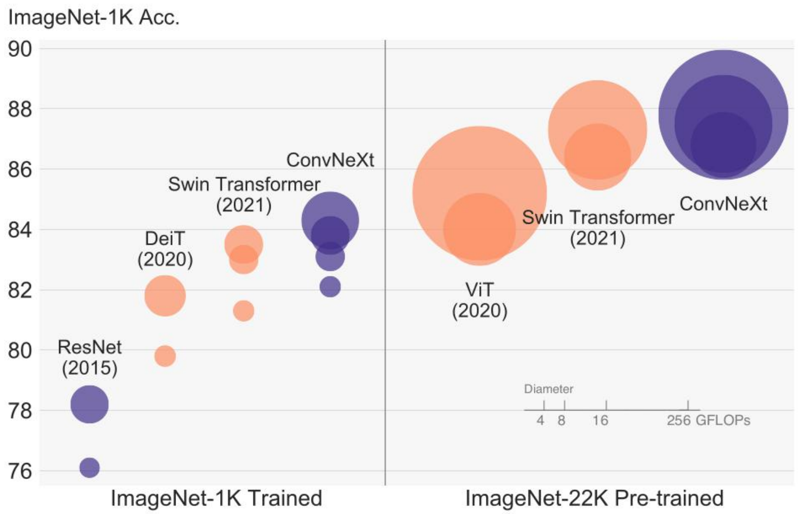
ConvNeXt

A ConvNet for the 2020s

Zhuang Liu[1,2*] Hanzi Mao[1] Chao-Yuan Wu[1] Christoph Feichtenhofer[1] Trevor Darrell[2] Saining Xie[1†]
[1]Facebook AI Research (FAIR) [2]UC Berkeley
Code: https://github.com/facebookresearch/ConvNeXt



论文地址：https://arxiv.org/abs/2201.03545
推荐博文：https://blog.csdn.net/qq_37541097/article/details/122556545

ConvNeXt

model	image size	FLOPs	throughput (image / s)	IN-1K / 22K trained, 1K acc.
○ Swin-T	224^2	4.5G	1325.6	81.3 / –
● ConvNeXt-T	224^2	4.5G	**1943.5** (+47%)	**82.1** / –
○ Swin-S	224^2	8.7G	857.3	83.0 / –
● ConvNeXt-S	224^2	8.7G	**1275.3** (+49%)	**83.1** / –
○ Swin-B	224^2	15.4G	662.8	83.5 / 85.2
● ConvNeXt-B	224^2	15.4G	**969.0** (+46%)	**83.8 / 85.8**
○ Swin-B	384^2	47.1G	242.5	84.5 / 86.4
● ConvNeXt-B	384^2	45.0G	**336.6** (+39%)	**85.1 / 86.8**
○ Swin-L	224^2	34.5G	435.9	– / 86.3
● ConvNeXt-L	224^2	34.4G	**611.5** (+40%)	**84.3 / 86.6**
○ Swin-L	384^2	103.9G	157.9	– / 87.3
● ConvNeXt-L	384^2	101.0G	**211.4** (+34%)	85.5 / **87.5**
● ConvNeXt-XL	224^2	60.9G	**424.4**	– / **87.0**
● ConvNeXt-XL	384^2	179.0G	**147.4**	– / **87.8**

Table 12. **Inference throughput comparisons on an A100 GPU.** ConvNeXt enjoys up to ∼49% higher throughput compared with a Swin Transformer with similar FLOPs.

ConvNeXt

backbone	FLOPs	FPS	AP^{box}	AP^{box}_{50}	AP^{box}_{75}	AP^{mask}	AP^{mask}_{50}	AP^{mask}_{75}
Mask-RCNN 3× schedule								
○ Swin-T	267G	23.1	46.0	68.1	50.3	41.6	65.1	44.9
● ConvNeXt-T	262G	25.6	**46.2**	67.9	50.8	**41.7**	65.0	44.9
Cascade Mask-RCNN 3× schedule								
● ResNet-50	739G	11.4	46.3	64.3	50.5	40.1	61.7	43.4
● X101-32	819G	9.2	48.1	66.5	52.4	41.6	63.9	45.2
● X101-64	972G	7.1	48.3	66.4	52.3	41.7	64.0	45.1
○ Swin-T	745G	12.2	50.4	69.2	54.7	43.7	66.6	47.3
● ConvNeXt-T	741G	13.5	**50.4**	69.1	54.8	**43.7**	66.5	47.3
○ Swin-S	838G	11.4	51.9	70.7	56.3	45.0	68.2	48.8
● ConvNeXt-S	827G	12.0	**51.9**	70.8	56.5	**45.0**	68.4	49.1
○ Swin-B	982G	10.7	51.9	70.5	56.4	45.0	68.1	48.9
● ConvNeXt-B	964G	11.4	**52.7**	71.3	57.2	**45.6**	68.9	49.5
○ Swin-B‡	982G	10.7	53.0	71.8	57.5	45.8	69.4	49.7
● ConvNeXt-B‡	964G	11.5	**54.0**	73.1	58.8	**46.9**	70.6	51.3
○ Swin-L‡	1382G	9.2	53.9	72.4	58.8	46.7	70.1	50.8
● ConvNeXt-L‡	1354G	10.0	**54.8**	73.8	59.8	**47.6**	71.3	51.7
● ConvNeXt-XL‡	1898G	8.6	**55.2**	74.2	59.9	**47.7**	71.6	52.2

Table 3. **COCO object detection and segmentation results** using Mask-RCNN and Cascade Mask-RCNN. ‡ indicates that the model is pre-trained on ImageNet-22K. ImageNet-1K pre-trained Swin results are from their Github repository [3]. AP numbers of the ResNet-50 and X101 models are from [42]. We measure FPS on an A100 GPU. FLOPs are calculated with image size (1280, 800).

