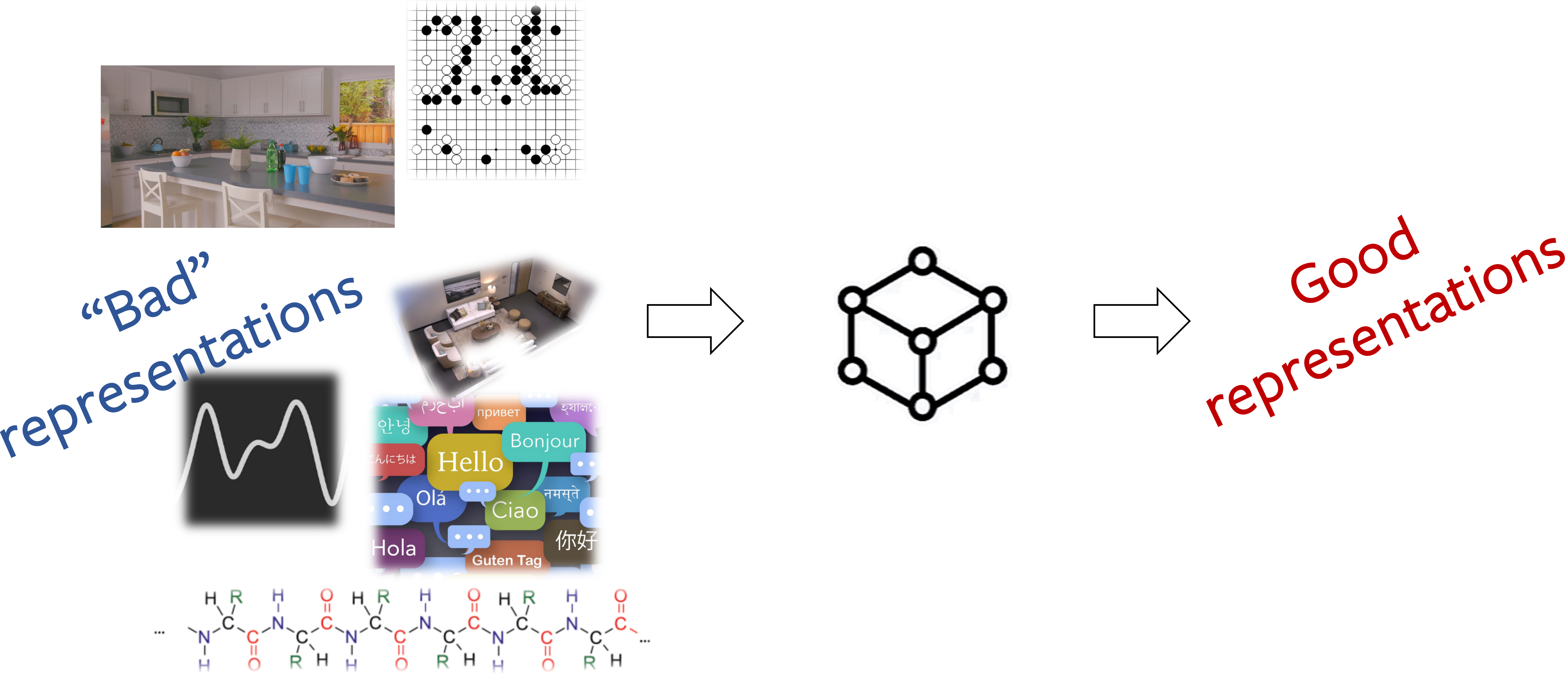
A ConvNet for the 2020s

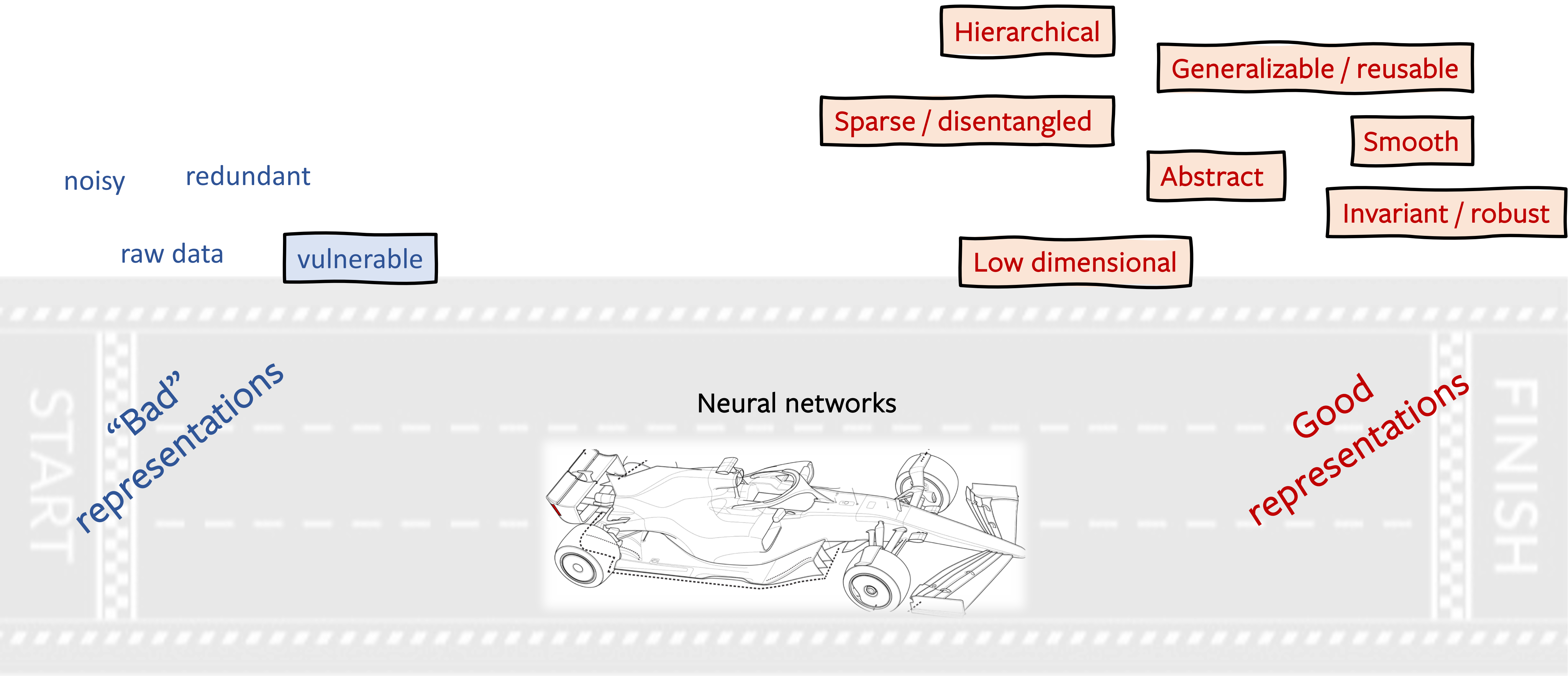
Code/Pre-trained models: https://github.com/facebookresearch/ConvNeXt

Saining Xie
Facebook AI Research (FAIR)

With Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell
FAIR & UC Berkeley



“Bad” representations
привет
Bonjour
Hello
Olá
Ciao
नमस्ते
Hola
你好
Guten Tag
Good representations


Hierarchical
Generalizable / reusable
Sparse / disentangled
Smooth
noisy
redundant
Abstract
Invariant / robust
raw data
vulnerable
Low dimensional
START
“Bad” representations
Neural networks
Good representations
FINISH

