UniRepLKNet: A Universal Perception Large-Kernel ConvNet for Audio, Video, Point Cloud, Time-Series and Image Recognition

Supplementary Material

Appendix A: General Transformation from Dialted Convolution to Non-dilated Large-Kernel Convolution

Since *ignoring pixels of the input is equivalent to inserting extra zero entries into the conv kernel*, *a dilated conv layer with a small kernel can be equivalently converted into a non-dilated layer with a sparse larger kernel*. Let k be the kernel size and r be the dilation rate of the dilated layer, by inserting zero entries, the kernel size of the corresponding non-dilated layer will be $(k-1)r+1$, which is referred to as the *equivalent kernel size* for brevity.

As discussed in the paper, to eliminate the inference costs of the extra dilated conv layers in the Dilated Reparam Block, we propose to equivalently transform the whole block into a single non-dilated conv layer for inference. As discussed before, let k and r be the kernel size and dilation rate, respectively, the transformation from a dilated conv layer's kernel $\mathrm{W} \in \mathcal{R}^{k\times k}$ to a non-dilated layer's kernel $\mathrm{W}' \in \mathcal{R}^{((k-1)r+1)\times((k-1)r+1)}$ can be elegantly realized by a transpose convolution with a stride of r and an identity kernel $\mathrm{I} \in \mathcal{R}^{1\times 1}$, which is scalar 1 but viewed as a kernel tensor. That is

$$\mathrm{W}' = \texttt{conv\_transpose2d}(\mathrm{W}, \mathrm{I}, \text{stride} = r)\,. \quad (6)$$

In general cases with multi-channel conv layers, let the input channels, output channels, and number of groups be c_{in}, c_{out}, and g, respectively, we denote the kernel by a 4D tensor whose shape is $c_{\text{out}} \times \frac{c_{\text{in}}}{g} \times k \times k$.

1) For a multi-channel depthwise (DW) layer, the transformation is easily generalized from 2D to 4D - the identity kernel I is viewed as a 4D tensor $\mathrm{I} \in \mathcal{R}^{1\times 1\times 1\times 1}$ and we still follow function 6 to derive the equivalent kernel by transpose convolution.

2) For non-DW cases (*i.e.*, $g < c_{\text{in}}$), the transformation can be seen as splitting the kernel into slices (which can each be seen as a DW kernel), converting the slices respectively, and concatenating the resultant non-dilated slices up. We present the code in pytorch (Fig. 4) and a test case demonstrating the equivalency (Fig. 5).

Appendix B: Training Configurations

We present the detailed training configurations for image classification, object detection, and semantic segmentation. We have publicly released a reproducible training script and trained weights for every model on GitHub.

ImageNet image classification. The training configurations for the ImageNet-1K-only results shown in Section 4 are presented in Table 14. These configurations are similar to common practices. For the experiments in Section 3, we use the same configurations, except that the training epochs are set to 100 and the drop path rate is set to 0.1. For the models pretrained with ImageNet-22K and then fine-tuned on ImageNet-22K, the configurations are shown in Table 14. Note that we follow the configurations adopted by ConvNeXt for a fair comparison with ConvNeXt-S/B, and the configurations used by InternImage for a fair comparison with InternImage-L/XL (the results with ImageNet-22K-pretrained InternImage-S/B were not reported).

COCO object detection. For fair comparisons, we follow common practices [49, 52] to initialize the backbone with pretrained weights and train the models using a 3× (36 epochs) schedule by default. The shorter side is resized to 480−800 pixels, while the longer side does not exceed 1,333 pixels. All the models are trained with a batch size of 16 and AdamW [53] optimizer with an initial learning rate of 1×10^{-4}. We have publicly released the training configuration files used in the MMDetection framework and trained weights.

ADE20K semantic segmentation. We evaluate UniRepLKNet models on the ADE20K dataset [97], and initialize them with the pre-trained classification weights. The learning rate is initialized with 1×10^{-4} and decayed with the polynomial decay schedule with a power of 1.0. Following previous methods [49, 52], the crop size is set to 512 for the ImageNet-1K-pretrained models, and 640 for ImageNet-22K-pretrained models. All segmentation models are trained with a batch size of 16 for 160k iterations. We have publicly released the training configuration files used in the MMSegmentation framework and trained weights.

