Efficient and Scalable Neural Architectures for Visual Recognition

by

Zhuang Liu

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Trevor Darrell, Chair
Professor Joseph Gonzalez
Professor Jiantao Jiao
Dr. Saining Xie

Summer 2022

Efficient and Scalable Neural Architectures for Visual Recognition

Abstract

Efficient and Scalable Neural Architectures for Visual Recognition

by

Zhuang Liu

Doctor of Philosophy in Computer Science

University of California, Berkeley

Professor Trevor Darrell, Chair

The successful application of ConvNets and other neural architectures to computer vision is central to the AI revolution seen in the past decade. There have been strong needs for scaling vision architectures to be both smaller and larger. Small models represent the demand for efficiency, as the deployment of visual recognition systems is often on edge devices; large models highlight the pursuit for scalability - the ability to utilize increasingly abundant compute and data to achieve ever-higher accuracy. Research in both directions are fruitful, producing many useful design principles, and the quest for more performant models never stops. Meanwhile, the very fast development pace in the literature can sometimes obscure the main mechanism responsible for certain methods' favorable results.

In this dissertation, we will present our research from two aspects in this area: (1) developing intuitive algorithms for efficient and flexible ConvNet model inference; (2) studying baseline approaches to reveal what is behind popular scaling methods' success. First, we will introduce our work on one of the first anytime algorithm for dense prediction. We will then examine the effectiveness of model pruning algorithms by comparing them with an extremely simple baseline, and argue their true value may lie in learning architectures. Finally, We present our work on questioning whether self-attention is responsible for Transformer's recent exceptional scalability in vision, by modernizing a traditional ConvNet with design techniques adapted from Transformers.

To my parents, Lilin Miao and Qing Liu

Contents

List of Figures

List of Tables

Acknowledgments

I want to thank all people who have helped or supported me during my Ph.D. journey. First, I would like to express my sincere gratitude to my advisor Prof. Trevor Darrell. Trevor has continued to guide me on how to do quality and impactful research, while also giving me the freedom to work on research topics that I'm interested in. His advices on both high-level research directions and execution plans have always been helpful. He would not withhold critical opinions either, which pushed me to distill and refine my work until it is above the high standard he sets and can withstand challenges. Of course, the help and lessons from Trevor are more than on research. When I was frustrated, Trevor always expressed his strongest support and reassured me of the value of my research. The career advices from him are also invaluable. I would also like to thank Prof. Joseph Gonzalez, Prof. Jiantao Jiao, and Dr. Saining Xie for being on my dissertation committee, who gave me lots of useful advices in delivering my dissertation talk and preparing this thesis.

I was fortunate enough to have other excellent mentors during my Ph.D. study. I would like to thank Vladlen Koltun for supervising me during my internship at Intel Labs, where I worked with wonderful colleagues John Lambert and Ozan Sener. I want to thank Evan Shelhamer, who mentored me during my internship at Adobe Research, and has been giving continued support throughout the rest of my Ph.D. I am also grateful to Saining Xie for giving me the opportunity for an internship at Facebook AI Research and guiding me through the research project. I want to thank Hanzi Mao for her crucial help in this project too. Additionally, I am thankful to Junyan Zhu and Tinghui Zhou who mentored me as senior students in my earlier Ph.D. years.

I am also grateful to my mentors and collaborators before I came to Berkeley. It was in Prof. Kilian Weinberger's group at Cornell University that I started my journey in the field of deep learning, during my visit there in 2016. I want to thank Gao Huang, Yu Sun and Shuang Li who helped me learn the basics of deep learning research there. I would like to thank Jianguo Li for taking me as a research intern at Intel Labs China when I was a senior undergraduate, and for the great mentorship he gave. I'm also grateful to Prof. Andrew Yao, who built the undergraduate program at Tsinghua that led me into the world of computer science. I want to thank Shiquan Wang, Farong Zhou, Huijuan Wang, Niya Wei and all my other teachers since childhood who committed their lives to the education of the next generations.

My gratitude goes to my other collaborators, colleagues and friends, at Berkeley or otherwise. I want to thank Bingyi Kang, Chubai Chen, Guanhua Wang, Zhiqiang Shen, Xiang Gao, Xuanlin Li, Mingjie Sun, Hung-Ju Wang, Zhiqiu Xu, Joseph Jin, Maolin Mao, Yinbo Chen, Brandon Hsieh, Yi Wu, Yang Gao, Yang You, Bichen Wu, Dequan Wang, Huazhe Xu, Xin Wang, Xiangyu Yue, Shizhan Zhu, Zhe Cao, Haozhi Qi, Hang Gao, Wenlong Mou, Zihao Chen, Xingyi Zhou, Hexiang Hu, Hengshuang Zhao, Zhipeng Cai, Yixing Lao, Ji Lin, Linnan Wang, Christoph Feichtenhofer, Chao-Yuan Wu, Chuan Guo, Huijuan Xu, Xiaolong Wang, Fisher Yu, Deepak Pathak, Kai Jin, Lihan Zhu, Rui Li, Cheng Zhu and Zijing Xia. I am thankful to Jiashi Feng, Qixing Huang, Jingdong Wang, Xiaoming Liu, Zhuowen Tu, Ng Teck Khim, who as senior members of the research community gave me many pieces of advices. I want to thank other members of Trevor's lab for building such a wonderful and supportive group.

I would like to thank my partner Chen Zhang, who has been the source of happiness, comfort and encouragement in my life. I look forward to starting the next chapter of my life with you.

Finally, I would like to thank my parents Lilin Miao and Qing Liu, and my grandparents, Fangcui Wang and Nengyuan Miao. You gave me myself, my life, and a family with love. Without your support all the way since my birth and childhood, I could not imagine myself getting a Ph.D.

Chapter 1

Introduction

The basic diagram of deep learning [103] dates back to decades ago with the proposal of gradient-based back-propagation learning algorithms [151] in the 1980s, and ConvNets have been applied for computer vision tasks such as hand-written digit recognition [105] since these early years. However, the true power of deep learning has only been revealed in 2012, with AlexNet [99] winning the ImageNet large-scale image classification challenge [34] that year. Increased data availability [34, 110], advancement of computing technologies [124, 134], and improved algorithms [64, 181, 95] are three pillars for deep learning's continued success in various application areas [93, 159]. The rapid progress in this field has yet to show signs of slowing with the recent rise of large models [13, 144].

Deep learning has not only made a remarkable impact on our everyday lives, but also changed the workflow of machine learning practitioners and researchers - the community has shifted from using hand-crafted features [123, 32] with shallow models (e.g., SVM [27]), to automatically extracting feature representations with multi-layer deep neural networks. Hand-crafted features are often highly task-specific and not generalizable, and it is often a dull process to design them. The transition has greatly freed the hands and minds of researchers, allowing them to focus more on the modeling aspect.

The promise of automatic representation learning is encouraging, but the real picture is not as bright. In practice, the network structure has a huge influence on the quality of the learned representations. The quality of learned representation also propagates to accuracy when the network is then fine-tuned on various downstream tasks. Therefore, designing the right neural network architecture is now crucially important, and as a result humans are tasked with designing architectures instead of features. In computer vision, the classical AlexNet is an example of intricate hand-designing convolutional neural networks – layers, kernel sizes, feature map sizes, depths, widths and other configurations are all carefully chosen and balanced by humans. Since then, various neural network architectures have been proposed, not only useful by themselves as feature extractors but also bringing new design principles. VGGNet [161] populates the usage of 3×3 kernel convolutions and serves as a pioneering example of homogeneous network designing. ResNet [64] introduces residual connections and made training networks with hundreds of layers possible. Transformers [181, 39] adopt multi-head self-attention as a new way of information exchange inside the network and has exceptional performance in large-scale training. Neural architecture search methods [218, 9] try to automate the design of neural architectures, and meanwhile borrow wisdom about search space designs [141] from human-designed networks. The successive innovations in this space, together with other training techniques, have advanced the top-1 image classification accuracy on ImageNet from AlexNet's 62.5% to near 90% nowadays. Among various objectives in architecture design, *efficiency* and *scalability* are two important concepts.

Efficiency is related to scaling models *down* while maintaining decent recognition capability, so that they can be deployed on edge devices and run at fast speed. A state-of-the-art model today usually requires large-scale parallel training with high-end GPU clusters, and the prohibiting cost measured by model size, memory consumption, and computing operations can sometimes significantly hinder their deployment in resource-constraint computing devices, such as mobile phones. Developing methods to boost models' efficiency is essential for such models' wider adoption. Researchers have pursued this goal with multiple approaches, including the design of new architectures specifically for mobile devices [79, 210], low-rank approximation of weights [35, 102], network pruning [62, 107], weight quantization [60, 28], adaptive computation [56, 43], knowledge distillation [77, 149], anytime recognition [94, 86], etc. With the increased amount of resources required by pre-trained large models, making them smaller and faster seems ever more in demand.

Scalability concerns how the models behave when equipped with huge model parameters and trained on very large data, i.e., when they are scaled *up*. An improperly designed network's performance can quickly saturate when we increase the data and compute available. Given the fast development of computing technologies, it would be outdated in a few years as its full potential is already reached with mediocre resources. Ideally, we would like the model accuracy to keep improving when scaled up. One classical example is the proposal of ResNets [64] compared with previous VGG-style [161] plain networks. For the latter, increasing the network depth causes both the training loss and test error to increase. Simple residual connections solved this problem and enabled the network to go much deeper and become more accurate. Recently, the community has shifted to vision Transformers, which discard certain ConvNet inductive biases and manifest better scalability. Despite the most prominent examples of scaling up are about designing new architectures, there are also other related research directions including algorithms for distributed training [54, 205], regularization [164, 84] and data augmentation [209, 29] techniques to ease large model overfitting.

Research progress in both directions has been fruitful, with many useful architecture design principles proposed and then adopted by later works. It has been a truly remarkable journey and the field is still evolving at an incredible speed. Meanwhile, partly because there are too many detailed design choices and hyperparameters in experiments, it is common to conduct a system-wide comparison on performance benchmarks, in which the researchers can choose favorable configurations and equip their methods with additional techniques. This may cause a failure to identify the source of empirical gains [112]. Baseline methods are sometimes not adequately tuned or adjusted, leading to our inability to understand the true effectiveness of the proposed method. In this dissertation, in addition to presenting a new framework for efficient architectures, we take a critical perspective and conduct empirical studies on methods or models that were considered trivial or old-fashioned baselines. We find them to be surprisingly competitive when supplied with the right techniques. This leads to a deeper understanding of certain new methods' underlying mechanisms, and helps us attribute their effectiveness more fairly and accurately.

1.1 Thesis Organization

In this thesis, we explore the design of neural network architectures for visual recognition from both efficiency and scalability perspectives. Chapter 2 proposes a novel framework for efficient and flexible ConvNet inference, while Chapter 3 and 4 focus on revisiting strong baseline methods. Despite each chapter having a different emphasis, we aim to provide guidelines or models for practical usage, and at the same time distill interesting insights.

In Chapter 2, we introduce an approach for pixel-wise dense prediction tasks in computer vision, that not only reduces the network's computational cost and makes them more efficient, but also allows the inference to be more flexible. In a typical machine learning setting, a user needs to wait until the model completes its inference process before seeing its first prediction. In certain scenarios this can be undesirable, e.g., an excessively long waiting time on a low-end device. We propose a framework that allows *anytime* prediction, i.e., the model can give intermediate outputs at any time during the inference when the user asks so. We adopt an early exiting framework from anytime image classification, and motivate our method by the fact that not all pixels need the same amount of computation. Some pixels are easier, and their predictions may be accurate enough in early exits, while others are harder and need deeper computing. We use prediction confidence as a difficulty indicator. The resulting anytime framework can reduce up to half of the original compute while maintaining the final prediction. The method can be considered "adaptive" as the computation spent on each image and each location is different, thus making the models more efficient.

In Chapter 3, we conduct an empirical study on popular neural network pruning methods. Pruning is widely used for reducing neural networks' sizes and compute. It differs from our anytime method above in that they are "static", i.e., the reduction of compute is solely on model and input-agnostic. The typical pruning pipeline is to train a large model first, then prune part of its weights according to a certain criterion, and finally fine-tune the remaining structure. Inheriting the kept weights from the large model is considered a necessity as those weight values are believed to be important. We conduct a thorough empirical study by comparing this traditional procedure with an embarrassingly simple baseline - training the smaller model from scratch. For structured pruning, to our surprise, this baseline can either match or beat the fine-tuned results. This prompts us to rethink the true value of network pruning. Our results show that pruning acts more like neural architecture search than selections of important weights, and encourages future research to employ more rigorous baseline comparisons.

In Chapter 4, we turn our eyes on ConvNet scalability. Vision Transformers are increasingly popular in computer vision since the year 2020, shadowing ConvNets. They exhibit superb scaling behavior when trained with large models and huge data. This desirable outcome is usually attributed to their self-attention mechanism, which is drastically different from convolutions. We take a closer look by analyzing other confounding factors – training recipes and detailed architecture designs. When we incorporate these strategies used by Transformers into a baseline ConvNet step by step, we observe consistent performance improvement. Our final models, which we name as ConvNeXts, are as scalable as vision Transformers. This study highlights the significance of seemingly detailed design choices, and showcases that when used collectively they can considerably change the behavior of an architecture. It also reminds us of the importance of ConvNet inductive biases in visual recognition.

Chapter 2

Anytime Dense Prediction by Confidence

2.1 Overview

In this chapter, we introduce a method for anytime dense visual recognition, which allows neural networks inference to be more flexible. Anytime inference requires a model to make a progression of predictions which might be halted at any time. Prior research on anytime visual recognition has mostly focused on image classification. We propose the first unified and end-to-end approach for anytime dense prediction. A cascade of "exits" is attached to the model to make multiple predictions. We redesign the exits to account for the depth and spatial resolution of the features for each exit. To reduce total computation, and make full use of prior predictions, we develop a novel spatially adaptive approach to avoid further computation on regions where early predictions are already sufficiently confident. Our full method, named anytime dense prediction with confidence (ADP-C), achieves the same level of final accuracy as the base model, and meanwhile significantly reduces total computation. We evaluate our method on Cityscapes semantic segmentation and MPII human pose estimation: ADP-C enables anytime inference without sacrificing accuracy while also reducing the total FLOPs of its base models by 44.4% and 59.1%. We compare with anytime inference by deep equilibrium networks and feature-based stochastic sampling, showing that ADP-C dominates both across the accuracy-computation curve. This chapter also serves as an example for typical efficient neural network inference methods, so that we are ready to present our work on examining scaling methods' mechanisms in the next chapters. Our code is available at `https://github.com/liuzhuang13/anytime`.

2.2 Introduction

Deep convolutional networks [99, 64] achieve high accuracy but at significant computational cost. Their computational burden hinders deployment, especially for time-critical or low-resource use cases that for instance require interactivity or inference on a mobile device. This efficiency problem is tackled by special-purpose libraries [20], compression by network pruning

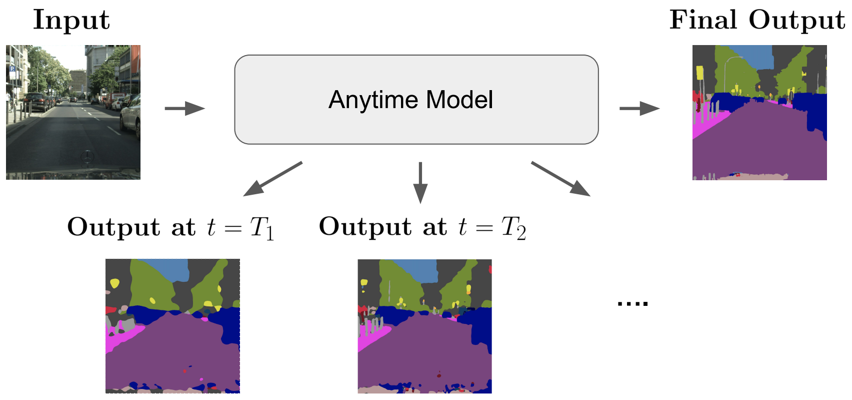


Figure 2.1: Anytime inference produces a progression of outputs.

[62, 107, 119], quantization [146, 90], and distillation [77, 149]. These solutions accelerate network computation but the entire network must still be computed; however, a prediction may be needed sooner. Time constraints vary, but the inference time of a standard deep network does not.

Anytime inference (Figure 2.1) mitigates this issue by bringing flexibility to model computation. An anytime algorithm [33] gradually improves its results as more computation time is given. It can be interrupted at any point during its computation to return a result as system or user requirements demand. In this way, the time to the first output is reduced while the quality of the last output is preserved.

An anytime model makes a progression of predictions between the first and last. This progression continues if time remains, or halts if it is either already satisfactory or out of time. For example, consider a user on a mobile device: an approximate result could be returned earlier if there is urgency, or the user could monitor the sequence of predictions as time goes by and stop the model once it is good enough. Note that anytime inference differs from *adaptive* or *dynamic* inference [182, 197, 187] where the *model* decides how much to compute instead of an *external* decision.

Prior research has explored anytime inference by feature selection [94] or ensembling models through boosting [58]. For end-to-end neural network models, research has focused on classification for anytime inference or adaptive inference. In particular, the multi-scale dense network [86] is an architecture for resource-efficient classification. The attraction of anytime inference is not limited to classification however, and the additional computation required for dense prediction tasks makes it even more desirable. For instance, an autonomous driving system may demand swifter reaction time for safety in the presence of pedestrians, and so an anytime semantic segmentor might sooner recognize their presence. In addition to urgency, an anytime segmentor could help efficiency, by not further processing already confident predictions of street pixels and therefore save power.

In this work, we develop the first single-model anytime approach for dense prediction tasks. We adopt an early exiting framework, where multiple predictors branch off from the intermediate stages of the model. The exits are trained end-to-end (both the original exit and intermediate exits), and during inference, each provides a prediction in turn. To compensate for differences in depth and spatial dimensions across stages, we redesign the predictors for earlier exits. For each exit, we choose an encoder-decoder architecture to enlarge receptive fields and smooth spatial noise.

Exits might suffice for anytime image classification, but dense prediction tasks have spatial structures. Simple regions may need less processing while complex ones need more. Standard inference applies an equal amount of computation at every pixel without taking advantage of spatial structure. Our spatially adaptive anytime inference scheme decides whether or not to continue computation at each exit and position. We mask the output of each exit by thresholding the confidence of its predictions: the remaining computation for sufficiently confident pixels is then reduced (Figure 2.2). For each masked pixel, its prediction will be persisted in the following exits, as it is already sufficiently confident. In the following layers, the features for the masked pixel will be interpolated, rather than convolved, and therefore reduce computation. The confidence measure can depend on the task, e.g., in segmentation, it could be the entropy of class predictions. This *confidence adaptivity* can substantially reduce the total computation while maintaining accuracy.

We experiment with two dense prediction tasks: Cityscapes semantic segmentation and MPII human pose estimation. Redesigning the exits and including confidence adaptivity significantly improves across accuracy-efficiency operating points. Our full approach, named anytime dense prediction with confidence (ADP-C), not only makes anytime predictions, but its final predictions achieve the same level of accuracy as the base model, with 40-60% *less* total computation. For analysis, we visualize predictions and confidence adaptivity across exits, and ablate design choices for the exits and masking.

2.3 Approach

2.3.1 Anytime Setting

In an anytime inference setting, the user can stop the inference process based on the input or a current event. Thus the computation budget for each instance x could be time or input-dependent. We use $B(x,t)$ to denote the computation budget assigned for instance x at time t, where the time variable t models events that can change the budget. $B(x,t)$ could be independent of x, i.e., the budget only depends on the time t, for example if a model on a server is asked to make predictions with less budget during high-traffic hours; $B(x,t)$ can also be independent of t, meaning the budget is only decided by input x, regardless of external events. The output of the anytime model depends on the budget given, and we denote it as $f(x, B(x,t))$. Assuming L is the task loss and y is the ground truth, the per-instance loss is $L(f\left(x, B\left(x,t\right)\right), y)$. This leads to the expected training loss to be $\mathbb{E}_{(x,y)\sim(X,Y),t\sim T}[L(f\left(x, B\left(x,t\right)\right), y)]$, where (X,Y) is the input-output joint distribution and

T is the distribution modeling the time or event variable.

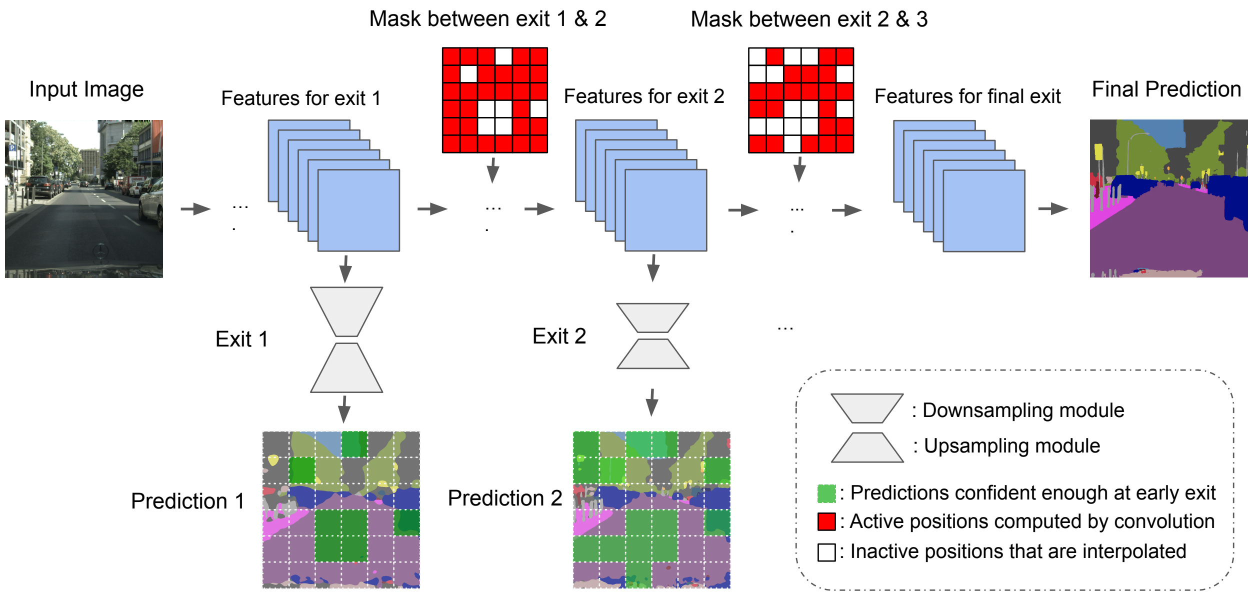


Figure 2.2: Proposed anytime dense prediction with confidence (ADP-C) approach. We equip the model with intermediate exits for anytime inference. We redesign each exit with encoder-decoder architecture to compensate for spatial resolution across model stages. At each exit's output, sufficiently confident predictions (green squares) are identified to skip further computation in the following layers.

2.3.2 Early Exiting

Next, we introduce the early exiting framework which has been used in prior works [86, 173] for anytime prediction. Standard convolutional networks only have one prediction "head" at its final stage. The network takes the input x, forwards it through intermediate layers, and finally outputs the prediction at its head. The concrete form of the head depends on the task. For dense prediction, the head is usually one or multiple convolutions that output spatial maps representing pixel-wise predictions.

To obtain an anytime model, we attach multiple heads to the network, branching from its intermediate features (Figure 2.2). We call these additional heads *early exits*, since they allow the network to give early predictions and stop the inference at the current layer. Suppose we add k early exits at intermediate layers with layer indices $l_1 \dots, l_k$. We denote the intermediate features at these layers $F_{l_1}(x) \dots, F_{l_k}(x)$, and the functions represented by the early exits $E_1 \dots, E_k$. Note that E_is may be of the same form but they do not share weights. The early prediction maps can be denoted as $\hat{y}_i = E_i(F_{l_i}(x)), i = 1 \dots k$. Together with the original final prediction $\hat{y}_{k+1}$, the total loss is:

$$L_{total} = \sum_{i=1}^{k+1} w_i L(\hat{y}_i, y) \tag{2.1}$$

where w_i is the weight coefficient at exit i. The original network, together with the added exits, will be trained end-to-end to optimize this total loss function. In experiments, we set all weights equal to 1. This corresponds to the minimization of the expected loss in Section 2.3.1 when the exiting probabilities at all exits are equal. We find this to be a simple yet effective scheme.

