Fast and Accurate Model Scaling

Piotr Dollár   Mannat Singh   Ross Girshick

Facebook AI Research (FAIR)

Abstract

In this work we analyze strategies for convolutional neural network scaling; that is, the process of scaling a base convolutional network to endow it with greater computational complexity and consequently representational power. Example scaling strategies may include increasing model width, depth, resolution, etc. While various scaling strategies exist, their tradeoffs are not fully understood. Existing analysis typically focuses on the interplay of accuracy and flops (floating point operations). Yet, as we demonstrate, various scaling strategies affect model parameters, activations, and consequently actual runtime quite differently. In our experiments we show the surprising result that numerous scaling strategies yield networks with similar accuracy but with widely varying properties. This leads us to propose a simple fast compound scaling strategy that encourages primarily scaling model width, while scaling depth and resolution to a lesser extent. Unlike currently popular scaling strategies, which result in about $O(s)$ increase in model activation w.r.t. scaling flops by a factor of s, the proposed fast compound scaling results in close to $O(\sqrt{s})$ increase in activations, while achieving excellent accuracy. Fewer activations leads to speedups on modern memory-bandwidth limited hardware (e.g., GPUs). More generally, we hope this work provides a framework for analyzing scaling strategies under various computational constraints.

1. Introduction

Advances in modern hardware for training and running convolutional neural networks over the past several years have been impressive. Highly-parallel hardware accelerators, such as GPUs and TPUs, allow for training and deploying ever larger and more accurate networks.

Interestingly, this rapid advancement has greatly benefited our ability to optimize models for the *low-compute* regime. In particular, whether via manual design, random search, or more complex neural architecture search strategies [37], it has become feasible to train a large number of small models and select the best one, in terms of both accuracy *and* speed. At intermediate-compute regimes, efficient search [17] or efficient design spaces [22, 23] can still pro-
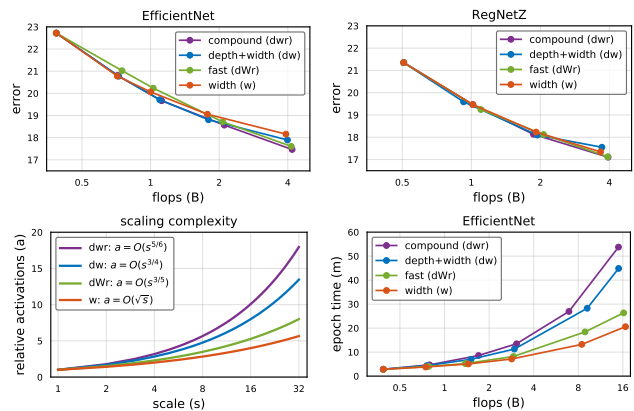

Figure 1. An analysis of four model scaling strategies: *width* scaling (w), in which only the width of a base model is scaled; *compound* scaling (dwr), in which the width, depth, and resolution are all scaled in roughly equal proportions; *depth and width* scaling (dw); and the proposed *fast compound* scaling (dWr), which emphasizes scaling primarily, but not only, the model width. *(Top):* We apply the four scaling strategies to two base models (EfficientNet-B0 and RegNetZ-500MF). Compound and fast scaling result in highest accuracy models, and both outperform width scaling. *(Bottom-left):* The scaling strategies have asymptotically different behavior in how they affect model activations. Given a scale factor of s, activations increase with about $O(\sqrt{s})$ for w and dWr scaling compared to almost $O(s)$ for dwr and dw scaling. *(Bottom-right):* Runtime of a model (EfficientNet-B0) scaled using the four scaling strategies. Fast scaling results in models nearly as fast as w scaling (but with higher accuracy), and much faster than dwr and dw scaling, closely reflecting model activations.

vide the ability to directly optimize neural networks. However, regardless of computational resources, there will necessarily exist a *high-compute* regime where it may only be feasible to train a handful of models, or possibly even only a *single* model. This regime motivates our work.

In the high-compute regime, *network scaling*, the process by which a lower-complexity model is enlarged by expanding one or more of its dimensions (*e.g.*, depth or width), becomes essential. Scaling has proven effective in terms of obtaining larger models with good accuracy [32]. However, existing work on model scaling focuses on model *accuracy*. In this work, we are interested in large, accurate models that are *fast* enough to deploy and use in practice.

The concept of network scaling emerged naturally in deep learning, with early work focused on scaling networks by increasing depth [27, 29, 10]. However, gains from depth scaling plateaued, leading to explorations of scaling width [35] and resolution [12]. More recently scaling multiple dimensions at once, coined *compound scaling* [32], has been shown to achieve excellent accuracy.

Existing explorations of model scaling typically focus on maximizing *accuracy* versus *flops*. Yet as we will show, two scaled models with the same flops can have very different runtime on modern accelerators. This leads us to the central question explored in our work: *can we design scaling strategies that optimize both accuracy and model runtime?*

Our first core observation is that there exists multiple scaling strategies that can yield similar accuracy models at the same flops. In Figure 1, top, we show that there exist multiple scaling strategies that can result in models with high accuracy. We will expand on this result in §6.

However, scaling a model to a fixed target flops using two scaling strategies can result in widely different runtimes, see Figure 1, bottom-right. To better understand this behavior at a more fundamental level, in §3 we develop a framework for analyzing the complexity of various scaling strategies, in terms of not just flops, but also parameters and activations. In particular, we show that different strategies scale activations at different asymptotic rates relative to flops. *E.g.*, when scaling a model from f flops to sf flops by scaling width, activations increase by $O(\sqrt{s})$, compared to nearly $O(s)$ for compound scaling. Figure 1, bottom-left, shows this asymptotic behavior for a few select strategies.

In §4 we will show that within a flop range of practical interest, on modern accelerators *the runtime of a scaled model is more strongly correlated with activations than flops*. We emphasize that this correlation holds over a diverse set of scaling strategies, which enables us to use activations as a proxy for predicting a scaled model's runtime.

Based on our analysis, in §5 we introduce a new family of scaling strategies parameterized by a single parameter α that controls the relative scaling along model width versus other dimensions. This lets us carefully control the asymptotic rate at which model activations scale. We show $0 \ll \alpha < 1$ yields models that are both fast and accurate. We refer to this scaling strategy as fast compound model scaling, or simply *fast scaling* for brevity.

As we will show in §6, fast scaling allows us to obtain large models that are as accurate as the state-of-the-art but faster. As a concrete example, we apply fast scaling to scale a RegNetY-4GF [23] model to 16GF (gigaflops), and find it uses less memory and is faster (and more accurate) than EfficientNet-B4 [32] – a model with $4\times$ fewer flops.

