Regularized Evolution for Image Classifier Architecture Search

Esteban Real [*1] **Alok Aggarwal** [*1] **Yanping Huang** [1] **Quoc V. Le** [1]

Abstract

The effort devoted to hand-crafting image classifiers has motivated the use of architecture search to discover them automatically. Reinforcement learning and evolution have both shown promise for this purpose. This study employs a regularized version of a popular asynchronous evolutionary algorithm. We rigorously compare it to the non-regularized form and to a highly successful reinforcement learning baseline. Using the same hardware, compute effort and training code, we conduct repeated experiments side-by-side, exploring different datasets, search spaces and scales. We show regularized evolution consistently produces models with similar or higher accuracy, across a variety of contexts without need for re-tuning parameters. In addition, evolution exhibits considerably better performance than reinforcement learning at early search stages, suggesting it may be the better choice when fewer compute resources are available. This constitutes the first controlled comparison of the two search algorithms in this context. Finally, we present new architectures discovered with evolution that we nickname *AmoebaNets*. These models achieve state-of-the-art results for CIFAR-10 (mean test error = 2.13%), mobile-size ImageNet (top-1 accuracy = 75.1% with 5.1 M parameters) and ImageNet (top-1 accuracy = 83.1%). This is the first time evolutionary algorithms produce state-of-the-art image classifiers.

1. Introduction

Recent neural network successes have encouraged a proliferation of model architectures (He et al. (2016); Huang et al. (2016); Szegedy et al. (2017); Xie et al. (2017); Chen et al. (2017); Hu et al. (2017), among many others). In turn, this has fueled a decades-old effort to discover them automatically, a field now known as *architecture search*. The traditional approach to architecture search is *neuroevolution of topologies* (Stanley & Miikkulainen, 2002; Floreano et al., 2008; Stanley et al., 2009). Improved hardware now allows evolving at scale, producing image classification models competitive with hand-designs (Real et al., 2017; Miikkulainen et al., 2017; Liu et al., 2017b). In parallel, a newer, alternative approach based on reinforcement learning (RL) was used in Zoph & Le (2016); Baker et al. (2016); Zoph et al. (2017); Zhong et al. (2017); Cai et al. (2017) and reached state-of-the-art results in Zoph et al. (2017). Both approaches seem suitable, but prior work does not guide researchers as to which approach to use in a given context (*i.e.* search space and dataset). Lacking a complete theoretical solution, an empirical first step in this direction would be to know how they compare in one context and how robust they are to context perturbations. Yet, comparison is difficult because every study uses a novel search space, preventing direct attribution of the results to the algorithm. For example, the search space may be small instead of the algorithm being fast. The picture is blurred further by the use of different training techniques that affect model accuracy (Ciregan et al., 2012; Wan et al., 2013; Srivastava et al., 2014), different definitions of *FLOPs* that affect model size[2] and different hardware platforms that affect algorithm runtime[3]. Accounting for all these factors, this study presents the first controlled comparison between evolution and RL in the context of image classifier architecture search. To achieve statistical significance, we undertake the task of running experiments repeatedly and without sampling bias.

Through these experiments, we present a regularized evolutionary algorithm. It is a very natural variant of the standard *tournament selection* strategy (Goldberg & Deb, 1991). Like in tournament selection, in every evolutionary cycle, we select the best of a random sample of individuals to "reproduce". The difference is that we also remove the oldest individual. This is similar to what happens in nature, where old individuals die. Here we show that this regularized version generally performs better than a recently used form (Real et al., 2017) and is more robust in a variety of image classification contexts. We therefore used it in our compara-

[*]Equal contribution [1]Google Brain, Mountain View, California, USA. Correspondence to: Esteban Real <ereal@google.com>.

[2]For example, see https://stackoverflow.com/questions/329174/what-is-flop-s-and-is-it-a-good-measure-of-performance.

[3]A Tesla P100 can be twice as fast as a K40, for example.

tive work as the representative algorithm for evolution. For RL, we used the algorithm in Zoph et al. (2017). We will refer to their paper as *the baseline study*. We chose this baseline because, when we began, it had obtained the most accurate results on CIFAR-10, a popular dataset for image classifier architecture search (Zoph & Le, 2016; Baker et al., 2016; Real et al., 2017; Miikkulainen et al., 2017). We also adopted the baseline study's search space to avoid disrupting any original tuning of their RL algorithm.

