(Summary on the last page 🐡 🐡 😉)

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

Abstract

The "Roaring 20s" of visual recognition began with the introduction of Vision Transformers (ViTs), which quickly superseded ConvNets as the state-of-the-art image classification model. A vanilla ViT, on the other hand, faces difficulties when applied to general computer vision tasks such as object detection and semantic segmentation. It is the hierarchical Transformers (e.g., Swin Transformers) that reintroduced several ConvNet priors, making Transformers practically viable as a generic vision backbone and demonstrating remarkable performance on a wide variety of vision tasks. However, the effectiveness of such hybrid approaches is still largely credited to the intrinsic superiority of Transformers, rather than the inherent inductive biases of convolutions. In this work, we reexamine the design spaces and test the limits of what a pure ConvNet can achieve. We gradually "modernize" a standard ResNet toward the design of a vision Transformer, and discover several key components that contribute to the performance difference along the way. The outcome of this exploration is a family of pure ConvNet models dubbed ConvNeXt. Constructed entirely from standard ConvNet modules, ConvNeXts compete favorably with Transformers in terms of accuracy and scalability, achieving 87.8% ImageNet top-1 accuracy and outperforming Swin Transformers on COCO detection and ADE20K segmentation, while maintaining the simplicity and efficiency of standard ConvNets.

1. Introduction

Looking back at the 2010s, the decade was marked by the monumental progress and impact of deep learning. The primary driver was the renaissance of neural networks, particularly convolutional neural networks (ConvNets). Through the decade, the field of visual recognition successfully shifted from engineering features to designing (ConvNet) architectures. Although the invention of back-propagation-trained ConvNets dates all the way back to the 1980s [39], it was not until late 2012 that we saw its true potential for visual feature learning. The introduction of AlexNet [37] precipitated the "ImageNet moment" [56], ushering in a new era of computer vision. The field has since evolved at a rapid speed. Representative ConvNets like VGGNet [61], Inceptions [64], ResNe(X)t [26, 82], DenseNet [33], MobileNet [32], EfficientNet [67] and RegNet [51] focused on different aspects of accuracy, efficiency and scalability, and popularized many useful design principles.

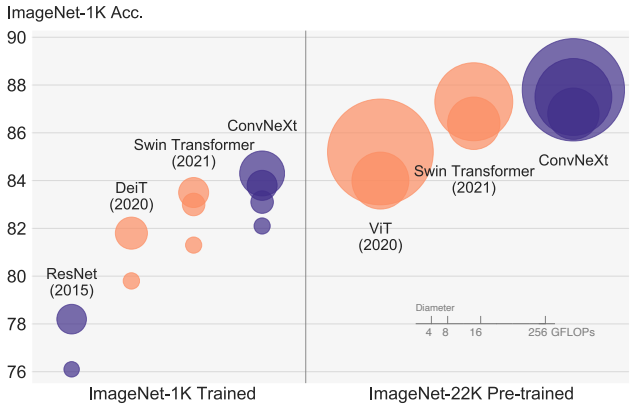

Figure 1. **ImageNet-1K classification** results for ● ConvNets and ○ vision Transformers. Each bubble's area is proportional to FLOPs of a variant in a model family. ImageNet-1K/22K models here take 224^2/384^2 images respectively. We demonstrate that a standard ConvNet model can achieve the same level of scalability as hierarchical vision Transformers while being much simpler in design.

The full dominance of ConvNets in computer vision was not a coincidence: in many application scenarios, a "sliding window" strategy is intrinsic to visual processing, particularly when working with high-resolution images. ConvNets have several built-in inductive biases that make them well-suited to a wide variety of computer vision applications. The most important one is translation equivariance, which is a desirable property for tasks like objection detection. ConvNets are also inherently efficient due to the fact that when used in a sliding-window manner, the computations are shared [59]. For many decades, this has been the default use of ConvNets, generally on limited object categories such as digits [40], faces [55, 71] and pedestrians [17, 60]. Entering the 2010s, the region-based detectors [21, 22, 25, 54] further elevated

What makes convnets attractive?

*Work done during an internship at Facebook AI Research.
†Corresponding author.

What makes transformers attractive?

Limitations with transformers when applied to computer vision

ConvNets to the position of being the fundamental building block in a visual recognition system.

Around the same time, the odyssey of neural network design for natural language processing (NLP) took a very different path, as the Transformers replaced recurrent neural networks to become the dominant backbone architecture. Despite the disparity in the task of interest between language and vision domains, the two streams surprisingly converged in the year 2020, as the introduction of Vision Transformers (ViT) completely altered the landscape of network architecture design. Except for the initial "patchify" layer, which splits an image into a sequence of patches, ViT introduces no image-specific inductive bias and makes minimal changes to the original NLP Transformers. One primary focus of ViT is on the scaling behavior: with the help of larger model and dataset sizes, Transformers can outperform standard ResNets by a significant margin. Those results on image classification tasks are inspiring, but computer vision is not limited to image classification. As discussed previously, solutions to numerous computer vision tasks in the past decade depended significantly on a sliding-window, fully-convolutional paradigm. Without the ConvNet inductive biases, a vanilla ViT model faces many challenges in being adopted as a generic vision backbone. The biggest challenge is ViT's global attention design, which has a quadratic complexity with respect to the input size. This might be acceptable for ImageNet classification, but quickly becomes intractable with higher-resolution inputs.

Hierarchical Transformers employ a hybrid approach to bridge this gap. For example, the "sliding window" strategy (*e.g.* attention within local windows) was reintroduced to Transformers, allowing them to behave more similarly to ConvNets. Swin Transformer [42] is a milestone work in this direction, demonstrating for the first time that Transformers can be adopted as a generic vision backbone and achieve state-of-the-art performance across a range of computer vision tasks beyond image classification. Swin Transformer's success and rapid adoption also revealed one thing: the essence of convolution is not becoming irrelevant; rather, it remains much desired and has never faded.

Under this perspective, many of the advancements of Transformers for computer vision have been aimed at bringing back convolutions. These attempts, however, come at a cost: a naive implementation of sliding window self-attention can be expensive [52]; with advanced approaches such as cyclic shifting [42], the speed can be optimized but the system becomes more sophisticated in design. On the other hand, it is almost ironic that a ConvNet already satisfies many of those desired properties, albeit in a straightforward, no-frills way. The only reason ConvNets appear to be losing steam is that (hierarchical) Transformers surpass them in many vision tasks, and the performance difference is usually attributed to the superior scaling behavior of Transformers, with multi-head self-attention being the key component.

Hybrid approaches

What "seems" odd while evaluation a hybrid models?

Unlike ConvNets, which have progressively improved over the last decade, the adoption of Vision Transformers was a step change. In recent literature, system-level comparisons (*e.g.* a Swin Transformer *vs.* a ResNet) are usually adopted when comparing the two. ConvNets and hierarchical vision Transformers become different and similar at the same time: they are both equipped with similar inductive biases, but differ significantly in the training procedure and macro/micro-level architecture design. In this work, we investigate the architectural distinctions between ConvNets and Transformers and try to identify the confounding variables when comparing the network performance. Our research is intended to bridge the gap between the pre-ViT and post-ViT eras for ConvNets, as well as to test the limits of what a pure ConvNet can achieve.

To do this, we start with a standard ResNet (*e.g.* ResNet-50) trained with an improved procedure. We gradually "modernize" the architecture to the construction of a hierarchical vision Transformer (*e.g.* Swin-T). Our exploration is directed by a key question: *How do design decisions in Transformers impact ConvNets' performance?* We discover several key components that contribute to the performance difference along the way. As a result, we propose a family of *pure ConvNets* dubbed ConvNeXt. We evaluate ConvNeXts on a variety of vision tasks such as ImageNet classification [15], object detection/segmentation on COCO [41], and semantic segmentation on ADE20K [87]. Surprisingly, ConvNeXts, constructed entirely from standard ConvNet modules, compete favorably with Transformers in terms of accuracy, scalability and robustness across all major benchmarks. ConvNeXt maintains the efficiency of standard ConvNets, and the fully-convolutional nature for both training and testing makes it extremely simple to implement.

Exploration

We hope the new observations and discussions can challenge some common beliefs and encourage people to rethink the importance of convolutions in computer vision.

2. Modernizing a ConvNet: a Roadmap

In this section, we provide a trajectory going from a ResNet to a ConvNet that bears a resemblance to Transformers. We consider two model sizes in terms of FLOPs, one is the ResNet-50 / Swin-T regime with FLOPs around 4.5×10^9 and the other being ResNet-200 / Swin-B regime which has FLOPs around 15.0×10^9. For simplicity, we will present the results with the ResNet-50 / Swin-T complexity models. The conclusions for higher capacity models are consistent and results can be found in Appendix C.