Scalable

Hierarchical

Sparse / disentangled

Generalizable / reusable

Smooth

Abstract

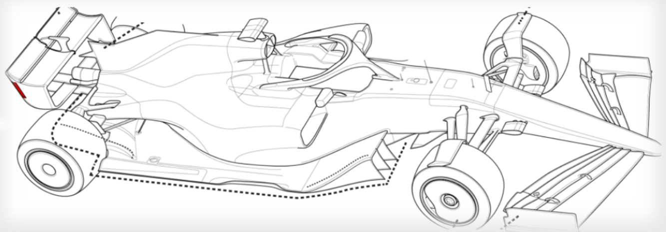
Invariant / robust

Low dimensional

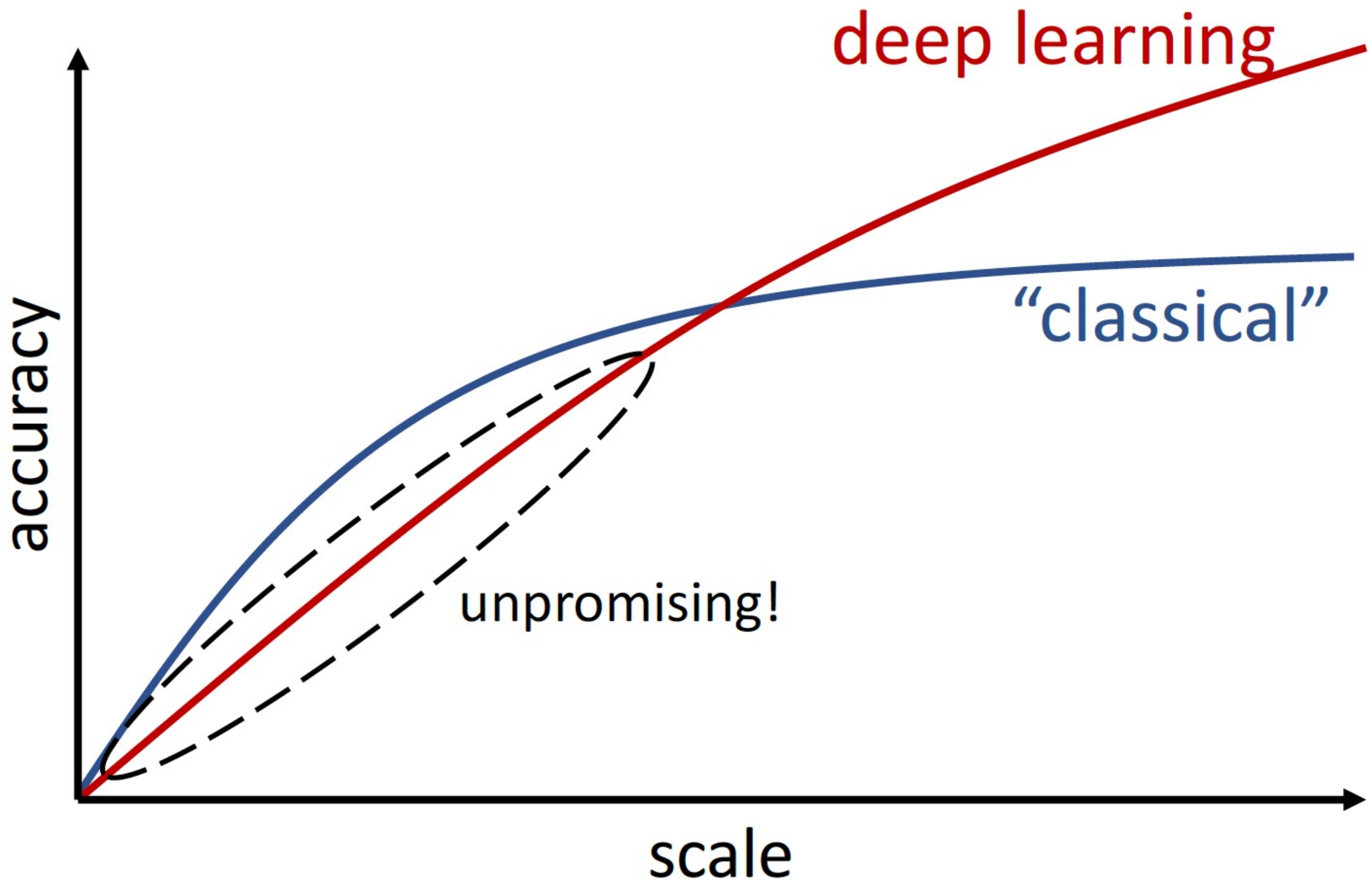
START

FINISH

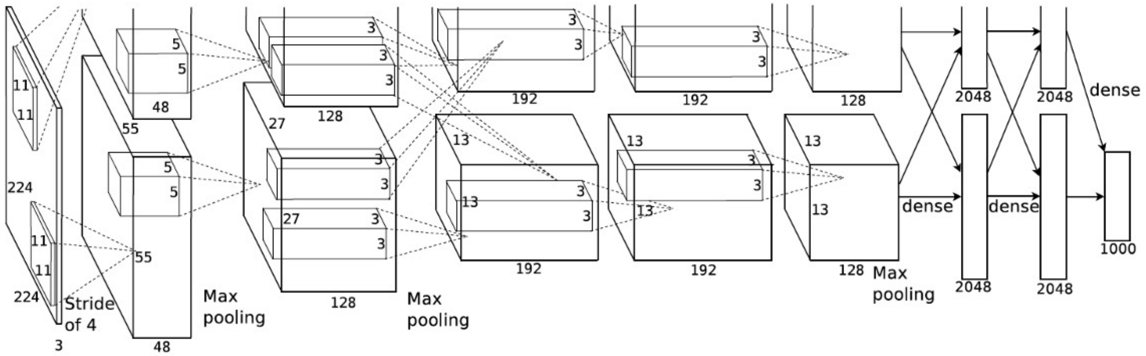
Neural networks



⭐ Scalable



2012 AlexNet



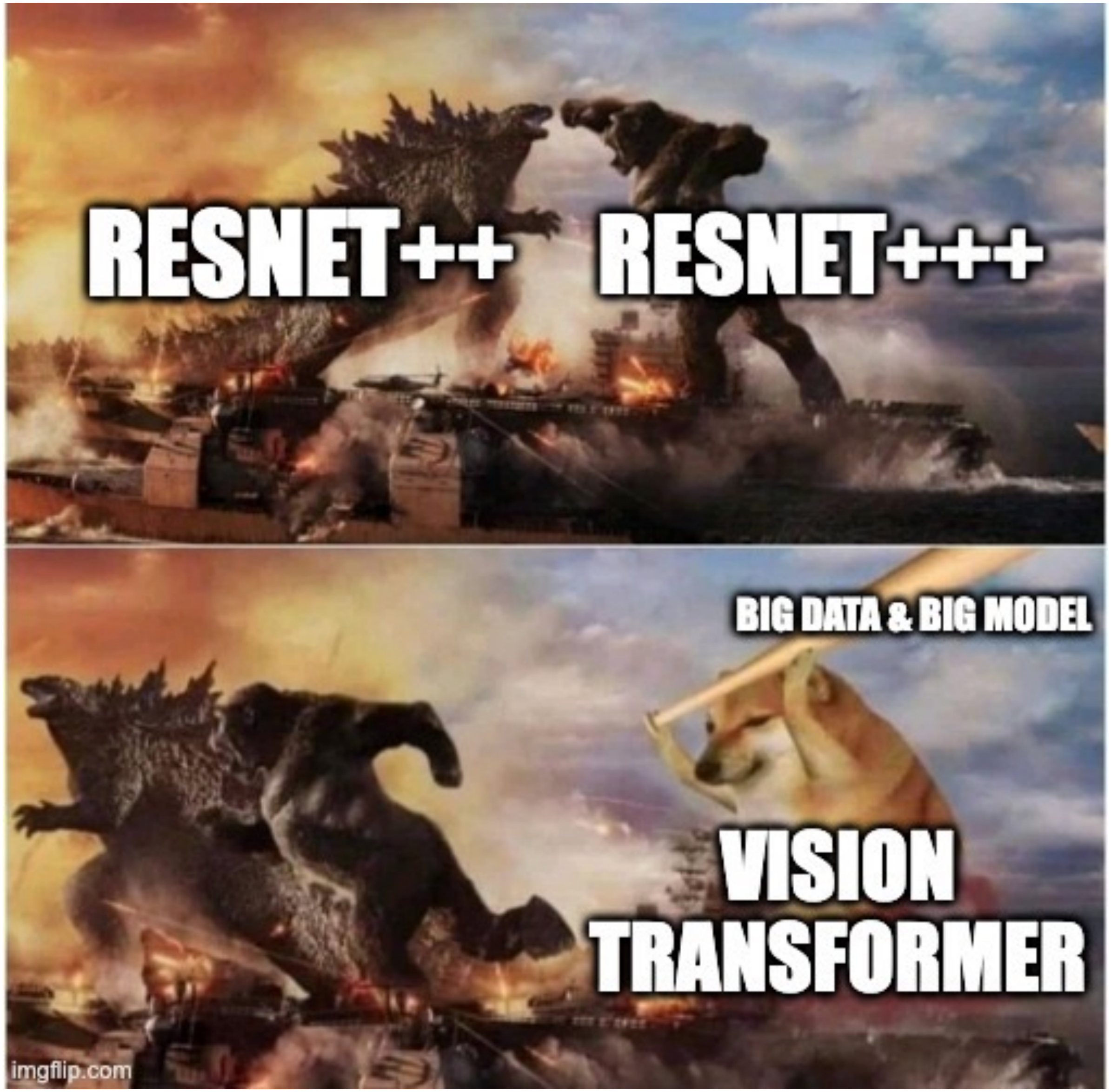
2012 AlexNet

2015 ResNet

2012 AlexNet

2015 ResNet

2020 Vision Transformer

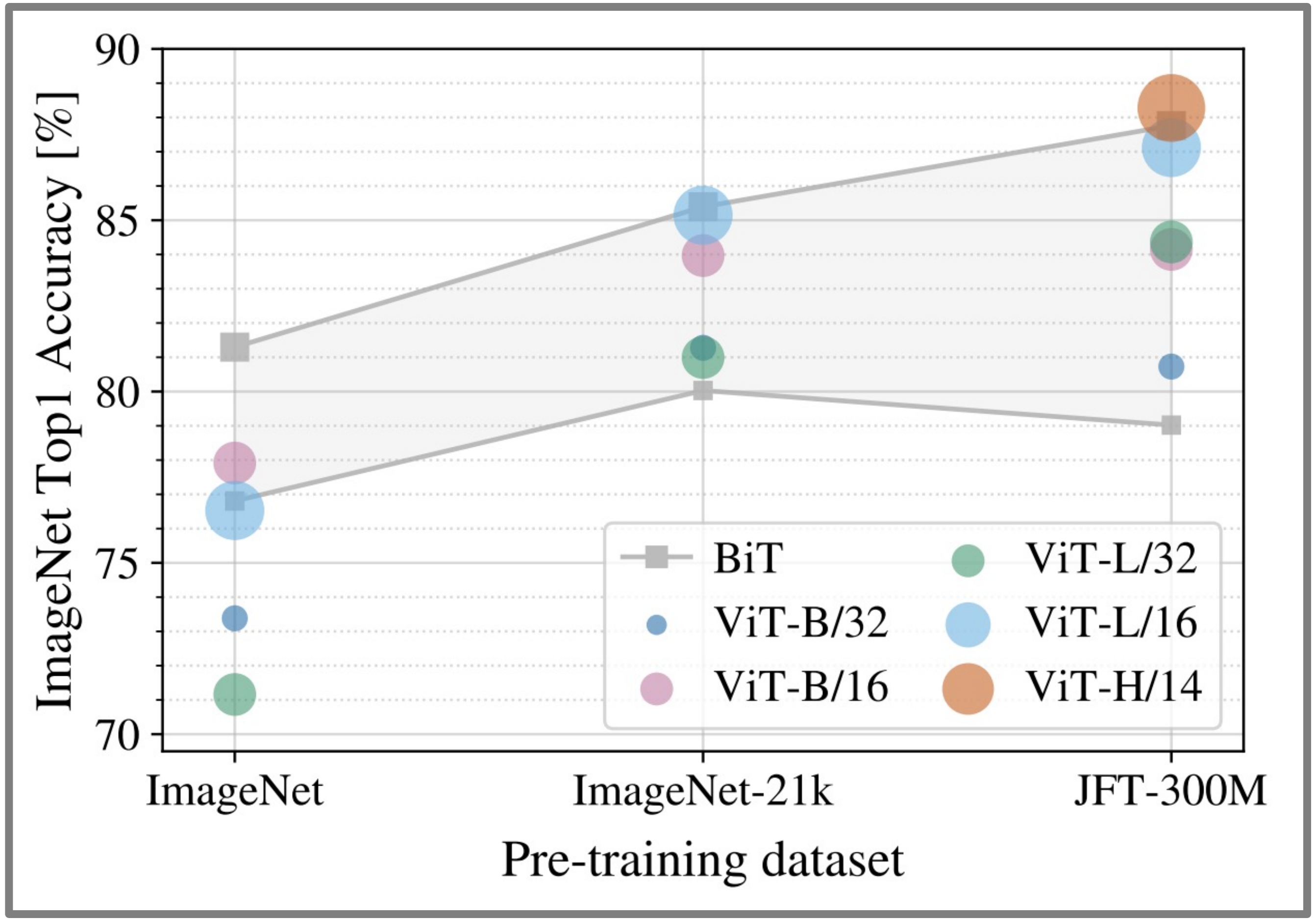


Vision Transformer (ViT)

- Self-attention:
  - Less inductive bias

⇩

- Scalable
  - w/ larger model
  - w/ bigger data



[Dosovitskiy et al, ICLR 2021]

Vision is not just about classification

- Expanding vision capabilities
  - Large resolution
  - Multi-scale

- Vision Transformer
  - Quadratic complexity
  - No hierarchy

Swin Transformer

[Liu, et al, ICCV 2021]

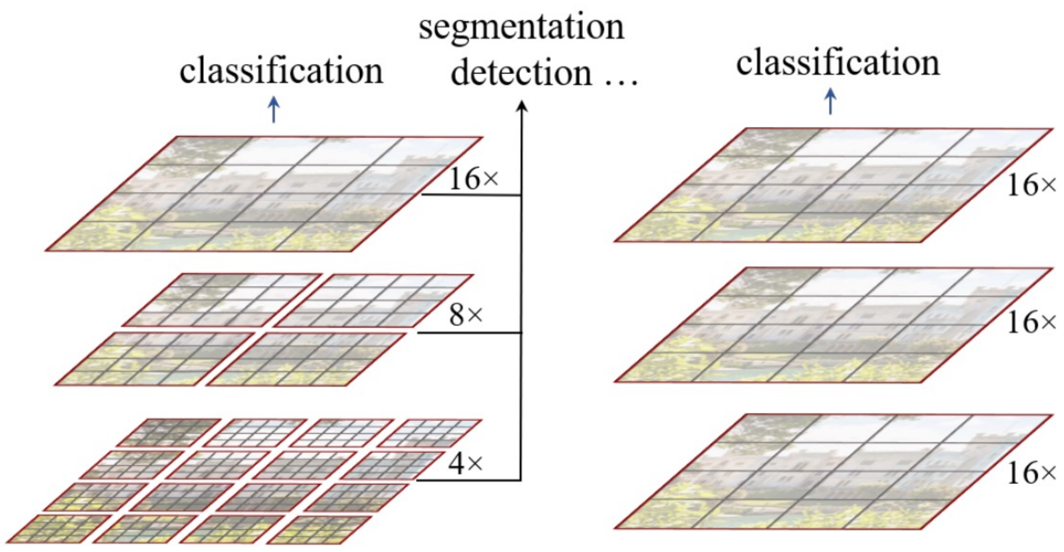
Self-attention (ViT)

+ Conv Priors

Locality

Translation equivariance

Hierarchical



(a) Swin Transformer (ours)

(b) ViT

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

Swin Transformer

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

ConvNet losing steam?

- Venue

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

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)

Swin Transformers is a hybrid architecture

- Similarity: Convolution inductive bias

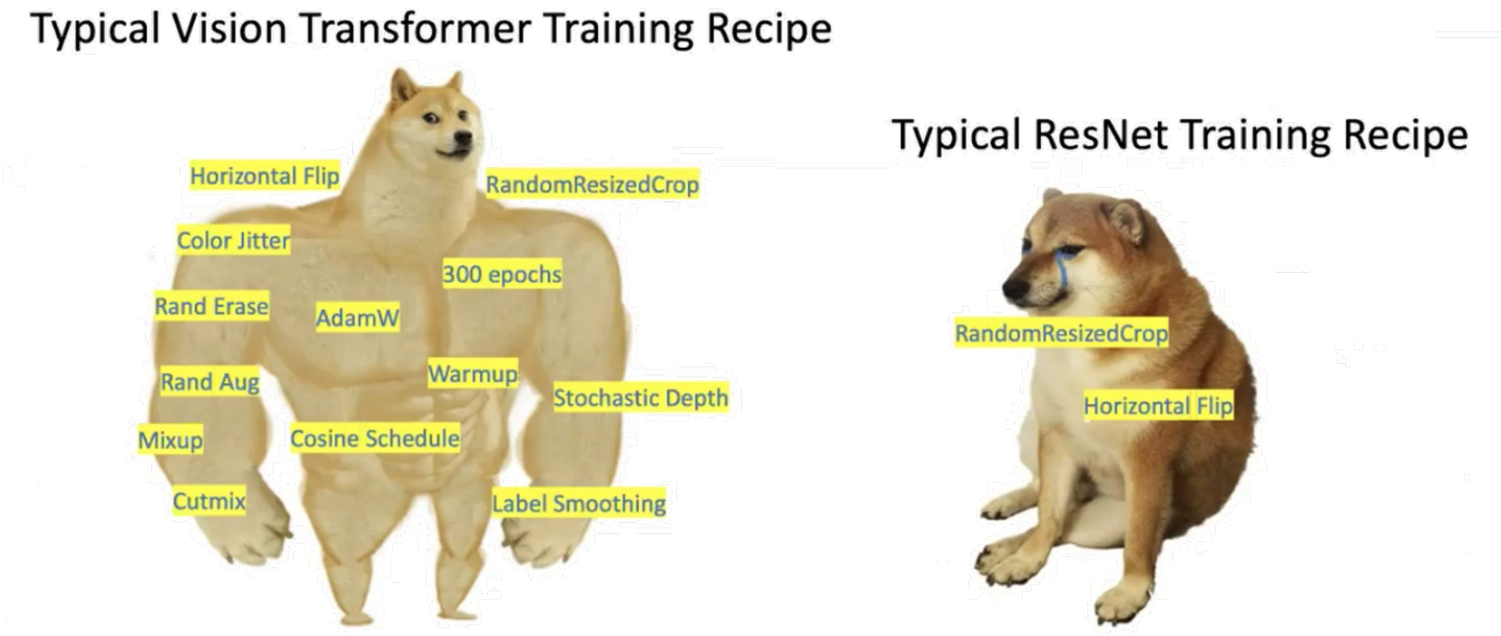
- Difference:
  - “Core” component (attention vs. convolution)
  - Training procedures
  - Macro and micro architecture design decisions

- Common belief in the 2020s:
  - **Self-attention** is the key for superior performance and scalability.
  - ConvNet is NOT a scalable architecture.

To do this, we…

- Start with a simple standard ResNet
- Modernize the architecture towards the construction of a hierarchical vision transformer
- Central question: How do the design choices in Transformers impact a ConvNet's performance?

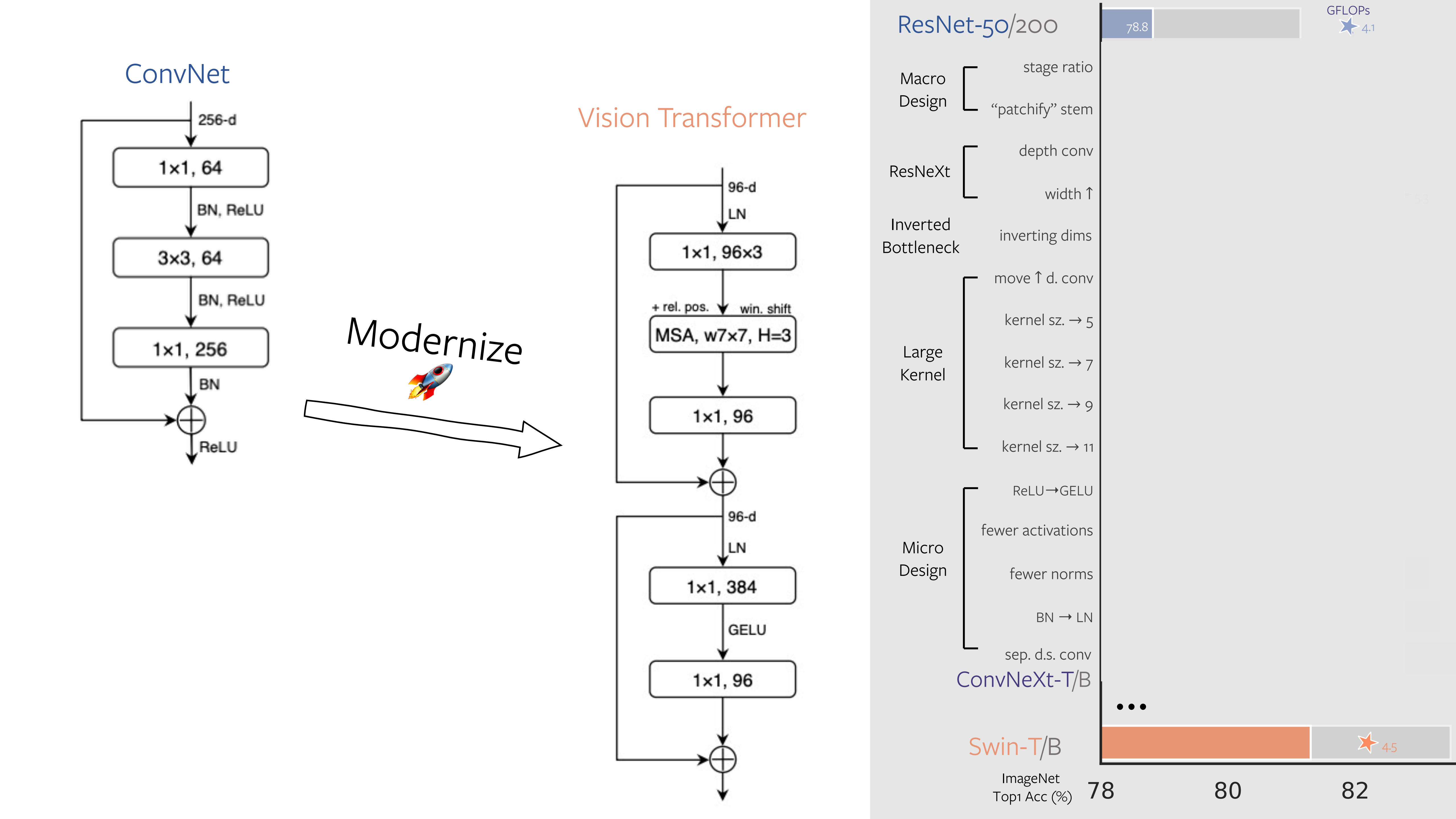
Improved training recipe



ResNet-50 ImageNet top-1: 76.7% -> 78.8% 🚀

[Revisiting ResNets: Improved Training and Scaling Strategies, Bello et al, 2021]

[ResNet strikes back: An improved training procedure in timm Wightman, et al, 2021]



