Ch3. ConvNeXt

A ConvNet for the 2020s

Modernizing a ConvNet

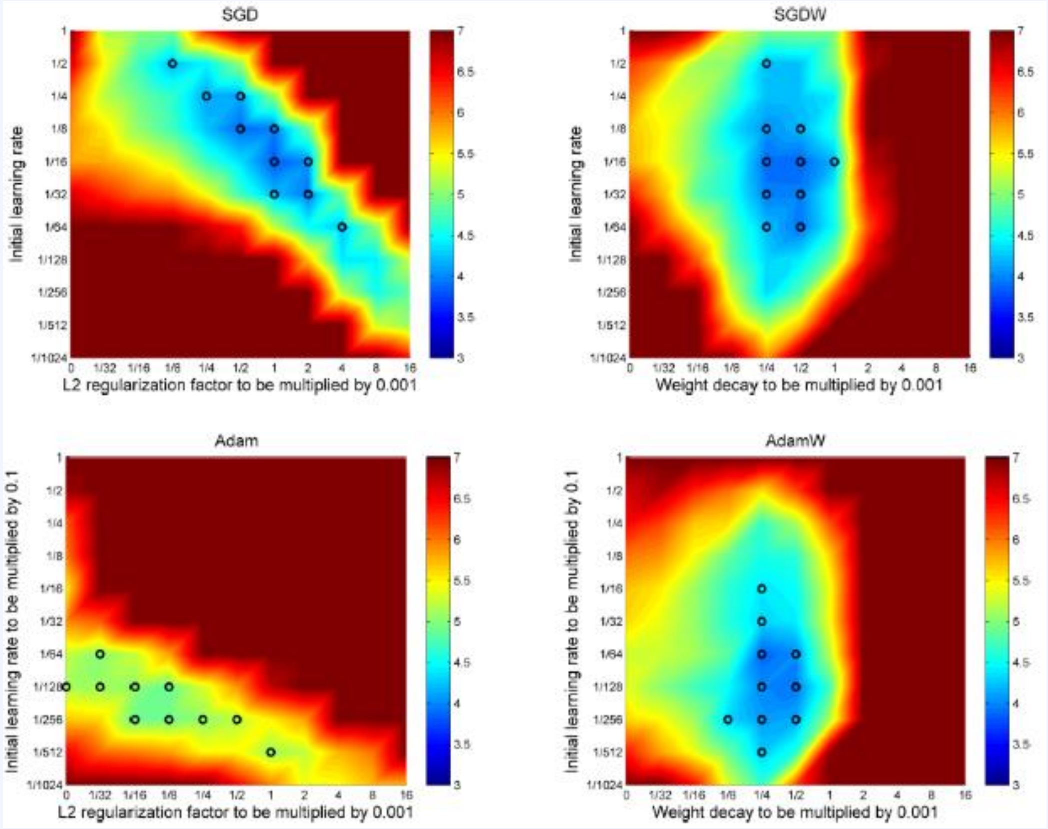
- Start model : ResNet-50
- ViT에 사용된 유사한 training technique들을 적용
- Marcro design
- ResNeXt
- Inverted bottleneck
- Large kernel size
- Varisous layer-wise micro designs

Marcro level

Training Techniques

- Training epochs: 300 (ResNet -90 epochs)
- Optimizer:AdamW

- Data augmentation
  - Mixup
  - Cutmix
  - RandAugment
  - Random Erasing
- Regularization
  - Stochastic Depth
  - Label Smoothing
- Accuracy : 76.1%→ 78.8%(+2.7%)



Change stage compute ratio

- Swin-T's 각 stage의 computation ratio :1:1:3:1
- 기존 ResNet50 stage 구성인 3,4,6,3을 Swin Transformer의 1:1:3:1 비율에 맞게 수정
  - ResNet50의 stage 구성:
    3,4,6,4→ 3,3,9,3
- Accuracy : 78.8%→ 79.4%(+0.6%)

