Scaling Up Your Kernels to 31x31: Revisiting Large Kernel Design in CNNs

Xiaohan Ding 1*  Xiangyu Zhang 2†  Jungong Han 3  Guiguang Ding 1‡
1 Beijing National Research Center for Information Science and Technology (BNRist); School of Software, Tsinghua University, Beijing, China
2 MEGVII Technology
3 Computer Science Department, Aberystwyth University, SY23 3FL, UK
dxh17@mails.tsinghua.edu.cn  zhangxiangyu@megvii.com
jungonghan77@gmail.com  dinggg@tsinghua.edu.cn

Abstract

We revisit large kernel design in modern convolutional neural networks (CNNs). Inspired by recent advances in vision transformers (ViTs), in this paper, we demonstrate that using a few large convolutional kernels instead of a stack of small kernels could be a more powerful paradigm. We suggested five guidelines, e.g., applying re-parameterized large depth-wise convolutions, to design efficient high-performance large-kernel CNNs. Following the guidelines, we propose RepLKNet, a pure CNN architecture whose kernel size is as large as 31×31, in contrast to commonly used 3×3. RepLKNet greatly closes the performance gap between CNNs and ViTs, e.g., achieving comparable or superior results than Swin Transformer on ImageNet and a few typical downstream tasks, with lower latency. RepLKNet also shows nice scalability to big data and large models, obtaining 87.8% top-1 accuracy on ImageNet and 56.0% mIoU on ADE20K, which is very competitive among the state-of-the-arts with similar model sizes. Code & models at https://github.com/megvii-research/RepLKNet.

1. Introduction

Convolutional neural networks (CNNs) [40, 53] used to be a common choice of visual encoders in modern computer vision systems. However, recently, CNNs [40, 53] have been greatly challenged by Vision Transformers (ViTs) [34, 59, 86, 94], which have shown leading performances on many visual tasks – not only image classification [34, 104] and representation learning [4, 9, 16, 100], but also many downstream tasks such as object detection [24, 59], semantic segmentation [94, 98] and image restoration [10, 54]. Why are ViTs super powerful? Some works believed that multi-head self-attention (MHSA) mechanism in ViTs plays a key role. They provided empirical results to demonstrate that, MHSA is more flexible [50], capable (less inductive bias) [20], more robust to distortions [66, 98], or able to model long-range dependencies [69, 90]. But some works challenge the necessity of MHSA [115], attributing the high performance of ViTs to the proper building blocks [33], and/or dynamic sparse weights [38, 111]. More works [20, 38, 42, 95, 115] explained the superiority of ViTs from different point of views.

In this work, we focus on one view: the way of building up large receptive fields. In ViTs, MHSA is usually designed to be either global [34, 78, 94] or local but with large kernels [59, 70, 89], thus each output from a single MHSA layer is able to gather information from a large region. However, large kernels are not popularly employed in CNNs (except for the first layer [40]). Instead, a typical fashion is to use a stack of many small spatial con-
volutions\textsuperscript{1} [40, 44, 47, 68, 77, 82, 109] (e.g., 3×3) to enlarge the receptive fields in state-of-the-art CNNs. Only some old-fashioned networks such as AlexNet [53], Inceptions [79–81] and a few architectures derived from neural architecture search [37, 43, 56, 116] adopt large spatial convolutions (whose size is greater than 5) as the main part. The above view naturally lead to a question: what if we use a few large instead of many small kernels to conventional CNNs? Is large kernel or the way of building large receptive fields the key to close the performance gap between CNNs and ViTs?

To answer this question, we systematically explore the large kernel design of CNNs. We follow a very simple “philosophy”: just introducing large depth-wise convolutions into conventional networks, whose sizes range from 3×3 to 31×31, although there exist other alternatives to introduce large receptive fields via a single or a few layers, e.g. feature pyramids [95], dilated convolutions [13, 101, 102] and deformable convolutions [23]. Through a series of experiments, we summarize five empirical guidelines to effectively employ large convolutions: 1) very large kernels can still be efficient in practice; 2) identity shortcut is vital especially for networks with very large kernels; 3) re-parameterizing [30] with small kernels helps to make up the optimization issue; 4) large convolutions boost downstream tasks much more than ImageNet; 5) large kernel is useful even on small feature maps.

