On Network Design Spaces for Visual Recognition

Ilija Radosavovic   Justin Johnson   Saining Xie   Wan-Yen Lo   Piotr Dollár

Facebook AI Research (FAIR)

Abstract

Over the past several years progress in designing better neural network architectures for visual recognition has been substantial. To help sustain this rate of progress, in this work we propose to reexamine the methodology for comparing network architectures. In particular, we introduce a new comparison paradigm of distribution estimates, *in which network design spaces are compared by applying statistical techniques to populations of sampled models, while controlling for confounding factors like network complexity. Compared to current methodologies of comparing point and curve estimates of model families, distribution estimates paint a more complete picture of the entire design landscape. As a case study, we examine design spaces used in neural architecture search (NAS). We find significant statistical differences between recent NAS design space variants that have been largely overlooked. Furthermore, our analysis reveals that the design spaces for standard model families like ResNeXt can be comparable to the more complex ones used in recent NAS work. We hope these insights into distribution analysis will enable more robust progress toward discovering better networks for visual recognition.*

1. Introduction

Our community has made substantial progress toward designing better convolutional neural network architectures for visual recognition tasks over the past several years. This overall research endeavor is analogous to a form of *stochastic gradient descent* where every new proposed model architecture is a noisy gradient step traversing the infinite-dimensional landscape of possible neural network designs. The overall objective of this optimization is to find network architectures that are easy to optimize, make reasonable tradeoffs between speed and accuracy, generalize to many tasks and datasets, and overall withstand the test of time. To make continued steady progress toward this goal, we must use the right *loss function* to guide our search—in other words, a research methodology for comparing network architectures that can reliably tell us whether newly proposed models are truly better than what has come before.
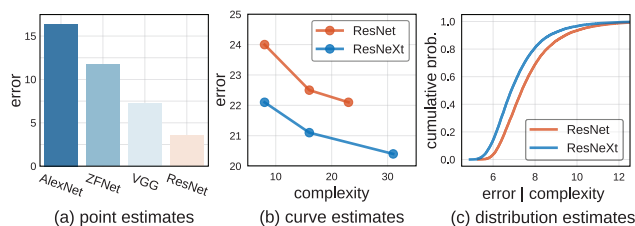

Figure 1. **Comparing networks.** (a) Early work on neural networks for visual recognition tasks used *point estimates* to compare architectures, often *irrespective of model complexity*. (b) More recent work compares *curve estimates* of error *vs.* complexity traced by a handful of selected models. (c) We propose to *sample* models from a parameterized model design space, and measure *distribution estimates* to compare design spaces. This methodology allows for a more complete and unbiased view of the design landscape.

One promising avenue lies in developing better *theoretical* understanding of neural networks to guide the development of novel network architectures. However, we need not wait for a general theory of neural networks to emerge to make continued progress. Classical statistics provides tools for drawing informed conclusions from *empirical* studies, even in the absence of a generalized theory governing the subject at hand. We believe that making use of such statistically grounded scientific methodologies in deep learning research may facilitate our future progress.

Overall, there has already been a general trend toward better empiricism in the literature on network architecture design. In the simplest and earliest methodology in this area (Figure 1a), progress was marked by simple *point estimates*: an architecture was deemed superior if it achieved lower error on a benchmark dataset [15, 19, 35, 31], often *irrespective of model complexity*.

An improved methodology adopted in more recent work compares *curve estimates* (Figure 1b) that explore design tradeoffs of network architectures by instantiating a handful of models from a loosely defined *model family* and tracing curves of error *vs.* model complexity [34, 11, 39]. A model family is then considered superior if it achieves lower error at every point along such a curve. Note, however, that in this methodology other *confounding factors* may vary between model families or may be suboptimal for one of them.

Comparing model families while varying a *single degree of freedom* to generate curve estimates hints at a more general methodology. For a given model family, rather than varying a single network hyperparameter (*e.g.* network depth) while keeping all others fixed (*e.g.* stagewise width, groups), what if instead *we vary all relevant network hyperparameters*? While in principle this would remove confounding factors that may affect conclusions about a model family, it would also yield a vast—often infinite—number of possible models. Are comparisons of model families under such unconstrained conditions even feasible?

To move toward such more robust settings, we introduce a new comparison paradigm: that of *distribution estimates* (Figure 1c). Rather than comparing a few selected members of a model family, we instead *sample* models from a *design space* parameterizing possible architectures, giving rise to *distributions* of error rates and model complexities. We then compare network design spaces by applying statistical techniques to these distributions, while controlling for confounding factors like network complexity. This paints a more complete and unbiased picture of the design landscape than is possible with point or curve estimates.

To validate our proposed methodology we perform a large-scale empirical study, *training over 100,000 models* spanning multiple model families on CIFAR [14]. This large set of trained models allows us to perform *simulations* of distribution estimates and draw robust conclusions about our methodology. In practice, however, we show that sampling between 100 to 1000 models is sufficient to perform robust estimates. We further validate our estimates by performing a study on ImageNet [4], see the arXiv version of this work[1]. This makes the proposed methodology feasible under typical settings and thus a practical tool that can be used to aid in the discovery of novel network architectures.

As a case study of our methodology, we examine the network design spaces used by several recent methods for neural architecture search (NAS) [39, 29, 20, 28, 21]. Surprisingly, we find that there are significant differences between the design spaces used by different NAS methods, and we hypothesize that these differences may explain some of the performance improvements between these methods. Furthermore, we demonstrate that design spaces for standard model families such as ResNeXt [34] can be comparable to the more complex ones used in recent NAS methods.