Appendix C: Shape Bias

A higher shape bias means the model makes predictions based more on the shape of objects rather than the textures, *i.e.*, the model behaves more similarly to humans. Therefore, a model with a higher shape bias may transfer better to downstream tasks. UniRepLKNet demonstrates significantly higher shape bias than existing ConvNets and ViTs. Concretely, we test the shape bias of ImageNet-22K-pretrained UniRepLKNet-L and RepLKNet-L with the *modelvshuman* toolbox [5]. Fig. 6 shows a significantly

[5] https://github.com/bethgelab/model-vs-human

```
import torch
import torch.nn as nn
import torch.nn.functional as F

def convert_dilated_to_nondilated(kernel, dilate_rate):
   identity_kernel = torch.ones((1, 1, 1, 1))
   if kernel.size(1) == 1:
      # This is a DW kernel
      dilated = F.conv_transpose2d(kernel, identity_kernel, stride=dilate_rate)
      return dilated
   else:
      # This is a dense or group-wise (but not DW) kernel
      slices = []
      for i in range(kernel.size(1)):
         dilated = F.conv_transpose2d(kernel[:,i:i+1,:,:], identity_kernel, stride=
             dilate_rate)
         slices.append(dilated)
      return torch.cat(slices, dim=1)
```

Figure 4. Pytorch code to convert a dilated conv layer's small kernel to a non-dilated layer's larger sparse kernel.

```
def test_equivalency(in_channels, out_channels, groups, large_kernel_size, small_conv_r,
    small_conv_k):
   equivalent_kernel_size = small_conv_r * (small_conv_k - 1) + 1
   large_conv = nn.Conv2d(in_channels, out_channels, kernel_size=large_kernel_size,
                     padding=large_kernel_size // 2, groups=groups, bias=False)
   dilated_conv = nn.Conv2d(in_channels, out_channels, kernel_size=small_conv_k,
   padding=equivalent_kernel_size // 2,
                      dilation=small_conv_r, groups=groups, bias=False)
   H, W = 19, 19
   x = torch.rand(2, in_channels, H, W)
   origin_y = large_conv(x) + dilated_conv(x)
   equivalent_kernel = convert_dilated_to_nondilated(dilated_conv.weight.data, small_conv_r)
   rows_to_pad = large_kernel_size // 2 - equivalent_kernel_size // 2
   merged_kernel = large_conv.weight.data + F.pad(equivalent_kernel, [rows_to_pad] * 4)
   equivalent_y = F.conv2d(x, merged_kernel, bias=None, padding=large_kernel_size // 2,
       groups=groups)
   print('relative error:', (equivalent_y - origin_y).abs().sum() / origin_y.abs().sum())

test_equivalency(in_channels=4, out_channels=4, groups=1,
   large_kernel_size=13, small_conv_r=3, small_conv_k=3)
```

Figure 5. A test case demonstrating the equivalency of the transformation.

higher shape bias of UniRepLKNet - UniRepLKNet makes 20% more decisions based on the overall shapes of objects. This improvement is particularly remarkable since RepLKNet is already known to have a high shape bias (Fig. 7 is directly taken from the supplementary material of the RepLKNet paper without any modifications).

6.1. Appendix D: Training Memory Footprint

The extra parallel dilated branches in Dilated Reparam Block consume more training resources, which is acceptable considering the performance improvements. We present the peak GPU memory footprint and training speed in Table 16. With a bigger model and bigger data, we may trade the performance for higher training speed and lower memory consumption by replacing the Dilated Reparam Block with a single large-kernel conv layer followed by Batch Normalization layer. We test the peak memory footprint and actual training throughput while training UniRepLKNet-S with 224×224 inputs and a batch size of 4096 on a node with eight A100 GPUs. Note that such re-

Table 14. **Detailed training configurations of ImageNet-1K-only models.** Apart from the configurations shown in the table, we use random left-right flipping, random resized crop, color jitter of 0.4, Auto-augment, and no repeated augmentation for every model.

settings	UniRepLKNet-A	UniRepLKNet-F	UniRepLKNet-P	UniRepLKNet-N	UniRepLKNet-T	UniRepLKNet-S
input scale	224	224	224	224	224	224
batch size	4096	4096	4096	4096	4096	4096
optimizer	AdamW	AdamW	AdamW	AdamW	AdamW	AdamW
LR	4×10^{-3}	4×10^{-3}	4×10^{-3}	4×10^{-3}	4×10^{-3}	4×10^{-3}
LR schedule	cosine	cosine	cosine	cosine	cosine	cosine
weight decay	0.05	0.05	0.05	0.05	0.05	0.05
warmup epochs	5	5	5	5	5	5
epochs	300	300	300	300	300	300
mixup alpha	0.3	0.3	0.3	0.5	0.8	0.8
cutmix alpha	0.3	0.3	0.3	0.5	1.0	1.0
erasing prob.	0.25	0.25	0.25	0.25	0.25	0.25
label smoothing ε	0.1	0.1	0.1	0.1	0.1	0.1
drop path rate	0.0	0.0	0.1	0.1	0.2	0.4