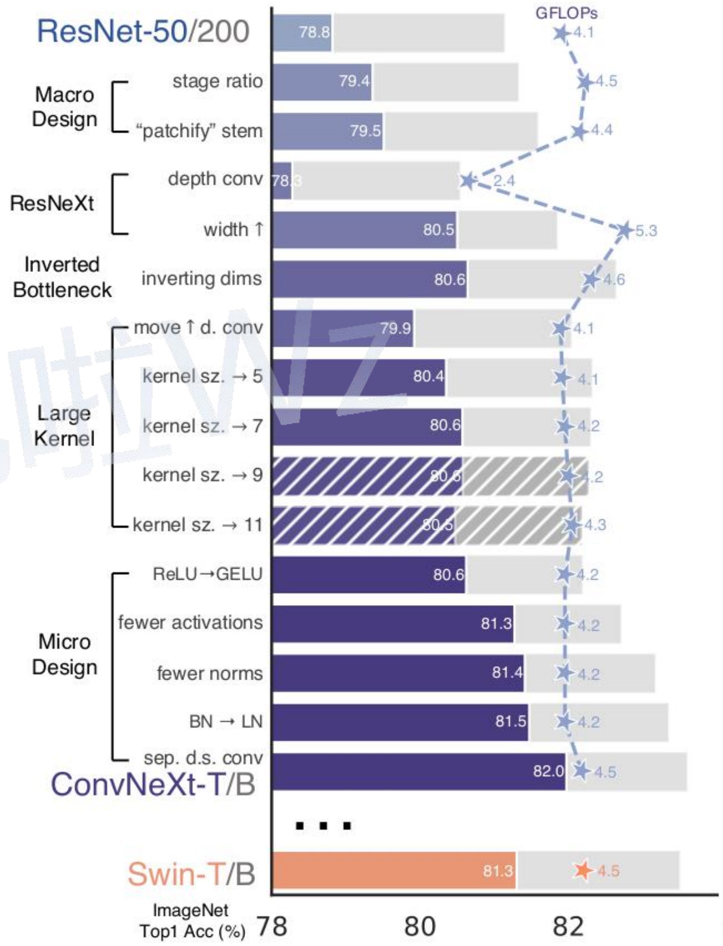
backbone	input crop.	mIoU	#param.	FLOPs
ImageNet-1K pre-trained				
○ Swin-T	512^2	45.8	60M	945G
● ConvNeXt-T	512^2	**46.7**	60M	939G
○ Swin-S	512^2	49.5	81M	1038G
● ConvNeXt-S	512^2	**49.6**	82M	1027G
○ Swin-B	512^2	49.7	121M	1188G
● ConvNeXt-B	512^2	**49.9**	122M	1170G
ImageNet-22K pre-trained				
○ Swin-B‡	640^2	51.7	121M	1841G
● ConvNeXt-B‡	640^2	**53.1**	122M	1828G
○ Swin-L‡	640^2	53.5	234M	2468G
● ConvNeXt-L‡	640^2	**53.7**	235M	2458G
● ConvNeXt-XL‡	640^2	**54.0**	391M	3335G

Table 4. **ADE20K validation results** using UperNet [80]. ‡ indicates IN-22K pre-training. Swins' results are from its GitHub repository [2]. Following Swin, we report mIoU results with multi-scale testing. FLOPs are based on input sizes of (2048, 512) and (2560, 640) for IN-1K and IN-22K pre-trained models, respectively.

ConvNeXt

ConvNeXt设计与实验

- Macro design
- ResNeXt
- Inverted bottleneck
- Large kerner size
- Various layer-wise Micro designs



