Designing Network Design Spaces

Ilija Radosavovic Raj Prateek Kosaraju Ross Girshick Kaiming He Piotr Dollár
Facebook AI Research (FAIR)

Abstract

In this work, we present a new network design paradigm. Our goal is to help advance the understanding of network design and discover design principles that generalize across settings. Instead of focusing on designing individual network instances, we design network design spaces that parametrize populations of networks. The overall process is analogous to classic manual design of networks, but elevated to the design space level. Using our methodology we explore the structure aspect of network design and arrive at a low-dimensional design space consisting of simple, regular networks that we call `RegNet`. *The core insight of the* `RegNet` *parametrization is surprisingly simple: widths and depths of good networks can be explained by a quantized linear function. We analyze the* `RegNet` *design space and arrive at interesting findings that do not match the current practice of network design. The* `RegNet` *design space provides simple and fast networks that work well across a wide range of flop regimes. Under comparable training settings and flops, the* `RegNet` *models outperform the popular EfficientNet models while being up to 5× faster on GPUs.*

1. Introduction

Deep convolutional neural networks are the engine of visual recognition. Over the past several years better architectures have resulted in considerable progress in a wide range of visual recognition tasks. Examples include LeNet [15], AlexNet [13], VGG [26], and ResNet [8]. This body of work advanced both the effectiveness of neural networks as well as our *understanding* of network design. In particular, the above sequence of works demonstrated the importance of convolution, network and data size, depth, and residuals, respectively. The outcome of these works is not just particular network instantiations, but also *design principles* that can be generalized and applied to numerous settings.

While *manual* network design has led to large advances, finding well-optimized networks manually can be challenging, especially as the number of design choices increases. A popular approach to address this limitation is neural architecture search (NAS). Given a *fixed* search space of possible networks, NAS automatically finds a good model within the search space. Recently, NAS has received a lot of attention and shown excellent results [34, 18, 29].

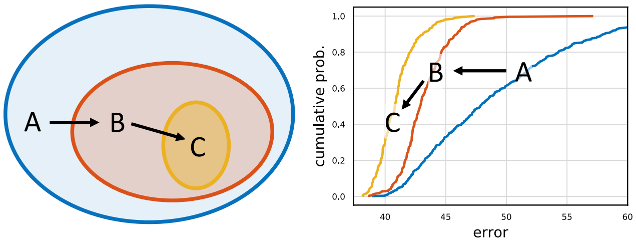


Figure 1. **Design space design.** We propose to *design* network design spaces, where a design space is a parametrized *set* of possible model architectures. Design space design is akin to manual network design, but elevated to the *population* level. In each step of our process the input is an initial design space and the output is a refined design space of simpler or better models. Following [21], we characterize the quality of a design space by sampling models and inspecting their *error distribution*. For example, in the figure above we start with an initial design space `A` and apply two refinement steps to yield design spaces `B` then `C`. In this case `C` ⊆ `B` ⊆`A` (left), and the error distributions are strictly improving from `A` to `B` to `C` (right). The hope is that *design principles* that apply to model populations are more likely to be robust and generalize.

Despite the effectiveness of NAS, the paradigm has limitations. The outcome of the search is a single network *instance* tuned to a specific setting (*e.g.*, hardware platform). This is sufficient in some cases; however, it does not enable discovery of *network design principles* that deepen our understanding and allow us to generalize to new settings. In particular, our aim is to find simple models that are easy to understand, build upon, and generalize.

In this work, we present a new network design paradigm that combines the advantages of manual design and NAS. Instead of focusing on designing individual network instances, we *design design spaces* that parametrize populations of networks.[1] Like in manual design, we aim for interpretability and to discover general *design principles* that describe networks that are simple, work well, and generalize across settings. Like in NAS, we aim to take advantage of semi-automated procedures to help achieve these goals.

[1] We use the term *design space* following [21], rather than search space, to emphasize that we are not searching for network instances within the space. Instead, we are designing the space itself.

The general strategy we adopt is to progressively design simplified versions of an initial, relatively unconstrained, design space while maintaining or improving its quality (Figure 1). The overall process is analogous to manual design, elevated to the population level and guided via distribution estimates of network design spaces [21].

As a testbed for this paradigm, our focus is on exploring network *structure* (*e.g.*, width, depth, groups, *etc.*) assuming standard model families including VGG [26], ResNet [8], and ResNeXt [31]. We start with a relatively unconstrained design space we call `AnyNet` (*e.g.*, widths and depths vary freely across stages) and apply our human-in-the-loop methodology to arrive at a low-dimensional design space consisting of simple "regular" networks, that we call `RegNet`. The core of the `RegNet` design space is simple: stage widths and depths are determined by a *quantized linear function*. Compared to `AnyNet`, the `RegNet` design space has simpler models, is easier to interpret, and has a higher concentration of good models.

We design the `RegNet` design space in a low-compute, low-epoch regime using a single network block type on ImageNet [3]. We then show that the `RegNet` design space generalizes to larger compute regimes, schedule lengths, and network block types. Furthermore, an important property of the design space design is that it is more interpretable and can lead to insights that we can learn from. We analyze the `RegNet` design space and arrive at interesting findings that do not match the current practice of network design. For example, we find that the depth of the best models is stable across compute regimes (~20 blocks) and that the best models do not use either a bottleneck or inverted bottleneck.

We compare top REGNET models to existing networks in various settings. First, REGNET models are surprisingly effective in the mobile regime. We hope that these simple models can serve as strong baselines for future work. Next, REGNET models lead to considerable improvements over standard RESNE(X)T [8, 31] models in all metrics. We highlight the improvements for fixed activations, which is of high practical interest as the number of activations can strongly influence the runtime on accelerators such as GPUs. Next, we compare to the state-of-the-art EFFICIENTNET [29] models across compute regimes. Under *comparable training settings* and flops, REGNET models outperform EFFICIENTNET models while being up to 5× *faster* on GPUs. We further test generalization on ImageNetV2 [24].

We note that network structure is arguably the simplest form of a design space design one can consider. Focusing on designing richer design spaces (*e.g.*, including operators) may lead to better networks. Nevertheless, the structure will likely remain a core component of such design spaces.

In order to facilitate future research we will release all code and pretrained models introduced in this work.[2]

2. Related Work

Manual network design. The introduction of AlexNet [13] catapulted network design into a thriving research area. In the following years, improved network designs were proposed; examples include VGG [26], Inception [27, 28], ResNet [8], ResNeXt [31], DenseNet [11], and MobileNet [9, 25]. The design process behind these networks was largely manual and focussed on discovering new design choices that improve accuracy *e.g.*, the use of deeper models or residuals. We likewise share the goal of discovering new design principles. In fact, our methodology is analogous to manual design but performed at the design space level.

Automated network design. Recently, the network design process has shifted from a manual exploration to more automated network design, popularized by NAS. NAS has proven to be an effective tool for finding good models, *e.g.*, [35, 23, 17, 20, 18, 29]. The majority of work in NAS focuses on the search algorithm, *i.e.*, efficiently finding the best network instances within a fixed, manually designed search space (which we call a *design space*). Instead, our focus is on a paradigm for designing novel design spaces. The two are complementary: better design spaces can improve the efficiency of NAS search algorithms and also lead to existence of better models by enriching the design space.

Network scaling. Both manual and semi-automated network design typically focus on finding best-performing network instances for a specific regime (*e.g.*, number of flops comparable to ResNet-50). Since the result of this procedure is a single network instance, it is not clear how to adapt the instance to a different regime (*e.g.*, fewer flops). A common practice is to apply network scaling rules, such as varying network depth [8], width [32], resolution [9], or all three jointly [29]. Instead, our goal is to discover general design principles that hold across regimes and allow for efficient tuning for the optimal network in any target regime.

Comparing networks. Given the vast number of possible network design spaces, it is essential to use a reliable comparison metric to guide our design process. Recently, the authors of [21] proposed a methodology for comparing and analyzing populations of networks sampled from a design space. This distribution-level view is fully-aligned with our goal of finding general design principles. Thus, we adopt this methodology and demonstrate that it can serve as a useful tool for the design space design process.

Parameterization. Our final quantized linear parameterization shares similarity with previous work, *e.g.* how stage widths are set [26, 7, 32, 11, 9]. However, there are two key differences. First, we provide an empirical study justifying the design choices we make. Second, we give insights into structural design choices that were not previously understood (*e.g.*, how to set the number of blocks in each stages).

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

3. Design Space Design

Our goal is to design better networks for visual recognition. Rather than designing or searching for a *single* best model under specific settings, we study the behavior of *populations* of models. We aim to discover general *design principles* that can apply to and improve an entire model population. Such design principles can provide insights into network design and are more likely to generalize to new settings (unlike a single model tuned for a specific scenario).