For anytime inference, as the network propagates features through its layers, if the computation budget is reached or the user asks the model to stop, it will output the latest $\hat{y}_i$ that is already computed. Similar early exiting strategies have been used in resource-efficient image classification [173, 86], but dense prediction tasks require further steps detailed in the following subsections.

2.3.3 Head Redesign

Typical convolutional networks have a hierarchical structure that begins with shallow, fine, and more local features and ends with deep, coarse, and more global features. These deeper features represent more image content by their larger receptive fields. For dense prediction, upsampling is done within the network to restore lost resolution during downsampling, and ensure precise spatial correspondence between the input and the output. This upsampling can be accomplished in a few [120] or many [212] layers, but no matter the architecture, the network learns its most local features in its earliest layers. This presents a challenge for the earliest exits, since these features are limited in depth and receptive field. Making direct predictions at these exits with the typical 1×1 convolution head produces spatially noisy and inaccurate results.

To compensate for these lacking early features, we redesign the prediction heads for the exits E_i. Each E_i first downsamples its input features $F_{l_i}(x)$, through a series of pooling and 1×1 convolution layers. Each pooling operation halves the spatial resolution, increasing its output's receptive fields. The following convolution provides the opportunity to learn new coarser-level features, specifically for that exit's prediction. After several (denoting this number as D) "pool-conv" layers, we upsample the features back to the original output resolution, with an equal number (D) of bilinear interpolation and 1×1 convolution layers. The output of this "interpolate-conv" sequence will be the prediction $\hat{y}_i$ at this exit. This is important for ensuring spatial accuracy for pixel-level dense prediction tasks. Our redesigned exits are essentially small "encoder-decoder" modules (Figure 2.2), where the encoder downsamples the features, the decoder upsamples them back.

The downsampling ratio at each exit is determined by D, the number of consecutive "pool-conv" layers. Intuitively, features at earlier layers are more fine-level, and the exit branching from them can potentially benefit from more downsampling. In experiments, we use an encoder with $D = N - i$ downsampling operations at exit i, where N is the total number of exits, including the original last exit. Empirically we find this strategy works well, and alternative strategies are compared in Section 2.5.

Finally, in all early exits, the first convolution will transform the number of channels to a fixed number for all exits. By setting the channel width relatively small, we can still save computation while adding layers with this redesigned encoder-decoder head structure.

2.3.4 Confidence Adaptivity

For dense prediction tasks, any early prediction $\hat{y}_i$ is a spatial map consisting of pixel-wise predictions at each position. While most convolution networks spend an equal amount of computation at each input position, it is likely that recognition at some regions are easier than others, where the network can make predictions with high confidence even at earlier exits. For instance, the inner part of a large sky segment may be easy to recognize, whereas the boundary between the bicycle and the person riding it may need more careful delineation.

Once an early prediction is made, we can inspect the "confidence" at each position. As an example, for semantic segmentation, the maximum probability over all classes can serve as a confidence measure. If the confidence has passed a pre-defined threshold at certain positions (green squares on predictions in Figure 2.2), we may decide these predictions are likely to be correct, and not continue the computation of further layers at this position. Suppose the pixels of the early prediction $\hat{y}_i$ are indexed by p, we form a mask M_i:

$$M_i(p) = \begin{cases} 0, & \text{if Confidence}(\hat{y}_i(p)) \geq \text{Threshold} \\ 1, & \text{otherwise} \end{cases} \tag{2.2}$$

For any convolution layer between exit i (E_i) and the next exit $i+1$ (E_{i+1}), we could choose whether to perform or skip computation at position p based on the mask (Figure 2.2). Assuming C is a convolution layer with input f_{in}, then by applying the mask, the output f_{out} at position p becomes:

$$f_{out}(p) = \begin{cases} C(f_{in})(p), & \text{if } M_i(p) = 1, \\ 0, & \text{if } M_i(p) = 0. \end{cases} \tag{2.3}$$

If C's output and the mask M_i do not share the same spatial size, we interpolate $\hat{y}_i$ in Eqn. 2.2 to the size of C's output, so that the mask M_i is compatible with C in Eqn. 2.3.

The output f_{out} could be sparse, with many positions being 0. This could potentially harm further convolutional computation. To compensate for this, we spatially interpolate these positions from their neighbors across all channels, using a similar approach as in [202]. Denoting the interpolation operation as I, the final output feature f^*_{out} is

$$f^*_{out}(p) = \begin{cases} f_{out}(p), & \text{if } M_i(p) = 1, \\ I(f_{out})(p), & \text{if } M_i(p) = 0. \end{cases} \tag{2.4}$$

Here, the value of $I(f_{out})(p)$ is a weighted average of all the neighboring pixels centered at p within a radius r:

$$I(f_{out})(p) = \frac{\sum_{s\in\Omega(p)} W_{(p,s)} f_{out}(s)}{\sum_{s\in\Omega(p)} W_{(p,s)}} \tag{2.5}$$

where s indexes p's neighboring pixels and $\Omega(p) = \{s | \|s-p\|_\infty \leq r, s \neq p\}$, the neighborhood of p. We set radius $r = 7$ in our experiments. $W_{(p,s)}$ is the weight assigned to point s for interpolating at p, for which we use the RBF kernel, a distance-based exponential decaying weighting scheme:

$$W_{(p,s)} = \exp\left(-\lambda^2 \|p-s\|_2^2\right) \tag{2.6}$$

with λ being a trainable parameter. This indicates that the closer s is to p, the larger its assigned weight will be. Note that masked-out features ($M_i(p) = 0$) still participate in the interpolation process as inputs with values of 0.

Replacing filtering by interpolation at these already confident spatial locations ($M_i(p) = 0$) could potentially save a substantial amount of computation. The mask M_i will be used for all convolutions between exit i and $i+1$, including the convolutions inside exit $i+1$. Once the forward pass arrives at the next exit, to make the prediction $\hat{y}_{i+1}$, the last prediction at positions where $M_i(p) = 0$ will be carried over, having already been deemed confident enough at the last exit and having been skipped during further computations. This means:

$$\hat{y}_{i+1}(p) = \begin{cases} E_{i+1}(F_{l_{i+1}}(x)), & \text{if } M_i(p) = 1, \\ \hat{y}_i(p), & \text{if } M_i(p) = 0. \end{cases} \tag{2.7}$$

The network then calculates a new mask M_{i+1} based on $\hat{y}_{i+1}$, and uses it to skip computation going forward. The process continues until we reach the final exit.

In summary, we incorporate spatial *confidence adaptivity* into the early exiting network, by not filtering at spatial locations that are already sufficiently confident in the latest prediction. At these positions interpolation is used instead, at a much reduced computational cost, to avoid excessive sparsity. Unless otherwise specified, confidence adaptivity is used in both training and inference. We dub our full approach as anytime dense prediction with confidence (ADP-C).

2.4 Experiments

We evaluate ADP-C with two dense prediction tasks: semantic segmentation and human pose estimation. Our experiments are implemented using PyTorch [136].

2.4.1 Architectures

We use the High-Resolution Network (HRNet) [185] architecture as our base model. HRNet is a multi-stage architecture, where each stage adds lower-resolution/larger-scale features.

Specifically, we adopt the standard HRNet-{W48,W32} models and the smaller HRNet-W18 model. HRNet-W48 is state-of-the-art for semantic segmentation and HRNet-W32 is suitable for pose estimation. HRNet-W18 is highly efficient and has been shown to outperform other efficient networks [211, 154] in its accuracy-efficiency tradeoff. The 48/32/18 denotes the number of channels in the bottleneck of the first stage. The original head for HRNet before our redesigning is two consecutive 1×1 convolutions. We attach three exits, one at the end of each stage before the final prediction. We follow the training/evaluation protocol and hyperparameters of the reference HRNet implementation at [167, 185] (except that our models include a loss at each exit).

2.4.2 Baselines

HRNet. We compare with a standard HRNet that has only one (final) exit with the same backbone architecture. The standard HRNet is not anytime, so we focus on comparing it with our anytime model's final exit.

MDEQ. MDEQ [8] is a recent deep *implicit* model, which achieves competitive performance on vision tasks without stacking explicit layers, but rather solves an optimization problem for inference. Its representation z^* is an equilibrium point of its learned transformation $f(z; x)$, i.e., $f(z^*; x) = z^*$ where x is the input. The representation is obtained by iteratively solving the equation $f(z; x) = z$, for which the quality of the solution improves with more iterations. The converged representation is then decoded into a prediction. We examine anytime prediction with the MDEQ by decoding intermediate iterates of the representation. To the best of our knowledge, this is the first study of anytime implicit modeling, as [8] only reports the predictions of implicit models at equilibrium, and does not produce or inspect intermediate predictions. We use the "small" version of the MDEQ [8] and the 4th, 6th, 8th, and 10th iterations of its equilibrium optimization to bound the amount of computation and align its iterations with our architecture's stages.

Feature-Based Stochastic Sampling. We follow [202] in using internal features to predict masking positions, with the Gumbel-Softmax trick [91] used for sampling. We use a 3×3 convolution upon the features for exits (Figure 2.2) for mask prediction until the next exit. During training, an L_1 sparsity regularization is applied on the stochastic continuous mask values, and only during inference the mask values are discretized. The interpolation procedure is the same as in our method. This baseline is evaluated on HRNet-W18.

2.4.3 Experimental Settings

For Cityscapes semantic segmentation, we follow the training settings at the official codebase [50] of HRNet for semantic segmentation. The HRNet-W18/48 models are pre-trained on ImageNet. During training, multi-scale and flipping data augmentation is used, and the input cropping size is 512×1024. The model is trained for 484 epochs, with an initial learning rate of 0.01 and a polynomial schedule of power 0.9, a weight decay of 0.0005, a batch size of 12, optimized by SGD with 0.9 momentum. In evaluation, we use single-scale testing without flipping, with input resolution 1024×2048. For the Feature-Based Stochastic Sampling baseline, we modify the exit weights from $(1, 1, 1, 1)$ to $(0.5, 0.5, 0.5, 1)$ as we find it produces more stable masking values during training. The L_1 sparsity regularization on masks is set to 0.1. The mask outputs at exit $(1, 2, 3)$ have additional weight factors of $(1/3, 2/3, 1)$ to encourage more sparse features towards the end.

For MPII human pose estimation, we follow the training settings at the official codebase [49] of HRNet for pose estimation. The HRNet-32 model we use is also pre-trained on ImageNet. The image size for both training and evaluation is 256×256. The model is trained for 210 epochs, with an initial learning rate of 0.001, and decaying of 0.1 at epoch 170 and 200. The optimization is done by Adam with $\gamma_1 = 0.99, \gamma_2 = 0$, a weight decay of 0.0001, and a momentum of 0.9. The batch size is 128. In evaluation, flipping test is used.

2.4.4 Semantic Segmentation

The Cityscapes dataset [26] consists of 2048×1024 images of urban street scenes with segmentation annotations of 19 classes. We train the models with the training set and report results on the validation set. The accuracy metric is the standard mean intersection-over-union (mIoU %), and the computation metric is the number of floating-point operations (FLOPs). Anytime inference improves with higher accuracy, less computation, and more predictions. We evaluate HRNet-W48 and HRNet-W18 for this task.

Redesigned heads (RH) use our encoder-decoder structure for exits. Since we have 4 exits in total, we repeat the downsampling operation 3/2/1 times at exit 1/2/3 to generate larger-scale features for earlier exits, as described in Section 2.3.3. We set the number of channels at all exits to 128/64 for HRNet-W48/W18. For confidence adaptivity (CA), we use the maximum probability among all classes as the confidence measure, and set the confidence threshold in Eqn. 2.2 to be 99.8% based on cross-validation. For CA, the computation for each input can differ, so we report the average FLOPs across all validation images at each exit.

The results for HRNet-W48 are shown in Table 2.1. We observe that our early exiting model based on HRNet-W48 outperforms the MDEQ model by a large margin, with significantly less FLOPs at each exit. With RH, we achieve notable accuracy gain in early predictions, especially at the first exit (more than 10%), with roughly the same computation. With CA added, we arrive at our full ADP-C method (RH + CA), which maintains roughly the same accuracy as the RH model but reduces the total computation at exits 3 and 4. Interestingly ADP-C has slightly higher mIoU at the final exit (81.3 vs. 80.7) with 44.4% less total computation (387.1 vs. 696.2 GFLOPs) compared to the base HRNet. This is possibly due to a potential regularization effect of confidence adaptivity: computing fewer intermediate features exactly may prevent overfitting.

The same results are plotted in Figure 2.3 (left). The plot shows accuracy (y-axis) and computation (x-axis) tradeoffs: points to the upper left indicate better anytime performance. The baseline HRNet is represented by a red cross, while anytime models are plotted as curves

		Accuracy (mIoU)					Computation (GFLOPs)				
Method / Output		1	2	3	4	Avg	1	2	3	4	Avg
One-exit	HRNet-W48 [185]	-	-	-	80.7	-	-	-	-	696.2	-
Baselines	MDEQ-Small [8]	17.3	38.7	65.5	72.4	48.5	521.6	717.9	914.2	1110.5	816.0
	EE (HRNet)	34.3	59.0	76.9	80.4	62.7	48.4	113.4	388.9	722.2	318.2
Ours	EE + RH (HRNet)	44.6	60.2	76.6	79.9	65.3	41.9	105.6	368.0	701.3	304.2
	ADP-C: EE + RH + CA (HRNet)	44.3	60.1	76.8	**81.3**	**65.7**	41.9	93.9	259.3	**387.1**	**195.6**

Table 2.1: Accuracy and inference computation for Cityscapes semantic segmentation with four exits. Our approach achieves higher accuracy in less computation than the HRNet and MDEQ baselines across exits. Early exiting (EE) makes progressive predictions. Redesigned heads (RH) improve early predictions (exits 1 and 2). Confidence Adaptivity (CA) reduces computation.

with a point for each prediction. We plot the results for the smaller HRNet-W18 model in Figure 2.3 (middle). RH improves early prediction accuracy from the basic early exiting model, and CA substantially reduces computation at later exits. The full model reaches the same-level of accuracy as the baseline HRNet with much less total computation. We note that our model with confidence values as mask indicators also outperforms the feature-based mask sampling method in the accuracy-computation tradeoff, demonstrating that confidence is effective at filtering out redundant computation despite its simplicity.

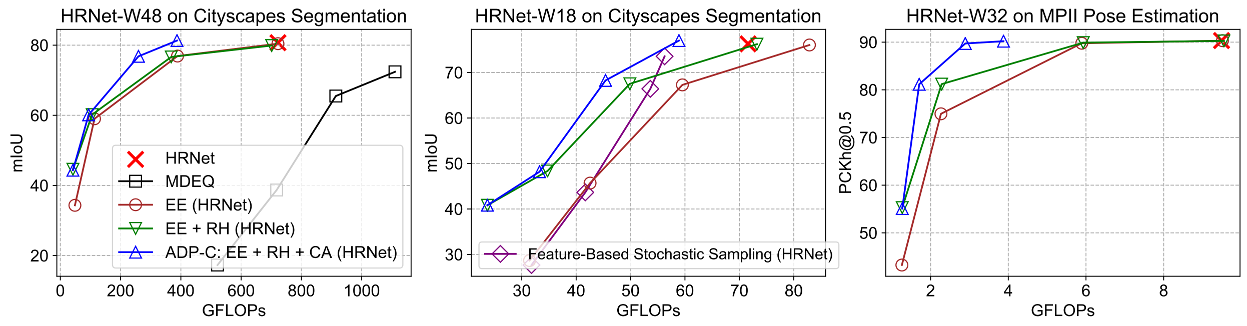


Figure 2.3: Accuracy and computation at four exits across architectures and tasks. Redesigned heads (RH) boost the accuracy at early exits, while confidence adaptivity (CA) reduces computation by up to more than half. ADP-C outperforms baseline methods across the accuracy-computation tradeoff curve.

Our experiments measure computation by FLOPs rather than time. Reporting FLOPs is common [43, 202, 86, 187, 119] and meaningful because it is hardware independent. However, similarly to spatially adaptive computation methods [43, 202], our model does not achieve wall clock speedup at this time due to the lack of software/hardware support for sparse convolution with current frameworks and GPU devices. To approximate CPU speedup, we conduct a profiling experiment on a multi-threading processor (specifically we measure computation time on a Linux machine with Intel Xeon Gold 5220R CPUs using 16 threads). We replace all convolutions with our implementations following [202]. ADP-C on HRNet-W48 achieves 1.48× speedup compared to the non-anytime baseline, measured in end-to-end latency (wall-clock time). There is a gap between this measured time and the theoretical 1.80× speedup measured by FLOPs. ADP-C and others can benefit from ongoing and future work on efficient sparse convolutions [55, 23, 183, 40]. We also refer readers to the Hardware Lottery [78] for a discussion on how hardware compatibility affects progress in AI research. Please see the supplement for an anytime inference video where each exit is timed to the computation it requires.

2.4.5 Human Pose Estimation

For human pose estimation, we evaluate on the MPII Human Pose dataset [4] of image crops annotated with body joints collected from everyday human activities. The positions of 16 joint types are annotated for the human-centered in each crop. We report the standard metric [4] for MPII, the PCKh (head-normalized probability of correct keypoint) score, on its validation set. We use HRNet-W32 for this task and follow the reference settings from [167]. The standard head for this task is 1×1 convolution. As in segmentation, our redesigned heads are encoder-decoder structures. The number of channels for all exits is 64.

Pose estimation task is formulated as regression. The HRNet model outputs 16 spatial feature maps, each one regressing the corresponding body joint. The only positive target for each type is coded as 1; all other points are negatives coded as 0. Unlike in segmentation, the output at each pixel is not a probability distribution, so we use the maximum value across channels as the confidence measure. A pixel is masked out if the maximum value at that position is smaller than the threshold, marking it unlikely to be a joint prediction. We choose 0.002 as the threshold by cross-validation, as a larger value makes the mask too sparse and hurts learning. The RH + CA model adopts adaptivity after 10 epochs of normal training, because nearly all outputs are too close to zero in the beginning.

Figure 2.3 (right) and Table 2.2 show the results. We observe a similar trend to segmentation: RH improves accuracy and CA reduces FLOPs. In this case, ADP-C reduces computation by 59.1% (9.49 to 3.88 GFLOPs) while accuracy only drops by 0.13% relative to the baseline HRNet.

2.5 Analysis

2.5.1 Visualizations

To inspect our anytime predictions and masking on Cityscapes, we visualize ADP-C exit results on a validation image with HRNet-W48. Figure 2.4 shows the predictions, confidence

	PCKh@0.5		GFLOPs	
Method / Output	Last	Avg	Last	Avg
HRNet-W32 [185]	90.33	-	9.49	-
EE (HRNet)	90.31	74.60	9.51	4.73
EE + RH (HRNet)	90.26	79.16	9.55	4.76
ADP-C: EE + RH + CA (HRNet)	90.20	79.04	3.88	2.44

Table 2.2: Accuracy and computation on MPII pose estimation at the last exit and averaged for all exits. Early exits (EE) make progress predictions, redesigned heads (RH) improve accuracy, and confidence adaptivity (CA) reduces computation.

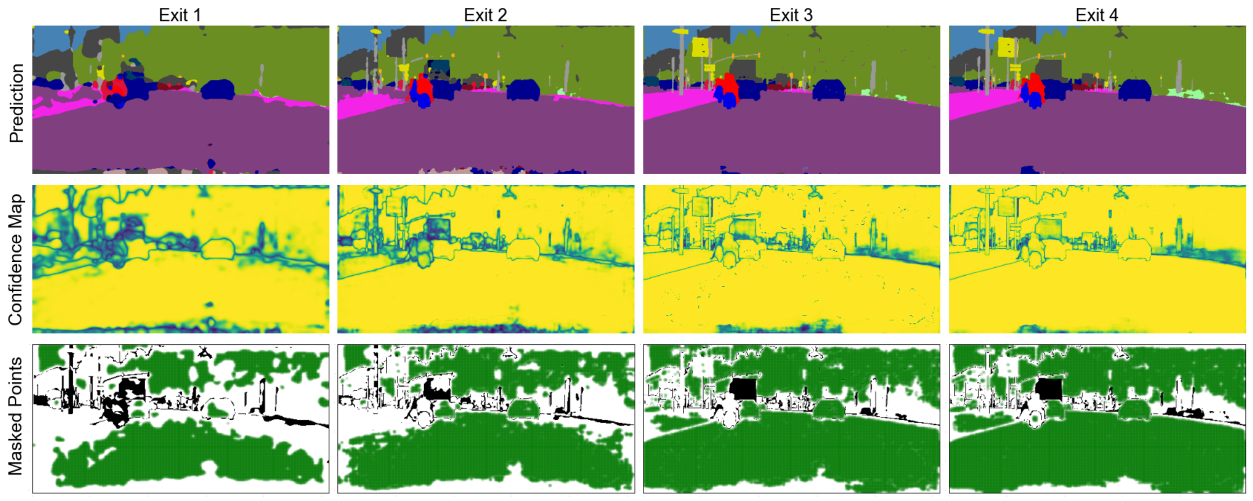


Figure 2.4: Visualizations of ADP-C results. Top: prediction results at all exits. Middle: confidence maps, lighter color indicates higher confidence. Bottom: correct/wrong predictions at the exit drawn as white/black. The confident points selected for masking are in green. Confidence adaptivity excludes calculation on already confident pixels (green) in early exits, mostly located at inner parts of large segments.

maps, and computation masks across exits. With each exit, the prediction accuracy improves, especially in more detailed areas with more segments. The confidence maps are shown with high lighter/yellow and low darker/green. Most unconfident points lie around segment boundaries, and the interior of large stuff segments (road, vegetation) are already confident at early exits. This motivates the use of confidence adaptivity to avoid unnecessary further computations on these areas. For computation masks, the correct/incorrect predictions at each exit are marked white/black. Pixels surpassing the confidence threshold (99.8%) are masked and marked green. Many pixels can be masked out in this way, and each exit masks more. Most of the masked pixels are found in the inner parts of large segments or already correct areas. In fact, the masked pixels are 100% correct at all exits for this instance, which partly justifies their exclusion from later computation. The predictions at these positions are already confident and correct at early exits, and so the only potential harm of skipping their computation later is their possible effect at less confident positions. On average, 19.3%, 38.4%, 63.0% pixels are masked out at exit 1, 2, 3, respectively.

2.5.2 Downsampling at Early Exits.

In Section 2.3.3, we described how many consecutive downsampling operations we use at each exit, by $D = N - i$, which means we use $D = 3/2/1$ consecutive "pool-conv" layers for downsampling at exit 1/2/3. Here we compare this strategy with $D = 1/1/1$ and 3/3/3, where the same level of downsampling and hence the same head structure is used at all exits, on HRNet-W18. Figure 2.5 (left) shows that our adopted $D = 3/2/1$ strategy obtains the highest accuracy at all exits among these choices.

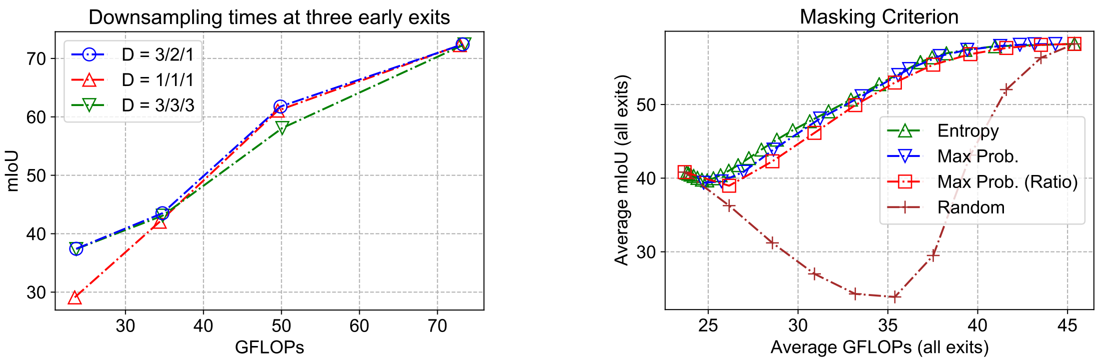


Figure 2.5: Analysis of ADP-C. *Left:* comparing downsampling strategies. $D = 3/2/1$ means downsampling the features 3/2/1 times at exit 1/2/3. *Right:* comparison between different masking criteria.

