high performance and efficiency.

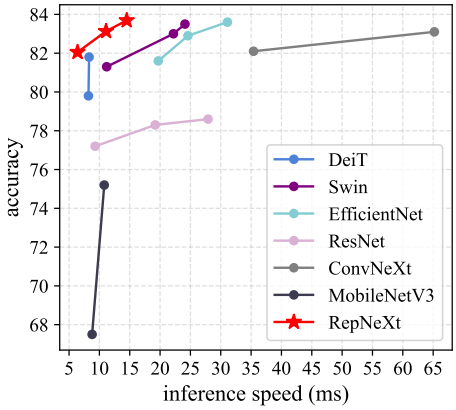

Figure 1. Top-1 accuracy on ImageNet-1K vs. inference speed. The speed is tested with an NVIDIA A100 GPU with a batch size of 1, an input resolution of 224^2, and full precision (fp32). **RepNeXt** performs well in both accuracy and inference speed.

ConvNet may be a better choice for fast inference because of the more mature hardware optimization. Nevertheless, challenges still exist to overcome to achieve high performance and efficiency simultaneously. Taking the SOTA model ConvNeXt as an example, in terms of performance, ConvNeXt uses large kernels to increase its receptive field, which makes it challenging to extract texture bias [18] and tough to achieve as good performance as on ImageNet [6] on fine-grained or low-resolution tasks (cf., Tables 2 and 3). In terms of efficiency, the inference of ConvNeXt is slow due to the use of some structures that sacrifice speed for higher accuracy. On the one hand, fragmented structures

[54], including shortcut branches and pointwise conv layers [22] implemented by fully connected layers and dimensional transformations, reduce the degree of parallelism of the model. On the other hand, inherently inefficient operations such as layer normalization and element-wise addition have a negative impact on inference speed.

Structural re-parameterization [8] decouples a model into a training-time structure and an inference-time structure, which are linearly equivalent in the inference time. The training-time structure is generally designed to be complex to obtain higher performance. After the training is completed, the model is re-parameterized into a relatively simple inference-time structure to avoid increasing its inference burden. Although structural re-parameterization is often used to improve the performance of networks, its potential for inference acceleration is not much discussed.

We consider implementing a ConvNet that combines high performance and efficiency by exploiting the full potential of structural re-parameterization. RepNeXt is thus proposed based on ConvNeXt. **To guarantee good performance**, A re-parameterizable structure RepUnit is designed as the main feature extractor, which consists of conv layers of different scales in parallel, closer to the concept of dynamic receptive field [12] and more capable of extracting texture bias. **To guarantee high efficiency**, RepUnit is re-parameterized into a single conv layer in the inference time to reduce its inference burden. More efficient components are used, such as BN layers and pointwise conv layers implemented by conv layers. Shortcut branches in the network are generically eliminated to improve memory access. In general, RepNeXt is a re-parameterizable network. **In the training time**, shortcut branches and multi-branch feature extraction structures are present to obtain higher performance and achieve adaptation to low-resolution and fine-grained tasks. **In the inference time**, branching structures are re-parameterized into plain structures to improve inference efficiency. As shown in Fig. 1, RepNeXt surpasses Swin Transformer [28] and obtains the same level of performance as ConvNeXt on ImageNet-1K. RepNeXt's inference is multiple folds faster than ConvNeXt, and the fastest is even in excess of MobileNetV3 [19]. Our contributions can be summarized as follows:

- We propose a re-parameterizable ConvNet named RepNeXt, which may be a riveting attempt to achieve both high performance and efficiency.
- We design a multi-scale re-parameterizable structure as the primary feature extractor. It is simple and easy to implement without excessively increasing the training cost and performs well on fine-grained and low-resolution tasks.
- We devise a general shortcut re-parameterization to eliminate the shortcut branches, which can bring a substantial acceleration to the inference of the network by providing valuable memory access efficiency.
- We use multiple experiments to demonstrate RepNeXt's good performance, efficiency, and adaptability for fine-grained and low-resolution tasks.

2. Related Work

2.1. ConvNet's high-performance design paradigm

Combining the advantages of ConvNets and ViTs is a high-performance model design trend. CoAtNet [5] replaced the shallow layers of ViTs with conv layers to extract universal features [49], incorporated certain translation invariance in the transformer structure of little prior knowledge [24], and showed excellent performance. BoTNet [40] used multi-head self-attention (MHSA) to replace the 3×3 conv layer in ResNet [17], which greatly improved its performance. **Mimicking the structure of ViTs** to design ConvNets can also achieve good performance. Inspired by ViTs' large receptive fields, RepLKNet [9] explored the use of very large kernels to enable ConvNets to outperform ViTs. To avoid the difficulty of extracting texture bias of large kernels, small kernels were trained in parallel with them, and the two were merged in the inference time using structural re-parameterization. ConvNeXt [29] achieved performance beyond Swin Transformer by adopting some design ideas from ViTs. The design paradigm for high-performance ConvNets is rethought and summarized as follows: the use of vit-style stem, larger kernels, layer normalization, depthwise conv [47], pointwise conv, wider conv layers [37], and shortcut branches.

2.2. Structural Re-parameterization

Structural re-parameterization has recently received much attention as a way to improve performance by decoupling a model into its training-time and inference-time structure. ACNet [7] and DBB [8] respectively used two re-parameterizable structures of different complexity to replace the $K \times K$ conv in the original network, which greatly improved the performance. RepVGG [10] used a re-parameterizable structure containing a shortcut branch with a BN layer, allowing the model to be trained like a ResNet and to infer like a VGG [39]. RMNet [33] innovatively eliminated the shortcut branches in ResNet, making ResNet a plain structure, which improved its memory access and made it more friendly for pruning.