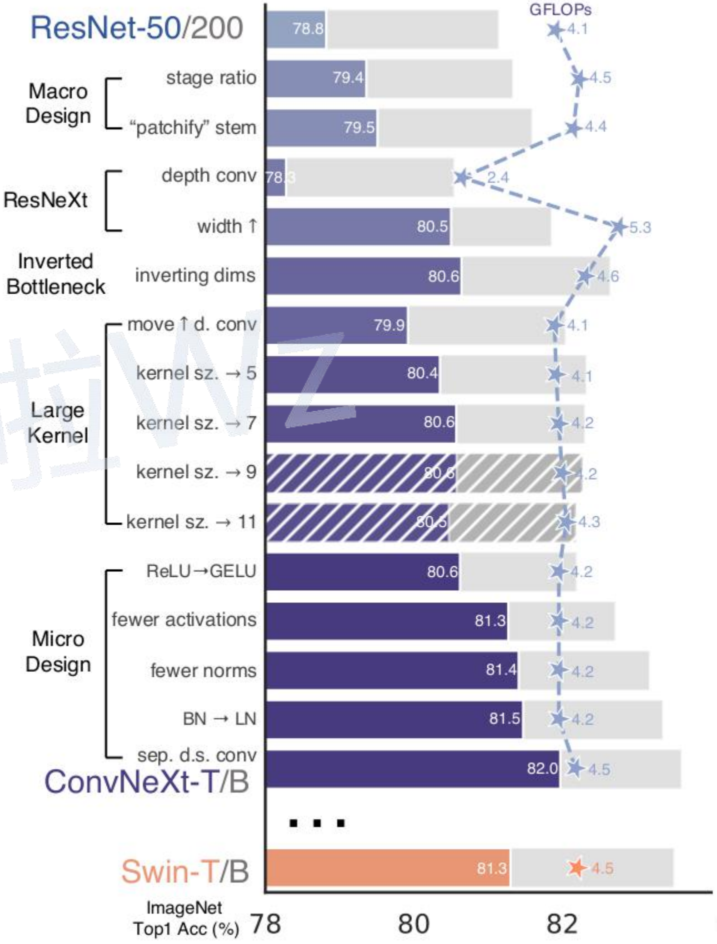
ConvNeXt

Macro design

Swin-T的比例是1:1:3:1
Swin-L的比例是1:1:9:1

作者将ResNet50中的堆叠次数由(3, 4, 6, 3)调整成(3, 3, 9, 3)

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		[3×3, 64; 3×3, 64]×2	[3×3, 64; 3×3, 64]×3	[1×1, 64; 3×3, 64; 1×1, 256]×3	[1×1, 64; 3×3, 64; 1×1, 256]×3	[1×1, 64; 3×3, 64; 1×1, 256]×3
conv3_x	28×28	[3×3, 128; 3×3, 128]×2	[3×3, 128; 3×3, 128]×4	[1×1, 128; 3×3, 128; 1×1, 512]×4	[1×1, 128; 3×3, 128; 1×1, 512]×4	[1×1, 128; 3×3, 128; 1×1, 512]×8
conv4_x	14×14	[3×3, 256; 3×3, 256]×2	[3×3, 256; 3×3, 256]×6	[1×1, 256; 3×3, 256; 1×1, 1024]×6	[1×1, 256; 3×3, 256; 1×1, 1024]×23	[1×1, 256; 3×3, 256; 1×1, 1024]×36
conv5_x	7×7	[3×3, 512; 3×3, 512]×2	[3×3, 512; 3×3, 512]×3	[1×1, 512; 3×3, 512; 1×1, 2048]×3	[1×1, 512; 3×3, 512; 1×1, 2048]×3	[1×1, 512; 3×3, 512; 1×1, 2048]×3
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

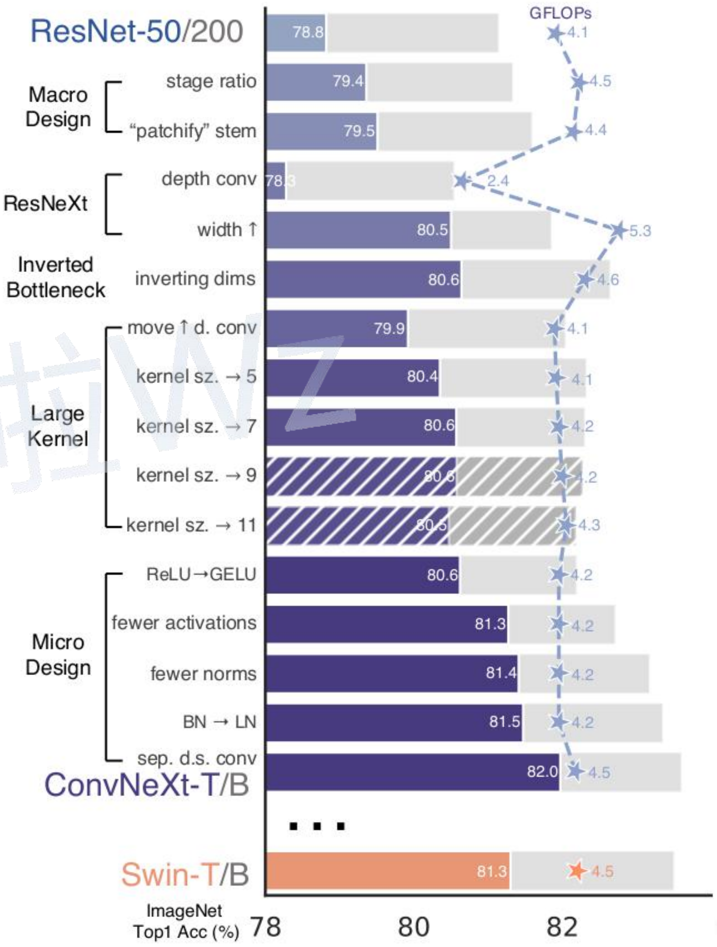


ConvNeXt

Macro design

作者将stem换成卷积核大小为4，步距为4的卷积层

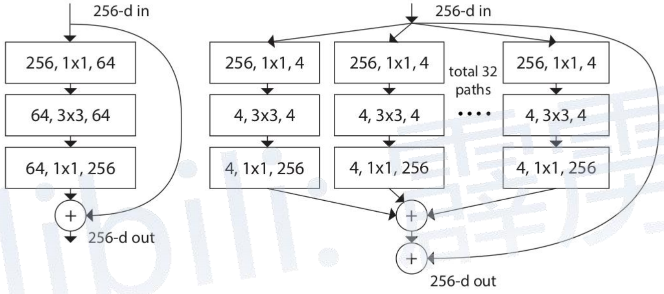
layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		[3×3, 64 3×3, 64] ×2	[3×3, 64 3×3, 64] ×3	[1×1, 64 3×3, 64 1×1, 256] ×3	[1×1, 64 3×3, 64 1×1, 256] ×3	[1×1, 64 3×3, 64 1×1, 256] ×3
conv3_x	28×28	[3×3, 128 3×3, 128] ×2	[3×3, 128 3×3, 128] ×4	[1×1, 128 3×3, 128 1×1, 512] ×4	[1×1, 128 3×3, 128 1×1, 512] ×4	[1×1, 128 3×3, 128 1×1, 512] ×8
conv4_x	14×14	[3×3, 256 3×3, 256] ×2	[3×3, 256 3×3, 256] ×6	[1×1, 256 3×3, 256 1×1, 1024] ×6	[1×1, 256 3×3, 256 1×1, 1024] ×23	[1×1, 256 3×3, 256 1×1, 1024] ×36
conv5_x	7×7	[3×3, 512 3×3, 512] ×2	[3×3, 512 3×3, 512] ×3	[1×1, 512 3×3, 512 1×1, 2048] ×3	[1×1, 512 3×3, 512 1×1, 2048] ×3	[1×1, 512 3×3, 512 1×1, 2048] ×3
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