2.5.3 Masking Criterion

We used the max probability as the confidence measure and a fixed threshold for masking. Here we consider a few alternatives. One is to mask out the top $k\%$ (by max prob.) of the pixels at each exit, regardless of their values. We also consider thresholding on the entropy of the probability distribution. In addition, we compare them with random masking. We use HRNet-W18, change the ratio or threshold for a wide range, and present the average mIoU vs. GFLOPs on all exits at Figure 2.5 (right). For this ablation, adaptivity is only applied during inference.

First, we notice all three confidence criteria largely outperform random masking. For max probability, using a threshold performs slightly better than a fixed ratio, possibly because this gives the flexibility for different exits to mask out different amounts of points. Finally, we observe using entropy as the confidence measure performs similarly to using max probability, but we stick to max probability in ADP-C because it is trivial to compute.

2.6 Related Work

2.6.1 Anytime Inference

Anytime algorithms [217, 33] can be interrupted at any point during computation to return a result, whose quality improves gradually with more computation time. In machine learning, anytime inference has been achieved by boosting [58], reinforcement learning [94], and random forests [45]. Anytime deep networks have been brought to bear on image classification, but not dense prediction. Branching architectures have been a common strategy [3, 173] along with other techniques such as adaptive loss balancing [80]. While there is work on the tasks of person re-identification [188] and stereo depth [189], these techniques are task-specific, while our method applies to multiple dense prediction tasks, as we show with semantic segmentation and pose estimation. Liu et al. [113] learn a hierarchy of models for anytime segmentation, but its multiple models complicate training and testing, and require more memory. Our work instead augments the base model architecture for simplicity and efficiency. The PointRend method [96] outputs an initial dense prediction first and then refine it adaptively, but the predictions are all made at full depth. Its majority of the computation is spent before the first output and thus cannot be practically "anytime". Our method is the first to selectively update anytime predictions across space and layers.

2.6.2 Adaptive Computation

An *adaptive* model adjusts its computation to each specific instance during inference. For deep networks, this is often done by adjusting which layers to execute, that is, choosing which layers to run or skip. This can be done by a supervised controller [182, 116], a routing policy optimized by reinforcement learning [187, 197, 109], or other training strategies [127]. Rather than choosing layers, *spatial adaptivity* chooses where to adjust the amount of computation across different spatial positions in the input. For example, the model could infer spatial masks for feature maps and skip computation on masked areas [158, 38, 111, 147, 15]. [43] maintains a halting score at each pixel and once it reaches a threshold the model will stop inference at those positions for spatially coarse tasks like classification or bounding box detection. [202] stochastically sample positions for computation from an end-to-end learned sampling distribution. [108] convert a deep network into a difficulty-aware cascade, where earlier steps handle easier regions and later steps tackle harder regions. These spatially adaptive models reduce computation, but are not anytime: they do not make a series of predictions and cannot be interrupted.

2.7 Additional Studies

2.7.1 Ablation on Interpolation Radius

We use a default radius of $r = 7$ when interpolating masked-out features (Eqn. 2.6 and following text). Here we demonstrate that the radius of 7 is reasonable through an ablation experiment, whose results are listed in Table 2.3. The experiment is conducted with HRNet-W18 on Cityscapes semantic segmentation, with both redesigned head (RH) and confidence adaptivity (CA). We observe that a radius of 7 outperforms lower ones (3 and 5) in the average mIoU slightly, with minimal FLOPs addition, because interpolation is done channel-wise. A further increase of radius to 9 does not bring significant gain in final or average mIoU.

	Accuracy (mIoU)					Computation (GFLOPs)				
Radius	1	2	3	4	Avg	1	2	3	4	Avg
3	41.06	48.25	67.56	75.82	58.17	23.7	33.0	44.4	57.0	39.5
5	40.86	48.01	67.64	76.21	58.18	23.7	33.1	44.6	57.4	39.7
7 (default)	41.05	48.35	67.73	76.10	58.31	23.7	33.1	44.9	58.1	40.0
9	41.01	48.41	67.88	76.08	58.35	23.7	33.2	45.4	59.1	40.4

Table 2.3: Accuracy and inference computation for Cityscapes semantic segmentation with four exits under different settings of interpolation radius. A minor increase in mIoU and GFLOPs is observed with a larger radius.

2.7.2 Inference-Only Confidence Adaptivity

	Accuracy (mIoU)					Computation (GFLOPs)				
Adaptivity	1	2	3	4	Avg	1	2	3	4	Avg
No Adaptivity	44.61	60.19	76.64	79.89	65.33	41.9	105.6	368.0	701.4	304.2
Training and Inference	44.34	60.13	76.82	81.31	65.65	41.9	93.9	259.4	387.1	195.6
Inference-only	44.61	59.97	76.37	79.69	65.16	41.9	94.1	291.8	484.8	228.1

Table 2.4: Accuracy and inference computation for Cityscapes semantic segmentation with four exits under different adaptivity settings.

By default, confidence adaptivity is used in both training and inference. Here we compare this with the setting where adaptivity is only used for inference in Table 2.4. We use HRNet-W48 with redesigned heads (RH) on Cityscapes for this experiment. "No Adaptivity" corresponds to the "EE + RH" row in Table 2.1. We observe that using adaptivity only at inference hurts the accuracy, compared with using it at both training and inference. It also increases the average FLOPs at exit 3 and 4.

2.7.3 PASCAL-Context Results

We present results with the PASCAL-Context semantic segmentation dataset [132] in Table 2.5. It consists of 59 segmentation classes. For the Early Exiting baseline, we use weights of (0.33, 0.33, 0.33, 1) for 4 exits, respectively, as we found increasing it to all 1 would hurt the final performance too much. The confidence threshold is set to 99.5%. We observe RH improve the accuracy in most exits. ADP-C outperforms the Early Exiting baseline in average mIoU, despite its final mIoU is worse. It also saves more than 20% FLOPs compared with the vanilla Early Exiting baseline.

		Accuracy (mIoU)					Computation (GFLOPs)				
Method / Output		1	2	3	4	Avg	1	2	3	4	Avg
One-exit	HRNet-W48 [185]	-	-	-	51.35	-	-	-	-	76.5	-
Baseline	Early Exiting (HRNet)	8.78	18.74	39.82	50.10	29.36	5.4	12.6	43.1	80.1	35.3
Ours	EE + RH (HRNet)	14.55	19.42	39.30	50.32	30.90	4.5	11.4	40.2	77.2	33.3
	ADP-C: EE + RH + CA (HRNet)	14.51	19.18	39.00	49.27	30.49	4.5	11.1	38.1	**62.7**	29.1

Table 2.5: Accuracy and inference computation for PASCAL-Context semantic segmentation.

2.7.4 More Visualizations

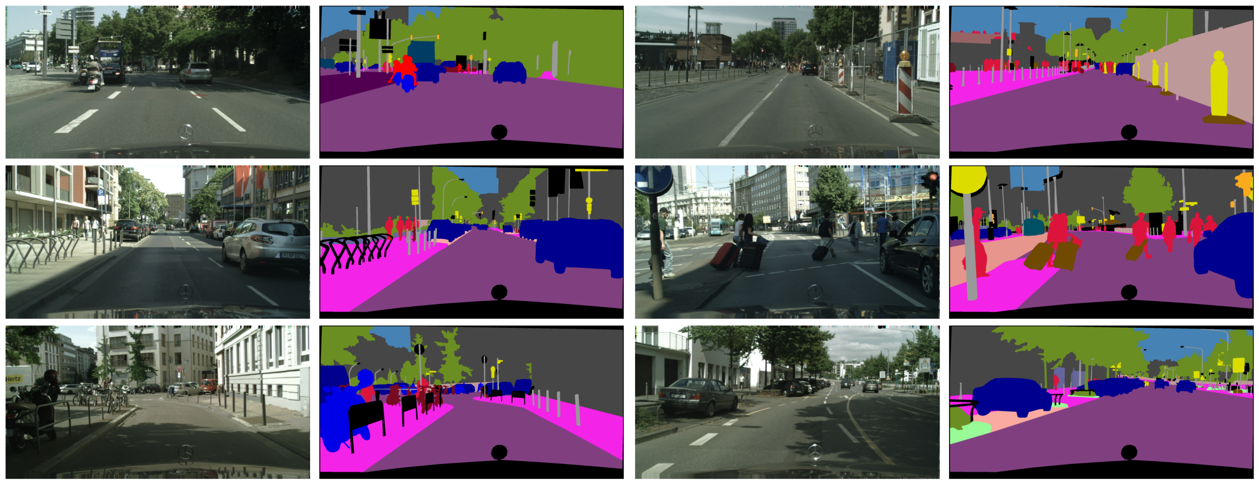


Figure 2.6: Input and ground truth of the example validation images visualized in Figure 2.7 and Figure 2.8

We present more visualizations of the same type as Figure 2.4, in Figure 2.7 and 2.8. Their input and ground truth are shown in Figure 2.6 in the same order. We can see the same trend as discussed previously still holds: the model will mask out confident points that are inside large segments (e.g., road, vegetable), which are mostly already predicted correctly in early exits.

2.8 Conclusion

We propose ADP-C, the first single-model anytime approach for dense visual prediction. Based on an early-exiting framework, our redesigned exiting heads and confidence adaptivity both improve the accuracy-computation tradeoff. On Cityscapes semantic segmentation and MPII pose estimation, ADP-C achieves 40%-60% FLOPs reduction with the same-level final accuracy, compared to the baseline HRNet. We further analyze confidence adaptivity with visualizations and ablate key design choices to justify our approach to anytime inference with confidence.

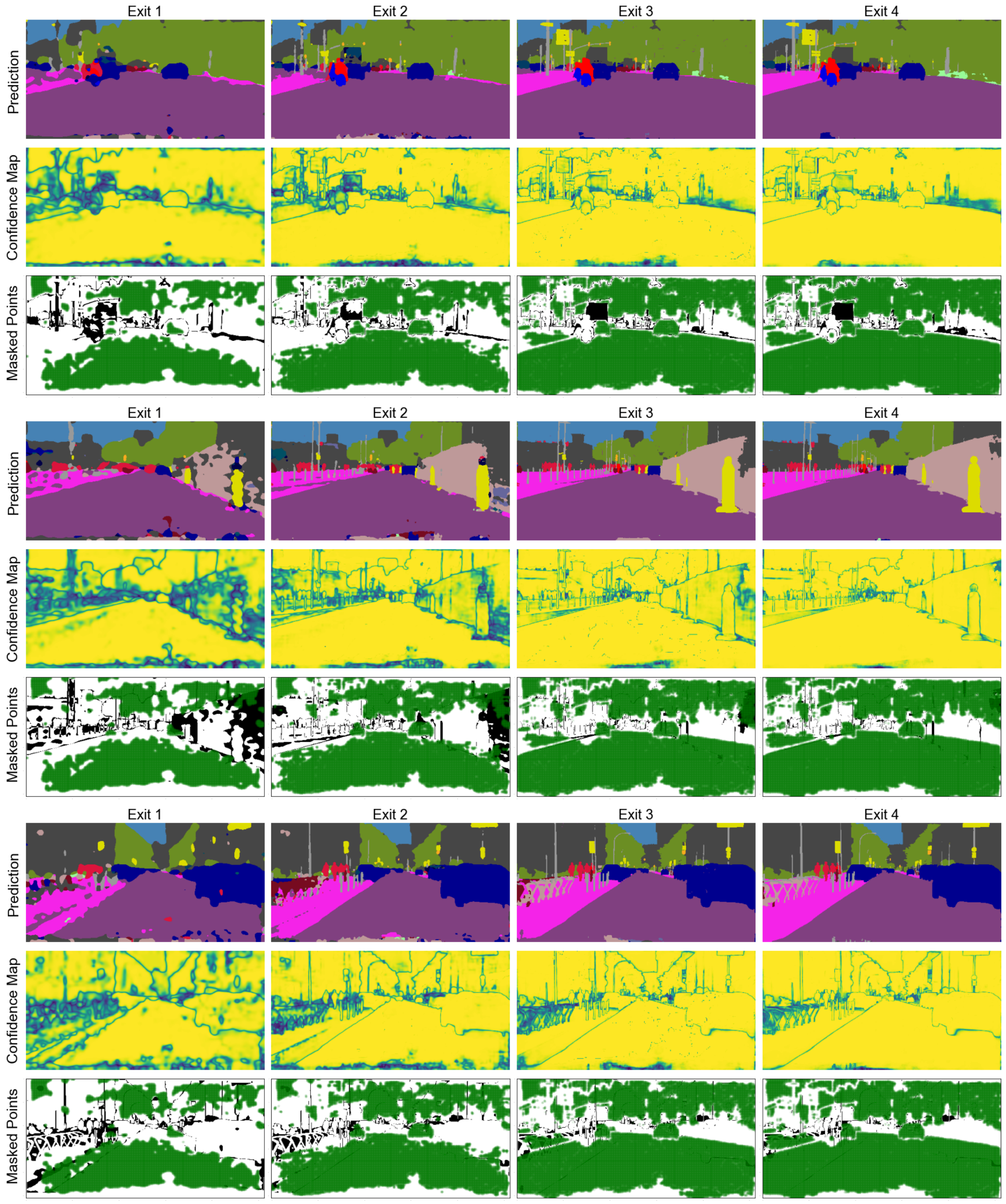


Figure 2.7: Additional visualizations of ADP-C results, part 1.

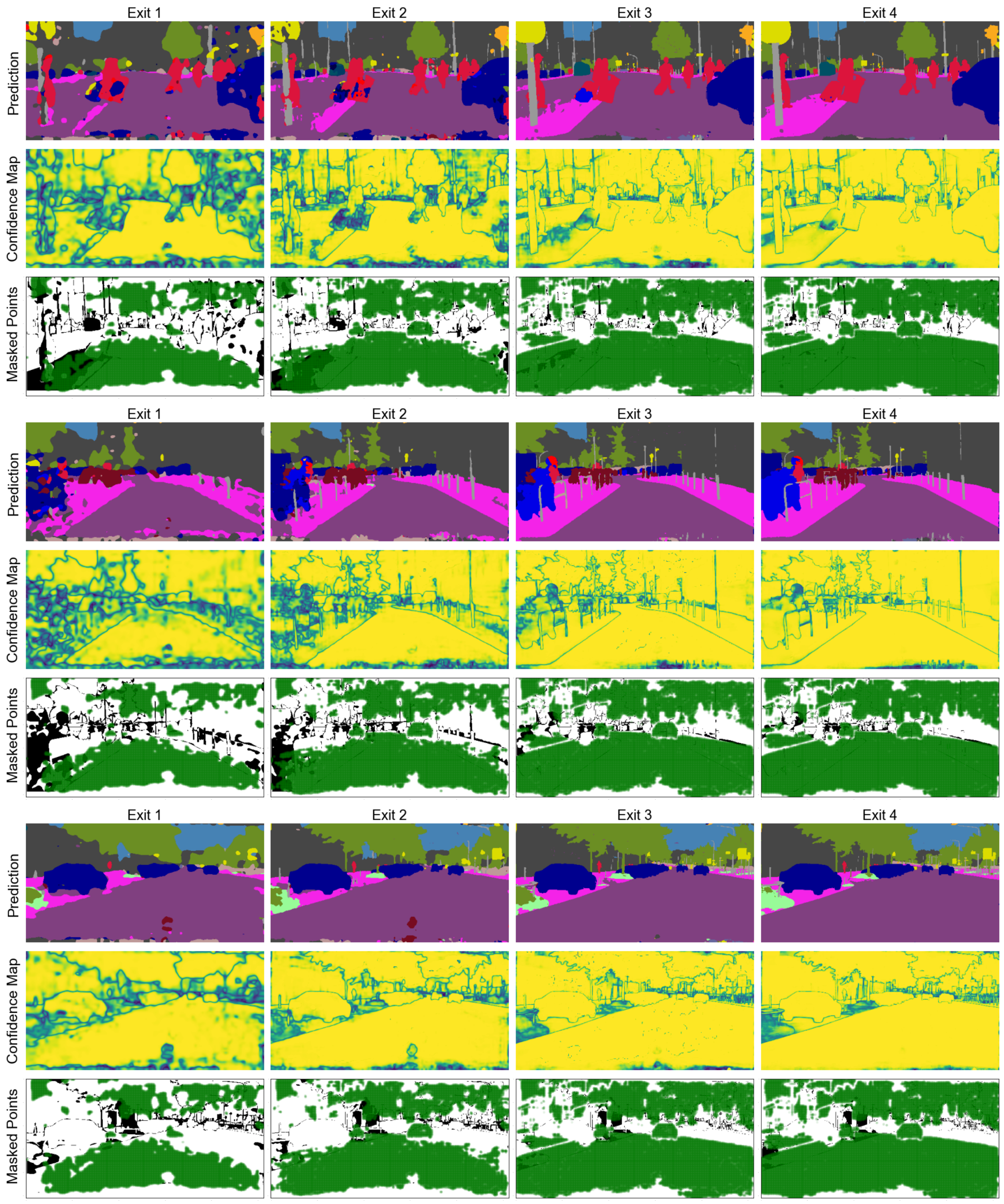


Figure 2.8: Additional visualizations of ADP-C results, part 2.

Chapter 3

Rethinking the Value of Network Pruning

3.1 Overview

Compared with the *adaptive* computation anytime inference method in the last chapter, *static* neural network pruning methods try to reduce the computation of a neural network in an input-agnostic manner. They are generally popular for scaling down models in practice due to their simplicity, effectiveness and sometimes better hardware compatibility. In this chapter, we try to understand what the true underlying mechanism is behind their success.

A typical pruning algorithm is a three-stage pipeline, i.e., training (a large model), pruning and fine-tuning. During pruning, according to a certain criterion, redundant weights are pruned and important weights are kept to best preserve the accuracy. In this work, we make several surprising observations which contradict common beliefs. For all state-of-the-art structured pruning algorithms we examined, fine-tuning a pruned model only gives comparable or worse performance than training that model with randomly initialized weights. For pruning algorithms which assume a predefined target network architecture, one can get rid of the full pipeline and directly train the target network from scratch. Our observations are consistent for multiple network architectures, datasets, and tasks, which imply that: 1) training a large, over-parameterized model is often not necessary to obtain an efficient final model, 2) learned "important" weights of the large model are typically not useful for the small pruned model, 3) the pruned architecture itself, rather than a set of inherited "important" weights, is more crucial to the efficiency in the final model, which suggests that in some cases pruning can be useful as an architecture search paradigm. Our results suggest the need for more careful baseline evaluations in future research on structured pruning methods. We also compare with the "Lottery Ticket Hypothesis" [44], and find that with optimal learning rate, the "winning ticket" initialization as used in [44] does not bring improvement over random initialization. Our code is available at `https://github.com/Eric-mingjie/rethinking-network-pruning`.

3.2 Introduction

Over-parameterization is a widely-recognized property of deep neural networks [35, 6], which leads to high computational cost and high memory footprint for inference. As a remedy, *network pruning* [104, 63, 62, 131, 107] has been identified as an effective technique to improve the efficiency of deep networks for applications with a limited computational budget. A typical procedure of network pruning consists of three stages: 1) train a large, over-parameterized model (sometimes there are pre-trained models available), 2) prune the trained large model according to a certain criterion, and 3) fine-tune the pruned model to regain the lost performance.

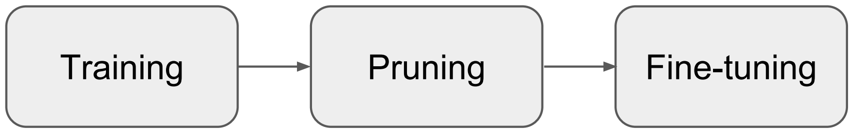


Figure 3.1: A typical three-stage network pruning pipeline.

Generally, there are two common beliefs behind this pruning procedure. First, it is believed that starting with training a large, over-parameterized network is important [125, 17], as it provides a high-performance model (due to stronger representation & optimization power) from which one can safely remove a set of redundant parameters without significantly hurting the accuracy. Therefore, this is usually believed, and reported to be superior to directly training a smaller network from scratch [107, 125, 71, 206] – a commonly used baseline approach. Second, both the pruned architecture *and* its associated weights are believed to be essential for obtaining the final efficient model [62]. Thus most existing pruning techniques choose to *fine-tune* a pruned model instead of training it from scratch. The preserved weights after pruning are usually considered to be critical, as how to accurately select the set of important weights is a very active research topic in the literature [131, 107, 125, 71, 118, 166].

In this work, we show that both of the beliefs mentioned above are not necessarily true for *structured* pruning methods, which prune at the levels of convolution channels or larger. Based on an extensive empirical evaluation of state-of-the-art pruning algorithms on multiple datasets with multiple network architectures, we make two surprising observations. First, for structured pruning methods with predefined target network architectures (Figure 3.2), directly training the small target model from random initialization can achieve the same, if not better, performance, as the model obtained from the three-stage pipeline. In this case, starting with a large model is not necessary and one could instead directly train the target model from scratch. Second, for structured pruning methods with auto-discovered target networks, training the pruned model from scratch can also achieve comparable or even better performance than fine-tuning. This observation shows that for these pruning methods, what matters more may be the obtained architecture, instead of the preserved weights, despite training the large model is needed to find that target architecture. Interestingly,

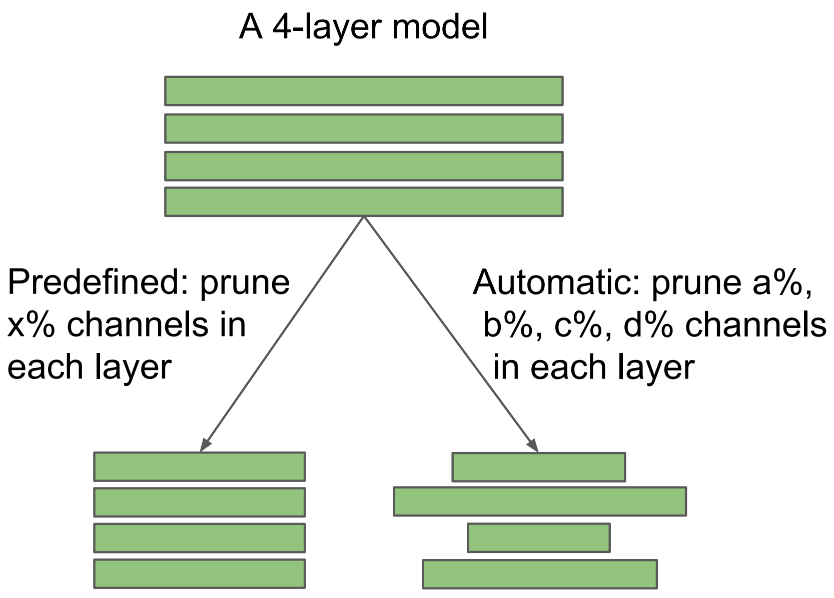

A 4-layer model
Predefined: prune x% channels in each layer
Automatic: prune a%, b%, c%, d% channels in each layer

Figure 3.2: Difference between predefined and automatically discovered target architectures, in channel pruning as an example. The pruning ratio x is user-specified, while a, b, c, d are determined by the pruning algorithm. Unstructured sparse pruning can also be viewed as automatic.

for a *unstructured* pruning method [62] that prunes individual parameters, we found that training from scratch can mostly achieve comparable accuracy with pruning and fine-tuning on smaller-scale datasets, but fails to do so on the large-scale ImageNet benchmark. Note that in some cases, if a pre-trained large model is already available, pruning and fine-tuning from it can save the training time required to obtain the efficient model. The contradiction between some of our results and those reported in the literature might be explained by less carefully chosen hyper-parameters, data augmentation schemes and unfair computation budget for evaluating baseline approaches.

Our results advocate a rethinking of existing structured network pruning algorithms. It seems that the over-parameterization during the first-stage training is not as beneficial as previously thought. Also, inheriting weights from a large model is not necessarily optimal, and might trap the pruned model into a bad local minimum, even if the weights are considered "important" by the pruning criterion. Instead, our results suggest that the value of automatic structured pruning algorithms sometimes lie in identifying efficient structures and performing implicit architecture search, rather than selecting "important" weights. For most structured pruning methods which prune channels/filters, this corresponds to searching the number of channels in each layer. In Section 3.6, we discuss this viewpoint through carefully designed experiments, and show the patterns in the pruned model could provide design guidelines for efficient architectures.

3.3 Background

The recent success of deep convolutional networks [106, 34, 47, 120, 64, 67] has been coupled with increased requirement of computation resources. In particular, the model size, memory footprint, the number of computation operations (FLOPs) and power usage are major aspects inhibiting the use of deep neural networks in some resource-constrained settings. Those large models can be infeasible to store, and run in real time on embedded systems. To address this issue, many methods have been proposed such as low-rank approximation of weights [35, 102], weight quantization [28, 146], knowledge distillation [77, 149] and network pruning [62, 107], among which network pruning has gained notable attention due to their competitive performance and compatibility.