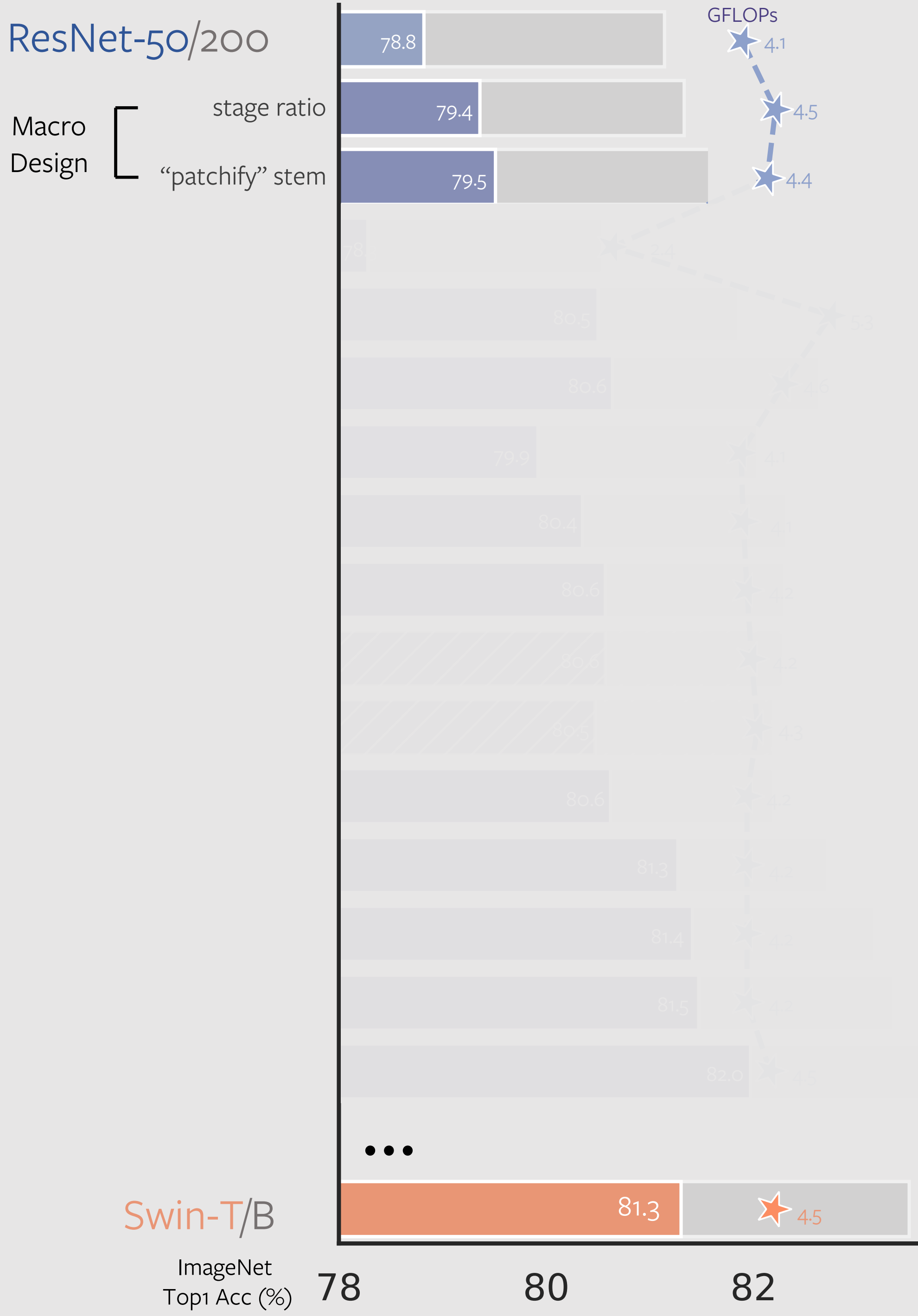
ConvNet
256-d
1×1, 64
BN, ReLU
3×3, 64
BN, ReLU
1×1, 256
BN
ReLU
Modernize
Vision Transformer
96-d
LN
1×1, 96×3
+ rel. pos.
win. shift
MSA, w7×7, H=3
1×1, 96
96-d
LN
1×1, 384
GELU
1×1, 96
ResNet-50/200
78.8
GFLOPs
4.1
Macro Design
stage ratio
"patchify" stem
ResNeXt
depth conv
width ↑
Inverted Bottleneck
inverting dims
Large Kernel
move ↑ d. conv
kernel sz. → 5
kernel sz. → 7
kernel sz. → 9
kernel sz. → 11
Micro Design
ReLU→GELU
fewer activations
fewer norms
BN → LN
sep. d.s. conv
ConvNeXt-T/B
Swin-T/B
4.5
ImageNet Top1 Acc (%)
78
80
82

Stage compute ratio

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

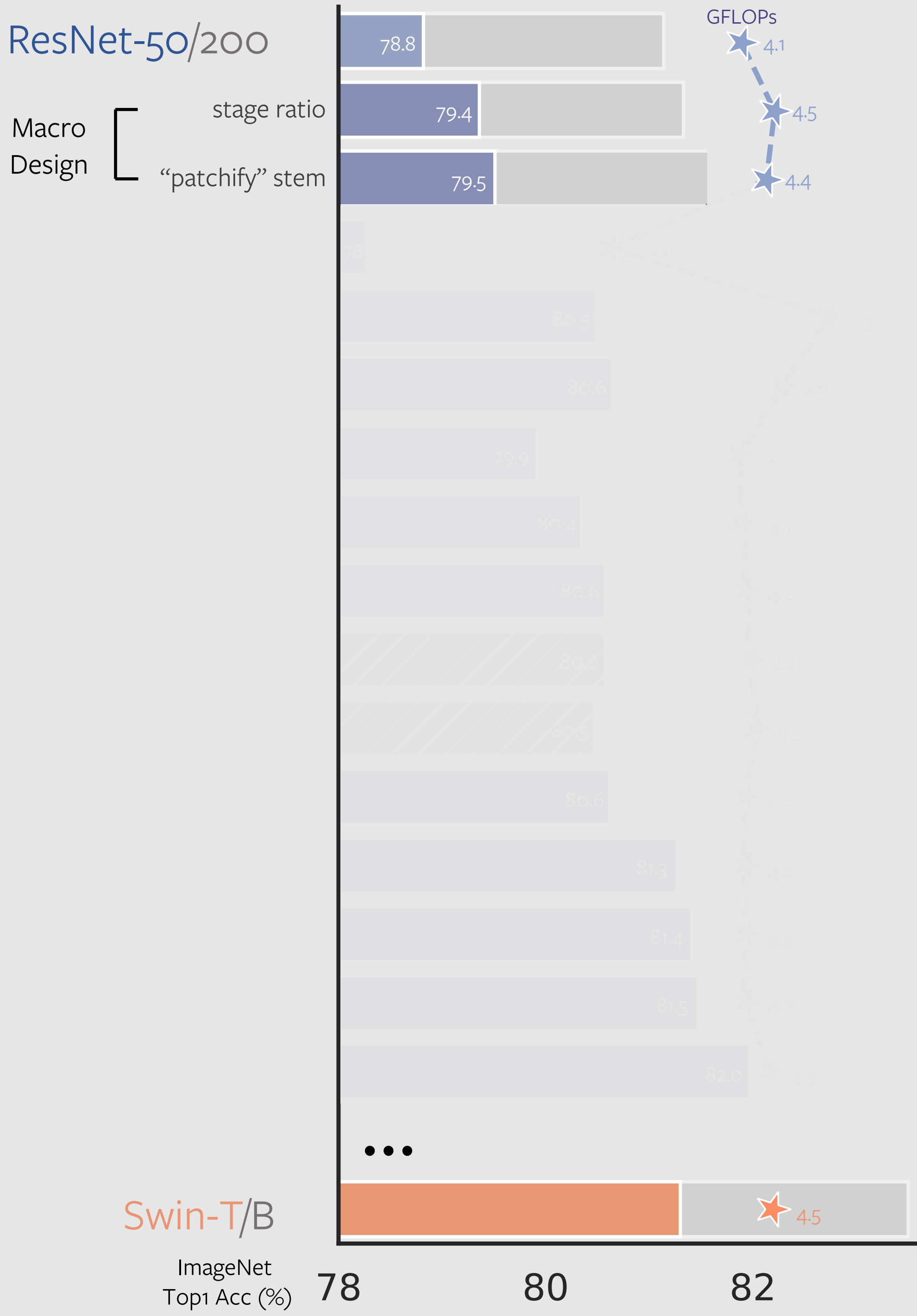
Swin-T 1:1:3:1
ResNet-50 [3,4,6,3]

Ours. [3,3,9,3]



ResNet Input Stem

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



Input Stem

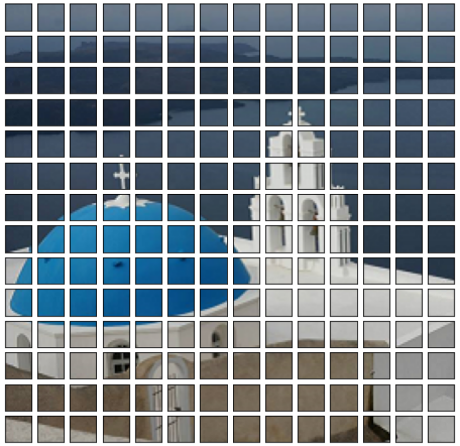
Overlapping Conv

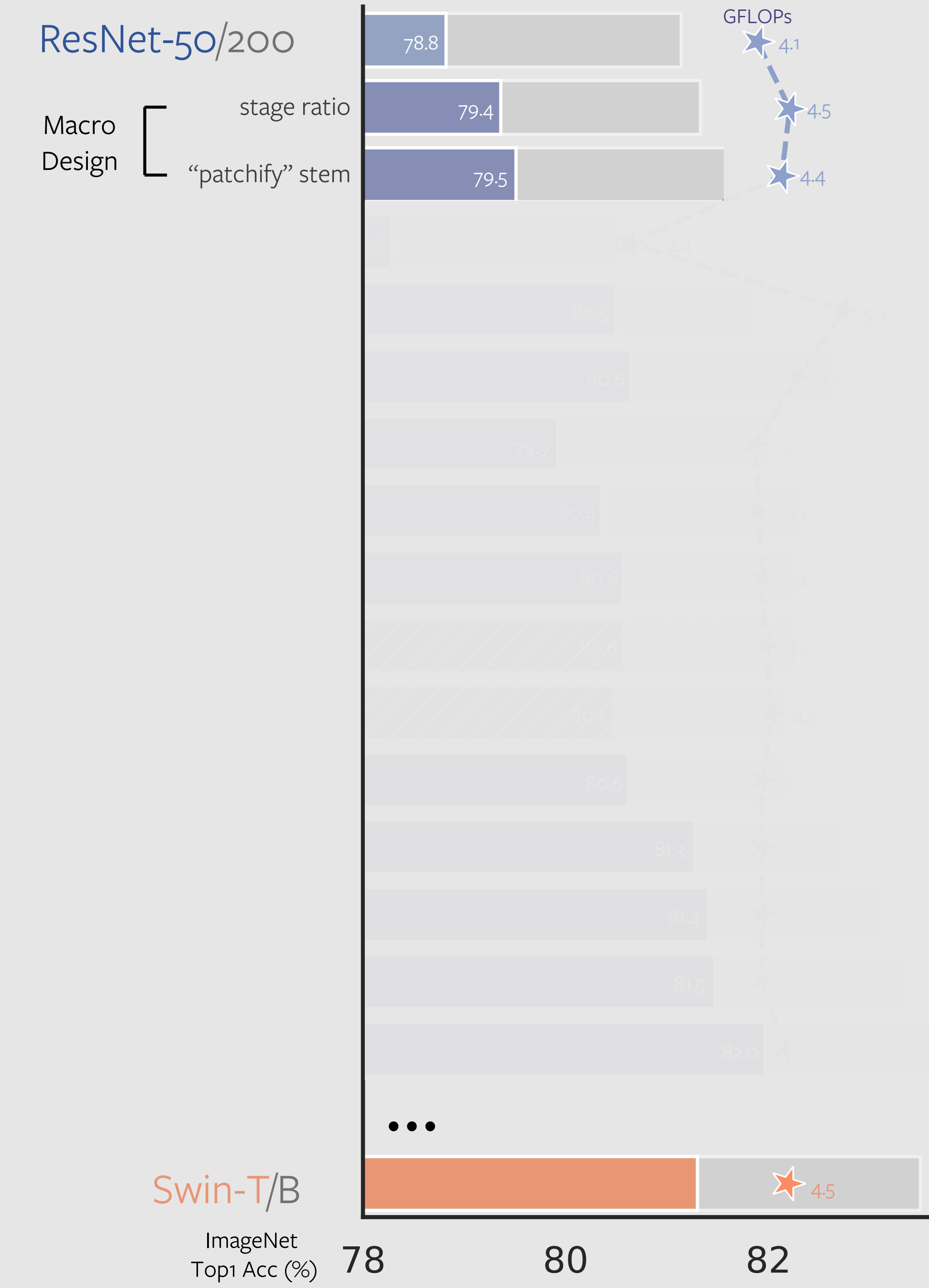
+

Max Pooling

Non-overlapping Conv (4x4, stride 4)

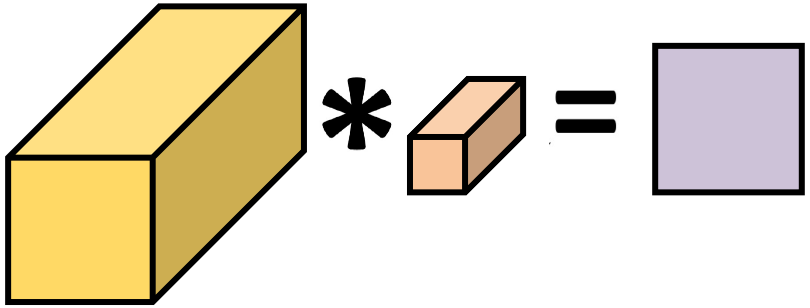
(a.k.a Patchify)



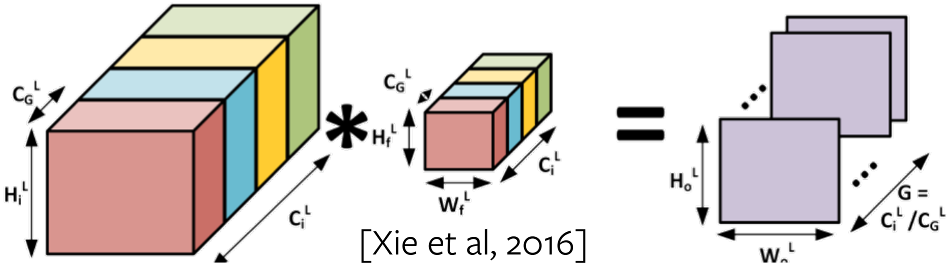


ResNeXt-ify

Dense Conv

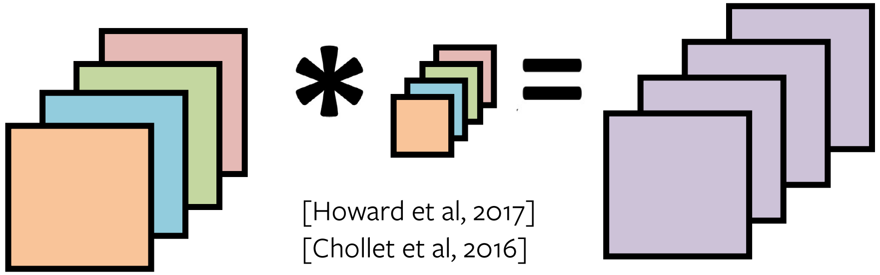


Grouped Conv

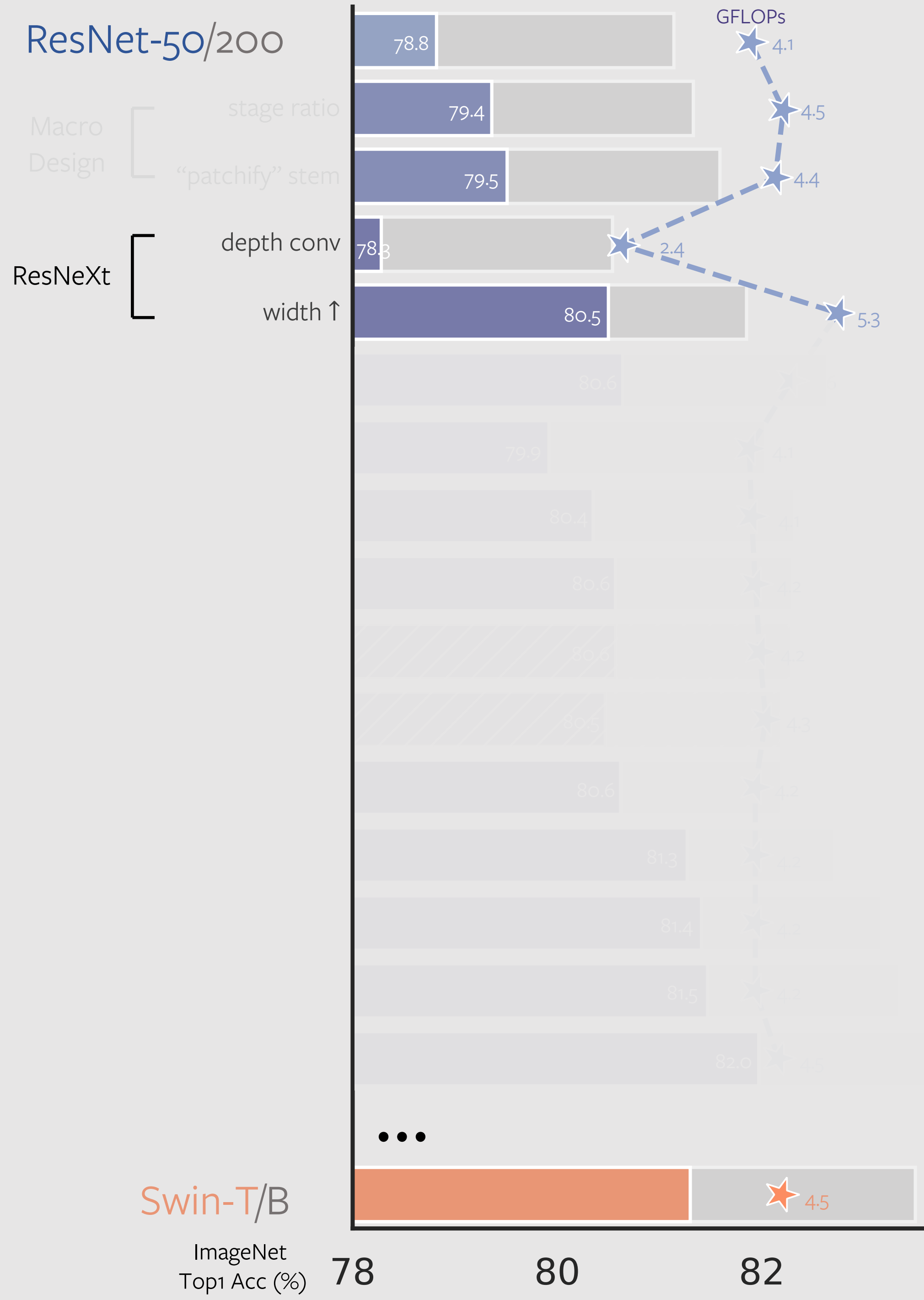


[Xie et al, 2016]