We end this section by summarizing the **7 rudimentary paradigms [8] for ConvNet's structural re-parameterization**. **1)** A conv layer and its subsequent BN layer can be merged into a single conv layer. **2)** Multiple parallel conv layers for branch addition can be merged into a single conv layer by element-wise adding their parameters. **3)** Multiple parallel conv layers for depth concatenation can be merged into a single conv layer by concatenating their

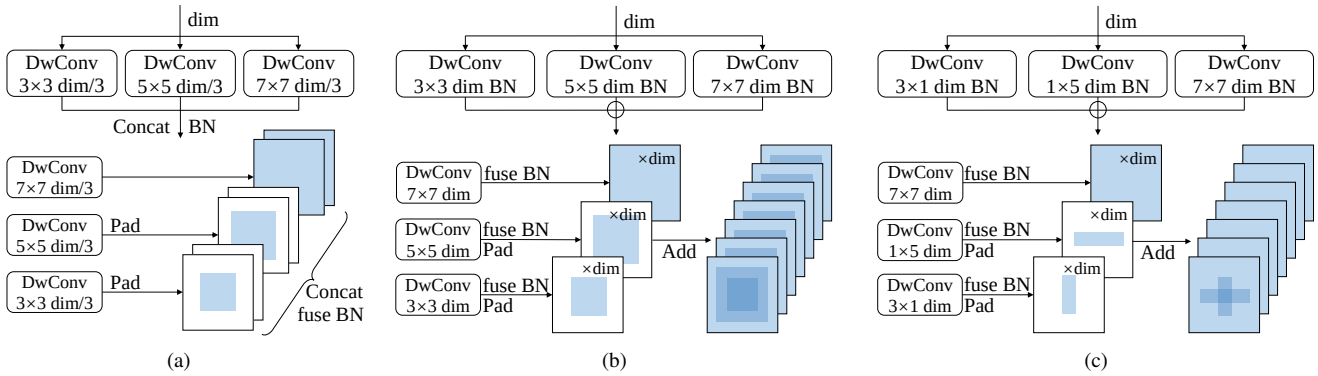

Figure 2. The **design route** and **re-parameterization process** for RepUnit. **RepUnit1 (a)** is implemented using three different sizes of parallel kernels to be concatenated. Different from RepUnit1, the implementation of **RepUnit2 (b)** uses parallel kernels to be element-wise added. **RepUnit3 (c)** is obtained by compressing some of the kernels of RepUnit2, which reduces the training cost.

parameters. **4)** The way for merging parallel conv layers of different sizes is the same as paradigms 2 and 3 in principle, but the only difference is that all kernels need to be padded with zero to the same size as the largest one. **5)** A pointwise conv layer and a conv layer in series can be merged into a single conv layer leveraging the associative axiom of convolutional operations. **6)** Shortcut branches can be eliminated at the cost of increasing the input or output dimension of the conv layers in the main branch. **7)** A pooling or BN layer can be converted into a conv layer to facilitate other structural re-parameterization operations.

3. RepNeXt

3.1. Block Design for Feature Extraction

Using larger kernels proved beneficial for the ImageNet classification task [9, 34]. However, experiments (cf., Tables 2 and 3) show that using purely large kernels is difficult to achieve satisfactory results as on ImageNet on fine-grained and low-resolution tasks. This phenomenon is because texture bias is more critical **for fine-grained tasks**, yet large kernels are more concerned with extracting shape bias [38]. **For low-resolution tasks**, the texture bias is inherently insufficient and degrades the recognition performance when ignored [14].

We consider using structural re-parameterization to extract both texture and shape bias in the training time without increasing the inference burden. Concretely, we use a re-parameterizable structure called RepUnit for feature extraction, whose inference-time structure is a depthwise conv layer of size 7×7. With the design principle of not increasing the training cost excessively and the coexistence of large and small kernels, the design route and re-parameterization process for RepUnit are shown in Fig. 2.

RepUnit1 (cf., Fig. 2a) uses three parallel depthwise conv layers of different sizes for feature extraction in the training time. The depthwise conv layer of each size receives one-third of the input feature map, and the outputs of all layers are concatenated as the output feature map. To describe the re-parameterization process, the three depthwise conv kernels are defined as $F_1 \in \mathbb{R}^{\frac{dim}{3} \times 1 \times 3 \times 3}$, $F_2 \in \mathbb{R}^{\frac{dim}{3} \times 1 \times 5 \times 5}$ and $F_3 \in \mathbb{R}^{\frac{dim}{3} \times 1 \times 7 \times 7}$. The input feature map is defined as $I \in \mathbb{R}^{dim \times H \times W}$. The output feature map is defined as $O \in \mathbb{R}^{dim \times H \times W}$. The running mean, running var, weight, and bias of the BN Layer are defined as μ, σ, γ, and β. In the training time, the forward propagation of RepUnit1 can be described as Eq. (1):

$$O = (\mathrm{cat}(F_1 * I[0 : \frac{dim}{3}], F_2 * I[\frac{dim}{3} : \frac{2 \cdot dim}{3}], F_2 * I[\frac{2 \cdot dim}{3} :]) - \mu)\frac{\gamma}{\sigma} + \beta \quad (1)$$

Where cat means the concatenation operation, $*$ means the convolutional operation, and [:] means the slice operation. According to the associative axiom of the convolutional operation, Eq. (1) and Eq. (2) are equivalents.

$$O = \frac{\gamma}{\sigma}\mathrm{cat}(\mathrm{pad}(F_1), \mathrm{pad}(F_2), F_3) * I - \frac{\gamma\mu}{\sigma} + \beta \quad (2)$$

Where pad means padding the kernel with zeros to 7×7 size. Thus RepUnit1 can be re-parameterized into a single conv layer, whose forward propagation can be described as $O = \hat{F} * I + \hat{B}$, where $\hat{F} = \frac{\gamma}{\sigma}\mathrm{cat}(\mathrm{pad}(F_1), \mathrm{pad}(F_2), F_3)$ and $\hat{B} = -\frac{\gamma\mu}{\sigma} + \beta$. The design of RepUnit1 is attractive because the presence of smaller kernels and the non-increased layer width decrease the training cost instead of increase.