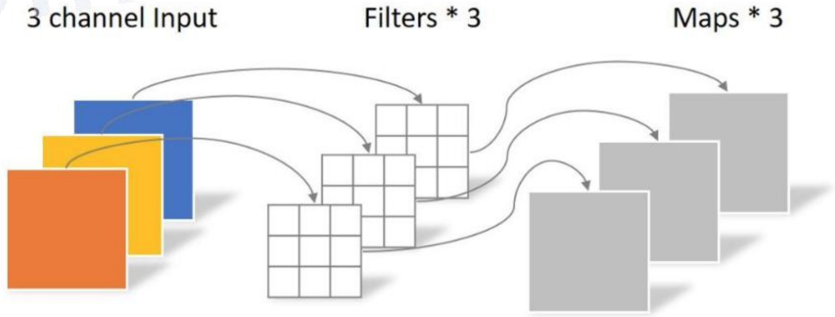


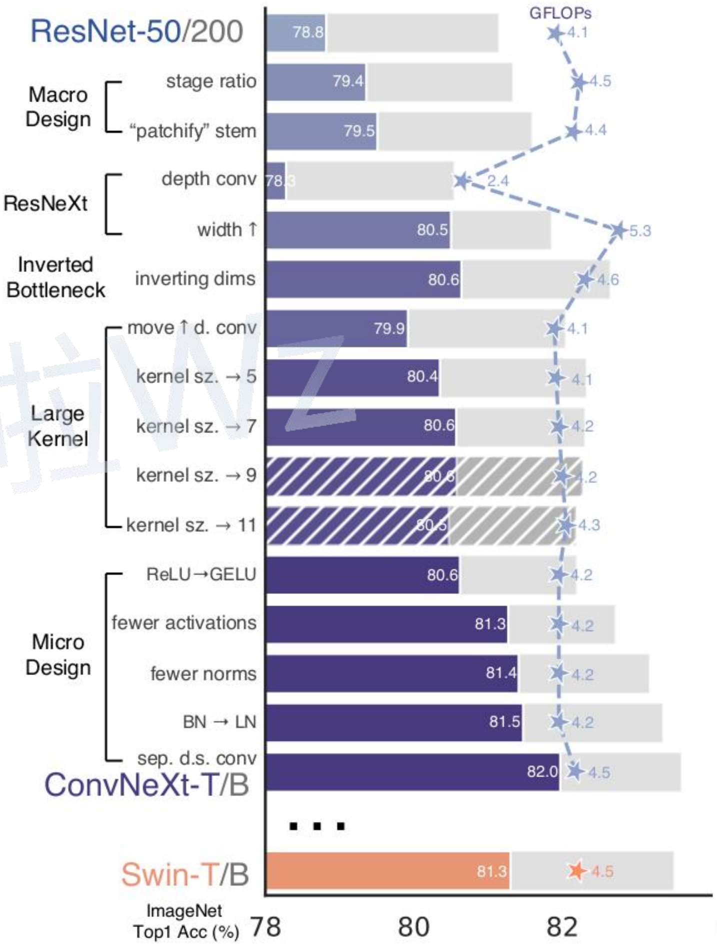
ConvNeXt

ResNeXt

ResNeXt相比普通的ResNet而言在FLOPs以及accuracy之间做到了更好的平衡。这里作者采用的是更激进的depthwise convolution。