Table 15. **Detailed training configurations of models pretrained with ImageNet-22K (IN-22K pt) and then finetuned on ImageNet-1K (IN-1K ft).** Apart from the configurations shown in the table, we use random left-right flipping, random resized crop, color jitter of 0.4, Auto-augment, and no repeated augmentation for every model.

settings	UniRepLKNet-S		UniRepLKNet-B		UniRepLKNet-L		UniRepLKNet-XL	
	IN-22K pt	IN-1K ft	IN-22K pt	IN-1K ft	IN-22K pt	IN-1K ft	IN-22K pt	IN-1K ft
input scale	224	384	224	384	192	384	192	384
batch size	4096	512	4096	512	4096	512	4096	512
optimizer	AdamW	AdamW	AdamW	AdamW	AdamW	AdamW	AdamW	AdamW
LR	4×10^{-3}	5×10^{-5}	4×10^{-3}	5×10^{-5}	4×10^{-3}	5×10^{-5}	4×10^{-3}	5×10^{-5}
LR schedule	cosine	cosine	cosine	cosine	cosine	cosine	cosine	cosine
weight decay	0.05	1×10^{-8}	0.05	1×10^{-8}	0.05	1×10^{-8}	0.05	1×10^{-8}
warmup epochs	5	0	5	0	5	0	5	0
epochs	90	30	90	30	90	20	90	20
mixup alpha	0.8	0.0	0.8	0.0	0.8	0.0	0.8	0.0
cutmix alpha	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0
erasing prob.	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25
label smoothing	0.1	0.1	0.1	0.1	0.1	0.3	0.1	0.3
drop path rate	0.1	0.2	0.1	0.2	0.1	0.3	0.2	0.3

Table 16. Training costs.

	Peak memory	Training throughput
Dilated Reparam Block	24.6GB	6642 images/s
Single large-kernel conv layer	20.8GB	9675 images/s

sults are significantly influenced by the hardware environment and specific implementation; thus, they should be considered as references only.

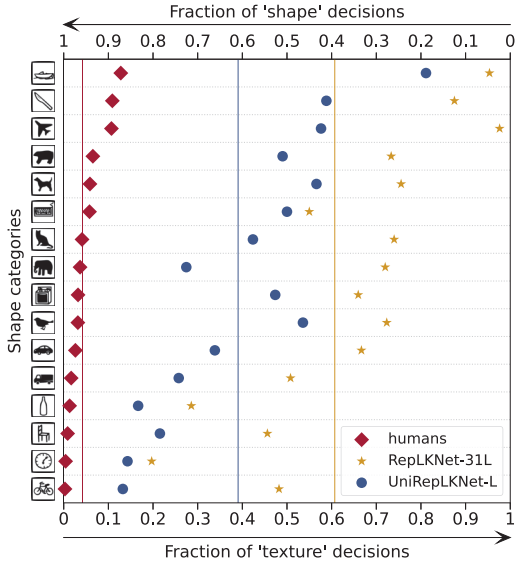

Figure 6. Shape bias of ImageNet-22K-pretrained UniRepLKNet-L and RepLKNet-31L.

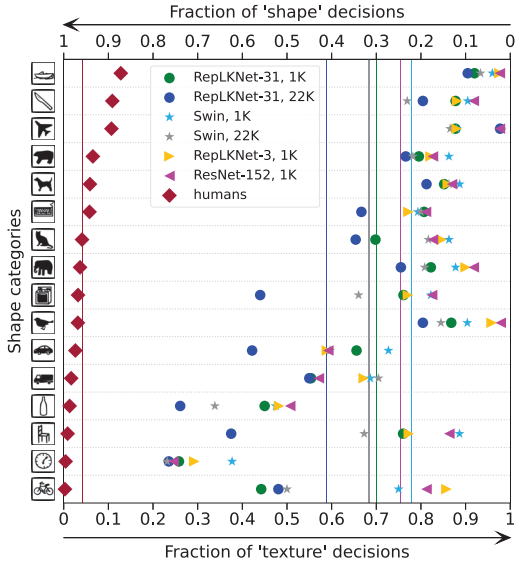

Figure 7. Shape bias of ImageNet-1K and ImageNet-22K-pretrained RepLKNet-31B and Swin-B. This figure is directly taken from the supplementary material of RepLKNet without any modifications