In order to facilitate future research we will release all code and pretrained models introduced in this work.[1]

---

[1] https://github.com/facebookresearch/pycls

2. Related Work

Manual network design. Since the impressive success of AlexNet [16], and with the steady progress of hardware accelerators, the community has pushed toward ever larger and more accurate models. Increasing model depth led to rapid gains, notable examples include VGG [27] and Inception [29, 30]. This trend culminated with the introduction of residual networks [10]. Next, wider models proved not only effective but particularly efficient [35, 12]. The use of depthwise [3] and group convolution [33] enabled even higher capacity models. Other notable design elements that led to larger and more accurate models include the inverted bottleneck [26], SE [13], and new nonlinearities [11, 24].

Automated network design. With the rapid advancement of hardware for training deep models, it has become more feasible to *automate* network design. Neural architecture search [37, 38, 25] has turned into a thriving research area and led to highly efficient models, especially in the low-compute regime. Model search is computationally expensive when training larger models, this has led to interest in developing efficient search algorithms [17, 21, 18]. For example, DARTS [18] proposed a differentiable search strategy that does not require training multiple separate models to optimize model structure. Nevertheless, in practice search is most effective in low or medium compute regimes.

Design space design. Despite the effectiveness of model search, the paradigm has limitations. The outcome of a search is a single model *instance* tuned to a specific setting (*e.g.*, dataset or flop regime). As an alternative, Radosavovic *et al.* [23] recently introduced the idea of *designing design spaces*, and designed a low-dimensional design space consisting of simple, easy-to-tune models. Given a new dataset or compute regime, a model can be selected from this design space by tuning a handful of parameters, allowing for highly efficient random search. This allows for optimizing models directly in fairly high-compute regimes. We utilize these efficient design spaces in our experiments.

Network scaling. Regardless of the model design strategy, there will exist some computational regime in which it is not feasible to train and compare a large number of models. Thus model scaling becomes crucial. Popular scaling strategies include scaling depth [27, 29, 10], width [35, 12] and resolution [12, 14]. The recently introduced compound scaling strategy [31], which scales along all three dimensions at once, achieves an excellent accuracy versus flops tradeoff and serves as a core baseline in our work.

Going bigger. There is substantial interest in scaling to massive datasets [28, 19] and compute regimes [14]. Moreover, recent progress in unsupervised learning [9, 2, 1] may create the potential to train with essentially unlimited data. These efforts motivate our work: we aim to enable scaling models to the size necessary for these brave new regimes.

dim	scaling			flops (f)	params (p)	acts (a)
none	d	w	r	dw^2r^2	dw^2	dwr^2
d	sd	w	r	sdw^2r^2	sdw^2	$sdwr^2$
w	d	$\sqrt{s}w$	r	sdw^2r^2	sdw^2	$\boxed{\sqrt{s}dwr^2}$
r	d	w	$\sqrt{s}r$	sdw^2r^2	$\boxed{1dw^2}$	$sdwr^2$

Table 1. **Simple scaling**: Complexity of scaling a stage (of d conv layers with width w and spatial resolution $r{\times}r$) by a factor of s using various simple scaling strategies where a single dimension is varied at a time. Note that parameters and activations vary substantially for the scaling strategies, especially for large s.

dims	scaling			flops (f)	params (p)	acts (a)
dw	$\sqrt{s}d$	$\sqrt[4]{s}w$	r	sdw^2r^2	sdw^2	$s^{3/4}dwr^2$
wr	d	$\sqrt[4]{s}w$	$\sqrt[4]{s}r$	sdw^2r^2	$\sqrt{s}dw^2$	$s^{3/4}dwr^2$
dr	$\sqrt{s}d$	w	$\sqrt[4]{s}r$	sdw^2r^2	$\sqrt{s}dw^2$	$sdwr^2$
dwr	$\sqrt[3]{s}d$	$\sqrt[6]{s}w$	$\sqrt[6]{s}r$	sdw^2r^2	$s^{2/3}dw^2$	$s^{5/6}dwr^2$

Table 2. **Compound scaling**: Complexity of compound scaling strategies with *uniform* scaling along each dimension, where the relative flops increase of scaling along each dimension is equal. Scaling uniformly along all dimensions, which closely resembles compound scaling [32], results in near linear scaling of activations.

3. Complexity of Scaled Models

In this section we present a general framework for analyzing the complexity of various network scaling strategies. While the framework is simple and intuitive, it proves powerful in understanding and extending model scaling.

3.1. Complexity Metrics

The three most relevant properties of models we consider are their *flops (f)*, *parameters (p)*, and *activations (a)*. Following common practice, we use flops to mean multiply-adds and parameters to denote the number of free variables in a model. We define activations as the number of elements in the output tensors of convolutional (conv) layers.

Flops and parameters are popular complexity measures of neural networks. We note, however, that parameters of a convolution are independent of input resolution and hence do not fully reflect the actual capacity or runtime of a convolutional network. Therefore, given that we study networks with varying input resolution, we report parameters but *we focus on flops as a primary complexity measure*.

Activations are less often reported but as we demonstrate play a key role in determining network speed on modern memory-bandwidth limited hardware. Hence, we carefully analyze the interplay between scaling and activations.

3.2. Network Complexity

While conv networks are composed of many heterogeneous layers, we focus our complexity analysis on conv layers. First, many layers such as normalization, pooling, or activation often account for a small percentage of a model's compute. Second, the number and complexity of these layers tends to be *proportional* to the number and size of conv layers (*e.g.*, every conv may be followed by an activation). For these reasons analyzing convs serves as an excellent proxy of how model scaling affects an entire network.

Consider a $k{\times}k$ conv layer with width (number of channels) w and spatial resolution r. The layer takes in a feature map of size $r{\times}r{\times}w$, and for each of the r^2 patches of size $k{\times}k{\times}w$ the network applies w dot products of size wk^2. Therefore the complexity of a conv layer is given by:

$$f = w^2r^2k^2, \quad p = k^2w^2, \quad a = wr^2 \quad (1)$$

As k is not scaled, we let $k = 1$ without loss of generality.

Common networks are composed of *stages*, where each stage consists of d uniform conv layers, each with the same w and r. The complexity of a stage of depth d is:

$$f = dw^2r^2, \quad p = dw^2, \quad a = dwr^2 \quad (2)$$

In subsequent analysis we will show how different scaling strategies affect the complexity of a single stage. For simplicity, we use the same scaling for each network stage, thus our complexity analysis applies to the entire network.