We rely on the concept of network *design spaces* introduced by Radosavovic *et al.* [21]. A design space is a large, possibly infinite, population of model architectures. The core insight from [21] is that we can sample models from a design space, giving rise to a model distribution, and turn to tools from classical statistics to analyze the design space. We note that this differs from architecture search, where the goal is to find the single best model from the space.

In this work, we propose to *design* progressively simplified versions of an initial, unconstrained design space. We refer to this process as *design space design*. Design space design is akin to sequential manual network design, but elevated to the population level. Specifically, in each step of our design process the input is an initial design space and the output is a refined design space, where the aim of each design step is to discover design principles that yield populations of simpler or better performing models.

We begin by describing the basic tools we use for design space design in §3.1. Next, in §3.2 we apply our methodology to a design space, called `AnyNet`, that allows unconstrained network structures. In §3.3, after a sequence of design steps, we obtain a simplified design space consisting of only *regular* network structures that we name `RegNet`. Finally, as our goal is not to design a design space for a single setting, but rather to discover general principles of network design that *generalize* to new settings, in §3.4 we test the generalization of the `RegNet` design space to new settings.

Relative to the `AnyNet` design space, the `RegNet` design space is: (1) simplified both in terms of its dimension and type of network configurations it permits, (2) contains a higher concentration of top-performing models, and (3) is more amenable to analysis and interpretation.

3.1. Tools for Design Space Design

We begin with an overview of tools for design space design. To evaluate and compare design spaces, we use the tools introduced by Radosavovic *et al.* [21], who propose to quantify the quality of a design space by *sampling* a set of models from that design space and characterizing the resulting model error *distribution*. The key intuition behind this approach is that comparing distributions is more robust and informative than using search (manual or automated) and comparing the best found models from two design spaces.

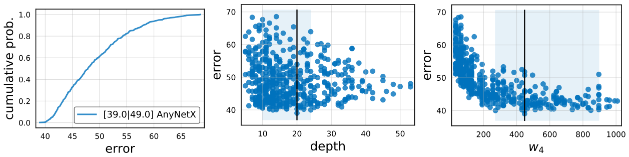


Figure 2. Statistics of the **`AnyNetX` design space** computed with $n = 500$ sampled models. *Left:* The error *empirical distribution function* (EDF) serves as our foundational tool for visualizing the quality of the design space. In the legend we report the min error and mean error (which corresponds to the area under the curve). *Middle:* Distribution of network depth d (number of blocks) versus error. *Right:* Distribution of block widths in the fourth stage (w_4) versus error. The blue shaded regions are ranges containing the best models with 95% confidence (obtained using an empirical bootstrap), and the black vertical line the most likely best value.

To obtain a distribution of models, we sample and train n models from a design space. For efficiency, we primarily do so in a *low-compute*, *low-epoch* training regime. In particular, in this section we use the 400 million flop[3] (400MF) regime and train each sampled model for 10 epochs on the ImageNet dataset [3]. We note that while we train many models, each training run is fast: training 100 models at 400MF for 10 epochs is roughly equivalent in flops to training a *single* ResNet-50 [8] model at 4GF for 100 epochs.

As in [21], our primary tool for analyzing design space quality is the error *empirical distribution function* (EDF). The error EDF of n models with errors e_i is given by:

$$F(e) = \frac{1}{n}\sum_{i=1}^{n} \mathbf{1}[e_i < e]. \tag{1}$$

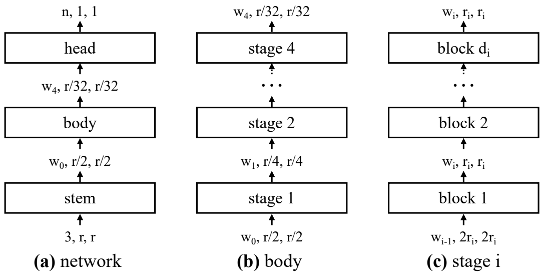
$F(e)$ gives the fraction of models with error less than e. We show the error EDF for $n = 500$ sampled models from the `AnyNetX` design space (described in §3.2) in Figure 2 (left).

Given a population of trained models, we can plot and analyze various network properties versus network error, see Figure 2 (middle) and (right) for two examples taken from the `AnyNetX` design space. Such visualizations show 1D projections of a complex, high-dimensional space, and can help obtain insights into the design space. For these plots, we employ an *empirical bootstrap*[4] [5] to estimate the likely range in which the best models fall.

To summarize: (1) we generate distributions of models obtained by sampling and training n models from a design space, (2) we compute and plot error EDFs to summarize design space quality, (3) we visualize various properties of a design space and use an empirical bootstrap to gain insight, and (4) we use these insights to refine the design space.

[3] Following common practice, we use flops to mean multiply-adds. Moreover, we use MF and GF to denote 10^6 and 10^9 flops, respectively.

[4] Given n pairs (x_i, e_i) of model statistic x_i (*e.g.* depth) and corresponding error e_i, we compute the *empirical bootstrap* by: (1) sampling with replacement 25% of the pairs, (2) selecting the pair with min error in the sample, (3) repeating this 10^4 times, and finally (4) computing the 95% CI for the min x value. The median gives the most likely best value.



(a) network **(b)** body **(c)** stage i

Figure 3. General **network structure** for models in our design spaces. (a) Each network consists of a stem (stride-two 3×3 conv with $w_0 = 32$ output channels), followed by the network body that performs the bulk of the computation, and then a head (average pooling followed by a fully connected layer) that predicts n output classes. (b) The network body is composed of a sequence of stages that operate at progressively reduced resolution r_i. (c) Each stage consists of a sequence of identical blocks, except the first block which uses stride-two conv. While the general structure is simple, the total number of possible network configurations is vast.

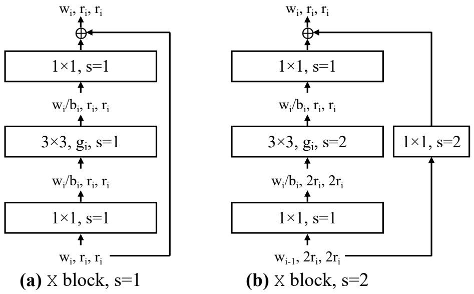
3.2. The AnyNet Design Space

We next introduce our initial AnyNet design space. Our focus is on exploring the *structure* of neural networks assuming standard, fixed network blocks (*e.g.*, residual bottleneck blocks). In our terminology the structure of the network includes elements such as the number of blocks (*i.e.* network depth), block widths (*i.e.* number of channels), and other block parameters such as bottleneck ratios or group widths. The structure of the network determines the distribution of compute, parameters, and memory throughout the computational graph of the network and is key in determining its accuracy and efficiency.

The basic design of networks in our AnyNet design space is straightforward. Given an input image, a network consists of a simple *stem*, followed by the network *body* that performs the bulk of the computation, and a final network *head* that predicts the output classes, see Figure 3a. We keep the stem and head fixed and as simple as possible, and instead focus on the structure of the network body that is central in determining network compute and accuracy.

The network body consists of 4 *stages* operating at progressively reduced resolution, see Figure 3b (we explore varying the number of stages in §3.4). Each stage consists of a sequence of identical blocks, see Figure 3c. In total, for each stage i the degrees of freedom include the number of blocks d_i, block width w_i, and any other block parameters. While the general structure is simple, the total number of possible networks in the AnyNet design space is vast.

Most of our experiments use the standard residual bottlenecks block with group convolution [31], shown in Figure 4. We refer to this as the X block, and the AnyNet design space built on it as AnyNetX (we explore other blocks in §3.4). While the X block is quite rudimentary, we show it can be surprisingly effective when network structure is optimized.



(a) X block, s=1 **(b)** X block, s=2

Figure 4. The **X block** is based on the standard residual bottleneck block with group convolution [31]. (a) Each X block consists of a 1×1 conv, a 3×3 group conv, and a final 1×1 conv, where the 1×1 convs alter the channel width. BatchNorm [12] and ReLU follow each conv. The block has 3 parameters: the width w_i, bottleneck ratio b_i, and group width g_i. (b) The stride-two ($s = 2$) version.

The AnyNetX design space has 16 degrees of freedom as each network consists of 4 stages and each stage i has 4 parameters: the number of blocks d_i, block width w_i, bottleneck ratio b_i, and group width g_i. We fix the input resolution $r = 224$ unless otherwise noted. To obtain valid models, we perform log-uniform sampling of $d_i \leq 16$, $w_i \leq 1024$ and divisible by 8, $b_i \in \{1, 2, 4\}$, and $g_i \in \{1, 2, \ldots, 32\}$ (we test these ranges later). We repeat the sampling until we obtain $n = 500$ models in our target complexity regime (360MF to 400MF), and train each model for 10 epochs.[5] Basic statistics for AnyNetX were shown in Figure 2.