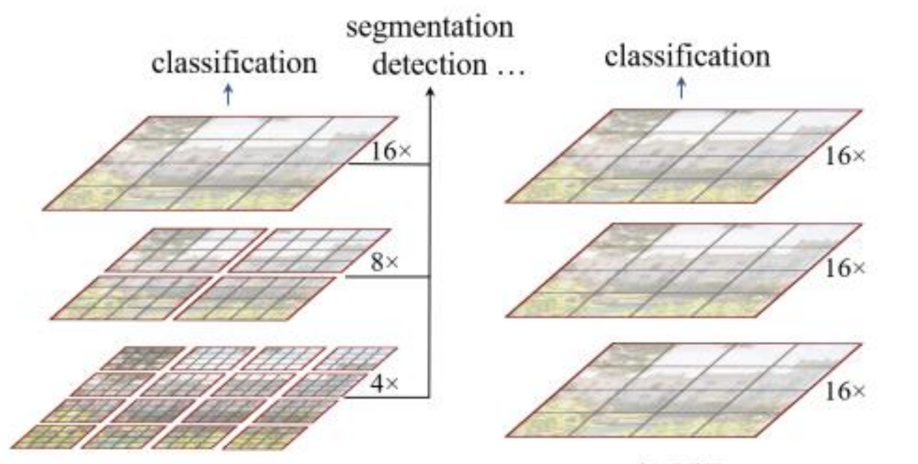
	output size	• ResNet-50	• ConvNeXt-T	∘ Swin-T
stem	56×56	7×7, 64, stride 2 3×3 max pool, stride 2	4×4, 96, stride 4	4×4, 96, stride 4
res2	56×56	[1×1, 64 3×3, 64 1×1, 256] × 3	[d7×7, 96 1×1, 384 1×1, 96] × 3	[1×1, 96×3 MSA, w7×7, H=3, rel. pos. 1×1, 96 1×1, 384 1×1, 96] × 2
res3	28×28	[1×1, 128 3×3, 128 1×1, 512] × 4	[d7×7, 192 1×1, 768 1×1, 192] × 3	[1×1, 192×3 MSA, w7×7, H=6, rel. pos. 1×1, 192 1×1, 768 1×1, 192] × 2
res4	14×14	[1×1, 256 3×3, 256 1×1, 1024] × 6	[d7×7, 384 1×1, 1536 1×1, 384] × 9	[1×1, 384×3 MSA, w7×7, H=12, rel. pos. 1×1, 384 1×1, 1536 1×1, 384] × 6
res5	7×7	[1×1, 512 3×3, 512 1×1, 2048] × 3	[d7×7, 768 1×1, 3072 1×1, 768] × 3	[1×1, 768×3 MSA, w7×7, H=24, rel. pos. 1×1, 768 1×1, 3072 1×1, 768] × 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

Table 9. **Detailed architecture specifications for ResNet-50, ConvNeXt-T and Swin-T.**

Changing Stem to “Patchify”

- ResNet stem cell : 7x7 Conv layer, stride 2, maxpooling, 4x downsampling
- Swin-T의 ‘patch merging’과 같이 4x4 kernel size, stride 4를 통해 patchify 수행
- Accuracy : 79.4%→ 79.5%(+0.1%)