At a high level, our explorations are directed to investigate and follow different levels of designs from a Swin Transformer while maintaining the network's simplicity as a standard ConvNet. The roadmap of our exploration is as follows. Our starting point is a ResNet-50 model. We first

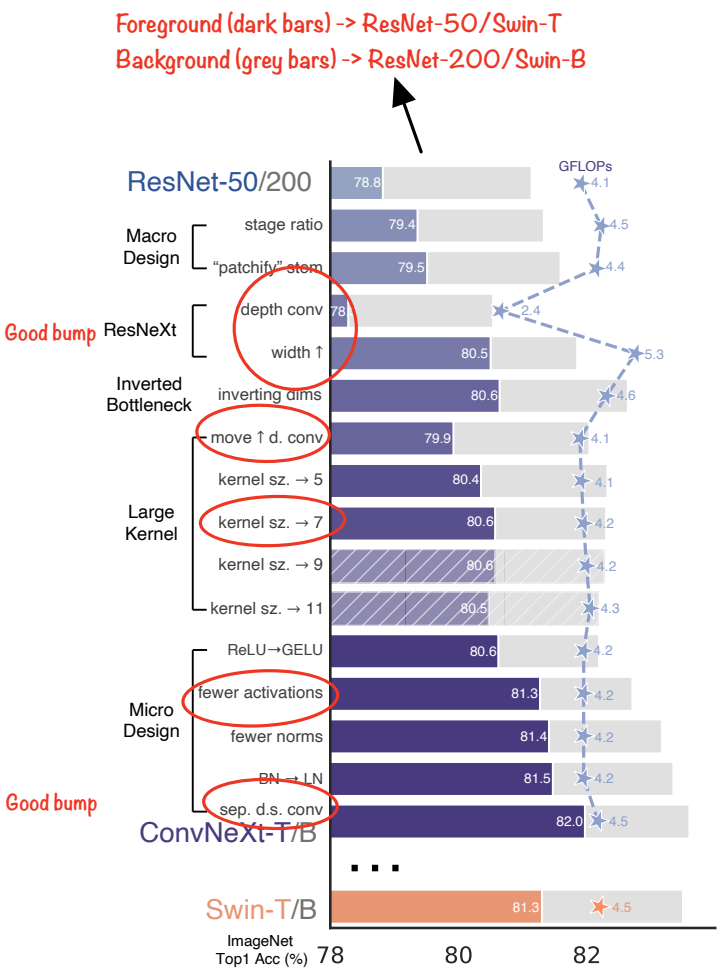

Figure 2. We modernize a standard ConvNet (ResNet) towards the design of a hierarchical vision Transformer (Swin), without introducing any attention-based modules. The foreground bars are model accuracies in the ResNet-50/Swin-T FLOP regime; results for the ResNet-200/Swin-B regime are shown with the gray bars. A hatched bar means the modification is not adopted. Detailed results for both regimes are in the appendix. Many Transformer architectural choices can be incorporated in a ConvNet, and they lead to increasingly better performance. In the end, our pure ConvNet model, named ConvNeXt, can outperform the Swin Transformer.

train it with similar training techniques used to train vision Transformers and obtain much improved results compared to the original ResNet-50. This will be our baseline. We then study a series of design decisions which we summarized as 1) macro design, 2) ResNeXt, 3) inverted bottleneck, 4) large kernel size, and 5) various layer-wise micro designs. In Figure 2, we show the procedure and the results we are able to achieve with each step of the "network modernization". Since network complexity is closely correlated with the final performance, the FLOPs are roughly controlled over the course of the exploration, though at intermediate steps the FLOPs might be higher or lower than the reference models. All models are trained and evaluated on ImageNet-1K.

2.1. Training Techniques

Apart from the design of the network architecture, the training procedure also affects the ultimate performance. Not

Training techniques used in the new architectures

only did vision Transformers bring a new set of modules and architectural design decisions, but they also introduced different training techniques (*e.g.* AdamW optimizer) to vision. This pertains mostly to the optimization strategy and associated hyper-parameter settings. Thus, the first step of our exploration is to train a baseline model with the vision Transformer training procedure, in this case, ResNet-50/200. A recent paper [76] demonstrates how a set of modern training techniques can significantly enhance the performance of a simple ResNet-50 model. In our study, we use a training recipe that is close to DeiT's [68] and Swin Transformer's [42]. The training is extended to 300 epochs from the original 90 epochs for ResNets. We use the AdamW optimizer [43], data augmentation techniques such as Mixup [85], Cutmix [84], RandAugment [12], Random Erasing [86], and regularization schemes including Stochastic Depth [33] and Label Smoothing [65]. The complete set of hyper-parameters we use can be found in Appendix A.1. By itself, this enhanced training recipe increased the performance of the ResNet-50 model from 76.1% [1] to 78.8% (+2.7%), implying that a significant portion of the performance difference between traditional ConvNets and vision Transformers may be due to the training techniques. We will use this fixed training recipe with the same hyperparameters throughout the "modernization" process. Each reported accuracy on the ResNet-50 regime is an average obtained from training with three different random seeds.

2.2. Macro Design

We now analyze Swin Transformers' macro network design. Swin Transformers follow ConvNets [26, 62] to use a multi-stage design, where each stage has a different feature map resolution. There are two interesting design considerations: the stage compute ratio, and the "stem cell" structure.

Changing stage compute ratio. The original design of the computation distribution across stages in ResNet was largely empirical. The heavy "res4" stage was meant to be compatible with downstream tasks like object detection, where a detector head operates on the 14×14 feature plane. Swin-T, on the other hand, followed the same principle but with a slightly different stage compute ratio of 1:1:3:1. For larger Swin Transformers, the ratio is 1:1:9:1. Following the design, we adjust the number of blocks in each stage from (3, 4, 6, 3) in ResNet-50 to (3, 3, 9, s3), which also aligns the FLOPs with Swin-T. This improves the model accuracy from 78.8% to 79.4%. Notably, researchers have thoroughly investigated the distribution of computation [50, 51], and a more optimal design is likely to exist.

Why the old design was made so?

From now on, we will use this stage compute ratio.

Changing stem to "Patchify". Typically, the stem cell design is concerned with how the input images will be processed at the network's beginning. Due to the redundancy

Why and how we aggressively downsample images in different types of models and how can we improve on top of them?

inherent in natural images, a common stem cell will aggressively downsample the input images to an appropriate feature map size in both standard ConvNets and vision Transformers. The stem cell in standard ResNet contains a 7×7 convolution layer with stride 2, followed by a max pool, which results in a 4× downsampling of the input images. In vision Transformers, a more aggressive "patchify" strategy is used as the stem cell, which corresponds to a large kernel size (e.g. kernel size = 14 or 16) and non-overlapping convolution. Swin Transformer uses a similar "patchify" layer, but with a smaller patch size of 4 to accommodate the architecture's multi-stage design. We replace the ResNet-style stem cell with a patchify layer implemented using a 4×4, stride 4 convolutional layer. The accuracy has changed from 79.4% to 79.5%. This suggests that the stem cell in a ResNet may be substituted with a simpler "patchify" layer à la ViT which will result in similar performance.

We will use the "patchify stem" (4×4 non-overlapping convolution) in the network.

2.3. ResNeXt-ify

In this part, we attempt to adopt the idea of ResNeXt [82], which has a better FLOPs/accuracy trade-off than a vanilla ResNet. The core component is grouped convolution, where the convolutional filters are separated into different groups. At a high level, ResNeXt's guiding principle is to "use more groups, expand width". More precisely, ResNeXt employs grouped convolution for the 3×3 conv layer in a bottleneck block. As this significantly reduces the FLOPs, the network width is expanded to compensate for the capacity loss.

In our case we use depthwise convolution, a special case of grouped convolution where the number of groups equals the number of channels. Depthwise conv has been popularized by MobileNet [32] and Xception [9]. We note that depthwise convolution is similar to the weighted sum operation in self-attention, which operates on a per-channel basis, *i.e.*, only mixing information in the spatial dimension. The use of depthwise convolution effectively reduces the network FLOPs and, as expected, the accuracy. Following the strategy proposed in ResNeXt, we increase the network width to the same number of channels as Swin-T's (from 64 to 96). This brings the network performance to 80.5% with increased FLOPs (5.3G).

We will now employ the ResNeXt design.