There are $(16{\cdot}128{\cdot}3{\cdot}6)^4 \approx 10^{18}$ possible model configurations in the AnyNetX design space. Rather than *searching* for the single best model out of these ~10^{18} configurations, we explore whether there are general *design principles* that can help us understand and refine this design space. To do so, we apply our approach of designing design spaces. In each step of this approach, our aims are:

1. to simplify the structure of the design space,
2. to improve the interpretability of the design space,
3. to improve or maintain the design space quality,
4. to maintain model diversity in the design space.

We now apply this approach to the AnyNetX design space.

AnyNetX$_\text{A}$. For clarity, going forward we refer to the initial, unconstrained AnyNetX design space as AnyNetX$_\text{A}$.

AnyNetX$_\text{B}$. We first test a *shared* bottleneck ratio $b_i = b$ for all stages i for the AnyNetX$_\text{A}$ design space, and refer to the resulting design space as AnyNetX$_\text{B}$. As before, we sample and train 500 models from AnyNetX$_\text{B}$ in the same settings. The EDFs of AnyNetX$_\text{A}$ and AnyNetX$_\text{B}$, shown in Figure 5 (left), are virtually identical both in the average and best case. This indicates no loss in accuracy when coupling the b_i. In addition to being simpler, the AnyNetX$_\text{B}$ is more amenable to analysis, see for example Figure 5 (right).

[5] Our training setup in §3 exactly follows [21]. We use SGD with momentum of 0.9, mini-batch size of 128 on 1 GPU, and a half-period cosine schedule with initial learning rate of 0.05 and weight decay of $5{\cdot}10^{-5}$. Ten epochs are usually sufficient to give robust *population* statistics.

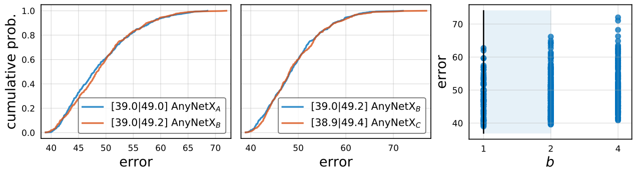


Figure 5. **AnyNetX$_B$** (left) and **AnyNetX$_C$** (middle) introduce a *shared* bottleneck ratio $b_i = b$ and shared group width $g_i = g$, respectively. This simplifies the design spaces while resulting in virtually no change in the error EDFs. Moreover, AnyNetX$_B$ and AnyNetX$_C$ are more amendable to analysis. Applying an empirical bootstrap to b and g we see trends emerge, *e.g.*, with 95% confidence $b \leq 2$ is best in this regime (right). No such trends are evident in the individual b_i and g_i in AnyNetX$_A$ (not shown).

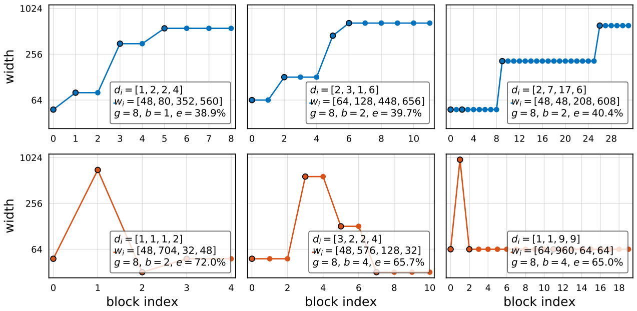


Figure 6. Example good and bad **AnyNetX$_C$ networks**, shown in the top and bottom rows, respectively. For each network, we plot the width w_j of every block j up to the network depth d. These *per-block* widths w_j are computed from the *per-stage* block depths d_i and block widths w_i (listed in the legends for reference).

AnyNetX$_C$. Our second refinement step closely follows the first. Starting with AnyNetX$_B$, we additionally use a *shared* group width $g_i = g$ for all stages to obtain AnyNetX$_C$. As before, the EDFs are nearly unchanged, see Figure 5 (middle). Overall, AnyNetX$_C$ has 6 fewer degrees of freedom than AnyNetX$_A$, and reduces the design space size nearly four orders of magnitude. Interestingly, we find $g > 1$ is best (not shown); we analyze this in more detail in §4.

AnyNetX$_D$. Next, we examine typical network structures of both good and bad networks from AnyNetX$_C$ in Figure 6. A pattern emerges: good network have increasing widths. We test the design principle of $w_{i+1} \geq w_i$, and refer to the design space with this constraint as AnyNetX$_D$. In Figure 7 (left) we see this improves the EDF substantially. We return to examining other options for controlling width shortly.

AnyNetX$_E$. Upon further inspection of many models (not shown), we observed another interesting trend. In addition to stage widths w_i increasing with i, the stage depths d_i likewise tend to increase for the best models, although not necessarily in the last stage. Nevertheless, we test a design space variant AnyNetX$_E$ with $d_{i+1} \geq d_i$ in Figure 7 (right), and see it also improves results. Finally, we note that the constraints on w_i and d_i each reduce the design space by 4!, with a cumulative reduction of $O(10^7)$ from AnyNetX$_A$.

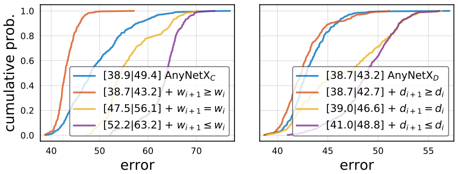


Figure 7. **AnyNetX$_D$** (left) and **AnyNetX$_E$** (right). We show various constraints on the per stage widths w_i and depths d_i. In both cases, having increasing w_i and d_i is beneficial, while using constant or decreasing values is much worse. Note that AnyNetX$_D$ = AnyNetX$_C$ + $w_{i+1} \geq w_i$, and AnyNetX$_E$ = AnyNetX$_D$ + $d_{i+1} \geq d_i$. We explore stronger constraints on w_i and d_i shortly.

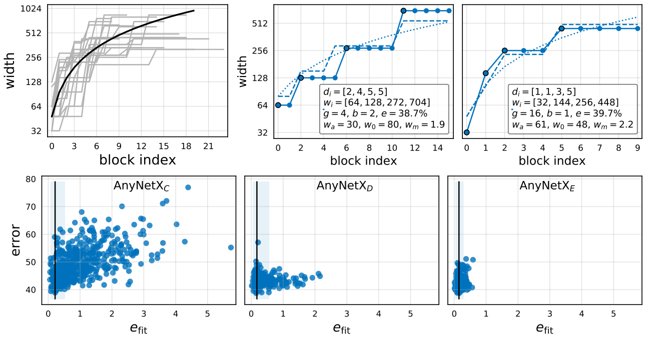


Figure 8. **Linear fits**. Top networks from the AnyNetX design space can be well modeled by a *quantized linear parameterization*, and conversely, networks for which this parameterization has a higher fitting error e_{fit} tend to perform poorly. See text for details.

3.3. The RegNet Design Space

To gain further insight into the model structure, we show the best 20 models from AnyNetX$_E$ in a single plot, see Figure 8 (top-left). For each model, we plot the per-block width w_j of every block j up to the network depth d (we use i and j to index over stages and blocks, respectively). See Figure 6 for reference of our model visualization.

While there is significant variance in the individual models (gray curves), in the aggregate a pattern emerges. In particular, in the same plot we show the line $w_j = 48 \cdot (j+1)$ for $0 \leq j \leq 20$ (solid black curve, please note that the y-axis is logarithmic). Remarkably, this trivial *linear fit* seems to explain the population trend of the growth of network widths for top models. Note, however, that this linear fit assigns a different width w_j to each block, whereas individual models have quantized widths (piecewise constant functions).

To see if a similar pattern applies to *individual* models, we need a strategy to *quantize* a line to a piecewise constant function. Inspired by our observations from AnyNetX$_D$ and AnyNetX$_E$, we propose the following approach. First, we introduce a *linear parameterization* for block widths:

$$u_j = w_0 + w_a \cdot j \quad \text{for} \quad 0 \leq j < d \tag{2}$$

This parameterization has three parameters: depth d, initial width $w_0 > 0$, and slope $w_a > 0$, and generates a different block width u_j for each block $j < d$. To quantize u_j,

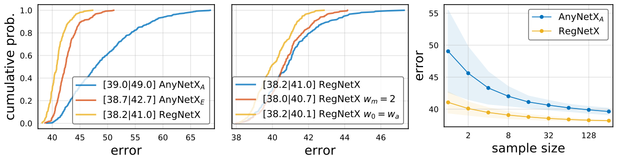


Figure 9. **RegNetX** design space. See text for details.

we introduce an additional parameter $w_m > 0$ that controls quantization as follows. First, given u_j from Eqn. (2), we compute s_j for each block j such that the following holds:

$$u_j = w_0 \cdot w_m^{s_j} \tag{3}$$

Then, to quantize u_j, we simply round s_j (denoted by $\lfloor s_j \rceil$) and compute quantized per-block widths w_j via:

$$w_j = w_0 \cdot w_m^{\lfloor s_j \rceil} \tag{4}$$

We can convert the per-block w_j to our per-stage format by simply counting the number of blocks with constant width, that is, each stage i has block width $w_i = w_0 {\cdot} w_m^i$ and number of blocks $d_i = \sum_j \mathbf{1}[\lfloor s_j \rceil = i]$. When only considering four stage networks, we ignore the parameter combinations that give rise to a different number of stages.