Swin-T ViT

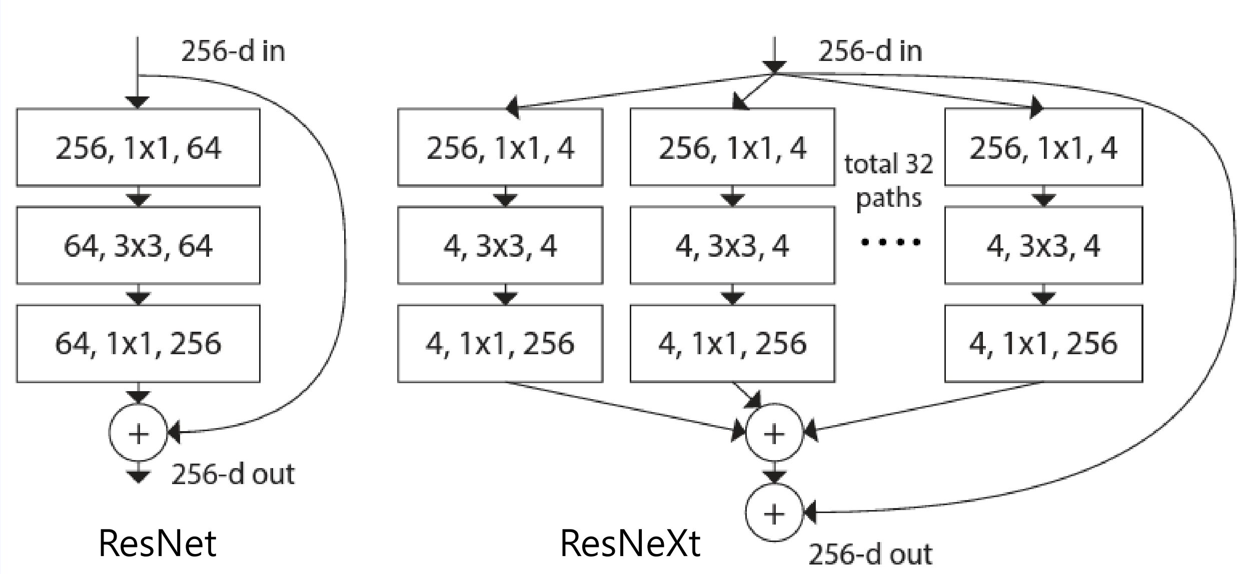


	output size	• ResNet-50	• ConvNeXt-T	○ Swin-T
stem	56×56	7×7, 64, stride 2 3×3 max pool, stride 2	4×4, 96, stride 4	4×4, 96, stride 4
res2	56×56	[1×1, 64 3×3, 64 1×1, 256] × 3	[d7×7, 96 1×1, 384 1×1, 96] × 3	[1×1, 96×3 MSA, w7×7, H=3, rel. pos. 1×1, 96 [1×1, 384 1×1, 96]] × 2
res3	28×28	[1×1, 128 3×3, 128 1×1, 512] × 4	[d7×7, 192 1×1, 768 1×1, 192] × 3	[1×1, 192×3 MSA, w7×7, H=6, rel. pos. 1×1, 192 [1×1, 768 1×1, 192]] × 2
res4	14×14	[1×1, 256 3×3, 256 1×1, 1024] × 6	[d7×7, 384 1×1, 1536 1×1, 384] × 9	[1×1, 384×3 MSA, w7×7, H=12, rel. pos. 1×1, 384 [1×1, 1536 1×1, 384]] × 6
res5	7×7	[1×1, 512 3×3, 512 1×1, 2048] × 3	[d7×7, 768 1×1, 3072 1×1, 768] × 3	[1×1, 768×3 MSA, w7×7, H=24, rel. pos. 1×1, 768 [1×1, 3072 1×1, 768]] × 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

Table 9. **Detailed architecture specifications for ResNet-50, ConvNeXt-T and Swin-T.**

