A ConvNet for the 2020s

Zhuang Liu, Hanzi Mao, Christoph Feichtenhofer , Chao-Yuan Wu, Trevor Darrell, Saining Xie
Facebook AI Research, UC Berkeley
CVPR 2022

Slides by: Zhuang Liu

An Evolution of ConvNets

AlexNet - “ImageNet moment” (2012)

VGGNet - stacking 3x3 layers (2014)

Inceptions -- parallel branches (2014)

ResNet - identity shortcuts (2015)

ResNeXt – grouped convolution (2016)

DenseNet - dense shortcut connection (2016)

MobileNets - depthwise conv; inverted residuals (2017/18)

EfficientNet – model scaling (2019)

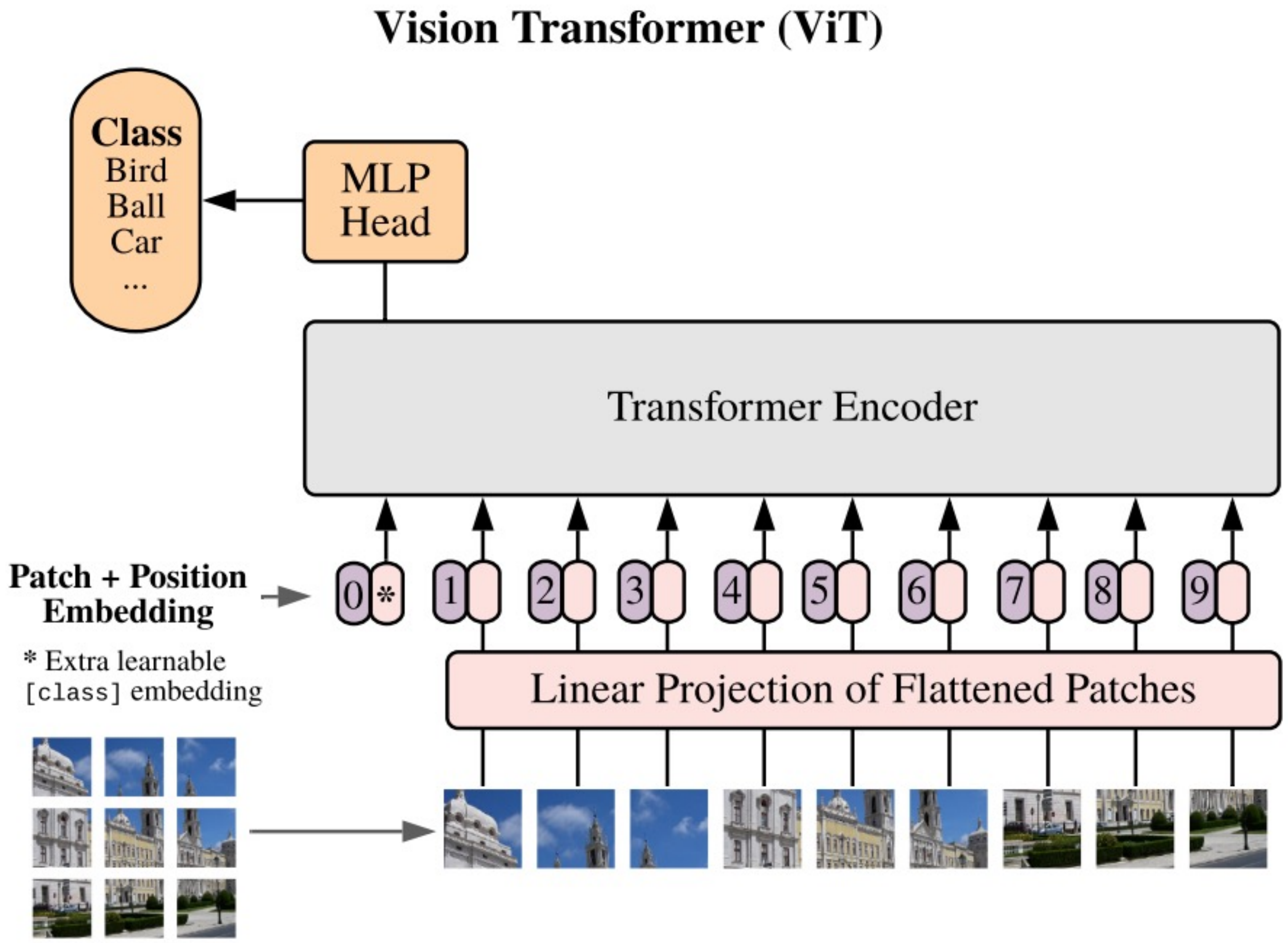
....

Behind the Success

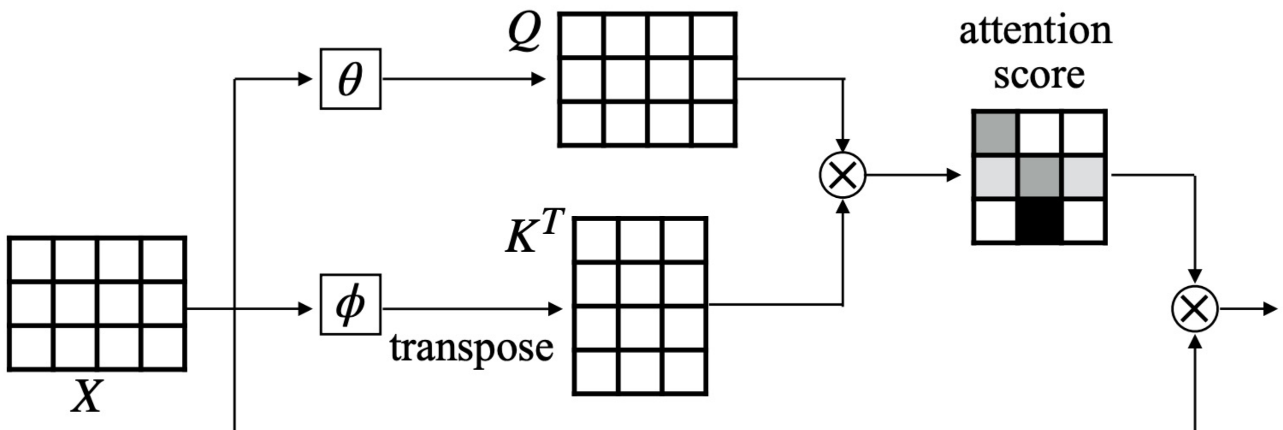
- Local Computation
- Translation Equivariance
- Feature Hierarchy

A Step Change from Vision Transformers

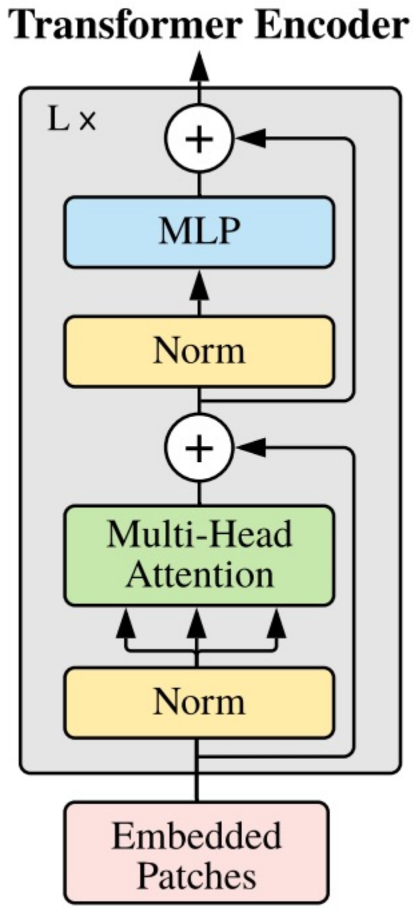
- NLP: RNN -> Transformers since 2017
- CV: Vision Transformer (ViT) emerged in 2020



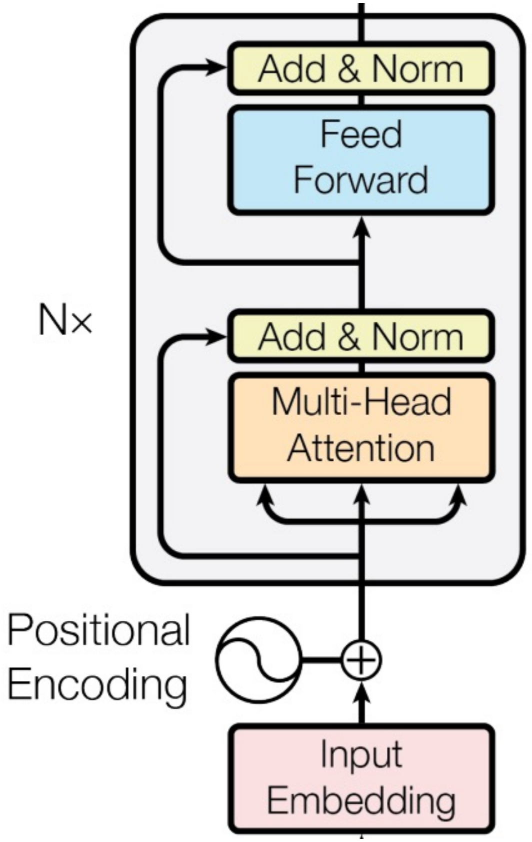
Self-Attention: Transformers' core module



Self-attention mechanism



Vision Transformer block



NLP Transformer block

Image: https://blogs.oracle.com/ai-and-datascience/post/multi-head-self-attention-in-nlp

Vanilla ViT’s Challenges

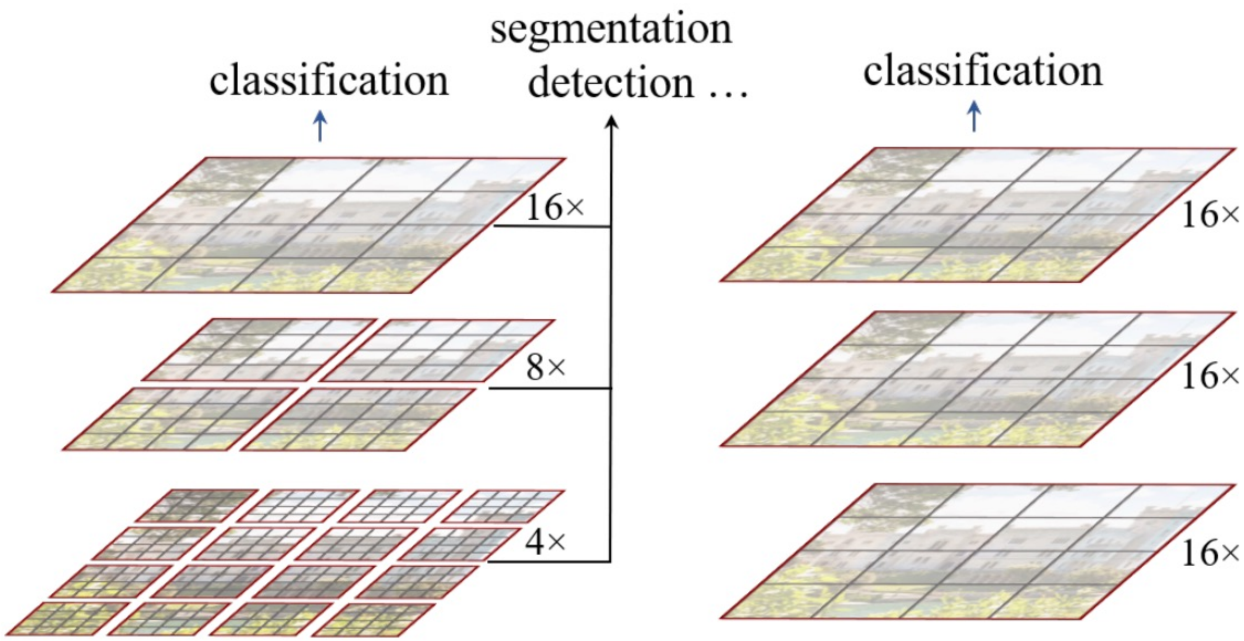
- ViT’s success was limited to image classification
  - but computer vision is not

- Global attention has quadratic complexity w.r.t. input size
  - compute becomes intractable with higher-resolution images

Hierarchical vision Transformers – Bringing back ConvNet priors

- Attention within local window
- Shared weights between windows
- Feature hierarchy

- SOTA across vision benchmarks
- ConvNet priors still much desired



(a) Swin Transformer (ours)
(b) ViT

But more complexity

- Naïve implementation for sliding window attention --> expensive
- Advanced techniques (e.g., cyclic shifting) --> complicated
- ConvNets have the desired properties already!