We test this parameterization by fitting to models from AnyNetX. In particular, given a model, we compute the fit by setting d to the network depth and performing a grid search over w_0, w_a and w_m to minimize the mean log-ratio (denoted by e_{fit}) of predicted to observed per-block widths. Results for two top networks from AnyNetX$_\text{E}$ are shown in Figure 8 (top-right). The quantized linear fits (dashed curves) are good fits of these best models (solid curves).

Next, we plot the fitting error e_{fit} versus network error for every network in AnyNetX$_\text{C}$ through AnyNetX$_\text{E}$ in Figure 8 (bottom). First, we note that *the best models in each design space all have good linear fits*. Indeed, an empirical bootstrap gives a narrow band of e_{fit} near 0 that likely contains the best models in each design space. Second, we note that on average, e_{fit} improves going from AnyNetX$_\text{C}$ to AnyNetX$_\text{E}$, showing that the linear parametrization naturally enforces related constraints to w_i and d_i increasing.

To further test the linear parameterization, we design a design space that only contains models with such linear structure. In particular, we specify a network structure via 6 parameters: d, w_0, w_a, w_m (and also b, g). Given these, we generate block widths and depths via Eqn. (2)-(4). We refer to the resulting design space as RegNet, as it contains only simple, *regular* models. We sample $d < 64$, $w_0, w_a < 256$, $1.5 \leq w_m \leq 3$ and b and g as before (ranges set based on e_{fit} on AnyNetX$_\text{E}$).

The error EDF of RegNetX is shown in Figure 9 (left). Models in RegNetX have better average error than AnyNetX while maintaining the best models. In Figure 9 (middle) we test two further simplifications. First, using $w_m = 2$ (doubling width between stages) slightly improves the EDF, but we note that using $w_m \geq 2$ performs better (shown later). Second, we test setting $w_0 = w_a$, further simplifying the linear parameterization to $u_j = w_a \cdot (j+1)$. Interestingly, this performs even better. However, to maintain the diversity of models, we do not impose either restriction. Finally, in Figure 9 (right) we show that random search efficiency is much higher for RegNetX; searching over just ~32 random models is likely to yield good models.

	restriction	dim.	combinations	total
AnyNetX$_\text{A}$	none	16	$(16{\cdot}128{\cdot}3{\cdot}6)^4$	$\sim 1.8{\cdot}10^{18}$
AnyNetX$_\text{B}$	$+\, b_{i+1} = b_i$	13	$(16{\cdot}128{\cdot}6)^4{\cdot}3$	$\sim 6.8{\cdot}10^{16}$
AnyNetX$_\text{C}$	$+\, g_{i+1} = g_i$	10	$(16{\cdot}128)^4{\cdot}3{\cdot}6$	$\sim 3.2{\cdot}10^{14}$
AnyNetX$_\text{D}$	$+\, w_{i+1} \geq w_i$	10	$(16{\cdot}128)^4{\cdot}3{\cdot}6/(4!)$	$\sim 1.3{\cdot}10^{13}$
AnyNetX$_\text{E}$	$+\, d_{i+1} \geq d_i$	10	$(16{\cdot}128)^4{\cdot}3{\cdot}6/(4!)^2$	$\sim 5.5{\cdot}10^{11}$
RegNet	quantized linear	6	$\sim 64^4{\cdot}6{\cdot}3$	$\sim 3.0{\cdot}10^8$

Table 1. **Design space summary.** See text for details.

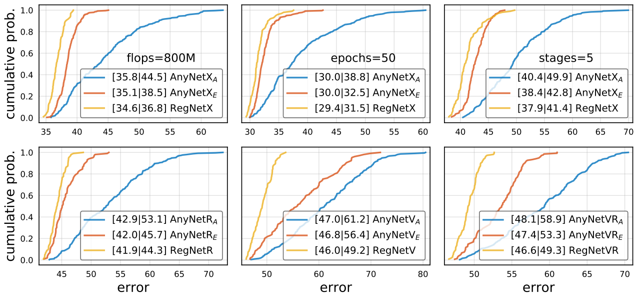


Figure 10. **RegNetX generalization**. We compare RegNetX to AnyNetX at higher flops (top-left), higher epochs (top-middle), with 5-stage networks (top-right), and with various block types (bottom). In all cases the ordering of the design spaces is consistent and we see no signs of design space overfitting.

Table 1 shows a summary of the design space sizes (for RegNet we estimate the size by quantizing its continuous parameters). In designing RegNetX, we reduced the dimension of the original AnyNetX design space from 16 to 6 dimensions, and the size nearly 10 orders of magnitude. We note, however, that RegNet still contains a good diversity of models that can be tuned for a variety of settings.

3.4. Design Space Generalization

We designed the RegNet design space in a low-compute, low-epoch training regime with only a single block type. However, our goal is not to design a design space for a single setting, but rather to discover general principles of network design that can *generalize* to new settings.

In Figure 10, we compare the RegNetX design space to AnyNetX$_\text{A}$ and AnyNetX$_\text{E}$ at higher flops, higher epochs, with 5-stage networks, and with various block types (described in the appendix). In all cases the ordering of the design spaces is consistent, with RegNetX > AnyNetX$_\text{E}$ > AnyNetX$_\text{A}$. In other words, we see no signs of overfitting. These results are promising because they show RegNet can generalize to new settings. The 5-stage results show the regular structure of RegNet can generalize to more stages, where AnyNetX$_\text{A}$ has even more degrees of freedom.

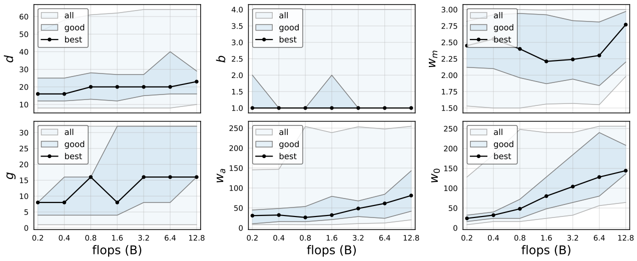
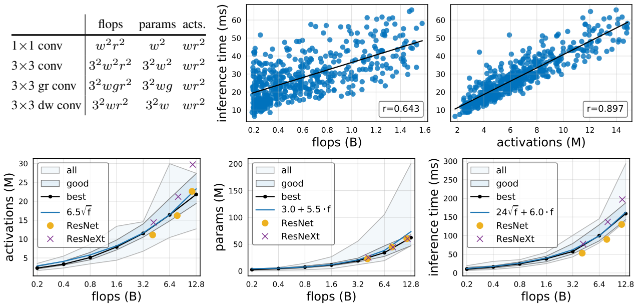


Figure 11. **`RegNetX` parameter trends**. For each parameter and each flop regime we apply an empirical bootstrap to obtain the range that contains best models with 95% confidence (shown with blue shading) and the likely best model (black line), see also Figure 2. We observe that for best models the depths d are remarkably stable across flops regimes, and $b = 1$ and $w_m \approx 2.5$ are best. Block and groups widths (w_a, w_0, g) tend to increase with flops.



	flops	params	acts.
1×1 conv	w^2r^2	w^2	wr^2
3×3 conv	$3^2w^2r^2$	3^2w^2	wr^2
3×3 gr conv	3^2wgr^2	3^2wg	wr^2
3×3 dw conv	3^2wr^2	3^2w	wr^2

Figure 12. **Complexity metrics**. *Top:* Activations can have a stronger correlation to runtime on hardware accelerators than flops (we measure inference time for 64 images on an NVIDIA V100 GPU). *Bottom:* Trend analysis of complexity *vs.* flops and best fit curves (shown in blue) of the trends for best models (black curves).

4. Analyzing the `RegNetX` Design Space

We next further analyze the `RegNetX` design space and revisit common deep network design choices. Our analysis yields surprising insights that don't match popular practice, which allows us to achieve good results with simple models.

As the `RegNetX` design space has a high concentration of good models, for the following results we switch to sampling fewer models (100) but training them for longer (25 epochs) with a learning rate of 0.1 (see appendix). We do so to observe more fine-grained trends in network behavior.