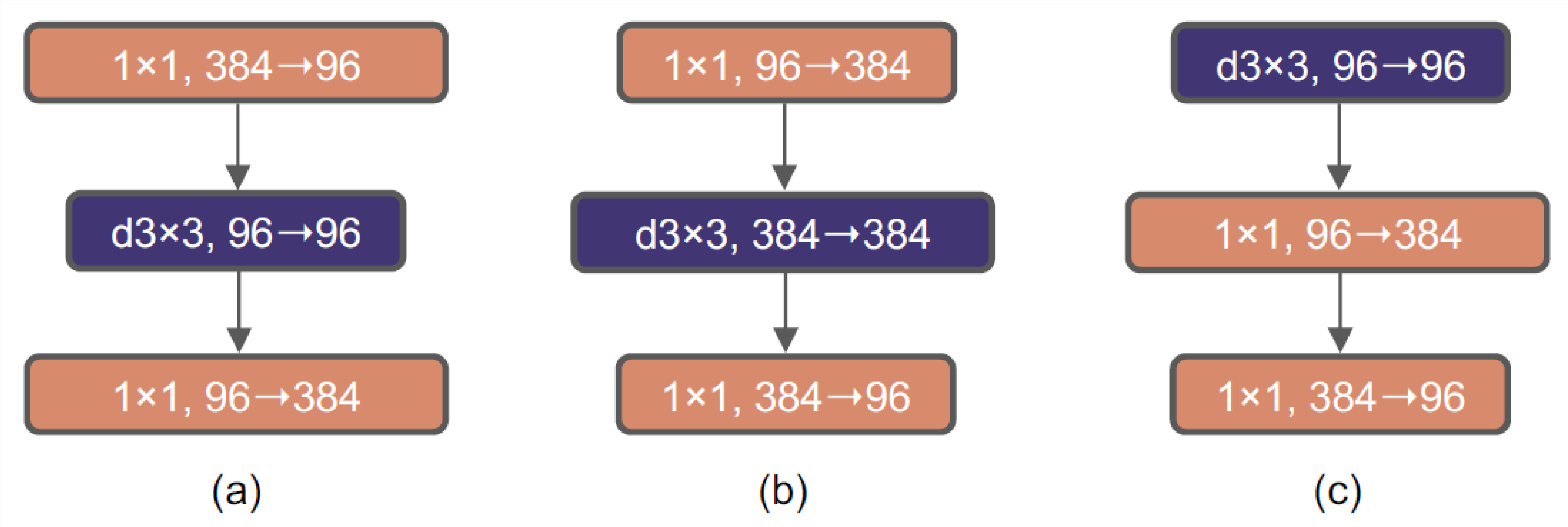
ResNeXt-ify

- ResNeXt에서 적용하는 depthwise seperable convolution을 사용하여 연산량(FLOPs)를 줄이고 capacity는 유지(width 또한 Swin-T와 동일하게 적용)
- FLOPS(5.3G)<-기대 연산량 4.5G
- Accuracy : 79.5%→ 80.5%(+1.0%)



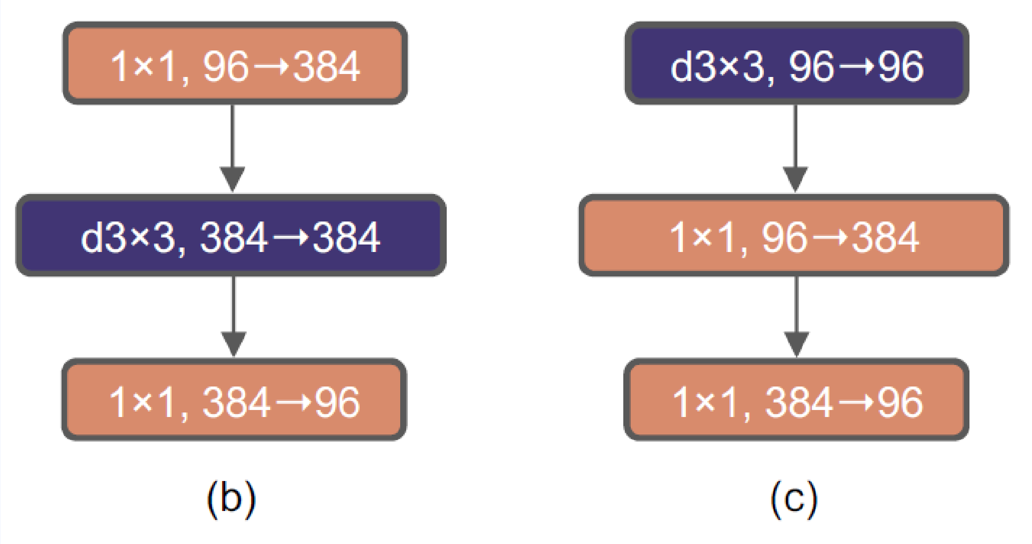
Inverted Bottleneck

- Inverted Bottleneck(MobileNetV2)
- Reduce network FLOPs → 4.6G
- Accuracy : 79.5%→ 80.5%(+1.0%)



Moving up Depthwise Conv Layer

- Depth-wise Conv = Swin-T의 Self-Attention
- Swin-T와 동일한 구조로 변경(그림b→그림c)
  - Transformer의 연산 순서: MSA→ MLP
- Reduce network FLOPs → 4.1G
- Accuracy : 80.5%→ 79.9%(-0.6%)



Increasing the Kernel Size

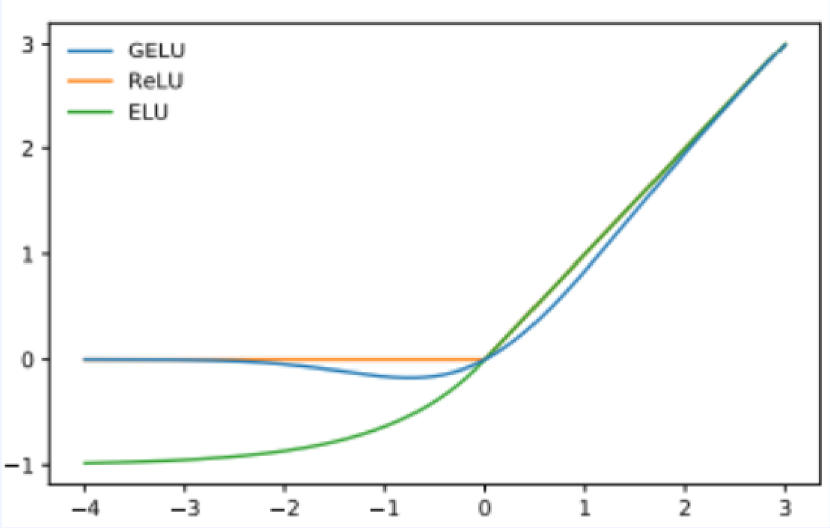
- VGG부터 3x3 kernel size가 일반적으로 사용
→Swin-T에 맞게 7x7로 수정(depth-wise convolution에 적용)
- Accuracy : 79.9%→ 80.6%(+0.7%)