3.3. Complexity of Simple Scaling

We define *simple scaling* of a stage as scaling a stage along a *single* dimension. In particular, we consider width (w), depth (d), and resolution (r) scaling. In addition to the *scaling dimension*, we define the *scaling factor s* to be the amount by which scaling increases model *flops*. Increasing d by s, w by $\sqrt{s}$, or r by $\sqrt{s}$ all increase flops by s (for simplicity we ignore quantization effects).

Table 1 shows the complexity of scaling a stage by a factor of s along different scaling dimensions. While in each case the resulting flops are the same (by design), the parameters and activations vary. In particular, activations increase by $\sqrt{s}$ when scaling width compared to by s when scaling along resolution or depth. This observation will play a central role in how we design new scaling strategies.

3.4. Complexity of Compound Scaling

Rather than scaling along a single dimension, an intuitive approach is to scale along multiple dimensions at once. Coined *compound scaling* by [32], such an approach has been shown to achieve higher accuracy than simple scaling.

In Table 2 we show the complexity for scaling along either two or three dimensions. In each case, we select ratios such that scaling is *uniform* w.r.t. flops along each dimension. *E.g.*, if scaling along all dimensions (dwr), we scale d by $\sqrt[3]{s}$, w by $\sqrt[6]{s}$, and r by $\sqrt[6]{s}$, such that flops increase by $\sqrt[3]{s}$ when scaling each dimension and by $\sqrt[3]{s}^3 = s$ in total.

Interestingly, the compound scaling rule discovered empirically in [32] scaled by 1.2, 1.1, and 1.15 along d, w, and r, which corresponds roughly to *uniform compound scaling* with $s = 2$ ($\sqrt[3]{s} \approx 1.26$, $\sqrt[6]{s} \approx 1.12$). We thus use uniform compound scaling as a simple proxy for the purpose of our analysis. Observe that for uniform compound scaling, activations increase nearly *linearly* with s.

dims	scaling		flops (f)	params (p)	acts (a)
none	w	g	wgr^2	wg	wr^2
w	sw	g	$swgr^2$	swg	swr^2
g	w	sg	$swgr^2$	swg	$1wr^2$
wg	$\sqrt{s}w$	$\sqrt{s}g$	$swgr^2$	swg	$\sqrt{s}wr^2$

Table 3. **Group width scaling**: Complexity of scaling a *group* conv by scaling only width w, only group width g, or both (we assume $g \le w$). Scaling only g does not impact activations; scaling uniformly in both w and g results in $\sqrt{s}$ increase in activations. Unless noted, we scale g proportionally to w for group conv.

3.5. Complexity of Group Width Scaling

Many top-performing networks rely heavily on *group conv* and *depthwise conv*. A group conv with channel width w and *group width g* is equivalent to splitting the w channels into w/g groups each of width g, applying a regular conv to each group, and concatenating the results. Depthwise conv is a special case with $g = 1$. Therefore, its complexity is:

$$f = wgr^2, \quad p = wg, \quad a = wr^2. \tag{3}$$

In Table 3 we show three basic strategies for scaling group conv. We observe that to obtain scaling behavior similar to scaling regular conv, both channel width and group width must be scaled. Therefore, unless otherwise noted, we scale g proportionally to w. For networks that use depthwise conv ($g = 1$), as in previous work [32], we do not scale g.

Finally, we note that when scaling g, we must ensure w is divisible by g. To address this, we set $g = w$ if $g > w$ and round w to be divisible by g otherwise (w will change by at most 1/3 under such a strategy [23]).

4. Runtime of Scaled Models

Our motivation is to design scaling strategies that result in *fast* and *accurate* models. In §3 we analyzed the behavior of flops, parameters, and activations for various scaling strategies. In this section we examine the relationship between these complexity metrics and model runtime. This will allow us to design new *fast* scaling strategies in §5.

How are the complexity metrics we analyzed in §3 related to model runtime on modern accelerators? To answer this question, in Figure 2 we report runtime for a large number of models scaled from three base models as a function of flops, parameters, and activations. From these plots we can make two observations: flops and parameters are only *weakly* predictive of runtime when scaling a single model via different scaling strategies; however, activations are *strongly* predictive of runtime for a model regardless of the scaling strategy. See Figure 2 for additional details.

This simple result leads us to *use model activations as a proxy for runtime*. Specifically, for scaled versions of a single model, the Pearson correlation between runtime and activations is $r \ge 0.99$, regardless of the scaling strategy, while correlation with flops and parameters is far lower (r
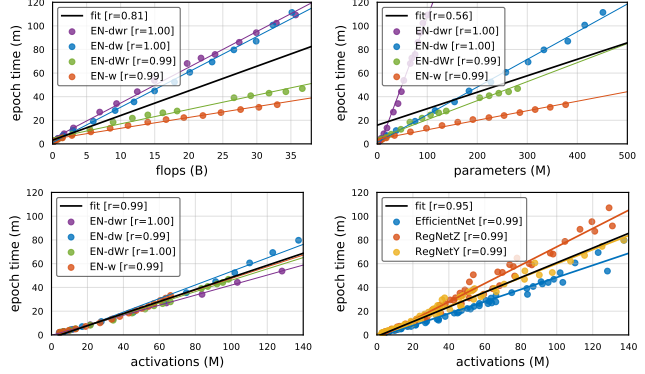

Figure 2. **Model runtime** as a function of various complexity metrics. *(Top-left):* We scale EfficientNet-B0 (EN-B0) using four scaling strategies (dwr, dw, dWr, w) with a wide range of scaling factors ($s < 100$). For each scaling strategy we plot epoch time versus flops for each model (along with a best fit line). For a single scaling strategy (*e.g.*, w), runtime is highly correlated with flops (*e.g.*, Pearson's $r = 0.99$). However, when comparing scaled versions of the same model using different scaling strategies, flops are only *weakly* predictive of runtime ($r = 0.81$). *(Top-right):* Using the same set of models, we plot runtime versus parameters, and again observe parameters are even more weakly correlated with runtime ($r = 0.56$). *(Bottom-left):* Repeating the same analysis for runtime versus activations, we see that activations are *strongly* predictive of runtime regardless of the scaling strategy ($r = 0.99$). *(Bottom-right):* We repeat the analysis of runtime versus activations for three models (see §6.1 for model details). For scaled versions of each model, activations are highly predictive of runtime ($r \ge 0.99$), and only very large models tend to be flop bound. This makes activations an excellent proxy for runtime. We note, however, that activations are less predictive of runtime when comparing scaled versions of different models ($r = 0.95$).

of 0.81 and 0.56, respectively). We caution, however, that activations cannot perfectly predict runtime across heterogeneous models ($r = 0.95$), as models may use operations with different runtimes, *e.g.* ReLU *vs.* SiLU. Moreover, some big models have runtimes higher than predicted from their activations indicating these models are *flop bound*.