`RegNet` trends. We show trends in the `RegNetX` parameters across flop regimes in Figure 11. Remarkably, the *depth of best models is stable across regimes* (top-left), with an optimal depth of ∼20 blocks (60 layers). This is in contrast to the common practice of using deeper models for higher flop regimes. We also observe that *the best models use a bottleneck ratio b of 1.0* (top-middle), which effectively removes the bottleneck (commonly used in practice). Next, we observe that the width multiplier w_m of good models is ∼2.5 (top-right), similar but not identical to the popular recipe of doubling widths across stages. The remaining parameters (g, w_a, w_0) increase with complexity (bottom).

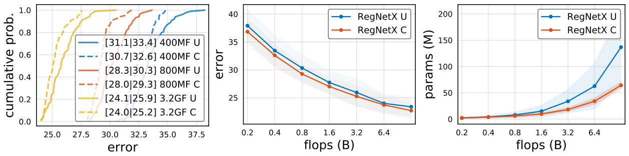


Figure 13. We **refine `RegNetX`** using various constraints (see text). The constrained variant (C) is best across all flop regimes while being more efficient in terms of parameters and activations.

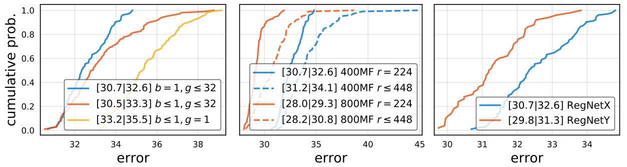


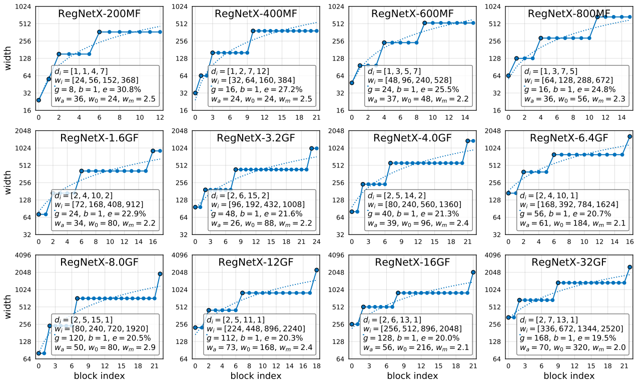
Figure 14. We evaluate `RegNetX` with **alternate design choices**. *Left:* Inverted bottleneck ($\frac{1}{8} \leq b \leq 1$) degrades results and depthwise conv ($g = 1$) is even worse. *Middle:* Varying resolution r harms results. *Right:* `RegNetY` (`Y=X+SE`) improves the EDF.

Complexity analysis. In addition to flops and parameters, we analyze network *activations*, which we define as the size of the output tensors of all conv layers (we list complexity measures of common conv operators in Figure 12, top-left). While not a common measure of network complexity, activations can heavily affect runtime on memory-bound hardware accelerators (*e.g.*, GPUs, TPUs), for example, see Figure 12 (top). In Figure 12 (bottom), we observe that for the best models in the population, activations increase with the square-root of flops, parameters increase linearly, and runtime is best modeled using both a linear and a square-root term due to its dependence on both flops and activations.

`RegNetX` constrained. Using these findings, we refine the `RegNetX` design space. First, based on Figure 11 (top), we set $b = 1$, $d \leq 40$, and $w_m \geq 2$. Second, we limit parameters and activations, following Figure 12 (bottom). This yields fast, low-parameter, low-memory models without affecting accuracy. In Figure 13, we test `RegNetX` with theses constraints and observe that the constrained version is superior across all flop regimes. We use this version in §5, and further limit depth to $12 \leq d \leq 28$ (see also Appendix D).

Alternate design choices. Modern mobile networks often employ the inverted bottleneck ($b < 1$) proposed in [25] along with depthwise conv [1] ($g = 1$). In Figure 14 (left), we observe that the *inverted bottleneck degrades the EDF slightly and depthwise conv performs even worse* relative to $b = 1$ and $g \geq 1$ (see appendix for further analysis). Next, motivated by [29] who found that scaling the input image resolution can be helpful, we test varying resolution in Figure 14 (middle). Contrary to [29], we find that for `RegNetX` a fixed resolution of 224×224 is best, even at higher flops.

SE. Finally, we evaluate `RegNetX` with the popular Squeeze-and-Excitation (SE) op [10] (we abbreviate `X+SE` as `Y` and refer to the resulting design space as `RegNetY`). In Figure 14 (right), we see that `RegNetY` yields good gains.



	flops (B)	params (M)	acts (M)	batch size	infer (ms)	train (hr)	error (top-1)
REGNETX-200MF	0.2	2.7	2.2	1024	10	2.8	31.1±0.09
REGNETX-400MF	0.4	5.2	3.1	1024	15	3.9	27.3±0.15
REGNETX-600MF	0.6	6.2	4.0	1024	17	4.4	25.9±0.03
REGNETX-800MF	0.8	7.3	5.1	1024	21	5.7	24.8±0.09
REGNETX-1.6GF	1.6	9.2	7.9	1024	33	8.7	23.0±0.13
REGNETX-3.2GF	3.2	15.3	11.4	512	57	14.3	21.7±0.08
REGNETX-4.0GF	4.0	22.1	12.2	512	69	17.1	21.4±0.19
REGNETX-6.4GF	6.5	26.2	16.4	512	92	23.5	20.8±0.07
REGNETX-8.0GF	8.0	39.6	14.1	512	94	22.6	20.7±0.07
REGNETX-12GF	12.1	46.1	21.4	512	137	32.9	20.3±0.04
REGNETX-16GF	15.9	54.3	25.5	512	168	39.7	20.0±0.11
REGNETX-32GF	31.7	107.8	36.3	256	318	76.9	19.5±0.12

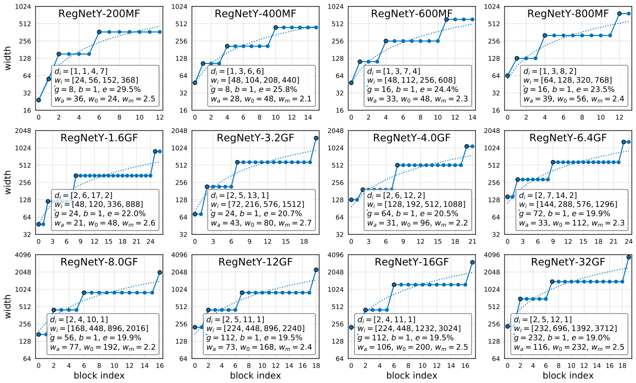
Figure 15. **Top REGNETX models.** We measure inference time for 64 images on an NVIDIA V100 GPU; train time is for 100 epochs on 8 GPUs with the batch size listed. Network diagram legends contain all information required to implement the models.

5. Comparison to Existing Networks

We now compare top models from the RegNetX and RegNetY design spaces at various complexities to the state-of-the-art on ImageNet [3]. We denote individual models using small caps, *e.g.* REGNETX. We also suffix the models with the flop regime, *e.g.* 400MF. For each flop regime, we pick the best model from 25 random settings of the RegNet parameters (d, g, w_m, w_a, w_0), and re-train the top model 5 times at 100 epochs to obtain robust error estimates.

Resulting top REGNETX and REGNETY models for each flop regime are shown in Figures 15 and 16, respectively. In addition to the simple linear structure and the trends we analyzed in §4, we observe an interesting pattern. Namely, the higher flop models have a large number of blocks in the third stage and a small number of blocks in the last stage. This is similar to the design of standard RESNET models. Moreover, we observe that the group width g increases with complexity, but depth d saturates for large models.

Our goal is to perform fair comparisons and provide simple and easy-to-reproduce baselines. We note that along with better architectures, much of the recently reported gains in network performance are based on enhancements to the *training setup and regularization scheme* (see Table 7). As our focus is on evaluating *network architectures*, we perform carefully controlled experiments under the same training setup. In particular, to provide fair comparisons to classic work, *we do not use any training-time enhancements*.



	flops (B)	params (M)	acts (M)	batch size	infer (ms)	train (hr)	error (top-1)
REGNETY-200MF	0.2	3.2	2.2	1024	11	3.1	29.6±0.11
REGNETY-400MF	0.4	4.3	3.9	1024	19	5.1	25.9±0.16
REGNETY-600MF	0.6	6.1	4.3	1024	19	5.2	24.5±0.07
REGNETY-800MF	0.8	6.3	5.2	1024	22	6.0	23.7±0.03
REGNETY-1.6GF	1.6	11.2	8.0	1024	39	10.1	22.0±0.08
REGNETY-3.2GF	3.2	19.4	11.3	512	67	16.5	21.0±0.05
REGNETY-4.0GF	4.0	20.6	12.3	512	68	16.8	20.6±0.08
REGNETY-6.4GF	6.4	30.6	16.4	512	104	26.1	20.1±0.04
REGNETY-8.0GF	8.0	39.2	18.0	512	113	28.1	20.1±0.09
REGNETY-12GF	12.1	51.8	21.4	512	150	36.0	19.7±0.06
REGNETY-16GF	15.9	83.6	23.0	512	189	45.6	19.6±0.16
REGNETY-32GF	32.3	145.0	30.3	256	319	76.0	19.0±0.12

Figure 16. **Top REGNETY models** (Y=X+SE). The benchmarking setup and the figure format is the same as in Figure 15.