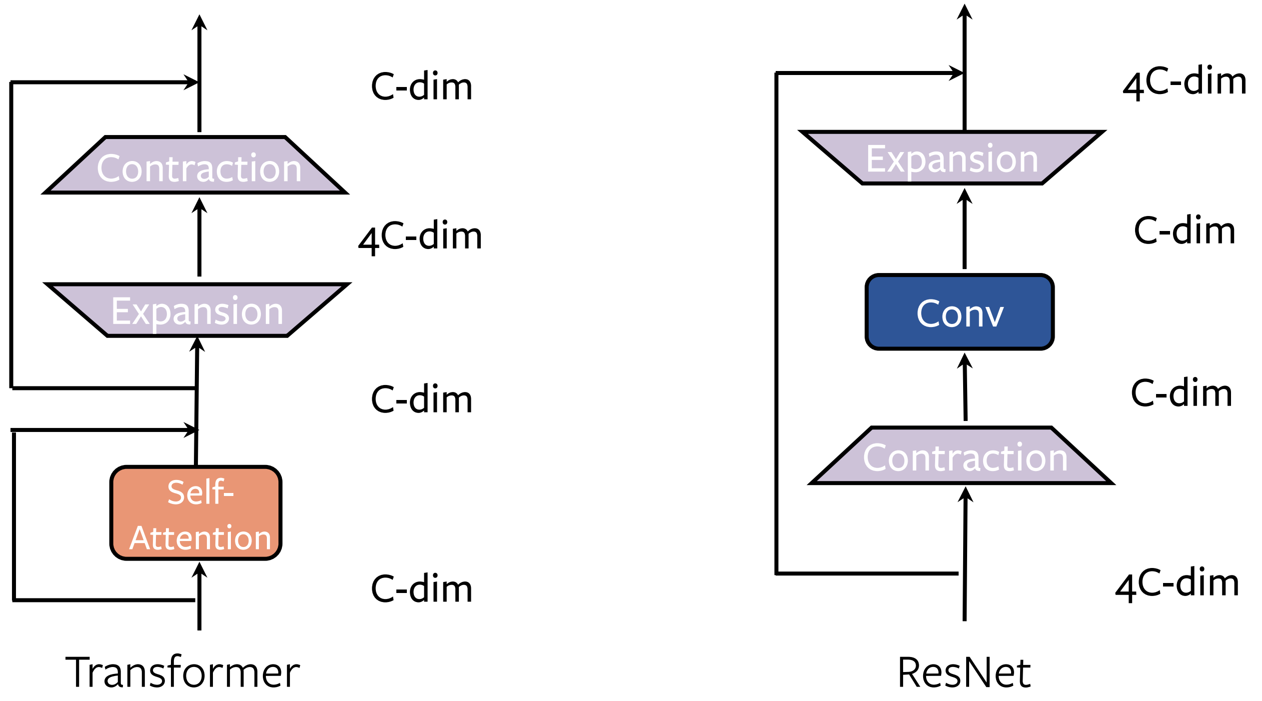
Depthwise Conv
groups = # channels

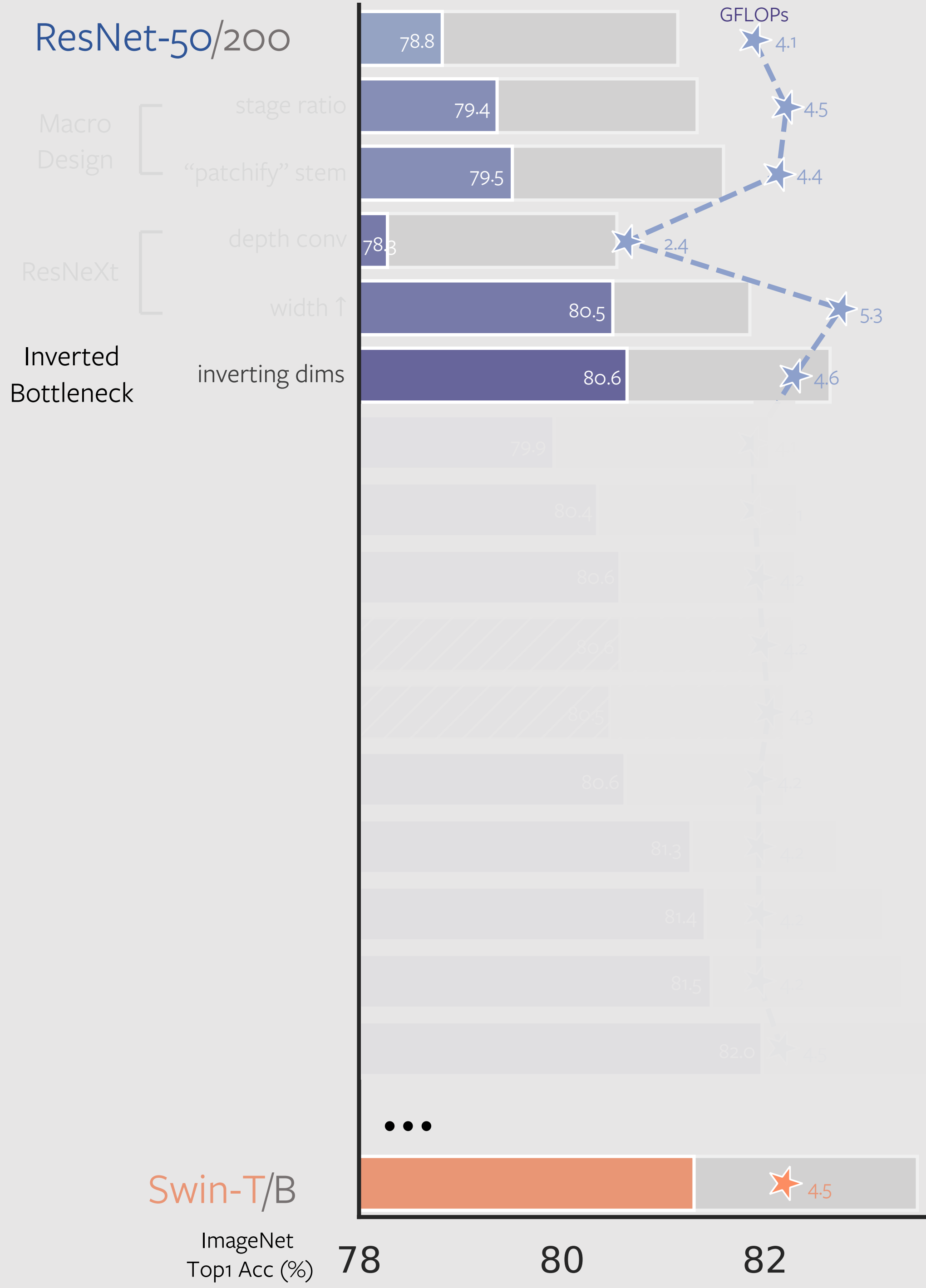


[Howard et al, 2017]
[Chollet et al, 2016]