2.4. Inverted Bottleneck

One important design in every Transformer block is that it creates an inverted bottleneck, *i.e.*, the hidden dimension of the MLP block is four times wider than the input dimension (see Figure 4). Interestingly, this Transformer design is connected to the inverted bottleneck design with an expansion ratio of 4 used in ConvNets. The idea was popularized by MobileNetV2 [58], and has subsequently gained traction in several advanced ConvNet architectures [66, 67].

What kind of op is depthwise convolution in convnets similar to ?
How does inverted bottlenecks are formed in transformers?

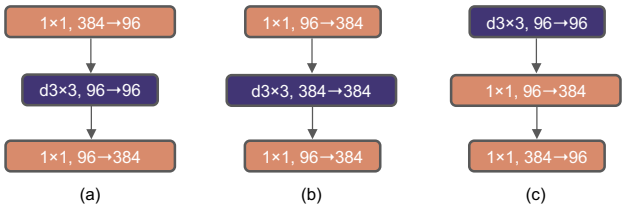

Figure 3. **Block modifications and resulted specifications. (a)** is a ResNeXt block; in **(b)** we create an inverted bottleneck block and in **(c)** the position of the spatial depthwise conv layer is moved up.

Here we explore the inverted bottleneck design. Figure 3 (a) to (b) illustrate the configurations. Despite the increased FLOPs for the depthwise convolution layer, this change reduces the whole network FLOPs to 4.6G, due to the significant FLOPs reduction in the downsampling residual blocks' shortcut 1×1 conv layer. Interestingly, this results in slightly improved performance (80.5% to 80.6%). In the ResNet-200 / Swin-B regime, this step brings even more gain (81.9% to 82.6%) also with reduced FLOPs.

We will now use inverted bottlenecks.

2.5. Large Kernel Sizes

In this part of the exploration, we focus on the behavior of large convolutional kernels. One of the most distinguishing aspects of vision Transformers is their non-local self-attention, which enables each layer to have a global receptive field. While large kernel sizes have been used in the past with ConvNets [37, 64], the gold standard (popularized by VGGNet [62]) is to stack small kernel-sized (3×3) conv layers, which have efficient hardware implementations on modern GPUs [38]. Although Swin Transformers reintroduced the local window to the self-attention block, the window size is at least 7×7, significantly larger than the ResNe(X)t kernel size of 3×3. Here we revisit the use of large kernel-sized convolutions for ConvNets.

Why self-attention is so special?

Moving up depthwise conv layer. To explore large kernels, one prerequisite is to move up the position of the depthwise conv layer (Figure 3 (b) to (c)). That is a design decision also evident in Transformers: the MSA block is placed prior to the MLP layers. As we have an inverted bottleneck block, this is a natural design choice — the complex/inefficient modules (MSA, large-kernel conv) will have fewer channels, while the efficient, dense 1×1 layers will do the heavy lifting. This intermediate step reduces the FLOPs to 4.1G, resulting in a temporary performance degradation to 79.9%.

Increasing the kernel size. With all of these preparations, the benefit of adopting larger kernel-sized convolutions is significant. We experimented with several kernel sizes, including 3, 5, 7, 9, and 11. The network's performance increases from 79.9% (3×3) to 80.6% (7×7), while the network's FLOPs stay roughly the same. Additionally, we observe that

Why bigger kernel sizes?
What adjustments can we need make for incorporating bigger kernels? Is there any benefit of doing so?

the benefit of larger kernel sizes reaches a saturation point at 7×7. We verified this behavior in the large capacity model too: a ResNet-200 regime model does not exhibit further gain when we increase the kernel size beyond 7×7.

We will use 7×7 depthwise conv in each block.

At this point, we have concluded our examination of network architectures on a macro scale. Intriguingly, a significant portion of the design choices taken in a vision Transformer may be mapped to ConvNet instantiations.

2.6. Micro Design

In this section, we investigate several other architectural differences at a micro scale — most of the explorations here are done at the layer level, focusing on specific choices of activation functions and normalization layers.

Replacing ReLU with GELU One discrepancy between NLP and vision architectures is the specifics of which activation functions to use. Numerous activation functions have been developed over time, but the Rectified Linear Unit (ReLU) [46] is still extensively used in ConvNets due to its simplicity and efficiency. ReLU is also used as an activation function in the original Transformer paper [72]. The Gaussian Error Linear Unit, or GELU [30], which can be thought of as a smoother variant of ReLU, is utilized in the most advanced Transformers, including Google's BERT [16] and OpenAI's GPT-2 [49], and, most recently, ViTs. We find that ReLU can be substituted with GELU in our ConvNet too, although the accuracy stays unchanged (80.6%).

Fewer activation functions. One minor distinction between a Transformer and a ResNet block is that Transformers have fewer activation functions. Consider a Transformer block with key/query/value linear embedding layers, the projection layer, and two linear layers in an MLP block. There is only one activation function present in the MLP block. In comparison, it is common practice to append an activation function to each convolutional layer, including the 1×1 convs. Here we examine how performance changes when we stick to the same strategy. As depicted in Figure 4, we eliminate all GELU layers from the residual block except for one between two 1×1 layers, replicating the style of a Transformer block. This process improves the result by 0.7% to 81.3%, practically matching the performance of Swin-T.

We will now use a single GELU activation in each block.

Fewer normalization layers. Transformer blocks usually have fewer normalization layers as well. Here we remove two BatchNorm (BN) layers, leaving only one BN layer before the conv 1×1 layers. This further *boosts* the performance to 81.4%, already surpassing Swin-T's result. Note that we have even fewer normalization layers per block than Transformers, as empirically we find that adding one additional BN layer at the beginning of the block does not improve the performance.

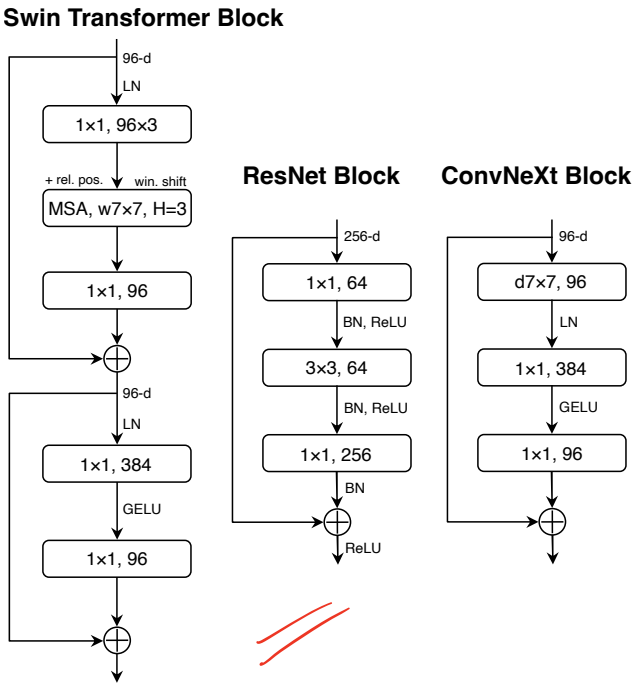

Figure 4. **Block designs** for a ResNet, a Swin Transformer, and a ConvNeXt. Swin Transformer's block is more sophisticated due to the presence of multiple specialized modules and two residual connections. For simplicity, we note the linear layers in Transformer MLP blocks also as "1×1 convs" since they are equivalent.

Substituting BN with LN. BatchNorm [35] is an essential component in ConvNets as it improves the convergence and reduces overfitting. However, BN also has many intricacies that can have a detrimental effect on the model's performance [79]. There have been numerous attempts at developing alternative normalization [57, 70, 78] techniques, but BN has remained the preferred option in most vision tasks. On the other hand, the simpler Layer Normalization [5] (LN) has been used in Transformers, resulting in good performance across different application scenarios.

Directly substituting LN for BN in the original ResNet will result in suboptimal performance [78]. With all the modifications in network architecture and training techniques, here we revisit the impact of using LN in place of BN. We observe that our ConvNet model does not have any difficulties training with LN; in fact, the performance is slightly better, obtaining an accuracy of 81.5%.

From now on, we will use one LayerNorm as our choice of normalization in each residual block.

Separate downsampling layers. In ResNet, the spatial downsampling is achieved by the residual block at the start of each stage, using 3×3 conv with stride 2 (and 1×1 conv with stride 2 at the shortcut connection). In Swin Transformers, a separate downsampling layer is added between stages. We explore a similar strategy in which we use 2×2 conv layers with stride 2 for spatial downsampling. This modification

How is downsampling done in the old ResNet?
Why should we consider separate layers for downsampling in the new design? What can go wrong with this modification?

surprisingly leads to diverged training. Further investigation shows that, adding normalization layers wherever spatial resolution is changed can help stablize training. These include several LN layers also used in Swin Transformers: one before each downsampling layer, one after the stem, and one after the final global average pooling. We can improve the accuracy to 82.0%, significantly exceeding Swin-T's 81.3%.