One major branch of network pruning methods is individual weight pruning, and it dates back to Optimal Brain Damage [104] and Optimal Brain Surgeon [63], which prune weights based on Hessian of the loss function. More recently, [62] proposes to prune network weights with small magnitude, and this technique is further incorporated into the "Deep Compression" pipeline [60] to obtain highly compressed models. [163] proposes a data-free algorithm to remove redundant neurons iteratively. Network-Trim [1] prune a trained model layer-wisely, by solving a convex optimization problem. [130] uses Variatonal Dropout [135] to prune redundant weights. [122] learns sparse networks through L_0-norm regularization based on stochastic gating. However, one drawback of these *unstructured* pruning methods is that the resulting weight matrices are sparse, which cannot lead to compression and speedup without dedicated hardware/libraries [61].

In contrast, *structured* pruning methods prune at the level of channels or even layers. Since the original convolution structure is still preserved, no dedicated hardware/libraries are required to realize the benefits. Among structured pruning methods, channel pruning is the most popular, since it operates at the most fine-grained level while still fitting in conventional deep learning frameworks. Some heuristic methods include pruning channels based on their corresponding filter weight norm [107] and the average percentage of zeros in the output [81]. Group sparsity is also widely used to smooth the pruning process after training [190, 2, 101, 215]. [118] and [204] impose sparsity constraints on channel-wise scaling factors during training, whose magnitudes are then used for channel pruning. [87] uses a similar technique to prune coarser structures such as residual blocks. [71] and [125] minimizes the next layer's feature reconstruction error to determine which channels to keep. Similarly, [206] optimizes the reconstruction error of the final response layer and propagates a "importance score" for each channel. [131] uses Taylor expansion to approximate each channel's influence over the final loss and prune accordingly. [166] analyzes the intrinsic correlation within each layer and prunes redundant channels. [21] proposes a layer-wise compensate filter pruning algorithm to improve commonly-adopted heuristic pruning metrics. [70] proposes to allow pruned filters to recover during the training process. [109, 187] prune certain structures in the network based on the current input.

Our work is also related to some recent studies on the characteristics of pruning algorithms. [129] shows that random channel pruning [5] can perform on par with a variety of more sophisticated pruning criteria, demonstrating the plasticity of network models. In the context of unstructured pruning, The Lottery Ticket Hypothesis [44] conjectures that certain connections together with their randomly initialized weights, can enable a comparable accuracy with the original network when trained in isolation. We provide comparisons between [44] and this work in Section 3.7. [216] shows that training a small-dense model cannot achieve the same accuracy as a pruned large-sparse model with an identical memory footprint. In this work, we reveal a different and rather surprising characteristic of structured network pruning methods: fine-tuning the pruned model with inherited weights is not better than training it from scratch; the resulting pruned architectures are more likely to be what brings the benefit.

3.4 Methodology

In this section, we describe in detail our methodology for training a small target model from scratch.

3.4.1 Target Pruned Architectures

We first divide network pruning methods into two categories. In a pruning pipeline, the target pruned model's architecture can be determined by either a human (i.e., predefined) or the pruning algorithm (i.e., automatic) (see Figure 3.2).

When a human predefines the target architecture, a common criterion is the ratio of channels to prune in each layer. For example, we may want to prune 50% channels in each layer of VGG. In this case, no matter which specific channels are pruned, the pruned target architecture remains the same, because the pruning algorithm only *locally* prunes the least important 50% channels in each layer. In practice, the ratio in each layer is usually selected through empirical studies or heuristics. Examples of predefined structured pruning include [107], [125], [71] and [70]

When the target architecture is automatically determined by a pruning algorithm, it is usually based on a pruning criterion that *globally* compares the importance of structures (e.g., channels) across layers. Examples of automatic structured pruning include [118], [87], [131] and [166].

Unstructured pruning [62, 130, 122] also falls in the category of automatic methods, where the positions of pruned weights are determined by the training process and the pruning algorithm, and it is usually not possible to predefine the positions of zeros before training starts.

3.4.2 Datasets, Network Architectures and Pruning Methods

In the network pruning literature, CIFAR-10, CIFAR-100 [98], and ImageNet [34] datasets are the de-facto benchmarks, while VGG [161], ResNet [64] and DenseNet [85] are the common network architectures. We evaluate four predefined pruning methods, [107], [125], [71], [70], two automatic structured pruning methods, [118], [87], and one unstructured pruning method [62]. For the first six methods, we evaluate using the same (target model, dataset) pairs as presented in the original paper to keep our results comparable. For the last one [62], we use the aforementioned architectures instead, since the ones in the original paper are no longer state-of-the-art. On CIFAR datasets, we run each experiment with 5 random seeds, and report the mean and standard deviation of the accuracy.

3.4.3 Training Budget

One crucial question is how long we should train the small pruned model from scratch. Naively training for the same number of epochs as we train the large model might be unfair, since the small pruned model requires significantly less computation for one epoch. Alternatively, we could compute the floating point operations (FLOPs) for both the pruned and large models, and choose the number of training epochs for the pruned model that would lead to the same amount of computation as training the large model. Note that it is not clear how to train the models to "full convergence" given the stepwise decaying learning rate schedule commonly used in the CIFAR/ImageNet classification tasks.

In our experiments, we use **Scratch-E** to denote training the small pruned models for the same epochs, and **Scratch-B** to denote training for the same amount of computation budget (on ImageNet, if the pruned model saves more than $2\times$ FLOPs, we just double the number of epochs for training Scratch-B, which amounts to less computation budget than large model training). When extending the number of epochs in Scratch-B, we also extend the learning rate decay schedules proportionally. One may argue that we should instead train the small target model for fewer epochs since it may converge faster. However, in practice we found that increasing the training epochs within a reasonable range is rarely harmful. In our experiments we found in most times Scratch-E is enough while in other cases Scratch-B is needed for a comparable accuracy as fine-tuning. Note that our evaluations use the same computation as large model training without considering the computation in fine-tuning, since in our evaluated methods fine-tuning does not take too long; if anything this still favors the pruning and fine-tuning pipeline.

3.4.4 Implementation

In order to keep our setup as close to the original papers as possible, we use the following protocols: 1) ff a previous pruning method's training setup is publicly available, e.g. [118], [87] and [70], we adopt the original implementation; 2) otherwise, for simpler pruning methods, e.g., [107] and [62], we re-implement the three-stage pruning procedure and generally achieve similar results as in the original papers; 3) for the remaining two methods [125, 71], the pruned models are publicly available but without the training setup, thus we choose to re-train both large and small target models from scratch. Interestingly, the accuracy of our re-trained large model is higher than what is reported in the original papers. This could be due to the difference in the deep learning frameworks: we used Pytorch [136] while the original papers used Caffe [92]. In this case, to accommodate the effects of different frameworks and training setups, we report the relative accuracy drop from the unpruned large model.

We use standard training hyper-parameters and data-augmentation schemes, which are used both in standard image classification models [64, 85] and network pruning methods [107, 118, 87, 70]. The optimization method is SGD with Nesterov momentum, using a stepwise decay learning rate schedule.

For random weight initialization, we adopt the scheme proposed in [65]. For results of models fine-tuned from inherited weights, we either use the released models from original papers (case 3 above) or follow the common practice of fine-tuning the model using the lowest learning rate when training the large model [107, 71]. For CIFAR, training/fine-tuning takes 160/40 epochs. For ImageNet, training/fine-tuning takes 90/20 epochs. For reproducing the results and more detailed knowledge about the settings, see our code at `https://github.com/Eric-mingjie/rethinking-network-pruning`.

3.5 Experiments

In this section we present our experimental results comparing training pruned models from scratch and fine-tuning from inherited weights, for both predefined and automatic (Figure 3.2) structured pruning, as well as a magnitude-based unstructured pruning method [62]. We put the results and discussions on a pruning method (Soft Filter pruning [70]) in Section 3.8.1, and include an experiment on transfer learning from image classification to object detection in Section 3.8.2.

3.5.1 Predefined Structured Pruning

L_1-norm based Filter Pruning [107] is one of the earliest works on filter/channel pruning for convolutional networks. In each layer, a certain percentage of filters with smaller L_1-norm will be pruned. Table 3.1 shows our results. The Pruned Model column shows the list of predefined target models (see [107] for configuration details on each model). We observe that in each row, scratch-trained models achieve at least the same level of accuracy as fine-tuned models, with Scratch-B slightly higher than Scratch-E in most cases. On ImageNet, both Scratch-B models are better than the fine-tuned ones by a noticeable margin.

ThiNet [125] greedily prunes the channel that has the smallest effect on the next layer's activation values. As shown in Table 3.2, for VGG-16 and ResNet-50, both Scratch-E and Scratch-B can almost always achieve better performance than the fine-tuned model, often by a significant margin. The only exception is Scratch-E for VGG-Tiny, where the model is

Dataset	Model	Unpruned	Pruned Model	Fine-tuned	Scratch-E	Scratch-B
CIFAR-10	VGG-16	93.63 (±0.16)	VGG-16-A	93.41 (±0.12)	93.62 (±0.11)	**93.78** (±0.15)
	ResNet-56	93.14 (±0.12)	ResNet-56-A	92.97 (±0.17)	92.96 (±0.26)	**93.09** (±0.14)
			ResNet-56-B	92.67 (±0.14)	92.54 (±0.19)	**93.05** (±0.18)
	ResNet-110	93.14 (±0.24)	ResNet-110-A	93.14 (±0.16)	**93.25** (±0.29)	93.22 (±0.22)
			ResNet-110-B	92.69 (±0.09)	92.89 (±0.43)	**93.60** (±0.25)
ImageNet	ResNet-34	73.31	ResNet-34-A	72.56	72.77	**73.03**
			ResNet-34-B	72.29	72.55	**72.91**

Table 3.1: Results (accuracy % by default) for L_1-norm based filter pruning [107]. "Pruned Model" is the model pruned from the large model. Configurations of Model and Pruned Model are both from the original paper.

Dataset	Unpruned	Strategy	Pruned Model		
ImageNet	VGG-16		VGG-Conv	VGG-GAP	VGG-Tiny
	71.03	Fine-tuned	−1.23	−3.67	−11.61
	71.51	Scratch-E	−2.75	−4.66	−14.36
		Scratch-B	**+0.21**	**−2.85**	**−11.58**
	ResNet-50		ResNet50-30%	ResNet50-50%	ResNet50-70%
	75.15	Fine-tuned	−6.72	−4.13	−3.10
	76.13	Scratch-E	−5.21	−2.82	−1.71
		Scratch-B	**−4.56**	**−2.23**	**−1.01**

Table 3.2: Results for ThiNet [125]. Names such as "VGG-GAP" and "ResNet50-30%" are pruned models whose configurations are defined in [125]. To accommodate the effects of different frameworks between our implementation and the original paper's, we compare the relative accuracy drop from the unpruned large model. For example, for the pruned model VGG-Conv, −1.23 is relative to 71.03 on the left, which is the reported accuracy of the unpruned large model VGG-16 in the original paper; −2.75 is relative to 71.51 on the left, which is VGG-16's accuracy in our implementation.

pruned very aggressively from VGG-16 (FLOPs reduced by 15×), and as a result, drastically reducing the training budget for Scratch-E. The training budget of Scratch-B for this model is also 7 times smaller than the original large model, yet it can achieve the same level of accuracy as the fine-tuned model.

Regression based Feature Reconstruction [71] prunes channels by minimizing the feature map reconstruction error of the next layer. In contrast to ThiNet [125], this optimization problem is solved by LASSO regression. Results are shown in Table 3.3. Again, in terms of relative accuracy drop from the large models, scratch-trained models are better than the fine-tuned models.

Dataset	Unpruned	Strategy	Pruned Model
ImageNet	VGG-16		VGG-16-5×
	71.03	Fine-tuned	−2.67
	71.51	Scratch-E	−3.46
		Scratch-B	**−0.51**
	ResNet-50		ResNet-50-2×
	75.51	Fine-tuned	−3.25
	76.13	Scratch-E	−1.55
		Scratch-B	**−1.07**

Table 3.3: Results for Regression based Feature Reconstruction [71]. Pruned models such as "VGG-16-5×" are defined in [71]. Similar to Table 3.2, we compare relative accuracy drop from unpruned large models.

3.5.2 Automatic Structured Pruning

Network Slimming [118] imposes L_1-sparsity on channel-wise scaling factors from Batch Normalization layers [89] during training, and prunes channels with lower scaling factors afterwards. Since the channel scaling factors are compared across layers, this method produces automatically discovered target architectures. As shown in Table 3.4, for all networks, the small models trained from scratch can reach the same accuracy as the fine-tuned models. More specifically, we found that Scratch-B consistently outperforms (8 out of 10 experiments) the fine-tuned models, while Scratch-E is slightly worse but still mostly within the standard deviation.

Sparse Structure Selection [87] also uses sparsified scaling factors to prune structures, and can be seen as a generalization of Network Slimming. Other than channels, pruning can be on residual blocks in ResNet or groups in ResNeXt [201]. We examine residual blocks pruning, where ResNet-50 are pruned to be ResNet-41, ResNet-32 and ResNet-26. Table 3.5 shows our results. On average Scratch-E outperforms pruned models, and for all models Scratch-B is better than both.

Dataset	Model	Unpruned	Prune Ratio	Fine-tuned	Scratch-E	Scratch-B
CIFAR-10	VGG-19	93.53 (±0.16)	70%	93.60 (±0.16)	93.30 (±0.11)	**93.81** (±0.14)
	PreResNet-164	95.04 (±0.16)	40%	94.77 (±0.12)	94.70 (±0.11)	**94.90** (±0.04)
			60%	94.23 (±0.21)	94.58 (±0.18)	**94.71** (±0.21)
	DenseNet-40	94.10 (±0.12)	40%	94.00 (±0.20)	93.68 (±0.18)	**94.06** (±0.12)
			60%	**93.87** (±0.13)	93.58 (±0.21)	93.85 (±0.25)
CIFAR-100	VGG-19	72.63 (±0.21)	50%	72.32 (±0.28)	71.94 (±0.17)	**73.08** (±0.22)
	PreResNet-164	76.80 (±0.19)	40%	76.22 (±0.20)	76.36 (±0.32)	**76.68** (±0.35)
			60%	74.17 (±0.33)	75.05 (± 0.08)	**75.73** (±0.29)
	DenseNet-40	73.82 (±0.34)	40%	**73.35** (±0.17)	73.24 (±0.29)	73.19 (±0.26)
			60%	72.46 (±0.22)	72.62 (±0.36)	**72.91** (±0.34)
ImageNet	VGG-11	70.84	50%	68.62	70.00	**71.18**

Table 3.4: Results for Network Slimming [118]. "Prune ratio" stands for total percentage of channels that are pruned in the whole network. The same ratios for each model are used as the original paper.

Dataset	Model	Unpruned	Pruned Model	Pruned	Scratch-E	Scratch-B
ImageNet	ResNet-50	76.12	ResNet-41	75.44	75.61	**76.17**
			ResNet-32	74.18	73.77	**74.67**
			ResNet-26	71.82	72.55	**73.41**

Table 3.5: Results for residual block pruning using Sparse Structure Selection [87]. In the original paper no fine-tuning is required so there is a "Pruned" column instead of "Fine-tuned" as before.

3.5.3 Unstructured Magnitude-based Pruning

Unstructured magnitude-based weight pruning [62] can also be treated as automatically discovering architectures, since the positions of exact zeros cannot be determined before training, but we highlight its differences with structured pruning using another subsection. Because all the network architectures we evaluated are fully-convolutional (except for the last fully-connected layer), for simplicity, we only prune weights in convolution layers here. Before training the pruned sparse model from scratch, we re-scale the standard deviation of the Gaussian distribution for weight initialization, based on how many non-zero weights remain in this layer. This is to keep a constant scale of backward gradient signal as in [65], which however in our observations does not bring gains compared with unscaled counterparts.

As shown in Table 3.6, on the smaller-scale CIFAR datasets, when the pruned ratio is small ($\leq 80\%$), Scratch-E sometimes falls short of the fine-tuned results, but Scratch-B is able to perform at least on par with the latter. However, we observe that in some cases, when

Dataset	Model	Unpruned	Prune Ratio	Fine-tuned	Scratch-E	Scratch-B
CIFAR-10	VGG-19	93.50 (±0.11)	30%	93.51 (±0.05)	**93.71** (±0.09)	93.31 (±0.26)
			80%	93.52 (±0.10)	**93.71** (±0.08)	93.64 (±0.09)
			95%	93.34 (±0.13)	93.21 (±0.17)	**93.63** (±0.18)
	PreResNet-110	95.04 (±0.15)	30%	95.06 (±0.05)	94.84 (±0.07)	**95.11** (±0.09)
			80%	**94.55** (±0.11)	93.76 (±0.10)	94.52 (±0.13)
			95%	**92.35** (±0.20)	91.23 (±0.11)	91.55 (±0.34)
	DenseNet-BC-100	95.24 (±0.17)	30%	95.21 (±0.17)	95.22 (±0.18)	**95.23** (±0.14)
			80%	95.04 (±0.15)	94.42 (±0.12)	**95.12** (±0.04)
			95%	**94.19** (±0.15)	92.91 (±0.22)	93.44 (±0.19)
CIFAR-100	VGG-19	71.70 (±0.31)	30%	71.96 (±0.36)	72.81 (±0.31)	**73.30** (±0.25)
			50%	71.85 (±0.30)	73.12 (±0.36)	**73.77** (±0.23)
			95%	70.22 (±0.38)	70.88 (±0.35)	**72.08** (±0.15)
	PreResNet-110	76.96 (±0.34)	30%	76.88 (±0.31)	76.36 (±0.26)	**76.96** (±0.31)
			50%	**76.60** (±0.36)	75.45 (±0.23)	76.42 (±0.39)
			95%	68.55 (±0.51)	68.13 (±0.64)	**68.99** (±0.32)
	DenseNet-BC-100	77.59 (±0.19)	30%	77.23 (±0.05)	77.58 (±0.25)	**77.97** (±0.31)
			50%	77.41 (±0.14)	77.65 (±0.09)	**77.80** (±0.23)
			95%	**73.67** (±0.03)	71.47 (±0.46)	72.57 (±0.37)
ImageNet	VGG-16	73.37	30%	73.68	72.75	**74.02**
			60%	**73.63**	71.50	73.42
	ResNet-50	76.15	30%	**76.06**	74.77	75.70
			60%	**76.09**	73.69	74.91

Table 3.6: Results for unstructured pruning [62]. "Prune Ratio" denotes the percentage of parameters pruned in the set of all convolutional weights.

the prune ratio is large (95%), fine-tuning can outperform training from scratch. On the large-scale ImageNet dataset, we note that Scratch-B results are mostly worse than fine-tuned results by a noticeable margin, despite at a decent accuracy level. This could be due to the increased difficulty of directly training on the highly sparse networks (CIFAR), or the scale/complexity of the dataset itself (ImageNet). Another possible reason is that compared with structured pruning, unstructured pruning significantly changes the weight distribution (more details in Section 3.8.6). The difference in scratch-training behaviors also suggests an important difference between structured and unstructured pruning.

3.6 Network Pruning as Architecture search

While we have shown that, for structured pruning, the inherited weights in the pruned architecture are not better than random, the pruned architecture itself turns out to be what brings the efficiency benefits. In this section, we assess the value of architecture search for automatic network pruning algorithms (Figure 3.2) by comparing pruning-obtained models and uniformly pruned models. Note that the connection between network pruning and architecture learning has also been made in prior works [62, 118, 53, 83], but to our knowledge, we are the first to isolate the effect of inheriting weights and solely compare pruning-obtained architectures with uniformly pruned ones, by training both of them from scratch.

3.6.1 Parameter Efficiency of Pruned Architectures

In Figure 3.3(left), we compare the parameter efficiency of architectures obtained by an automatic channel pruning method (Network Slimming [118]), with a naive predefined pruning strategy that uniformly prunes the same percentage of channels in each layer. All architectures are trained from random initialization for the same number of epochs. We see that the architectures obtained by Network Slimming are more parameter efficient, as they could achieve the same level of accuracy using 5× fewer parameters than uniformly pruning architectures. For unstructured magnitude-based pruning [62], we conducted a similar experiment shown in Figure 3.3 (right). Here we uniformly sparsify all individual weights at a fixed probability, and the architectures obtained this way are much less efficient than the pruned architectures.

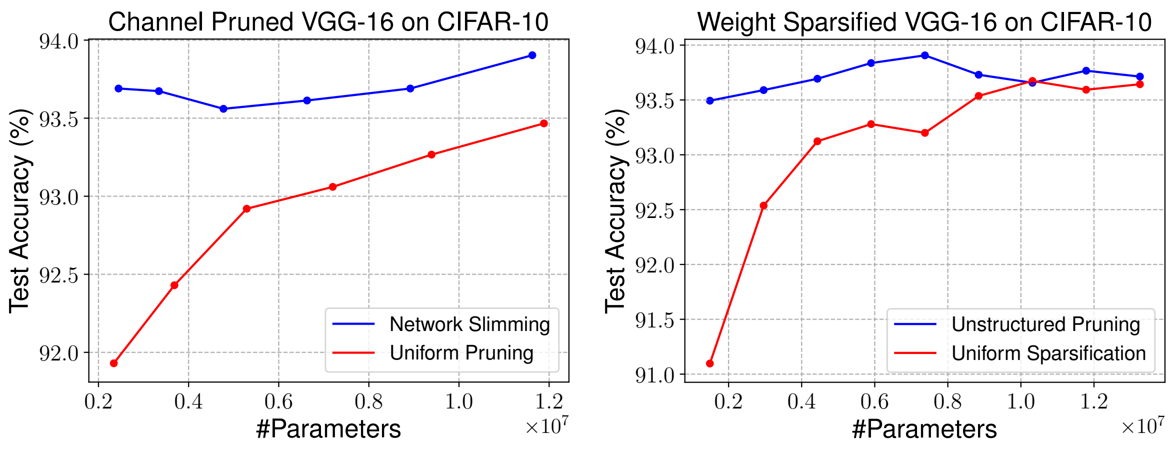


Figure 3.3: Pruned architectures obtained by different approaches, all *trained from scratch*, averaged over 5 runs. Architectures obtained by automatic pruning methods have better parameter efficiency than uniformly pruning channels or sparsifying weights in the whole network.

We also found the channel/weight pruned architectures exhibit very consistent patterns (see Table 3.7 and Figure 3.4). This suggests the original large models may be redundantly designed for the task and the pruning algorithm can help us improve the efficiency. This

Layer	Width	Width*	Layer	Width	Width*
1	64	39.0±3.7	8	512	217.3±6.6
2	64	64.0±0.0	9	512	120.0±4.4
3	128	127.8±0.4	10	512	63.0±1.9
4	128	128.0±0.0	11	512	47.8±2.9
5	256	255.0±1.0	12	512	62.0±3.4
6	256	250.5±0.5	13	512	88.8±3.1
7	256	226.0±2.5	Total	4224	1689.2

Table 3.7: Network architectures obtained by pruning 60% channels on VGG-16 (in total 13 conv-layers) using Network Slimming. Width and Width* are number of channels in the original and pruned architectures, averaged over 5 runs.

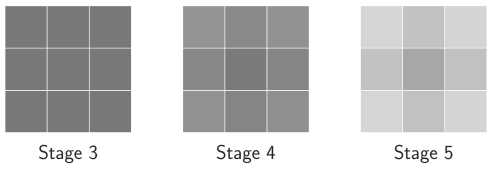


Figure 3.4: The average sparsity pattern of all 3×3 convolutional kernels in certain layer stages in a unstructured pruned VGG-16. Darker color means higher probability of weight being kept.

also confirms the value of automatic pruning methods for searching efficient models on the architectures evaluated.