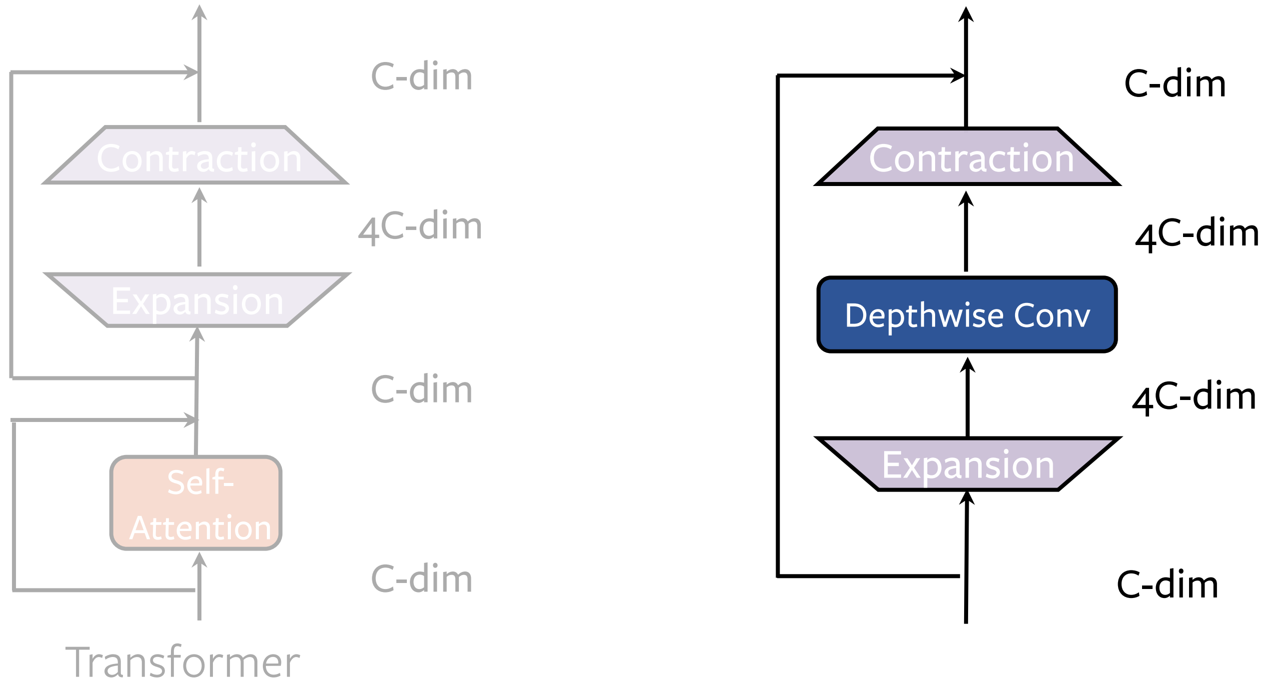


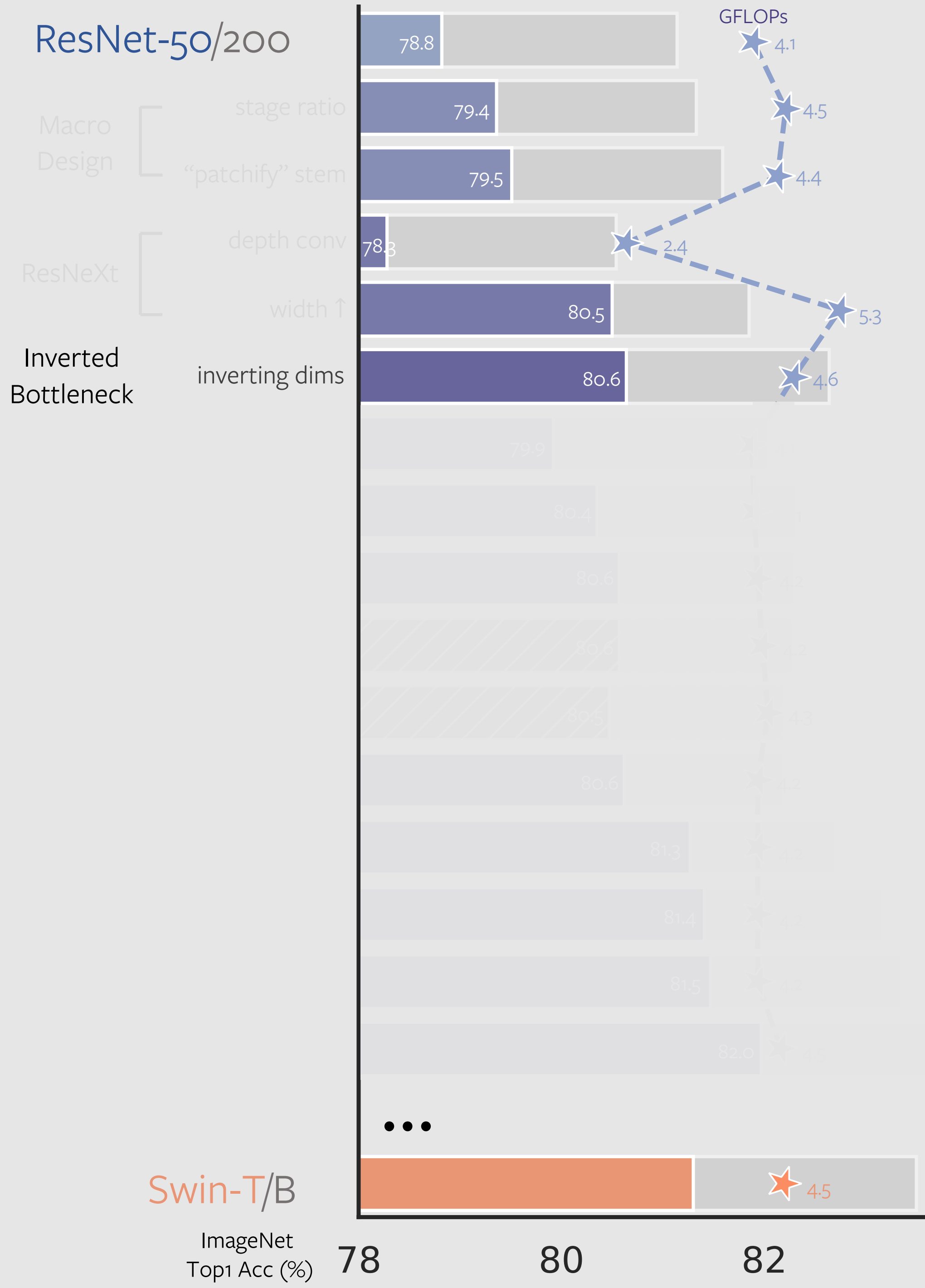
Inverted Bottleneck



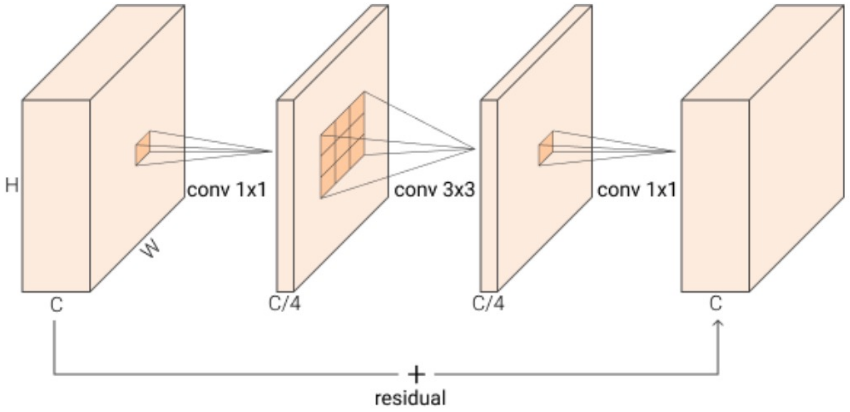


Inverted Bottleneck





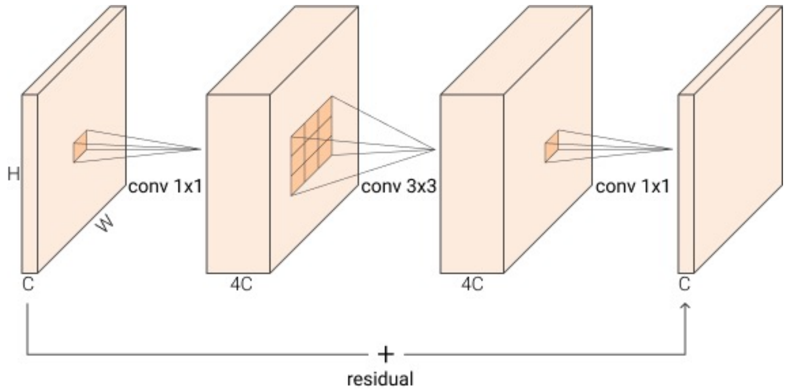
2016 ResNet

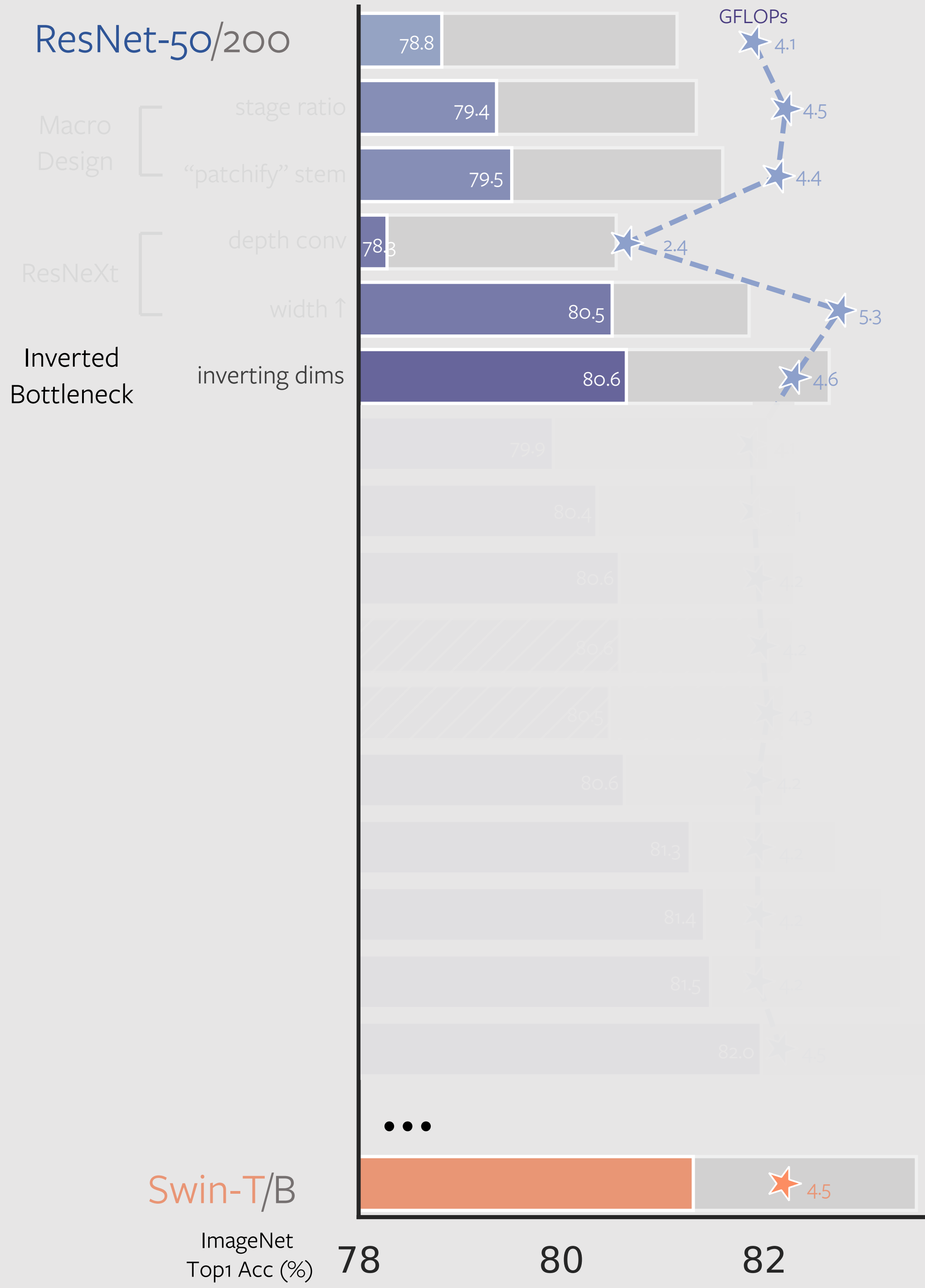


2018 Inverted Bottleneck

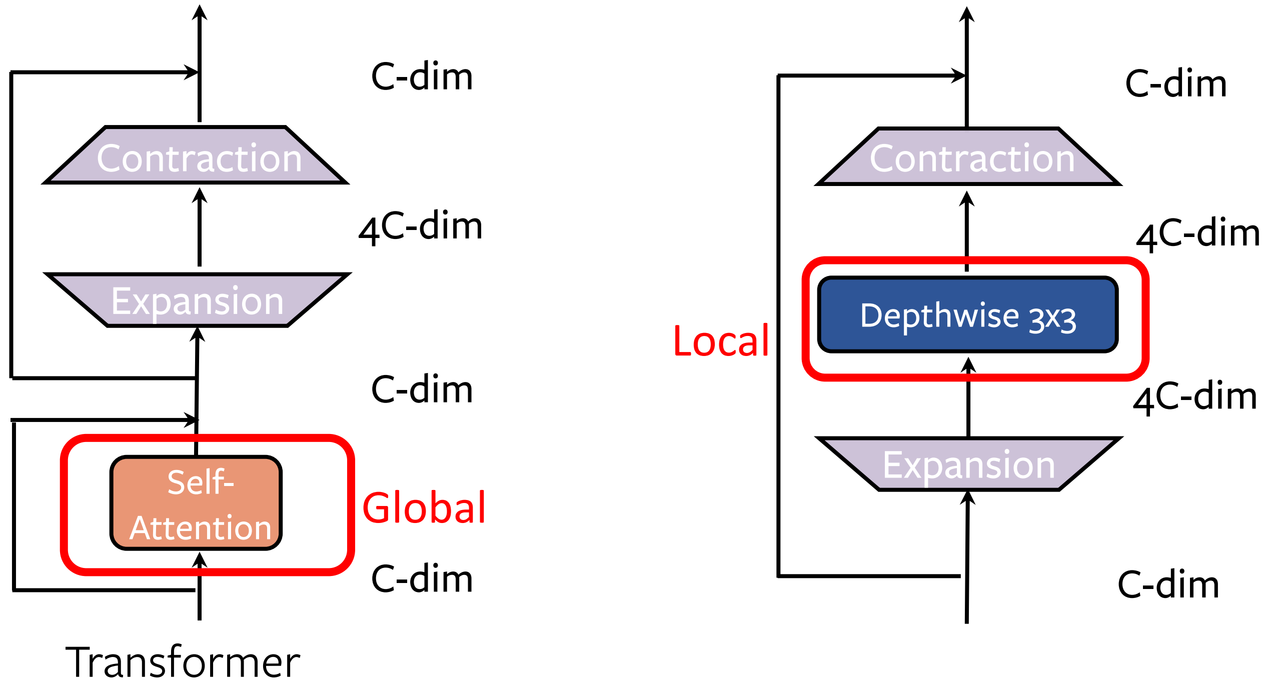
MobileNet v2

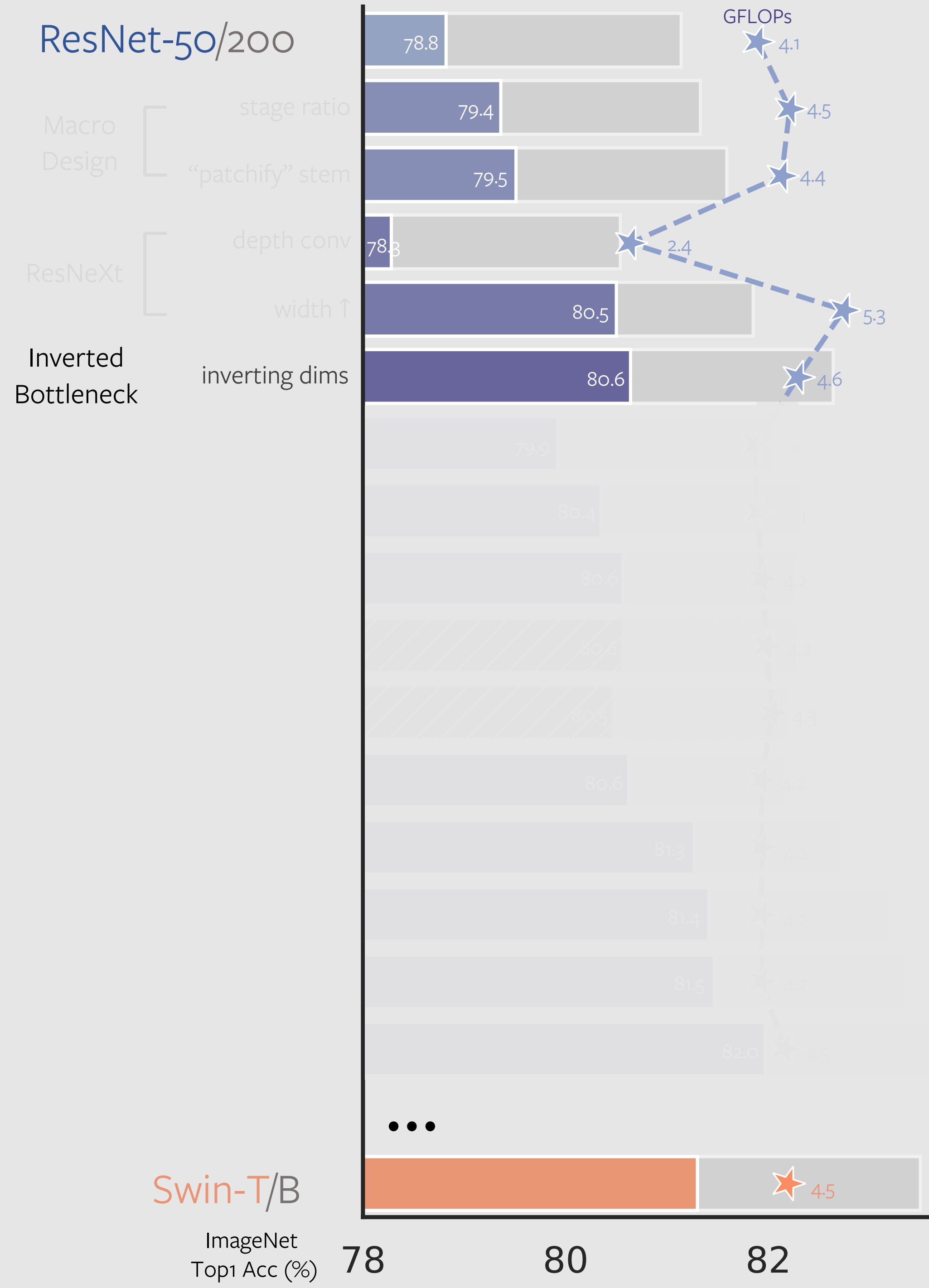
[Sandler et al, 2018]



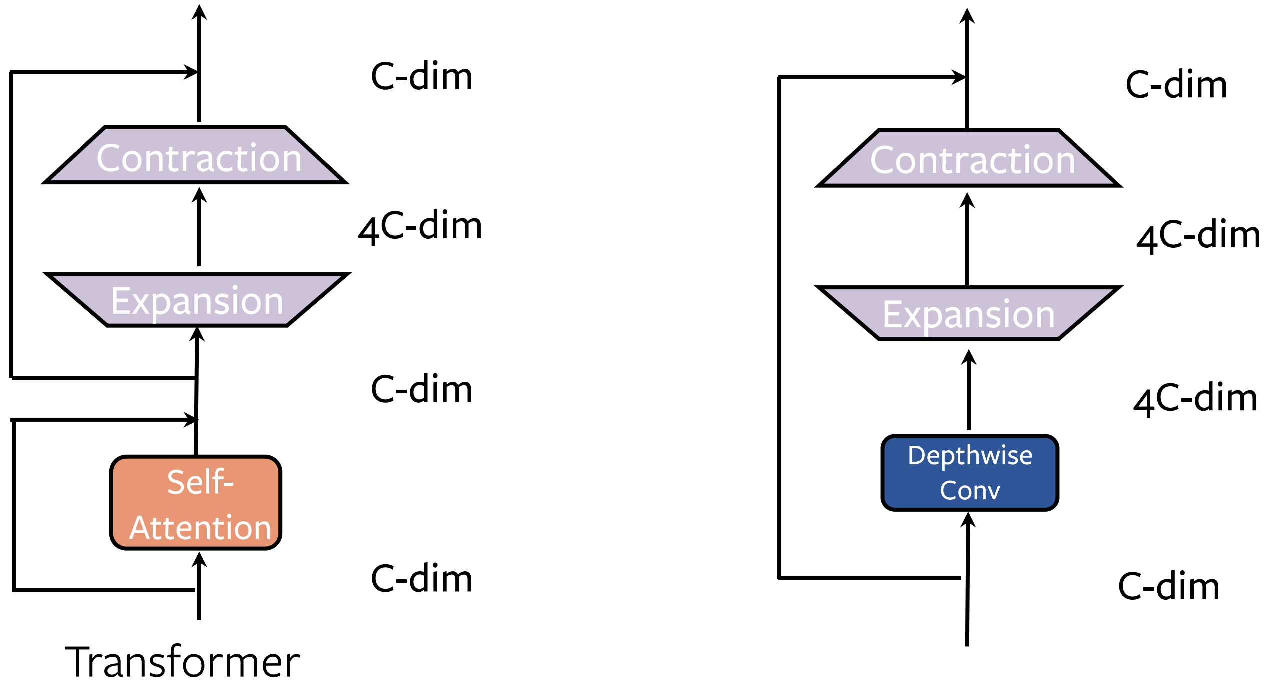


Large kernel size

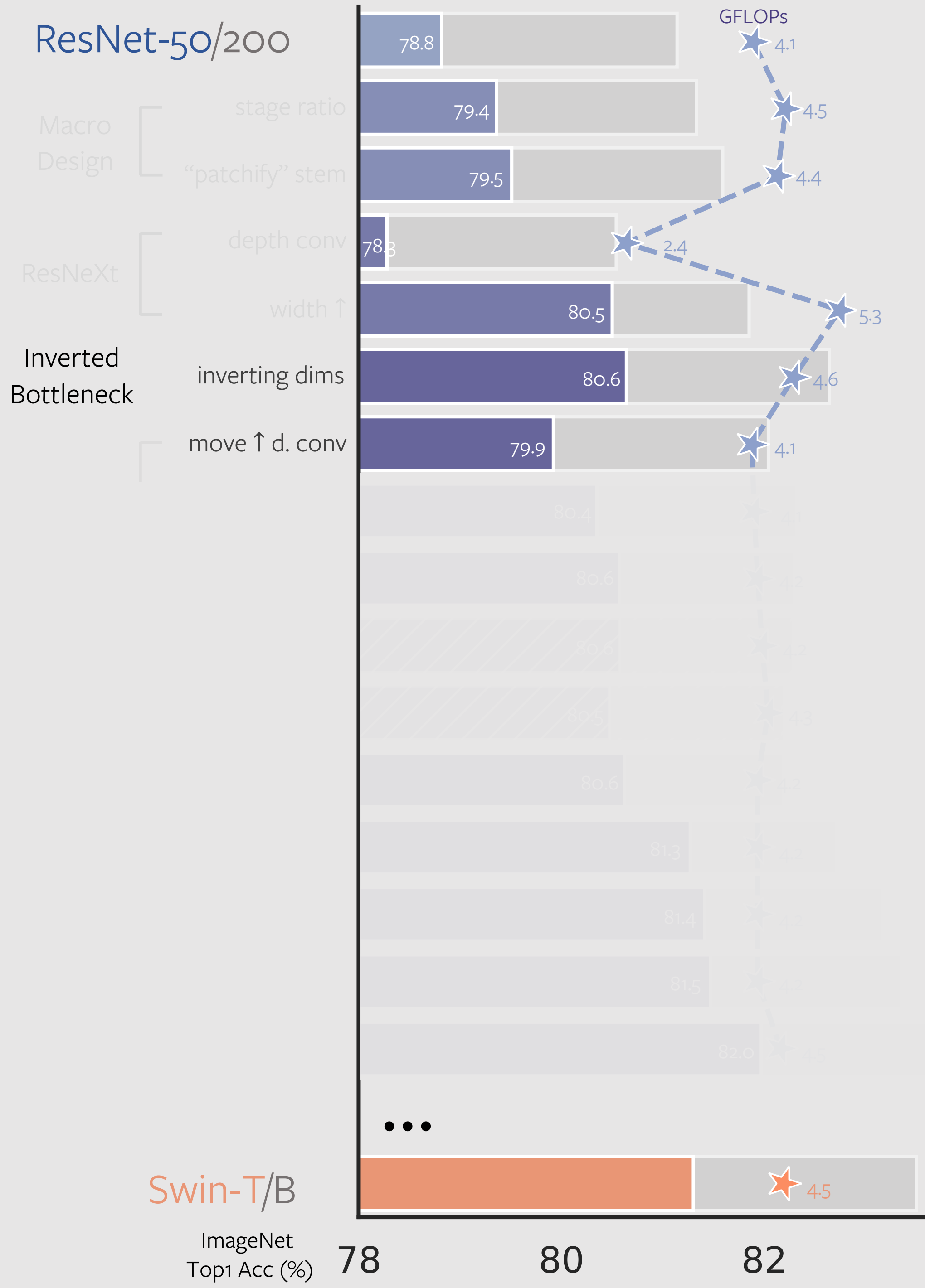




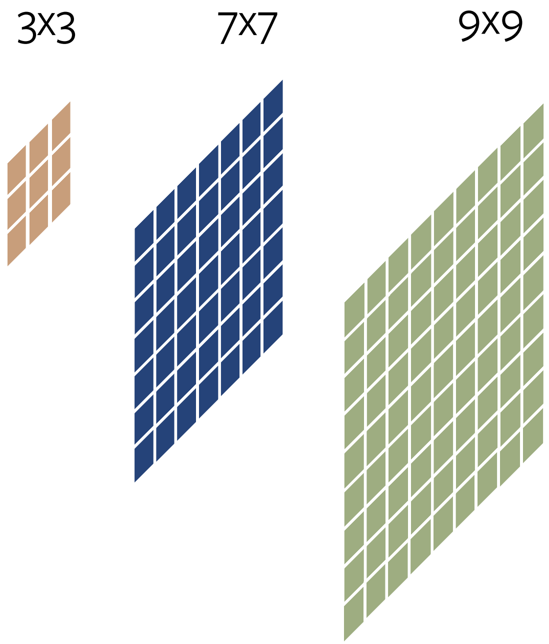
Large kernel size

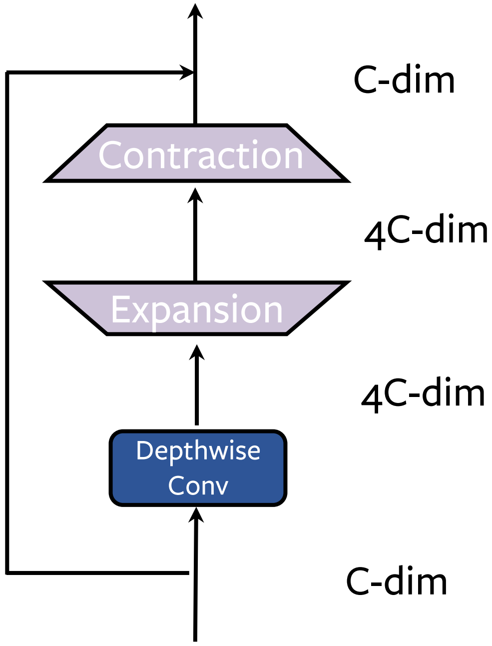


Prerequisite: move depth-wise before "expansion".
Reduce flops w/ larger kernel size.