We will use separate downsampling layers. This brings us to our final model, which we have dubbed ConvNeXt.

A comparison of ResNet, Swin, and ConvNeXt block structures can be found in Figure 4. A comparison of ResNet-50, Swin-T and ConvNeXt-T's detailed architecture specifications can be found in Table 9.

Closing remarks. We have finished our first "playthrough" and discovered ConvNeXt, a pure ConvNet, that can outperform the Swin Transformer for ImageNet-1K classification in this compute regime. It is also worth noting that *none of the design options discussed thus far is novel* — they have all been researched separately, but not collectively, over the last decade. Our ConvNeXt model has approximately the same FLOPs, #params., throughput, and memory use as the Swin Transformer, but does not require specialized modules such as shifted window attention or relative position biases.

These findings are encouraging but not yet completely convincing — our exploration thus far has been limited to a small scale, but vision Transformers' scaling behavior is what truly distinguishes them. Additionally, the question of whether a ConvNet can compete with Swin Transformers on downstream tasks such as object detection and semantic segmentation is a central concern for computer vision practitioners. In the next section, we will scale up our ConvNeXt models both in terms of data and model size, and evaluate them on a diverse set of visual recognition tasks.

3. Empirical Evaluations on ImageNet

We construct different ConvNeXt variants, ConvNeXt-T/S/B/L, to be of similar complexities to Swin-T/S/B/L [42]. ConvNeXt-T/B is the end product of the "modernizing" procedure on ResNet-50/200 regime, respectively. In addition, we build a larger ConvNeXt-XL to further test the scalability of ConvNeXt. The variants only differ in the number of channels C, and the number of blocks B in each stage. Following both ResNets and Swin Transformers, the number of channels doubles at each new stage. We summarize the configurations below:

- 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: $C = (256, 512, 1024, 2048)$, $B = (3, 3, 27, 3)$

3.1. Settings

The ImageNet-1K dataset consists of 1000 object classes with 1.2M training images. We report ImageNet-1K top-1 accuracy on the validation set. We also conduct pre-training on ImageNet-22K, a larger dataset of 21841 classes (a superset of the 1000 ImageNet-1K classes) with ∼14M images for pre-training, and then fine-tune the pre-trained model on ImageNet-1K for evaluation. We summarize our training setups below. More details can be found in Appendix A.

Training on ImageNet-1K. We train ConvNeXts for 300 epochs using AdamW [43] with a learning rate of 4e-3. There is a 20-epoch linear warmup and a cosine decaying schedule afterward. We use a batch size of 4096 and a weight decay of 0.05. For data augmentations, we adopt common schemes including Mixup [85], Cutmix [84], RandAugment [12], and Random Erasing [86]. We regularize the networks with Stochastic Depth [34] and Label Smoothing [65]. Layer Scale [69] of initial value 1e-6 is applied. We use Exponential Moving Average (EMA) [48] as we find it alleviates larger models' overfitting.

Pre-training on ImageNet-22K. We pre-train ConvNeXts on ImageNet-22K for 90 epochs with a warmup of 5 epochs. We do not use EMA. Other settings follow ImageNet-1K.

Fine-tuning on ImageNet-1K. We fine-tune ImageNet-22K pre-trained models on ImageNet-1K for 30 epochs. We use AdamW, a learning rate of 5e-5, cosine learning rate schedule, layer-wise learning rate decay [6, 10], no warmup, a batch size of 512, and weight decay of 1e-8. The default pre-training, fine-tuning, and testing resolution is 224^2. Additionally, we fine-tune at a larger resolution of 384^2, for both ImageNet-22K and ImageNet-1K pre-trained models.

Compared with ViTs/Swin Transformers, ConvNeXts are simpler to fine-tune at different resolutions, as the network is fully-convolutional and there is no need to adjust the input patch size or interpolate absolute/relative position biases.

3.2. Results

ImageNet-1K. Table 1 (upper) shows the result comparison with two recent Transformer variants, DeiT [68] and Swin Transformers [42], as well as two ConvNets from architecture search - RegNets [51] and EfficientNets [67].

ConvNeXt competes favorably with two strong ConvNet baselines (RegNet [51] and EfficientNet [67]) in terms of the accuracy-computation trade-off, as well as the inference throughputs. ConvNeXt also outperforms Swin Transformer of similar complexities *across the board*, sometimes with a substantial margin (*e.g.* 0.8% for ConvNeXt-T). Without specialized modules such as shifted windows or relative position bias, ConvNeXts also enjoy improved throughput compared to Swin Transformers.

Nice 🤩🤩

A highlight from the results is ConvNeXt-B at 384^2: it outperforms Swin-B by 0.6% (85.1% vs. 84.5%), but with

model	image size	#param.	FLOPs	throughput (image / s)	IN-1K top-1 acc.
ImageNet-1K trained models					
• RegNetY-4G [51]	224^2	21M	4.0G	1156.7	80.0
• RegNetY-8G [51]	224^2	39M	8.0G	591.6	81.7
• RegNetY-16G [51]	224^2	84M	16.0G	334.7	82.9
• EffNet-B3 [67]	300^2	12M	1.8G	732.1	81.6
• EffNet-B4 [67]	380^2	19M	4.2G	349.4	82.9
• EffNet-B5 [67]	456^2	30M	9.9G	169.1	83.6
• EffNet-B6 [67]	528^2	43M	19.0G	96.9	84.0
• EffNet-B7 [67]	600^2	66M	37.0G	55.1	84.3
○ DeiT-S [68]	224^2	22M	4.6G	978.5	79.8
○ DeiT-B [68]	224^2	87M	17.6G	302.1	81.8
○ Swin-T	224^2	28M	4.5G	757.9	81.3
• ConvNeXt-T	224^2	29M	4.5G	774.7	**82.1**
○ Swin-S	224^2	50M	8.7G	436.7	83.0
• ConvNeXt-S	224^2	50M	8.7G	447.1	**83.1**
○ Swin-B	224^2	88M	15.4G	286.6	83.5
• ConvNeXt-B	224^2	89M	15.4G	292.1	**83.8**
○ Swin-B	384^2	88M	47.1G	85.1	84.5
• ConvNeXt-B	384^2	89M	45.0G	95.7	**85.1**
• ConvNeXt-L	224^2	198M	34.4G	146.8	**84.3**
• ConvNeXt-L	384^2	198M	101.0G	50.4	**85.5**
ImageNet-22K pre-trained models					
• R-101x3 [36]	384^2	388M	204.6G	-	84.4
• R-152x4 [36]	480^2	937M	840.5G	-	85.4
○ ViT-B/16 [18]	384^2	87M	55.5G	93.1	84.0
○ ViT-L/16 [18]	384^2	305M	191.1G	28.5	85.2
○ Swin-B	224^2	88M	15.4G	286.6	85.2
• ConvNeXt-B	224^2	89M	15.4G	292.1	**85.8**
○ Swin-B	384^2	88M	47.0G	85.1	86.4
• ConvNeXt-B	384^2	89M	45.1G	95.7	**86.8**
○ Swin-L	224^2	197M	34.5G	145.0	86.3
• ConvNeXt-L	224^2	198M	34.4G	146.8	**86.6**
○ Swin-L	384^2	197M	103.9G	46.0	87.3
• ConvNeXt-L	384^2	198M	101.0G	50.4	**87.5**
• ConvNeXt-XL	224^2	350M	60.9G	89.3	**87.0**
• ConvNeXt-XL	384^2	350M	179.0G	30.2	**87.8**

Table 1. **Classification accuracy on ImageNet-1K.** Similar to Transformers, ConvNeXt also shows promising scaling behavior with higher-capacity models and a larger (pre-training) dataset. Inference throughput is measured on a V100 GPU, following [42]. On an A100 GPU, ConvNeXt can have a much higher throughput than Swin Transformer. See Appendix E.

12.5% higher inference throughput (95.7 vs. 85.1 image/s). We note that the FLOPs/throughput advantage of ConvNeXt-B over Swin-B becomes larger when the resolution increases from 224^2 to 384^2. Additionally, we observe an improved result of 85.5% when further scaling to ConvNeXt-L.