ConvNet losing steam?

- Appearances in paper titles at CV conferences

Query	Convolution, CNN, ConvNet	Attention, “-Former”
ECCV 2020	56	54
CVPR 2021	49	78
ICCV 2021	44	176
CVPR 2022	?	?

The only reason seems to be …

Swin Transformer

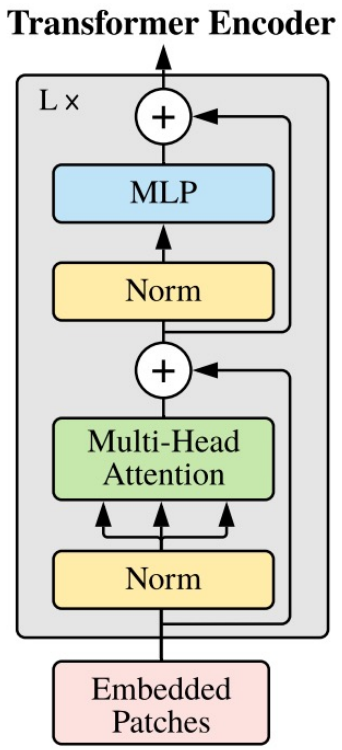
- State of the Art | Object Detection on COCO test-dev (using additional training data)
- State of the Art | Instance Segmentation on COCO test-dev
- State of the Art | Semantic Segmentation on ADE20K (using additional training data)
- Ranked #4 | Action Classification on Kinetics-400 (using additional training data)

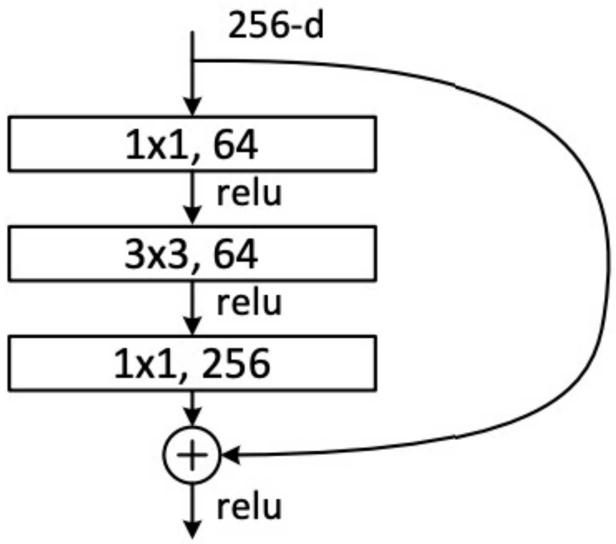
Credit is given to the “Transformer” part, but not the hidden “ConvNet” part

Vision Transformers and ConvNets

- Similar: ConvNet inductive biases
- Different
  - supposedly "core" component (attention vs. conv)
  - training procedures
  - subtle architectural designs

- Too early to give credit to self-attention?





In this work

- Identify the confounding variables
- Test the limits of what a pure ConvNet can achieve
- Level the playing field for ConvNets in post-ViT eras

To do this, we…

- Start with a simple standard ResNet
- “Modernize” the architecture towards a hierarchical vision Transformer
- Central question: How do design choices in Transformers impact ConvNet’s performance?