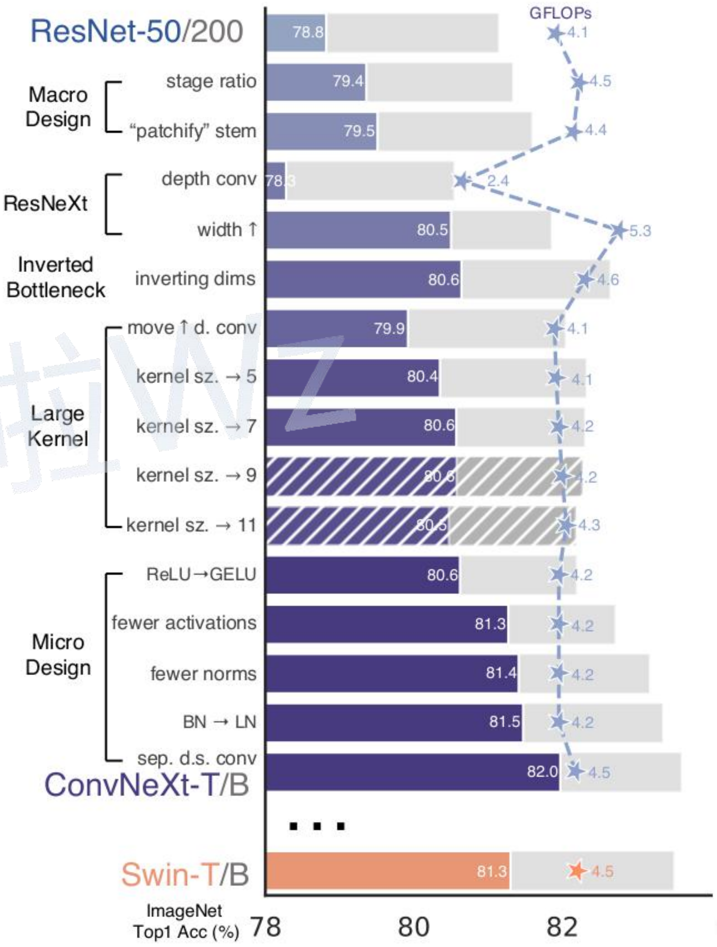
ConvNeXt

Macro design

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		[3×3, 64 3×3, 64] ×2	[3×3, 64 3×3, 64] ×3	[1×1, 64 3×3, 64 1×1, 256] ×3	[1×1, 64 3×3, 64 1×1, 256] ×3	[1×1, 64 3×3, 64 1×1, 256] ×3
conv3_x	28×28	[3×3, 128 3×3, 128] ×2	[3×3, 128 3×3, 128] ×4	[1×1, 128 3×3, 128 1×1, 512] ×4	[1×1, 128 3×3, 128 1×1, 512] ×4	[1×1, 128 3×3, 128 1×1, 512] ×8
conv4_x	14×14	[3×3, 256 3×3, 256] ×2	[3×3, 256 3×3, 256] ×6	[1×1, 256 3×3, 256 1×1, 1024] ×6	[1×1, 256 3×3, 256 1×1, 1024] ×23	[1×1, 256 3×3, 256 1×1, 1024] ×36
conv5_x	7×7	[3×3, 512 3×3, 512] ×2	[3×3, 512 3×3, 512] ×3	[1×1, 512 3×3, 512 1×1, 2048] ×3	[1×1, 512 3×3, 512 1×1, 2048] ×3	[1×1, 512 3×3, 512 1×1, 2048] ×3
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

	downsp. rate (output size)	Swin-T	Swin-S	Swin-B	Swin-L
stage 1	4× (56×56)	concat 4×4, 96-d, LN [win. sz. 7×7, dim 96, head 3] × 2	concat 4×4, 96-d, LN [win. sz. 7×7, dim 96, head 3] × 2	concat 4×4, 128-d, LN [win. sz. 7×7, dim 128, head 4] × 2	concat 4×4, 192-d, LN [win. sz. 7×7, dim 192, head 6] × 2
stage 2	8× (28×28)	concat 2×2, 192-d , LN [win. sz. 7×7, dim 192, head 6] × 2	concat 2×2, 192-d , LN [win. sz. 7×7, dim 192, head 6] × 2	concat 2×2, 256-d , LN [win. sz. 7×7, dim 256, head 8] × 2	concat 2×2, 384-d , LN [win. sz. 7×7, dim 384, head 12] × 2
stage 3	16× (14×14)	concat 2×2, 384-d , LN [win. sz. 7×7, dim 384, head 12] × 6	concat 2×2, 384-d , LN [win. sz. 7×7, dim 384, head 12] × 18	concat 2×2, 512-d , LN [win. sz. 7×7, dim 512, head 16] × 18	concat 2×2, 768-d , LN [win. sz. 7×7, dim 768, head 24] × 18
stage 4	32× (7×7)	concat 2×2, 768-d , LN [win. sz. 7×7, dim 768, head 24] × 2	concat 2×2, 768-d , LN [win. sz. 7×7, dim 768, head 24] × 2	concat 2×2, 1024-d , LN [win. sz. 7×7, dim 1024, head 32] × 2	concat 2×2, 1536-d , LN [win. sz. 7×7, dim 1536, head 48] × 2

Table 7. Detailed architecture specifications.