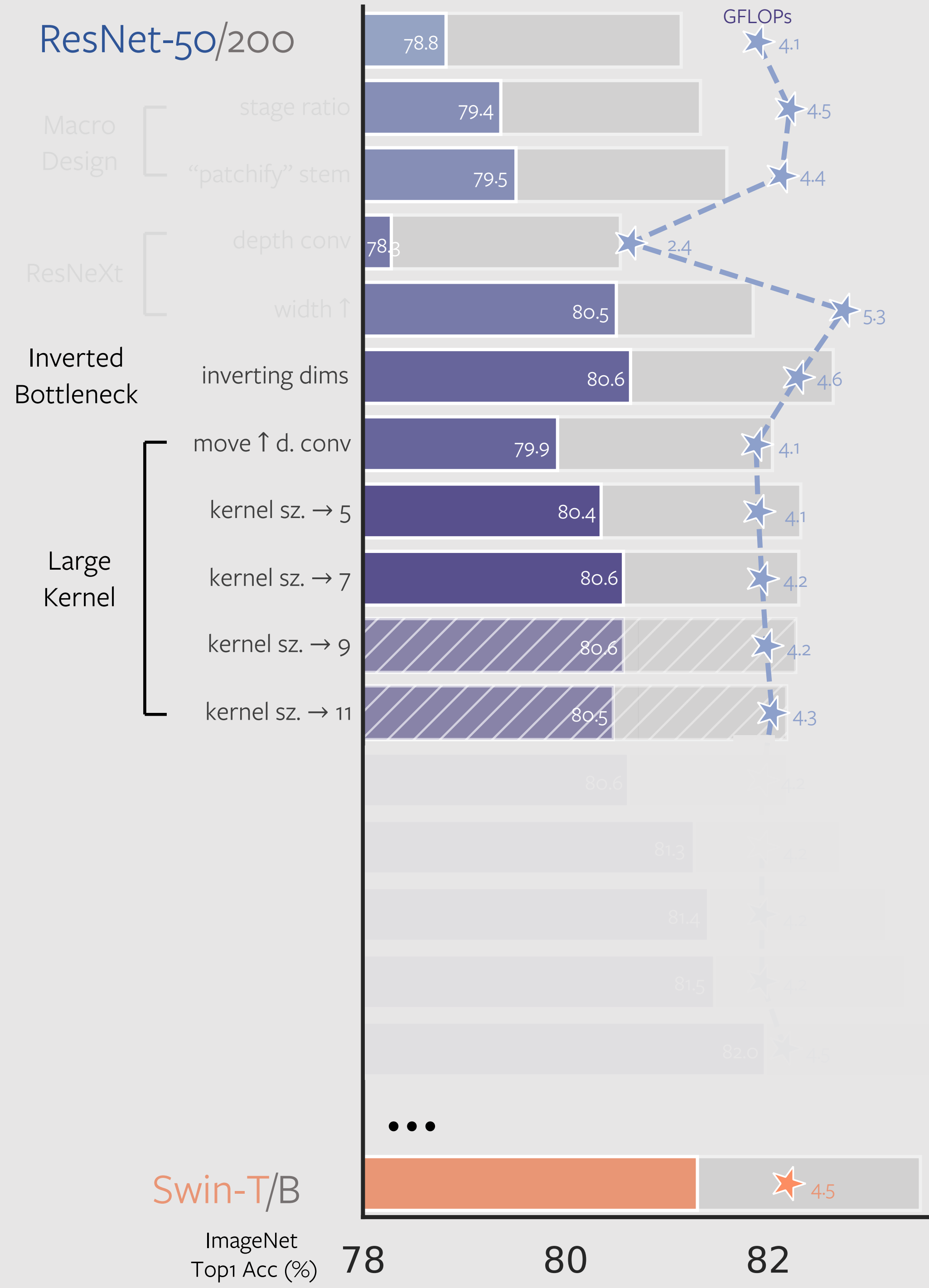


Large kernel size

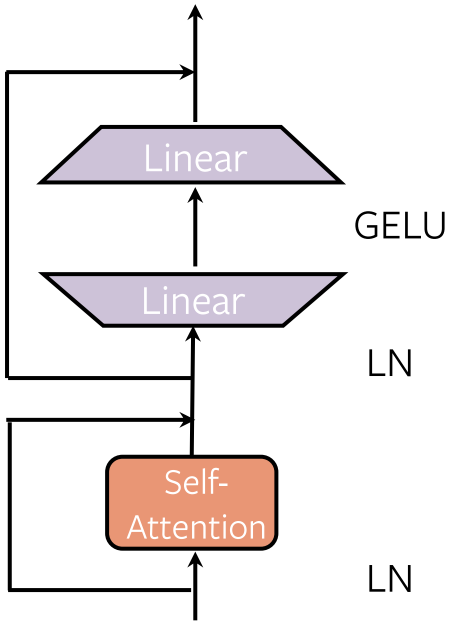




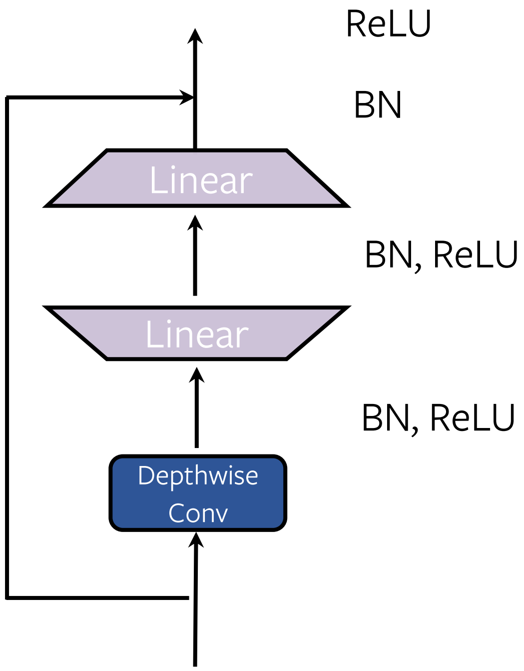
Larger kernel size helps, performance saturate at 7x7
Swin's choice of local window size is also 7 🤔