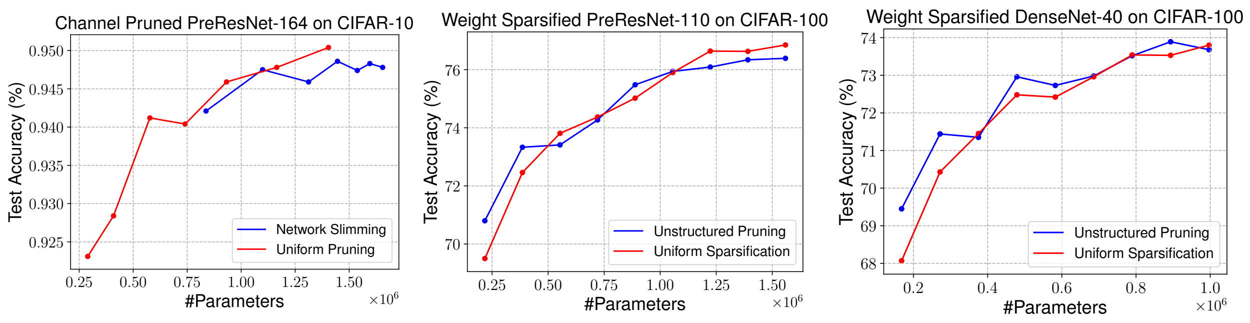


Figure 3.5: Pruned architectures obtained by different approaches, *all trained from scratch*, averaged over 5 runs. *Left:* Results for PreResNet-164 pruned on CIFAR-10 by Network Slimming [118]. *Middle* and *Right*: Results for PreResNet-110 and DenseNet-40 pruned on CIFAR-100 by unstructured pruning [62].

3.6.2 More Analysis

However, there also exist cases where the architectures obtained by pruning are not better than uniformly pruned ones. We present such results in Figure 3.5, where the architectures obtained by pruning (blue) are not significantly more efficient than uniform pruned architectures (red). This phenomenon happens more likely on modern architectures like ResNets and DenseNets. When we investigate the sparsity patterns of those pruned architectures (shown in Table 3.18, 3.19 and 3.20 in Section 3.8.7), we find that they exhibit near-uniform sparsity patterns across stages, which might be the reason why it can only perform on par with uniform pruning. In contrast, for VGG, the pruned sparsity patterns can always beat the uniform ones as shown in Figure 3.3 and 3.6. We also show the sparsity patterns of VGG pruned by Network Slimming [118] in Table 3.21 of Section 3.8.7, and they are rather far from uniform. Compared to ResNet and DenseNet, we can see that VGG's redundancy is rather imbalanced across layer stages. Network pruning techniques may help us identify redundancy better in such cases.

3.6.3 Generalizable Design Principles from Pruned Architectures

Given that the automatically discovered architectures tend to be parameter efficient on the VGG networks, one may wonder: can we derive generalizable principles from them on how to design a better architecture? We conduct several experiments to answer this question.

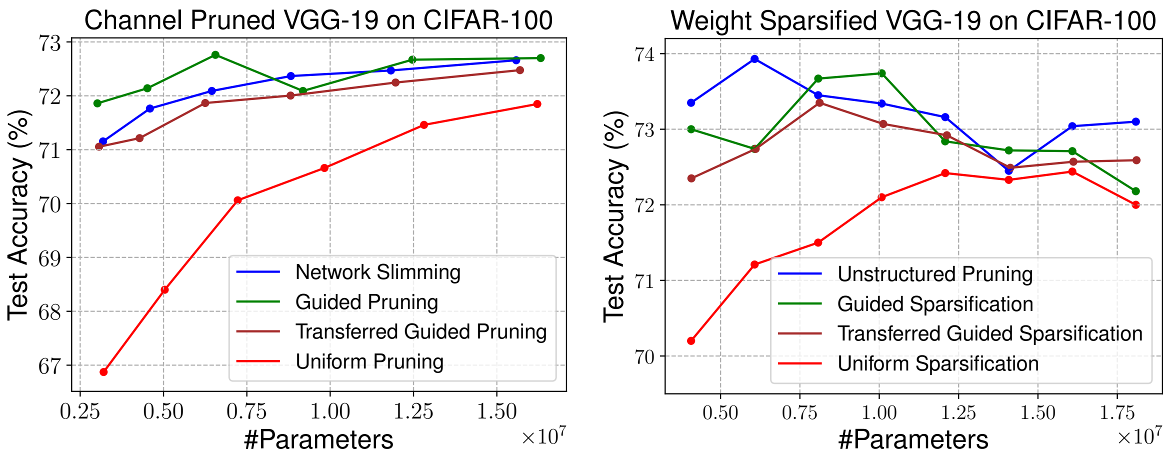


Figure 3.6: Pruned architectures obtained by different approaches, *all trained from scratch*, averaged over 5 runs. "Guided Pruning/Sparsification" means using the average sparsity patterns in each layer stage to design the network; "Transferred Guided Pruning/Sparsification" means using the sparsity patterns obtained by a pruned VGG-16 on CIFAR-10, to design the network for VGG-19 on CIFAR-100. Following the design guidelines provided by the pruned architectures, we achieve better parameter efficiency, even when the guidelines are transferred from another dataset and model.

For Network Slimming, we use the average number of channels in each layer stage (layers with the same feature map size) from pruned architectures to construct a new set of architectures, and we call this approach "Guided Pruning"; for magnitude-based pruning, we analyze the sparsity patterns (Figure 3.4) in the pruned architectures, and apply them to construct a new set of sparse models, which we call "Guided Sparsification". The results are shown in Figure 3.6. It can be seen that for both Network Slimming (Figure 3.6 left) and unstructured pruning (Figure 3.6 right), guided design of architectures (green) can perform on par with pruned architectures (blue).

Interestingly, these guided design patterns can sometimes be transferred to a different VGG-variant and/or dataset. In Figure 3.6, we distill the patterns of pruned architectures from VGG-16 on CIFAR-10 and apply them to design efficient VGG-19 on CIFAR-100. These sets of architectures are denoted as "Transferred Guided Pruning/Sparsification". We can observe that they (brown) may sometimes be slightly worse than architectures directly pruned (blue), but are significantly better than uniform pruning/sparsification (red). In these cases, one does not need to train a large model to obtain an efficient model as well, as transferred design patterns can help us achieve efficiency directly.

3.6.4 Discussions with Conventional Architecture Search Methods

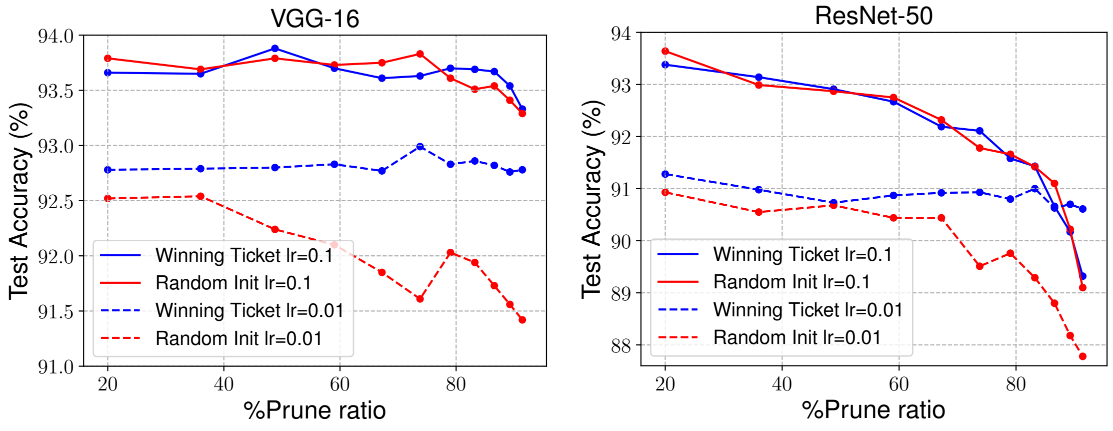
Popular techniques for network architecture search include reinforcement learning [218, 9] and evolutionary algorithms [200, 115]. In each iteration, a randomly initialized network is trained and evaluated to guide the search, and the search process usually requires thousands of iterations to find the goal architecture. In contrast, using network pruning as architecture search only requires a one-pass training, however, the search space is restricted to the set of all "sub-networks" inside a large network, whereas traditional methods can search for more variations, e.g., activation functions or different layer orders.

Recently, [53] uses a similar pruning technique to Network Slimming [118] to automate the design of network architectures; [72] prune channels using reinforcement learning and automatically compresses the architecture. On the other hand, in the network architecture search literature, sharing/inheriting trained parameters [137, 114] during searching has become a popular approach for reducing the training budgets, but once the target architecture is found, it is still trained from scratch to maximize the accuracy.

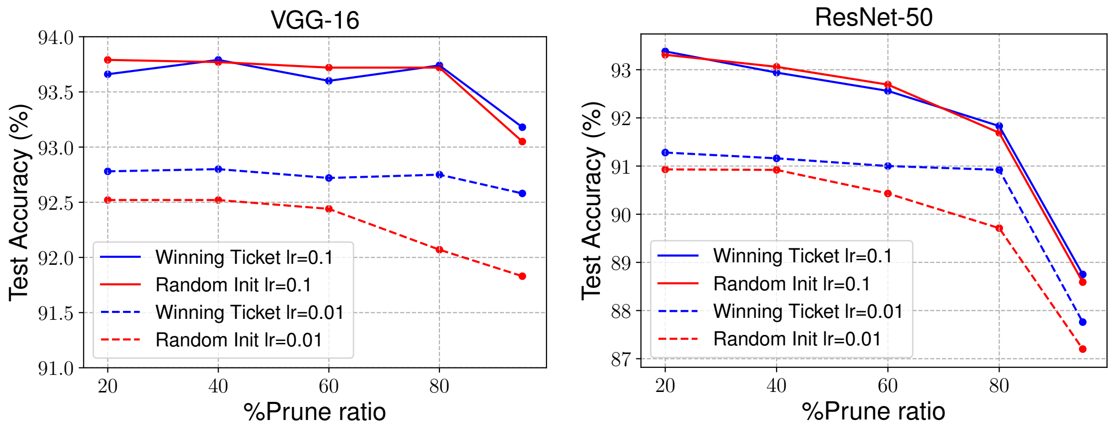
3.7 Experimenting with the Lottery Ticket Hypothesis

The Lottery Ticket Hypothesis [44] conjectures that inside the large network, a sub-network together with their initialization makes the training particularly effective, and together they are termed the "winning ticket". In this hypothesis, the original initialization of the sub-network (before large model training) is needed for it to achieve competitive performance when trained in isolation. Their experiments show that training the sub-network with randomly re-initialized weights performs worse than training it with the original initialization inside the large network. In contrast, our work does not require the reuse of the original initialization of the pruned model, and shows that random initialization is enough for the pruned model to achieve competitive performance.

The conclusions seem to be contradictory, but there are several important differences in the evaluation settings: a) Our main conclusion is drawn on *structured* pruning methods, despite for small-scale problems (CIFAR) it also holds on unstructured pruning; [44] only evaluates on unstructured pruning. b) Our evaluated network architectures are all relatively large modern models used in the original pruning methods, while most of the experiments in [44] use small shallow networks (< 6 layers). c) We use momentum SGD with a large initial learning rate (0.1), which is widely used in prior image classification [64, 85] and pruning works [107, 118, 71, 125, 70, 87] to achieve high accuracy, and is the de facto default optimization setting on CIFAR and ImageNet; while [44] mostly uses Adam with much lower learning rates. d) Our experiments include the large-scale ImageNet dataset, while [44] only considers MNIST and CIFAR.



(a) Iterative Pruning



(b) One-shot Pruning

Figure 3.7: Comparisons with the Lottery Ticket Hypothesis for iterative/one-shot unstructured pruning [62] with two initial learning rates 0.1 and 0.01, on CIFAR-10 dataset. Each point is averaged over 5 runs. Using the winning ticket as initialization only brings improvement when the learning rate is small (0.01), however such small learning rate leads to lower accuracy than the widely used large learning rate (0.1).

In this section, we show that the difference in learning rate is what causes the seemingly contradicting behaviors between our work and [44], in the case of unstructured pruning on CIFAR. For structured pruning, when using both large and small learning rates, the winning ticket does not outperform random initialization.

Dataset	Model	Unpruned	Pruned Model	Winning Ticket	Random Init
CIFAR-10	VGG-16	93.63 (±0.16)	VGG-16-A	**93.62** (±0.09)	93.60 (±0.15)
	ResNet-56	93.14 (±0.12)	ResNet-56-A	92.72 (±0.10)	**92.75** (±0.26)
			ResNet-56-B	92.78 (±0.23)	**92.90** (±0.27)
	ResNet-110	93.14 (±0.24)	ResNet-110-A	**93.21** (±0.09)	**93.21** (±0.21)
			ResNet-110-B	93.15 (±0.12)	**93.37** (±0.29)

(a) Initial learning rate 0.1

Dataset	Model	Unpruned	Pruned Model	Winning Ticket	Random Init
CIFAR-10	VGG-16	92.64 (±0.05)	VGG-16-A	92.65 (±0.18)	**92.67** (±0.22)
	ResNet-56	89.81 (±0.27)	ResNet-56-A	**90.00** (±0.15)	89.87 (±0.25)
			ResNet-56-B	89.75 (±0.35)	**89.81** (±0.24)
	ResNet-110	89.43 (±0.39)	ResNet-110-A	89.48 (±0.35)	**89.49** (±0.10)
			ResNet-110-B	**89.36** (±0.30)	89.35 (±0.16)

(b) Initial learning rate 0.01

Table 3.8: Comparisons with the Lottery Ticket Hypothesis on a structured pruning method (L_1-norm based filter pruning [107]) with two initial learning rates: 0.1 and 0.01. In both cases, using winning tickets does not bring improvement on accuracy.

We test the Lottery Ticket Hypothesis by comparing the models trained with original initialization ("winning ticket") and that trained from randomly re-initialized weights. We experiment with two choices of initial learning rate (0.1 and 0.01) with a stepwise decay schedule, using momentum SGD. 0.1 is used in our previous experiments and most prior works on CIFAR and ImageNet. Following [44], we investigate both iterative pruning (prune 20% in each iteration) and one-shot pruning for unstructured pruning. We show our results for unstructured pruning [62] in Figure 3.7 and Table 3.9, and L_1-norm based filter pruning [107] in Table 3.8.

From Figure 3.7 and Table 3.9, we see that for unstructured pruning, using the original initialization as in [44] only provides an advantage over random initialization with a small initial learning rate of 0.01. For structured pruning as [107], it can be seen from Table 3.8 that using the original initialization is only on par with random initialization for both large and small initial learning rates. In both cases, we can see that the small learning rate gives lower accuracy than the widely-used large learning rate. To summarize, in our evaluated settings,

Dataset	Model	Unpruned	Prune Ratio	Winning Ticket	Random Init
CIFAR-10	VGG-16	93.76 (±0.20)	20%	93.66 (±0.20)	**93.79** (±0.11)
			40%	**93.79** (±0.12)	93.77 (±0.10)
			60%	93.60 (±0.13)	**93.72** (±0.11)
			80%	**93.74** (±0.15)	93.72 (±0.16)
			95%	**93.18** (±0.12)	93.05 (±0.21)
	ResNet-50	93.48 (±0.20)	20%	**93.38** (±0.18)	93.31 (±0.24)
			40%	92.94 (±0.12)	**93.06** (±0.22)
			60%	92.56 (±0.20)	**92.69** (±0.11)
			80%	**91.83** (±0.20)	91.69 (±0.21)
			95%	**88.75** (±0.18)	88.59 (±0.09)

(a) One-shot pruning with initial learning rate 0.1

Dataset	Model	Unpruned	Prune Ratio	Winning Ticket	Random Init
CIFAR-10	VGG-16	92.69 (±0.12)	20%	**92.78** (±0.11)	92.52 (±0.15)
			40%	**92.80** (±0.18)	92.52 (±0.15)
			60%	**92.72** (±0.16)	92.44 (±0.19)
			80%	**92.75** (±0.07)	92.07 (±0.25)
			95%	**92.58** (±0.25)	91.83 (±0.11)
	ResNet-50	91.06 (±0.28)	20%	**91.28** (±0.15)	90.93 (±0.34)
			40%	**91.16** (±0.07)	90.92 (±0.10)
			60%	**91.00** (±0.15)	90.43 (±0.16)
			80%	**90.92** (±0.08)	89.71 (±0.18)
			95%	**87.76** (±0.19)	87.20 (±0.17)

(b) One-shot pruning with initial learning rate 0.01

Table 3.9: Comparisons with the Lottery Ticket Hypothesis for one-shot unstructured pruning [62] with two initial learning rates: 0.1 and 0.01. The same results are visualized in Figure 3.7b. Using the winning ticket as initialization only brings improvement when the learning rate is small (0.01), however such small learning rate leads to lower accuracy than the widely used large learning rate (0.1).

the winning ticket only brings improvement in the case of unstructured pruning, with a small initial learning rate, but this small learning rate yields inferior accuracy compared with the widely-used large learning rate. Note that [44] also report in their Section 5, that the winning ticket cannot be found on ResNet-18/VGG using the large learning rate. The reason why the original initialization is helpful when the learning rate is small, might be the weights of the final trained model are not too far from the original initialization due to the small parameter updating stepsize.

3.8 Additional Studies

3.8.1 Results on Soft Filter Pruning

Soft Filter Pruning (SFP) [70] prunes filters every epoch during training but also keeps updating the pruned filters, i.e., the pruned weights have the chance to be recovered. In the original paper, SFP can either run upon a randomly initialized model or a pre-trained model. It falls into the category of predefined methods (Figure 3.2). Table 3.10 shows our results without using pre-trained models and Table 3.11 shows the results with a pre-trained model. We use authors' official code [70] for obtaining the results. It can be seen that Scratch-E outperforms pruned models most of the time and Scratch-B outperforms pruned models in nearly all cases. Therefore, our observation also holds on this method.

Dataset	Model	Unpruned	Prune Ratio	Pruned	Scratch-E	Scratch-B
CIFAR-10	ResNet-20	92.41 (±0.12)	10%	92.00 (±0.32)	**92.22** (±0.15)	92.13 (±0.10)
			20%	91.50 (±0.30)	91.62 (±0.12)	**91.67** (±0.15)
			30%	90.78 (±0.15)	90.93 (±0.10)	**91.07** (±0.23)
	ResNet-32	93.22 (±0.16)	10%	93.28 (±0.05)	**93.42** (±0.40)	93.08 (±0.13)
			20%	92.50 (±0.17)	92.68 (±0.20)	**92.96** (±0.11)
			30%	92.02 (±0.11)	92.37 (±0.12)	**92.56** (±0.06)
	ResNet-56	93.80 (±0.12)	10%	93.77 (±0.07)	93.42 (±0.40)	**93.98** (±0.21)
			20%	93.14 (±0.42)	93.44 (±0.05)	**93.71** (±0.14)
			30%	93.01 (±0.09)	93.19 (±0.20)	**93.57** (±0.12)
			40%	92.59 (±0.14)	92.80 (±0.25)	**93.07** (±0.25)
	ResNet-110	93.77 (±0.23)	10%	93.60 (±0.50)	**94.21** (±0.39)	94.13 (±0.37)
			20%	93.63 (±0.44)	93.52 (±0.18)	**94.29** (±0.18)
			30%	93.26 (±0.37)	93.70 (±0.16)	**93.92** (±0.13)
ImageNet	ResNet-34	73.92	30%	71.83	71.67	**72.97**
	ResNet-50	76.15	30%	74.61	74.98	**75.56**

Table 3.10: Results for Soft Filter Pruning [70] without pretrained models.

Dataset	Model	Unpruned	Prune Ratio	Pruned	Scratch-E	Scratch-B
CIFAR-10	ResNet-56	93.80 (±0.12)	30%	93.51 (±0.26)	**94.45** (±0.30)	93.77 (±0.25)
			40%	93.10 (±0.34)	**93.84** (±0.16)	93.41 (±0.08)
	ResNet-110	93.77 (±0.23)	30%	93.46 (±0.19)	93.89 (±0.17)	**94.37** (±0.24)

Table 3.11: Results for Soft Filter Pruning [70] using pre-trained models.

3.8.2 Transfer Learning to Object Detection

We have shown that for structured pruning the small pruned model can be trained from scratch to match the accuracy of the fine-tuned model in classification tasks. To see whether this phenomenon would also hold for transfer learning to other vision tasks, we evaluate the L_1-norm based filter pruning method [107] on the PASCAL VOC object detection task, using Faster-RCNN [148].

Object detection frameworks usually require transferring model weights pre-trained on ImageNet classification, and one can perform pruning either before or after the weight transfer. More specifically, the former could be described as "train on classification, prune on classification, fine-tune on classification, transfer to detection", while the latter is "train on classification, transfer to detection, prune on detection, fine-tune on detection". We call these two approaches Prune-C (classification) and Prune-D (detection) respectively, and report the results in Table 3.12. With a slight abuse of notation, here Scratch-E/B denotes "train the small model on classification, transfer to detection", and is different from the setup of detection without ImageNet pre-training as in [157].

Dataset	Model	Unpruned	Pruned Model	Prune-C	Prune-D	Scratch-E	Scratch-B
PASCAL VOC 07	ResNet-34	71.69	ResNet34-A	71.47	70.99	71.64	**71.90**
			ResNet34-B	70.84	69.62	**71.68**	71.26

Table 3.12: Results (mAP) for pruning on detection task. The pruned models are chosen from [107]. Prune-C refers to pruning on classifcation pre-trained weights, Prune-D refers to pruning after the weights are transferred to detection task. Scratch-E/B means pre-training the pruned model from scratch on classification and transfer to detection.

For this experiment, we adopt the code and default hyper-parameters from [48], and use PASCAL VOC 07 trainval/test set as our training/test set. For backbone networks, we evaluate ResNet-34-A and ResNet-34-B from the L_1-norm based filter pruning [107], which are pruned from ResNet-34. Table 3.12 shows our result, and we can see that the model trained from scratch can surpass the performance of fine-tuned models under the transfer setting.

Another interesting observation from Table 3.12 is that Prune-C is able to outperform Prune-D, which is surprising since if our goal task is detection, directly pruning away weights that are considered unimportant for detection should presumably be better than pruning on the pre-trained classification models. We hypothesize that this might be because pruning early in the classification stage makes the final model less prone to being trapped in a bad local minimum caused by inheriting weights from the large model. This is in line with our observation that Scratch-E/B, which trains the small models from scratch starting even earlier at the classification stage, can achieve further performance improvement.

3.8.3 Aggressively Pruned Models

It would be interesting to see whether our observation still holds if the model is very aggressively pruned, since they might not have enough capacity to be trained from scratch and achieve decent accuracy. Here we provide results using Network Slimming [118] and L_1-norm based filter pruning [107]. From Table 3.13, Table 3.14 and Table 3.15, it can be seen that when the prune ratio is large, training from scratch is better than fine-tuned models by an even larger margin, and in this case fine-tuned models are significantly worse than the unpruned models. Note that in Table 3.2, the VGG-Tiny model we evaluated for ThiNet [125] is also a very aggressively pruned model (FLOPs reduced by 15× and parameters reduced by 100×).

Dataset	Model	Unpruned	Prune Ratio	Fine-tuned	Scratch-E	Scratch-B
CIFAR-10	PreResNet-56	93.69 (±0.07)	80%	74.66 (±0.96)	88.25 (±0.38)	**88.65** (±0.32)
	PreResNet-164	95.04 (±0.16)	80%	91.76 (±0.38)	93.21 (±0.17)	**93.49** (±0.20)
			90%	82.06 (±0.92)	87.55 (±0.68)	**88.44** (±0.19)
	DenseNet-40	94.10 (±0.12)	80%	92.64 (±0.12)	93.07 (±0.08)	**93.61** (±0.12)
CIFAR-100	DenseNet-40	73.82 (±0.34)	80%	69.60 (±0.22)	71.04 (±0.36)	**71.45** (±0.30)

Table 3.13: Results for Network Slimming [118] when the models are aggressively pruned. "Prune ratio" stands for the total percentage of channels that are pruned in the whole network. Larger ratios are used than in the original paper of [118].

Dataset	Model	Unpruned	Prune Ratio	Fine-tuned	Scratch-E	Scratch-B
CIFAR-10	ResNet-56	93.14 (±0.12)	90%	89.17 (±0.08)	91.02 (±0.12)	**91.93** (±0.26)

Table 3.14: Results for L_1-norm based filter pruning [107] when the models are aggressively pruned.

Dataset	Model	Unpruned	Prune Ratio	Fine-tuned	Scratch-E	Scratch-B
CIFAR-10	VGG-19	93.50 (±0.11)	95%	93.34 (±0.13)	93.21 (±0.17)	**93.63** (±0.18)
CIFAR-100	VGG-19	71.70 (±0.31)	95%	70.22 (±0.38)	70.88 (±0.35)	**72.08** (±0.15)

Table 3.15: Results for unstructured pruning [62] when the models are aggressively pruned.