We note that our work complements NAS. Whereas NAS is focused on finding the single best model in a given model family, our work focuses on characterizing the model family itself. In other words, our methodology may enable research into *designing the design space* for model search.

We have released all code[2] and data[3] used in this study.

---

2. Related Work

Reproducible research. There has been an encouraging recent trend toward better reproducibility in machine learning [25, 22, 9]. For example, Henderson *et al.* [9] examine recent research in reinforcement learning (RL) and propose guidelines to improve reproducibility and thus enable continued progress in the field. Likewise, we share the goal of introducing a more robust methodology for evaluating model architectures in the domain of visual recognition.

Empirical studies. In the absence of rigorous theoretical understanding of deep networks, it is imperative to perform large-scale empirical studies of deep networks to aid development [6, 3, 27]. For example, in natural language processing, recent large-scale studies [25, 26] demonstrate that when design spaces are well explored, the original LSTM [10] can outperform more recent models on language modeling benchmarks. These results suggest the crucial role empirical studies and robust methodology play in enabling progress toward discovering better architectures.

Hyperparameter search. General hyperparameter search techniques [2, 32] address the laborious model tuning process in machine learning. A possible approach for comparing networks from two different model families is to first tune their hyperparameters [16]. However, such comparisons can be challenging in practice. Instead, [1] advocates using random search as a strong baseline for hyperparameter search and suggests that it additionally helps improve reproducibility. In our work we propose to directly compare the *full* model distributions (not just their minima).

Neural architecture search. Recently, NAS has proven effective for learning networks architectures [38]. A NAS instantiation has two components: a network *design space* and a *search algorithm* over that space. Most work on NAS focuses on the *search algorithm*, and various search strategies have been studied, including RL [38, 39, 28], heuristic search [20], gradient-based search [21, 23], and evolutionary algorithms [29]. Instead, in our work, we focus on characterizing the model *design space*. As a case study we analyze recent NAS design spaces [39, 29, 20, 28, 21] and find significant differences that have been largely overlooked.

Complexity measures. In this work we focus on analyzing network design spaces while controlling for confounding factors like network complexity. While statistical learning theory [30, 33] has introduced theoretical notions of complexity of machine learning models, these are often not predictive of neural network behavior [36, 37]. Instead, we adopt commonly-used network complexity measures, including the number of model parameters or multiply-add operations [7, 34, 11, 39]. Other measures, *e.g.* wall-clock speed [12], can easily be integrated into our paradigm.

stage	operation	output		R-56	R-110
stem	3×3 conv	32×32×16	flops (B)	0.13	0.26
stage 1	{block}×d_1	32×32×w_1	params (M)	0.86	1.73
stage 2	{block}×d_2	16×16×w_2	error [8]	6.97	6.61
stage 3	{block}×d_3	8×8×w_3	error [ours]	6.22	5.91
head	pool + fc	1×1×10			

Table 1. **Design space parameterization**. *(Left)* The general network structure for the standard model families used in our work. Each stage consists of a sequence of d blocks with w output channels (block type varies between design spaces). *(Right)* Statistics of ResNet models [8] for reference. In our notation, R-56 has $d_i=9$ and $w_i=8\cdot2^i$ and R-110 doubles the blocks per stage d_i. We report original errors from [8] and those in our reproduction.

	depth	width	ratio	groups	total
Vanilla	1,24,9	16,256,12			1,259,712
ResNet	1,24,9	16,256,12			1,259,712
ResNeXt-A	1,16,5	16,256,5	1,4,3	1,4,3	11,390,625
ResNeXt-B	1,16,5	64,1024,5	1,4,3	1,16,5	52,734,375

Table 2. **Design spaces.** Independently for each of the three network stages i, we select the number of blocks d_i and the number of channels per block w_i. Notation a, b, n means we sample from n values spaced about evenly (in log space) in the range a to b. For the ResNeXt design spaces, we also select the bottleneck width ratio r_i and the number of groups g_i per stage. The total number of models is $(dw)^3$ and $(dwrg)^3$ for models w/o and with groups.

3. Design Spaces

We begin by describing the core concepts defining a design space in §3.1 and give more details about the actual design spaces used in our experiments in §3.2.

3.1. Definitions

I. Model family. A *model family* is a large (possibly infinite) collection of related neural network architectures, typically sharing some high-level architectural structures or design principles (*e.g.* residual connections). Example model families include standard feedforward networks such as ResNets [8] or the NAS model family from [38, 39].

II. Design space. Performing empirical studies on model families is difficult since they are broadly defined and typically not fully specified. As such we make a distinction between abstract model families, and a *design space* which is a concrete set of architectures that can be instantiated from the model family. A design space consists of two components: a parametrization of a model family such that specifying a set of model hyperparameters fully defines a network instantiation and a set of allowable values for each hyperparameter. For example, a design space for the ResNet model family could include a parameter controlling network depth and a limit on its maximum allowable value.

III. Model distribution. To perform empirical studies of design spaces, we must instantiate and evaluate a set of network architectures. As a design space can contain an exponential number of candidate models, exhaustive evaluation is not feasible. Therefore, we sample and evaluate a fixed set of models from a design space, giving rise to a *model distribution*, and turn to tools from classical statistics for analysis. Any standard distribution, as well as learned distributions like in NAS, can be integrated into our paradigm.