ImageNet-22K. We present results with models fine-tuned from ImageNet-22K pre-training at Table 1 (lower). These experiments are important since a widely held view is that vision Transformers have fewer inductive biases thus can perform better than ConvNets when pre-trained on a larger scale. Our results demonstrate that properly designed ConvNets

model	#param.	FLOPs	throughput (image / s)	training mem. (GB)	IN-1K acc.
○ ViT-S	22M	4.6G	978.5	4.9	79.8
• ConvNeXt-S (*iso.*)	22M	4.3G	1038.7	4.2	79.7
○ ViT-B	87M	17.6G	302.1	9.1	81.8
• ConvNeXt-B (*iso.*)	87M	16.9G	320.1	7.7	82.0
○ ViT-L	304M	61.6G	93.1	22.5	82.6
• ConvNeXt-L (*iso.*)	306M	59.7G	94.4	20.4	82.6

Table 2. **Comparing isotropic ConvNeXt and ViT.** Training memory is measured on V100 GPUs with 32 per-GPU batch size.

are *not* inferior to vision Transformers when pre-trained with large dataset — ConvNeXts still perform on par or better than similarly-sized Swin Transformers, with slightly higher throughput. Additionally, our ConvNeXt-XL model achieves an accuracy of 87.8% — a decent improvement over ConvNeXt-L at 384^2, demonstrating that ConvNeXts are scalable architectures.

In Appendix B, we discuss robustness and out-of-domain generalization results for ConvNeXt.

3.3. Isotropic ConvNeXt *vs.* ViT

In this ablation, we examine if our ConvNeXt block design is generalizable to ViT-style [18] isotropic architectures which have no downsampling layers and keep the same feature resolutions (*e.g.* 14×14) at all depths. We construct isotropic ConvNeXt-S/B/L using the same feature dimensions as ViT-S/B/L (384/768/1024). Depths are set at 18/18/36 to match the number of parameters and FLOPs. The block structure remains the same (Fig. 4). We use the supervised training results from DeiT [68] for ViT-S/B and MAE [24] for ViT-L, as they employ improved training procedures over the original ViTs [18]. ConvNeXt models are trained with the same settings as before, but with longer warmup epochs. Results for ImageNet-1K at 224^2 resolution are in Table 2. We observe ConvNeXt can perform generally on par with ViT, showing that our ConvNeXt block design is competitive when used in non-hierarchical models.

4. Empirical Evaluation on Downstream Tasks

Object detection and segmentation on COCO. We fine-tune Mask R-CNN [25] and Cascade Mask R-CNN [7] on the COCO dataset with ConvNeXt backbones. Following Swin Transformer [42], we use multi-scale training, AdamW optimizer, and 3x schedule. Further details and hyper-parameter settings can be found in Appendix A.3.

Table 3 shows object detection and instance segmentation results comparing Swin Transformer, ConvNeXt, and traditional ConvNet such as ResNeXt. Across different model complexities, ConvNeXt achieves on-par or better performance than Swin Transformer. When scaled up to bigger models (ConvNeXt-B/L/XL) pre-trained on ImageNet-22K, in many cases *ConvNeXt is significantly better* (*e.g.* +1.0 AP)

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).

than Swin Transformers in terms of box and mask AP.

Semantic segmentation on ADE20K. We also evaluate ConvNeXt backbones on the ADE20K semantic segmentation task with UperNet [80]. All model variants are trained for 160K iterations with a batch size of 16. Other experimental settings follow [6] (see Appendix A.3 for more details). In Table 4, we report validation mIoU with multi-scale testing. ConvNeXt models can achieve competitive performance across different model capacities, further validating the effectiveness of our architecture design.

Remarks on model efficiency. Under similar FLOPs, models with depthwise convolutions are known to be slower and consume more memory than ConvNets with only dense convolutions. It is natural to ask whether the design of ConvNeXt will render it practically inefficient. As demonstrated throughout the paper, the inference throughputs of ConvNeXts are comparable to or exceed that of Swin Transformers. This is true for both classification and other tasks requiring higher-resolution inputs (see Table 1,3 for comparisons of throughput/FPS). Furthermore, we notice that training ConvNeXts requires less memory than training Swin Transformers. For example, training Cascade Mask-RCNN using ConvNeXt-B backbone consumes 17.4GB of peak memory with a per-GPU batch size of 2, while the reference number for Swin-B is 18.5GB. In comparison to vanilla ViT, both ConvNeXt and Swin Transformer exhibit a more favorable accuracy-FLOPs trade-off due to the local computations. It is worth noting that this improved efficiency is a result of

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.

the *ConvNet inductive bias*, and is not directly related to the self-attention mechanism in vision Transformers.

Finally, credit to the inductive bias of the ConvNets

5. Related Work

Hybrid models. In both the pre- and post-ViT eras, the hybrid model combining convolutions and self-attentions has been actively studied. Prior to ViT, the focus was on augmenting a ConvNet with self-attention/non-local modules [52, 63, 74] to capture long-range dependencies. The original ViT [18] first studied a hybrid configuration, and a large body of follow-up works focused on reintroducing convolutional priors to ViT, either in an explicit [13, 14, 19, 77, 81, 83] or implicit [42] fashion.

Recent convolution-based approaches. Han *et al.* [23] show that local Transformer attention is equivalent to inhomogeneous dynamic depthwise conv. The MSA block in Swin is then replaced with a dynamic or regular depthwise convolution, achieving comparable performance to Swin. A concurrent work ConvMixer [4] demonstrates that, in small-scale settings, depthwise convolution can be used as a promising mixing strategy. ConvMixer uses a smaller patch size to achieve the best results, making the throughput much lower than other baselines. GFNet [53] adopts Fast Fourier Transform (FFT) for token mixing. FFT is also a form of convolution, but with a global kernel size and circular padding. Unlike many recent Transformer or ConvNet designs, one primary goal of our study is to provide an in-depth look at the process of modernizing a standard ResNet and achieving state-of-the-art performance.

6. Conclusions

In the 2020s, vision Transformers, particularly hierarchical ones such as Swin Transformers, began to overtake ConvNets as the favored choice for generic vision backbones. The widely held belief is that vision Transformers are more accurate, efficient, and scalable than ConvNets. We propose ConvNeXts, a pure ConvNet model that can compete favorably with state-of-the-art hierarchical vision Transformers across multiple computer vision benchmarks, while retaining the simplicity and efficiency of standard ConvNets. In some ways, our observations are surprising while our ConvNeXt model itself is not completely new — many design choices have all been examined separately over the last decade, but not collectively. We hope that the new results reported in this study will challenge several widely held views and prompt people to rethink the importance of convolution in computer vision.

Acknowledgments. We thank Kaiming He, Eric Mintun, Xingyi Zhou, Ross Girshick, and Yann LeCun for valuable discussions and feedback.

Appendix

In this Appendix, we provide further experimental details (§A), robustness evaluation results (§B), more modernization experiment results (§C), and a detailed network specification (§D). We further benchmark model throughput on A100 GPUs (§E). Finally, we discuss the limitations (§F) and societal impact (§G) of our work.

A. Experimental Settings

A.1. ImageNet (Pre-)training

We provide ConvNeXts' ImageNet-1K training and ImageNet-22K pre-training settings in Table 5. The settings are used for our main results in Table 1 (Section 3.2). All ConvNeXt variants use the same setting, except the stochastic depth rate is customized for model variants.

For experiments in "modernizing a ConvNet" (Section 2), we also use Table 5's setting for ImageNet-1K, except EMA is disabled, as we find using EMA severely hurts models with BatchNorm layers.

For isotropic ConvNeXts (Section 3.3), the setting for ImageNet-1K in Table A is also adopted, but warmup is extended to 50 epochs, and layer scale is disabled for isotropic ConvNeXt-S/B. The stochastic depth rates are 0.1/0.2/0.5 for isotropic ConvNeXt-S/B/L.

A.2. ImageNet Fine-tuning

We list the settings for fine-tuning on ImageNet-1K in Table 6. The fine-tuning starts from the final model weights obtained in pre-training, without using the EMA weights, even if in pre-training EMA is used and EMA accuracy is reported. This is because we do not observe improvement if we fine-tune with the EMA weights (consistent with observations in [68]). The only exception is ConvNeXt-L pre-trained on ImageNet-1K, where the model accuracy is significantly lower than the EMA accuracy due to overfitting, and we select its best EMA model during pre-training as the starting point for fine-tuning.

Why training is no longer "simple", especially if you want a close to optimal baseline?