Based on the above guidelines, we propose a new architecture named RepLKNet, a pure\textsuperscript{2} CNN where re-parameterized large convolutions are employed to build up large receptive fields. Our network in general follows the macro architecture of Swin Transformer [59] with a few modifications, while replacing the multi-head self-attentions with large depth-wise convolutions. We mainly benchmark middle-size and large-size models, since ViTs used to be believed to surpass CNNs on large data and models. On ImageNet classification, our baseline (similar model size with ours, the motivations are different: [38] does not investigate the relationship between ERFs, large kernels and performances; instead, it attributes the superior performances to sparse connections, shared parameters and dynamic mechanisms. Another three representative works are Global Filter Networks (GFNets) [72], CKConv [74] and FlexConv [73]. GFNet optimizes the spatial connection weights in the Fourier domain, which is equivalent to circular global convolutions in the spatial domain. CKConv formulates kernels as continuous functions to process sequential data, which can construct arbitrarily large kernels. FlexConv learns different kernel sizes for different layers, which can be as large as the feature maps. Although they use very large kernels, they do not intend to answer the

\textsuperscript{1}Convolutional kernels (including the variants such as depth-wise/group convolutions) whose spatial size is larger than 1×1.

\textsuperscript{2}Namely CNNs free of any attention or dynamic mechanism, e.g., squeeze-and-excitation [46], multi-head self-attention, dynamic weights [38, 95], and etc.
Table 1. Inference speed of a stack of 24-layer depth-wise convolutions with various kernel sizes and resolutions on a single GTX 2080Ti GPU. The input shape is (64, 384, R, R). Baselines are evaluated with Pytorch 1.9.0 + cuDNN 7.6.5, in FP32 precision.

<table>
<thead>
<tr>
<th>Resolution R</th>
<th>Impl</th>
<th>Latency (ms) @ Kernel size</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>16 × 16</td>
<td>5.6</td>
<td>11.0</td>
</tr>
<tr>
<td></td>
<td>5.6</td>
<td>6.5</td>
</tr>
<tr>
<td>32 × 32</td>
<td>21.9</td>
<td>34.1</td>
</tr>
<tr>
<td></td>
<td>21.9</td>
<td>28.7</td>
</tr>
<tr>
<td>64 × 64</td>
<td>69.6</td>
<td>141.2</td>
</tr>
<tr>
<td></td>
<td>69.6</td>
<td>112.6</td>
</tr>
</tbody>
</table>

Guidelines of Applying Large Convolutions

Trivially applying large convolutions to CNNs usually leads to inferior performance and speed. In this section, we summarize 5 guidelines for effectively using large kernels.

**Guideline 1:** large depth-wise convolutions can be efficient in practice. It is believed that large-kernel convolutions are computationally expensive because the kernel size quadratically increases the number of parameters and FLOPs. The drawback can be greatly overcome by applying depth-wise (DW) convolutions [17, 44]. For example, in our proposed RepLKNet (see Table 5 for details), increasing the kernel sizes in different stages from [3, 3, 3, 3] to [31, 29, 27, 13] only increases the FLOPs and number of parameters by 18.6% and 10.4% respectively, which is acceptable. The remaining 1 × 1 convolutions actually dominate most of the complexity.

One may concern that DW convolutions could be very inefficient on modern parallel computing devices like GPUs. It is true for conventional DW 3 × 3 kernels [44, 75, 109], because DW operations introduce low ratio of computation vs. memory access cost [64], which is not friendly to modern computing architecture. However, we find when kernel size becomes large, the computational density increases: for example, in a DW 11 × 11 kernel, each time we load a value from the feature map, it can attend at most 121 multiplications, while in a 3 × 3 kernel the number is only 9. Therefore, according to the roofline model, the actual latency should not increase as much as the increasing of FLOPs when kernel size becomes larger.