5.1. State-of-the-Art Comparison: Mobile Regime

Much of the recent work on network design has focused on the mobile regime (~600MF). In Table 2, we compare REGNET models at 600MF to existing mobile networks. We observe that REGNETS are surprisingly effective in this regime considering the substantial body of work on finding better mobile networks via both manual design [9, 25, 19] and NAS [35, 23, 17, 18].

We emphasize that REGNET models use our basic 100 epoch schedule with no regularization except weight decay, while most mobile networks use longer schedules with various enhancements, such as deep supervision [16], Cutout [4], DropPath [14], AutoAugment [2], and so on. As such, we hope our strong results obtained with a short training schedule without enhancements can serve as a simple baseline for future work.

	flops (B)	params (M)	top-1 error
MOBILENET [9]	0.57	4.2	29.4
MOBILENET-V2 [25]	0.59	6.9	25.3
SHUFFLENET [33]	0.52	-	26.3
SHUFFLENET-V2 [19]	0.59	-	25.1
NASNET-A [35]	0.56	5.3	26.0
AMOEBANET-C [23]	0.57	6.4	24.3
PNASNET-5 [17]	0.59	5.1	25.8
DARTS [18]	0.57	4.7	26.7
REGNETX-600MF	0.60	6.2	25.9±0.03
REGNETY-600MF	0.60	6.1	24.5±0.07

Table 2. **Mobile regime.** We compare existing models using *originally* reported errors to RegNet models trained in a *basic* setup. Our *simple* RegNet models achieve surprisingly good results given the effort focused on this regime in the past few years.

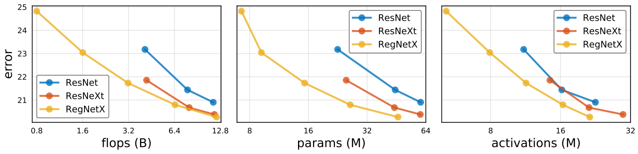


Figure 17. **ResNe(X)t comparisons.** RegNetX models versus ResNe(X)t-(50,101,152) under various complexity metrics. As all models use the identical components and training settings, all observed gains are from the design of the `RegNetX` design space.

	flops (B)	params (M)	acts (M)	infer (ms)	train (hr)	top-1 error ours±std [orig]
ResNet-50	4.1	22.6	11.1	53	12.2	23.2±0.09 [23.9]
RegNetX-3.2GF	3.2	15.3	11.4	57	14.3	**21.7**±0.08
ResNeXt-50	4.2	25.0	14.4	78	18.0	21.9±0.10 [22.2]
ResNet-101	7.8	44.6	16.2	90	20.4	21.4±0.11 [22.0]
RegNetX-6.4GF	6.5	26.2	16.4	92	23.5	**20.8**±0.07
ResNeXt-101	8.0	44.2	21.2	137	31.8	20.7±0.08 [21.2]
ResNet-152	11.5	60.2	22.6	130	29.2	20.9±0.12 [21.6]
RegNetX-12GF	12.1	46.1	21.4	137	32.9	**20.3**±0.04

(a) Comparisons grouped by **activations**.

	flops (B)	params (M)	acts (M)	infer (ms)	train (hr)	top-1 error ours±std [orig]
ResNet-50	4.1	22.6	11.1	53	12.2	23.2±0.09 [23.9]
ResNeXt-50	4.2	25.0	14.4	78	18.0	21.9±0.10 [22.2]
RegNetX-4.0GF	4.0	22.1	12.2	69	17.1	**21.4**±0.19
ResNet-101	7.8	44.6	16.2	90	20.4	21.4±0.11 [22.0]
ResNeXt-101	8.0	44.2	21.2	137	31.8	**20.7**±0.08 [21.2]
RegNetX-8.0GF	8.0	39.6	14.1	94	22.6	**20.7**±0.07
ResNet-152	11.5	60.2	22.6	130	29.2	20.9±0.12 [21.6]
ResNeXt-152	11.7	60.0	29.7	197	45.7	20.4±0.06 [21.1]
RegNetX-12GF	12.1	46.1	21.4	137	32.9	**20.3**±0.04

(b) Comparisons grouped by **flops**.

Table 3. **ResNe(X)t comparisons.** (a) Grouped by activations, RegNetX show considerable gains (note that for each group GPU inference and training times are similar). (b) RegNetX models outperform ResNe(X)t models under fixed flops as well.

5.2. Standard Baselines Comparison: ResNe(X)t

Next, we compare RegNetX to standard ResNet [8] and ResNeXt [31] models. All of the models in this experiment come from the exact same design space, the former being manually designed, the latter being obtained through design space design. For fair comparisons, we compare RegNet and ResNe(X)t models under the same training setup (our standard RegNet training setup). We note that this results in improved ResNe(X)t baselines and highlights the importance of carefully controlling the training setup.

Comparisons are shown in Figure 17 and Table 3. Overall, we see that RegNetX models, by optimizing the network structure alone, provide considerable improvements under all complexity metrics. We emphasize that good RegNet models are available across a wide range of compute regimes, including in low-compute regimes where good ResNe(X)t models are not available.

Table 3a shows comparisons grouped by activations (which can strongly influence runtime on accelerators such as GPUs). This setting is of particular interest to the research community where model training time is a bottleneck and will likely have more real-world use cases in the future, especially as accelerators gain more use at inference time (*e.g.*, in self-driving cars). RegNetX models are quite effective given a fixed inference or training time budget.

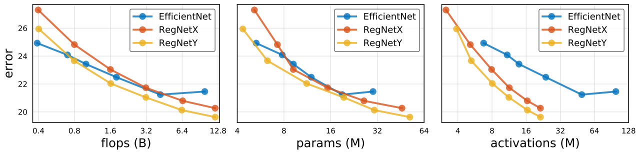


Figure 18. **EfficientNet comparisons.** RegNets outperform the state of the art, especially when considering activations.

	flops (B)	params (M)	acts (M)	batch size	infer (ms)	train (hr)	top-1 error ours±std [orig]
EfficientNet-B0	0.4	5.3	6.7	256	34	11.7	**24.9**±0.03 [23.7]
RegNetY-400MF	0.4	4.3	**3.9**	1024	19	5.1	25.9±0.16
EfficientNet-B1	0.7	7.8	10.9	256	52	15.6	**24.1**±0.16 [21.2]
RegNetY-600MF	0.6	6.1	**4.3**	1024	19	5.2	24.5±0.07
EfficientNet-B2	1.0	9.2	13.8	256	68	18.4	**23.4**±0.06 [20.2]
RegNetY-800MF	0.8	6.3	**5.2**	1024	22	6.0	23.7±0.03
EfficientNet-B3	1.8	12.0	23.8	256	114	32.1	22.5±0.05 [18.9]
RegNetY-1.6GF	1.6	11.2	**8.0**	1024	39	10.1	**22.0**±0.08
EfficientNet-B4	4.2	19.0	48.5	128	240	65.1	21.2±0.06 [17.4]
RegNetY-4.0GF	4.0	20.6	**12.3**	512	68	16.8	**20.6**±0.08
EfficientNet-B5	9.9	30.0	98.9	64	504	135.1	21.5±0.11 [16.7]
RegNetY-8.0GF	8.0	39.2	**18.0**	512	113	28.1	**20.1**±0.09

Table 4. **EfficientNet comparisons** using our standard training schedule. Under *comparable training settings*, RegNetY outperforms EfficientNet for most flop regimes. Moreover, RegNet models are considerably faster, *e.g.*, RegNetX-F8000 is about *5× faster* than EfficientNet-B5. Note that originally reported errors for EfficientNet (shown grayed out), are much lower but use longer and enhanced training schedules, see Table 7.

5.3. State-of-the-Art Comparison: Full Regime

We focus our comparison on EfficientNet [29], which is representative of the state of the art and has reported impressive gains using a combination of NAS and an interesting model scaling rule across complexity regimes.

To enable direct comparisons, and to isolate gains due to improvements solely of the *network architecture*, we opt to reproduce the exact EfficientNet models but using our standard training setup, with a 100 epoch schedule and no regularization except weight decay (effect of longer schedule and stronger regularization are shown in Table 7). We optimize only lr and wd, see Figure 22 in appendix. This is the same setup as RegNet and enables fair comparisons.