Since there is no guarantee that every channel can be fed into the most appropriate branch, we design RepUnit2 (cf., Fig. 2b). Each branch accepts the complete input feature map. The final output feature map results from the element-wise addition of the output feature map through each branch

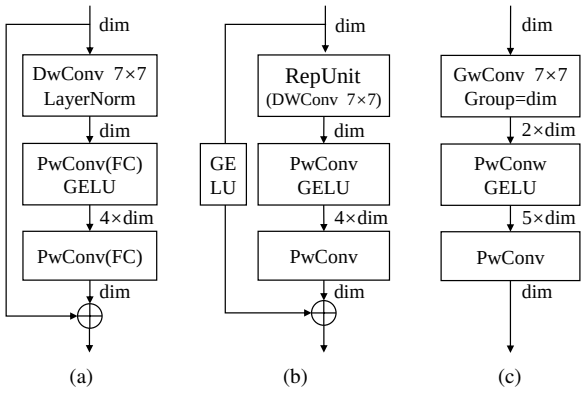

Figure 3. **Block designs** for a ConvNeXt **(a)**, a training-time RepNeXt **(b)**, and a inference-time RepNeXt **(c)**. Compared to ConvNeXt, RepNeXt uses RepUnit as the main feature extractor, reuses the BN layer for normalization, uses conv layers to implement pointwise conv layers, and adds an activation layer to the shortcut branch. After re-parameterization, RepNeXt will become a branchless plain architecture.

after batch normalization. Note that the scaling function of the normalization layer is essential for multi-branch structures, which plays a role in controlling the saliency of branches [21]. Let $F_1 \in \mathbb{R}^{dim\times1\times3\times3}$, $F_2 \in \mathbb{R}^{dim\times1\times5\times5}$, and $F_3 \in \mathbb{R}^{dim\times1\times7\times7}$ be the three parallel depthwise conv kernels. Let μ_i, σ_i, γ_i, and β_i be the parameters of the ith BN layer, the forward propagation of RepUnit2 in the training time can be described as Eq. (3).

$$O = \sum_{i=1}^{3}((F_i * I - \mu_i)\frac{\gamma_i}{\sigma_i} + \beta_i) \tag{3}$$

Equation (3) can also be equivalently written as Eq. (4) according to the associative axiom of the convolutional operation.

$$O = (\sum_{i=1}^{2}\mathrm{pad}(\frac{\gamma_i F_i}{\sigma_i}) + \frac{\gamma_3 F_3}{\sigma_3}) * I + \sum_{i=1}^{3}(\beta_i - \frac{\mu_i\gamma_i}{\sigma_i}) \tag{4}$$

Thus RepUnit2 can be re-parameterized as a single conv layer, whose forward propagation can be described as $O = \hat{F} * I + \hat{B}$, where $\hat{F} = \sum_{i=1}^{2}\mathrm{pad}(\frac{\gamma_i F_i}{\sigma_i}) + \frac{\gamma_3 F_3}{\sigma_3}$ and $\hat{B} = \sum_{i=1}^{3}(\beta_i - \frac{\mu_i\gamma_i}{\sigma_i})$. RepUnit2 has more parameters and a larger width, which will increase the training cost, although it does not affect the inference speed.

To reduce the training cost, we design RepUnit3 (cf., Fig. 2c), which compresses the depthwise conv kernel of size 3×3 to 3×1 and the depthwise conv kernel of size 5×5 to 1×5 based on RepUnit2, whose re-parameterization procedure is the same as RepUnit2.

To maintain good performance, when designing the RepNeXt block, we take adequate advantage of the high-performance design paradigm summarized by ConvNeXt (cf., Fig. 3a). They are the use of large kernels, the use of inverted bottleneck structures, fewer activation, fewer normalization, etc. However, structures such as LN layers and pointwise conv layers implemented by fully connected layers and dimensional transformations are unsuitable for re-parameterization and the subsequent acceleration, e.g., operator fusion [3]. Therefore, we re-enable BN layers and pointwise conv layers implemented by conv layers to form the RepNeXt block. Furthermore, an activation layer is added to the shortcut branch to facilitate the re-parameterization for shortcut branches (cf., Sec. 3.2). The training-time RepNeXt block (cf., Fig. 3b) is thus formed. After training, the network is re-parameterized to obtain the inference-time RepNeXt, whose block is shown in Fig. 3c.

3.2. Shortcut Re-parameterization

For many models, especially the popular multi-branch models, the number of parameters and the floating-point operations (FLOPs) may no longer describe the model's efficiency as reasonably and straightforwardly as the inference speed. This is because multiple branches increase the memory occupation [10]. A high memory occupation implies a low memory access efficiency, which slows down the model's inference and is an obstacle to applying the model to real-time inference scenarios. For example, the high memory occupation of ResNet is reflected in its shortcut branches because the shortcut branch will keep the input feature map in memory until the main branch is fully computed. Hence, shortcut re-parameterization is valuable.

Figure 4 illustrates the shortcut re-parameterization process of the RepNeXt block. The core tool for shortcut re-parameterization is dirac initialization, which aims to make the input and output feature map identical. The dirac initialization of a kernel $K \in \mathbb{R}^{O\times I\times H\times W}$ can be described as Eq. (5).

$$K_{o,i,h,w} = \begin{cases} 1 & \text{if } h = \frac{H}{2}, w = \frac{W}{2}, o = i \\ 0 & \text{elsewise} \end{cases} \tag{5}$$

Non-all-zero and full-zero weights after dirac initialization are marked with different fill textures in Fig. 4. The output dimension of all conv layers except the last one is increased, and the corresponding weights are dirac initialized. Thus, the original feature map can appear in the output feature map in a concatenated form. The input dimension of the last conv layer is increased, and the corresponding weights are also dirac initialized. Thus, the element-wise addition of the original feature map and the output feature map of the main branch can be equivalently converted into a convolutional operation. Some concrete instructions are as follows: **1)** Dirac-initialized kernels need to be inserted into the depthwise conv layer one after another to form a