(pre-)training config	ConvNeXt-T/S/B/L ImageNet-1K 224^2	ConvNeXt-B/L/XL ImageNet-22K 224^2
optimizer	AdamW	AdamW
base learning rate	4e-3	4e-3
weight decay	0.05	0.05
optimizer momentum	β_1, β_2=0.9, 0.999	β_1, β_2=0.9, 0.999
batch size	4096	4096
training epochs	300	90
learning rate schedule	cosine decay	cosine decay
warmup epochs	20	5
warmup schedule	linear	linear
layer-wise lr decay [6, 10]	None	None
randaugment [12]	(9, 0.5)	(9, 0.5)
label smoothing [65]	0.1	0.1
mixup [85]	0.8	0.8
cutmix [84]	1.0	1.0
stochastic depth [34]	0.1/0.4/0.5/0.5	0.1/0.1/0.2
layer scale [69]	1e-6	1e-6
gradient clip	None	None
exp. mov. avg. (EMA) [48]	0.9999	None

Table 5. **ImageNet-1K/22K (pre-)training settings**. Multiple stochastic depth rates (e.g., 0.1/0.4/0.5/0.5) are for each model (e.g., ConvNeXt-T/S/B/L) respectively.

pre-training config	ConvNeXt-B/L ImageNet-1K 224^2	ConvNeXt-B/L/XL ImageNet-22K 224^2
fine-tuning config	ImageNet-1K 384^2	ImageNet-1K 224^2 and 384^2
optimizer	AdamW	AdamW
base learning rate	5e-5	5e-5
weight decay	1e-8	1e-8
optimizer momentum	β_1, β_2=0.9, 0.999	β_1, β_2=0.9, 0.999
batch size	512	512
training epochs	30	30
learning rate schedule	cosine decay	cosine decay
layer-wise lr decay	0.7	0.8
warmup epochs	None	None
warmup schedule	N/A	N/A
randaugment	(9, 0.5)	(9, 0.5)
label smoothing	0.1	0.1
mixup	None	None
cutmix	None	None
stochastic depth	0.8/0.95	0.2/0.3/0.4
layer scale	pre-trained	pre-trained
gradient clip	None	None
exp. mov. avg. (EMA)	None	None/None/0.9999

Table 6. **ImageNet-1K fine-tuning settings**. Multiple values (e.g., 0.8/0.95) are for each model (e.g., ConvNeXt-B/L) respectively.

In fine-tuning, we use layer-wise learning rate decay [6, 10] with every 3 consecutive blocks forming a group. When the model is fine-tuned at 384^2 resolution, we use a crop ratio of 1.0 (i.e., no cropping) during testing following [2, 69, 75], instead of 0.875 at 224^2.

A.3. Downstream Tasks

For ADE20K and COCO experiments, we follow the training settings used in BEiT [6] and Swin [42]. We also use MMDetection [8] and MMSegmentation [11] toolboxes. We use the final model weights (instead of EMA weights) from ImageNet pre-training as network initializations.

We conduct a lightweight sweep for COCO experiments including learning rate {1e-4, 2e-4}, layer-wise learning rate decay [6] {0.7, 0.8, 0.9, 0.95}, and stochastic depth rate {0.3, 0.4, 0.5, 0.6, 0.7, 0.8}. We fine-tune the ImageNet-22K pre-trained Swin-B/L on COCO using the same sweep. We use the official code and pre-trained model weights [3].

The hyperparameters we sweep for ADE20K experiments include learning rate {8e-5, 1e-4}, layer-wise learning rate decay {0.8, 0.9}, and stochastic depth rate {0.3, 0.4, 0.5}. We report validation mIoU results using multi-scale testing. Additional single-scale testing results can be found in Table 7.

backbone	input crop.	mIoU
ImageNet-1K pre-trained		
• ConvNeXt-T	512^2	46.0
• ConvNeXt-S	512^2	48.7
• ConvNeXt-B	512^2	49.1
ImageNet-22K pre-trained		
• ConvNeXt-B‡	640^2	52.6
• ConvNeXt-L‡	640^2	53.2
• ConvNeXt-XL‡	640^2	53.6

Table 7. **ADE20K validation results** with single-scale testing.

B. Robustness Evaluation

Additional robustness evaluation results for ConvNeXt models are presented in Table 8. We directly test our ImageNet-1K trained/fine-tuned classification models on several robustness benchmark datasets such as ImageNet-A [31], ImageNet-R [28], ImageNet-Sketch [73] and ImageNet-C/$\bar{\text{C}}$ [29, 45] datasets. We report mean corruption error (mCE) for ImageNet-C, corruption error for ImageNet-$\bar{\text{C}}$, and top-1 Accuracy for all other datasets.

ConvNeXt (in particular the large-scale model variants) exhibits promising robustness behaviors, outperforming state-of-the-art robust transformer models [44] on several benchmarks. With extra ImageNet-22K data, ConvNeXt-XL demonstrates strong domain generalization capabilities (*e.g.* achieving 69.3%/68.2%/55.0% accuracy on ImageNet-A/R/Sketch benchmarks, respectively). We note that these robustness evaluation results were acquired without using any specialized modules or additional fine-tuning procedures.

Model	Data/Size	FLOPs / Params	Clean	C (↓)	$\bar{\text{C}}$ (↓)	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.

C. Modernizing ResNets: detailed results

Here we provide detailed tabulated results for the *modernization* experiments, at both ResNet-50 / Swin-T and ResNet-200 / Swin-B regimes. The ImageNet-1K top-1 accuracies and FLOPs for each step are shown in Table 10 and 11. ResNet-50 regime experiments are run with 3 random seeds.

For ResNet-200, the initial number of blocks at each stage is (3, 24, 36, 3). We change it to Swin-B's (3, 3, 27, 3) at the step of changing stage ratio. This drastically reduces the FLOPs, so at the same time, we also increase the width from 64 to 84 to keep the FLOPs at a similar level. After the step of adopting depthwise convolutions, we further increase the width to 128 (same as Swin-B's) as a separate step.

The observations on the ResNet-200 regime are mostly consistent with those on ResNet-50 as described in the main paper. One interesting difference is that inverting dimensions brings a larger improvement at ResNet-200 regime than at ResNet-50 regime (+0.79% *vs.* +0.14%). The performance gained by increasing kernel size also seems to saturate at kernel size 5 instead of 7. Using fewer normalization layers also has a bigger gain compared with the ResNet-50 regime (+0.46% *vs.* +0.14%).

D. Detailed Architectures

We present a detailed architecture comparison between ResNet-50, ConvNeXt-T and Swin-T in Table 9. For differently sized ConvNeXts, only the number of blocks and the number of channels at each stage differ from ConvNeXt-T (see Section 3 for details). ConvNeXts enjoy the simplicity of standard ConvNets, but compete favorably with Swin Transformers in visual recognition.

	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.

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

Table 10. **Detailed results for modernizing a ResNet-50.** Mean and standard deviation are obtained by training the network with three different random seeds.

model	IN-1K acc.	GFLOPs
ResNet-200 [27]	78.20	15.01
ResNet-200 (enhanced recipe)	81.14	15.01
stage ratio and increase width	81.33	14.52
"patchify" stem	81.59	14.38
depthwise conv	80.54	7.23
increase width	81.85	16.76
inverting dimensions	82.64	15.68
move up depthwise conv	82.04	14.63
kernel size → 5	82.32	14.70
kernel size → 7	82.30	14.81
kernel size → 9	82.27	14.95
kernel size → 11	82.18	15.13
ReLU → GELU	82.19	14.81
fewer activations	82.71	14.81
fewer norms	83.17	14.81
BN → LN	83.35	14.81
separate d.s. conv (ConvNeXt-B)	83.60	15.35
Swin-B [42]	83.50	15.43

Table 11. **Detailed results for modernizing a ResNet-200.**

E. Benchmarking on A100 GPUs

Following Swin Transformer [42], the ImageNet models' inference throughputs in Table 1 are benchmarked using a V100 GPU, where ConvNeXt is slightly faster in inference than Swin Transformer with a similar number of parameters. We now benchmark them on the more advanced A100 GPUs, which support the TensorFloat32 (TF32) tensor cores. We employ PyTorch [47] version 1.10 to use the latest "Channel Last" memory layout [20] for further speedup.

We present the results in Table 12. Swin Transformers and ConvNeXts both achieve faster inference throughput than V100 GPUs, but ConvNeXts' advantage is now significantly greater, sometimes *up to 49% faster*. This preliminary study shows promising signals that ConvNeXt, employed with

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.

standard ConvNet modules and simple in design, could be practically more efficient models on modern hardwares.