**Remark 1.** Unfortunately, we find off-the-shelf deep learning tools (such as Pytorch) support large DW convolutions poorly, as shown in Table 1. Hence we try several approaches to optimize the CUDA kernels. FFT-based approach [65] appears reasonable to implement large convolutions. However, in practice we find block-wise (inverse) implicit gemm algorithm is a better choice. The implementation has been integrated into the open-sourced framework MegEngine [1] and we omit the details here. We have also released an efficient implementation [2] for PyTorch. Table 1 shows that our implementation is far more efficient, compared with the Pytorch baseline. With our optimization, the latency contribution of DW convolutions in RepLKNet
Guideline 2: identity shortcut is vital especially for networks with very large kernels. To demonstrate this, we use MobileNet V2 [75] to benchmark, since it heavily uses DW layers and has two published variants (with or without shortcuts). For the large-kernel counterparts, we simply replace all the DW 3×3 layers with 13×13. All the models are trained on ImageNet with the identical training configurations for 100 epochs (see Appendix A for details). Table 2 shows large kernels improve the accuracy of MobileNet V2 with shortcuts by 0.77%. However, without shortcuts, large kernels reduce the accuracy to only 53.98%. Hence, without shortcuts, large kernels improves the optimization issue.

Guideline 3: re-parameterizing [30] with small kernels helps to make up the optimization issue. We replace the 3×3 layers of MobileNet V2 by 9×9 and 13×13 respectively, and optionally adopt Structural Re-parameterization [26,27,30] methodology. Specifically, we construct a 3×3 layer parallel to the large one, then add up their outputs after Batch normalization (BN) [49] layers (Fig. 2). After training, we merge the small kernel as well as BN parameters into the large kernel, so the resultant model is equivalent to the model for training but no longer has small kernels. Table 3 shows directly increasing the kernel size from 9 to 13 reduces the accuracy, while re-parameterization addresses the issue.

We then transfer the ImageNet-trained models to semantic segmentation with DeepLabv3+ [15] on Cityscapes [21]. We only replace the backbone and keep all the default training settings provided by MM Segmentation [19]. The observation is similar to that on ImageNet: 3×3 re-param improves the mIoU of the 9×9 model by 0.19 and the 13×13 model by 0.93. With such simple re-parameterization, increasing kernel size from 9 to 13 no longer degrades the performance on both ImageNet and Cityscapes.

Remark 3. It is known that ViTs have optimization problem especially on small datasets [34, 57]. A common workaround is to introduce convolutional prior, e.g., add a DW 3×3 convolution to each self-attention block [18, 96], which is analogous to ours. Those strategies introduce additional translational equivariance and locality prior to the network, making it easier to optimize on small dataset without loss of generality. Similar to what ViT behaves [34], we also find when the pretraining dataset increases to 73 million images (refer to RepLKNet-XL in the next section), re-parameterization can be omitted without degradation.

Guideline 4: large convolutions boost downstream tasks much more than ImageNet classification. Table 3 (after re-param) shows increasing the kernel size of MobileNet V2 from 3×3 to 9×9 improves the ImageNet accuracy by 1.33% but the Cityscapes mIoU by 3.99%. Table 5 shows a similar trend: as the kernel sizes increase from 3, 3, 3, 3 to 31, 29, 27, 13, the ImageNet accuracy improves by only 0.96%, while the mIoU on ADE20K [114] improves by 3.12%. Such phenomenon indicates that models of similar ImageNet scores could have very different capability in downstream tasks (just as the bottom 3 models in Table 5).

Remark 4. What causes the phenomenon? First, large kernel design significantly increases the Effective Receptive Fields (ERFs) [63]. Numerous works have demonstrated “contextual” information, which implies large ERFs, is crucial in many downstream tasks like object detection and semantic segmentation [61, 67, 93, 101, 102]. We will discuss the topic in Sec. 5. Second, We deem another reason might be that large kernel design contributes more shape biases to the network. Briefly speaking, ImageNet pictures can be correctly classified according to either texture or shape, as proposed in [7, 35]. However, humans recognize objects mainly based on shape cue rather than texture, therefore a model with stronger shape bias may transfer better to downstream tasks. A recent study [88] points out ViTs are strong in shape bias, which partially explains why ViTs are super powerful in transfer tasks. In contrast, conventional CNNs trained on ImageNet tend to bias towards texture [7, 35]. Fortunately, we find simply enlarging the kernel size in CNNs can effectively improve the shape bias. Please refer to Appendix C for details.
Table 4. Results of various kernel sizes in the last stage of MobileNet V2. Kernel sizes in previous stages remain to be 3 × 3.