Results are shown in Figure 18 and Table 4. At low flops, EfficientNet outperforms the RegNetY. At intermediate flops, RegNetY outperforms EfficientNet, and at higher flops both RegNetX and RegNetY perform better.

We also observe that for EfficientNet, activations scale *linearly* with flops (due to the scaling of both resolution and depth), compared to activations scaling with the *square-root* of flops for RegNets. This leads to slow GPU training and inference times for EfficientNet. *E.g.*, RegNetX-8000 is *5× faster* than EfficientNet-B5, while having lower error.

6. Conclusion

In this work, we present a new network design paradigm. Our results suggest that *designing network design spaces* is a promising avenue for future research.

	flops (B)	params (M)	acts (M)	infer (ms)	train (hr)	error (top-1)
ResNet-50	4.1	22.6	11.1	53	12.2	35.0±0.20
RegNetX-3.2GF	3.2	15.3	11.4	57	14.3	**33.6**±0.25
ResNeXt-50	4.2	25.0	14.4	78	18.0	33.5±0.10
ResNet-101	7.8	44.6	16.2	90	20.4	33.2±0.24
RegNetX-6.4GF	6.5	26.2	16.4	92	23.5	**32.6**±0.15
ResNeXt-101	8.0	44.2	21.2	137	31.8	32.1±0.30
ResNet-152	11.5	60.2	22.6	130	29.2	32.2±0.22
RegNetX-12GF	12.1	46.1	21.4	137	32.9	**32.0**±0.27

(a) Comparisons grouped by **activations**.

	flops (B)	params (M)	acts (M)	infer (ms)	train (hr)	error (top-1)
ResNet-50	4.1	22.6	11.1	53	12.2	35.0±0.20
ResNeXt-50	4.2	25.0	14.4	78	18.0	33.5±0.10
RegNetX-4.0GF	4.0	22.1	12.2	69	17.1	**33.2**±0.20
ResNet-101	7.8	44.6	16.2	90	20.4	33.2±0.24
ResNeXt-101	8.0	44.2	21.2	137	31.8	**32.1**±0.30
RegNetX-8.0GF	8.0	39.6	14.1	94	22.6	32.5±0.18
ResNet-152	11.5	60.2	22.6	130	29.2	32.2±0.22
ResNeXt-152	11.7	60.0	29.7	197	45.7	**31.5**±0.26
RegNetX-12GF	12.1	46.1	21.4	137	32.9	32.0±0.27

(b) Comparisons grouped by **flops**.

Table 5. **ResNe(X)t comparisons on ImageNetV2.**

	flops (B)	params (M)	acts (M)	batch size	infer (ms)	train (hr)	error (top-1)
EfficientNet-B0	0.4	5.3	6.7	256	34	11.7	**37.1**±0.22
RegNetY-400MF	0.4	4.3	**3.9**	1024	19	5.1	38.3±0.26
EfficientNet-B1	0.7	7.8	10.9	256	52	15.6	**36.4**±0.10
RegNetY-600MF	0.6	6.1	**4.3**	1024	19	5.2	36.9±0.17
EfficientNet-B2	1.0	9.2	13.8	256	68	18.4	**35.3**±0.25
RegNetY-800MF	0.8	6.3	**5.2**	1024	22	6.0	35.7±0.40
EfficientNet-B3	1.8	12.0	23.8	256	114	32.1	34.4±0.27
RegNetY-1.6GF	1.6	11.2	**8.0**	1024	39	10.1	**33.9**±0.19
EfficientNet-B4	4.2	19.0	48.5	128	240	65.1	32.5±0.23
RegNetY-4.0GF	4.0	20.6	**12.3**	512	68	16.8	**32.3**±0.28
EfficientNet-B5	9.9	30.0	98.9	64	504	135.1	31.5±0.17
RegNetY-8.0GF	8.0	39.2	**18.0**	512	113	28.1	**31.3**±0.08

Table 6. **EfficientNet comparisons on ImageNetV2**.

Appendix A: Test Set Evaluation

In the main paper we perform all experiments on the ImageNet [3] validation set. Here we evaluate our models on the ImageNetV2 [24] test set (original test set unavailable).

Evaluation setup. To study generalization of models developed on ImageNet, the authors of [24] collect a new test set following the original procedure (ImageNetV2). They find that the overall model ranks are preserved on the new test set. The absolute errors, however, increase. We repeat the comparisons from §5 on the ImageNetV2 test set.

ResNe(X)t comparisons. We compare to ResNe(X)t models in Table 5. We observe that while model ranks are generally consistent, the gap between them decreases. Nevertheless, RegNetX models still compare favorably, and provide good models *across flop regimes*, including in low-compute regimes where good ResNe(X)t models are not available. Best results can be achieved using RegNetY.

EfficientNet comparisons. We compare to EfficientNet models in Table 6. As before, we observe that the model ranks are generally consistent but the gap decreases. Overall, the results confirm that the RegNet models perform comparably to state-of-the-art EfficientNet while being up to 5× *faster* on GPUs.

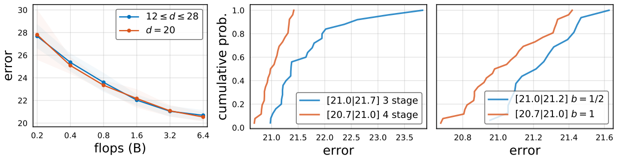


Figure 19. **Additional ablations**. *Left:* Fixed depth networks ($d = 20$) are effective across flop regimes. *Middle:* Three stage networks perform poorly at high flops. *Right:* Inverted bottleneck ($b < 1$) is also ineffective at high flops. See text for more context.

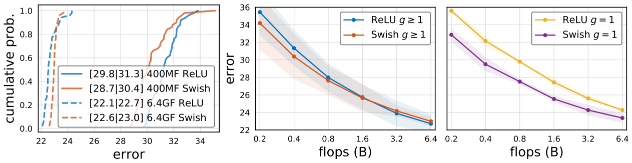


Figure 20. **Swish *vs*. ReLU.** *Left:* `RegNetY` performs better with Swish than ReLU at 400MF but worse at 6.4GF. *Middle:* Results across wider flop regimes show similar trends. *Right:* If, however, g is restricted to be 1 (depthwise conv), Swish is much better.

Appendix B: Additional Ablations

In this section we perform additional ablations to further support or supplement the results of the main text.

Fixed depth. In §5 we observed that the depths of our top models are fairly stable (∼20 blocks). In Figure 19 (left) we compare using fixed depth ($d = 20$) across flop regimes. To compare to our best results, we trained each model for 100 epochs. Surprisingly, we find that *fixed-depth networks can match the performance of variable depth networks* for all flop regimes, in both the average and best case. Indeed, these fixed depth networks match our best results in §5.

Fewer stages. In §5 we observed that the top RegNet models at high flops have few blocks in the fourth stage (one or two). Hence we tested 3 stage networks at 6.4GF, trained for 100 epochs each. In Figure 19 (middle), we show the results and observe that the three stage networks perform considerably worse. We note, however, that additional changes (*e.g.*, in the stem or head) may be necessary for three stage networks to perform well (left for future work).

Inverted Bottleneck. In §4 we observed that using the inverted bottleneck ($b < 1$) degrades performance. Since our results were in a low-compute regime, in Figure 19 (right) we re-test at 6.4GF and 100 epochs. Surprisingly, we find that in this regime $b < 1$ degrades results further.

Swish *vs*. ReLU Many recent methods employ the Swish [22] activation function, *e.g*. [29]. In Figure 20, we study `RegNetY` with Swish and ReLU. We find that Swish outperforms ReLU at low flops, but ReLU is better at high flops. Interestingly, if g is restricted to be 1 (depthwise conv), Swish performs much better than ReLU. This suggests that depthwise conv and Swish interact favorably, although the underlying reason is not at all clear.

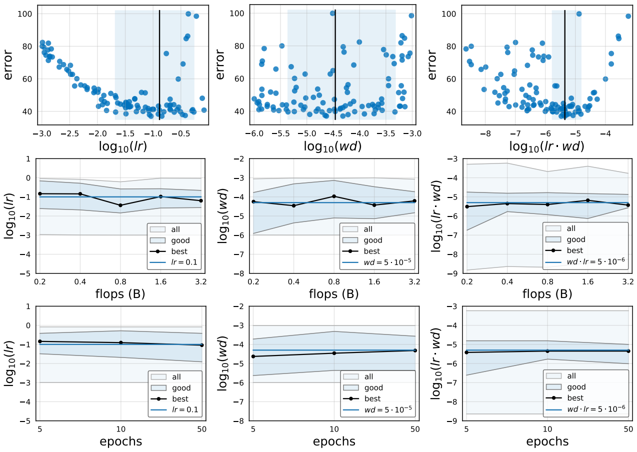


Figure 21. **Optimization settings.** For these results, we generate a population of `RegNetX` models while also randomly varying the initial learning rate (lr) and weight decay (wd) for each model. These results use a batch size of 128 and are trained on 1 GPU. *Top*: Distribution of model error versus lr, wd, and also $lr{\cdot}wd$ (at 10 epochs and 400MF). Applying an empirical bootstrap, we see that clear trends emerge, especially for lr and $lr{\cdot}wd$. *Middle*: We repeat this experiment but across various flop regimes (trained for 10 epochs each); the trends are stable. *Bottom*: Similarly, we repeat the above while training for various number of epochs (in the 400MF regime), and observe the same trends. Based on these results, we use an $lr = 0.1$ and $wd = 5{\cdot}10^{-5}$ starting with §4 across all training schedules and flop regimes.