3.8.4 Extending Fine-tuning Epochs

Generally, pruning algorithms use fewer epochs for fine-tuning than training the large model [107, 71, 125]. For example, L_1-norm based filter pruning [107] uses 164 epochs for training on CIFAR-10 datasets, and only fine-tunes the pruned networks for 40 epochs. This is due to that mostly small learning rates are used for fine-tuning to better preserve the weights from the large model. Here we experiment with fine-tuning for more epochs (e.g., for the same number of epochs as Scratch-E) and show it does not bring noticeable performance improvement.

Dataset	Model	Pruned Model	Fine-tune-40	Fine-tune-80	Fine-tune-160	Scratch-E
CIFAR-10	VGG-16	VGG-16-A	93.40 (±0.12)	93.45 (±0.06)	93.45 (±0.08)	**93.62** (±0.11)
	ResNet-56	ResNet-56-A	**92.97** (±0.17)	92.92 (±0.15)	92.94 (±0.16)	92.96 (±0.26)
		ResNet-56-B	92.68 (±0.19)	92.67 (±0.14)	**92.76** (±0.16)	92.54 (±0.19)
	ResNet-110	ResNet-110-A	93.14 (±0.16)	93.12 (±0.19)	93.04 (±0.22)	**93.25** (±0.29)
		ResNet-110-B	92.69 (±0.09)	92.75 (±0.15)	92.76 (±0.16)	**92.89** (±0.43)

Table 3.16: Results for extending fine-tuning. "Fine-tune-40" stands for fine-tuning 40 epochs and so on. Scratch-E models are trained for 160 epochs. We observe that fine-tuning for more epochs does not help improve the accuracy much, and models trained from scratch can still perform on par with fine-tuned models.

We use L_1-norm filter pruning [107] for this experiment. Table 3.16 shows our results with different numbers of epochs for fine-tuning. It can be seen that fine-tuning for more epochs gives negligible accuracy increase and sometimes small decrease, and Scratch-E models are still on par with models fine-tuned for enough epochs.

3.8.5 Extending the Standard Training Schedule

In our experiments, we use the standard training schedule for both CIFAR (160 epochs) and ImageNet (90 epochs). Here we show that our observation still holds after we extend the standard training schedule. We use L_1-norm based filter pruning [107] for this experiment. Table 3.17 shows our results when we extend the standard training schedule of CIFAR from 160 to 300 epochs. We observe that scratch-trained models still perform at least on par with fine-tuned models.

Dataset	Model	Unpruned	Pruned Model	Fine-tuned	Scratch-E	Scratch-B
CIFAR-10	VGG-16	93.79 (±0.05)	VGG-16-A	93.67 (±0.11)	93.74 (±0.14)	**93.80** (±0.09)
	ResNet-56	93.52 (±0.05)	ResNet-56-A	93.44 (±0.15)	93.34 (±0.17)	**93.56** (±0.09)
			ResNet-56-B	93.12 (±0.20)	93.14 (±0.21)	**93.30** (±0.17)
	ResNet-110	93.82 (±0.32)	ResNet-110-A	93.75 (±0.24)	93.80 (±0.15)	**94.10** (±0.12)
			ResNet-110-B	93.36 (±0.28)	93.75 (±0.16)	**93.90** (±0.17)

Table 3.17: Results for L_1-norm filter pruning [107] when the training schedule of the large model is extended from 160 to 300 epochs.

3.8.6 Weight Distributions

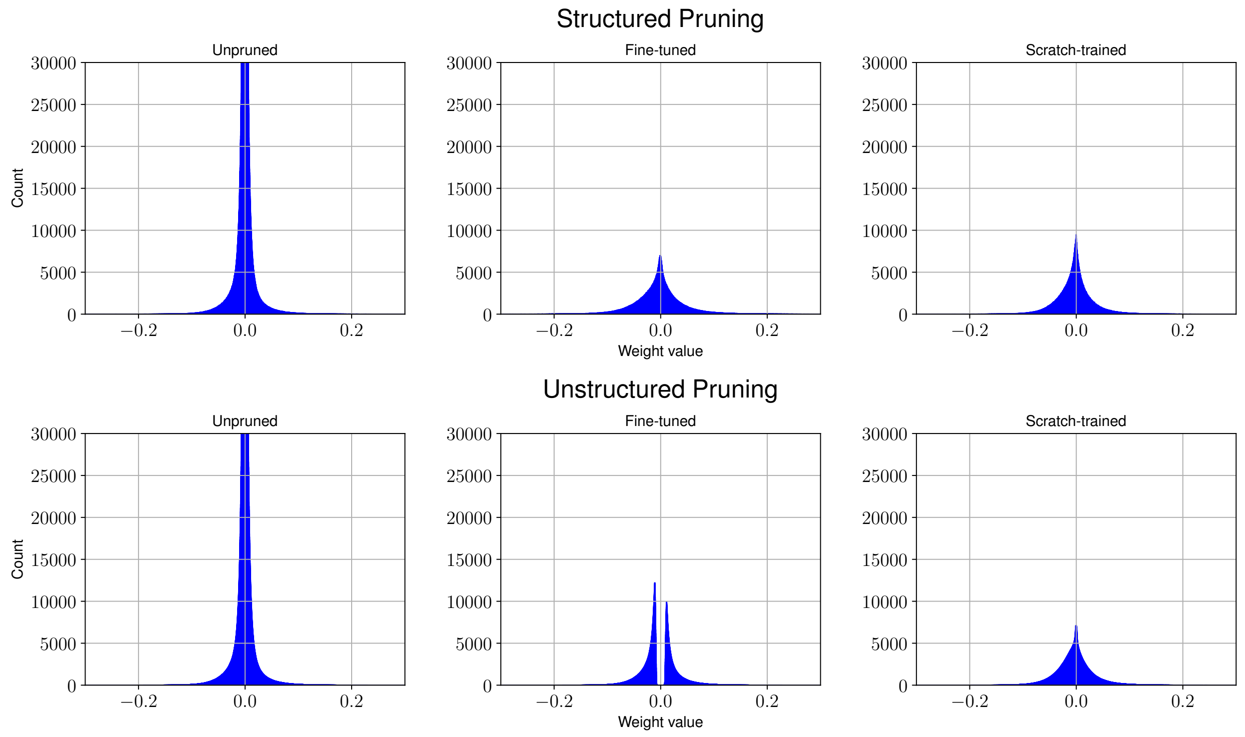


Figure 3.8: Weight distribution of convolutional layers for different pruning methods. We use VGG-16 and CIFAR-10 for this visualization. We compare the weight distribution of unpruned models, fine-tuned models and scratch-trained models. *Top*: Results for Network Slimming. *Bottom*: Results for unstructured pruning.

Accompanying the discussion in Section 3.5.3, we show the weight distribution of unpruned large models, fine-tuned pruned models and scratch-trained pruned models, for two pruning methods: (structured) Network Slimming [118] and unstructured pruning [62]. We choose

VGG-16 and CIFAR-10 for visualization and compare the weight distribution of unpruned models, fine-tuned models and scratch-trained models. For Network Slimming, the prune ratio is 50%. For unstructured pruning, the prune ratio is 80%. Figure 3.8 shows our result. We can see that the weight distribution of fine-tuned models and scratch-trained pruned models are different from the unpruned large models – the weights that are close to zero are much fewer. This seems to imply that there are fewer redundant structures in the found pruned architecture, and support the view of architecture learning for automatic pruning methods.

For unstructured pruning, the fine-tuned model also has a significantly different weight distribution from the scratch-trained model – it has nearly no close-to-zero values. This might be a potential reason why sometimes models trained from scratch cannot achieve the accuracy of the fine-tuned models, as shown in Table 3.6.

3.8.7 More Sparsity Patterns for Pruned Architectures

In this section we provide the additional results on sparsity patterns for the pruned models, accompanying the discussions of "More Analysis" in Section 3.6.

	10%	20%	30%	40%	50%	60%	70%
Stage 1	0.879	0.729	0.557	0.484	0.421	0.349	0.271
Stage 2	0.959	0.863	0.754	0.651	0.537	0.428	0.320
Stage 3	0.889	0.798	0.716	0.610	0.507	0.403	0.301

Table 3.18: Sparsity patterns of PreResNet-164 pruned on CIFAR-10 by Network Slimming shown in Figure 3.5 (left) under different prune ratio. The top row denotes the total prune ratio. The values denote the ratio of channels to be kept. We can observe that for a certain prune ratio, the sparsity patterns are close to uniform across stages.

	10%			50%			90%		
	0.905	0.905	0.909	0.530	0.561	0.538	0.129	0.171	0.133
Stage 1	0.900	0.912	0.899	0.559	0.588	0.551	0.166	0.217	0.176
	0.903	0.913	0.902	0.532	0.563	0.547	0.142	0.172	0.163
	0.906	0.911	0.906	0.485	0.523	0.503	0.073	0.102	0.085
Stage 2	0.912	0.911	0.915	0.508	0.529	0.525	0.099	0.114	0.111
	0.911	0.916	0.912	0.502	0.529	0.519	0.080	0.113	0.096
	0.901	0.904	0.900	0.454	0.475	0.454	0.043	0.059	0.048
Stage 3	0.885	0.891	0.889	0.409	0.420	0.415	0.032	0.033	0.035
	0.898	0.903	0.902	0.450	0.468	0.458	0.042	0.055	0.046

Table 3.19: Average sparsity patterns of 3×3 kernels of PreResNet-110 pruned on CIFAR-100 by unstructured pruning shown in Figure 3.5 (middle) under different prune ratio. The top row denotes the total prune ratio. The values denote the ratio of weights to be kept. We can observe that for a certain prune ratio, the sparsity patterns are close to uniform across stages.

	10%			50%			90%		
	0.861	0.856	0.858	0.507	0.495	0.510	0.145	0.129	0.142
Stage 1	0.843	0.844	0.851	0.484	0.486	0.479	0.123	0.115	0.126
	0.850	0.854	0.857	0.509	0.490	0.511	0.136	0.131	0.147
	0.907	0.905	0.906	0.498	0.487	0.499	0.099	0.088	0.100
Stage 2	0.892	0.888	0.892	0.442	0.427	0.444	0.064	0.043	0.065
	0.907	0.906	0.905	0.497	0.485	0.493	0.095	0.082	0.098
	0.897	0.901	0.899	0.470	0.475	0.472	0.060	0.060	0.064
Stage 3	0.888	0.890	0.889	0.433	0.437	0.435	0.040	0.040	0.042
	0.898	0.900	0.899	0.473	0.477	0.473	0.060	0.061	0.063

Table 3.20: Average sparsity patterns of 3×3 kernels of DenseNet-40 pruned on CIFAR-100 by unstructured pruning in Figure 3.5 (right) under different prune ratios. The top row denotes the total prune ratio. The values denote the ratio of weights to be kept. We can observe that for a certain prune ratio, the sparsity patterns are close to uniform across stages.

	10%	20%	30%	40%	50%	60%
Stage 1	0.969	0.914	0.883	0.875	0.844	0.836
Stage 2	1.000	1.000	1.000	1.000	1.000	1.000
Stage 3	0.991	0.975	0.966	0.957	0.947	0.947
Stage 4	0.861	0.718	0.575	0.446	0.312	0.258
Stage 5	0.871	0.751	0.626	0.486	0.352	0.132

Table 3.21: Sparsity patterns of VGG-16 pruned on CIFAR-10 by Network Slimming in Figure 3.3 (left) under different prune ratios. The top row denotes the total prune ratio. The values denote the ratio of channels to be kept. For each prune ratio, the latter stages tend to have more redundancy than the earlier stages.

3.9 Conclusions

Our results encourage more careful and fair baseline evaluations of structured pruning methods. In addition to high accuracy, training predefined target models from scratch has the following benefits over conventional network pruning procedures: a) since the model is smaller, we can train the model using less GPU memory and possibly faster than training the original large model; b) there is no need to implement the pruning criterion and procedure, which sometimes requires fine-tuning layer by layer [125] and/or needs to be customized for different network architectures [107, 118]; c) we avoid tuning additional hyper-parameters involved in the pruning procedure.

Our results do support the viewpoint that automatic structured pruning finds efficient architectures in some cases. However, if the accuracy of pruning and fine-tuning is achievable by training the pruned model from scratch, it is also important to evaluate the pruned architectures against uniformly pruned baselines (both training from scratch), to demonstrate the method's value in identifying efficient architectures. If the uniformly pruned models are not worse, one could also skip the pipeline and train them from scratch.

Even if pruning and fine-tuning fail to outperform the mentioned baselines in terms of accuracy, there are still some cases where using this conventional wisdom can be much faster than training from scratch: a) when a pre-trained large model is already given and little or no training budget is available; we also note that pre-trained models can only be used when the method does not require modifications to the large model training process; b) there is a need to obtain multiple models of different sizes, or one does not know what the desired size is, in which situations one can train a large model and then prune it by different ratios.

Chapter 4

A ConvNet for the 2020s

4.1 Overview

Pruning is a popular type of methods for scaling models *down*. In the last chapter, through empirical study, we demonstrated the true value of structured pruning is not to obtain a specific set of weight values, but to identify a useful sub-architecture. Now we turn our focus on scaling *up* computer vision neural architectures.

A classical example is the invention of ResNets [64] with its proposal of skip connections. Scaling a "plain" network without skip connections to tens of layers will cause the training loss to *increase*, not to mention the worse test accuracy. However, a ResNet be scaled up to more than 100 layers with both improved training loss and test accuracy. In the 2020s, Vision Transformers [39] start to show even greater scalability than convolution-based ResNets, with extremely large data and models. In this study, we try to understand what is behind Transformers' scaling success by comparing them with modernized ConvNets.

As mentioned above, 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. Our code is available at `https://github.com/facebookresearch/ConvNeXt`.

4.2 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 [105], it was not until late 2012 that we saw its true potential for visual feature learning. The introduction of AlexNet [99] precipitated the "ImageNet moment" [152], ushering in a new era of computer vision. The field has since evolved at a rapid speed. Representative ConvNets like VGGNet [160], Inceptions [168], ResNe(X)t [64, 201], DenseNet [85], MobileNet [79], EfficientNet [170] and RegNet [141] focused on different aspects of accuracy, efficiency and scalability, and popularized many useful design principles.

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 [155]. For many decades, this has been the default use of ConvNets, generally on limited object categories such as digits [106], faces [180, 150] and pedestrians [156, 37]. Entering the 2010s, the region-based detectors [47, 46, 148, 67] further elevated 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

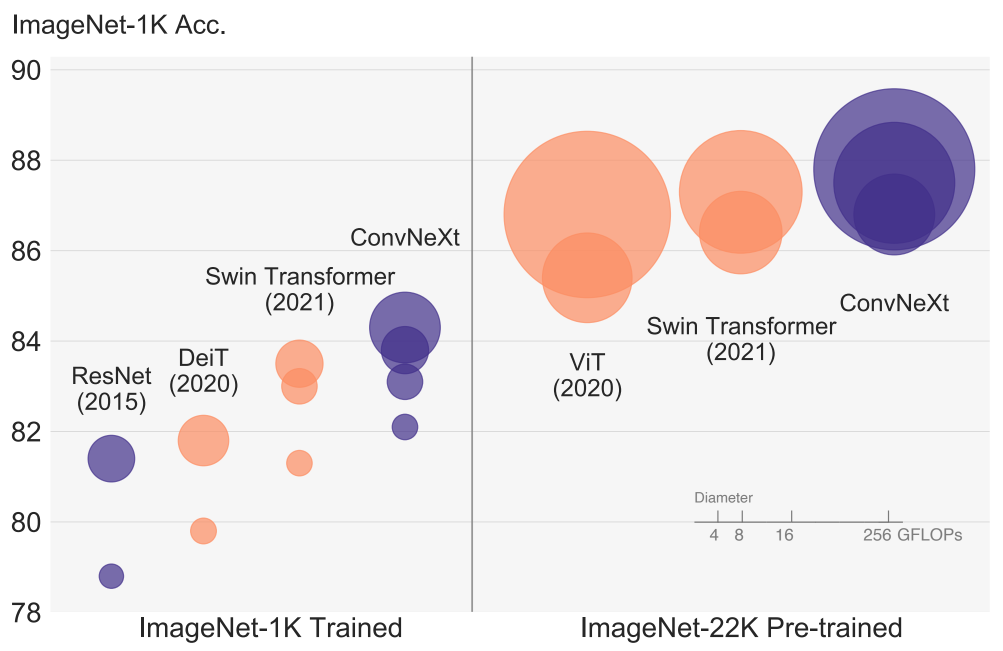


Figure 4.1: ImageNet-1K classification results for ● ConvNets and ○ vision Transformers. Each bubble's area is proportional to FLOPs of a variant in a model family. ImageNet-1K/22K models here take $224^2/384^2$ images respectively. ResNet and ViT results were obtained with improved training procedures over the original papers. We demonstrate that a standard ConvNet model can achieve the same level of scalability as hierarchical vision Transformers while being much simpler in design.

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 [117] 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 [143]; with advanced approaches such as cyclic shifting [117], 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.

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 [34], object detection/segmentation on COCO[110], and semantic segmentation on ADE20K [214]. 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.

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

4.3 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 Section 4.8.2.

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

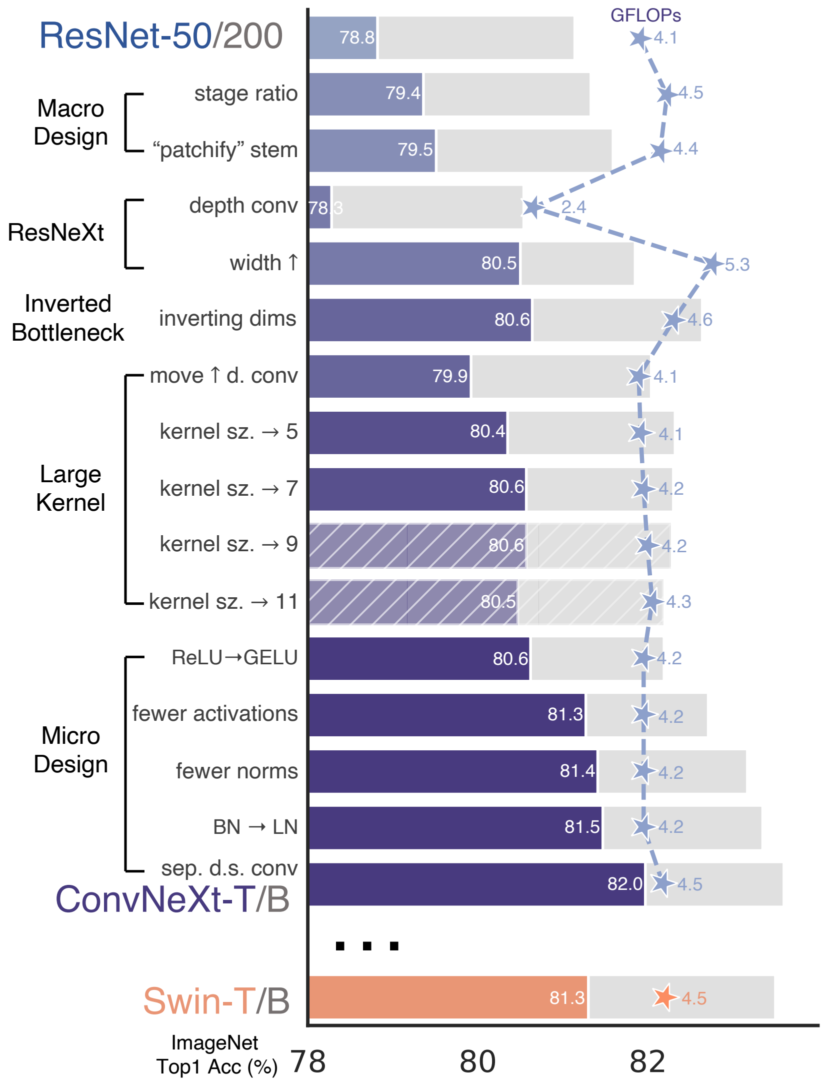


Figure 4.2: We modernize a standard ConvNet (ResNet) towards the design of a hierarchical vision Transformer (Swin), without introducing any attention-based modules. The foreground bars are model accuracies in the ResNet-50/Swin-T FLOP regime; results for the ResNet-200/Swin-B regime are shown with the gray bars. A hatched bar means the modification is not adopted. Detailed results for both regimes are in Section 4.3. 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.

model. We first 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 4.2, 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.

4.3.1 Training Techniques

Apart from the design of the network architecture, the training procedure also affects the ultimate performance. Not 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. Recent studies [12, 192] demonstrate that 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 [177] and Swin Transformer's [117]. The training is extended to 300 epochs from the original 90 epochs for ResNets. We use the AdamW optimizer [121], data augmentation techniques such as Mixup [209], Cutmix [208], RandAugment [29], Random Erasing [213], and regularization schemes including Stochastic Depth [84] and Label Smoothing [169]. The complete set of hyper-parameters we use can be found in Section 4.7.1. By itself, this enhanced training recipe increased the performance of the ResNet-50 model from 76.1% [139] 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.

4.3.2 Macro Design

We now analyze Swin Transformers' macro network design. Swin Transformers follow ConvNets [161, 64] 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, 3), 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 [142, 141], and a more optimal design is likely to exist.

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 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.

4.3.3 ResNeXt-ify

In this part, we attempt to adopt the idea of ResNeXt [201], 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 [79] and Xception [22]. 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 combination of depthwise conv and 1×1 convs leads to a separation of spatial and channel mixing, a property shared by vision Transformers, where each operation either mixes information across spatial or channel dimension, but not both. 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.

4.3.4 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 4.4). 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 [154], and has subsequently gained traction in several advanced ConvNet architectures [170, 172].

Here we explore the inverted bottleneck design. Figure 4.3 (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.

4.3.5 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 [99, 168], the gold standard (popularized by VGGNet [161]) is to stack small kernel-sized (3×3) conv layers, which have efficient hardware implementations on modern GPUs [100]. 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.

Moving up depthwise conv layer. To explore large kernels, one prerequisite is to move up the position of the depthwise conv layer (Figure 4.3 (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 the benefit of larger kernel sizes reaches a saturation point at 7×7. We verified this behavior in the

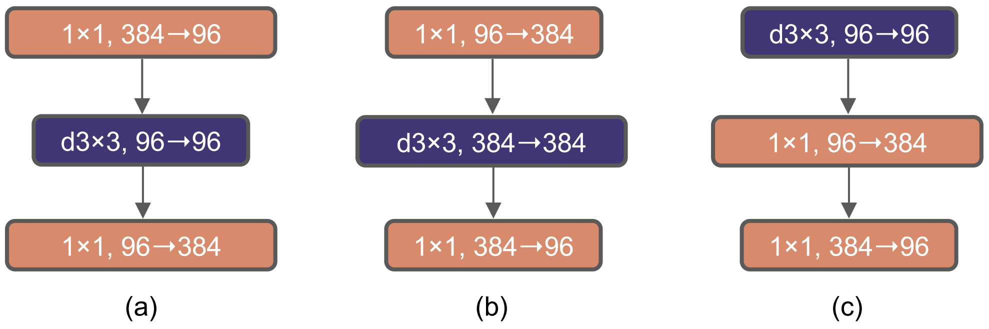


Figure 4.3: Block modifications and resulted specifications. (a) is a ResNeXt block; in (b) we create an inverted bottleneck block and in (c) the position of the spatial depthwise conv layer is moved up.

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.