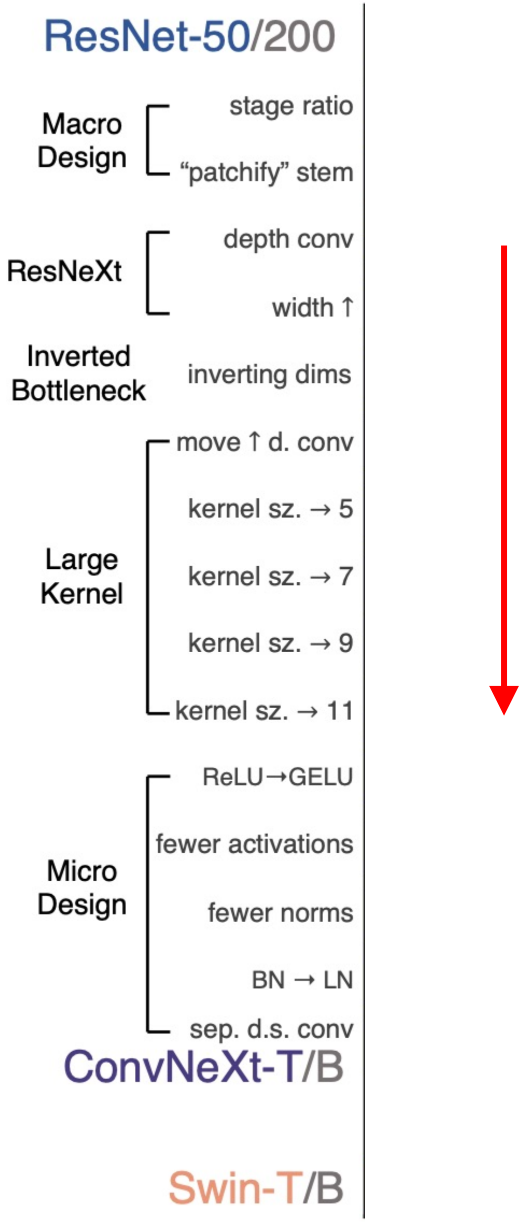
First step: change recipe

Typical Vision Transformer Training Recipe

Typical ResNet Training Recipe

ResNet-50 ImageNet top-1: 76.1% -> 78.7% 🚀

Next: a journey of Transformer-inspired architecture changes



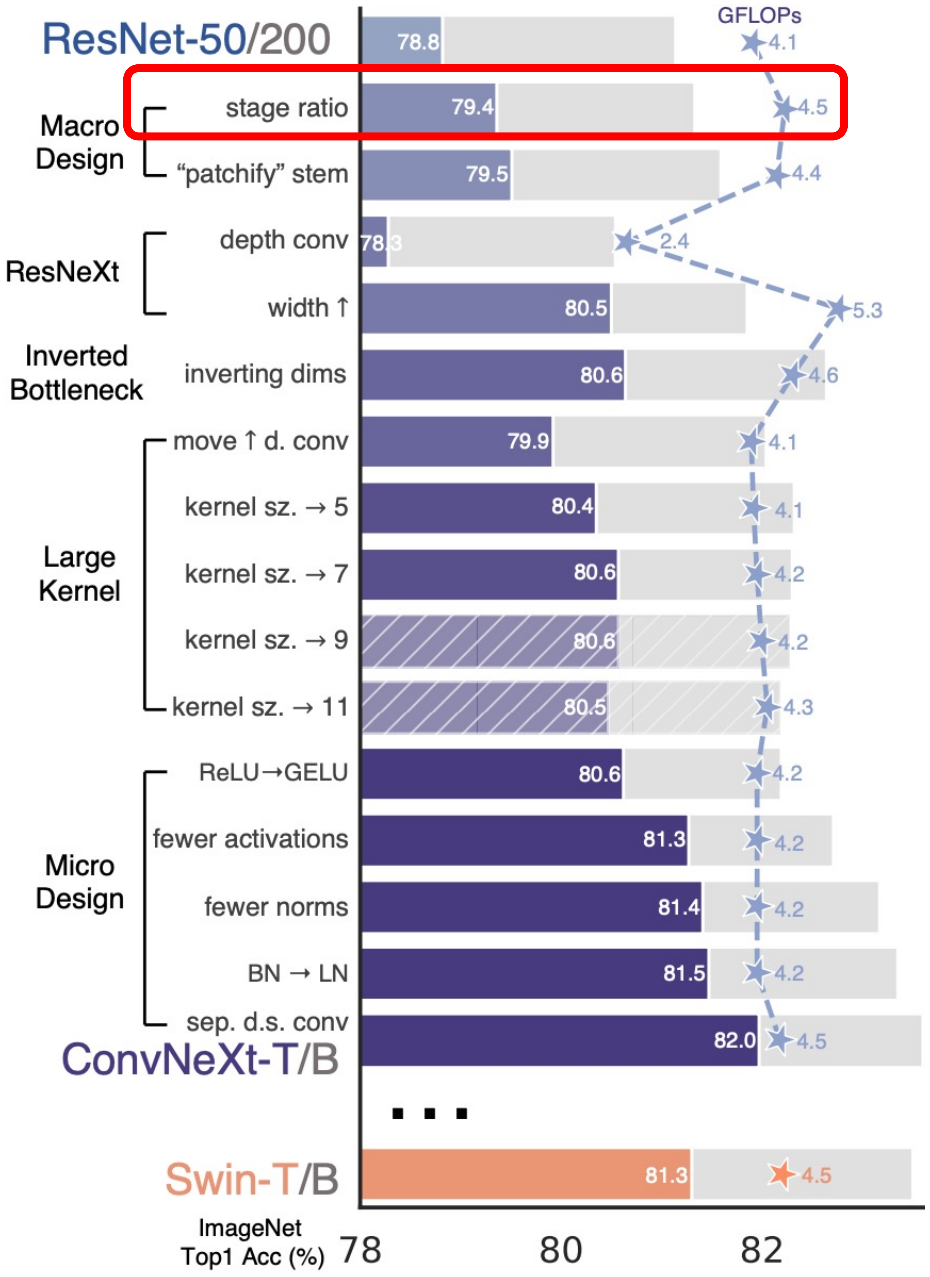
Macro Design

Align Stage Compute Ratio

ResNet-50: 3:4:6:3

Swin-T: 1:1:3:1

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



Macro Design

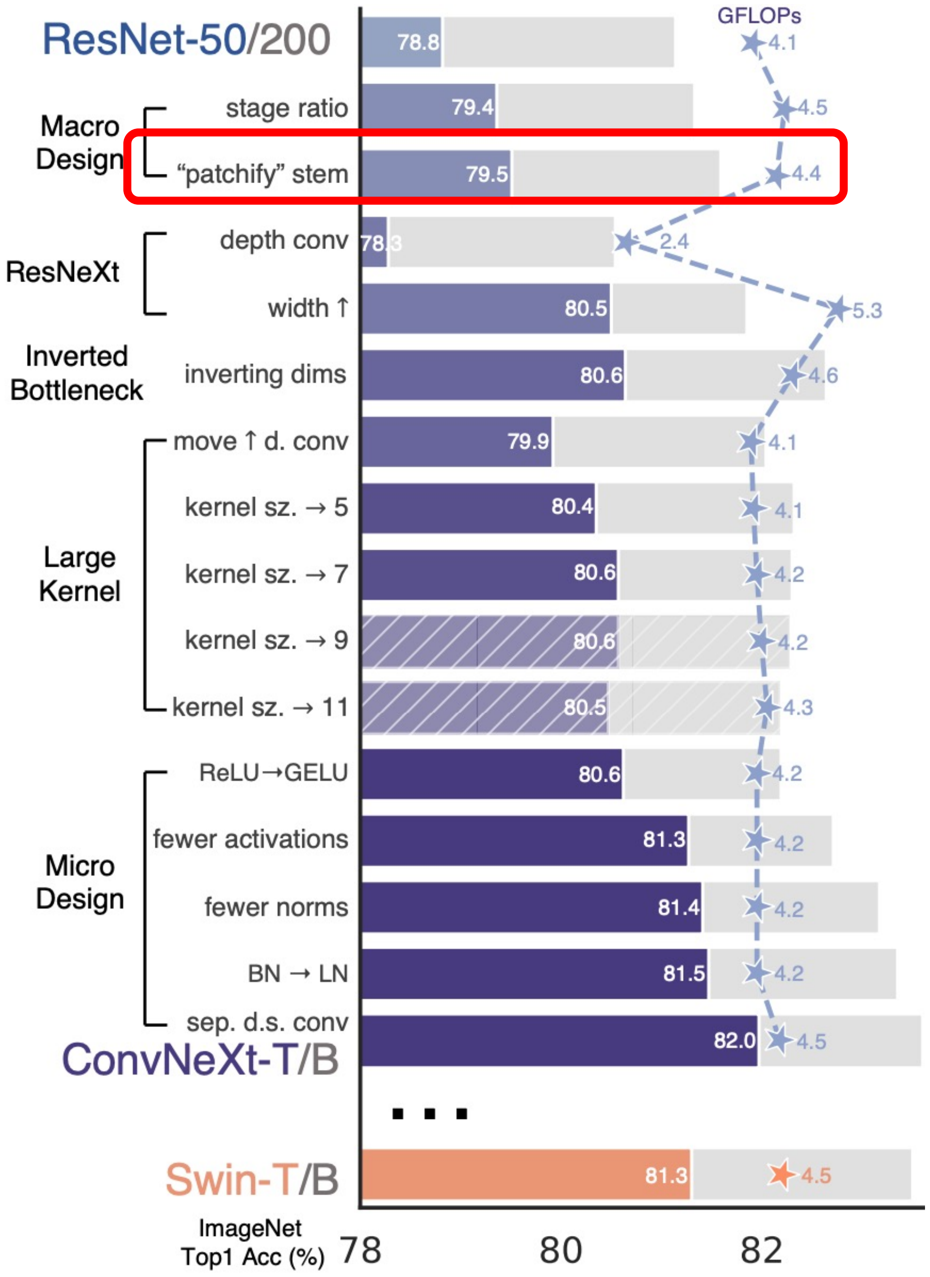
"Patchify" Stem

ResNet: 7x7 stride-2 conv, 3x3 stride-2 max pool

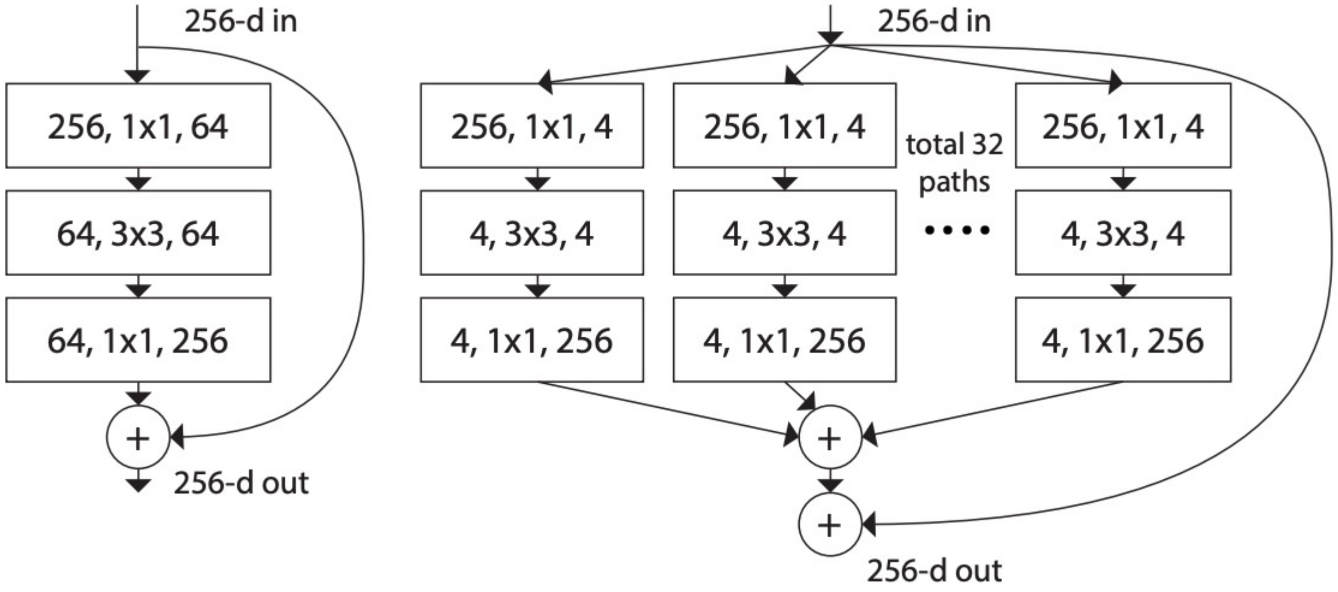
ViT: 16x16 stride-16 conv ("patch embedding")

Swin: 4x4 stride-4 conv

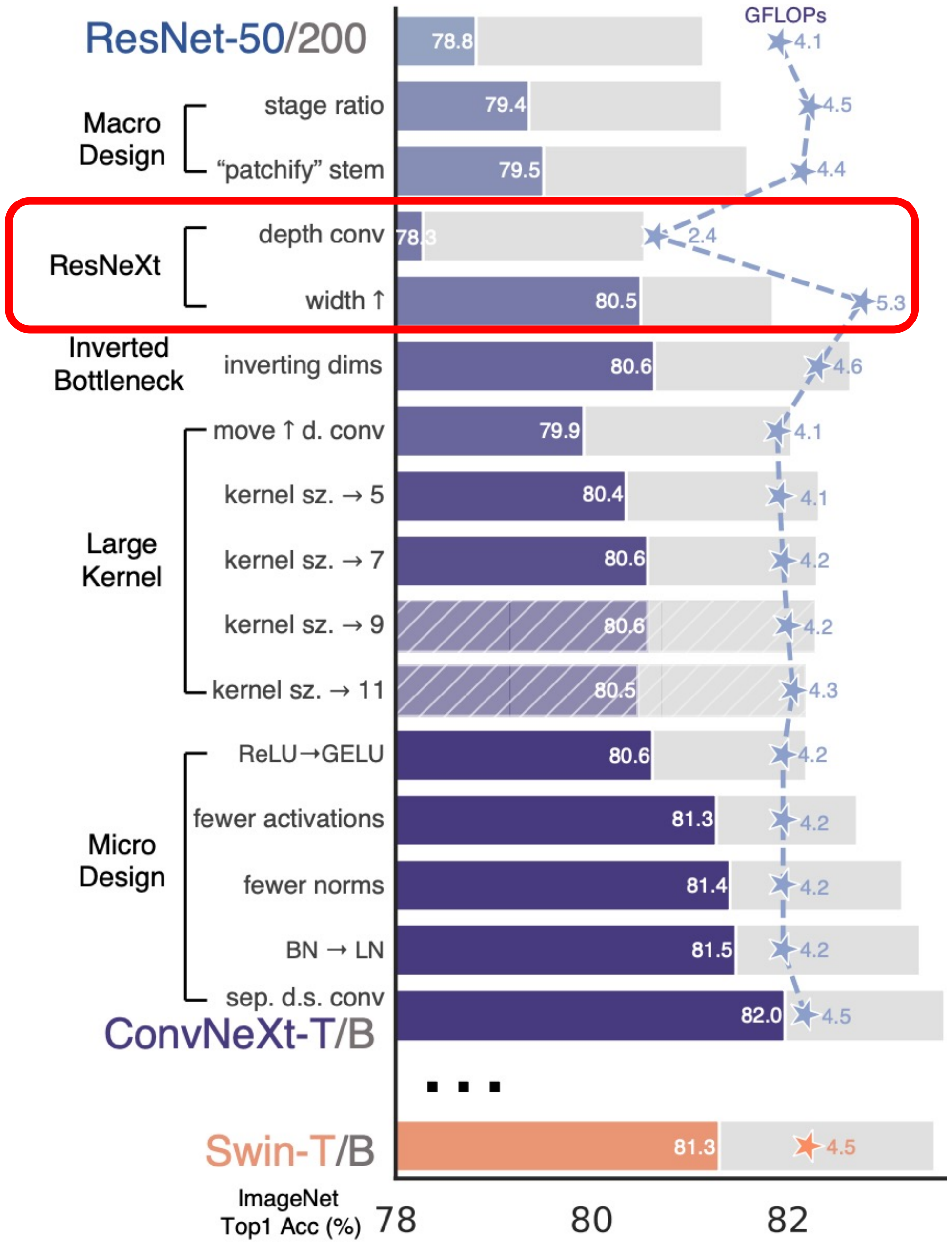
layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		[3×3, 64; 3×3, 64] ×2	[3×3, 64; 3×3, 64] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3
conv3_x	28×28	[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