<table>
<thead>
<tr>
<th>Kernel size</th>
<th>ImageNet acc (%)</th>
<th>Cityscapes mIoU (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 × 3</td>
<td>71.76</td>
<td>72.31</td>
</tr>
<tr>
<td>7 × 7</td>
<td>72.00</td>
<td>74.30</td>
</tr>
<tr>
<td>13 × 13</td>
<td>71.97</td>
<td>74.62</td>
</tr>
</tbody>
</table>

Guideline 5: large kernel (e.g., 13 × 13) is useful even on small feature maps (e.g., 7 × 7). To validate it, we enlarge the DW convolutions in the last stage of MobileNet V2 to 7 × 7 or 13 × 13, hence the kernel size is on par with or even larger than feature map size (7 × 7 by default). We apply re-parameterization to the large kernels as suggested by Guideline 3. Table 4 shows although convolutions in the last stage already involve very large receptive field, further increasing the kernel sizes still leads to performance improvements, especially on downstream tasks such as Cityscapes.

Remark 5. When kernel size becomes large, notice that translational equivariance of CNNs does not strictly hold. As illustrated in Fig. 3, two outputs at adjacent spatial locations share only a fraction of the kernel weights, i.e., transformed by different mappings. The property also agrees with the “philosophy” of ViTs – relaxing the symmetric prior to obtain more capacity. Interestingly, we find 2D Relative Position Embedding (RPE) [5, 76], which is widely used in the transformer community, can also be viewed as a large depth-wise kernel of size (2H − 1) × (2W − 1), where H and W are feature map height and width respectively. Large kernels not only help to learn the relative positions between concepts, but also encode the absolute position information due to padding effect [51].


Following the above guidelines, in this section we propose RepLKNet, a pure CNN architecture with large kernel design. To our knowledge, up to now CNNs still dominate small models [108, 110], while vision transformers are believed to be better than CNNs under more complexity budget. Therefore, in the paper we mainly focus on relatively large models (whose complexity is on par with or larger than ResNet-152 [40] or Swin-B [59]), in order to verify whether large kernel design could eliminate the performance gap between CNNs and ViTs.

4.1. Architecture Specification

We sketch the architecture of RepLKNet in Fig. 4:

Stem refers to the beginning layers. Since we target at high performance on downstream dense-prediction tasks, we desire to capture more details by several conv layers at the beginning. After the first 3 × 3 with 2 × downsampling, we arrange a DW 3 × 3 layer to capture low-level patterns, a 1 × 1 conv, and another DW 3 × 3 layer for downsampling.

Stages 1-4 each contains several RepLK Blocks, which use shortcuts (Guideline 2) and DW large kernels (Guideline 1). We use 1 × 1 conv before and after DW conv as a common practice. Note that each DW large conv uses a 5 × 5 kernel for re-parameterization (Guideline 3), which is not shown in Fig. 4. Except for the large conv layers which provide sufficient receptive field and the ability to aggregate spatial information, the model’s representational capacity is also closely related to the depth. To provide more nonlinearities and information communications across channels, we desire to use 1 × 1 layers to increase the depth. Inspired by the Feed-Forward Network (FFN) which has been widely used in transformers [34, 59] and MLPs [26, 84, 85], we use a similar CNN-style block composed of shortcut, BN, two 1 × 1 layers and GELU [41], so it is referred to as ConvFFN Block. Compared to the classic FFN which uses Layer Normalization [3] before the fully-connected layers, BN has an advantage that it can be fused into conv for efficient inference. As a common practice, the number of internal channels of the ConvFFN Block is 4 × as the input. Simply following ViT and Swin, which interleave attention and FFN blocks, we place a ConvFFN after each RepLK Block.

Transition Blocks are placed between stages, which first increase the channel dimension via 1 × 1 conv and then conduct 2 × downsampling with DW 3 × 3 conv.