4.3.6 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) [133] 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 [181]. The Gaussian Error Linear Unit, or GELU [74], which can be thought of as a smoother variant of ReLU, is utilized in the most advanced Transformers, including Google's BERT [36] and OpenAI's GPT-2 [140], 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

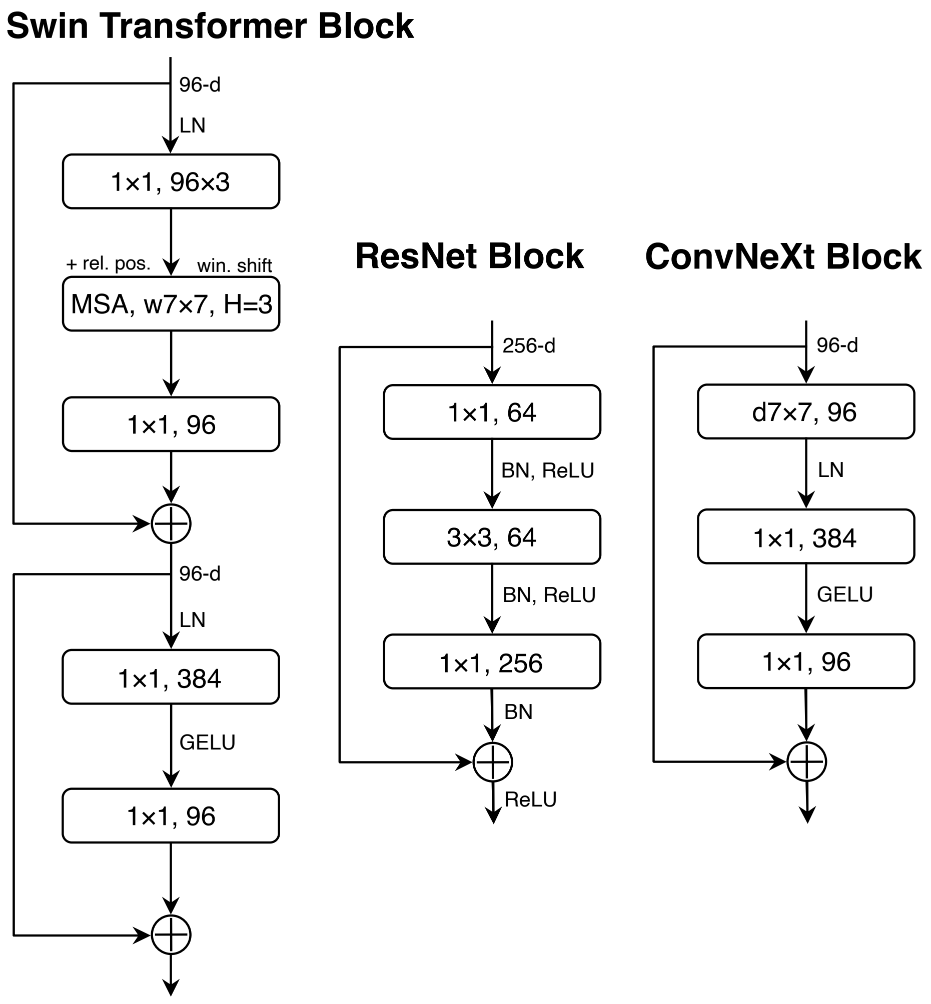


Figure 4.4: Block designs for a ResNet, a Swin Transformer, and a ConvNeXt. Swin Transformer's block is more sophisticated due to the presence of multiple specialized modules and two residual connections. For simplicity, we note the linear layers in Transformer MLP blocks also as "1×1 convs" since they are equivalent.

layer, including the 1×1 convs. Here we examine how performance changes when we stick to the same strategy. As depicted in Figure 4.4, 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.

Substituting BN with LN. BatchNorm [88] 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 [195]. There have been numerous attempts at developing alternative normalization [153, 179, 194] techniques, but BN has remained the preferred option in most vision tasks. On the other hand, the simpler Layer Normalization [7] (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 [194]. 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 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 4.4. A comparison of ResNet-50, Swin-T and ConvNeXt-T's detailed architecture specifications can be found in Table 4.8.

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 worth noting that *all design choices discussed so far are adapted from vision Transformers.* In addition, *these designs are not novel even in the ConvNet literature — 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.

4.4 Experiments on ImageNet

We construct different ConvNeXt variants, ConvNeXt-T/S/B/L, to be of similar complexities to Swin-T/S/B/L [117]. 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: $C = (256, 512, 1024, 2048)$, $B = (3, 3, 27, 3)$

4.4.1 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 Section 4.7.

Training on ImageNet-1K. We train ConvNeXts for 300 epochs using AdamW [121] 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 [209], Cutmix [208], RandAugment [29], and Random Erasing [213]. We regularize the networks with Stochastic Depth [84] and Label Smoothing [169]. Layer Scale [175] of initial value 1e-6 is applied. We use Exponential Moving Average (EMA) [138] 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 [24, 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.

4.4.2 Results

ImageNet-1K. Table 4.1 (upper) shows the result comparison with two recent Transformer variants, DeiT [177] and Swin Transformers [117], as well as two ConvNets from architecture search - RegNets [141], EfficientNets [170] and EfficientNetsV2 [171]. ConvNeXt competes favorably with two strong ConvNet baselines (RegNet [141] and EfficientNet [170]) 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.

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 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 4.1 (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 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.

On ImageNet-1K, EfficientNetV2-L, a searched architecture equipped with advanced modules (such as Squeeze-and-Excitation [82]) and progressive training procedure achieves top performance. However, with ImageNet-22K pre-training, ConvNeXt is able to outperform EfficientNetV2, further demonstrating the importance of large-scale training.

model	image size	#param.	FLOPs	throughput (image / s)	IN-1K top-1 acc.
ImageNet-1K trained models					
● RegNetY-16G [141]	224^2	84M	16.0G	334.7	82.9
● EffNet-B7 [170]	600^2	66M	37.0G	55.1	84.3
● EffNetV2-L [171]	480^2	120M	53.0G	83.7	85.7
○ DeiT-S [177]	224^2	22M	4.6G	978.5	79.8
○ DeiT-B [177]	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 [97]	384^2	388M	204.6G	-	84.4
● R-152x4 [97]	480^2	937M	840.5G	-	85.4
● EffNetV2-L [171]	480^2	120M	53.0G	83.7	86.8
● EffNetV2-XL [171]	480^2	208M	94.0G	56.5	87.3
○ ViT-B/16 (☎) [165]	384^2	87M	55.5G	93.1	85.4
○ ViT-L/16 (☎) [165]	384^2	305M	191.1G	28.5	86.8
● ConvNeXt-T	224^2	29M	4.5G	774.7	**82.9**
● ConvNeXt-T	384^2	29M	13.1G	282.8	**84.1**
● ConvNeXt-S	224^2	50M	8.7G	447.1	**84.6**
● ConvNeXt-S	384^2	50M	25.5G	163.5	**85.8**
○ 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 4.1: 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 [117]. On an A100 GPU, ConvNeXt can have a much higher throughput than Swin Transformer. See Section 4.8.3. (☎) ViT results with 90-epoch AugReg [165] training, provided through personal communication with the authors.

In Section 4.8.1, we discuss robustness and out-of-domain generalization results for ConvNeXt.

4.4.3 Isotropic ConvNeXt *vs.* ViT

In this ablation, we examine if our ConvNeXt block design is generalizable to ViT-style [39] 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 (Figure 4.4). We use the supervised training results from DeiT [177] for ViT-S/B and MAE [68] for ViT-L, as they employ improved training procedures over the original ViTs [39]. 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 4.2. We observe ConvNeXt can perform generally on par with ViT, showing that our ConvNeXt block design is competitive when used in non-hierarchical models.

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

4.5 Experiments on Downstream Tasks

4.5.1 Object detection and segmentation on COCO.

We fine-tune Mask R-CNN [67] and Cascade Mask R-CNN [14] on the COCO dataset with ConvNeXt backbones. Following Swin Transformer [117], we use multi-scale training, AdamW optimizer, and a 3× schedule. Further details and hyper-parameter settings can be found in Section 4.7.3.

Table 4.3 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) than Swin Transformers in terms of box and mask 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	16.2	46.3	64.3	50.5	40.1	61.7	43.4
● X101-32	819G	13.8	48.1	66.5	52.4	41.6	63.9	45.2
● X101-64	972G	12.6	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 4.3: 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 [52]. AP numbers of the ResNet-50 and X101 models are from [117]. We measure FPS on an A100 GPU. FLOPs are calculated with image size (1280, 800).

4.5.2 Semantic segmentation on ADE20K.

We also evaluate ConvNeXt backbones on the ADE20K semantic segmentation task with UperNet [199]. All model variants are trained for 160K iterations with a batch size of 16. Other experimental settings follow [10] (see Section 4.7.3 for more details). In Table 4.4, 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.

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 4.4: ADE20K validation results using UperNet [199]. ‡ indicates IN-22K pre-training. Swins' results are from its GitHub repository [51]. 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.

4.5.3 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 before, 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 4.1 and 4.3 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 the *ConvNet inductive bias*, and is not directly related to the self-attention mechanism in vision Transformers.

4.6 Related Work

4.6.1 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 [186, 11, 162, 143] to capture long-range dependencies. The original ViT [39] first studied a hybrid configuration, and a large body of follow-up works focused on reintroducing convolutional priors to ViT, either in an explicit [193, 203, 30, 31, 198, 41] or implicit [117] fashion.

4.6.2 Recent Convolution-based Approaches

Han *et al.* [59] 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 [178] 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 [145] 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.

4.7 Full Experimental Settings

4.7.1 ImageNet (Pre-)training

We provide ConvNeXts' ImageNet-1K training and ImageNet-22K pre-training settings in Table 4.5. The settings are used for our main results in Table 4.1 (Section 4.4.2). All ConvNeXt variants use the same setting, except the stochastic depth rate is customized for model variants.

For experiments in "modernizing a ConvNet" (Section 4.3), we also use Table 4.5's setting for ImageNet-1K, except EMA is disabled, as we find using EMA severely hurts models with BatchNorm layers.

For isotropic ConvNeXts (Section 4.4.3), the setting for ImageNet-1K in Table 4.7 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.

(pre-)training config	ConvNeXt-T/S/B/L ImageNet-1K 224^2	ConvNeXt-T/S/B/L/XL ImageNet-22K 224^2
weight init	trunc. normal (0.2)	trunc. normal (0.2)
optimizer	AdamW	AdamW
base learning rate	4e-3	4e-3
weight decay	0.05	0.05
optimizer momentum	$\beta_1, \beta_2{=}0.9, 0.999$	$\beta_1, \beta_2{=}0.9, 0.999$
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 [24, 10]	None	None
randaugment [29]	(9, 0.5)	(9, 0.5)
mixup [209]	0.8	0.8
cutmix [208]	1.0	1.0
random erasing [213]	0.25	0.25
label smoothing [169]	0.1	0.1
stochastic depth [84]	0.1/0.4/0.5/0.5	0.0/0.0/0.1/0.1/0.2
layer scale [175]	1e-6	1e-6
head init scale [175]	None	None
gradient clip	None	None
exp. mov. avg. (EMA) [138]	0.9999	None

Table 4.5: 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-T/S/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	$\beta_1, \beta_2{=}0.9, 0.999$	$\beta_1, \beta_2{=}0.9, 0.999$
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)
mixup	None	None
cutmix	None	None
random erasing	0.25	0.25
label smoothing	0.1	0.1
stochastic depth	0.8/0.95	0.0/0.1/0.2/0.3/0.4
layer scale	pre-trained	pre-trained
head init scale	0.001	0.001
gradient clip	None	None
exp. mov. avg. (EMA)	None	None(T-L)/0.9999(XL)

Table 4.6: ImageNet-1K fine-tuning settings. Multiple values (e.g., 0.8/0.95) are for each model (e.g., ConvNeXt-B/L) respectively.

4.7.2 ImageNet Fine-tuning

We list the settings for fine-tuning on ImageNet-1K in Table 4.6. 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 [177]). 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.

In fine-tuning, we use layer-wise learning rate decay [24, 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 [191, 175, 51], instead of 0.875 at 224^2.

4.7.3 Downstream Tasks

For ADE20K and COCO experiments, we follow the training settings used in BEiT [10] and Swin [117]. We also use MMDetection [18] and MMSegmentation [25] 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 [10] {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 [52].

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 are in Table 4.7.

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 4.7: ADE20K validation results with single-scale testing.

4.7.4 Detailed Architectures

We present a detailed architecture comparison between ResNet-50, ConvNeXt-T and Swin-T in Table 4.8. For differently sized ConvNeXts, only the number of blocks and the number of channels at each stage differ from ConvNeXt-T (see Section 4.4 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	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} \text{d}7\times7, 96 \\ 1\times1, 384 \\ 1\times1, 96 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 96\times3 \\ \text{MSA, w}7\times7\text{, H=3, rel. pos.} \\ 1\times1, 96 \end{bmatrix} \begin{bmatrix} 1\times1, 384 \\ 1\times1, 96 \end{bmatrix} \times 2$
res3	28×28	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} \text{d}7\times7, 192 \\ 1\times1, 768 \\ 1\times1, 192 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 192\times3 \\ \text{MSA, w}7\times7\text{, H=6, rel. pos.} \\ 1\times1, 192 \end{bmatrix} \begin{bmatrix} 1\times1, 768 \\ 1\times1, 192 \end{bmatrix} \times 2$
res4	14×14	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} \text{d}7\times7, 384 \\ 1\times1, 1536 \\ 1\times1, 384 \end{bmatrix} \times 9$	$\begin{bmatrix} 1\times1, 384\times3 \\ \text{MSA, w}7\times7\text{, H=12, rel. pos.} \\ 1\times1, 384 \end{bmatrix} \begin{bmatrix} 1\times1, 1536 \\ 1\times1, 384 \end{bmatrix} \times 6$
res5	7×7	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} \text{d}7\times7, 768 \\ 1\times1, 3072 \\ 1\times1, 768 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 768\times3 \\ \text{MSA, w}7\times7\text{, H=24, rel. pos.} \\ 1\times1, 768 \end{bmatrix} \begin{bmatrix} 1\times1, 3072 \\ 1\times1, 768 \end{bmatrix} \times 2$
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

Table 4.8: Detailed architecture specifications for ResNet-50, ConvNeXt-T and Swin-T.

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 [117]	1K/224^2	4.5 / 28.3	81.2	62.0	-	21.6	41.3	29.1
RVT-S* [126]	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 [117]	1K/224^2	15.4 / 87.8	83.4	54.4	-	35.8	46.6	32.4
RVT-B* [126]	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 4.9: Robustness evaluation of ConvNeXt. We do not make use of any specialized modules or additional fine-tuning procedures.

4.8 Additional Studies

4.8.1 Robustness Evaluation

Additional robustness evaluation results for ConvNeXt models are presented in Table 4.9. We directly test our ImageNet-1K trained/fine-tuned classification models on several robustness benchmark datasets such as ImageNet-A [75], ImageNet-R [76], ImageNet-Sketch [184] and ImageNet-C/$\bar{\text{C}}$ [73, 128] 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 [126] 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.

4.8.2 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 4.10 and 4.11. 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 before. 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%).

model	IN-1K acc.	GFLOPs
ResNet-50 (PyTorch[139])	76.13	4.09
ResNet-50 (enhanced recipe)	78.82 $\pm$ 0.07	4.09
stage ratio	79.36 $\pm$ 0.07	4.53
"patchify" stem	79.51 $\pm$ 0.18	4.42
depthwise conv	78.28 $\pm$ 0.08	2.35
increase width	80.50 $\pm$ 0.02	5.27
inverting dimensions	80.64 $\pm$ 0.03	4.64
move up depthwise conv	79.92 $\pm$ 0.08	4.07
kernel size $\rightarrow$ 5	80.35 $\pm$ 0.08	4.10
kernel size $\rightarrow$ 7	80.57 $\pm$ 0.14	4.15
kernel size $\rightarrow$ 9	80.57 $\pm$ 0.06	4.21
kernel size $\rightarrow$ 11	80.47 $\pm$ 0.11	4.29
ReLU $\rightarrow$ GELU	80.62 $\pm$ 0.14	4.15
fewer activations	81.27 $\pm$ 0.06	4.15
fewer norms	81.41 $\pm$ 0.09	4.15
BN $\rightarrow$ LN	81.47 $\pm$ 0.09	4.46
separate d.s. conv (ConvNeXt-T)	81.97 $\pm$ 0.06	4.49
Swin-T [117]	81.30	4.50

Table 4.10: 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 [66]	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 $\rightarrow$ 5	82.32	14.70
kernel size $\rightarrow$ 7	82.30	14.81
kernel size $\rightarrow$ 9	82.27	14.95
kernel size $\rightarrow$ 11	82.18	15.13
ReLU $\rightarrow$ GELU	82.19	14.81
fewer activations	82.71	14.81
fewer norms	83.17	14.81
BN $\rightarrow$ LN	83.35	14.81
separate d.s. conv (ConvNeXt-B)	83.60	15.35
Swin-B[117]	83.50	15.43

Table 4.11: Detailed results for modernizing a ResNet-200.

4.8.3 Benchmarking on A100 GPUs

Following Swin Transformer [117], the ImageNet models' inference throughputs in Table 4.1 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 [136] version 1.10 to use the latest "Channel Last" memory layout [42] for further speedup.

We present the results in Table 4.12. 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 standard ConvNet modules and simple in design, could be practically more efficient models on modern hardwares.

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 4.12: Inference throughput comparisons on an A100 GPU. Using TF32 data format and "channel last" memory layout, ConvNeXt enjoys up to ~49% higher throughput compared with a Swin Transformer with similar FLOPs.

4.9 Discussions

4.9.1 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.

4.9.2 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 work, 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.

4.10 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.

Chapter 5

Conclusions

This thesis presents three contributions that propose new network architectures or study existing architectures, for visual recognition applications. The two goals we strive for are efficiency, which stands for the pursuit of small, fast, flexible yet accurate models; and scalability, which aims for superb model accuracy when we have large compute and data. In Chapter 2, we described a method that enables both anytime and more efficient inference for pixel-wise visual recognition tasks. The framework was built on early exiting from neural networks. We further equipped it with confidence adaptivity, motivated by the observation that certain regions tend to be recognized accurately enough even at early exits. In Chapter 3, we studied a more popular family of solutions for scaling models down - pruning. The central empirical finding was that training the same pruned model from scratch is at least on par with fine-tuning it, for structured pruning methods. Our results suggests pruning's value does not lie in identifying important weights, but instead they act as a form of network architecture search. This encourages a more thorough comparison with baseline methods for future research on this topic. In Chapter 4, we looked at vision Transformers, which seem much more scalable than ConvNets, from a similar challenging angle. Our "modernizing" process gradually incorporated various training techniques and detailed architecture designs borrowed from vision Transformers into vanilla ResNets, and the eventual models, ConvNeXts, were able to surpass the performance of vision Transformers in several visual recognition benchmarks. Our findings prompt the community to reevaluate the importance of convolutional inductive biases in computer vision.

Our studies on both scaling down (pruning) and scaling up (vision Transformers) emphasizes the need for comparing with more carefully designed baselines, and attributing the reasons for performance improvement accurately. In addition to helping us gain a deeper understanding of neural network properties, it could lead to much simpler algorithms that are easier to work with in practice.

5.1 Future Directions

Inductive Biases for Vision. As Chapter 4 demonstrated, the comparison between ConvNets and Transformers is often conflated with detailed architecture design choices, leading to an underestimation of ConvNets' potential. If we also look at other recent works in the architecture world, e.g., ConvMixer [178], GFNet [145], MLP-Mixer [174], ResMLP [176], PoolFormer [207] and others, we found there are three major choices on inductive biases in designing networks: 1. hierarchical or plain features. If it uses hierarchical features, the representation resolution gradually goes down with downsampling, and typically the number of features gradually increases. If it is plain, the same feature resolutions and dimensions are used across the network; 2. Local or global computation. This has to do with each layer's operation: whether the representation in the new layer takes inputs from all spatial locations in the previous layer, or only a local neighborhood; 3. Spatial mixing, i.e., using self-attention, convolution, MLP, pooling or other alternative operations. Each network is an instantiation of various choices on these three, among other more detailed designs including activations, normalizations, etc. For example, ConvNeXt differs from ViTs in all three aspects. Which ones of these three play the most important role in affecting a network's performance in small and large model/data regimes? The understanding in the community about this question is still very much qualitative. An empirical study on different model and data scales may help us learn about the effects of these common inductive biases used in computer vision.

Self-Supervised Learning. We have evaluated our proposed models and algorithms on a variety of visual recognition tasks, including image classification, object detection, semantic segmentation and human pose estimation. Despite that, all models presented in this dissertation are trained in a supervised manner, using human annotations as learning targets. This can be undesirable for multiple reasons: 1. the heavy human effort involved in labeling prevents the scaling of data. 2. Human annotations can be inaccurate. 3. The discrete one-hot classification of images, objects or pixels is not fully representative of their true semantic meaning. Recent efforts in visual self-supervised learning have shown potential in dethroning supervised learning. Promising directions include contrastive learning [196, 69, 19], prediction-based learning [57, 16], and masked image modeling [10, 68]. In particular, masked image modeling methods showcased they could surpass supervised learning in representation power, using ViTs as backbones. However, in our preliminary studies on ConvNeXts, we encountered some difficulties trying to achieve the results obtained by ViTs with the masked image modeling paradigm, perhaps partly because ConvNets tend to average the features thus blurring the masked patches' representations. In future explorations, we would like to investigate ways to circumvent this issue and develop competitive self-supervised methods for ConvNeXts.

Bibliography

[1] Alireza Aghasi et al. "Net-trim: Convex pruning of deep neural networks with performance guarantee". In: *NIPS*. 2017.

[2] Jose M Alvarez and Mathieu Salzmann. "Learning the number of neurons in deep networks". In: *NIPS*. 2016.

[3] Manuel Amthor, Erik Rodner, and Joachim Denzler. "Impatient dnns-deep neural networks with dynamic time budgets". In: *arXiv preprint arXiv:1610.02850*. 2016.

[4] Mykhaylo Andriluka et al. "2D Human Pose Estimation: New Benchmark and State of the Art Analysis". In: *CVPR*. 2014.

[5] Sajid Anwar and Wonyong Sung. "Compact deep convolutional neural networks with coarse pruning". In: *arXiv preprint arXiv:1610.09639*. 2016.

[6] Jimmy Ba and Rich Caruana. "Do deep nets really need to be deep?" In: *NIPS*. 2014.

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

[8] Shaojie Bai, Vladlen Koltun, and J Zico Kolter. "Multiscale deep equilibrium models". In: *arXiv preprint arXiv:2006.08656*. 2020.

[9] Bowen Baker et al. "Designing neural network architectures using reinforcement learning". In: *ICLR*. 2017.

[10] Hangbo Bao, Li Dong, and Furu Wei. "BEiT: BERT Pre-Training of Image Transformers". In: *ICLR*. 2022.

[11] Irwan Bello et al. "Attention augmented convolutional networks". In: *ICCV*. 2019.

[12] Irwan Bello et al. "Revisiting resnets: Improved training and scaling strategies". In: *NeurIPS*. 2021.

[13] Tom Brown et al. "Language Models are Few-Shot Learners". In: *NeurIPS*. 2020.

[14] Zhaowei Cai and Nuno Vasconcelos. "Cascade R-CNN: Delving into High Quality Object Detection". In: *CVPR*. 2018.

[15] Shijie Cao et al. "Seernet: Predicting convolutional neural network feature-map sparsity through low-bit quantization". In: *CVPR*. 2019.

[16] Mathilde Caron et al. “Emerging Properties in Self-Supervised Vision Transformers”. In: *arXiv:2104.14294*. 2021.

[17] Miguel A Carreira-Perpinán and Yerlan Idelbayev. ““Learning-Compression” Algorithms for Neural Net Pruning”. In: *CVPR*. 2018.

[18] Kai Chen et al. “MMDetection: Open MMLab Detection Toolbox and Benchmark”. In: *arXiv:1906.07155*. 2019.

[19] Ting Chen et al. “A simple framework for contrastive learning of visual representations”. In: *ICML*. 2020.

[20] Sharan Chetlur et al. “cuDNN: Efficient Primitives for Deep Learning”. In: *arXiv preprint arXiv:1410.0759*. 2014.

[21] Ting-Wu Chin, Cha Zhang, and Diana Marculescu. “Layer-compensated pruning for resource-constrained convolutional neural networks”. In: *arXiv preprint arXiv:1810.00518*. 2018.

[22] François Chollet. “Xception: Deep learning with depthwise separable convolutions”. In: *CVPR*. 2017.

[23] Christopher Choy, JunYoung Gwak, and Silvio Savarese. “4D Spatio-Temporal ConvNets: Minkowski Convolutional Neural Networks”. In: *CVPR*. 2019.

[24] Kevin Clark et al. “ELECTRA: Pre-training text encoders as discriminators rather than generators”. In: *arXiv:2003.10555*. 2020.

[25] MMSegmentation contributors. *MMSegmentation: OpenMMLab Semantic Segmentation Toolbox and Benchmark*. `https://github.com/open-mmlab/mmsegmentation`. 2020.

[26] Marius Cordts et al. “The cityscapes dataset for semantic urban scene understanding”. In: *CVPR*. 2016.

[27] Corinna Cortes and Vladimir Vapnik. “Support-vector networks”. In: *Machine learning*. 1995.