IV. Data generation. To analyze network design spaces, we sample and evaluate numerous models from each design space. In doing so, we effectively generate *datasets* of trained models upon which we perform empirical studies.

3.2. Instantiations

We provide precise description of design spaces used in the analysis of our methodology. We introduce additional design spaces for NAS model families in §5.

I. Model family. We study three standard model families. We consider a 'vanilla' model family (feedforward network loosely inspired by VGG [31]), as well as model families based on ResNet [8] and ResNeXt [34] which use residuals and grouped convolutions. We provide more details next.

II. Design space. Following [8], we use networks consisting of a stem, followed by three stages, and a head, see Table 1 (left). Each stage consists of a sequence of blocks. For our ResNet design space, a single block consists of two convolutions[4] and a residual connection. Our Vanilla design space uses an identical block structure but without residuals. Finally, in case of the ResNeXt design spaces, we use bottleneck blocks with groups [34]. Table 1 (right) shows some baseline ResNet models for reference (for details of the training setup see the arXiv). To complete the design space definitions, in Table 2 we specify the set of allowable hyperparameters for each. Note that we consider two variants of the ResNeXt design spaces with different hyperparameter sets: ResNeXt-A and ResNeXt-B.

III. Model distribution. We generate model distributions by *uniformly* sampling hyperparameters from the allowable values for each design spaces (as specified in Table 2).

IV. Data generation. Our main experiments use image classification models trained on CIFAR-10 [14]. This setting enables large-scale analysis and is often used as a testbed for recognition networks, including for NAS. While we find that sparsely sampling models from a given design space is sufficient to obtain robust estimates, we perform much denser sampling *to evaluate our methodology*. We sample and train 25k models from each of the design spaces from Table 2, for a total of 100k models. To reduce computational load, we consider models for which the flops[5] or parameters are below the ResNet-56 values (Table 1, right).

---

[4]All convs are 3×3 and are followed by Batch Norm [13] and ReLU.
[5]Following common practice, we use flops to mean multiply-adds.

4. Proposed Methodology

In this section we introduce and evaluate our methodology for comparing design spaces. Throughout this section we use the design spaces introduced in §3.2.

4.1. Comparing Distributions

When developing a new network architecture, human experts employ a combination of grid and manual search to evaluate models from a design space, and select the model achieving the lowest error (*e.g.* as described in [16]). The final model is a *point estimate* of the design space. As a community we commonly use such point estimates to draw conclusions about which methods are superior to others.

Unfortunately, comparing design spaces via point estimates can be misleading. We illustrate this using a simple example: we consider comparing two sets of models of *different sizes* sampled from the *same design space*.

Point estimates. As a proxy for human derived point estimates, we use random search [1]. We generate a *baseline* model set (B) by uniformly sampling 100 architectures from our `ResNet` design space (see Table 2). To generate the second model set (M), we instead use 1000 samples. In practice, the difference in number of samples could arise from more effort in the development of M over the baseline, or simply access to more computational resources for generating M. Such *imbalanced comparisons* are common in practice.

After training, M's *minimum error* is lower than B's *minimum error*. Since the best error is lower, a naive comparison of *point estimates* concludes that M is superior. Repeating this experiment yields the same result: Figure 2 (left) plots the difference in the minimum error of B and M over multiple trials (simulated by repeatedly sampling B and M from our pool of 25k pre-trained models). In 90% of cases M has a lower minimum than B, often by a large margin. Yet clearly B and M were drawn from the *same* design space, so this analysis based on point estimates can be misleading.

Distributions. In this work we make the case that one can draw more robust conclusions by directly comparing *distributions* rather than *point estimates* such as minimum error.

To compare distributions, we use *empirical distribution functions* (EDFs). Let **1** be the indicator function. Given a set of n models with errors $\{e_i\}$, the error EDF is given by:

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

$F(e)$ gives the fraction of models with error less than e.

We revisit our B *vs.* M comparison in Figure 2 (right), but this time plotting the full error distributions instead of just their minimums. Their shape is typical of error EDFs: the small tail to the bottom left indicates a small population of models with low error and the long tail on the upper right shows there are few models with error over 10%.
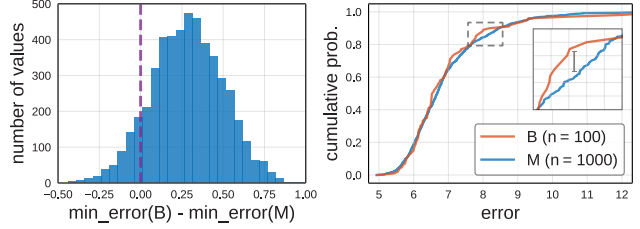

Figure 2. **Point *vs.* distribution comparison**. Consider two sets of models, B and M, generated by randomly sampling 100 and 1000 models, respectively, from the *same* design space. Such situations commonly arise in practice, *e.g.* due to more effort being devoted to model development of a new method. *(Left)* The difference in the *minimum* error of B and M over 5000 random trials. In 90% of cases M has lower minimum than B, leading to incorrect conclusions. *(Right)* Comparing EDFs of the errors (Eqn. 1) from B and M immediately suggests the distributions of the two sets are likely the same. We can quantitatively characterize this by measuring the KS statistic D (Eqn. 2), computed as the maximum vertical discrepancy between two EDFs, as shown in the zoomed in panel.