	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

Micro level

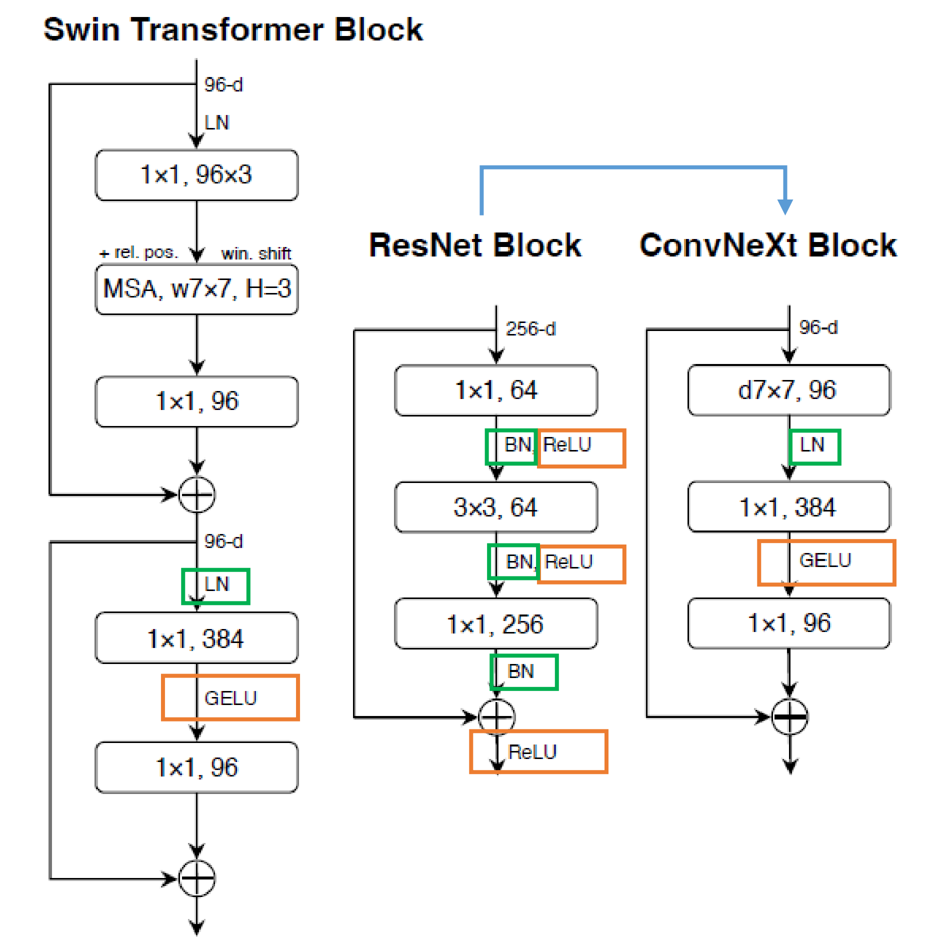
Replacing ReLU with GELU

- ReLU→ GELU(Gaussian Error Linear Unit)
- 기존 Transformer에도 ReLU가 사용되었지만 BERT 이후로 GELU로 모두 대체
- Accuracy : 80.6 %→ 80.6%(+0.0%)