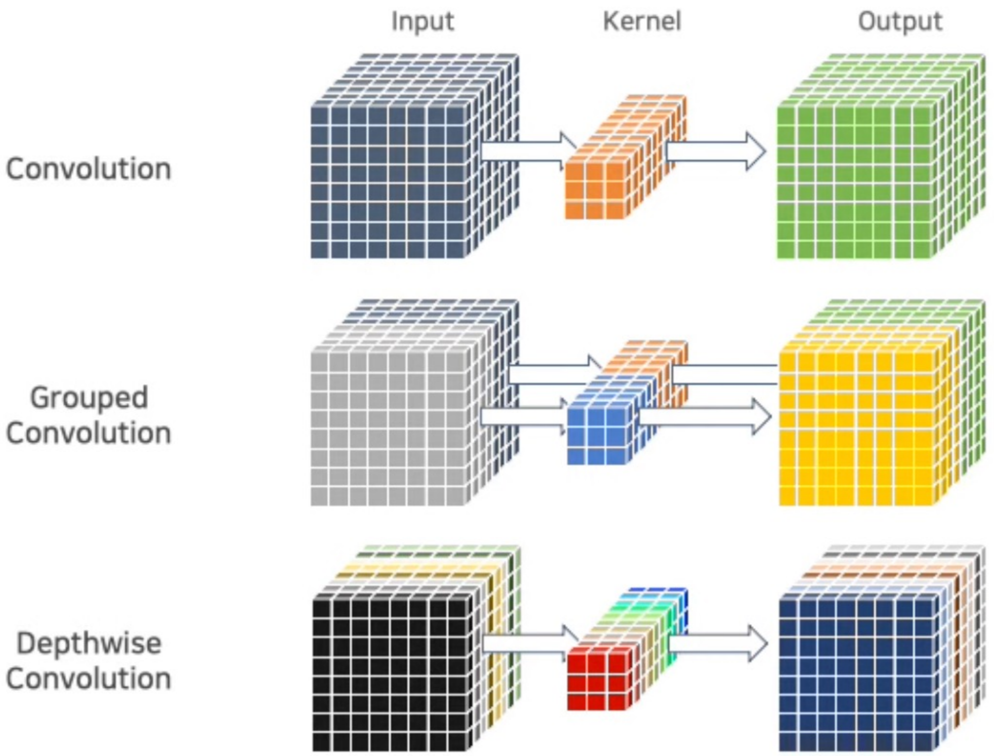
ResNeXt-ify

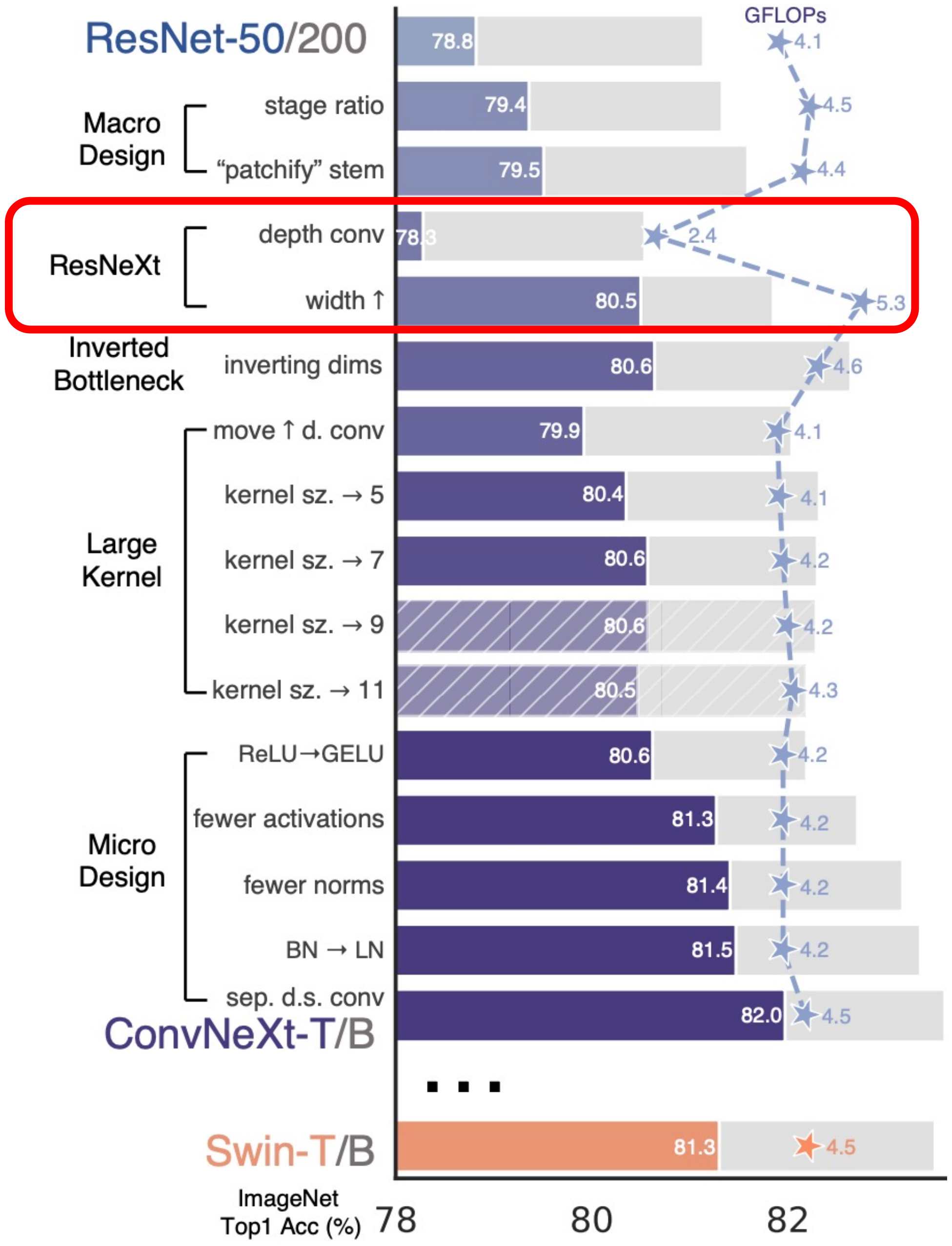


ResNet (dense convolution) vs. ResNeXt (grouped convolution)



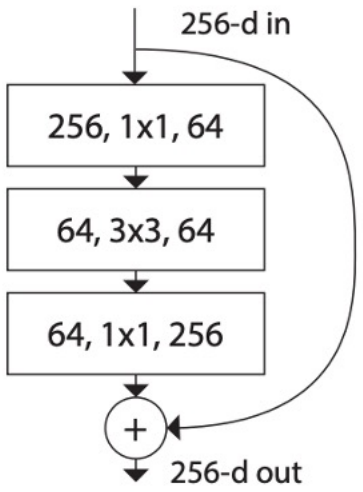
ResNeXt-ify

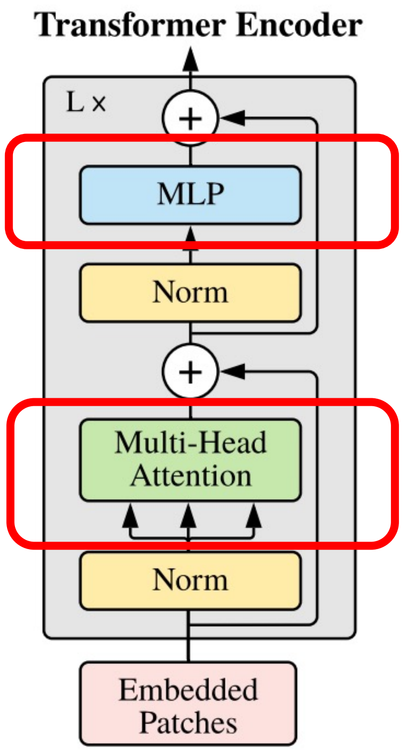


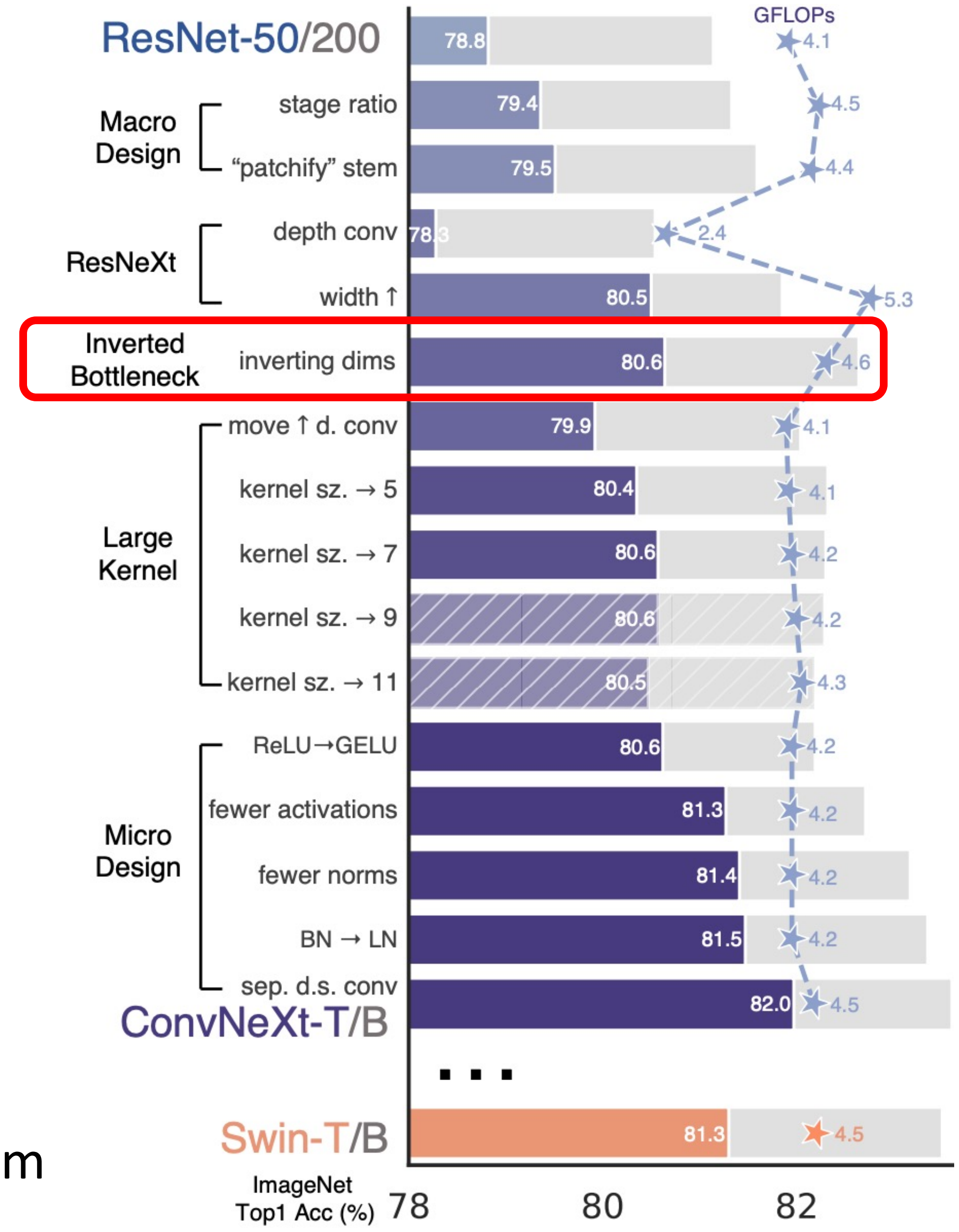


- Popularized by MobileNets and Xception
- Evident in the weighted-sum operation in Transformers, also in a per-channel basis
- Grow width (64->96) as well, aligning with Swin-T

Inverted Bottleneck





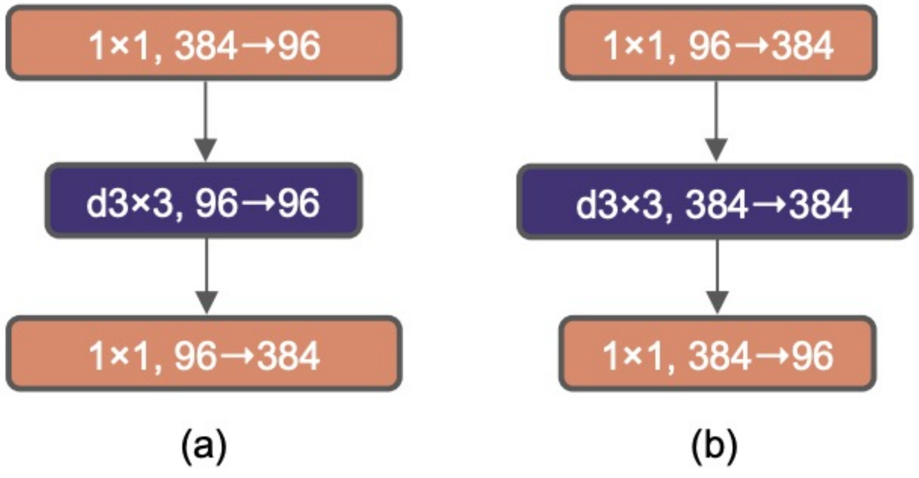


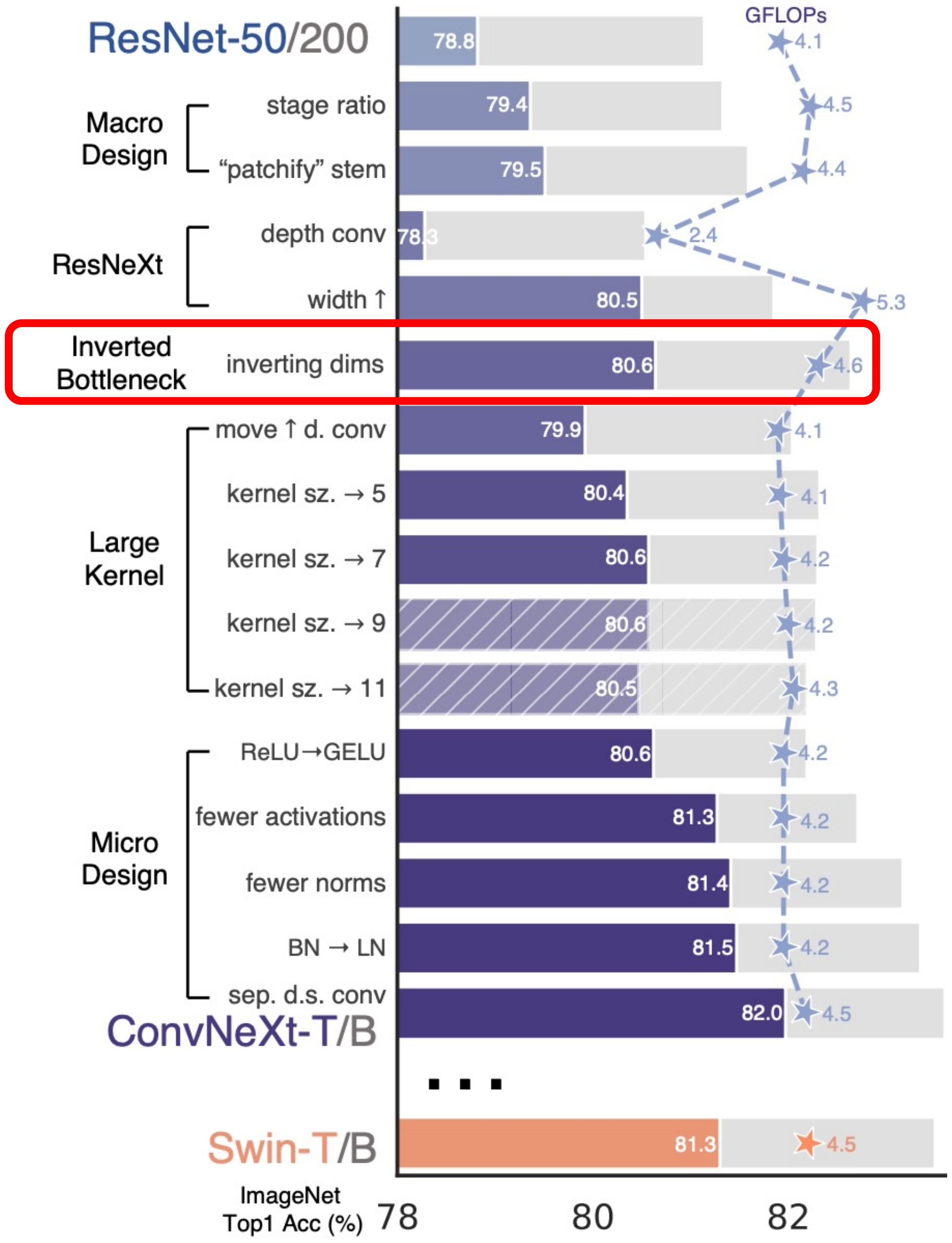
MSA – input & output: C dim; qkv: 3C dim