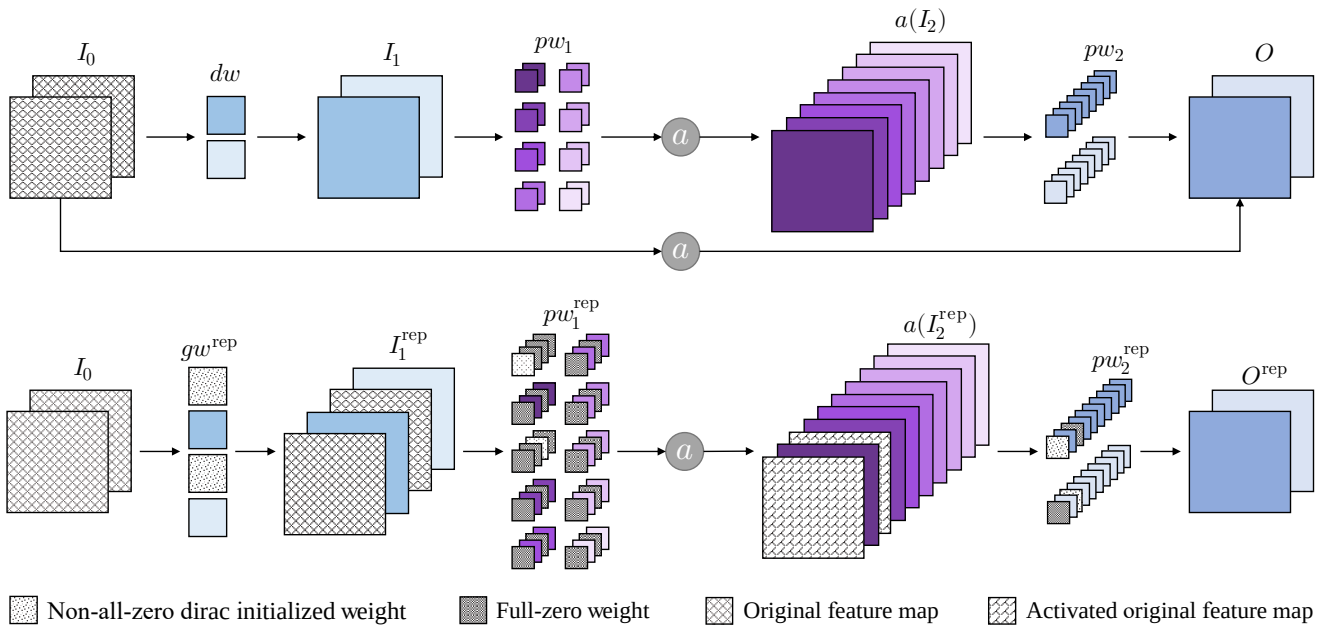

Figure 4. **Shortcut re-parameterization** for RepNeXt. The upper and lower parts represent the RepNeXt block before and after the shortcut re-parameterization. Equivalent elimination of the shortcut branch is achieved by dirac initialization as the main method with the prerequisite that the shortcut branch contains the same activation layer as the main branch.

groupwise conv layer [53] (group=dim). This is because the groupwise conv always uses adjacent kernels as a group. **2)** To accommodate instruction 1, dirac-initialized kernels also need to be inserted into the first pointwise conv layer one after another. Some additional all-zero weights need to be introduced in the input dimension, as pw_1^{rep} in Fig. 4 demonstrates. **3)** To adapt shortcut re-parameterization to various activation layers, instead of relying on the prerequisite that the feature map remains non-negative after relu activation [15, 33], we add an activation layer on the shortcut branch that is identical to the main branch.

This way, a re-parameterized plain architecture can replace the training-time architecture without changing its input and output. Since eliminating multiple branches improves memory access, the inference speed of the model can be greatly improved.

O and O^{rep} can be proved to be equivalent as follows:

According to Fig. 4, it is easy to define the kernels associated with structural re-parameterization as $pw_2^{\text{rep}} = [pw_2^{\text{C}\times 4\text{C}}, E^{\text{C}\times\text{C}}]^{\text{C}\times 5\text{C}}$, $B(pw_2^{\text{rep}}) = B^{\text{C}}(pw_2)$, $pw_1^{\text{rep}} = [[pw_1^{4\text{C}\times\text{C}}, 0^{\text{C}\times\text{C}}]^{5\text{C}\times\text{C}}, [0^{4\text{C}\times\text{C}}, E^{\text{C}\times\text{C}}]^{5\text{C}\times\text{C}}]^{5\text{C}\times 2\text{C}}$, $B(pw_1^{\text{rep}}) = [B^{4\text{C}}(pw_1), 0^{\text{C}}]^{5\text{C}}$, $gw^{\text{rep}} = [dw^{\text{C}\times 1}, E^{\text{C}\times 1}]^{2\text{C}\times 1}$, $B(gw^{\text{rep}}) = [B^{\text{C}}(dw), 0^{\text{C}}]^{2\text{C}}$, where $[\cdot,\cdot]$ means the depth concatenation of the two elements. For ease of understanding, the dimensions of noticeable elements have been marked in their upper right.

Let us calculate the intermediate results in the order of forward propagation. I_1^{rep} can be derived as Eq. (6):

$$\begin{aligned} I_1^{\text{rep}} &= gw^{\text{rep}} * I_0^{\text{C}} + B(gw^{\text{rep}}) \\ &= [dw^{\text{C}\times 1}, E^{\text{C}\times 1}]^{2\text{C}\times 1} * I_0^{\text{C}} + [B^{\text{C}}(dw), 0^{\text{C}}]^{2\text{C}} \\ &= [dw^{\text{C}\times 1} * I_0^{\text{C}} + B^{C}(dw), E^{\text{C}\times 1} * I_0^{\text{C}} + 0^{\text{C}}] \\ &= [I_1^{\text{C}}, I_0^{\text{C}}]^{2\text{C}} \end{aligned} \tag{6}$$

Using the results of I_1^{rep}, $\text{a}(I_2^{\text{rep}})$ can be calculated as Eq. (7), where a can be any kind of activation.

$$\begin{aligned} \text{a}(I_2^{\text{rep}}) &= \text{a}(pw_1^{\text{rep}} * I_1^{\text{rep}} + B(pw_1^{\text{rep}})) \\ &= \text{a}([[pw_1^{4\text{C}\times\text{C}}, 0^{\text{C}\times\text{C}}]^{5\text{C}\times\text{C}}, [0^{4\text{C}\times\text{C}}, E^{\text{C}\times\text{C}}]^{5\text{C}\times\text{C}}]^{5\text{C}\times 2\text{C}} * [I_1^{\text{C}}, I_0^{\text{C}}]^{2\text{C}} + [B^{4\text{C}}(pw_1), 0^{\text{C}}]^{5\text{C}}) \\ &= \text{a}([pw_1^{4\text{C}\times\text{C}} * I_1^{\text{C}}, 0^{\text{C}}]^{5\text{C}} + [0^{4\text{C}}, I_0^{\text{C}}]^{5\text{C}} + [B^{4\text{C}}(pw_1), 0^{\text{C}}]^{5\text{C}}) \\ &= \text{a}([pw_1^{4\text{C}\times\text{C}} * I_1^{\text{C}} + 0^{4\text{C}} + B^{4\text{C}}(pw_1), 0^{\text{C}} + I_0^{\text{C}} + 0^{\text{C}}]) \\ &= [\text{a}^{4\text{C}}(I_2), \text{a}^{\text{C}}(I_0)]^{5\text{C}} \end{aligned} \tag{7}$$