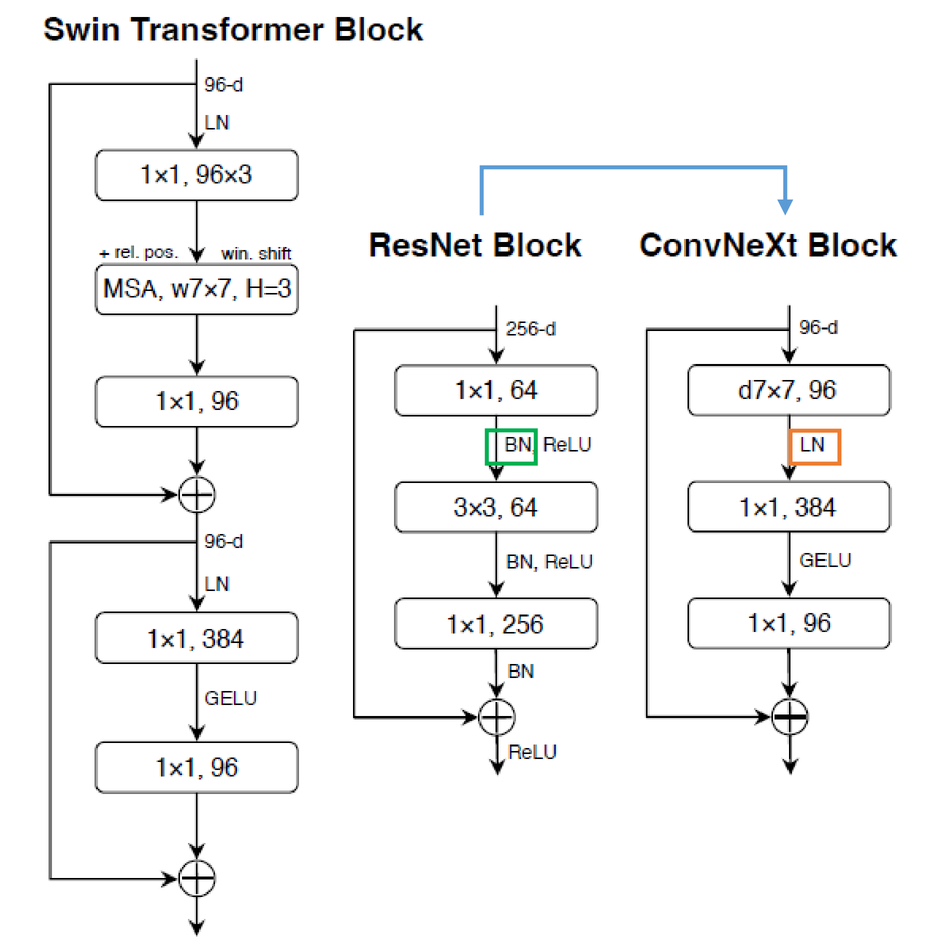
Fewer Activation Functions/ Normalization Layers

- Activation/Normalization을 매 layer 마다 적용
  → 한번 만 적용
- Accuracy : 80.6 %→ 81.4%(+0.8%)



Substituting BN with LN

- Batch Normalization→ Layer Normalization으로 변경
- Accuracy : 81.4 %→ 81.5%(+0.1%)



Separate Downsampling Layers

- 각 Stage마다 첫 블록에서 downsampling → stage와 stage 사이에 downsampling + normalization
- Accuracy : 81.5 %→ 82.0%(+0.5%)