In summary, each stage has three architectural hyper-parameters: the number of RepLK Blocks B, the channel dimension C, and the kernel size.
Table 5. RepLKNet with different kernel sizes. The models are pretrained on ImageNet-1K in 120 epochs with 224×224 input and finetuned on ADE20K with UperNet in 80K iterations. On ADE20K, we test the single-scale mIoU, and compute the FLOPs with input of 2048×512, following Swin.

<table>
<thead>
<tr>
<th>Kernel size</th>
<th>ImageNet Top-1 acc</th>
<th>ImageNet mIoU</th>
<th>ADE20K Params</th>
<th>ADE20K FLOPs</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-3-3-3</td>
<td>82.11</td>
<td>71.8M</td>
<td>12.9G</td>
<td>46.05</td>
</tr>
<tr>
<td>7-7-7-7</td>
<td>82.73</td>
<td>72.2M</td>
<td>13.1G</td>
<td>48.05</td>
</tr>
<tr>
<td>13-13-13-13</td>
<td>83.02</td>
<td>73.7M</td>
<td>13.4G</td>
<td>48.35</td>
</tr>
<tr>
<td>25-25-25-13</td>
<td>83.00</td>
<td>78.2M</td>
<td>14.8G</td>
<td>48.68</td>
</tr>
<tr>
<td>31-29-27-13</td>
<td>83.07</td>
<td>79.3M</td>
<td>15.3G</td>
<td>49.17</td>
</tr>
</tbody>
</table>

K. So that a RepLKNet architecture is defined by \([B_1, B_2, B_3, B_4], [C_1, C_2, C_3, C_4], [K_1, K_2, K_3, K_4]\).

4.2. Making Large Kernels Even Larger

We continue to evaluate large kernels on RepLKNet via fixing \(B=[2, 2, 18, 2]\), \(C=[128, 256, 512, 1024]\), varying \(K\) and observing the performance of both classification and semantic segmentation. Without careful tuning of the hyper-parameters, we casually set the kernel sizes as \([13, 13, 13, 13]\), \([25, 25, 25, 13]\), \([31, 29, 27, 13]\), respectively, and refer to the models as RepLKNet-13/25/31. We also construct two small-kernel baselines where the kernel sizes are all 3 or 7 (RepLKNet-3/7).

On ImageNet, we train for 120 epochs with AdamW [62] optimizer, RandAugment [22], mixup [106], CutMix [105], Rand Erasing [113] and Stochastic Depth [48], following the recent works [4, 59, 60, 86]. The detailed training configurations are presented in Appendix A.

For semantic segmentation, we use ADE20K [114], which is a widely-used large-scale semantic segmentation dataset containing 20K images of 150 categories for training and 2K for validation. We use the ImageNet-trained models as backbones and adopt UperNet [97] implemented by MMSegmentation [19] with the 80K-iteration training setting and test the single-scale mIoU.

Table 5 shows our results with different kernel sizes. On ImageNet, though increasing the kernel sizes from 3 to 13 improves the accuracy, making them even larger brings no further improvements. However, on ADE20K, scaling up the kernels from \([13, 13, 13, 13]\) to \([31, 29, 27, 13]\) brings 0.82 higher mIoU with only 5.3% more parameters and 3.5% higher FLOPs, which highlights the significance of large kernels for downstream tasks.

In the following subsections, we use RepLKNet-31 with stronger training configurations to compare with the state-of-the-arts on ImageNet classification, Cityscapes/ADE20K semantic segmentation and COCO [55] object detection. We refer to the aforementioned model as RepLKNet-31B (B for Base) and a wider model with \(C = [192, 384, 768, 1536]\) as RepLKNet-31L (Large). We construct another RepLKNet-XL with \(C = [256, 512, 1024, 2048]\) and 1.5× inverted bottleneck design in the RepLK Blocks (i.e., the channels of the DW large conv layers are 1.5× as the inputs).