F. Limitations

We demonstrate ConvNeXt, a pure ConvNet model, can perform as good as a hierarchical vision Transformer on image classification, object detection, instance and semantic segmentation tasks. While our goal is to offer a broad range of evaluation tasks, we recognize computer vision applications are even more diverse. ConvNeXt may be more suited for certain tasks, while Transformers may be more flexible for others. A case in point is multi-modal learning, in which a cross-attention module may be preferable for modeling feature interactions across many modalities. Additionally, Transformers may be more flexible when used for tasks requiring discretized, sparse, or structured outputs. We believe the architecture choice should meet the needs of the task at hand while striving for simplicity.

G. Societal Impact

In the 2020s, research on visual representation learning began to place enormous demands on computing resources. While larger models and datasets improve performance across the board, they also introduce a slew of challenges. ViT, Swin, and ConvNeXt all perform best with their huge model variants. Investigating those model designs inevitably results in an increase in carbon emissions. One important direction, and a motivation for our paper, is to strive for simplicity — with more sophisticated modules, the network's design space expands enormously, obscuring critical components that contribute to the performance difference. Additionally, large models and datasets present issues in terms of model robustness and fairness. Further investigation on the robustness behavior of ConvNeXt vs. Transformer will be an interesting research direction. In terms of data, our findings indicate that ConvNeXt models benefit from pre-training on large-scale datasets. While our method makes use of the publicly available ImageNet-22K dataset, individuals may wish to acquire their own data for pre-training. A more circumspect and responsible approach to data selection is required to avoid potential concerns with data biases.

References

[1] PyTorch Vision Models. https://pytorch.org/vision/stable/models.html. Accessed: 2021-10-01.

[2] GitHub repository: Swin transformer. https://github.com/microsoft/Swin-Transformer, 2021.

[3] GitHub repository: Swin transformer for object detection. https://github.com/SwinTransformer/Swin-Transformer-Object-Detection, 2021.

[4] Anonymous. Patches are all you need? *Openreview*, 2021.

[5] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv:1607.06450*, 2016.

[6] Hangbo Bao, Li Dong, and Furu Wei. BEiT: BERT pre-training of image transformers. *arXiv:2106.08254*, 2021.

[7] Zhaowei Cai and Nuno Vasconcelos. Cascade R-CNN: Delving into high quality object detection. In *CVPR*, 2018.

[8] Kai Chen, Jiaqi Wang, Jiangmiao Pang, Yuhang Cao, Yu Xiong, Xiaoxiao Li, Shuyang Sun, Wansen Feng, Ziwei Liu, Jiarui Xu, Zheng Zhang, Dazhi Cheng, Chenchen Zhu, Tianheng Cheng, Qijie Zhao, Buyu Li, Xin Lu, Rui Zhu, Yue Wu, Jifeng Dai, Jingdong Wang, Jianping Shi, Wanli Ouyang, Chen Change Loy, and Dahua Lin. MMDetection: Open mmlab detection toolbox and benchmark. *arXiv:1906.07155*, 2019.

[9] François Chollet. Xception: Deep learning with depthwise separable convolutions. In *CVPR*, 2017.

[10] Kevin Clark, Minh-Thang Luong, Quoc V Le, and Christopher D Manning. ELECTRA: Pre-training text encoders as discriminators rather than generators. In *ICLR*, 2020.

[11] MMSegmentation contributors. MMSegmentation: Openmmlab semantic segmentation toolbox and benchmark. https://github.com/open-mmlab/mmsegmentation, 2020.

[12] Ekin D Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V Le. Randaugment: Practical automated data augmentation with a reduced search space. In *CVPR Workshops*, 2020.

[13] Zihang Dai, Hanxiao Liu, Quoc V Le, and Mingxing Tan. Coatnet: Marrying convolution and attention for all data sizes. *NeurIPS*, 2021.

[14] Stéphane d'Ascoli, Hugo Touvron, Matthew Leavitt, Ari Morcos, Giulio Biroli, and Levent Sagun. ConViT: Improving vision transformers with soft convolutional inductive biases. *ICML*, 2021.

[15] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *CVPR*, 2009.

[16] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *NAACL*, 2019.

[17] Piotr Dollár, Serge Belongie, and Pietro Perona. The fastest pedestrian detector in the west. In *BMVC*, 2010.

[18] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *ICLR*, 2021.

[19] Haoqi Fan, Bo Xiong, Karttikeya Mangalam, Yanghao Li, Zhicheng Yan, Jitendra Malik, and Christoph Feichtenhofer. Multiscale vision transformers. *ICCV*, 2021.

[20] Vitaly Fedyunin. Tutorial: Channel last memory format in PyTorch. https://pytorch.org/tutorials/intermediate/memory_format_tutorial.html, 2021. Accessed: 2021-10-01.

[21] Ross Girshick. Fast R-CNN. In *ICCV*, 2015.

[22] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*, 2014.

[23] Qi Han, Zejia Fan, Qi Dai, Lei Sun, Ming-Ming Cheng, Jiaying Liu, and Jingdong Wang. Demystifying local vision transformer: Sparse connectivity, weight sharing, and dynamic weight. *arXiv:2106.04263*, 2021.

[24] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked autoencoders are scalable vision learners. *arXiv:2111.06377*, 2021.

[25] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask R-CNN. In *ICCV*, 2017.

[26] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[27] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *ECCV*, 2016.

[28] Dan Hendrycks, Steven Basart, Norman Mu, Saurav Kadavath, Frank Wang, Evan Dorundo, Rahul Desai, Tyler Zhu, Samyak Parajuli, Mike Guo, et al. The many faces of robustness: A critical analysis of out-of-distribution generalization. In *ICCV*, 2021.

[29] Dan Hendrycks and Thomas Dietterich. Benchmarking neural network robustness to common corruptions and perturbations. In *ICLR*, 2018.

[30] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv:1606.08415*, 2016.

[31] Dan Hendrycks, Kevin Zhao, Steven Basart, Jacob Steinhardt, and Dawn Song. Natural adversarial examples. In *CVPR*, 2021.

[32] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. MobileNets: Efficient convolutional neural networks for mobile vision applications. *arXiv:1704.04861*, 2017.

[33] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *CVPR*, 2017.

[34] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q Weinberger. Deep networks with stochastic depth. In *ECCV*, 2016.

[35] Sergey Ioffe. Batch renormalization: Towards reducing minibatch dependence in batch-normalized models. In *NeurIPS*, 2017.

[36] Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Joan Puigcerver, Jessica Yung, Sylvain Gelly, and Neil Houlsby. Big Transfer (BiT): General visual representation learning. In *ECCV*, 2020.

[37] Alex Krizhevsky, Ilya Sutskever, and Geoff Hinton. Imagenet classification with deep convolutional neural networks. In *NeurIPS*, 2012.

[38] Andrew Lavin and Scott Gray. Fast algorithms for convolutional neural networks. In *CVPR*, 2016.

[39] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1989.

[40] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 1998.

[41] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft COCO: Common objects in context. In *ECCV*. 2014.

[42] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. 2021.

[43] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *ICLR*, 2019.

[44] Xiaofeng Mao, Gege Qi, Yuefeng Chen, Xiaodan Li, Ranjie Duan, Shaokai Ye, Yuan He, and Hui Xue. Towards robust vision transformer. *arXiv preprint arXiv:2105.07926*, 2021.

[45] Eric Mintun, Alexander Kirillov, and Saining Xie. On interaction between augmentations and corruptions in natural corruption robustness. *NeurIPS*, 2021.

[46] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, 2010.

[47] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. PyTorch: An imperative style, high-performance deep learning library. In *NeurIPS*, 2019.

[48] Boris T Polyak and Anatoli B Juditsky. Acceleration of stochastic approximation by averaging. *SIAM Journal on Control and Optimization*, 1992.

[49] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

[50] Ilija Radosavovic, Justin Johnson, Saining Xie, Wan-Yen Lo, and Piotr Dollár. On network design spaces for visual recognition. In *ICCV*, 2019.

[51] Ilija Radosavovic, Raj Prateek Kosaraju, Ross Girshick, Kaiming He, and Piotr Dollár. Designing network design spaces. In *CVPR*, 2020.

[52] Prajit Ramachandran, Niki Parmar, Ashish Vaswani, Irwan Bello, Anselm Levskaya, and Jonathon Shlens. Stand-alone self-attention in vision models. *NeurIPS*, 2019.

[53] Yongming Rao, Wenliang Zhao, Zheng Zhu, Jiwen Lu, and Jie Zhou. Global filter networks for image classification. *NeurIPS*, 2021.

[54] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *NeurIPS*, 2015.