MLP – input & output: C dim; intermediate layer: 4C dim

Fat in the middle, thin at two sides

Inverted Bottleneck





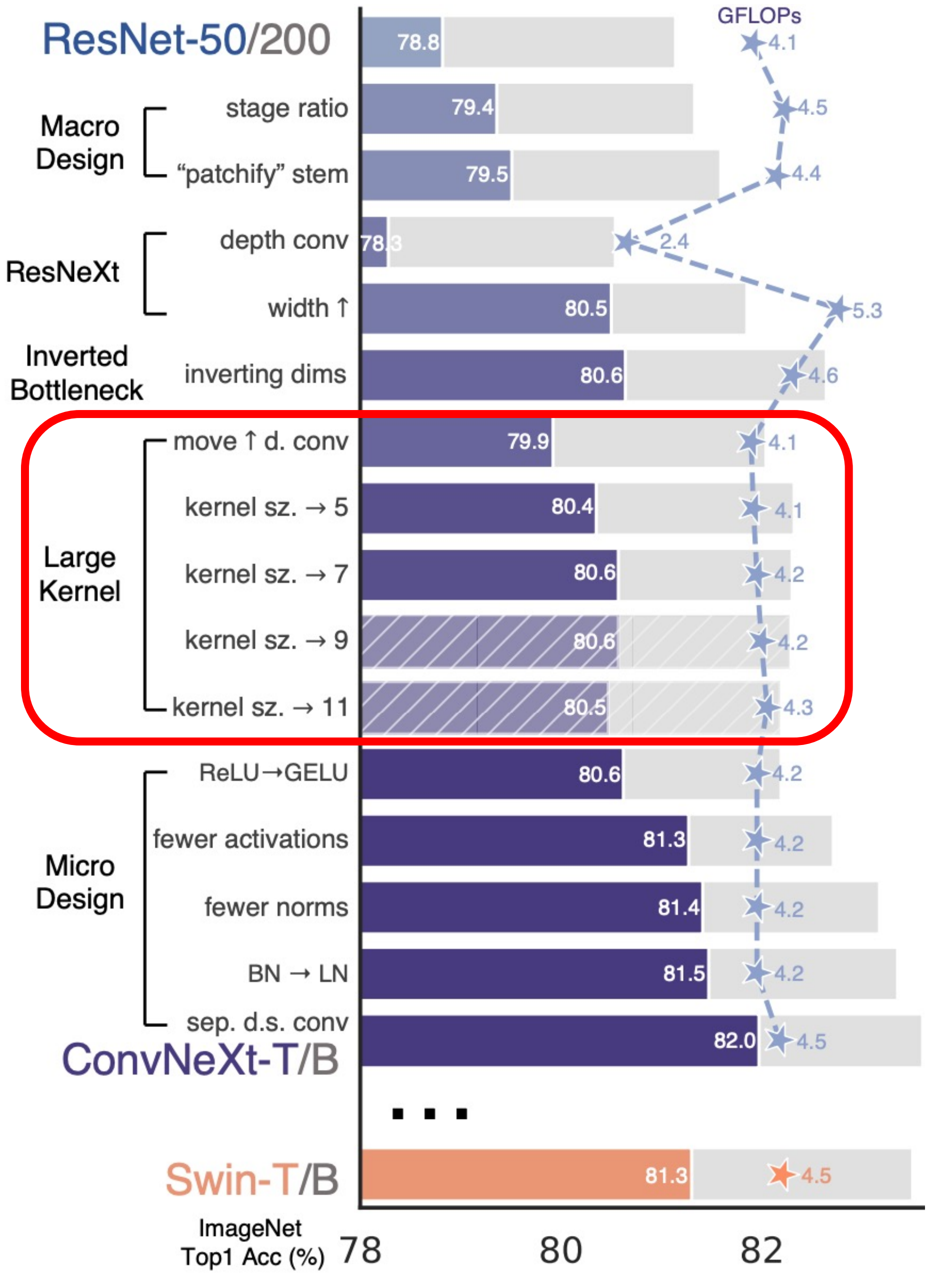
Left: Bottleneck proposed in original ResNets

Right: Inverted Bottleneck proposed in MobileNetV2

We use inverted bottlenecks

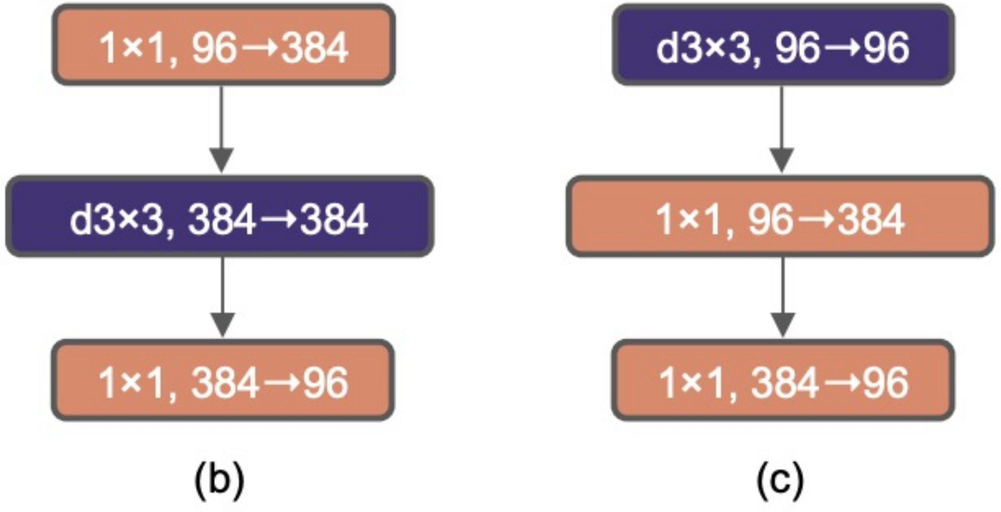
Large Kernels

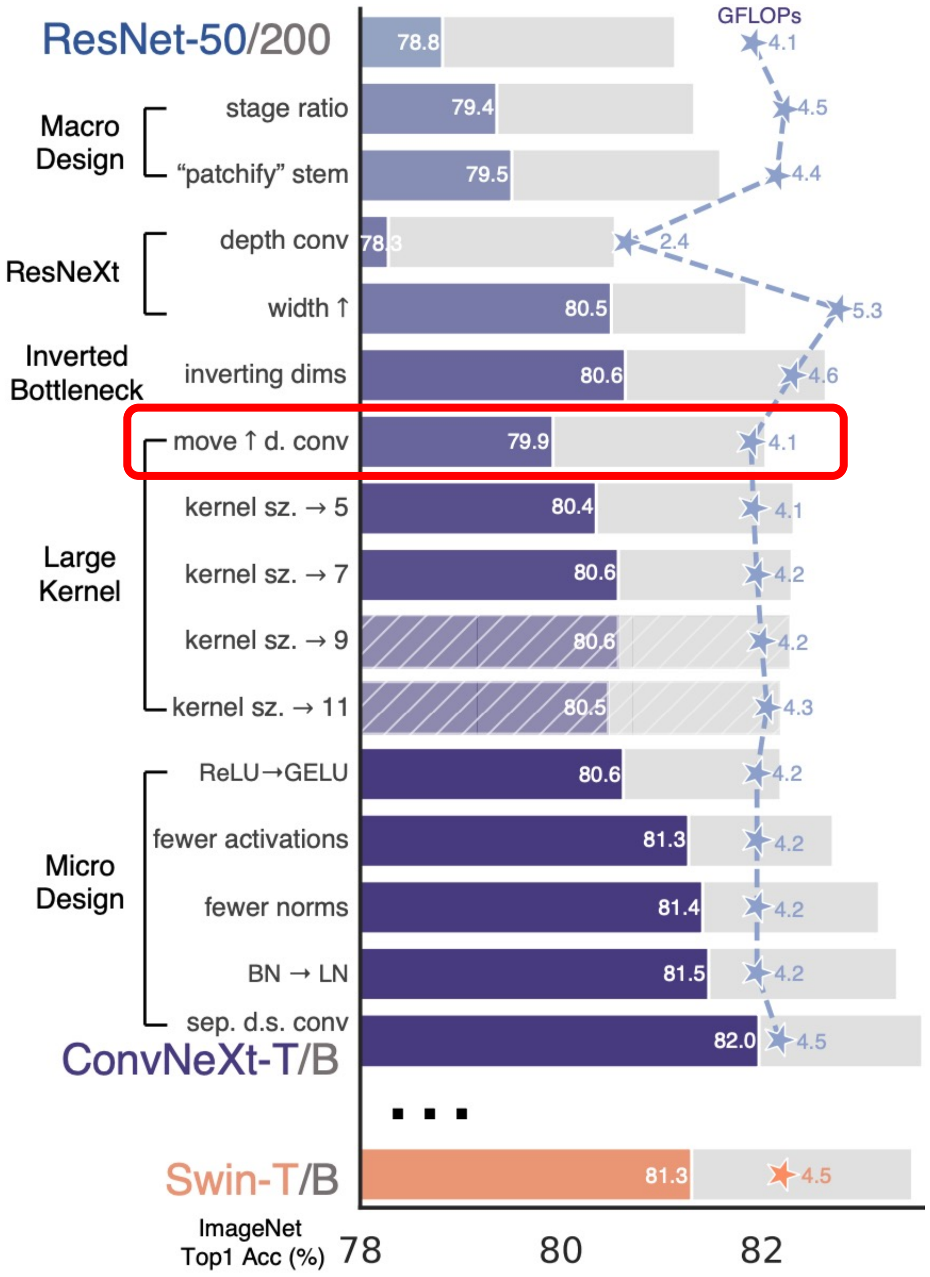
- ViT: global attention
- Swin: local attention, window size 7x7
- Your typical ConvNet: 3x3