Qualitatively, there is little visible difference between the error EDFs for B and M, suggesting that these two sets of models were drawn from an identical design space. We can make this comparison *quantitative* using the (two sample) Kolmogorov-Smirnov (KS) test [24], a nonparametric statistical test for the null hypothesis that two samples were drawn from the same distribution. Given EDFs F_1 and F_2, the test computes the KS statistic D, defined as:

$$D = \sup_x |F_1(x) - F_2(x)| \qquad (2)$$

D measures the maximum vertical discrepancy between EDFs (see the zoomed in panel in Figure 2); small values suggest that F_1 and F_2 are sampled from the same distribution. For our example, the KS test gives $D = 0.079$ and a p-value of 0.60, so with high confidence we fail to reject the null hypothesis that B and M follow the same distribution.

Discussion. While pedagogical, the above example demonstrates the necessity of comparing distributions rather than point estimates, as the latter can give misleading results in even simple cases. We emphasize that such imbalanced comparisons occur frequently in practice. Typically, most work reports results for only a small number of best models, and rarely reports the number of total points explored during model development, which can vary substantially.

4.2. Controlling for Complexity

While comparing distributions can lead to more robust conclusions about design spaces, when performing such comparisons, we need to control for *confounding factors* that correlate with model error to avoid biased conclusions. A particularly relevant confounding factor is *model complexity*. We study controlling for complexity next.
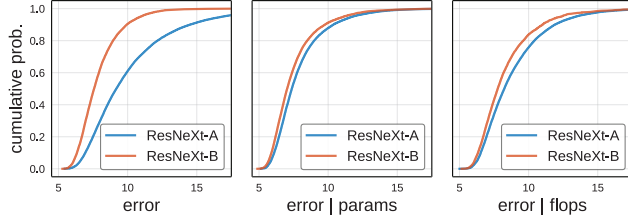
Figure 3. **Comparisons conditioned on complexity.** Without considering complexity, there is a large gap between the related `ResNeXt-A` and `ResNeXt-B` design spaces (left). We control for params (middle) and flops (right) to obtain distributions *conditioned* on complexity (Eqn. 4), which we denote by 'error | complexity'. Doing so removes most of the observed gap and isolates differences that cannot be explained by model complexity alone.

Unnormalized comparison. Figure 3 (left) shows the error EDFs for the `ResNeXt-A` and `ResNeXt-B` design spaces, which differ only in the allowable hyperparameter sets for sampling models (see Table 2). Nevertheless, the curves have clear qualitative differences and suggest that `ResNeXt-B` is a better design space. In particular, the EDF for `ResNeXt-B` is higher; *i.e.*, it has a higher fraction of better models at every error threshold. This clear difference illustrates that different design spaces from the *same model family* under the same model distribution can result in very different error distributions. We investigate this gap further.

Error vs complexity. From prior work we know that a model's error is related to its complexity; in particular more complex models are often more accurate [7, 34, 11, 39]. We explore this relationship using our large-scale data. Figure 4 plots the error of each trained model against its complexity, measured by either its parameter or flop counts. While there are poorly-performing high-complexity models, the *best models have the highest complexity*. Moreover, in this setting, we see no evidence of saturation: as complexity increases we continue to find better models.

Complexity distribution. Can the differences between the `ResNeXt-A` and `ResNeXt-B` EDFs in Figure 3 (left) be due to differences in their complexity distributions? In Figure 5, we plot the empirical distributions of model complexity for the two design spaces. We see that `ResNeXt-A` contains a much larger number of low-complexity models, while `ResNeXt-B` contains a heavy tail of high-complexity models. It therefore seems plausible that `ResNeXt-B`'s apparent superiority is due to the confounding effect of complexity.

Normalized comparison. We propose a *normalization* procedure to factor out the *confounding* effect of the differences in the complexity of model distributions. Given a set of n models where each model has complexity c_i, the idea is to assign to each model a weight w_i, where $\Sigma_i w_i = 1$, to create a more *representative* set under that complexity measure.

Specifically, given a set of n models with errors, complexities, and weights given by $\{e_i\}$, $\{c_i\}$, and $\{w_i\}$, re-
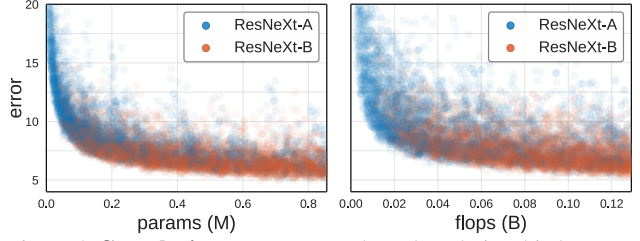

Figure 4. **Complexity *vs*. error.** We show the relationship between model complexity and performance for two different complexity metrics and design spaces. Each point is a trained model.
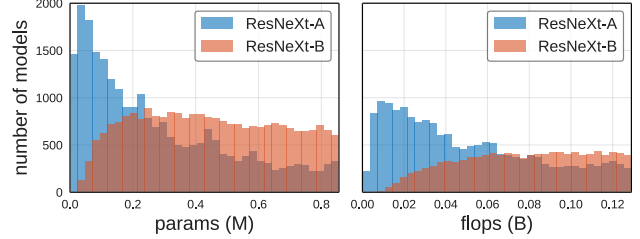

Figure 5. **Complexity distribution.** Two different ResNeXt design spaces have different complexity distributions. We need to account for this difference in order to be able to compare the two.

spectively, we define the *normalized complexity* EDF as:

$$C(c) = \sum_{i=1}^{n} w_i \mathbf{1}[c_i < c] \qquad (3)$$

Likewise, we define the *normalized error* EDF as:

$$\hat{F}(e) = \sum_{i=1}^{n} w_i \mathbf{1}[e_i < e]. \qquad (4)$$

Then, given two model sets, our goal is to find weights for each model set such that $C_1(c) \approx C_2(c)$ for all c in a given complexity range. Once we have such weights, comparisons between $\hat{F}_1$ and $\hat{F}_2$ reveal differences between design spaces that cannot be explained by model complexity alone.