Structure	Output Size	Training-time RepNeXt	Inference-time RepNeXt
Stem	56×56	$\left[\begin{array}{c} 4 \times 4, s = 4, 96 \\ BN \end{array}\right]$	$4 \times 4, s = 4, 96$
Stage1	56×56	$\left[\begin{array}{c} \text{RepUnit}, dw, 96 \\ 1 \times 1, 384 \\ 1 \times 1, 96 \end{array}\right] \times 3$	$\left[\begin{array}{c} 7 \times 7, 192, \text{ group } = 96 \\ 1 \times 1, 480 \\ 1 \times 1, 96 \end{array}\right] \times 3$
Stage2	28×28	$\left[\begin{array}{c} \text{RepUnit}, dw, 192 \\ 1 \times 1, 768 \\ 1 \times 1, 192 \end{array}\right] \times 3$	$\left[\begin{array}{c} 7 \times 7, 384, \text{ group } = 192 \\ 1 \times 1, 960 \\ 1 \times 1, 192 \end{array}\right] \times 3$
Stage3	14×14	$\left[\begin{array}{c} \text{RepUnit}, dw, 384 \\ 1 \times 1, 1536 \\ 1 \times 1, 384 \end{array}\right] \times 9$	$\left[\begin{array}{c} 7 \times 7, 768, \text{ group } = 384 \\ 1 \times 1, 1920 \\ 1 \times 1, 384 \end{array}\right] \times 9$
Stage4	7×7	$\left[\begin{array}{c} \text{RepUnit}, dw, 768 \\ 1 \times 1, 3072 \\ 1 \times 1, 768 \end{array}\right] \times 3$	$\left[\begin{array}{c} 7 \times 7, 1536, \text{ group } = 768 \\ 1 \times 1, 3840 \\ 1 \times 1, 768 \end{array}\right] \times 3$

Table 1. **Detailed architecture specifications** for the training-time RepNeXt-T and the inference-time RepNeXt-T. The resolution of the image input to the model is 224^2.

Finally, we use the result of $\mathrm{a}(I_2^{\mathrm{rep}})$ to calculate O^{rep}, and you will find that it is equivalent to the original output O. As shown in Eq. (8).

$$\begin{aligned} O^{\mathrm{rep}} &= pw_2^{\mathrm{rep}} * \mathrm{a}(I_2^{\mathrm{rep}}) + B(pw_2^{\mathrm{rep}}) \\ &= [pw_2^{\mathrm{C}\times 4\mathrm{C}}, E^{\mathrm{C}\times\mathrm{C}}]^{\mathrm{C}\times 5\mathrm{C}} * [\mathrm{a}^{4\mathrm{C}}(I_2), \mathrm{a}^{\mathrm{C}}(I_0)]^{5\mathrm{C}} \\ &\quad + B^{\mathrm{C}}(pw_2) \\ &= pw_2^{\mathrm{C}\times 4\mathrm{C}} * \mathrm{a}^{4\mathrm{C}}(I_2) + B^{\mathrm{C}}(pw_2) + \mathrm{a}^{\mathrm{C}}(I_0) \\ &= O \end{aligned} \tag{8}$$

3.3. Architecture Specification

We define the architecture of RepNeXt in Table 1.

Stem denotes the previous few layers of the network. Following the concept of patch embedding, our stem consists of a conv layer of size and stride equal to 4 and a BN layer. The reason for using batch normalization instead of layer normalization is that it can be merged with the conv layer (Paradigm 1 in Sec. 2.2) to accelerate inference.

stage1 to stage4 each contains several RepNeXt blocks. In the training time, each block contains a RepUnit (cf., Sec. 3.1) for feature extraction, two pointwise conv layers for implementing the inverted bottleneck structure, and a shortcut branch with an activation layer (not shown in Table 1). After training and structural re-parameterization, an inference-time block is obtained, which contains a conv layer of 7×7 size with doubled width, two pointwise conv layers with different input or output dimensions from the training-time block, and no more multiple branches.

Downsampling layers (not shown in Table 1) are located between stages and consist of a BN layer and a conv layer of size and stride equal to 2. Downsampling layers are responsible for downsampling the input feature map to half its size and doubling the number of output channels. The case where the BN layer is in front of the conv layer can also be re-parameterized. The BN layer is equivalently converted into a pointwise conv layer (Paradigm 7 in Sec. 2.2) and fused with the conv layer (Paradigm 5 in Sec. 2.2).

There are three architectural hyper-parameters for RepNeXt: the RepUnit chosen, the number of RepNeXt Blocks B, and the channel dimension C. So that a RepNeXt architecture is mainly defined by $[B_1, B_2, B_3, B_4]$ and $[C_1, C_2, C_3, C_4]$. Following the design of Swin Transformers and ConvNeXt for models of different volumes, we summarize the training-time configurations of RepNeXt below (we do not design the mega-scale models like ConvNeXt-L because the structural re-parameterization and acceleration for them are relatively meaningless):

- RepNeXt-T: $C = [96, 192, 384, 768], B = [3, 3, 9, 3]$
- RepNeXt-S: $C = [96, 192, 384, 768], B = [3, 3, 27, 3]$
- RepNeXt-B: $C = [128, 256, 512, 1024], B = [3, 3, 27, 3]$

Which RepUnit is used in the network will be reflected in its name, e.g., RepNeXt-u2-S.

4. Experiments

4.1. Image Classification

Image classification is commonly used to measure the performance of a model. For comparison, we need to train our models until convergence. Concretely, the models are trained using the AdamW optimizer [31]. The learning rate varies according to the cosine decaying schedule with a 20-epoch linear warmup. Typical training schemes, including data augmentation [4,50,52,55], regularization [23,41], and exponential moving average (EMA) [35] are used.