[28] Matthieu Courbariaux et al. “Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1”. In: *arXiv preprint arXiv:1602.02830*. 2016.

[29] Ekin D Cubuk et al. “Randaugment: Practical automated data augmentation with a reduced search space”. In: *CVPR Workshops*. 2020.

[30] Stéphane d’Ascoli et al. “ConViT: Improving vision transformers with soft convolutional inductive biases”. In: *ICML*. 2021.

[31] Zihang Dai et al. “CoAtNet: Marrying Convolution and Attention for All Data Sizes”. In: *NeurIPS*. 2021.

[32] Navneet Dalal and Bill Triggs. “Histograms of oriented gradients for human detection”. In: *CVPR*. 2005.

[33] Thomas L Dean and Mark S Boddy. “An Analysis of Time-Dependent Planning”. In: *AAAI*. 1988.

[34] Jia Deng et al. “ImageNet: A large-scale hierarchical image database”. In: *CVPR*. 2009.

[35] Emily L Denton et al. “Exploiting linear structure within convolutional networks for efficient evaluation”. In: *NIPS*. 2014.

[36] Jacob Devlin et al. “BERT: Pre-training of deep bidirectional transformers for language understanding”. In: *arXiv:1810.04805*. 2019.

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

[38] Xuanyi Dong et al. “More is less: A more complicated network with less inference complexity”. In: *CVPR*. 2017.

[39] Alexey Dosovitskiy et al. “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale”. In: *ICLR*. 2021.

[40] Erich Elsen et al. “Fast sparse convnets”. In: *CVPR*. 2020.

[41] Haoqi Fan et al. “Multiscale vision transformers”. In: *ICCV*. 2021.

[42] Vitaly Fedyunin. *Tutorial: Channel Last Memory Format in PyTorch*. https://pytorch.org/tutorials/intermediate/memory_format_tutorial.html. Accessed: 2021-10-01. 2021.

[43] Michael Figurnov et al. “Spatially adaptive computation time for residual networks”. In: *CVPR*. 2017.

[44] Jonathan Frankle and Michael Carbin. “The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks”. In: *ICLR*. 2019.

[45] Björn Fröhlich, Erik Rodner, and Joachim Denzler. “As time goes by—anytime semantic segmentation with iterative context forests”. In: *Joint DAGM (German Association for Pattern Recognition) and OAGM Symposium*. Springer. 2012.

[46] Ross Girshick. “Fast R-CNN”. In: *arXiv:1504.08083*. 2015.

[47] Ross Girshick et al. “Rich feature hierarchies for accurate object detection and semantic segmentation”. In: *CVPR*. 2014.

[48] *GitHub repository: A Faster Pytorch Implementation of Faster R-CNN*. https://github.com/jwyang/faster-rcnn.pytorch. 2017.

[49] *GitHub repository: Deep High-Resolution Representation Learning for Human Pose Estimation*. https://github.com/leoxiaobin/deep-high-resolution-net.pytorch. 2019.

[50] *GitHub repository: Deep High-Resolution Representation Learning for Visual Recognition*. https://github.com/HRNet/HRNet-Semantic-Segmentation. 2019.

[51] *GitHub repository: Swin Transformer.* `https://github.com/microsoft/Swin-Transformer`. 2021.

[52] *GitHub repository: Swin Transformer for Object Detection.* `https://github.com/SwinTransformer/Swin-Transformer-Object-Detection`. 2021.

[53] Ariel Gordon et al. "Morphnet: Fast & simple resource-constrained structure learning of deep networks". In: *CVPR*. 2018.

[54] Priya Goyal et al. "Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour". In: *arXiv:1706.02677*. 2017.

[55] Benjamin Graham and Laurens van der Maaten. "Submanifold Sparse Convolutional Networks". In: *arXiv preprint arXiv:1706.01307*. 2017.

[56] Alex Graves. "Adaptive computation time for recurrent neural networks". In: *arXiv preprint arXiv:1603.08983*. 2016.

[57] Jean-Bastien Grill et al. "Bootstrap Your Own Latent - A New Approach to Self-Supervised Learning". In: *NeurIPS*. 2020.

[58] Alex Grubb and Drew Bagnell. "Speedboost: Anytime prediction with uniform near-optimality". In: *Artificial Intelligence and Statistics*. 2012.

[59] Qi Han et al. "Demystifying Local Vision Transformer: Sparse Connectivity, Weight Sharing, and Dynamic Weight". In: *arXiv:2106.04263*. 2021.

[60] Song Han, Huizi Mao, and William J Dally. "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding". In: *ICLR*. 2016.

[61] Song Han et al. "EIE: efficient inference engine on compressed deep neural network". In: *Computer Architecture (ISCA), 2016 ACM/IEEE 43rd Annual International Symposium on*. 2016.

[62] Song Han et al. "Learning both weights and connections for efficient neural network". In: *NIPS*. 2015.

[63] Babak Hassibi and David G Stork. "Second order derivatives for network pruning: Optimal brain surgeon". In: *NIPS*. 1993.

[64] Kaiming He et al. "Deep residual learning for image recognition". In: *CVPR*. 2016.

[65] Kaiming He et al. "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification". In: *CVPR*. 2015.

[66] Kaiming He et al. "Identity mappings in deep residual networks". In: *ECCV*. 2016.

[67] Kaiming He et al. "Mask R-CNN". In: *ICCV*. 2017.

[68] Kaiming He et al. "Masked autoencoders are scalable vision learners". In: *CVPR*. 2022.

[69] Kaiming He et al. "Momentum contrast for unsupervised visual representation learning". In: *CVPR*. 2020.

[70] Yang He et al. “Soft Filter Pruning for Accelerating Deep Convolutional Neural Networks”. In: *IJCAI*. 2018.

[71] Yihui He, Xiangyu Zhang, and Jian Sun. “Channel pruning for accelerating very deep neural networks”. In: *ICCV*. 2017.

[72] Yihui He et al. “Amc: Automl for model compression and acceleration on mobile devices”. In: *ECCV*. 2018.

[73] Dan Hendrycks and Thomas Dietterich. “Benchmarking Neural Network Robustness to Common Corruptions and Perturbations”. In: *ICLR*. 2018.

[74] Dan Hendrycks and Kevin Gimpel. “Gaussian error linear units (GeLUs)”. In: *arXiv:1606.08415*. 2016.

[75] Dan Hendrycks et al. “Natural adversarial examples”. In: *CVPR*. 2021.

[76] Dan Hendrycks et al. “The many faces of robustness: A critical analysis of out-of-distribution generalization”. In: *ICCV*. 2021.

[77] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. “Distilling the knowledge in a neural network”. In: *arXiv preprint arXiv:1503.02531*. 2015.

[78] Sara Hooker. “The hardware lottery”. In: *Communications of the ACM*. Vol. 64. 12. ACM New York, NY, USA, 2021, pp. 58–65.

[79] Andrew G Howard et al. “MobileNets: Efficient convolutional neural networks for mobile vision applications”. In: *arXiv:1704.04861*. 2017.

[80] Hanzhang Hu et al. “Learning anytime predictions in neural networks via adaptive loss balancing”. In: *AAAI*. 2019.

[81] Hengyuan Hu et al. “Network trimming: A data-driven neuron pruning approach towards efficient deep architectures”. In: *arXiv preprint arXiv:1607.03250*. 2016.

[82] Jie Hu, Li Shen, and Gang Sun. “Squeeze-and-excitation networks”. In: *CVPR*. 2018.

[83] Gao Huang et al. “Condensenet: An efficient densenet using learned group convolutions”. In: *CVPR*. 2018.

[84] Gao Huang et al. “Deep networks with stochastic depth”. In: *ECCV*. 2016.

[85] Gao Huang et al. “Densely Connected Convolutional Networks”. In: *CVPR*. 2017.

[86] Gao Huang et al. “Multi-scale dense networks for resource efficient image classification”. In: *arXiv preprint arXiv:1703.09844*. 2017.

[87] Zehao Huang and Naiyan Wang. “Data-Driven Sparse Structure Selection for Deep Neural Networks”. In: *ECCV*. 2018.

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

[89] Sergey Ioffe and Christian Szegedy. “Batch normalization: Accelerating deep network training by reducing internal covariate shift”. In: *arXiv preprint arXiv:1502.03167*. 2015.

[90] Benoit Jacob et al. “Quantization and training of neural networks for efficient integer-arithmetic-only inference”. In: *CVPR*. 2018.

[91] Eric Jang, Shixiang Gu, and Ben Poole. “Categorical reparameterization with gumbel-softmax”. In: *arXiv preprint arXiv:1611.01144*. 2016.

[92] Yangqing Jia et al. “Caffe: Convolutional architecture for fast feature embedding”. In: *ACM Multimedia*. 2014.

[93] John Jumper et al. “Highly accurate protein structure prediction with AlphaFold”. In: *Nature*. 2021.

[94] Sergey Karayev, Mario Fritz, and Trevor Darrell. “Anytime recognition of objects and scenes”. In: *CVPR*. 2014.

[95] Diederik P Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *ICLR*. 2015.

[96] Alexander Kirillov et al. “Pointrend: Image segmentation as rendering”. In: *CVPR*. 2020.

[97] Alexander Kolesnikov et al. “Big Transfer (BiT): General visual representation learning”. In: *ECCV*. 2020.

[98] Alex Krizhevsky. “Learning multiple layers of features from tiny images”. In: *Tech Report*. 2009.

[99] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “Imagenet classification with deep convolutional neural networks”. In: *NIPS*. 2012.

[100] Andrew Lavin and Scott Gray. “Fast Algorithms for Convolutional Neural Networks”. In: *CVPR*. 2016.

[101] Vadim Lebedev and Victor Lempitsky. “Fast convnets using group-wise brain damage”. In: *CVPR*. 2016.

[102] Vadim Lebedev et al. “Speeding-up convolutional neural networks using fine-tuned cp-decomposition”. In: *ICLR*. 2014.

[103] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. “Deep learning”. In: *Nature*. 2015.

[104] Yann LeCun, John S Denker, and Sara A Solla. “Optimal brain damage”. In: *NIPS*. 1990.

[105] Yann LeCun et al. “Backpropagation applied to handwritten zip code recognition”. In: *Neural computation*. MIT Press, 1989.

[106] Yann LeCun et al. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE*. 1998.

[107] Hao Li et al. "Pruning filters for efficient convnets". In: *ICLR*. 2017.

[108] Xiaoxiao Li et al. "Not all pixels are equal: Difficulty-aware semantic segmentation via deep layer cascade". In: *CVPR*. 2017.

[109] Ji Lin et al. "Runtime neural pruning". In: *NIPS*. 2017.

[110] Tsung-Yi Lin et al. "Microsoft COCO: Common objects in context". In: *ECCV*. 2014.

[111] Yingyan Lin et al. "Predictivenet: An energy-efficient convolutional neural network via zero prediction". In: *2017 IEEE international symposium on circuits and systems (ISCAS)*. IEEE. 2017.

[112] Zachary C Lipton and Jacob Steinhardt. "Troubling trends in machine learning scholarship". In: *arXiv preprint arXiv:1807.03341*. 2018.

[113] Buyu Liu and Xuming He. "Learning dynamic hierarchical models for anytime scene labeling". In: *ECCV*. Springer. 2016.

[114] Hanxiao Liu, Karen Simonyan, and Yiming Yang. "Darts: Differentiable architecture search". In: *arXiv preprint arXiv:1806.09055*. 2018.

[115] Hanxiao Liu et al. "Hierarchical representations for efficient architecture search". In: *ICLR*. 2018.

[116] Lanlan Liu and Jia Deng. "Dynamic deep neural networks: Optimizing accuracy-efficiency trade-offs by selective execution". In: *arXiv preprint arXiv:1701.00299*. 2017.

[117] Ze Liu et al. "Swin transformer: Hierarchical vision transformer using shifted windows". In: *ICCV*. 2021.

[118] Zhuang Liu et al. "Learning efficient convolutional networks through network slimming". In: *ICCV*. 2017.

[119] Zhuang Liu et al. "Rethinking the value of network pruning". In: *ICLR*. 2019.

[120] Jonathan Long, Evan Shelhamer, and Trevor Darrell. "Fully convolutional networks for semantic segmentation". In: *CVPR*. 2015.

[121] Ilya Loshchilov and Frank Hutter. "Decoupled weight decay regularization". In: *ICLR*. 2019.

[122] Christos Louizos, Max Welling, and Diederik P Kingma. "Learning Sparse Neural Networks through L_0 Regularization". In: *ICLR*. 2018.

[123] David G. Lowe. "Distinctive image features from scale-invariant keypoints". In: *IJCV*. 2004.

[124] David Luebke. "CUDA: Scalable parallel programming for high-performance scientific computing". In: *IEEE international symposium on biomedical imaging: from nano to macro*. 2008.

[125] Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. "Thinet: A filter level pruning method for deep neural network compression". In: *ICCV*. 2017.

[126] Xiaofeng Mao et al. "Towards robust vision transformer". In: *arXiv preprint arXiv:2105.07926*. 2021.

[127] Mason McGill and Pietro Perona. "Deciding how to decide: Dynamic routing in artificial neural networks". In: *arXiv preprint arXiv:1703.06217*. 2017.

[128] Eric Mintun, Alexander Kirillov, and Saining Xie. "On Interaction Between Augmentations and Corruptions in Natural Corruption Robustness". In: *NeurIPS*. 2021.

[129] Deepak Mittal et al. "Recovering from random pruning: On the plasticity of deep convolutional neural networks". In: *arXiv preprint arXiv:1801.10447*. 2018.

[130] Dmitry Molchanov, Arsenii Ashukha, and Dmitry Vetrov. "Variational dropout sparsifies deep neural networks". In: *ICML*. 2017.

[131] Pavlo Molchanov et al. "Pruning convolutional neural networks for resource efficient inference". In: *arXiv preprint arXiv:1611.06440*. 2016.

[132] Roozbeh Mottaghi et al. "The Role of Context for Object Detection and Semantic Segmentation in the Wild". In: *CVPR*. 2014.

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

[134] John D Owens et al. "GPU computing". In: *Proceedings of the IEEE*. 2008.

[135] Diederik P. Kingma, Tim Salimans, and Max Welling. "Variational Dropout and the Local Reparameterization Trick". In: *NIPS*. 2015.

[136] Adam Paszke et al. "Pytorch: An imperative style, high-performance deep learning library". In: *NeurIPS*. 2019.

[137] Hieu Pham et al. "Efficient Neural Architecture Search via Parameter Sharing". In: *ICML*. 2018.

[138] Boris T Polyak and Anatoli B Juditsky. "Acceleration of stochastic approximation by averaging". In: *SIAM booktitle on Control and Optimization*. 1992.

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

[140] Alec Radford et al. "Language models are unsupervised multitask learners". In: 2019.

[141] Ilija Radosavovic et al. "Designing network design spaces". In: *CVPR*. 2020.

[142] Ilija Radosavovic et al. "On network design spaces for visual recognition". In: *ICCV*. 2019.

[143] Prajit Ramachandran et al. "Stand-alone self-attention in vision models". In: *NeurIPS*. 2019.

[144] Aditya Ramesh et al. "Hierarchical text-conditional image generation with clip latents". In: *arXiv preprint arXiv:2204.06125*. 2022.

[145] Yongming Rao et al. “Global filter networks for image classification”. In: *NeurIPS*. 2021.

[146] Mohammad Rastegari et al. “Xnor-net: Imagenet classification using binary convolutional neural networks”. In: *ECCV*. 2016.

[147] Mengye Ren et al. “Sbnet: Sparse blocks network for fast inference”. In: *CVPR*. 2018.

[148] Shaoqing Ren et al. “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks”. In: *NeurIPS*. 2015.

[149] Adriana Romero et al. “Fitnets: Hints for thin deep nets”. In: *ICLR*. 2015.

[150] Henry A Rowley, Shumeet Baluja, and Takeo Kanade. “Neural network-based face detection”. In: *TPAMI*. 1998.

[151] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. “Learning representations by back-propagating errors”. In: *Nature*. 1986.

[152] Olga Russakovsky et al. “ImageNet Large Scale Visual Recognition Challenge”. In: *IJCV*. 2015.

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

[154] Mark Sandler et al. “Mobilenetv2: Inverted residuals and linear bottlenecks”. In: *CVPR*. 2018.

[155] Pierre Sermanet et al. “OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks”. In: *ICLR*. 2014.

[156] Pierre Sermanet et al. “Pedestrian detection with unsupervised multi-stage feature learning”. In: *CVPR*. 2013.

[157] Zhiqiang Shen et al. “DSOD: Learning deeply supervised object detectors from scratch”. In: *ICCV*. 2017.

[158] Gil Shomron et al. “Thanks for Nothing: Predicting Zero-Valued Activations with Lightweight Convolutional Neural Networks”. In: *arXiv preprint arXiv:1909.07636*. 2019.

[159] David Silver et al. “Mastering the game of go without human knowledge”. In: *Nature*. 2017.

[160] Karen Simonyan and Andrew Zisserman. “Two-stream convolutional networks for action recognition in videos”. In: *NeurIPS*. 2014.

[161] Karen Simonyan and Andrew Zisserman. “Very deep convolutional networks for large-scale image recognition”. In: *ICLR*. 2015.

[162] Aravind Srinivas et al. “Bottleneck transformers for visual recognition”. In: *CVPR*. 2021.

[163] Suraj Srinivas and R Venkatesh Babu. “Data-free parameter pruning for deep neural networks”. In: *BMVC*. 2015.

[164] Nitish Srivastava et al. “Dropout: A simple way to prevent neural networks from overfitting”. In: *The booktitle of Machine Learning Research*. 2014, pp. 1929–1958.

[165] Andreas Steiner et al. “How to train your vit? data, augmentation, and regularization in vision transformers”. In: *arXiv preprint arXiv:2106.10270*. 2021.

[166] Xavier Suau et al. “Principal Filter Analysis for Guided Network Compression”. In: *arXiv preprint arXiv:1807.10585*. 2018.

[167] Ke Sun et al. “Deep high-resolution representation learning for human pose estimation”. In: *CVPR*. 2019.

[168] Christian Szegedy et al. “Going deeper with convolutions”. In: *arXiv:1409.4842*. 2015.

[169] Christian Szegedy et al. “Rethinking the Inception Architecture for Computer Vision”. In: *CVPR*. 2016.

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

[171] Mingxing Tan and Quoc Le. “Efficientnetv2: Smaller models and faster training”. In: *ICML*. 2021.

[172] Mingxing Tan et al. “Mnasnet: Platform-aware neural architecture search for mobile”. In: *CVPR*. 2019.

[173] Surat Teerapittayanon, Bradley McDanel, and Hsiang-Tsung Kung. “Branchynet: Fast inference via early exiting from deep neural networks”. In: *ICPR*. IEEE. 2016.

[174] Ilya O Tolstikhin et al. “Mlp-mixer: An all-mlp architecture for vision”. In: *NeurIPS*. 2021.

[175] Hugo Touvron et al. “Going deeper with Image Transformers”. In: *ICCV*. 2021.

[176] Hugo Touvron et al. “Resmlp: Feedforward networks for image classification with data-efficient training”. In: *arXiv preprint arXiv:2105.03404*. 2021.

[177] Hugo Touvron et al. “Training data-efficient image transformers & distillation through attention”. In: *arXiv:2012.12877*. 2020.

[178] Asher Trockman and J Zico Kolter. “Patches are all you need?” In: *arXiv preprint arXiv:2201.09792*. 2022.

[179] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. “Instance Normalization: The Missing Ingredient for Fast Stylization”. In: *arXiv:1607.08022*. 2016.

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

[181] Ashish Vaswani et al. “Attention Is All You Need”. In: *arXiv:1706.03762*. 2017.

[182] Andreas Veit and Serge Belongie. “Convolutional networks with adaptive inference graphs”. In: *ECCV*. 2018.

[183] Thomas Verelst and Tinne Tuytelaars. “Dynamic convolutions: Exploiting spatial sparsity for faster inference”. In: *CVPR*. 2020.

[184] Haohan Wang et al. “Learning robust global representations by penalizing local predictive power”. In: *NeurIPS*. 2019.

[185] Jingdong Wang et al. “Deep high-resolution representation learning for visual recognition”. In: *TPAMI*. IEEE, 2020.

[186] Xiaolong Wang et al. “Non-local Neural Networks”. In: *CVPR*. 2018.

[187] Xin Wang et al. “Skipnet: Learning dynamic routing in convolutional networks”. In: *ECCV*. 2018.

[188] Yan Wang et al. “Anytime stereo image depth estimation on mobile devices”. In: *ICRA*. 2019.

[189] Yan Wang et al. “Resource aware person re-identification across multiple resolutions”. In: *CVPR*. 2018.

[190] Wei Wen et al. “Learning structured sparsity in deep neural networks”. In: *NIPS*. 2016.

[191] Ross Wightman. *GitHub repository: PyTorch Image Models*. `https://github.com/rwightman/pytorch-image-models`. 2019.

[192] Ross Wightman, Hugo Touvron, and Hervé Jégou. “ResNet strikes back: An improved training procedure in timm”. In: *arXiv:2110.00476*. 2021.

[193] Haiping Wu et al. “Cvt: Introducing convolutions to vision transformers”. In: *ICCV*. 2021.

[194] Yuxin Wu and Kaiming He. “Group Normalization”. In: *ECCV*. 2018.

[195] Yuxin Wu and Justin Johnson. “Rethinking "Batch" in BatchNorm”. In: *arXiv:2105.07576*. 2021.

[196] Zhirong Wu et al. “Unsupervised Feature Learning via Non-Parametric Instance Discrimination”. In: *arXiv:1805.01978*. 2018.

[197] Zuxuan Wu et al. “BlockDrop: Dynamic Inference Paths in Residual Networks”. In: *CVPR*. 2018.

[198] Tete Xiao et al. “Early convolutions help transformers see better”. In: *NeurIPS*. 2021.

[199] Tete Xiao et al. “Unified perceptual parsing for scene understanding”. In: *ECCV*. 2018.

[200] Lingxi Xie and Alan L Yuille. “Genetic CNN.” In: *ICCV*. 2017.

[201] Saining Xie et al. “Aggregated residual transformations for deep neural networks”. In: *CVPR*. 2017.

[202] Zhenda Xie et al. “Spatially Adaptive Inference with Stochastic Feature Sampling and Interpolation”. In: *arXiv preprint arXiv:2003.08866*. 2020.

[203] Weijian Xu et al. “Co-scale conv-attentional image transformers”. In: *ICCV*. 2021.

[204] Jianbo Ye et al. “Rethinking the Smaller-Norm-Less-Informative Assumption in Channel Pruning of Convolution Layers”. In: *ICLR*. 2018.

[205] Yang You et al. “Large batch optimization for deep learning: Training bert in 76 minutes”. In: *arXiv preprint arXiv:1904.00962*. 2019.

[206] Ruichi Yu et al. “Nisp: Pruning networks using neuron importance score propagation”. In: *CVPR*. 2018.

[207] Weihao Yu et al. “MetaFormer Is Actually What You Need for Vision”. In: *arXiv preprint arXiv:2111.11418*. 2021.

[208] Sangdoo Yun et al. “Cutmix: Regularization strategy to train strong classifiers with localizable features”. In: *ICCV*. 2019.

[209] Hongyi Zhang et al. “mixup: Beyond empirical risk minimization”. In: *arXiv:1710.09412*. 2018.

[210] Xiangyu Zhang et al. “Shufflenet: An extremely efficient convolutional neural network for mobile devices”. In: *CVPR*. 2018.

[211] Hengshuang Zhao et al. “ICNet for Real-Time Semantic Segmentation on High-Resolution Images”. In: *ECCV*. 2018.

[212] Hengshuang Zhao et al. “Pyramid scene parsing network”. In: *CVPR*. 2017.

[213] Zhun Zhong et al. “Random erasing data augmentation”. In: *AAAI*. 2020.

[214] Bolei Zhou et al. “Semantic understanding of scenes through the ADE20K dataset”. In: *IJCV*. 2019.

[215] Hao Zhou, Jose M Alvarez, and Fatih Porikli. “Less is more: Towards compact cnns”. In: *ECCV*. 2016.

[216] Michael Zhu and Suyog Gupta. “To prune, or not to prune: exploring the efficacy of pruning for model compression”. In: *ICLR Workshop*. 2018.

[217] Shlomo Zilberstein. “Using anytime algorithms in intelligent systems”. In: *AI magazine*. 1996.

[218] Barret Zoph and Quoc V Le. “Neural architecture search with reinforcement learning”. In: *ICLR*. 2017.