In practice, we set the weights for a model set such that its complexity distribution (Eqn. 3) is *uniform*. Specifically, we bin the complexity range into k bins, and assign each of the m_j models that fall into a bin j a weight $w_j = 1/km_j$. Given sparse data, the assignment into bins could be made in a soft manner to obtain smoother estimates. While other options for matching $C_1 \approx C_2$ are possible, we found normalizing both C_1 and C_2 to be uniform to be effective.

In Figure 3 we show `ResNeXt-A` and `ResNeXt-B` error EDFs, normalized by parameters (middle) and flops (right). Controlling for complexity brings the curves closer, suggesting that much of the original gap was due to mismatched complexity distributions. This is not unexpected as the design spaces are similar and both parameterize the same underlying model family. We observe, however, that their normalized EDFs still show a small gap. We note that `ResNeXt-B` contains wider models with more groups (see Table 2), which may account for this remaining difference.
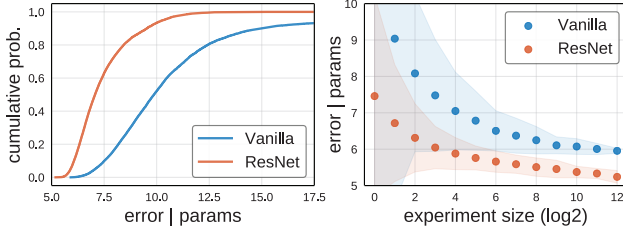
Figure 6. **Finding good models quickly.** *(Left)* The `ResNet` design space contains a much larger proportion of good models than the `Vanilla` design space, making *finding a good model quickly* easier. *(Right)* This can also be seen by measuring the number of model evaluations random search requires to reach a given error.

4.3. Characterizing Distributions

An advantage of examining the full error distribution of a design space is it gives insights beyond the minimum achievable error. Often, we indeed focus on finding the *best model* under some complexity constraint, for example, if a model will be deployed in a production system. In other cases, however, we may be interested in *finding a good model quickly*, *e.g.* when experimenting in a new domain or under constrained computation. Examining distributions allows us to more fully characterize a design space.

Distribution shape. Figure 6 (left) shows EDFs for the `Vanilla` and `ResNet` design spaces (see Table 2). In the case of `ResNet`, the majority ($>80\%$) of models have error under 8%. In contrast, the `Vanilla` design space has a much smaller fraction of such models ($\sim 15\%$). This makes it easier to find a good `ResNet` model. While this is not surprising given the well-known effectiveness of residual connections, it does demonstrate how the shape of the EDFs can give additional insight into characterizing a design space.

Distribution area. We can summarize an EDF by the average area under the curve up to some max ϵ. That is we can compute $\int_0^\epsilon \hat{F}(e)/\epsilon \, \mathrm{d}e = 1 - \sum w_i \min(1, \frac{e_i}{\epsilon})$. For our example, `ResNet` has a larger area under the curve. However, like the min, the area gives only a partial view of the EDF.

Random search efficiency. Another way to assess the ease of finding a good model is to measure random search efficiency. To *simulate random search* experiments of varying size m we follow the procedure described in [1]. Specifically, for each experiment size m, we sample m models from our pool of n models and take their *minimum* error. We repeat this process $\sim n/m$ times to obtain the mean along with error bars for each m. To factor out the confounding effect of complexity, we assign a weight to each model such that $C_1 \approx C_2$ (Eqn. 3) and use these weights for sampling.

In Figure 6 (right), we use our 50k pre-trained models from the `Vanilla` and `ResNet` design spaces to simulate random search (conditioned on parameters) for varying m. We observe consistent findings as before: random search finds better models faster in the `ResNet` design space.
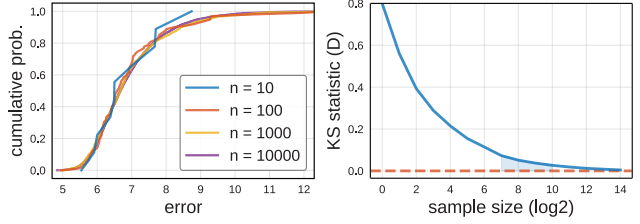

Figure 7. **Number of samples.** We analyze the number of samples required for performing analysis of design spaces using our methodology. *(Left)* We show a qualitative comparison of EDFs generated using a varying number of samples. *(Right)* We compute KS statistic between the full sample and sub-samples of increasing size. In both cases, we arrive at the conclusion that between 100 and 1000 samples is a reasonable range for our methodology.

4.4. Minimal Sample Size

Our experiments thus far used very large sets of trained models. In practice, however, far fewer samples can be used to compare distributions of models as we now demonstrate.