Implementation details. We report the time to perform *one epoch of training* on ImageNet [6] which contains ~1.2M training images. For each model, we use the largest batch size that fits in memory. We note that inference time is highly correlated with training time, but we report epoch time as it is easy to interpret (inference performance depends heavily on the use case). We time all models using PyTorch and 8 32GB Volta GPUs. Runtime is of course hardware dependent; however, we believe timing on GPUs is reasonable for two reasons. First, hardware accelerators (such as GPUs, TPUs, *etc.*) are highly prevalent. Second, accelerators are extremely efficient in terms of compute but tend to be memory-bandwidth bound [34], and this trend is expected to become more pronounced.

dims	α	e_d	e_w	e_r	f	p	a
	α	$\frac{1-\alpha}{2}$	α	$\frac{1-\alpha}{2}$	s	$s^{\frac{1+\alpha}{2}}$	$s^{\frac{2-\alpha}{2}}$
dr	0	0.5	0	0.5	s	$s^{0.50}$	$s^{1.00}$
dwr	1/3	1/3	1/3	1/3	s	$s^{0.67}$	$s^{0.83}$
dWr	0.8	0.1	0.8	0.1	s	$s^{0.90}$	$s^{0.60}$
w	1	0	1	0	s	$s^{1.00}$	$s^{0.50}$

Table 4. **Fast scaling.** We introduce a *family* of scaling strategies parameterized by α. The top row shows scaling factor exponents e_d, e_w, and e_r as a function of α, and the relative increase in model complexity as a function of α and scaling factor s. The remaining rows show instantiations of the scaling strategy for various α. $\alpha = 1$ corresponds to width (w) scaling, and $\alpha = 1/3$ corresponds to uniform compound scaling (dwr). The new regime we explore in this work is $1/3 < \alpha < 1$. In particular, using α near 1 results in fewer activations and thus faster networks (see Figure 2). In our experiments, we find $\alpha = 0.8$ results in an excellent tradeoff between speed and accuracy. We use dWr to denote fast scaling to emphasize scaling is primarily, but not only, along w.

5. Fast Compound Model Scaling

Given the strong dependency of runtime on activations, we aim to design scaling strategies that minimize the increase in model activations. As our results from Tables 1-3 indicate, of all scaling strategies that involve scaling width, depth, and resolution, scaling a network by increasing its *channel width* and *group width* results in the smallest increase in activations. Indeed, it is well known that *wide* networks are quite efficient in wall-clock time [35]. Unfortunately, wide networks may not always achieve top results compared to deeper or higher-resolution models [10, 32].

To address this, in this work we introduce the concept of *fast compound model scaling*, or simply *fast scaling* for brevity. The idea is simple: we design and test scaling strategies that primary increase model width, but also increase depth and resolution to a lesser extent.

We formalize this by introducing a *family* of scaling strategies parameterized by α. Given α we define:

$$e_d = \frac{1-\alpha}{2}, \quad e_w = \alpha, \quad e_r = \frac{1-\alpha}{2}, \qquad (4)$$

and when scaling a network by a factor of s, we set:

$$d' = s^{e_d}d, \quad w' = \sqrt{s}^{e_w}w, \quad r' = \sqrt{s}^{e_r}r. \qquad (5)$$

If using group conv, we also set $g' = \sqrt{s}^{e_w}g$ (same scaling as for w). The resulting complexity of the scaled model is:

$$f = sdw^2r^2, \quad p = s^{\frac{1+\alpha}{2}}dw^2, \quad a = s^{\frac{2-\alpha}{2}}dwr^2. \qquad (6)$$

Instantiations for scaling strategies using various α are shown in Table 4. Setting $\alpha = 1$ results in width (w) scaling (lowest activations). Setting $\alpha = 0$ results in depth and resolution (dr) scaling (highest activations). $\alpha = 1/3$ corresponds to uniform compound scaling (dwr).

The interesting new regime we explore is $1/3 < \alpha < 1$. In particular, we refer to scaling strategies with α near 1 as fast scaling. Unless specified, we use $\alpha = 0.8$ by default, which we denote using dWr. Next, in §6 we show that fast scaling results in good speed *and* accuracy.

6. Experiments

We next evaluate the effectiveness of our proposed fast scaling strategy. We introduce the baseline networks we test along with optimization settings in §6.1. In §6.2, we evaluate existing scaling strategies, then we perform extensive experiments and comparisons of fast scaling in §6.3. Finally we compare scaling *vs.* random search in §6.4 and compare larger models in §6.5. Additional experiments are presented in the appendix in the extended version of this work [7].

6.1. Baselines and Optimization Settings

Baseline networks. In this work we evaluate scaling strategies on three networks families: EfficientNet [32], RegNetY [23], and RegNetZ (described below). We chose these models as they are representative of the state-of-the-art and are well suited for our scaling experiments. Moreover, EfficientNet was introduced in the context of model scaling work [32], making it an excellent candidate for our study.

EfficientNet. EfficientNets have been shown to achieve a good flop-to-accuracy tradeoff. These models use inverted bottlenecks [26], depthwise conv, and the SiLU nonlinearity [11] (also popularly known as Swish [24]). An EfficientNet is composed of seven stages with varying width, depth, stride and kernel size. The original model (EfficientNet-B0) was optimized in the mobile regime (400MF) using neural architecture search [31] and scaled to larger sizes (B1-B7) via compound scaling. For further details, please see [32].

Note that EfficientNets are specified by ~30 parameters (input resolution, 7 stages with 4 parameters each, and stem and head width). Given this high-dimensional search space, optimizing an EfficientNet is only feasible in a low-compute regime, and scaling must be used to obtain larger models.

RegNets. As an alternative to neural architecture search, Radosavovic *et al.* [23] introduced the idea of *designing design spaces*, where a design space is a parameterized population of models. Using this methodology, [23] designed a design space consisting of simple, regular networks called RegNets that are effective across a wide range of block types and flop regimes. Importantly for our work, a RegNet model is specified by a handful of parameters (~6), which then allows for fast model selection using random search. Thus, unlike EfficientNets, RegNets allow us to compare large models obtained either via *scaling* or *random search*.