We show that evolution can match and surpass RL. We also show that the same holds true when we switch datasets or perturb the search space. Experiments in these alternative contexts were carried out at a smaller compute scale, attaining a compromise between variety and resource use. At the larger compute scale of the baseline study, evolved models under identical conditions achieve better accuracy and smaller size. Finally, we ran evolution for a longer duration with more workers, obtaining state-of-the-art results for CIFAR-10 and ImageNet.

In summary, this paper centers on large-scale search for image classifier architectures, where its contributions are:

1. a variant of the tournament selection evolutionary algorithm, which we show to work better in this domain;
2. the first controlled comparison of RL and evolution, in which we show that evolution matches or surpasses RL;
3. novel evolved architectures, *AmoebaNets*, which achieve state-of-the-art results.

2. Related Work

The most pertinent work was mentioned in Section 1, but we want to highlight studies that stand out due to their efficient search methods, such as Zhong et al. (2017) and Suganuma et al. (2017). This efficiency, however, may not be entirely due to their algorithm (see Section 1). Also, while Zhong et al. (2017) got very close to the state of the art, actually reaching it might require much more compute power (as it did in Zoph et al. (2017) or Liu et al. (2017a), for example). Diminishing accuracy returns at the high-resource regime would not be surprising.

Architecture search speed can be improved with a variety of techniques: progressive-complexity search stages (Liu et al., 2017a), hypernets (Brock et al., 2017), accuracy prediction (Baker et al., 2017), warm-starting and ensembling (Feurer et al., 2015), parallelization, reward shaping and early stopping (Zhong et al., 2017) or Net2Net transformations (Cai et al., 2017). Most of these methods could in principle be applied to evolution too. Miikkulainen et al. (2017) took the orthogonal strategy of splitting up the search into two different model scales in two co-evolving populations, which could be approached from the RL angle as well.

The regularization technique employed removes the oldest model from a population undergoing tournament selection. This has precedent in generational evolutionary algorithms, which discard all models at regular intervals (Miikkulainen et al., 2017; Xie & Yuille, 2017; Suganuma et al., 2017). We avoided such generational algorithms due to their synchronous nature. Tournament selection is asynchronous. This makes it more resource efficient and so it was recently used in its non-regularized form for large-scale evolution (Real et al., 2017; Liu et al., 2017b). Yet, when it was introduced decades ago, it had a regularizing element—albeit a more complex one: sometimes a *random* individual was selected (Goldberg & Deb, 1991); no individuals were removed. Removal may be desirable for garbage-collection purposes. The version in Real et al. (2017) removes the *worst* individual, which is not regularizing. Our version is regularized, natural, and permits garbage collection.

Other than through RL or evolution, architecture search was also explored with cascade-correlation (Fahlman & Lebiere, 1990), boosting (Cortes et al., 2016; Huang et al., 2017), hill-climbing (Elsken et al., 2017), MCTS (Negrinho & Gordon, 2017), SMBO (Mendoza et al., 2016; Liu et al., 2017a), random search (Bergstra & Bengio, 2012) and grid search (Zagoruyko & Komodakis, 2016). Some even forewent the idea of individual architectures (Saxena & Verbeek, 2016; Fernando et al., 2017) and some used evolution to train a single architecture (Jaderberg et al., 2017) or to find its weights (Such et al., 2017). There is much architecture search work beyond image classification too, but we could not do it justice here.

3. Methods

We searched through spaces of neural network classifiers (Section 3.1) using different algorithms (Section 3.2). Following the baseline study, the best models found were then augmented to larger sizes (Section 3.4) to produce high quality image classifiers. We executed the search process at different compute scales (Section 3.3). In addition, we studied the evolutionary algorithms in non-neural network simulations (Section 3.5).

3.1. Search Space

All evolution and RL experiments used the search space design of *the baseline study*, Zoph et al. (2017). It consists in finding the architectures of two Inception-like modules, called the *normal cell* and the *reduction cell*, which preserve and reduce input size, respectively. These cells are stacked in feed-forward patterns to form image classifiers. These resulting models have two hyper-parameters that control their size and impact their accuracy: convolution channel depth (F) and cell stacking depth (N). We used these parameters only to trade accuracy against size. We refer the reader to the baseline study for most details, since only knowledge of

their existence is relevant here.