Large Kernels

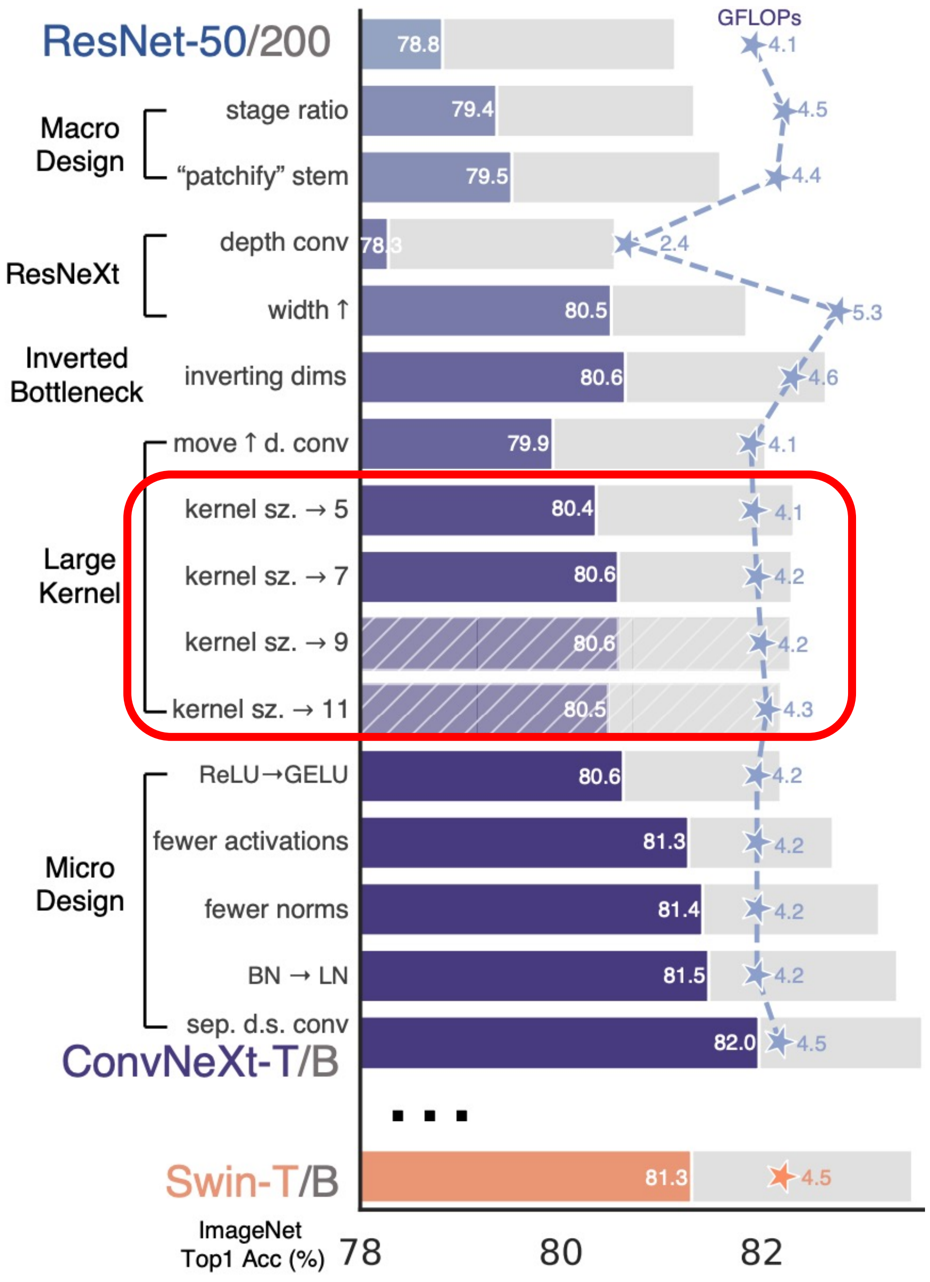
- The spatial mixing MSA is before the channel-mixing MLP in Transformer block
- We move d.w. conv up before using large ks
- Reduces FLOPs w/ large ks





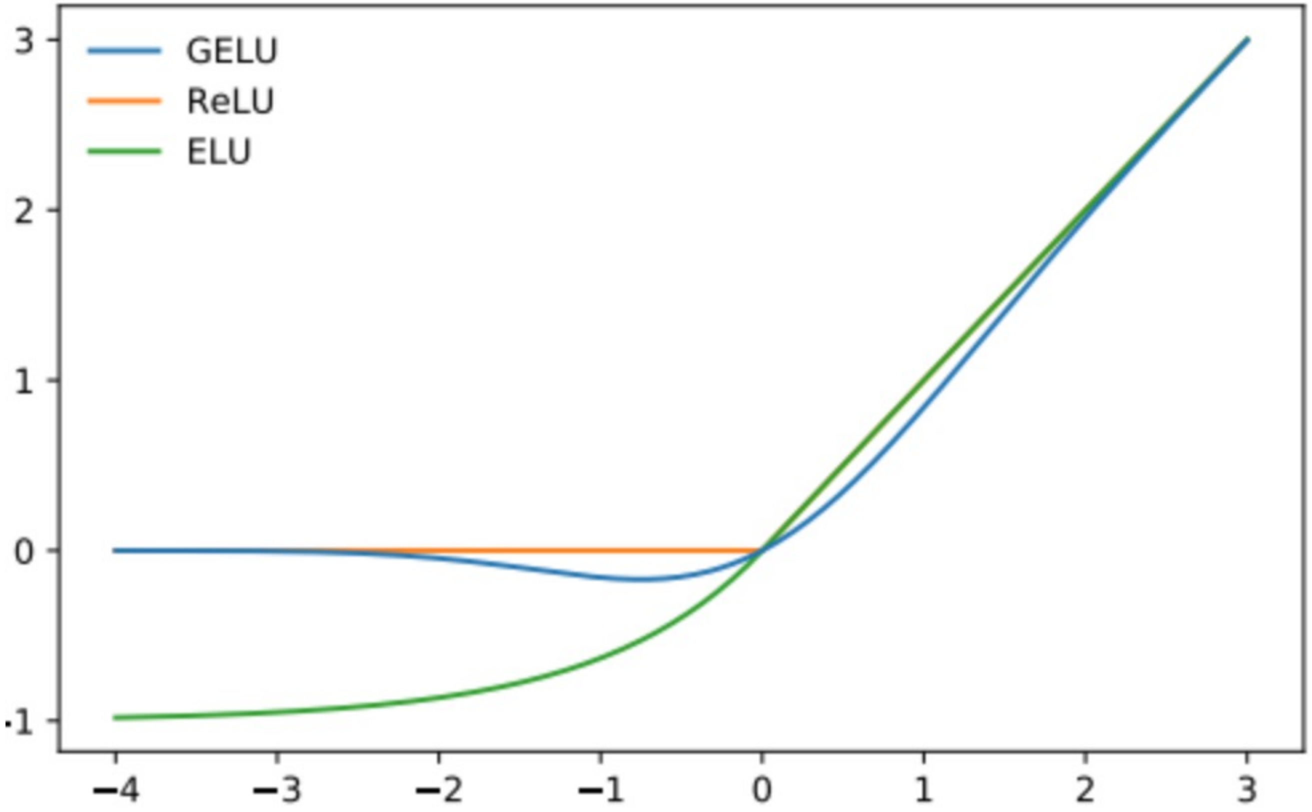
Large Kernels

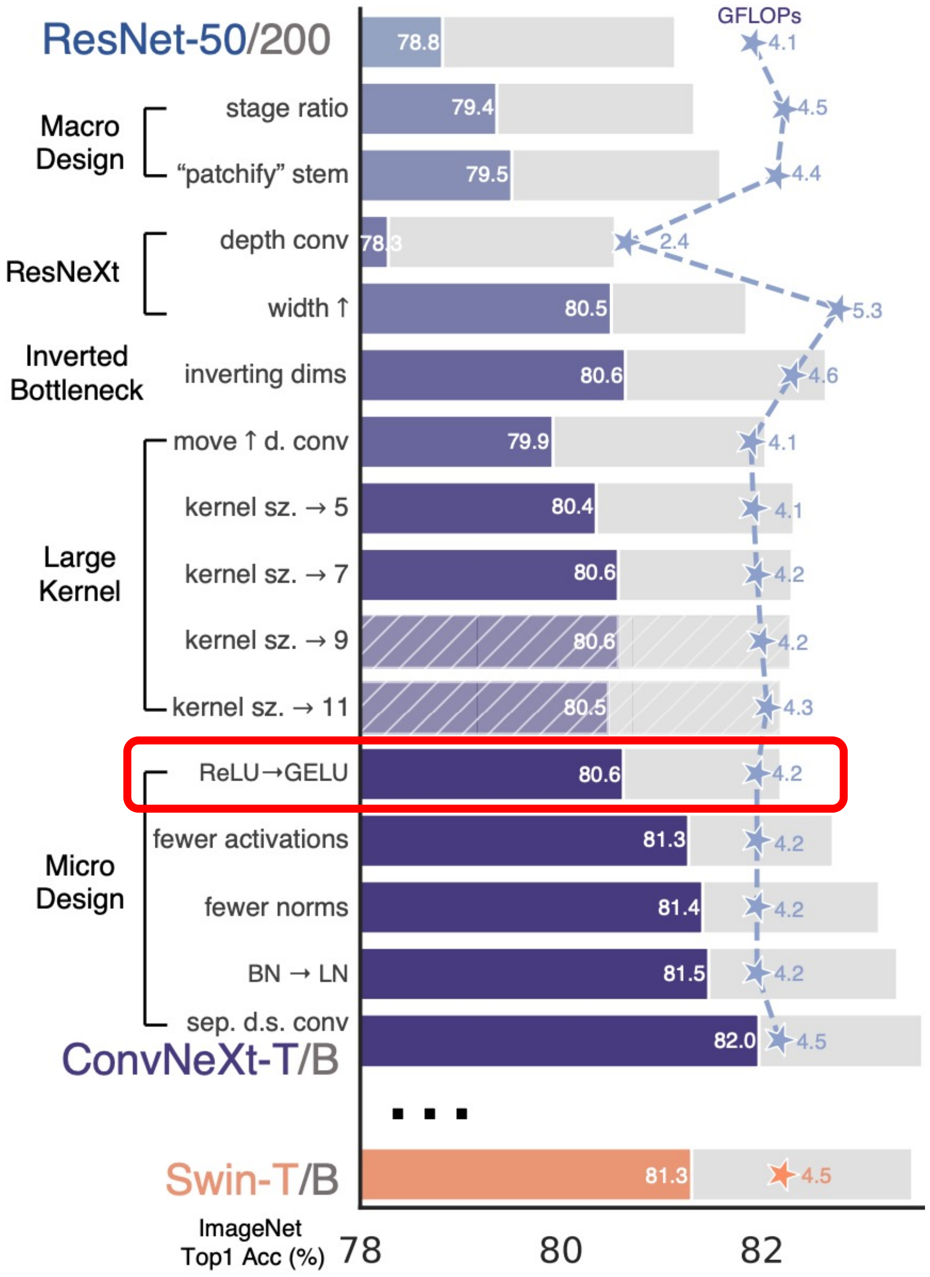
- Now we increase kernel size from 3 to 11
- The performance saturates at 7
- Swin's choice of local window size is also 7
- Note: the optimal kernel sizes depend on tasks and regularization strength



Micro Designs

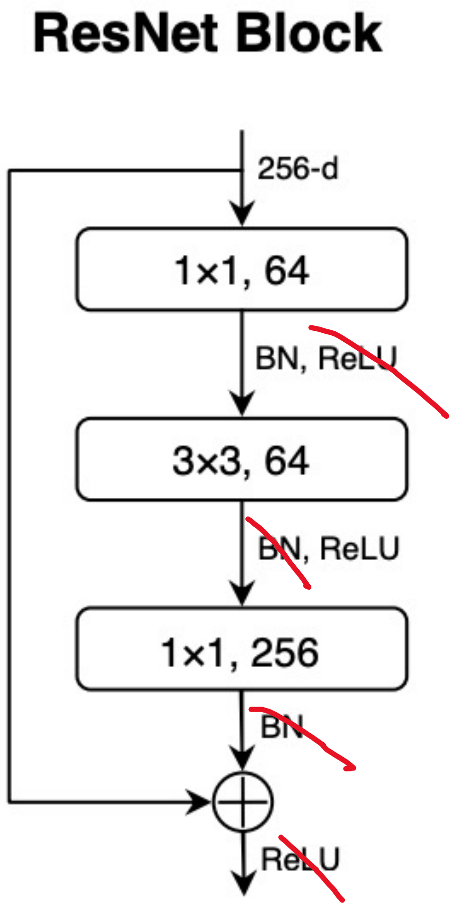
ReLU -> GELU (standard activations in Transformers)