A RegNet consists of a stem, a body with four stages, and a head. Each stage consists of a sequence of identical blocks. The block type can vary depending on the model (the two block types we use are shown in Figure 3). Importantly, the widths and depths of a RegNet are not specified independently per stage, but are determined by a *quantized linear function* which has 4 parameters (d, w_0, w_a, w_m), for details see [23]. Any other block parameters (like group width or bottleneck ratio) are kept constant across stages.

(a) Y block, s=2   **(b)** Y block, s=1   **(c)** Z block, s=2   **(d)** Z block, s=1
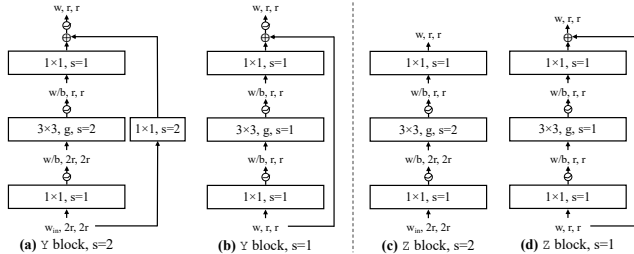
Figure 3. **RegNet blocks.** Each stage consists of a stride $s = 2$ block that halves r and increases w followed by multiple stride $s = 1$ blocks with constant r and w. (a-b) The Y block is based on *residual bottlenecks* with group conv [33]. Each block consists of a 1×1 conv, a 3×3 group conv, and a final 1×1 conv. The 1×1 convs can change w via the bottleneck ratio b, however, we set $b = 1$ following [23]. BatchNorm [15] and ReLU follow each conv. (c-d) We introduce the Z block based on *inverted bottlenecks* [26]. The Z block is similar to the Y block with 4 differences: no non-linearity follows the final 1×1 conv, (2) SiLU [11] is used in place of ReLU, (3) the stride 2 variant of the block has no residual, and (4) $b < 1$ (we use $b = 1/4$ in all experiments). Finally, a Squeeze-and-Excitation (SE) op [13] (reduction ratio of $1/4$) follows the 3×3 conv for both the Y and Z blocks (not shown).

RegNetY. The RegNetY block (Y) is shown in Figure 3 (a-b). The Y block resembles the standard *residual bottleneck* block with group conv [33]. Additionally it uses a Squeeze-and-Excitation (SE) layer [13]. Following [23], we set the bottleneck ratio b to 1 (effectively no bottleneck). A Reg-NetY model is thus fully specified with 5 parameters: d, w_0, w_a, w_m, and g. Unlike [23], we additionally vary the image input resolution r (bringing the total parameters to 6).

RegNetZ. We introduce a new Z block based on *inverted bottlenecks* [26]. The Z block resembles the Y block except it omits the last nonlinearity and inverts the bottleneck (we use $b = 1/4$ in all experiments). See Figure 3 (c-d) for additional details. A RegNetZ model, built using the Z block, is fully specified with the same 6 parameters as a RegNetY model. We note that EfficientNet also uses inverted bottlenecks, but we introduce RegNetZ to allow us to compare large models obtained via scaling *and* random search.

Optimization settings. Our goal is to enable *fair* and *reproducible* results. However, we also aim to achieve *state-of-the-art* results. This creates a tension between using a simple yet weak optimization setup (*e.g.*, [23]) versus a strong setup that yields good results but may be difficult to reproduce (*e.g.*, [32]). To address this, we use a training setup that effectively balances between these two objectives.

Our setup is as follows: we use SGD with a momentum of 0.9, label smoothing with $\epsilon = 0.1$ [30], mixup with $\alpha = 0.2$ [36], AutoAugment [4], stochastic weight averaging (SWA) [5], and mixed precision training [20]. For all models we use 5 epochs of gradual warmup [8]. We use an exponential learning rate schedule with a batch size of

	flops (B)	params (M)	acts (M)	time (min)	publication ICML	publication arXiv	schedule 1×	schedule 2×	schedule 4×
EN-B0	0.4	5.3	6.7	2.8	23.7	22.7	23.6±0.09	22.7±0.08	**22.3**±0.04
EN-B1	0.7	7.8	10.9	4.6	21.2	20.8	21.7±0.18	20.8±0.10	**20.5**±0.09
EN-B2	1.0	9.1	13.8	5.9	20.2	19.7	20.7±0.06	20.0±0.12	**19.6**±0.09
EN-B3	1.8	12.2	23.8	9.5	18.9	**18.3**	19.4±0.07	18.8±0.10	**18.3**±0.11
EN-B4	4.4	19.3	49.5	19.2	17.4	**17.0**	18.0±0.05	17.4±0.07	17.3±0.06
EN-B5	10.3	30.4	98.9	40.8	16.7	**16.3**	17.1±0.13	16.7±0.05	–

Table 5. **EfficientNet reproduction.** The first set of results includes the originally reported errors (from ICML [32] and updated numbers later reported on arXiv), the second set our reproduction under three schedule lengths ($1\times$ corresponds to 100 epochs), averaged over 3 trials. Our results match or outperform the originally reported results (for the biggest nets they slightly lag the updated arXiv errors). We emphasize that unlike the results in [32], we use the same, easy to reproduce optimization setup for all models.

	flops (B)	params (M)	acts (M)	time (min)	schedule 1×	schedule 2×	schedule 4×
RegNetY-500MF	0.5	5.6	4.2	2.3	24.8±0.07	23.9±0.14	23.2±0.05
RegNetZ-500MF	0.5	7.1	5.9	3.1	22.2±0.04	21.3±0.02	21.0±0.08
RegNetY-4GF-224	4.0	20.6	12.3	6.4	19.4±0.07	18.4±0.05	18.1±0.07
RegNetY-4GF	4.1	22.4	14.5	7.7	18.8±0.04	18.0±0.07	17.7±0.09
RegNetZ-4GF-224	4.0	26.9	20.8	11.3	17.7±0.04	17.2±0.06	17.0±0.04
RegNetZ-4GF	4.0	28.1	24.3	11.7	17.5±0.09	17.0±0.12	16.9±0.04

Table 6. **RegNet Baselines**. The two 500MF RegNet models use a default resolution of 224. RegNetY-4GF-224 uses the default 224 resolution; RegNetY-4GF uses a resolution found by random search. Likewise there are two versions of RegNetZ-4GF with default and discovered resolutions.

1024 (distributed on 8 32GB GPUs), learning rate $\lambda = 2.0$, and decay $\beta = 0.02$.[2] For RegNets we use a weight decay of 2e-5 and for EfficientNets we use 1e-5. Batch norm parameters are not decayed. For large models we reduce the batch size and learning rate proportionally as in [8]. For reproducibility, we will release code for our setup.