ConvNeXt

Inverted bottleneck

作者认为Transformer block中的MLP模块非常像MobileNetV2中的Inverted Bottleneck模块，即两头细中间粗。

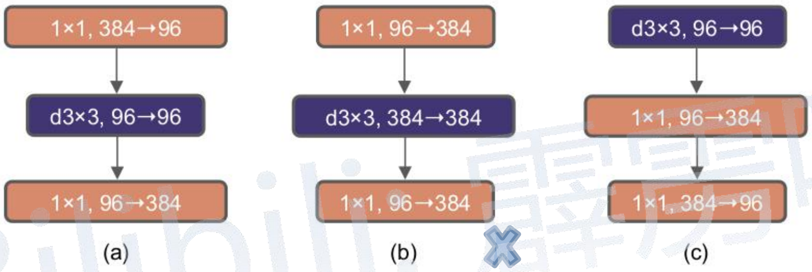
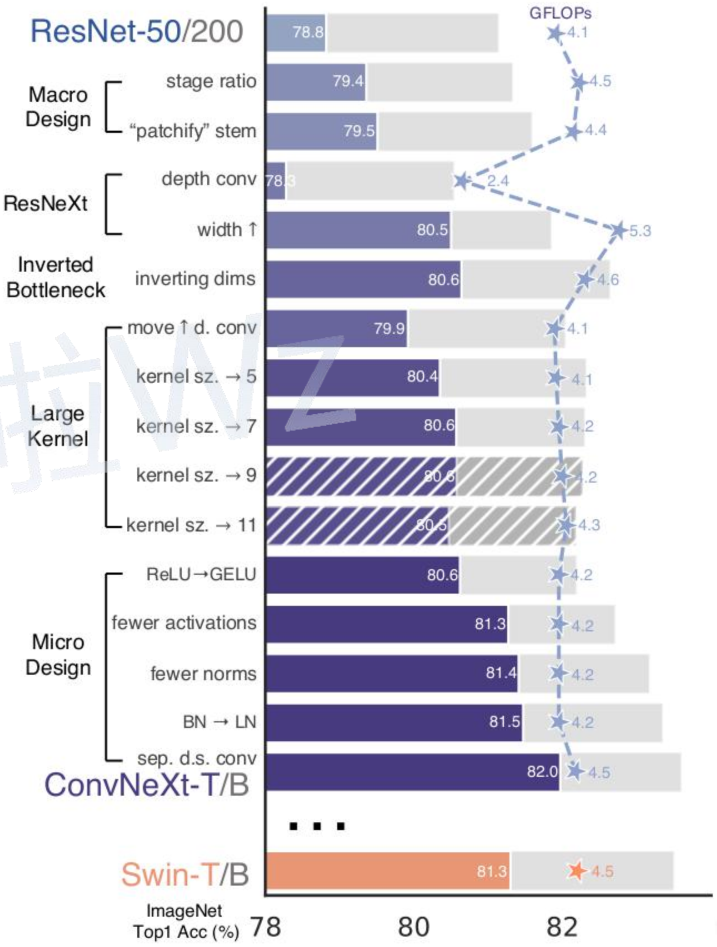


Figure 3. **Block modifications and resulted specifications. (a)** is a ResNeXt block; in **(b)** we create an inverted bottleneck block and in **(c)** the position of the spatial depthwise conv layer is moved up.

在较小的模型上准确率由80.5%提升到了80.6%
在较大的模型上准确率由81.9%提升到82.6%



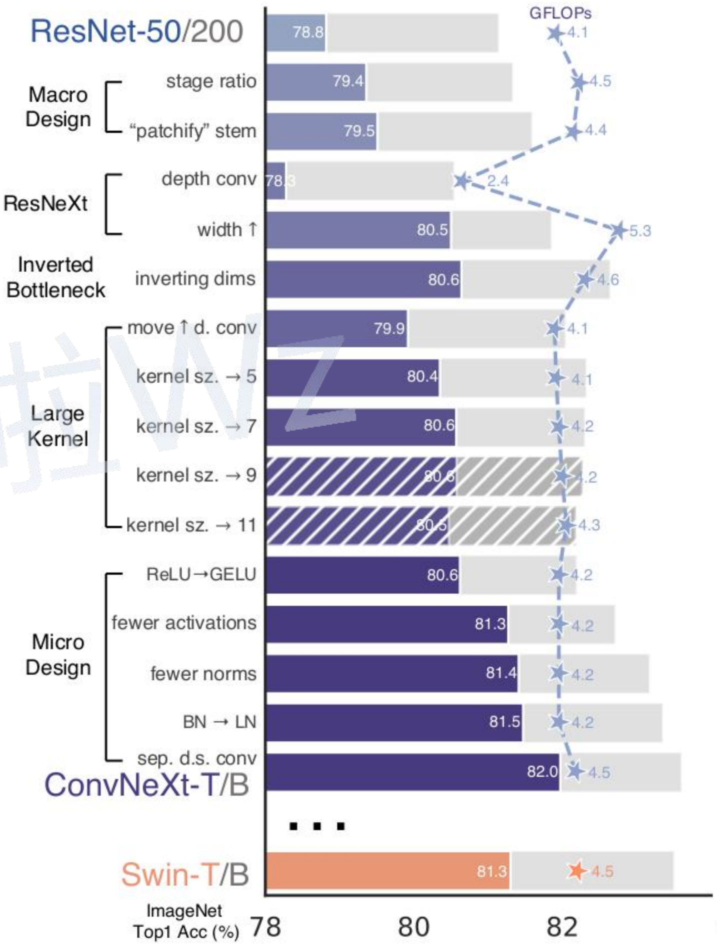
ConvNeXt

Large kerner size

Moving up depthwise conv layer，将depthwise conv模块上移
原来是　　1x1 conv -> depthwise conv -> 1x1 conv
现在变成　depthwise conv -> 1x1 conv -> 1x1 conv