For fine-grained and low-resolution image classification, we compare the three versions (different RepUnit) of RepNeXt with the SOTA models ResNet [17] and ConvNeXt [29] on ImageWoof [20] and Cifar100 [25]. Imagewoof is a

model	#param. (M)	FLOPs (B)	speed (ms)	top-1 acc.
ResNet50	26	4.1	9.3	87.7
ConvNeXt-T	29	4.5	35.4	88.0
RepNeXt-u1-T	29$\xrightarrow{r}$52	4.4$\xrightarrow{r}$8.2	10.4$\xrightarrow{r}$**6.4**	90.5
RepNeXt-u2-T	29$\xrightarrow{r}$52	4.6$\xrightarrow{r}$8.2	12.2$\xrightarrow{r}$**6.4**	90.7
RepNeXt-u3-T	29$\xrightarrow{r}$52	4.5$\xrightarrow{r}$8.2	11.9$\xrightarrow{r}$**6.4**	**90.8**

Table 2. **Results of ImageWoof classification**. The models are trained for 300 epochs with a batch size of 512, a learning rate of 5e-4, and an input resolution of 224^2. The speed is tested on an NVIDIA A100 GPU with a batch size of 1, full precision (fp32).

model	#param. (M)	FLOPs ($\times$ 10M)	speed (ms)	top-1 acc.
ResNet50	26	34	10.6	77.2
ConvNeXt-T	29	37	36.0	78.6
RepNeXt-u1-T	29$\xrightarrow{r}$52	36$\xrightarrow{r}$67	10.8$\xrightarrow{r}$**6.3**	79.7
RepNeXt-u2-T	29$\xrightarrow{r}$52	37$\xrightarrow{r}$67	12.8$\xrightarrow{r}$**6.3**	79.4
RepNeXt-u3-T	29$\xrightarrow{r}$52	37$\xrightarrow{r}$67	12.6$\xrightarrow{r}$**6.3**	**79.8**

Table 3. **Results of Cifar100 classification**. The models are trained for 600 epochs with a batch size of 512, a learning rate of 5e-4, and an input resolution of 64^2. The speed is tested on an NVIDIA A100 GPU with a batch size of 1, full precision (fp32).

fine-grained subset of Imagenet [6], which contains 10 dog classes and comprises 9.0K images for training and 3.9K for validation. Cifar100 is both a low-resolution and fine-grained dataset with 100 classes, comprising 50K images for training and 10K for validation.

Tables 2 and 3 show the experimental results for fine-grained and low-resolution image classification. The performance of RepNeXt is higher than that of ResNet50 and ConvNeXt on the two datasets, and the advantage of the coexistence of large and small kernels is reflected. The fluctuations in the number of parameters, FLOPs, and performance of the three versions of RepNeXt are relatively slight, and in a comprehensive comparison, RepNeXt-u3 may be the best. After structural re-parameterization (noted as $\xrightarrow{r}$ in the tables), the number of parameters and FLOPs of RepNeXt will increase by at most 86%, and its inference speed will increase by at least 37%. Note that structural re-parameterization only statistically increases the number of parameters and FLOPs of the network. Since the added parameters are very sparse (lots of zeros and a few ones), most of the additional computation is arithmetic with zeros, which can not seriously slow down the inference. The great speed improvement comes mainly from the structural re-parameterization of RepUnit and shortcut branches.

We further compare RepNeXt with the SOTA models, including ResNet [17], MobileNetV3 [19], EfficientNet [42], DeiT [43], Swin Transformer [28], and ConvNeXt [29] on ImageNet-1K [6], which consists of 1000 categories with

model	#param. (M)	FLOPs (B)	speed (ms)	top-1 acc.
ResNet50	26	4.1	9.3	77.2
ResNet101	45	7.9	19.2	78.3
ResNet152	60	11.6	27.9	78.6
MobileNetV3-S	2.5	0.06	8.8	67.5
MobileNetV3-L	5.4	0.2	10.8	75.2
EffNet-B3	12	1.0	19.7	81.6
EffNet-B4	19	1.5	24.6	82.9
EffNet-B5	30	2.4	31.1	83.6
DeiT-S	22	4.6	8.2	79.8
DeiT-B	87	17.6	8.3	81.8
Swin-T	28	4.5	11.2	81.3
ConvNeXt-T	29	4.5	35.4	**82.1**
RepNeXt-u3-T	29$\xrightarrow{r}$52	4.5$\xrightarrow{r}$8.2	11.9$\xrightarrow{r}$**6.4**	**82.1**
Swin-S	50	8.7	22.2	83.0
ConvNeXt-S	50	8.7	65.2	**83.1**
RepNeXt-u3-S	50$\xrightarrow{r}$92	8.8$\xrightarrow{r}$16.1	22.6$\xrightarrow{r}$**11.1**	**83.1**
Swin-B	88	15.4	24.1	83.5
ConvNeXt-B	89	15.4	87.2	**83.8**
RepNeXt-u3-B	89$\xrightarrow{r}$163	15.5$\xrightarrow{r}$28.5	22.4$\xrightarrow{r}$**14.5**	83.7

Table 4. **Results of ImageNet-1K classification**. The models are trained for 450 epochs with a batch size of 1024, a learning rate of 1e-3, and an input resolution of 224^2. The speed is tested on an NVIDIA A100 GPU with a batch size of 1, full precision (fp32).

1.28M training images and 50K validation images.

Table 4 shows the experimental results for ImageNet-1K classification. In terms of performance, RepNeXt-u3 surpasses Swin Transformer and achieves the same level of performance as ConvNeXt. However, as the volume of RepNeXt increases, the potential for performance improvement seems to be limited, resulting in RepNeXt-u3-B's failure to reach the same level as ConvNeXt-B in performance, see Sec. 4.2 for analysis. In terms of efficiency, RepNeXt achieves the fastest inference speed, which exceeds ConvNeXt almost 6 fold, and noticeably surpasses the lightweight model MobileNetV3. On the one hand, the elimination of multiple branches brings acceleration. On the other hand, less fragmented structure and reduced time-consuming components also have a valid effect on speedups, as shown in Fig. 5. Considering both accuracy and speed, the superiority of RepNeXt is obvious.

4.2. Ablation study

To compare the impact of each component on performance and efficiency, we modify ConvNeXt into RepNeXt step by step. The accuracy and inference speed of each intermediate network are recorded, as shown in Table 5.