EfficientNet baselines. In Table 5, we report Efficient-Net results using our optimization setup versus results from [32]. We report our results using a '$1\times$', '$2\times$', or '$4\times$' schedule (corresponding to 100, 200, and 400 epochs, respectively). Our $2\times$ schedule achieves competitive results, our $4\times$ schedule outperforms the originally reported results for all but the largest model tested. We use the $2\times$ schedule in all following experiments unless otherwise noted.

RegNet baselines. In Table 6 we report results for baseline RegNet models. We obtain these models via random search as in [23].[3] Note that there are two versions of the 4GF RegNets (using default and discovered resolutions).

---

[2]We parameterize the exponential learning rate via $\lambda_t = \lambda \beta^{\frac{t}{T}}$, where t is the current epoch, T the final epoch, λ is the initial learning rate, and $\lambda\beta$ is the final learning rate. We use this parameterization (as opposed to $\lambda_t = \lambda\gamma^t$) as it allows us to use a single setting for the decay β regardless of the schedule length T (setting $\gamma = \beta^{1/T}$ makes the two equivalent).

[3]We sample RegNet model configurations until we obtain 32 models in a given flop regime, train each of these model using the $1\times$ schedule, and finally select the best one. Sampling just 32 random models in a given flop regime is typically sufficient to obtain accurate models as shown in [23].
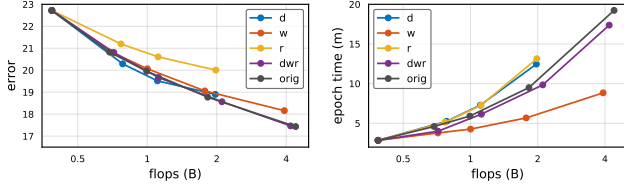
Figure 4. **Compound scaling: EfficientNet.** (Left) Uniform compound scaling (*dwr*) offers the best accuracy relative to simple scaling along depth (*d*), width (*w*), or resolution (*r*). All models are scaled from EfficientNet-B0 (400MF) up to at most 4GF. (Right) Models obtained with *w* scaling are much faster than those from *dwr* scaling. Both of these results are expected. However, as we will show, it is possible to obtain models that are both fast *and* accurate. For reference, we also show the original EfficientNet models (orig) obtained via non-uniform compound scaling [32], the results closely match uniform compound scaling (*dwr*).
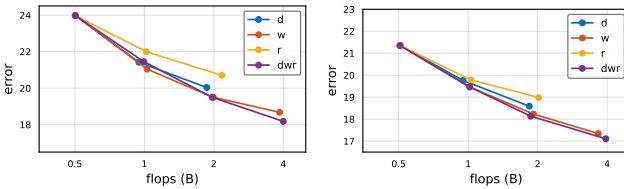

Figure 5. **Compound scaling: RegNet.** We apply simple and compound scaling to RegNetY-500MF (left) and RegNetZ-500MF (right). As in Figure 4, *dwr* scaling achieves the best error, but at significant increase in runtime (see appendix) relative to *w* scaling.

6.2. Simple and Compound Scaling

We now turn to evaluation of simple and compound scaling [32] described in §3.3 and §3.4, respectively. For these experiments we scale the baseline models from §6.1.

In Figure 4, we evaluate the accuracy (left) and runtime (right) of EfficientNet-B0 scaled either via simple scaling along width (*w*), depth (*d*), or resolution (*r*) or via uniform compound scaling (*dwr*). As expected, *dwr* scaling provides the best accuracy, but results in slower models than *w* scaling. This suggests a tradeoff between speed and accuracy, but as we will show shortly, this need not be the case. Finally, we tested uniform scaling along pairs of dimensions (see Table 2), but *dwr* scaling proved best (not shown).

We also compare uniform compound scaling (*dwr*) to the original compound scaling rule (orig) from [32], which empirically set the per-dimension scalings factors. As expected from our analysis in §3.4, *dwr* scaling is close in both accuracy and runtime to the original compound scaling rule without the need to optimize individual scaling factors.

In Figure 5 we repeat the same experiment but for the RegNetY-500MF and RegNetZ-500MF baselines. We see a similar behavior, where *dwr* scaling achieves the strongest results. Runtimes (see appendix [7]) exhibit very similar behaviors (*w* scaling is much faster). Note that as discussed, group width *g* is scaled proportionally to width *w*.
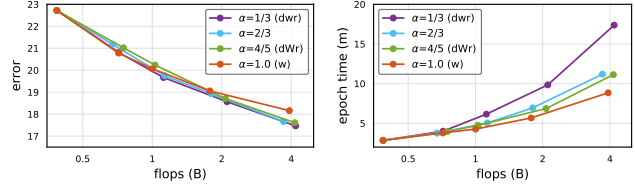

Figure 6. **Fast scaling: EfficientNet.** We test scaling EfficientNet-B0 using our family of scaling strategies parameterized by α (see Table 4). (Left) Scaling with any $\alpha < 1$ achieves good accuracy and results in a sizable gap in error to scaling with $\alpha = 1$ (*w*). The exact value of $\alpha < 1$ does not greatly influence the error. (Right) While all scaling strategies with $\alpha < 1$ give good accuracy, their runtime differs substantially. A setting of $\alpha = 4/5$ (*dWr*) gives the best of both worlds: models that are both fast *and* accurate.
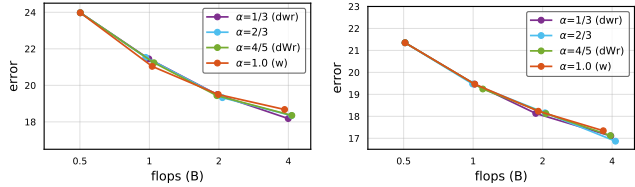

Figure 7. **Fast scaling: RegNet.** We apply scaling with different α to RegNetY-500MF (left) and RegNetZ-500MF (right). As in Figure 6, *dWr* scaling yields good accuracy *and* speed (see appendix for rutnimes). We note that α may potentially be be further tuned to tradeoff speed and accuracy, but we use $\alpha = 4/5$ in this work.

6.3. Fast Scaling

We now perform an empirical analysis of the *effectiveness* of our fast scaling strategy. Recall that in §5 we introduced a family of scaling strategies parameterized by α that interpolates between uniform compound scaling (*dwr*) when $\alpha = 1/3$ to width scaling (*w*) when $\alpha = 1$. As α goes toward 1, the model activations increase least as we scale a model, resulting in faster models. In particular, we define $\alpha = 4/5$ as *fast scaling*, and denote it by *dWr*.