4.3. ImageNet Classification

Since the overall architecture of RepLKNet is akin to Swin, we desire to make a comparison at first. For RepLKNet-31B on ImageNet-1K, we extend the aforementioned training schedule to 300 epochs for a fair comparison. Then we finetune for 30 epochs with input resolution of 384×384, so that the total training cost is much lower than the Swin-B model, which was trained with 384×384 from scratch. Then we pretrained RepLKNet-B/L models on ImageNet-22K and finetune on ImageNet-1K. RepLKNet-XL is pretrained on our private semi-supervised dataset.
Table 7. Cityscapes results. The FLOPs is computed with 1024×2048 inputs. The mIoU is tested with single-scale (ss) and multi-scale (ms). The results with 1K-pretrained Swin are cited from the official GitHub repository. ‡ indicates ImageNet-22K pretraining.

<table>
<thead>
<tr>
<th>Backbone</th>
<th>Method</th>
<th>mIoU (ss)</th>
<th>mIoU (ms)</th>
<th>Param (M)</th>
<th>FLOPs (G)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RepLKNet-31B</td>
<td>UperNet [97]</td>
<td>83.1</td>
<td>83.5</td>
<td>110</td>
<td>2315</td>
</tr>
<tr>
<td>ResNetSt-200</td>
<td>DeepLabv3 [14]</td>
<td>-</td>
<td>82.7</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Axial-Res-XL</td>
<td>Axial-DL [92]</td>
<td>80.6</td>
<td>81.1</td>
<td>173</td>
<td>2446</td>
</tr>
<tr>
<td>Swin-B</td>
<td>UperNet</td>
<td>80.4</td>
<td>81.5</td>
<td>121</td>
<td>2613</td>
</tr>
<tr>
<td>Swin-B</td>
<td>UperNet + [36]</td>
<td>80.8</td>
<td>81.8</td>
<td>121</td>
<td>-</td>
</tr>
<tr>
<td>ViT-L ‡</td>
<td>SETR-PUP [112]</td>
<td>79.3</td>
<td>82.1</td>
<td>318</td>
<td>-</td>
</tr>
<tr>
<td>ViT-L ‡</td>
<td>SETR-MLA</td>
<td>77.2</td>
<td>-</td>
<td>310</td>
<td>-</td>
</tr>
<tr>
<td>Swin-L ‡</td>
<td>UperNet</td>
<td>82.3</td>
<td>83.1</td>
<td>234</td>
<td>3771</td>
</tr>
<tr>
<td>Swin-L ‡</td>
<td>UperNet + [36]</td>
<td>82.7</td>
<td>83.6</td>
<td>234</td>
<td>-</td>
</tr>
</tbody>
</table>

For the ImageNet-22K and larger, following Swin.

22K pretraining and 640 finetuning on ADE20K. Faster, which highlights the efficiency of very large kernels.

RepLKNet-XL has higher FLOPs than Swin-L, it runs higher than Swin-B, and runs 43% faster. And even though RepLKNet-31B achieves an mIoU of 2.7, and even outperforms the ImageNet-22K-pretrained Swin-L. Even equipped with DiversePatch [36], a technique customized for vision transformers, the single-scale mIoU of the 22K-pretrained Swin-L is still lower than our 1K-pretrained RepLKNet-31B, though the former has 2× parameters.

On ADE20K, RepLKNet-31B outperforms Swin-B with both 1K and 22K pretraining, and the margins of single-scale mIoU are particularly significant. Pretrained with our semi-supervised dataset MegData73M, RepLKNet-XL achieves an mIoU of 56.0, which shows feasible scalability towards large-scale vision applications.

4.4. Semantic Segmentation

We then use the pretrained models as the backbones on Cityscapes (Table 7) and ADE20K (Table 8). Specifically, we use the UperNet [97] implemented by MMsegmentation [19] with the 80K-iteration training schedule for Cityscapes and 160K for ADE20K. Since we desire to evaluate the backbone only, we do not use any advanced techniques, tricks, or custom algorithms.

On Cityscapes, ImageNet-1K-pretrained RepLKNet-31B outperforms Swin-B by a significant margin (single-scale mIoU of 2.7), and even outperforms the ImageNet-22K-pretrained Swin-L. Even equipped with DiversePatch [36], a technique customized for vision transformers, the single-scale mIoU of the 22K-pretrained Swin-L is still lower than our 1K-pretrained RepLKNet-31B, though the former has 2× parameters.