Qualitative analysis. Figure 7 (left) shows EDFs for the `ResNet` design space with varying number of samples. Using 10 samples to generate the EDF is quite noisy; however, 100 gives a reasonable approximation and 1000 is visually indistinguishable from 10,000. This suggests that 100 to 1000 samples may be sufficient to compare distributions.

Quantitative analysis. We perform quantitative analysis to give a more precise characterization of the number of samples necessary to compare distributions. In particular, we compute the KS statistic D (Eqn. 2) between the full sample of 25k models and sub-samples of increasing size n. The results are shown in Figure 7 (right). As expected, as n increases, D decreases. At 100 samples D is about 0.1, and at 1000 D begins to saturate. Beyond 1000 samples shows diminishing returns. Thus, our earlier estimate of 100 samples is indeed a reasonable lower-bound, and 1000 should be sufficient for more precise comparisons. We note, however, that these bounds may vary under other circumstances.

Feasibility discussion. One might wonder about the feasibility of training between 100 and 1000 models for evaluating a distribution. In our setting, training 500 CIFAR models requires about 250 GPU hours. In comparison, training a typical ResNet-50 baseline on ImageNet requires about 192 GPU hours. Thus, evaluating the full distribution for a small-sized problem like CIFAR requires a computational budget on par with a point estimate for a medium-sized problem like ImageNet. To put this in further perspective, NAS methods can require as much as $O(10^5)$ GPU hours on CIFAR [28]. Overall, we expect distribution comparisons to be quite feasible under typical settings. To further aid such comparisons, we have released data for all studied design spaces to serve as baselines for future work.

	#ops	#nodes	output	#cells (B)
NASNet [39]	13	5	L	71,465,842
Amoeba [29]	8	5	L	556,628
PNAS [20]	8	5	A	556,628
ENAS [28]	5	5	L	5,063
DARTS [21]	8	4	A	242

Table 3. **NAS design spaces.** We summarize the *cell structure* for five NAS design spaces. We list the number of candidate ops (*e.g.* 5×5 conv, 3×3 max pool, *etc.*), number of nodes (excluding the inputs), and which nodes are concatenated for the output ('A' if 'all' nodes, 'L' if 'loose' nodes not used as input to other nodes). Given o ops to choose from, there are $o^2 \cdot (j{+}1)^2$ choices when adding the jth node, leading to $o^{2k} \cdot ((k{+}1)!)^2$ possible cells with k nodes (of course many of these cells are redundant). The spaces vary substantially; indeed, even exact candidate ops for each vary.

5. Case Study: NAS

As a case study of our methodology we examine design spaces from recent neural architecture search (NAS) literature. In this section we perform studies on CIFAR [14] and in the arXiv we further validate our results by replicating the study on ImageNet [4], yielding similar conclusions.

NAS has two core components: a *design space* and a *search algorithm* over that space. While normally the focus is on the search algorithm (which can be viewed as inducing a distribution over the design space), we instead focus on comparing the design spaces under a fixed distribution. Our main finding is that in recent NAS papers, significant design space differences have been largely overlooked. Our approach complements NAS by *decoupling* the design of the design space from the design of the search algorithm, which we hope will aid the study of new design spaces.

5.1. Design Spaces

I. Model family. The general NAS model family was introduced in [38, 39]. A NAS model is constructed by repeatedly stacking a single computational unit, called a *cell*, where a cell can vary in the operations it performs and in its connectivity pattern. In particular, a cell takes outputs from two previous cells as inputs and contains a number of *nodes*. Each node in a cell takes as input two previously constructed nodes (or the two cell inputs), applies an operator to each input (*e.g.* convolution), and combines the output of the two operators (*e.g.* by summing). We refer to ENAS [28] for a more detailed description.

II. Design space. In spite of many recent papers using the general NAS model family, most recent approaches use different design space instantiations. In particular, we carefully examined the design spaces described in NAS-Net [39], AmoebaNet [29], PNAS [20], ENAS [28], and DARTS [21]. The *cell structure* differs substantially between them, see Table 3 for details. In our work, we define five design spaces by reproducing these five cell structures, and name them accordingly, *i.e.*, NASNet, Amoeba, *etc*.
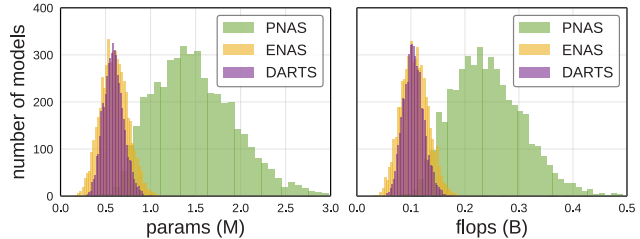

Figure 8. **NAS complexity distribution.** Complexity of design spaces with different cell structures (see Table 3) vary significantly given fixed width ($w{=}16$) and depth ($d{=}20$). To compare design spaces we need to normalize for complexity, which requires the complexity distributions to fall in the same range. We achieve this by allowing w and d to vary (and setting a max complexity).

How the cells are stacked to generate the full *network architecture* also varies slightly between recent papers, but less so than the cell structure. We therefore standardize this aspect of the design spaces; that is we adopt the network architecture setting from DARTS [21]. Core aspects include the stem structure, even placement of the three reduction cells, and filter width that doubles after each reduction cell.