Increasing the kernel size，将depthwise conv的卷积核大小由3x3改成了7x7

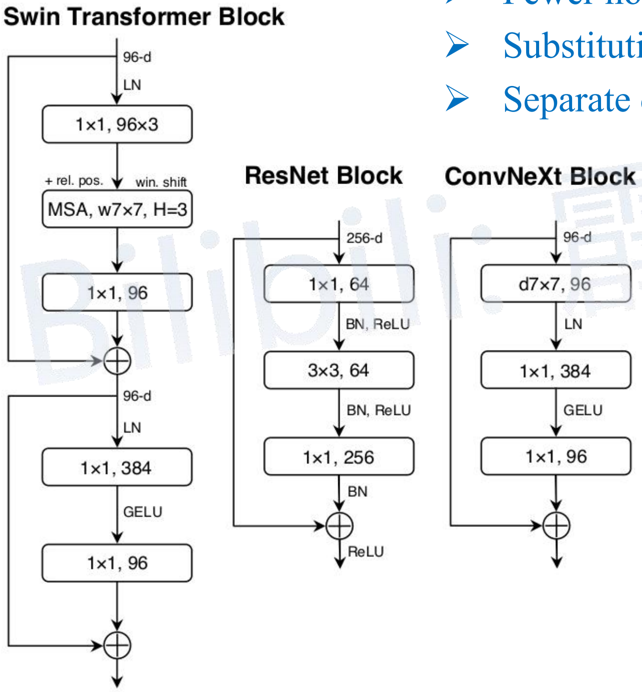
model	IN-1K acc.	GFLOPs
ResNet-50 (PyTorch [1])	76.13	4.09
ResNet-50 (enhanced recipe)	78.82 ± 0.07	4.09
stage ratio	79.36 ± 0.07	4.53
"patchify" stem	79.51 ± 0.18	4.42
depthwise conv	78.28 ± 0.08	2.35
increase width	80.50 ± 0.02	5.27
inverting dimensions	80.64 ± 0.03	4.64
move up depthwise conv	79.92 ± 0.08	4.07
kernel size → 5	80.35 ± 0.08	4.10
kernel size → 7	80.57 ± 0.14	4.15
kernel size → 9	80.57 ± 0.06	4.21
kernel size → 11	80.47 ± 0.11	4.29
ReLU → GELU	80.62 ± 0.14	4.15
fewer activations	81.27 ± 0.06	4.15
fewer norms	81.41 ± 0.09	4.15
BN → LN	81.47 ± 0.09	4.46
separate d.s. conv (ConvNeXt-T)	81.97 ± 0.06	4.49
Swin-T [42]	81.30	4.50

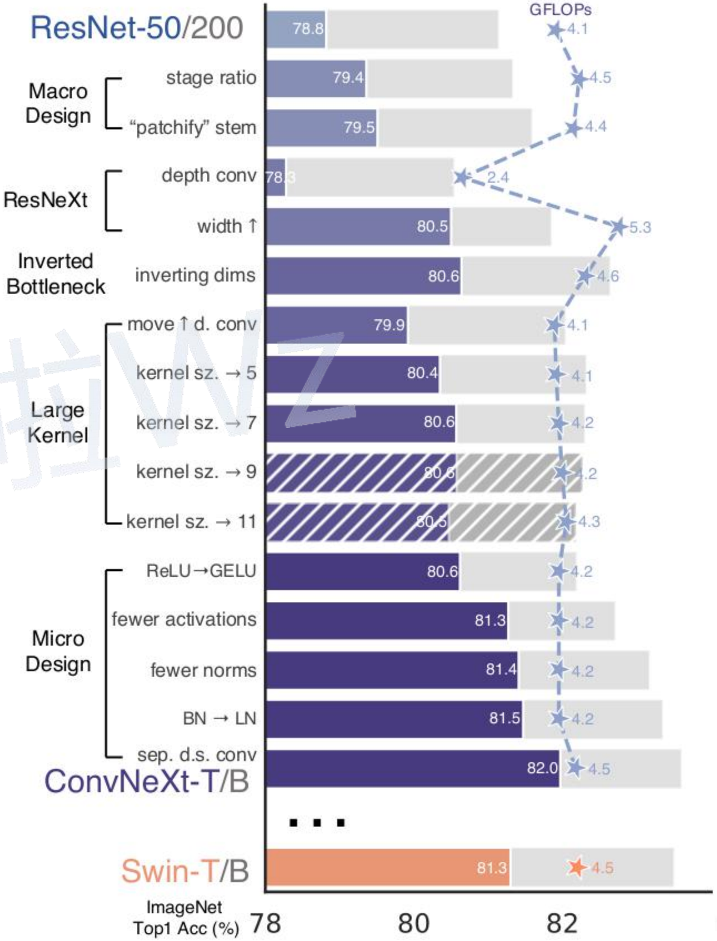


ConvNeXt

Micro designs

- Replacing ReLU with GELU
- Fewer activation functions
- Fewer normalization layers
- Substituting BN with LN
- Separate downsampling layers