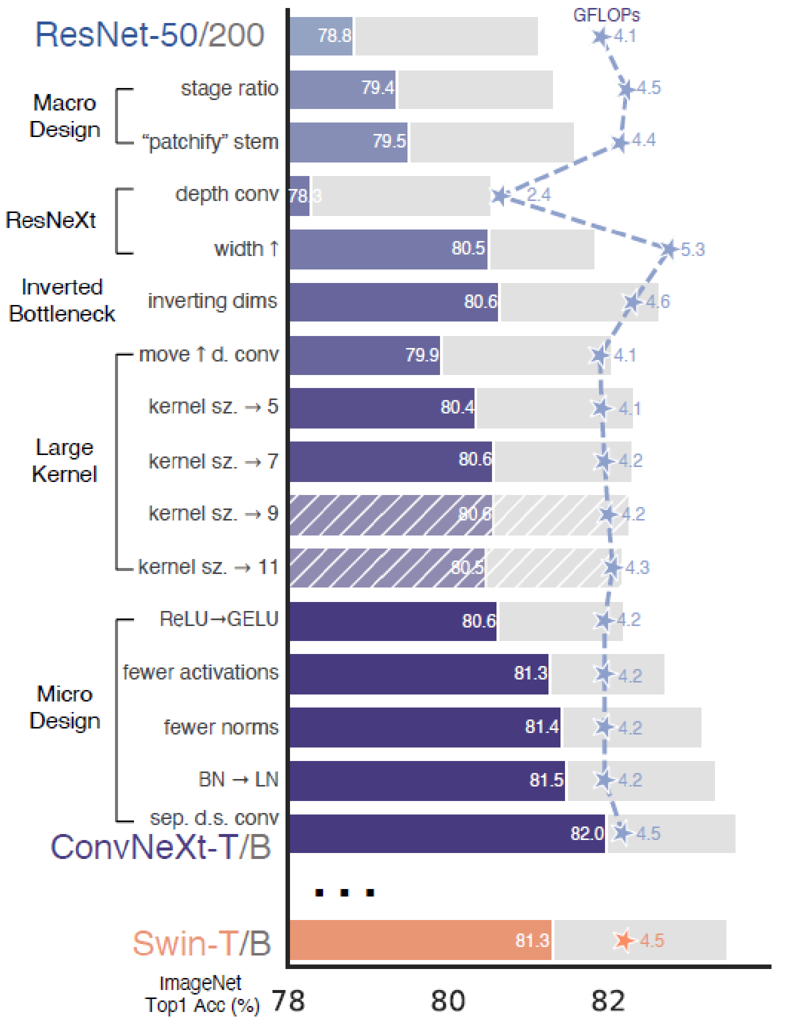
	output size	• ResNet-50	• ConvNeXt-T	○ Swin-T
stem	56×56	7×7, 64, stride 2 3×3 max pool, stride 2	4×4, 96, stride 4	4×4, 96, stride 4
res2	56×56	[1×1, 64 3×3, 64 1×1, 256] × 3	[d7×7, 96 1×1, 384 1×1, 96] × 3	[1×1, 96×3 MSA, w7×7, H=3, rel. pos. 1×1, 96 1×1, 384 1×1, 96] × 2
res3	28×28	[1×1, 128 3×3, 128 1×1, 512] × 4	[d7×7, 192 1×1, 768 1×1, 192] × 3	[1×1, 192×3 MSA, w7×7, H=6, rel. pos. 1×1, 192 1×1, 768 1×1, 192] × 2
res4	14×14	[1×1, 256 3×3, 256 1×1, 1024] × 6	[d7×7, 384 1×1, 1536 1×1, 384] × 9	[1×1, 384×3 MSA, w7×7, H=12, rel. pos. 1×1, 384 1×1, 1536 1×1, 384] × 6
res5	7×7	[1×1, 512 3×3, 512 1×1, 2048] × 3	[d7×7, 768 1×1, 3072 1×1, 768] × 3	[1×1, 768×3 MSA, w7×7, H=24, rel. pos. 1×1, 768 1×1, 3072 1×1, 768] × 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