The network depth d and initial filter width w are typically kept fixed. However, these hyperparameters directly affect model complexity. Specifically, Figure 8 shows the complexity distribution generated with different cell structures with w and d kept fixed. The ranges of the distributions differ due to the varying cell structures designs. To factor out this confounding factor, we let w and d vary (selecting $w \in \{16, 24, 32\}$ and $d \in \{4, 8, 12, 16, 20\}$). This spreads the range of the complexity distributions for each design space, allowing for more controlled comparisons.

III. Model distribution. We sample NAS cells by using uniform sampling at each step (*e.g.* operator and node selection). Likewise, we sample w and d uniformly at random.

IV. Data generation. We train ~1k models on CIFAR for each of the five NAS design spaces in Table 3 (see §4.4 for a discussion of sample size). In particular, we ensure we have 1k models per design space for both the full flop range and the full parameter range (upper-bounded by R-56, see §3).

5.2. Design Space Comparisons

We adopt our distribution comparison tools (EDFs, KS test, *etc.*) from §4 to compare the five NAS design spaces, each of which varies in its cell structure (see Table 3).

Distribution comparisons. Figure 9 shows normalized error EDFs for each of the NAS design spaces. Our main observation is that the EDFs vary considerably: the NASNet and Amoeba design spaces are noticeably worse than the others, while DARTS is best overall. Comparing ENAS and PNAS shows that while the two are similar, PNAS has more models with intermediate errors while ENAS has more lower/higher performing models, causing the EDFs to cross.
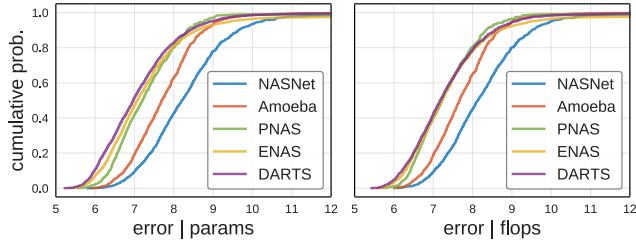
Figure 9. **NAS distribution comparisons.** EDFs for five NAS design spaces from Table 3. The EDFs differ substantially (max KS test $D = 0.51$ between DARTS and NASNet) even though the design spaces are all instantiations of the NAS model family.
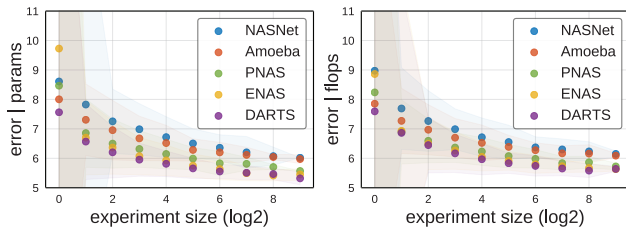

Figure 10. **NAS random search efficiency.** Design space differences lead to clear differences in the efficiency of random search on the five tested NAS design spaces. This highlights the importance of decoupling the *search algorithm* and *design space*.

Interestingly, according to our analysis the design spaces corresponding to newer work outperform the earliest design spaces introduced in NASNet [39] and Amoeba [29]. While the NAS literature typically focuses on the search algorithm, the design spaces also seem to be improving. For example, PNAS [20] removed five ops from NASNet that were not selected in the NASNet search, effectively pruning the design space. Hence, at least part of the gains in each paper may come from improvements of the design space.

Random search efficiency. We simulate random search in the NAS design spaces (after normalizing for complexity) following the setup from §4.3. Results are shown in Figure 10. First, we observe that ordering of design spaces by random search efficiency is consistent with the ordering of the EDFs in Figure 9. Second, for a fixed search algorithm (random search in this case), this shows the differences in the design spaces alone leads to clear differences in performance. This reinforces that care should be taken to keep the design space fixed if the search algorithm is varied.

5.3. Comparisons to Standard Design Spaces

We next compare the NAS design spaces with the design spaces from §3. We select the best and worst performing NAS design spaces (DARTS and NASNet) and compare them to the two ResNeXt design spaces from Table 2. EDFs are shown in Figure 11. ResNeXt-B is on par with DARTS when normalizing by params (left), while DARTS outperforms ResNeXt-B when normalizing by flops (right). ResNeXt-A is worse than DARTS in both cases.
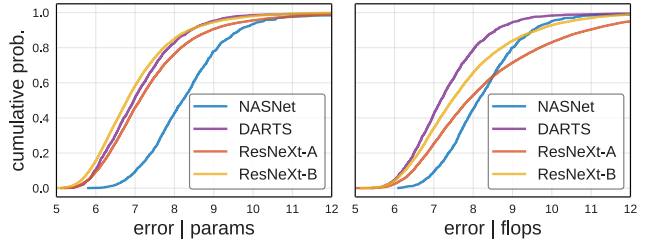

Figure 11. **NAS *vs*. standard design spaces.** Normalized for params, ResNeXt-B is comparable to the strong DARTS design space (KS test D=0.09). Normalized for flops, DARTS outperforms ResNeXt-B, but not by a relatively large margin.

	flops (B)	params (M)	error original	error default	error enhanced
ResNet-110	0.26	1.7	6.61	5.91	3.65
ResNeXt⋆	0.38	2.5	–	4.90	2.75
DARTS⋆	0.54	3.4	2.83	5.21	2.63

Table 4. **Point comparisons.** We compare selected higher complexity models using *originally* reported errors to our *default* training setup and the *enhanced* training setup from [21]. The results show the importance of carefully controlling the training setup.