According to Table 5, conclusions can be drawn: **1)** A captivating observation is that using fully connected layers is more capable of achieving high performance than using conv layers to implement pointwise conv layers. How-

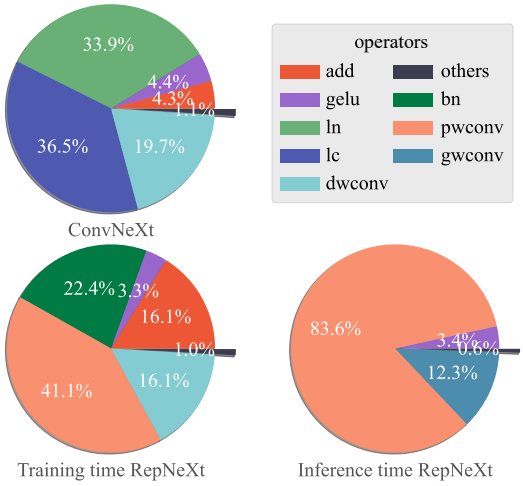

Figure 5. **Inference time decomposition of ConvNeXt, training-time and inference-time RepNeXt.** The inference of ConvNeXt is time-consuming. LN and fully connected layers that rely on dimensional transformations take up much inference time. The former is inherently time-consuming, and the latter is overly fragmented. RepNeXt has been accelerated during training, thanks to eliminating tedious dimensional transformations and using more efficient BN layers for normalization. After re-parameterization, the inference is further accelerated with the disappearance of multiple branches and BN layers.

model	rep.	speed (ms)	Top-1 acc.
ConvNeXt-T	✘	35.4 ± 6.1	**82.1**
Lc2pwconv-T	✘	11.5 ± 2.8	81.9
Ln2bn-T	✘	8.4 ± 1.9	81.8
Shortcut-act-T	✔	$8.5 \pm 2.1 \xrightarrow{r} 6.4 \pm 1.6$	81.6
RepNeXt-u3-T	✔	$11.9 \pm 3.0 \xrightarrow{r} 6.4 \pm 1.6$	**82.1**
shortcut-act-S	✔	$15.3 \pm 2.5 \xrightarrow{r} 11.1 \pm 2.0$	82.6
RepNeXt-u3-S	✔	$22.6 \pm 3.7 \xrightarrow{r} 11.1 \pm 2.0$	**83.1**
shortcut-act-B	✔	$16.0 \pm 3.1 \xrightarrow{r} 14.5 \pm 2.2$	82.9
RepNeXt-u3-B	✔	$22.4 \pm 3.5 \xrightarrow{r} 14.5 \pm 2.2$	**83.7**

Table 5. **Results of ablation study**. Lc2pwconv is the network that uses conv layers to implement pointwise conv layers based on ConvNeXt. Ln2bn is the network after replacing the LN layer with the BN layer. Shortcut-act is the network with activation layers added to the shortcut branch. Each of the above operations is done based on the previous operation. Only networks that could be shortcut re-parameterized (rep.) were tested for speed after re-parameterization.

ever, this kind of implementation dramatically slows down the inference speed of the network. This phenomenon might be influenced by the frequent dimensional transformations incidental to the fully connected layer. Fortunately, a simple paradigm of structural re-parameterization may be practical, which is to use fully connected layers to implement pointwise conv layers in the training time and re-parameterize them into conv layers in the inference time. **2)** The BN layer is still a functional structure for normalization. Although it is slightly less capable of improving the accuracy of the network than the LN layer because of its local normalization [1], it is efficient and can be merged with its adjacent conv layer. **3)** When activation layers are added to the shortcut branch, the accuracy of the network decreases. This is mainly because the number of activations in the network is boosted from $[0, \mathrm{N}]$ to $[\mathrm{N}, 2\mathrm{N}]$, where N denotes the number of activations in the original network. This breaks the high-performance ConvNet design paradigm summarized by ConvNeXt: fewer activations. Not adding activation layers, instead using selective activation after re-parameterization, is a way to maintain accuracy, but it still introduces additional complexity. **4)** The comparison between the Shortcut-act network and RepNeXt reflects the role of RepUnit, which can improve the accuracy to a higher level with a stronger feature extraction capability. It also confirms that texture bias is still critical to ImageNet classification [26]. **5)** The improvements to ConvNeXt and structural re-parameterization facilitate a more stable inference speed (smaller variance) since a cleaner network structure means less external interference.

5. Limitations

Shortcut re-parameterization introduces additional parameters and FLOPs to the network but does not affect the inference speed because of their sparsity. However, it brings extra space occupation. This makes RepNeXt probably more suitable for inference servers or terminal GPUs. It may not be as popular as lightweight models [19, 32] on devices such as microchips and cell phones.

6. Conclusion

We propose RepNeXt. The potential of structural re-parameterization is fully exploited to obtain superb performance and efficiency. In terms of performance, RepNeXt surpasses Swin Transformer and achieves the same level of accuracy as ConvNeXt on ImageNet-1K. RepNeXt also shows strong adaptability to fine-grained and low-resolution tasks. In terms of efficiency, the inference speed of RepNeXt is noticeably faster than other SOTA models.

References

[1] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv:1607.06450*, 2016. 1, 8

[2] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *ECCV*, 2020. 1

[3] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Haichen Shen, Meghan Cowan, Leyuan Wang, Yuwei Hu, Luis Ceze, et al. Tvm: An automated end-to-end optimizing compiler for deep learning. In *13th USENIX*

Symposium on Operating Systems Design and Implementation (OSDI 18), 2018. 4

[4] ED Cubuk, B Zoph, J Shlens, and Q Le Randaugment. Practical automated data augmentation with a reduced search space. In *CVPRW*, 2020. 6

[5] Zihang Dai, Hanxiao Liu, Quoc V Le, and Mingxing Tan. Coatnet: Marrying convolution and attention for all data sizes. *NeurIPS*, 2021. 2

[6] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009. 1, 7

[7] Xiaohan Ding, Yuchen Guo, Guiguang Ding, and Jungong Han. Acnet: Strengthening the kernel skeletons for powerful cnn via asymmetric convolution blocks. In *ICCV*, 2019. 2

[8] Xiaohan Ding, Xiangyu Zhang, Jungong Han, and Guiguang Ding. Diverse branch block: Building a convolution as an inception-like unit. In *CVPR*, 2021. 2

[9] Xiaohan Ding, Xiangyu Zhang, Jungong Han, and Guiguang Ding. Scaling up your kernels to 31x31: Revisiting large kernel design in cnns. In *CVPR*, 2022. 1, 2, 3

[10] Xiaohan Ding, Xiangyu Zhang, Ningning Ma, Jungong Han, Guiguang Ding, and Jian Sun. Repvgg: Making vgg-style convnets great again. In *CVPR*, 2021. 2, 4