In Figure 6, we evaluate the accuracy (left) and runtime (right) of EfficientNet-B0 scaled with various settings of α. Interestingly, for all tested values of $\alpha < 1$ model accuracy was quite similar and substantially higher than for *w* scaling ($\alpha = 1$), especially for larger models. In terms of runtime, *dWr* scaling is nearly as fast as *w* scaling, and substantially faster than *dwr* scaling. We emphasize that the differences in memory and speed increase *asymptotically*, hence the difference in runtime for models scaled with different α becomes more pronounced at larger scales.

In Figure 7 we repeat the same experiment but for the RegNet baselines. Results are similar, *dWr* scaling ($\alpha = 4/5$) achieves excellent accuracy and runtime. Finally, we observe that for RegNets, *w* scaling is more effective than for EfficientNet. This can be partially explained as for RegNets we scale the group width *g* along width *w* (EfficientNet always uses *g* = 1), indeed setting *g* = 1 and scaling RegNets by just *w* performs worse (see appendix).

	flops (B)	params (M)	acts (M)	time (min)	schedule 1×	2×	4×
RegNetY-500MF→4GF	4.1	36.2	13.3	7.2	$19.1_{\pm0.07}$	$18.6_{\pm0.09}$	$18.3_{\pm0.06}$
RegNetY-4GF [optimized]	4.1	22.4	14.5	7.7	$18.8_{\pm0.04}$	$18.0_{\pm0.07}$	$17.7_{\pm0.09}$
RegNetZ-500MF→4GF	4.0	41.1	19.4	10.5	$17.7_{\pm0.07}$	$17.2_{\pm0.07}$	$17.0_{\pm0.05}$
RegNetZ-4GF [optimized]	4.0	28.1	24.3	11.7	$17.5_{\pm0.09}$	$17.0_{\pm0.12}$	$16.9_{\pm0.04}$
RegNetY-500MF→16GF	16.2	112.7	29.4	17.8	$17.8_{\pm0.18}$	$17.2_{\pm0.06}$	$16.9_{\pm0.10}$
RegNetY-4GF→16GF	15.5	72.3	30.7	16.4	$17.3_{\pm0.09}$	$16.8_{\pm0.11}$	$16.6_{\pm0.03}$

Table 7. **Scaling *vs*. Search**. Models optimized for a given flop regime (via random search) outperform scaled models (rows 1-4). Nevertheless, scaling is necessary in flop regimes where optimization is computationally prohibitive. A *hybrid* approach is to optimize a model in an intermediate regime (*e.g.* 4GF) prior to scaling to a higher flop regime (*e.g.* 16GF), as in rows 5-6.
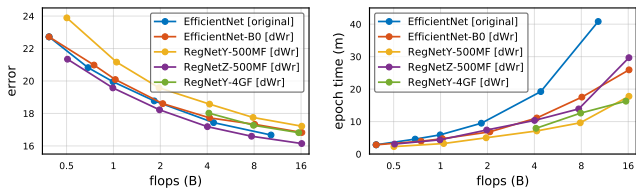

Figure 8. **Large models.** We scale four models via fast scaling (dWr) up to 16GF (1× to 32× scaling). We include the original EfficientNet model for reference. All results use our 2× schedule. See §6.5 for details and discussion.

6.4. Scaling versus Search

How do *scaled* models compare to models obtained via *random search*? Recall that RegNets only have 6 free parameters, so optimizing a RegNet directly by random search in an intermediate flop regime is feasible (see §6.1).

Table 7 compares three sets of models. First, we compare RegNetY at 4GF obtained either via dWr scaling (denoted by RegNetY-500MF→4GF) or search (RegNetY-4GF) in rows 1-2. The best sampled model outperforms the scaled model by 0.6% with a 4× schedule. We repeat this analysis for RegNetZ (rows 3-4) and find the best sampled model outperforms the scaled model by 0.1%. These results indicate that scaling a high-accuracy model is not guaranteed to yield an optimal model. Nevertheless, scaling is often necessary for targeting high compute regimes where model optimization is not feasible.

The above results suggest a *hybrid scaling strategy*, in which we optimize a model at an intermediate flop regime prior to scaling the model to larger scales. In Table 7, rows 5-6, we compare two 16GF RegNetY models, one scaled by 32× from a 500MF model and one scaled 4× from an optimized 4GF model. The model obtained with the hybrid strategy of scaling an intermediate model is 0.3% better.

Finally, observe that the best sampled models have far fewer parameters than the scaled models. We found that at higher flop regimes, optimized models have fewer blocks in the last stage, which greatly reduces their parameters. This shows a limitation of uniformly scaling model stages without redistributing blocks across stages.

	flops (B)	params (M)	acts (M)	time (min)	schedule 1×	2×	4×
ResNet50 [10]	4.1	25.6	11.3	3.5	$22.0_{\pm0.12}$	$21.0_{\pm0.08}$	$20.5_{\pm0.07}$
ResNeXt50 [33]	4.2	25.0	14.6	5.8	$20.8_{\pm0.06}$	$19.9_{\pm0.16}$	$19.5_{\pm0.05}$
EfficientNet-B4 [32]	4.4	19.3	49.5	19.2	$18.0_{\pm0.05}$	$17.4_{\pm0.07}$	$17.3_{\pm0.06}$
RegNetY-4GF	4.1	22.4	14.5	7.7	$18.8_{\pm0.04}$	$18.0_{\pm0.07}$	$17.7_{\pm0.09}$
RegNetZ-4GF	4.0	28.1	24.3	11.7	$17.5_{\pm0.09}$	$17.0_{\pm0.12}$	$16.9_{\pm0.04}$
EfficientNet-B0→4GF	4.1	36.1	29.2	11.1	$18.4_{\pm0.11}$	$17.7_{\pm0.07}$	$17.4_{\pm0.11}$
RegNetY-500MF→4GF	4.1	36.2	13.3	7.2	$19.1_{\pm0.07}$	$18.6_{\pm0.09}$	$18.3_{\pm0.06}$
RegNetZ-500MF→4GF	4.0	41.1	19.4	10.5	$17.7_{\pm0.07}$	$17.2_{\pm0.07}$	$17.0_{\pm0.05}$
EfficientNet-B0→16GF	16.2	122.8	61.8	25.8	$17.4_{\pm0.08}$	$16.8_{\pm0.09}$	–
RegNetY-500MF→16GF	16.2	112.7	29.4	17.8	$17.8_{\pm0.18}$	$17.2_{\pm0.06}$	$16.9_{\pm0.10}$
RegNetY-4GF→16GF	15.5	72.3	30.7	16.4	$17.3_{\pm0.09}$	$16.8_{\pm0.11}$	$16.6_{\pm0.03}$
RegNetZ-500MF→16GF	16.2	134.8	42.6	29.4	$16.6_{\pm0.04}$	$16.1_{\pm0.06}$	$16.1_{\pm0.07}$
RegNetZ-4GF→16GF	15.9	95.3	51.3	33.2	$16.5_{\pm0.05}$	$16.0_{\pm0.10}$	$16.0_{\pm0.05}$