It is interesting that ResNeXt design spaces *can be* comparable to the NAS design spaces (which vary in cell structure in addition to width and depth). These results demonstrate that the *design* of the design space plays a key role and suggest that *designing design spaces*, manually or via data-driven approaches, is a promising avenue for future work.

5.4. Sanity Check: Point Comparisons

We note that recent NAS papers report lower overall errors due to higher complexity models and *enhanced* training settings. As a *sanity check*, we perform *point comparisons* using larger models and the *exact* training settings from DARTS [21] which uses a 600 epoch schedule with deep supervision [18], Cutout [5], and modified DropPath [17]. We consider three models: DARTS⋆ (the best model found in DARTS [21]), ResNeXt⋆ (the best model from ResNeXt-B with increased widths), and ResNet-110 [8].

Results are shown in Table 4. With the *enhanced* setup, ResNeXt⋆ achieves similar error as DARTS⋆ (with comparable complexity). This reinforces that performing comparisons under the same settings is crucial, simply using an enhanced training setup gives over 2% gain; even the original ResNet-110 is competitive under these settings.

6. Conclusion

We present a methodology for analyzing and comparing model design spaces. Although we focus on convolutional networks for image classification, our methodology should be applicable to other model types (*e.g.* RNNs), domains (*e.g.* NLP), and tasks (*e.g.* detection). We hope our work will encourage the community to consider design spaces as a core part of model development and evaluation.

References

[1] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *JMLR*, 2012. 2, 4, 6

[2] James S Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *NIPS*, 2011. 2

[3] Jasmine Collins, Jascha Sohl-Dickstein, and David Sussillo. Capacity and trainability in recurrent neural networks. In *ICLR*, 2017. 2

[4] 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. 2, 7

[5] Terrance DeVries and Graham W. Taylor. Improved regularization of convolutional neural networks with cutout. *arXiv:1708.04552*, 2017. 8

[6] Klaus Greff, Rupesh K Srivastava, Jan Koutník, Bas R Steunebrink, and Jürgen Schmidhuber. Lstm: A search space odyssey. *arXiv:1503.04069*, 2015. 2

[7] Kaiming He and Jian Sun. Convolutional neural networks at constrained time cost. In *CVPR*, 2015. 2, 5

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 3, 8

[9] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. In *AAAI*, 2018. 2

[10] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 1997. 2

[11] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *CVPR*, 2017. 1, 2, 5

[12] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, et al. Speed/accuracy trade-offs for modern convolutional object detectors. In *CVPR*, 2017. 2

[13] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015. 3

[14] Alex Krizhevsky. Learning multiple layers of features from tiny images. *Tech Report*, 2009. 2, 3, 7

[15] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012. 1

[16] Hugo Larochelle, Dumitru Erhan, Aaron Courville, James Bergstra, and Yoshua Bengio. An empirical evaluation of deep architectures on problems with many factors of variation. In *ICML*, 2007. 2, 4

[17] Gustav Larsson, Michael Maire, and Gregory Shakhnarovich. Fractalnet: Ultra-deep neural networks without residuals. In *ICLR*, 2017. 8

[18] Chen-Yu Lee, Saining Xie, Patrick Gallagher, Zhengyou Zhang, and Zhuowen Tu. Deeply-supervised nets. In *AISTATS*, 2015. 8

[19] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *arXiv:1312.4400*, 2013. 1

[20] 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. 2, 7, 8

[21] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. In *ICLR*, 2019. 2, 7, 8

[22] Mario Lucic, Karol Kurach, Marcin Michalski, Sylvain Gelly, and Olivier Bousquet. Are gans created equal? a large-scale study. In *NIPS*, 2018. 2

[23] Renqian Luo, Fei Tian, Tao Qin, Enhong Chen, and Tie-Yan Liu. Neural architecture optimization. In *NIPS*, 2018. 2

[24] Frank J Massey Jr. The kolmogorov-smirnov test for goodness of fit. *Journal of the American statistical Association*, 1951. 4

[25] Gábor Melis, Chris Dyer, and Phil Blunsom. On the state of the art of evaluation in neural language models. In *ICLR*, 2018. 2

[26] Stephen Merity, Nitish Shirish Keskar, and Richard Socher. Regularizing and optimizing lstm language models. In *ICLR*, 2018. 2

[27] Roman Novak, Yasaman Bahri, Daniel A Abolafia, Jeffrey Pennington, and Jascha Sohl-Dickstein. Sensitivity and generalization in neural networks: an empirical study. In *ICLR*, 2018. 2

[28] Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. In *ICML*, 2018. 2, 6, 7

[29] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *AAAI*, 2019. 2, 7, 8

[30] Robert E Schapire. The strength of weak learnability. *Machine learning*, 1990. 2

[31] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015. 1, 3

[32] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In *NIPS*, 2012. 2

[33] Vladimir Vapnik. *The nature of statistical learning theory*. Springer science & business media, 2013. 2

[34] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *CVPR*, 2017. 1, 2, 3, 5

[35] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *ECCV*, 2014. 1

[36] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. In *ICLR*, 2017. 2

[37] Chiyuan Zhang, Samy Bengio, and Yoram Singer. Are all layers created equal? *arXiv:1902.01996*, 2019. 2

[38] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. In *ICLR*, 2017. 2, 3, 7

[39] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *CVPR*, 2018. 1, 2, 3, 5, 7, 8