[11] Xiaoyi Dong, Jianmin Bao, Dongdong Chen, Weiming Zhang, Nenghai Yu, Lu Yuan, Dong Chen, and Baining Guo. Cswin transformer: A general vision transformer backbone with cross-shaped windows. In *CVPR*, 2022. 1

[12] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *ICLR*, 2021. 2

[13] Yuxin Fang, Bencheng Liao, Xinggang Wang, Jiemin Fang, Jiyang Qi, Rui Wu, Jianwei Niu, and Wenyu Liu. You only look at one sequence: Rethinking transformer in vision through object detection. *NeurIPS*, 2021. 1

[14] Yunhao Ge, Yao Xiao, Zhi Xu, Xingrui Wang, and Laurent Itti. Contributions of shape, texture, and color in visual recognition. In *ECCV*, 2022. 3

[15] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, 2011. 5

[16] Sorin Grigorescu, Bogdan Trasnea, Tiberiu Cocias, and Gigel Macesanu. A survey of deep learning techniques for autonomous driving. *Journal of Field Robotics*, 2020. 1

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 2, 6, 7

[18] Katherine Hermann, Ting Chen, and Simon Kornblith. The origins and prevalence of texture bias in convolutional neural networks. *NeurIPS*, 2020. 1

[19] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. In *ICCV*, 2019. 1, 2, 7, 8

[20] Jeremy Howard. Imagewoof: a subset of 10 classes from imagenet that aren't so easy to classify, March 2019. 6

[21] Mu Hu, Junyi Feng, Jiashen Hua, Baisheng Lai, Jianqiang Huang, Xiaojin Gong, and Xian-Sheng Hua. Online convolutional re-parameterization. In *CVPR*, 2022. 4

[22] Binh-Son Hua, Minh-Khoi Tran, and Sai-Kit Yeung. Pointwise convolutional neural networks. In *CVPR*, 2018. 2

[23] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q Weinberger. Deep networks with stochastic depth. In *ECCV*, 2016. 6

[24] Salman Khan, Muzammal Naseer, Munawar Hayat, Syed Waqas Zamir, Fahad Shahbaz Khan, and Mubarak Shah. Transformers in vision: A survey. *ACM Computing Surveys (CSUR)*, 2021. 2

[25] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009. 6

[26] Yingwei Li, Qihang Yu, Mingxing Tan, Jieru Mei, Peng Tang, Wei Shen, Alan Yuille, et al. Shape-texture debiased neural network training. In *ICLR*, 2020. 8

[27] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *ICCV*, 2017. 1

[28] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *CVPR*, 2021. 1, 2, 7

[29] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s. In *CVPR*, 2022. 1, 2, 6, 7

[30] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. In *ICLR*, 2018. 1

[31] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *ICLR*, 2018. 6

[32] Sachin Mehta and Mohammad Rastegari. Mobilevit: Lightweight, general-purpose, and mobile-friendly vision transformer. In *ICLR*, 2021. 1, 8

[33] Fanxu Meng, Hao Cheng, Jiaxin Zhuang, Ke Li, and Xing Sun. Rmnet: Equivalently removing residual connection from networks. *arXiv:2111.00687*, 2021. 2, 5

[34] Chao Peng, Xiangyu Zhang, Gang Yu, Guiming Luo, and Jian Sun. Large kernel matters–improve semantic segmentation by global convolutional network. In *CVPR*, 2017. 3

[35] Boris T Polyak and Anatoli B Juditsky. Acceleration of stochastic approximation by averaging. *SIAM journal on control and optimization*, 1992. 6

[36] Haotong Qin, Ruihao Gong, Xianglong Liu, Xiao Bai, Jingkuan Song, and Nicu Sebe. Binary neural networks: A survey. *PR*, 105:107281, 2020. 1

[37] Ilija Radosavovic, Raj Prateek Kosaraju, Ross Girshick, Kaiming He, and Piotr Dollár. Designing network design spaces. In *CVPR*, 2020. 2

[38] Samuel Ritter, David GT Barrett, Adam Santoro, and Matt M Botvinick. Cognitive psychology for deep neural networks: A shape bias case study. In *International conference on machine learning*, 2017. 3

[39] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv:1409.1556*, 2014. 2

[40] Aravind Srinivas, Tsung-Yi Lin, Niki Parmar, Jonathon Shlens, Pieter Abbeel, and Ashish Vaswani. Bottleneck transformers for visual recognition. In *CVPR*, 2021. 2

[41] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *CVPR*, 2016. 6

[42] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, 2019. 7

[43] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. In *International Conference on Machine Learning*, 2021. 7

[44] Asher Trockman and J Zico Kolter. Patches are all you need? *arXiv:2201.09792*, 2022. 1

[45] Zhengzhong Tu, Hossein Talebi, Han Zhang, Feng Yang, Peyman Milanfar, Alan Bovik, and Yinxiao Li. Maxvit: Multi-axis vision transformer. 2022. 1

[46] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, 2017. 1

[47] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *CVPR*, 2017. 2

[48] Zhewei Yao, Zhen Dong, Zhangcheng Zheng, Amir Gholami, Jiali Yu, Eric Tan, Leyuan Wang, Qijing Huang, Yida Wang, Michael Mahoney, et al. Hawq-v3: Dyadic neural network quantization. In *International Conference on Machine Learning*, 2021. 1

[49] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? *NeurIPS*, 2014. 2

[50] Sangdoo Yun, Dongyoon Han, Seong Joon Oh, Sanghyuk Chun, Junsuk Choe, and Youngjoon Yoo. Cutmix: Regularization strategy to train strong classifiers with localizable features. In *ICCV*, 2019. 6

[51] Bowen Zhang, Shuyang Gu, Bo Zhang, Jianmin Bao, Dong Chen, Fang Wen, Yong Wang, and Baining Guo. Styleswin: Transformer-based gan for high-resolution image generation. In *CVPR*, 2022. 1

[52] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. In *ICLR*, 2018. 6

[53] Ting Zhang, Guo-Jun Qi, Bin Xiao, and Jingdong Wang. Interleaved group convolutions. In *ICCV*, 2017. 5

[54] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *CVPR*, 2018. 1, 2

[55] Zhun Zhong, Liang Zheng, Guoliang Kang, Shaozi Li, and Yi Yang. Random erasing data augmentation. In *AAAI*, 2020. 6