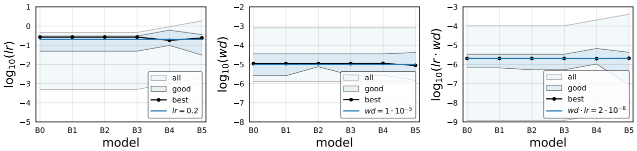


Figure 22. We repeat the sweep over lr **and** wd **for** **EFFICIENTNET** training each model for 25 epochs. The lr (with reference to a batch size of 128) and wd is stable across regimes. We use these values for all EFFICIENTNET experiments in the main text (adjusting for batch size accordingly). See Figure 21 for comparison.

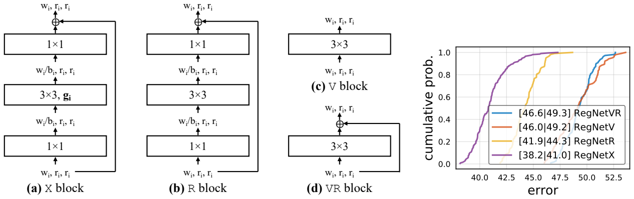
Appendix C: Optimization Settings

Our basic training settings follow [21] as discussed in §3. To tune the learning rate lr and weight decay wd for REGNET models, we perform a study, described in Figure 21. Based on this, we set $lr = 0.1$ and $wd = 5{\cdot}10^{-5}$ for all models in §4 and §5. To enable faster training of our final models at 100 epochs, we increase the number of GPUs to 8, while keeping the number of images per GPU fixed. When scaling the batch size, we adjust lr using the linear scaling rule and apply 5 epoch gradual warmup [6].

To enable fair comparisons, we repeat the same optimization for EFFICIENTNET in Figure 22. Interestingly, learning rate and weight decay are again stable across complexity regimes. Finally, in Table 7 we report the sizable effect of training enhancement on EFFICIENTNET-B0. The gap may be even larger for larger models (see Table 4).

	flops (B)	params (M)	acts (M)	epochs	enhance	error
EFFICIENTNET-B0	0.39	5.3	6.7	100		25.6
EFFICIENTNET-B0	0.39	5.3	6.7	250		25.0
EFFICIENTNET-B0	0.39	5.3	6.7	250	✓	24.4
EFFICIENTNET-B0 [29]	0.39	5.3	6.7	350	✓✓✓	23.7

Table 7. **Training enhancements** to EFFICIENTNET-B0. Our EFFICIENTNET-B0 reproduction with DropPath [14] and a 250 epoch training schedule (third row), achieves results slightly inferior to original results (bottom row), which additionally used RMSProp [30], AutoAugment [2], *etc*. Without these enhancements to the training setup results are ∼2% lower (top row), highlighting the importance of carefully controlling the training setup.



(a) `X` block **(b)** `R` block **(c)** `V` block **(d)** `VR` block

Figure 23. **Block types** used in our generalization experiments (see §3.4 and Figure 10). *Left:* Block diagrams. *Right:* Comparison of `RegNet` with the four block types. The `X` block performs best. Interestingly, for the `V` block, residual connections give no gain.

Appendix D: Implementation Details

We conclude with additional implementation details.

Group width compatibility. When sampling widths w and groups widths g for our models, we may end up with incompatible values (*i.e.* w not divisible by g). To address this, we employ a simple strategy. Namely, we set $g = w$ if $g > w$ and round w to be divisible by g otherwise. The final w can be at most 1/3 different from the original w (proof omitted). For models with bottlenecks, we apply this strategy to the bottleneck width instead (and adjust widths accordingly).

Group width ranges. As discussed in §4, we notice the general trend that the group widths of good models are larger in higher compute regimes. To account for this, we gradually adjust the group width ranges for higher compute regimes. For example, instead of sampling $g \leq 32$, at 3.2GF we use $16 \leq g \leq 64$ and allow any g divisible by 8.

Block types. In §3, we showed that the `RegNet` design space generalizes to different block types. We describe these additional block types, shown in Figure 23, next:

1. `R` block: same as the `X` block except without groups,
2. `V` block: a basic block with only a single 3×3 conv,
3. `VR` block: same as `V` block plus residual connections.

We note that good parameter values may differ across block types. *E.g.*, in contrast to the `X` block, for the `R` block using $b > 1$ is better than $b = 1$. Our approach is robust to this.

`Y` block details. To obtain the `Y` block, we add the SE op after the 3×3 conv of the `X` block, and we use an SE reduction ratio of $1/4$. We experimented with these choices but found that they performed comparably (not shown).

References

[1] F. Chollet. Xception: Deep learning with depthwise separable convolutions. In *CVPR*, 2017. 7

[2] E. D. Cubuk, B. Zoph, D. Mane, V. Vasudevan, and Q. V. Le. AutoAugment: Learning augmentation policies from data. *arXiv:1805.09501*, 2018. 8, 11

[3] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009. 2, 3, 8, 10

[4] T. DeVries and G. W. Taylor. Improved regularization of convolutional neural networks with cutout. *arXiv:1708.04552*, 2017. 8

[5] B. Efron and R. J. Tibshirani. *An introduction to the bootstrap*. CRC press, 1994. 3

[6] P. Goyal, P. Dollár, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv:1706.02677*, 2017. 11

[7] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *ICCV*, 2015. 2

[8] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 1, 2, 3, 9

[9] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv:1704.04861*, 2017. 2, 8

[10] J. Hu, L. Shen, and G. Sun. Squeeze-and-excitation networks. In *CVPR*, 2018. 7

[11] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *CVPR*, 2017. 2

[12] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015. 4

[13] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012. 1, 2

[14] G. Larsson, M. Maire, and G. Shakhnarovich. Fractalnet: Ultra-deep neural networks without residuals. In *ICLR*, 2017. 8, 11

[15] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1989. 1

[16] C.-Y. Lee, S. Xie, P. Gallagher, Z. Zhang, and Z. Tu. Deeply-supervised nets. In *AISTATS*, 2015. 8

[17] C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy. Progressive neural architecture search. In *ECCV*, 2018. 2, 8

[18] H. Liu, K. Simonyan, and Y. Yang. Darts: Differentiable architecture search. In *ICLR*, 2019. 1, 2, 8

[19] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *ECCV*, 2018. 8

[20] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean. Efficient neural architecture search via parameter sharing. In *ICML*, 2018. 2

[21] I. Radosavovic, J. Johnson, S. Xie, W.-Y. Lo, and P. Dollár. On network design spaces for visual recognition. In *ICCV*, 2019. 1, 2, 3, 4, 11

[22] P. Ramachandran, B. Zoph, and Q. V. Le. Searching for activation functions. *arXiv:1710.05941*, 2017. 10

[23] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le. Regularized evolution for image classifier architecture search. In *AAAI*, 2019. 2, 8

[24] B. Recht, R. Roelofs, L. Schmidt, and V. Shankar. Do imagenet classifiers generalize to imagenet? *arXiv:1902.10811*, 2019. 2, 10

[25] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *CVPR*, 2018. 2, 7, 8

[26] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015. 1, 2

[27] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *CVPR*, 2015. 2

[28] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. In *CVPR*, 2016. 2

[29] M. Tan and Q. V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *ICML*, 2019. 1, 2, 7, 9, 10, 11

[30] T. Tieleman and G. Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *Coursera: Neural networks for machine learning*, 2012. 11

[31] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He. Aggregated residual transformations for deep neural networks. In *CVPR*, 2017. 2, 4, 9

[32] S. Zagoruyko and N. Komodakis. Wide residual networks. In *BMVC*, 2016. 2

[33] X. Zhang, X. Zhou, M. Lin, and J. Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *CVPR*, 2018. 8

[34] B. Zoph and Q. V. Le. Neural architecture search with reinforcement learning. In *ICLR*, 2017. 1

[35] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le. Learning transferable architectures for scalable image recognition. In *CVPR*, 2018. 2, 8