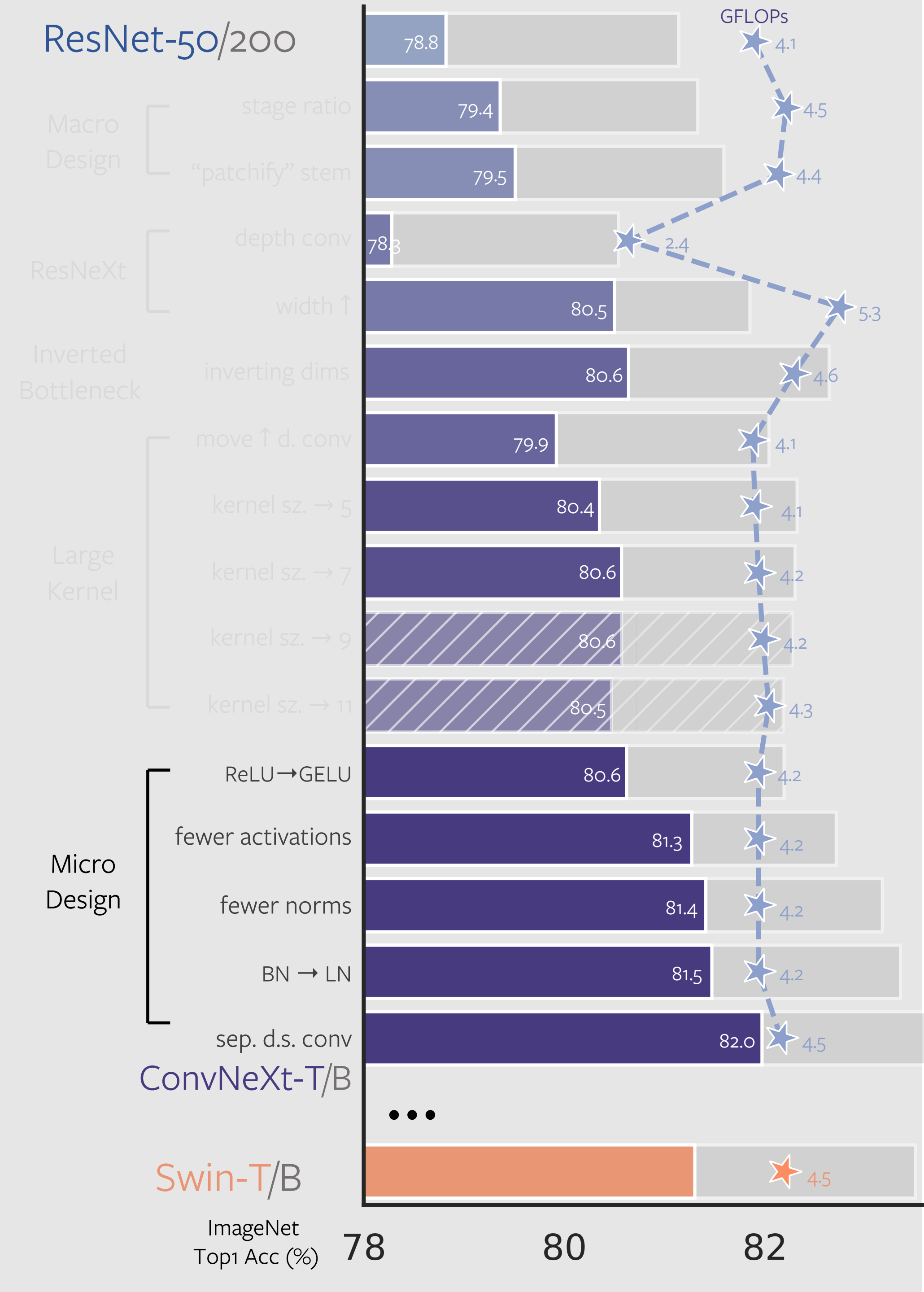


Micro Design

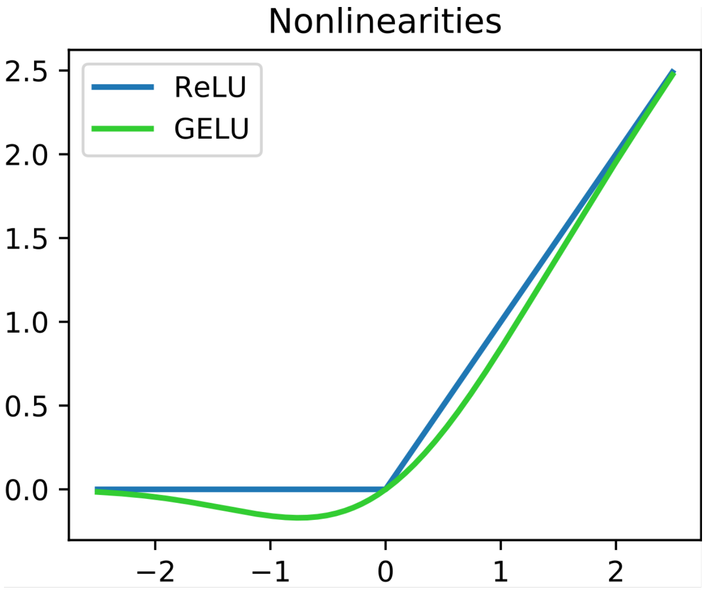


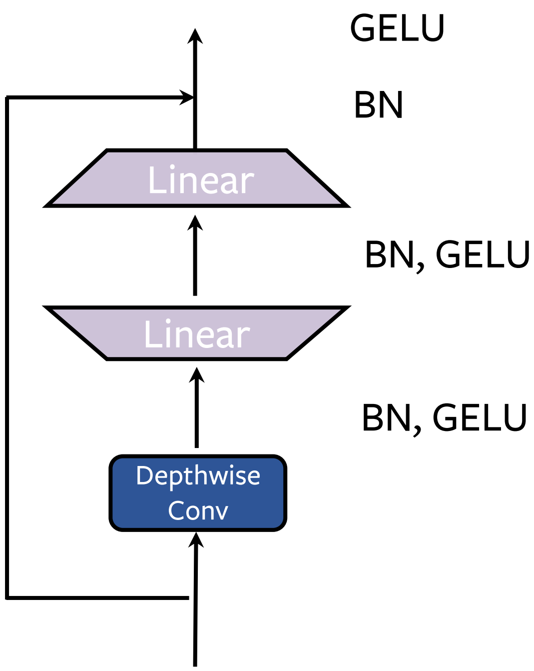
Transformer

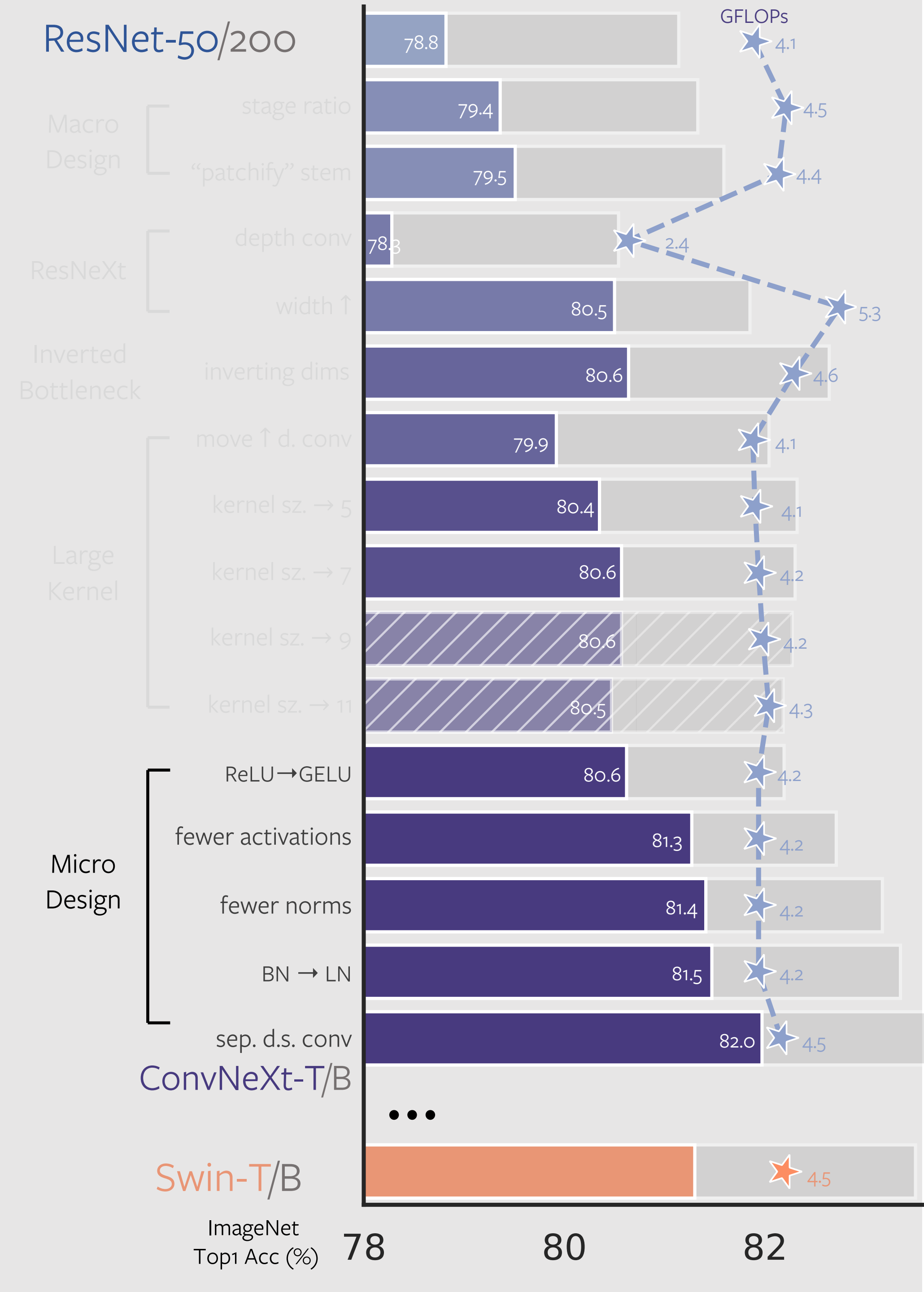




Activations

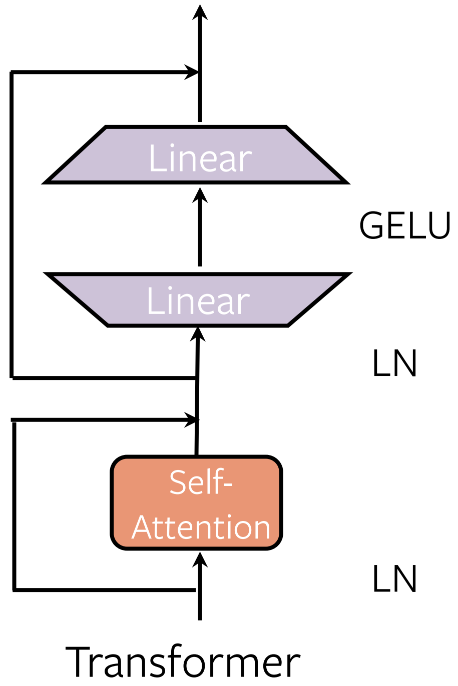


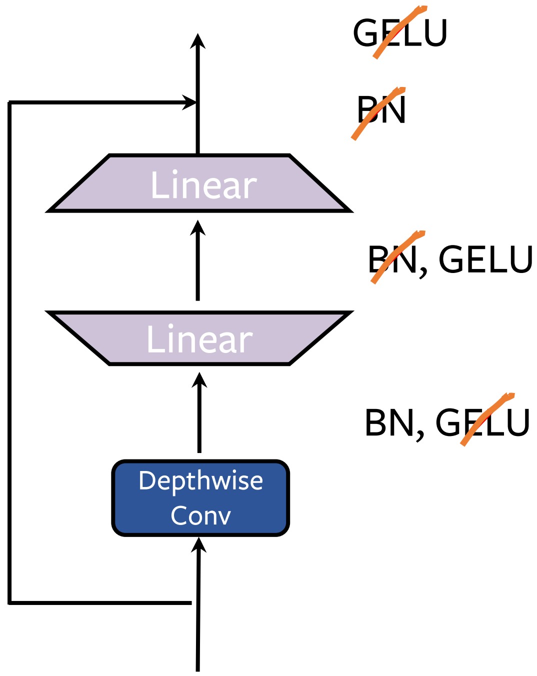




ReLU-> GELU

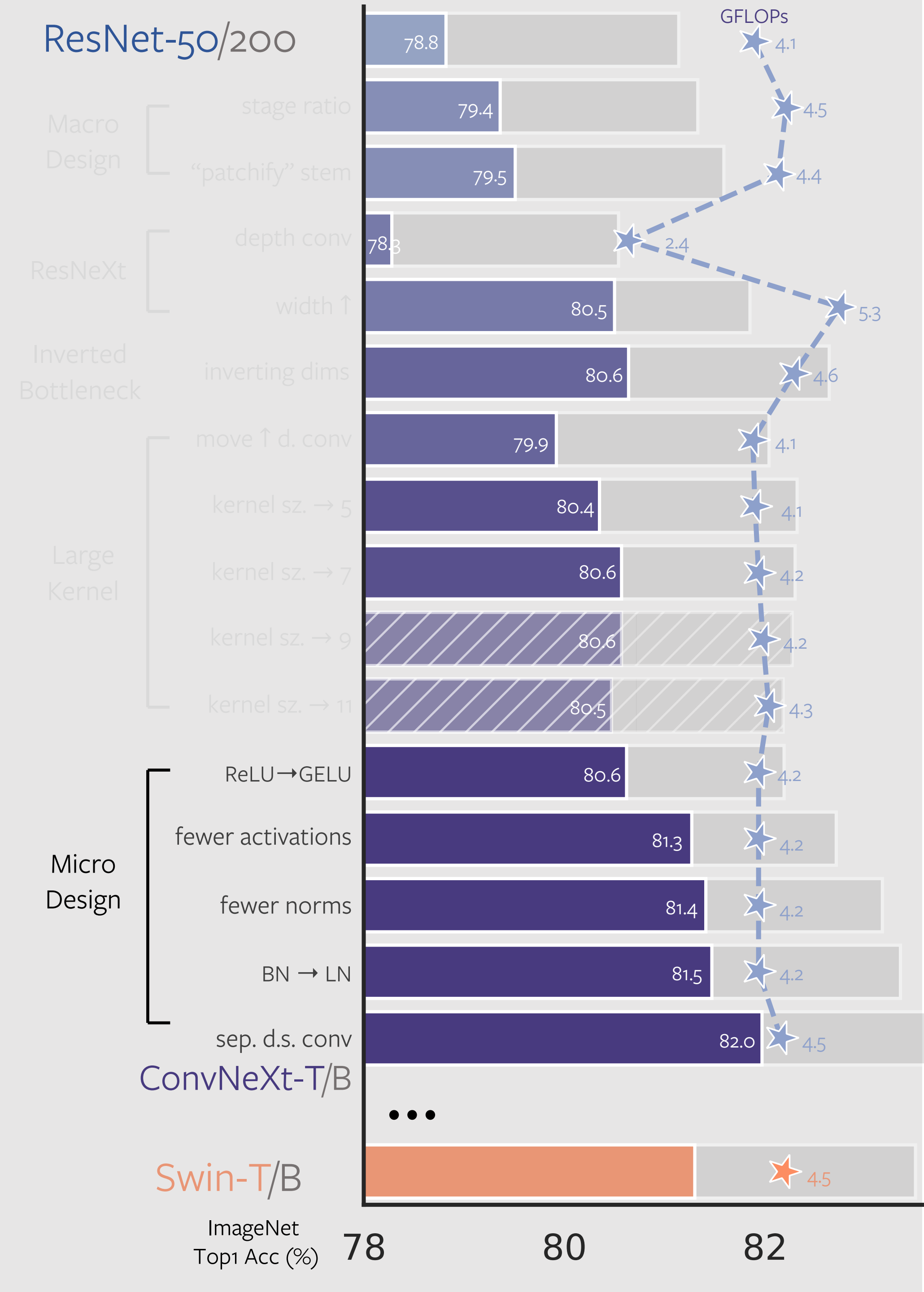
Fewer Activations/Norms



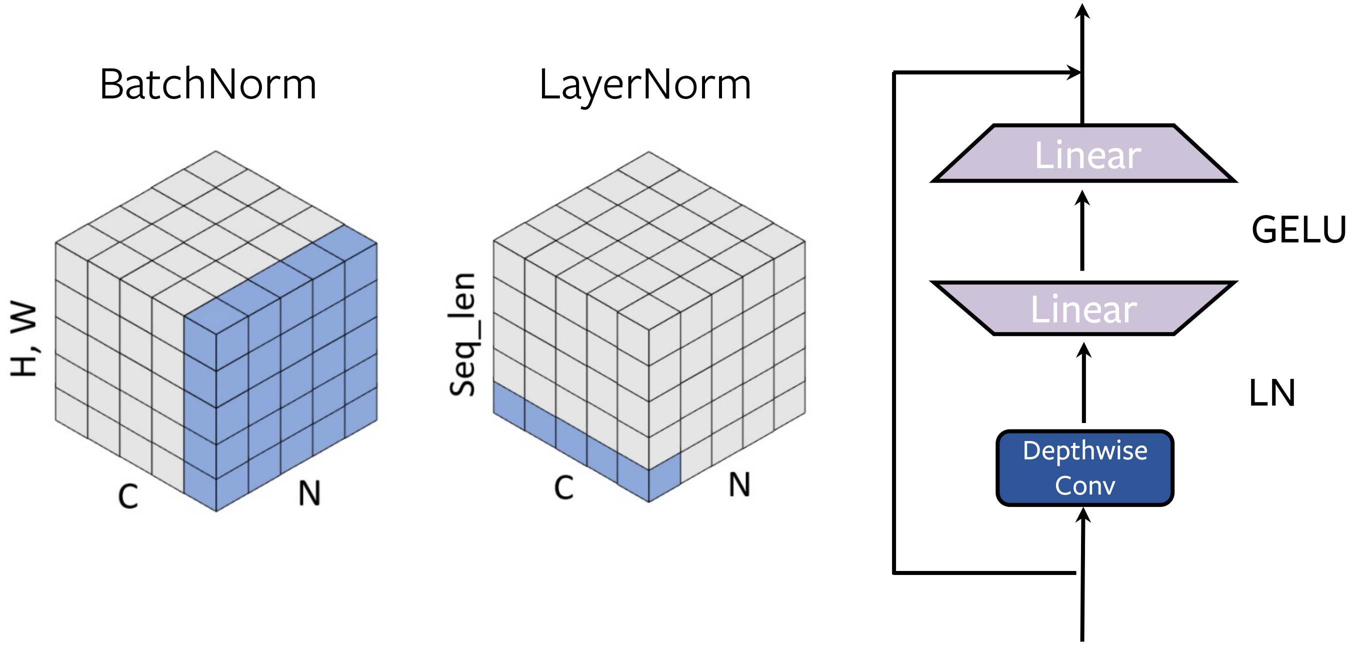


One norm, one activation is enough per block

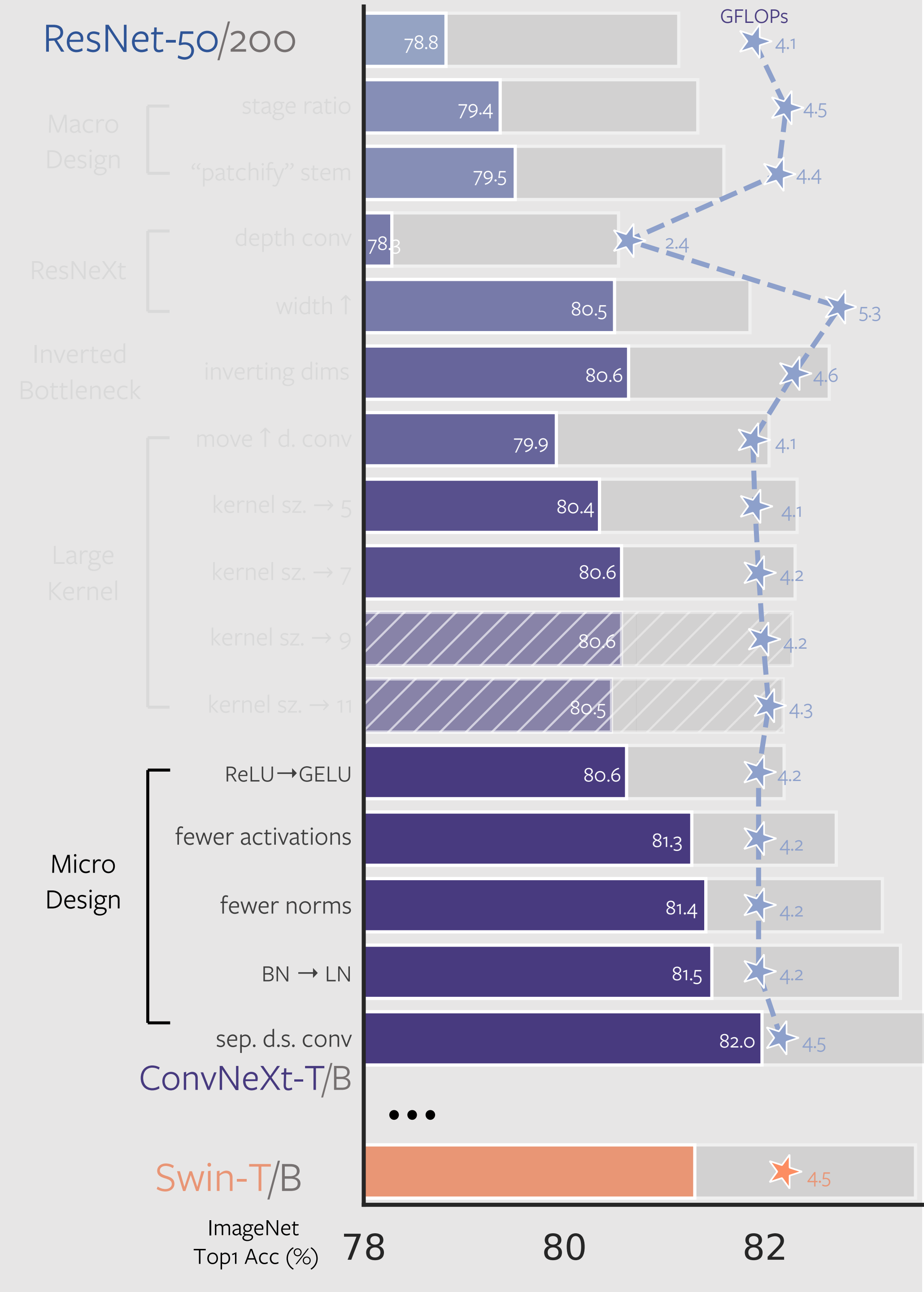
+0.8 without changing flops



Normalization



One LayerNorm is good enough.
Say 👋 to BatchNorm pitfalls!