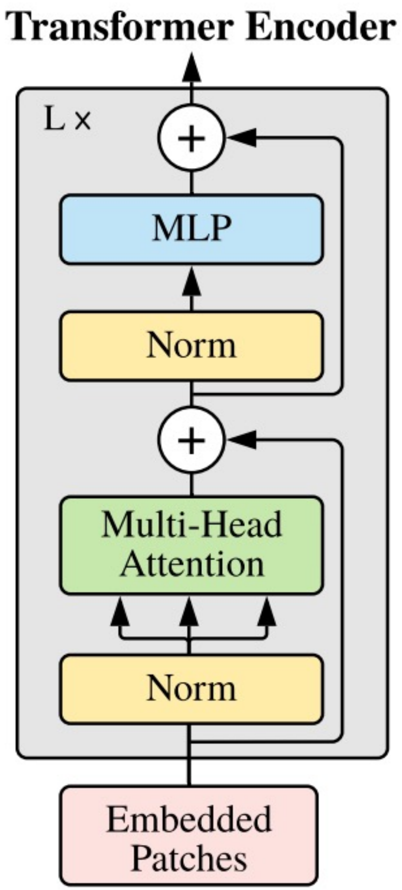


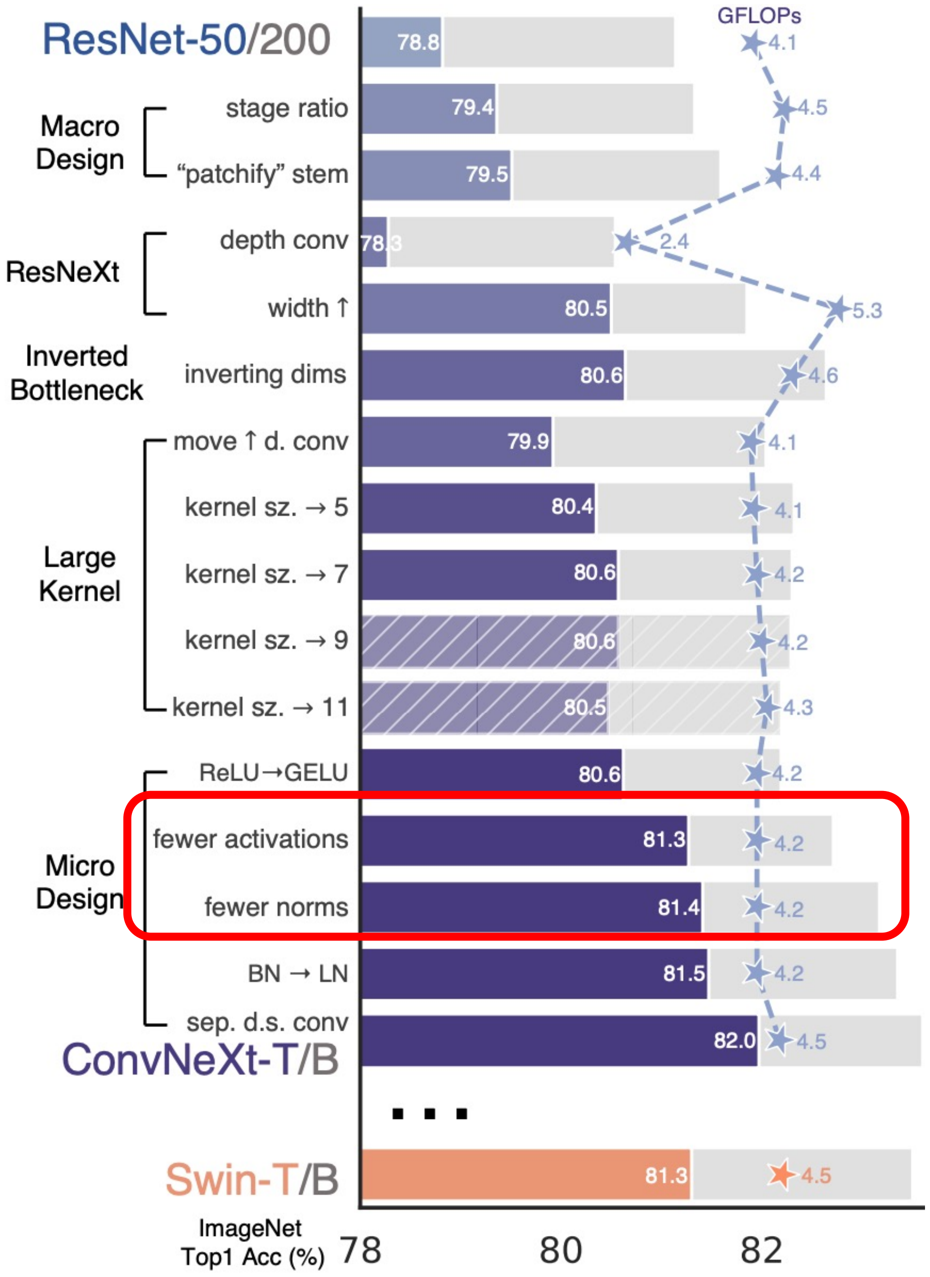


Micro Designs

Aggressively removing acts & norms





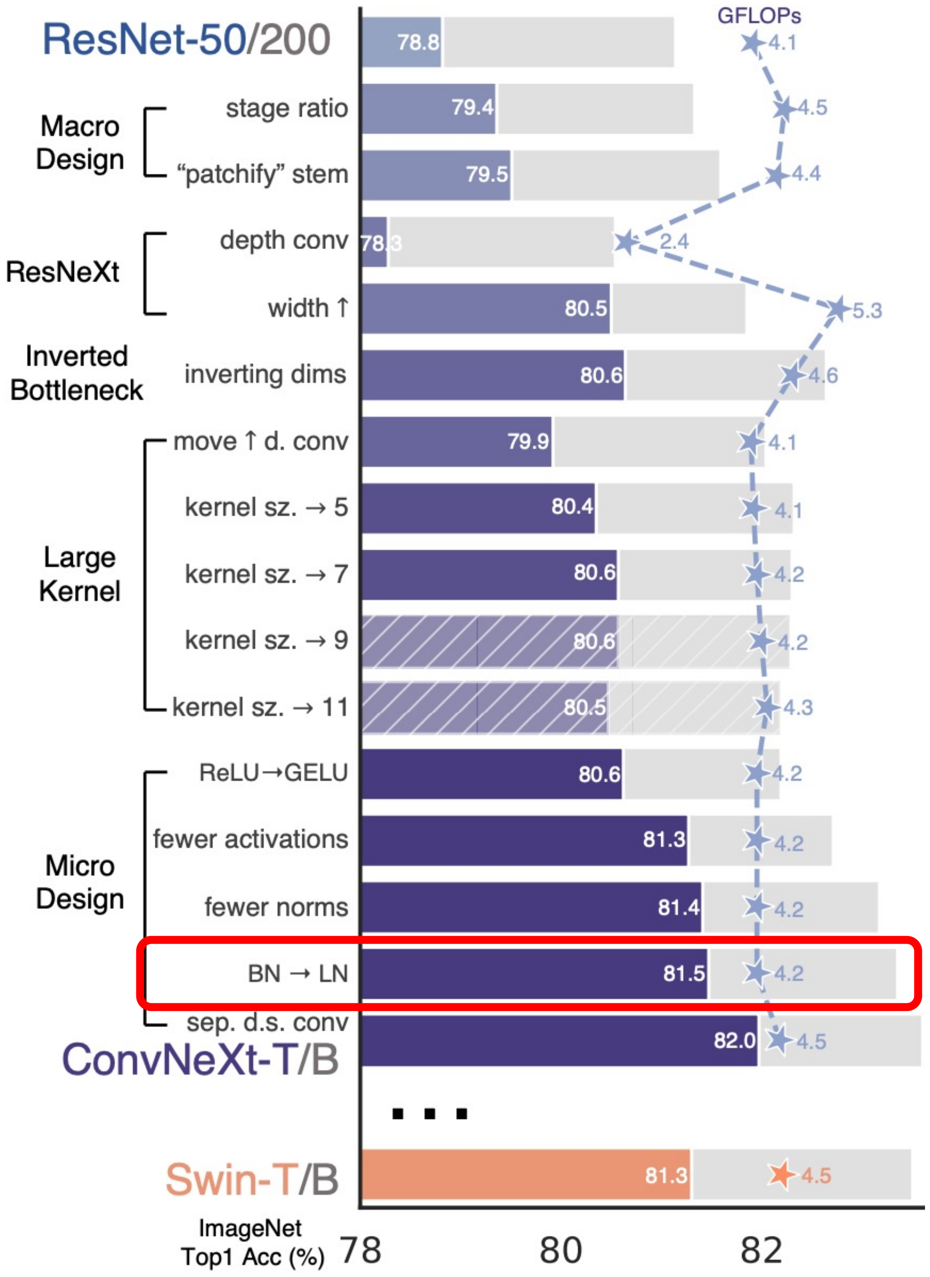


Micro Designs

BN -> LN

Prior attempts failed

Don't need BN statistics no more! Say 👋 to BN-related engineering headaches

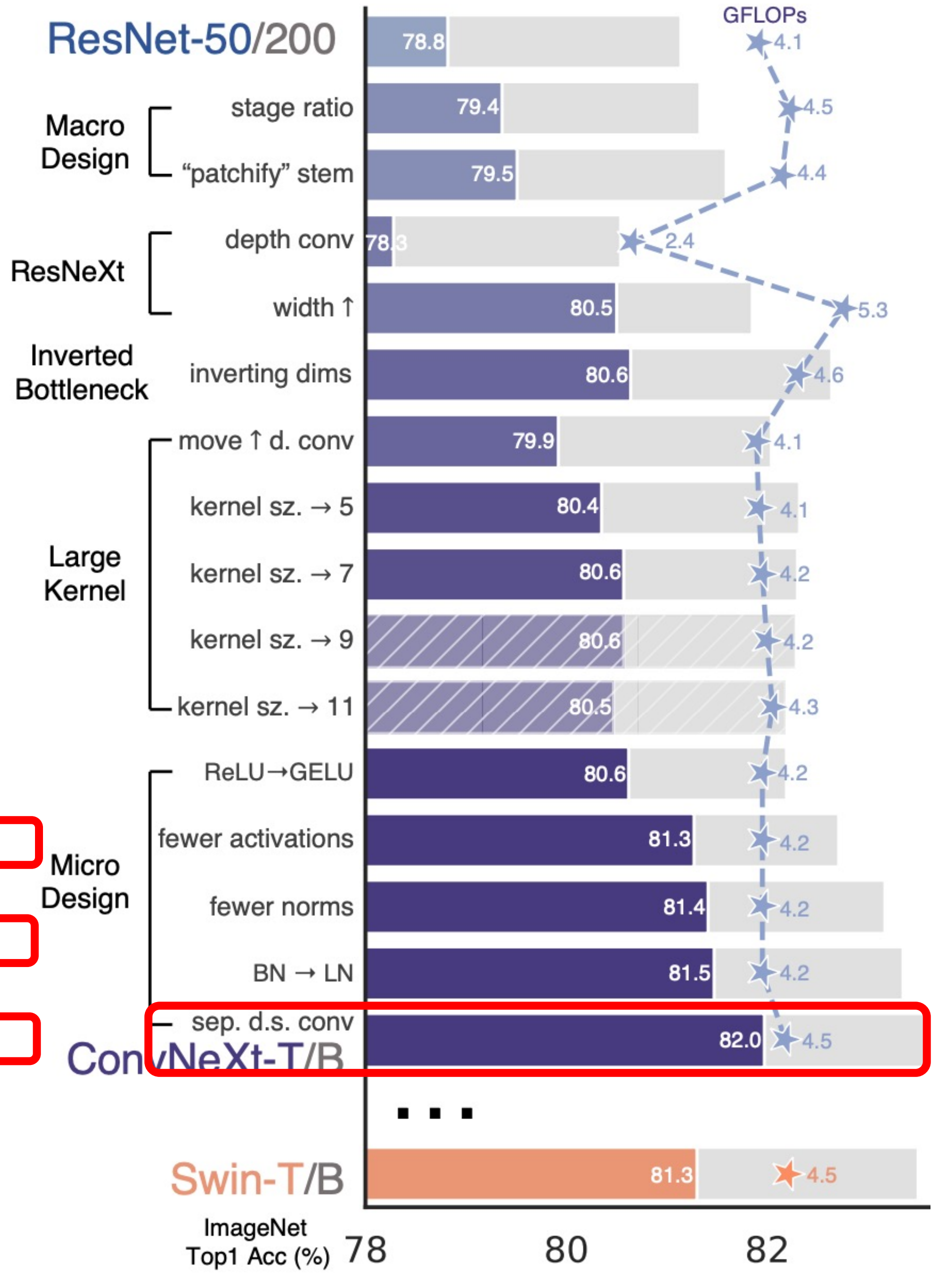


Micro Designs

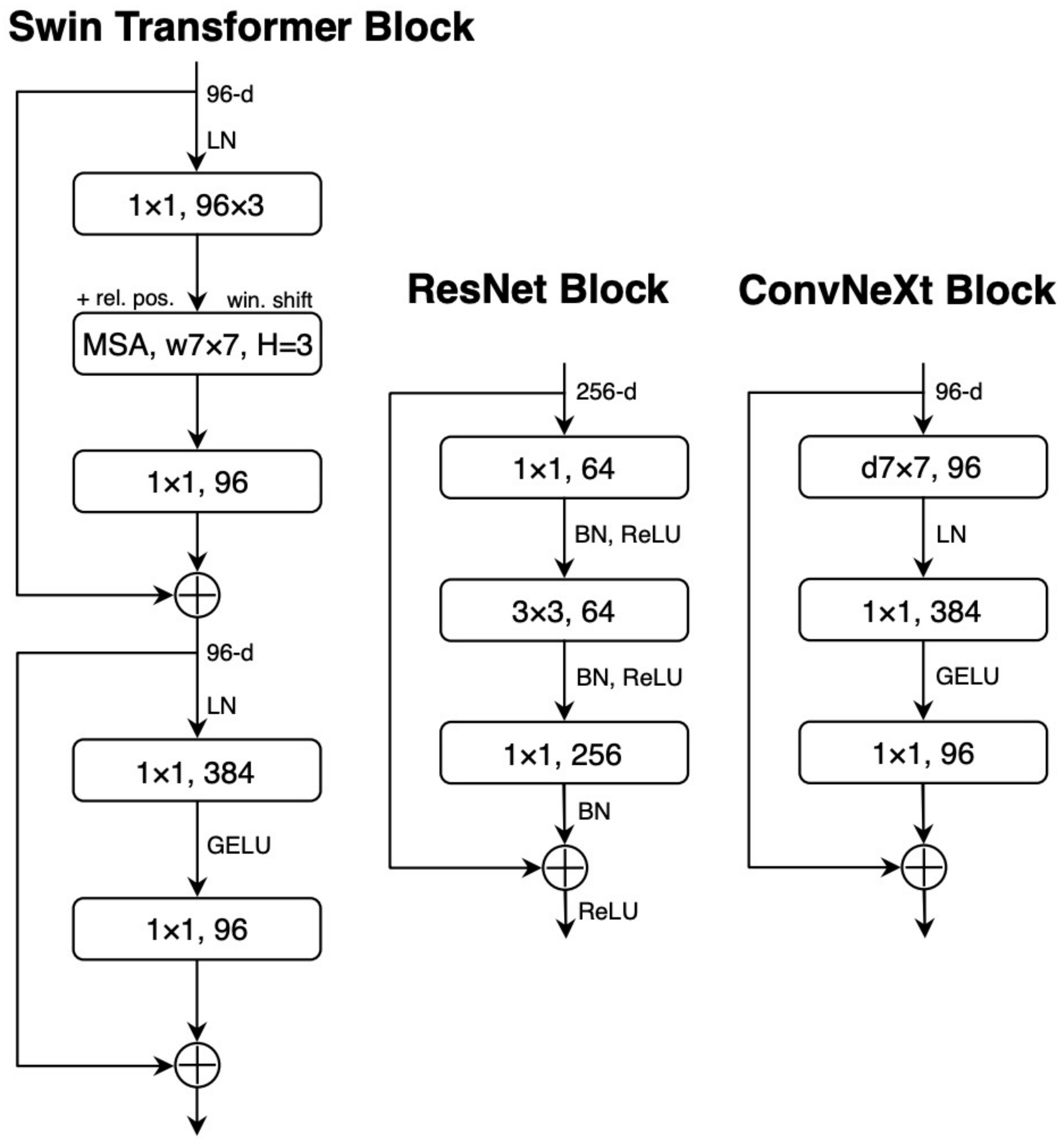
Use separate downsampling layers, and add corresponding norm layers

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

Table 7. Detailed architecture specifications.



Block Comparison



Overall Architecture

	output size	● ResNet-50	● ConvNeXt-T	○ Swin-T
stem	56×56	7×7, 64, stride 2 3×3 max pool, stride 2	4×4, 96, stride 4	4×4, 96, stride 4
res2	56×56	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} \text{d}7\times7, 96 \\ 1\times1, 384 \\ 1\times1, 96 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 96\times3 \\ \text{MSA, w}7\times7\text{, H=3, rel. pos.} \\ 1\times1, 96 \\ 1\times1, 384 \\ 1\times1, 96 \end{bmatrix} \times 2$
res3	28×28	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} \text{d}7\times7, 192 \\ 1\times1, 768 \\ 1\times1, 192 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 192\times3 \\ \text{MSA, w}7\times7\text{, H=6, rel. pos.} \\ 1\times1, 192 \\ 1\times1, 768 \\ 1\times1, 192 \end{bmatrix} \times 2$
res4	14×14	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} \text{d}7\times7, 384 \\ 1\times1, 1536 \\ 1\times1, 384 \end{bmatrix} \times 9$	$\begin{bmatrix} 1\times1, 384\times3 \\ \text{MSA, w}7\times7\text{, H=12, rel. pos.} \\ 1\times1, 384 \\ 1\times1, 1536 \\ 1\times1, 384 \end{bmatrix} \times 6$
res5	7×7	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} \text{d}7\times7, 768 \\ 1\times1, 3072 \\ 1\times1, 768 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 768\times3 \\ \text{MSA, w}7\times7\text{, H=24, rel. pos.} \\ 1\times1, 768 \\ 1\times1, 3072 \\ 1\times1, 768 \end{bmatrix} \times 2$