[55] Henry A Rowley, Shumeet Baluja, and Takeo Kanade. Neural network-based face detection. *TPAMI*, 1998.

[56] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *IJCV*, 2015.

[57] Tim Salimans and Diederik P Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *NeurIPS*, 2016.

[58] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *CVPR*, 2018.

[59] Pierre Sermanet, David Eigen, Xiang Zhang, Michael Mathieu, Rob Fergus, and Yann LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. In *ICLR*, 2014.

[60] Pierre Sermanet, Koray Kavukcuoglu, Soumith Chintala, and Yann LeCun. Pedestrian detection with unsupervised multi-stage feature learning. In *CVPR*, 2013.

[61] Karen Simonyan and Andrew Zisserman. Two-stream convolutional networks for action recognition in videos. In *NeurIPS*, 2014.

[62] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

[63] Aravind Srinivas, Tsung-Yi Lin, Niki Parmar, Jonathon Shlens, Pieter Abbeel, and Ashish Vaswani. Bottleneck transformers for visual recognition. In *CVPR*, 2021.

[64] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *CVPR*, 2015.

[65] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *CVPR*, 2016.

[66] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. In *CVPR*, 2019.

[67] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *ICML*, 2019.

[68] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. *arXiv:2012.12877*, 2020.

[69] Hugo Touvron, Matthieu Cord, Alexandre Sablayrolles, Gabriel Synnaeve, and Herv'e J'egou. Going deeper with image transformers. *ICCV*, 2021.

[70] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization. *arXiv:1607.08022*, 2016.

[71] Régis Vaillant, Christophe Monrocq, and Yann Le Cun. Original approach for the localisation of objects in images. *Vision, Image and Signal Processing*, 1994.

[72] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, 2017.

[73] Haohan Wang, Songwei Ge, Eric P Xing, and Zachary C Lipton. Learning robust global representations by penalizing local predictive power. *NeurIPS*, 2019.

[74] Xiaolong Wang, Ross Girshick, Abhinav Gupta, and Kaiming He. Non-local neural networks. In *CVPR*, 2018.

[75] Ross Wightman. GitHub repository: Pytorch image models. https://github.com/rwightman/pytorch-image-models, 2019.

[76] Ross Wightman, Hugo Touvron, and Hervé Jégou. Resnet strikes back: An improved training procedure in timm. *arXiv:2110.00476*, 2021.

[77] Haiping Wu, Bin Xiao, Noel Codella, Mengchen Liu, Xiyang Dai, Lu Yuan, and Lei Zhang. Cvt: Introducing convolutions to vision transformers. *ICCV*, 2021.

[78] Yuxin Wu and Kaiming He. Group normalization. In *ECCV*, 2018.

[79] Yuxin Wu and Justin Johnson. Rethinking "batch" in batchnorm. *arXiv:2105.07576*, 2021.

[80] Tete Xiao, Yingcheng Liu, Bolei Zhou, Yuning Jiang, and Jian Sun. Unified perceptual parsing for scene understanding. In *ECCV*, 2018.

[81] Tete Xiao, Mannat Singh, Eric Mintun, Trevor Darrell, Piotr Dollár, and Ross Girshick. Early convolutions help transformers see better. In *NeurIPS*, 2021.

[82] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *CVPR*, 2017.

[83] Weijian Xu, Yifan Xu, Tyler Chang, and Zhuowen Tu. Co-scale conv-attentional image transformers. *ICCV*, 2021.

[84] Sangdoo Yun, Dongyoon Han, Seong Joon Oh, Sanghyuk Chun, Junsuk Choe, and Youngjoon Yoo. Cutmix: Regularization strategy to train strong classifiers with localizable features. In *ICCV*, 2019.

[85] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. In *ICLR*, 2018.

[86] Zhun Zhong, Liang Zheng, Guoliang Kang, Shaozi Li, and Yi Yang. Random erasing data augmentation. In *AAAI*, 2020.

[87] Bolei Zhou, Hang Zhao, Xavier Puig, Tete Xiao, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Semantic understanding of scenes through the ADE20K dataset. *IJCV*, 2019.

Summary

What the authors are trying to address through this paper?

- Since the inception of Vision Transformers (ViT), computer vision has a seen a lot of advanced works favouring transformers over convnets largely because of the generic design and ability of transformers to scale as the size of the datasets grows.
- It has also been noted in the past that vanilla ViT doesn't perform well on tasks such as object detection and semantic segmentation. To address this, hybrid approaches combining convnets and transformers have been introduced. Swin Transformer is a typical example of this. However the success of hybrid models is largely credited to the transformers and not so to the inductive biases of convolutions
- The authors aim to revisit the convnets, and try to modernise a standard ResNet towards the design of a vision transformer in to order to test the limits of a pure convnet. The family of network coming out of this experiment is dubbed as ConvNexts

Modernising a ConvNet: The modernisation is broken down into six steps. The base model used in these experiments is ResNet-50 which is then modified at each stage to convert it into a ConvNext model.

A. Training Techniques: A good training is equally important as building a good model. ViTs not only introduced a set of new modules but they also introduced a set of training techniques for an optimal performing model. Hence the authors used the new training strategies to a get a good baseline ResNet-50 model.
   a. Increased training epochs from 90 to 300
   b. AdamW optimiser is used. Warmup, cosine schedule
   c. Data augmentation: Cutmix, randaugment, mixup, random erasing
   d. Regularization: Stochastic depth, label smoothing, weight decay

B. Macro Design
   a. Changing stage compute ratio: Following the design of swin transformers which follow a 1:1:3:1 compute ratio, the number of blocks in each resnet stage is changed from (3, 4, 6, 3) to (3, 3, 9, 3). This design improves the accuracy approximately by 1% while keeping the FLOPs aligned with Swin-T
   b. Changing stem to patchify: Due to natural redundancy, aggressive down sampling strategies are used both in original ResNet and ViT. ResNets used a 7x7 convolution with a stride of 2 followed by max pooling, whereas ViT used patchify scheme which is equivalent to large kernel sizes (14 or 16) and non-overlapping convolution. Swin-T uses a similar patchify layer but with a smaller patch size of 4. The authors thus modified the stem layer in standard ResNet with a patchify layer implemented using a 4x4, stride 4 convolutions layer.

C. ResNext-ify: ResNext used grouped convolutions to achieve a better FLOPs/accuracy tradeoff. The authors, instead of using grouped convolutions used depth wise convolutions. Depthwise convolution is similar to weighted sum operation in self-attention. Also any depthwise convolution layer reduces the FLOPs by a big factor.

D. Inverted bottleneck: The idea of inverted residual blocks was popularised by MobileNetV2. The authors used inverted bottlenecks

E. kernel sizes: One of the biggest advantages that transformers have over standard convnets is their non-local self-attention, enabling ea h layer to have a global receptive field.
   a. Large kernel sizes: The authors experimented with kernel sizes of 3, 5, 7, 9, and 11. Kernel size of 7 works best for most use cases
   b. Moving depthwise convolution layer up in the block: One of the things that we need to do if we want to use large kernel sizes, is that we would want to place it at the top of the inverted bottleneck. Doing this makes the network more efficient as depthwise isn'that efficient in nature, hence we would want to process a relatively lower number of channels. This will also help in reducing the overall FLOps as well (check Fig 3)

F. Micro Design: Layer level modifications are done in this stage
   a. Replace ReLU with GELU: No performance improvement
   b. Fewer activations in blocks: Transformers have relatively fewer activations compared to a standard ResNet. In ResNet, it is common practice to append an activation to each convolutions layer whereas in transformers there is only one activation used in the MLP block. The authors thus removed all the GELU activations from a residual block except for the one between two 1x1 convolutions. This boosted the performance by 0.7% practically matching the performance at this stage (given all the above modifications have been done)
   c. Fewer Normalization layers: Following the same logic as above, the authors removed all the normalizatioj layers except for one normalizatioj layer before the 1x1 convolution layers
   d. Layer Normalization instead of Batch Normalization: BN is a well known pain to work with, and transformers any way uses LN (though the logical reasoning there is diff). The authors noted that replacing BN with LN improved the performance slightly
   e. Separate downsampling layers: ResNet uses a 3x3 conv with stride 2 at the start of each block to down sample the images whereas swin transformers use separate Dow sampling layers at each stage. The authors used a similar strategy in which they used 2x2 conv layers with stride 2 for spatial doensampling. One important thing to note here is that this setup leads to diverged training but can be stabilised by adding Normalization layers wherever spatial resolution is changed

With all the above modifications, the authors got a family of modernised ResNets dubbed as ConvNexts