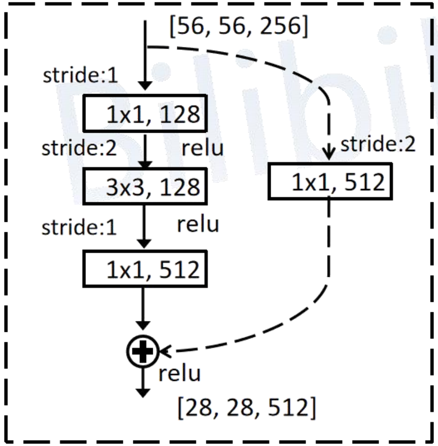


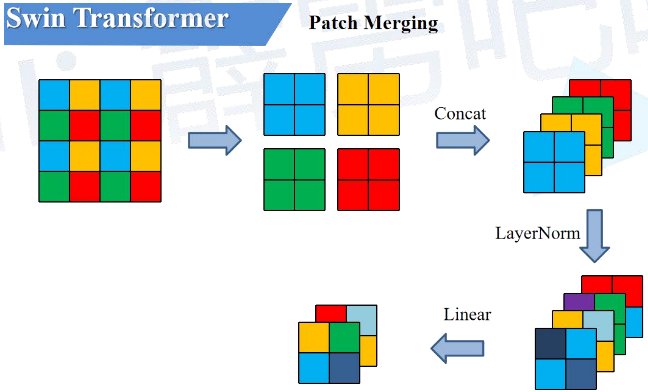


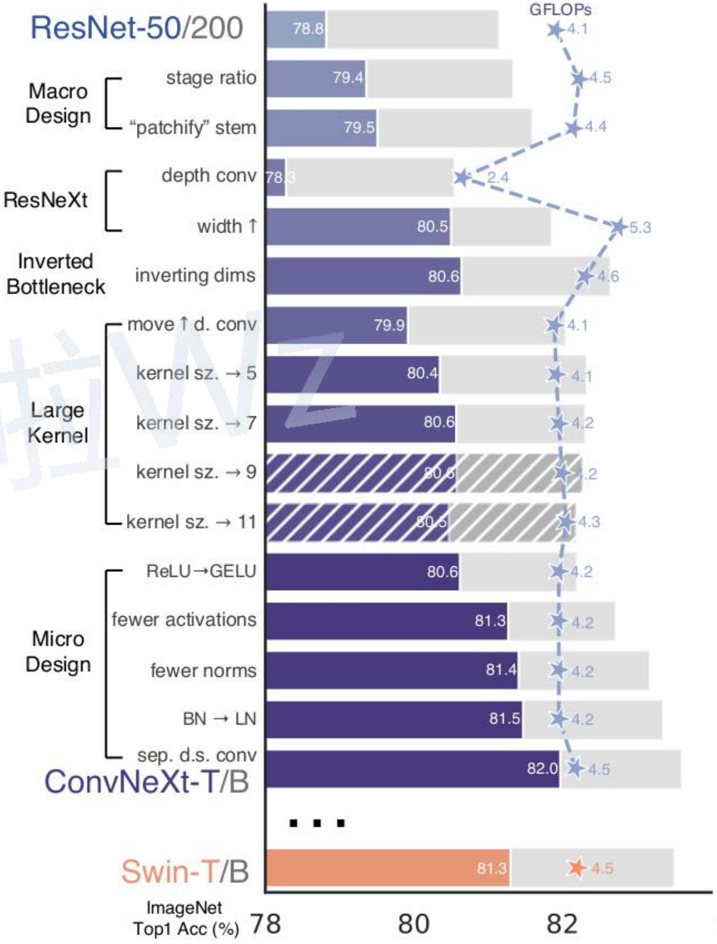
ConvNeXt

Micro designs

- Replacing ReLU with GELU
- Fewer activation functions
- Fewer normalization layers
- Substituting BN with LN
- Separate downsampling layers







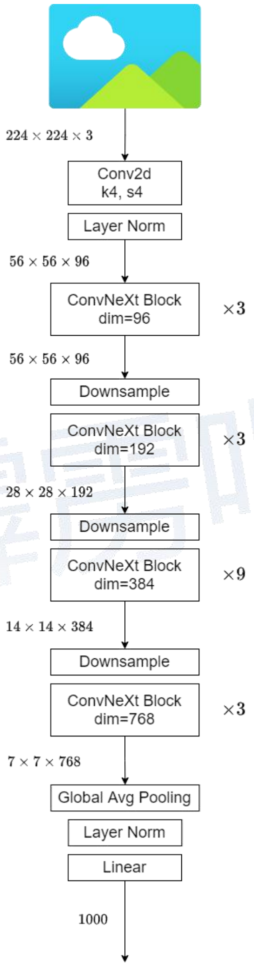
ConvNeXt

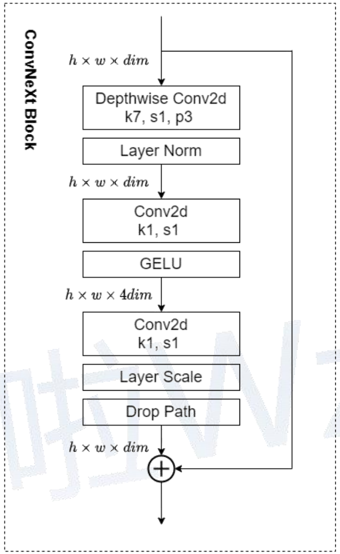
- ConvNeXt-T: C = (96, 192, 384, 768), B = (3, 3, 9, 3)
- ConvNeXt-S: C = (96, 192, 384, 768), B = (3, 3, 27, 3)
- ConvNeXt-B: C = (128, 256, 512, 1024), B = (3, 3, 27, 3)
- ConvNeXt-L: C = (192, 384, 768, 1536), B = (3, 3, 27, 3)
- ConvNeXt-XL: C = (256, 512, 1024, 2048), B = (3, 3, 27, 3)

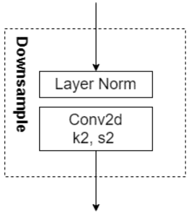
• ConvNeXt-T
4×4, 96, stride 4
$\begin{bmatrix} \text{d}7\times7, 96 \\ 1\times1, 384 \\ 1\times1, 96 \end{bmatrix} \times 3$
$\begin{bmatrix} \text{d}7\times7, 192 \\ 1\times1, 768 \\ 1\times1, 192 \end{bmatrix} \times 3$
$\begin{bmatrix} \text{d}7\times7, 384 \\ 1\times1, 1536 \\ 1\times1, 384 \end{bmatrix} \times 9$
$\begin{bmatrix} \text{d}7\times7, 768 \\ 1\times1, 3072 \\ 1\times1, 768 \end{bmatrix} \times 3$

ConvNeXt

ConvNeXt-T 结构图







沟通方式

1.github

https://github.com/WZMIAOMIAO/deep-learning-for-image-processing

2.bilibili

https://space.bilibili.com/18161609/channel/index

3.CSDN

https://blog.csdn.net/qq_37541097/article/details/103482003