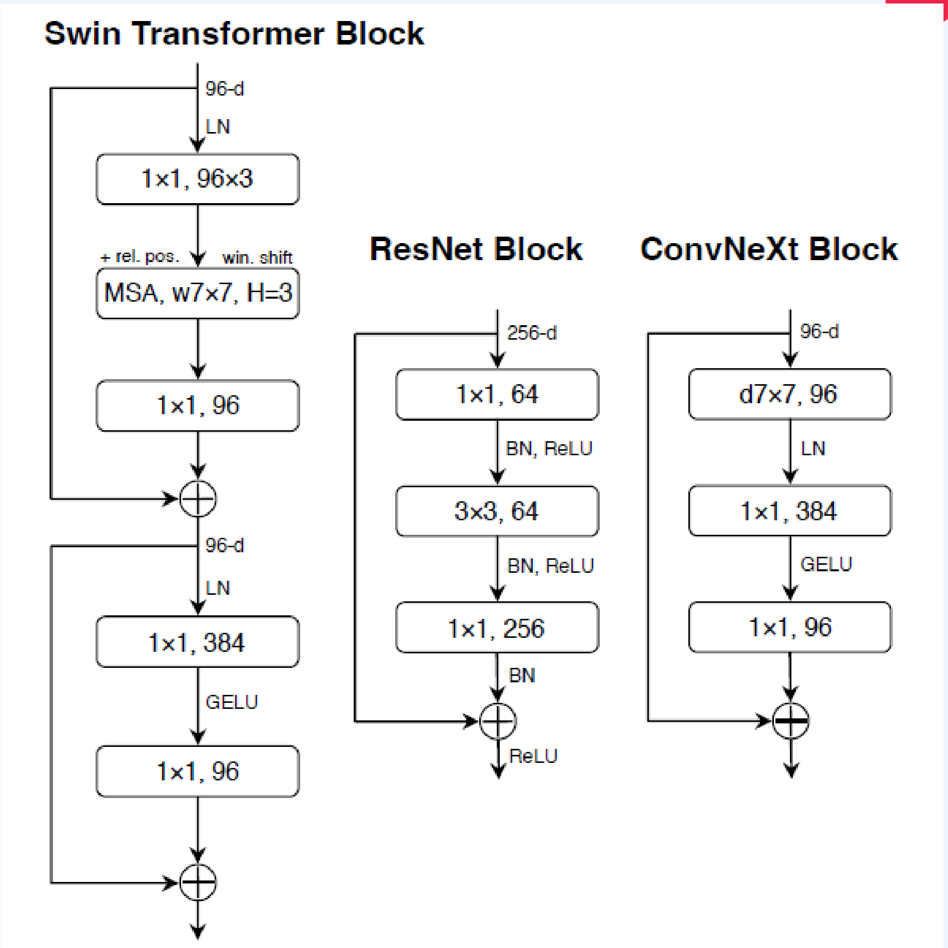
Downsampling

Downsampling

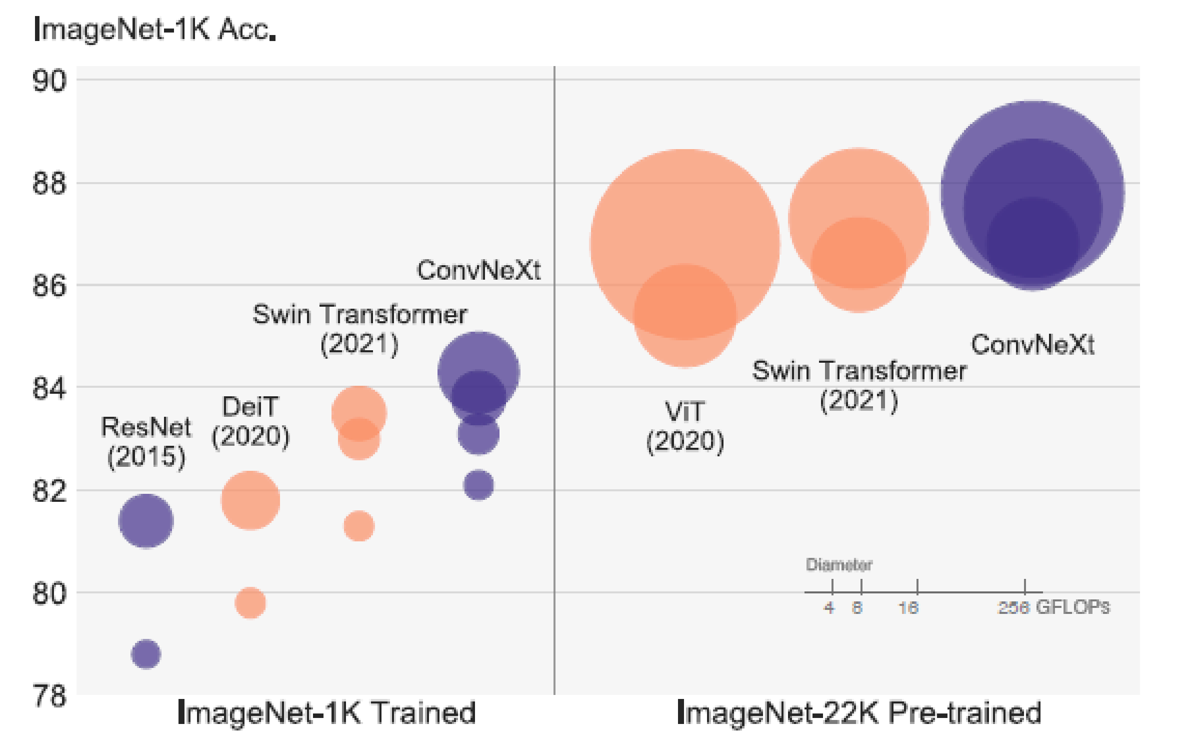
Downsampling

ConvNeXt Block Design





Compare


ImageNet-1K Acc.
90
88
86
84
82
80
78
ResNet (2015)
DeiT (2020)
Swin Transformer (2021)
ConvNeXt
ViT (2020)
Swin Transformer (2021)
ConvNeXt
Diameter
4 8 16 256 GFLOPs
ImageNet-1K Trained
ImageNet-22K Pre-trained

Summary

- Optimizer는 AdamW
- Residual Block을 Transformer Block 처럼 구성
- Convolution은 Depthwise convolution(width를 넓게)
- Kernel size는 7x7
- Activation과 normalization layer는 블록마다 적용
- Down-sampling은 Stage와 Stage 사이에 적용