FLOPs		4.1×10^9	4.5×10^9	4.5×10^9
# params.		25.6×10^6	28.6×10^6	28.3×10^6

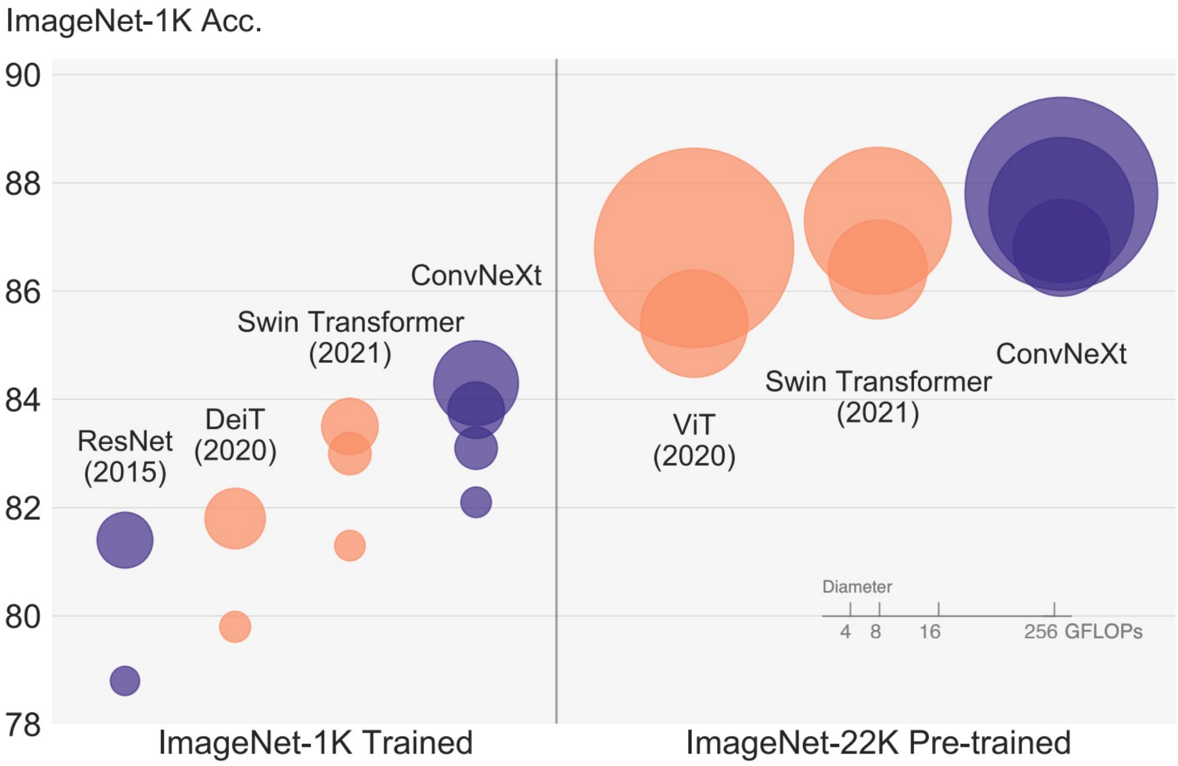
ConvNeXt Variants

- One appealing property of Transformers is its scaling behavior
  - Model/compute size
  - Data size

- We build ConvNeXt variants to compete
  - 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: $C = (256, 512, 1024, 2048)$, $B = (3, 3, 27, 3)$

ImageNet-1K/22K Results

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**



Downstream Transfers

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

COCO Detection and Instance Segmentation

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

ADE20K Semantic Segmentation

Robustness Benchmarks

Model	Data/Size	FLOPs / Params	Clean	C (↓)	$\bar{C}$ (↓)	A	R	SK
ResNet-50	1K/224^2	4.1 / 25.6	76.1	76.7	57.7	0.0	36.1	24.1
Swin-T [42]	1K/224^2	4.5 / 28.3	81.2	62.0	-	21.6	41.3	29.1
RVT-S* [44]	1K/224^2	4.7 / 23.3	81.9	49.4	37.5	25.7	47.7	34.7
ConvNeXt-T	1K/224^2	4.5 / 28.6	82.1	53.2	40.0	24.2	47.2	33.8
Swin-B [42]	1K/224^2	15.4 / 87.8	83.4	54.4	-	35.8	46.6	32.4
RVT-B* [44]	1K/224^2	17.7 / 91.8	82.6	46.8	**30.8**	28.5	48.7	36.0
ConvNeXt-B	1K/224^2	15.4 / 88.6	**83.8**	**46.8**	34.4	**36.7**	**51.3**	**38.2**
ConvNeXt-B	22K/384^2	45.1 / 88.6	86.8	43.1	30.7	62.3	64.9	51.6
ConvNeXt-L	22K/384^2	101.0 / 197.8	87.5	40.2	29.9	65.5	66.7	52.8
ConvNeXt-XL	22K/384^2	179.0 / 350.2	**87.8**	**38.8**	**27.1**	**69.3**	**68.2**	**55.0**

ConvNeXt is easy to implement and use

- ~100 lines of PyTorch

- available timm (pytorch-image-models library)

- available in torchvision; with even higher accuracy

Summary

- ConvNeXt, a simple and pure ConvNet

- As good as SOTA hierarchical vision Transformers

- Modifications inspired by Transformers; architecture not novel

- Challenge some beliefs and rethink the importance of convolution