Table 8. **Large models.** For reference and reproducibility, we list details of our scaled 4GF and 16GF models models trained using our 1×, 2×, and 4× schedules. For reference, we also retrain ResNet50 [10] and ResNeXt50 [33] using our strong setup (and obtain an ~3% reduced error than originally reported).

6.5. Comparison of Large Models

The primary benefit of model scaling is it allows us to scale to larger models where optimization is not feasible. In Figure 8, we scale four models up to 16GF using fast scaling. We make the following observations:

1. Model ranking is *consistent* across flop regimes, with scaled versions RegNetZ achieving the best accuracy.
2. All models obtained via fast scaling (dWr) are asymptotically faster than the original EfficientNet models, including our scaled versions of EfficientNet-B0.
3. The *gap* between the highest and lowest error models (RegNetY and RegNetZ) *shrinks* from 2.2% at 500MF to 0.8% at 16GF, implying that on ImageNet model optimization may be less important at high flop regimes.
4. The *hybrid* approach of scaling an intermediate flop regime model to higher flops (4GF→16GF) closes much of the gap between RegNetY and RegNetZ.
5. RegNetY is the fastest model tested and a good choice if runtime is constrained, especially at higher flops.

In Table 8 we give further details of the 4GF and 16GF models we tested, along with additional baselines. We note that RegNetY-4GF→16GF uses less memory and is *faster* than EfficientNet-B4, even though this RegNetY model has ~4× as many flops. This emphasizes the importance of looking at metrics beyond flops when comparing models.

7. Discussion

In this work we presented a general framework for analyzing model scaling strategies that takes into account not just flops but also other network properties, including activations, which we showed are highly correlated with runtime on modern hardware. Given our analysis, we presented a fast scaling strategy that primarily, but not exclusively, scales model width. Fast scaling results in accurate models that also have fast runtime. While the optimal scaling approach may be task dependent, we hope our work provides a general framework for reasoning about model scaling.

References

[1] Mathilde Caron, Ishan Misra, Julien Mairal, Priya Goyal, Piotr Bojanowski, and Armand Joulin. Unsupervised learning of visual features by contrasting cluster assignments. In *NeurIPS*, 2020. 2

[2] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. *arXiv:2002.05709*, 2020. 2

[3] François Chollet. Xception: Deep learning with depthwise separable convolutions. In *CVPR*, 2017. 2

[4] Ekin D Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V Le. AutoAugment: Learning augmentation policies from data. *arXiv:1805.09501*, 2018. 6

[5] Xiaoliang Dai, Alvin Wan, Peizhao Zhang, Bichen Wu, Zijian He, Zhen Wei, Kan Chen, Yuandong Tian, Matthew Yu, Peter Vajda, and Joseph E. Gonzalez. FBNetV3: Joint architecture-recipe search using neural acquisition function. *arXiv:2006.02049*, 2020. 6

[6] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *CVPR*, 2009. 4

[7] Piotr Dollár, Mannat Singh, and Ross Girshick. Fast and accurate model scaling. *arXiv:2103.06877*, 2021. 5, 7

[8] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training ImageNet in 1 hour. *arXiv:1706.02677*, 2017. 6

[9] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. In *CVPR*, 2020. 2

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 2, 5, 8

[11] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (GELUs). *arXiv:1606.08415*, 2016. 2, 5, 6

[12] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv:1704.04861*, 2017. 2

[13] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *CVPR*, 2018. 2, 6

[14] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V Le, Yonghui Wu, et al. Gpipe: Efficient training of giant neural networks using pipeline parallelism. In *NeurIPS*, 2019. 2

[15] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015. 6

[16] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet classification with deep convolutional neural networks. In *NeurIPS*, 2012. 2

[17] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *ECCV*, 2018. 1, 2

[18] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. In *ICLR*, 2019. 2

[19] Dhruv Mahajan, Ross Girshick, Vignesh Ramanathan, Kaiming He, Manohar Paluri, Yixuan Li, Ashwin Bharambe, and Laurens van der Maaten. Exploring the limits of weakly supervised pretraining. In *ECCV*, 2018. 2

[20] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, and Hao Wu. Mixed precision training. In *ICLR*, 2018. 6

[21] Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. In *ICML*, 2018. 2

[22] Ilija Radosavovic, Justin Johnson, Saining Xie, Wan-Yen Lo, and Piotr Dollár. On network design spaces for visual recognition. In *ICCV*, 2019. 1

[23] Ilija Radosavovic, Raj Prateek Kosaraju, Ross Girshick, Kaiming He, and Piotr Dollár. Designing network design spaces. In *CVPR*, 2020. 1, 2, 4, 5, 6

[24] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. Searching for activation functions. *arXiv:1710.05941*, 2017. 2, 5

[25] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *AAAI*, 2019. 2

[26] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *CVPR*, 2018. 2, 5, 6

[27] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015. 2

[28] Chen Sun, Abhinav Shrivastava, Saurabh Singh, and Abhinav Gupta. Revisiting unreasonable effectiveness of data in deep learning era. In *ICCV*, 2017. 2

[29] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *CVPR*, 2015. 2

[30] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *CVPR*, 2016. 2, 6

[31] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V. Le. MnasNet: Platform-aware neural architecture search for mobile. In *CVPR*, 2019. 2, 5

[32] Mingxing Tan and Quoc V Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *ICML*, 2019. 1, 2, 3, 4, 5, 6, 7, 8

[33] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *CVPR*, 2017. 2, 6, 8

[34] C Yang and S Williams. Performance analysis of GPU-accelerated applications using the roofline model, 2019. 4

[35] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *BMVC*, 2016. 2, 5

[36] Hongyi Zhang, Moustapha Cissé, Yann N. Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. In *ICLR*, 2018. 6

[37] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. In *ICLR*, 2017. 1, 2

[38] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *CVPR*, 2018. 2