On ADE20K, RepLKNet-31B outperforms Swin-B with both 1K and 22K pretraining, and the margins of single-scale mIoU are particularly significant. Pretrained with our semi-supervised dataset MegData73M, RepLKNet-XL achieves an mIoU of 56.0, which shows feasible scalability towards large-scale vision applications.

4.5. Object Detection

For object detection, we use RepLKNe as the backbone of FCOS [83] and Cascade Mask R-CNN [8,39], which are representatives of one-stage and two-stage detection methods, and the default configurations in MMDetection [12]. The FCOS model is trained with the 2x (24-epoch) training schedule for a fair comparison with the X101 (short for ResNeXt-101 [99]) baseline from the same code base [19], and the other results with Cascade Mask R-CNN all use 3x (36-epoch). Again, we simply replace the backbone and do not use any advanced techniques. Table 9 shows RepLKNe outperforms ResNeXt-101-64x4d by up to 4.4 mAP while have fewer parameters and lower FLOPs. Note that the results may be further improved with the advanced techniques like HTC [11], HTC++ [59], Soft-NMS [6] or a 6x (72-epoch) schedule. Compared to Swin, RepLKNe achieves higher or comparable mAP with fewer parameters and lower FLOPs. Notably, RepLKNet-XL achieves an mAP of 55.5, which demonstrates the scalability again.

5. Discussions

1) Large-Kernel CNNs have Larger ERF than Deep Small-Kernel Models. We have demonstrated large kernel design can significantly boost CNNs (especially on downstream tasks). However, it is worth noting that large kernel can be expressed by a series of small convolutions [77], e.g., a 7×7 convolution can be decomposed into a stack of three 3×3 kernels without information loss (more channels are required after the decomposition to maintain the degree.
Briefly, we produce an aggregated contribution score made to be used in the ERF, we use a simple yet effective method (code released at [2]) as introduced in Appendix B, following [52].

We have found out that RepLKNet-31B has much higher shape bias than Swin Transformer and small-kernel CNNs. Please refer to Appendix C for details.

2) Large-kernel Models are More Similar to Human in Shape Bias. We demonstrate that using a few large kernels instead of many small kernels results in larger ERFs. ResNet-152 shows a similar pattern, suggesting the more 3×3 layers do not significantly increase the ERF. On the other hand, the high-contribution pixels in Fig. 1 (C) are more evenly distributed, suggesting RepLKNet-13 attends to more outer pixels. With larger kernels, RepLKNet-31 makes the high-contribution pixels spread more uniformly, indicating an even larger ERF. Apart from the visualization, a quantitative analysis is also presented in Appendix B.

3) Large kernel design is a generic design element that works with ConvNeXt. Replacing the $7 \times 7$ convolutions in ConvNeXt [60] by kernels as large as $31 \times 31$ brings significant improvements, e.g., ConNeXt-Tiny + large kernel > ConNeXt-Small , and ConNeXt-Small + large kernel > ConNeXt-Base. Please refer to Appendix D.

4) Large kernels outperform small kernels with high dilation rates. Please refer to Appendix E for details.

6. Limitations

Although large kernel design greatly improves CNNs on both ImageNet and downstream tasks, however, according to Table 6, as the scale of data and model increases, RepLK Nets start to fall behind Swin Transformers, e.g., the ImageNet top-1 accuracy of RepLKNet-31L is 0.7% lower than Swin-L with ImageNet-22K pretraining (while the downstream scores are still comparable). It is not clear whether the gap is resulted from suboptimal hyperparameter tuning or some other fundamental drawback of CNNs which emerges when data/model scales up. We are working in progress on the problem.

7. Conclusion

This paper revisits large convolutional kernels, which have long been neglected in designing CNN architectures. We demonstrate that using a few large kernels instead of many small kernels results in larger effective receptive field more efficiently, boosting CNN’s performances especially on downstream tasks by a large margin, and greatly closing the performance gap between CNNs and ViTs when data and models scale up. We hope our work could advance both studies of CNNs and ViTs. On one hand, for CNN community, our findings suggest that we should pay special attention to ERFs, which may be the key to high performances. On the other hand, for ViT community, since large convolutions act as an alternative to multi-head self-attentions with similar behaviors, it may help to understand the intrinsic mechanism of self-attentions.
References