Normalization

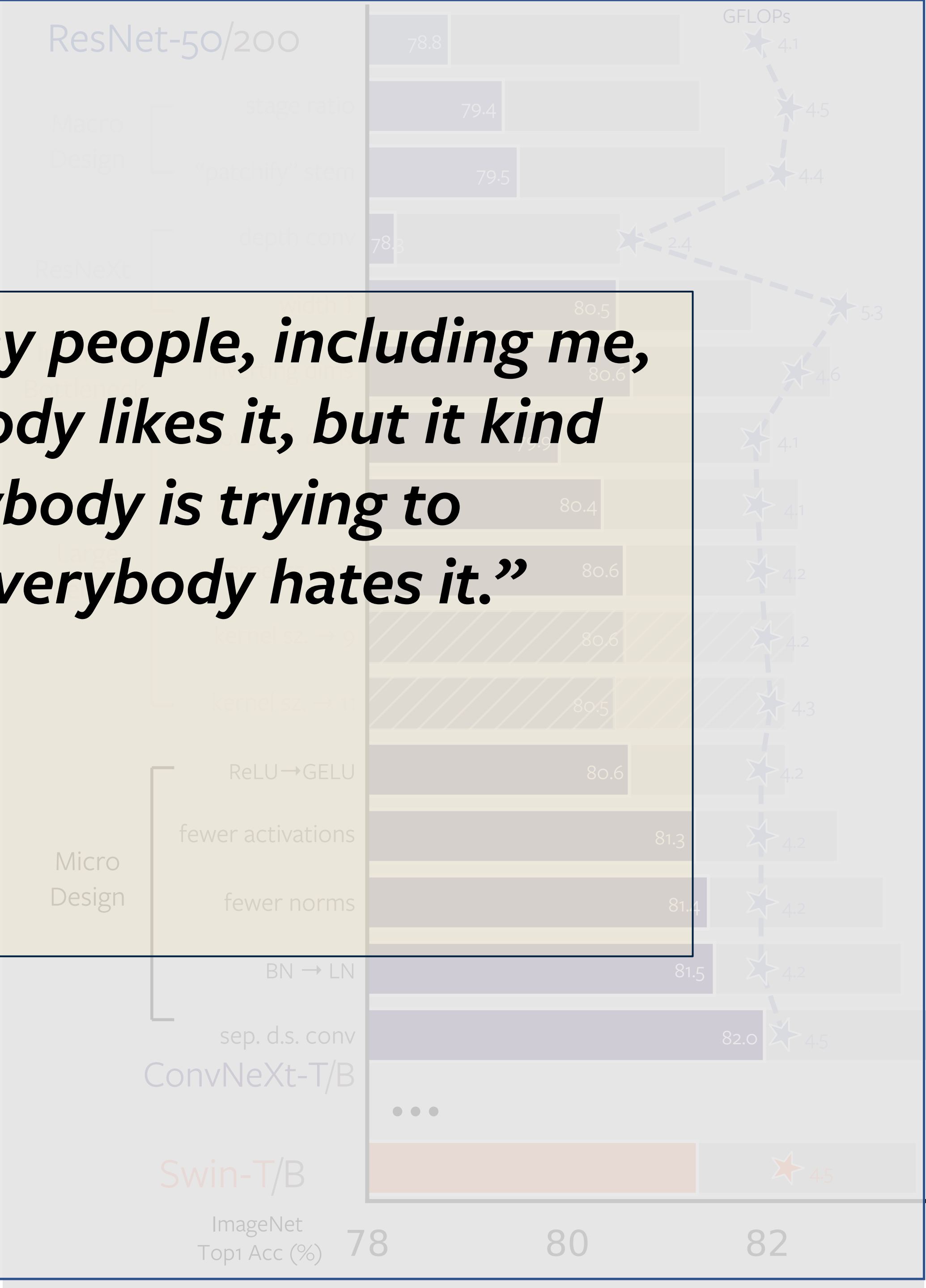
"Batch normalization in the mind of many people, including me, is a necessary evil. In the sense that nobody likes it, but it kind of works, so everybody uses it, but everybody is trying to replace it with something else because everybody hates it."

– Yann LeCun, 2018

" A very common source of bugs"

– CS231n 2019

One LayerNorm is good enough.
Say 👋 to BN pitfalls!

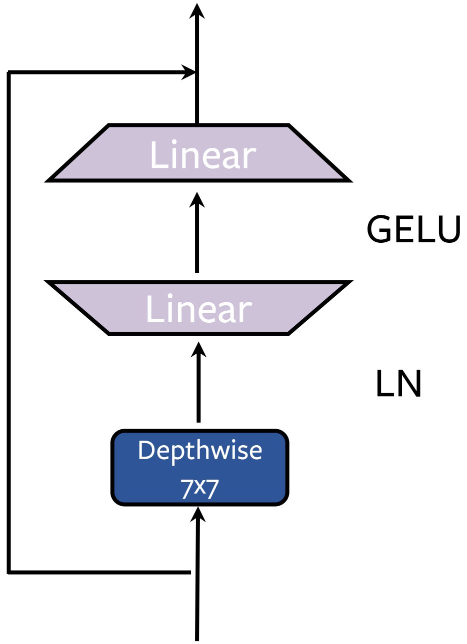


Separate downsampling layer

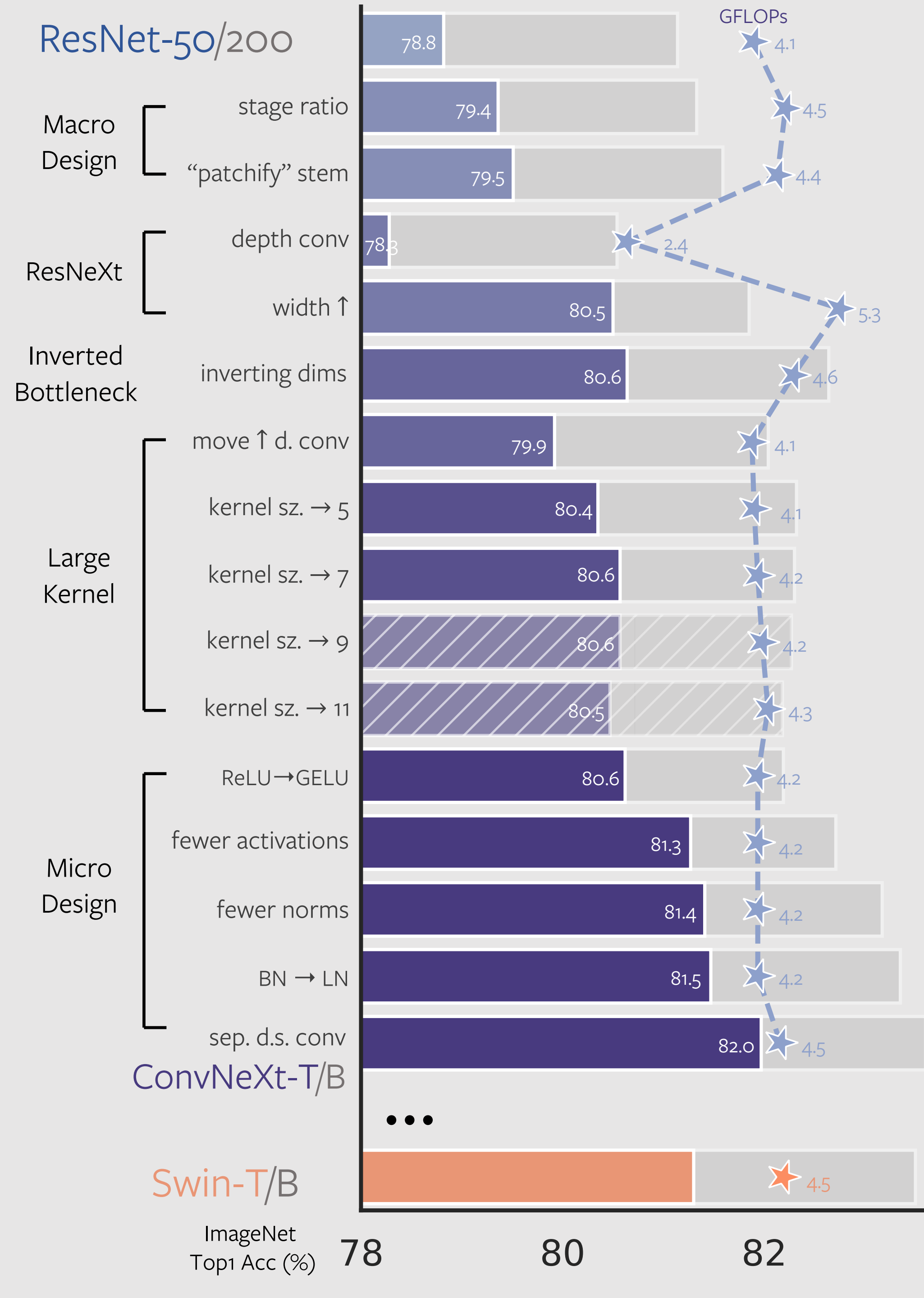
	downsp. rate (output size)	Swin-T	Swin-S	Swin-B	Swin-L
stage 1	4× (56×56)	concat 4×4, 96-d, LN	concat 4×4, 96-d, LN	concat 4×4, 128-d, LN	concat 4×4, 192-d, LN
		[win. sz. 7×7, dim 96, head 3] × 2	[win. sz. 7×7, dim 96, head 3] × 2	[win. sz. 7×7, dim 128, head 4] × 2	[win. sz. 7×7, dim 192, head 6] × 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.

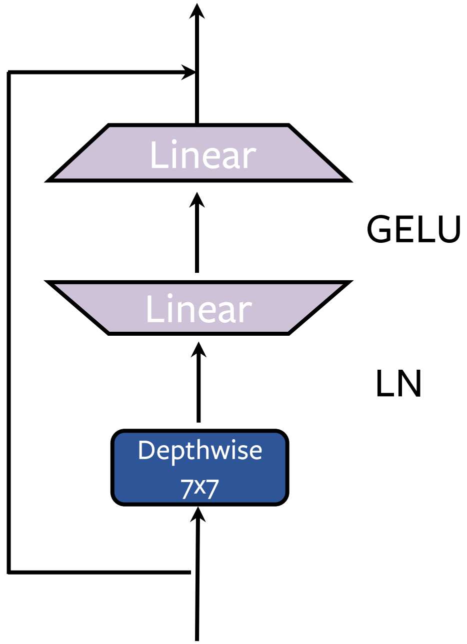
ConvNeXt block



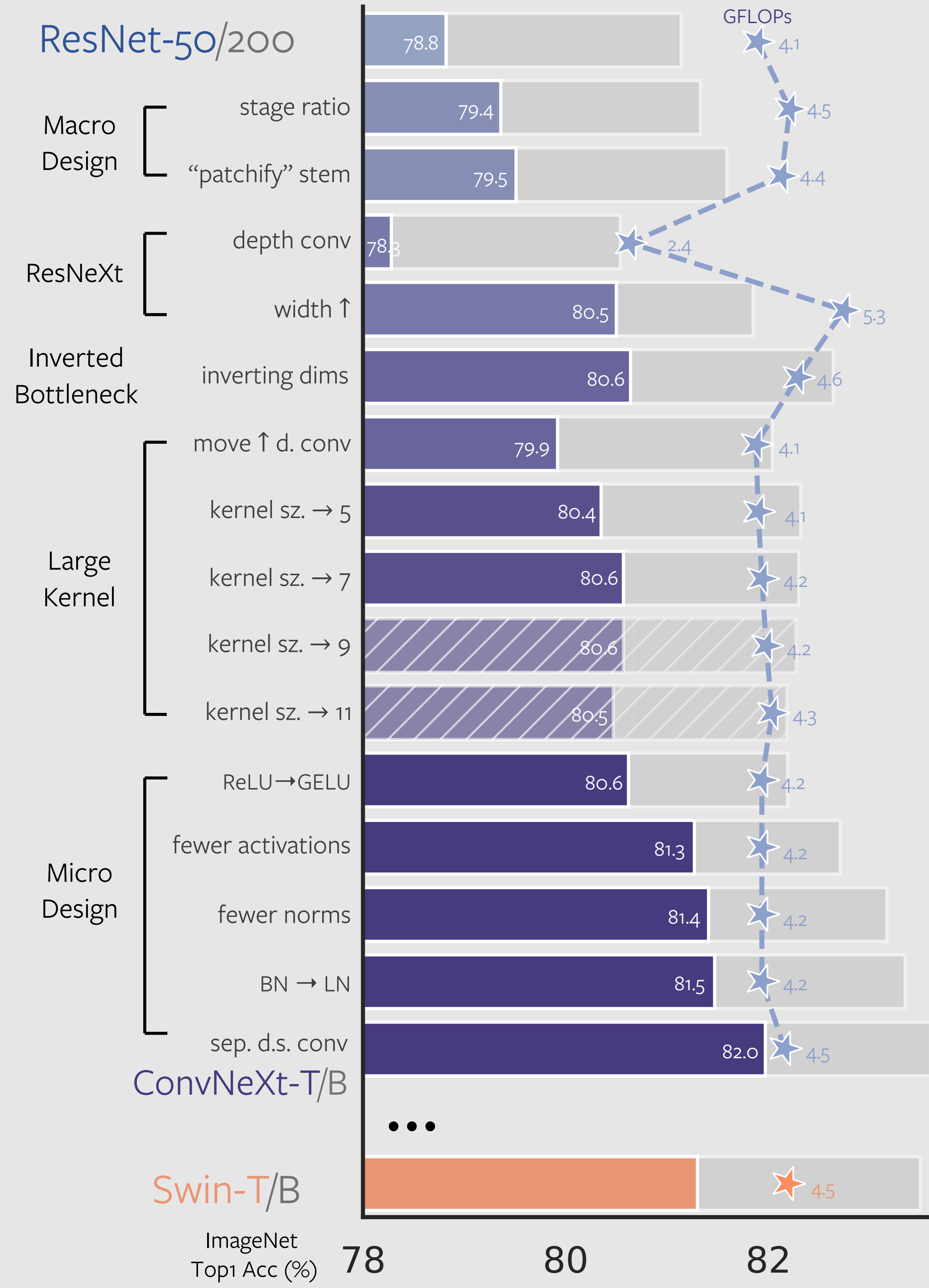
Minimal design:
as simple as possible, but not simpler.

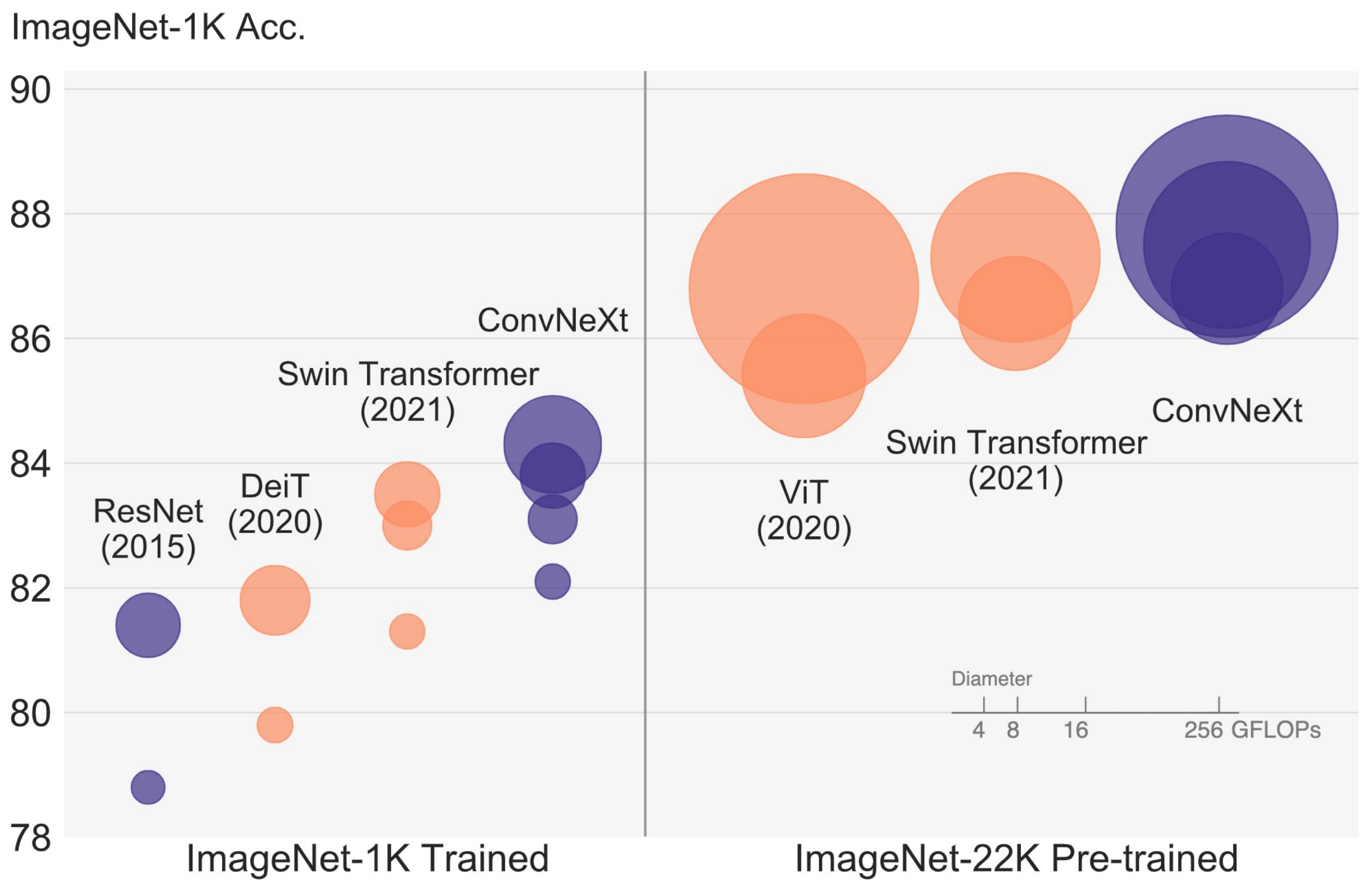


ConvNeXt block



1. 100 lines of code (vs 500+ for Swin Transformer)
2. No specialized module





Challenge the common belief:

- **Self-attention** is NOT the key for superior scalability.
- ConvNets can be scalable architectures.

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

Preliminary observation re: inference speed

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.

Robust models? Just scale it!

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**

Table 8. **Robustness evaluation of ConvNeXt**. We do not make use of any specialized modules or additional fine-tuning procedures.

Thank you!