During the search phase, only the structure of the cells can be altered. These cells each look like a graph with C vertices or *combinations*. A single combination takes two inputs, applies an operation (op) to each and then adds them to generate an output. All unused outputs are concatenated to form the final output of the cell. Within this design, we define three concrete search spaces that differ in the value of C and in the number of ops allowed. In order of increasing size, we will refer to them as SP-I (*e.g.* Figure 2f), SP-II, and SP-III (*e.g.* Figure 2g). SP-I is the exact variant used in the baseline study, SP-II has more allowed ops (more types of convolutions, for example) and SP-III allows for larger tree structures within the cells (details in Section S1.1).

3.2. Architecture Search Algorithms

For evolution, we used either tournament selection or a regularized variant of it. The standard tournament selection method (Goldberg & Deb, 1991) was implemented as in Real et al. (2017): a population of P trained models is improved in *cycles*. At each cycle, a sample of S models is selected at random. The best model of the sample is *mutated* to produce a *child* with an altered architecture, which is trained and added to the population. The worst model in the sample is removed from the population. We will refer to this approach as non-regularized evolution (*NRE*). The variant, regularized evolution (*RE*), is a natural modification: instead of removing the worst model in the sample, we remove the oldest model in the population (*i.e.* the first to have been trained). In both *NRE* and *RE*, populations are initialized with random architectures. The mutations modify these by either randomly substituting an op or randomly reconnecting a combination's input. All random distributions were uniform. For RL experiments, we employed the algorithm in the baseline study without changes.

3.3. Experiment Setups

We ran evolution and RL experiments for comparison purposes at different compute scales, always ensuring both approaches competed under identical conditions. In particular, evolution and RL used *the same* code for network construction, training and evaluation. The scales are as follows (more details in Section S1.3).

Small-scale experiments. These were experiments that could run on CPU. They employed the SP-I, SP-II or SP-III search spaces (Section 3.1). We used the G-CIFAR, MNIST or G-ImageNet classification datasets, where G-CIFAR and G-ImageNet are grayscaled versions of CIFAR-10 and ImageNet 1k, respectively (Section S1.2). These grayscale datasets allowed running experiments with 450 CPU workers lasting 2–5 days. Where unstated, SP-I and G-CIFAR were used.

Large-scale experiments. These used the baseline study's setup. In particular, experiments always used the SP-I search space (Section 3.1) and the CIFAR-10 dataset (Section S1.2). Each ran on 450 GPUs for approximately 7 days to achieve the same number of trained models as the baseline.

3.4. Model Augmentation

By *augmentation* we refer to the process of taking an architecture discovered by evolution or RL and turning it into a full-size, accurate model. This involves enlarging it by increasing N and F, as well as training it for a long time—much longer than during the architecture search phase. Augmented models were trained on the CIFAR-10 or the ImageNet classification datasets (Section S1.2). For consistency, we followed the same procedure as in the baseline study as far as we knew (Section S1.6).

3.5. Simulations Setup

In addition to *experiments* searching for neural network architectures, we carried out *simulations* that evolve solutions to a very simple, single-optimum, D-dimensional, noisy optimization problem with a signal-to-noise ratio matching that of our architecture evolution experiments. Section S1.7 describes this setup more thoroughly, but the details are not necessary to follow the results.

4. Results

We will first show the benefits of regularization in simulations, small-compute-scale and large-compute-scale experiments (Section 4.1). Then we will compare regularized evolution and RL at the small scale in various contexts (Section 4.2) and at the large scale in the context of the baseline study (Section 4.3). The different compute scales were defined in Section 3.3 and the simulations in Section 3.5. Finally, we will evolve models at an even larger scale (Section 4.4).

4.1. Regularized *vs*. Non-Regularized Evolution

This section compares the *regularized evolution* variant (*RE*) with the more standard non-regularized method (*NRE*) described in Section 3.2.

We started by applying evolutionary search in noisy simulations (Section 3.5). Optimized *NRE* and *RE* perform similarly in the low-dimensional problems, which are easier. As the dimensionality increases, *RE* becomes relatively better than *NRE* (Figure 1c). The results suggest that regularization may help navigate noise (more in Section 5).

Encouraged by that finding, we performed small-scale experiments (Section 3.3) using various settings for the algorithm's meta-parameters. Figure 1a shows that *RE* was

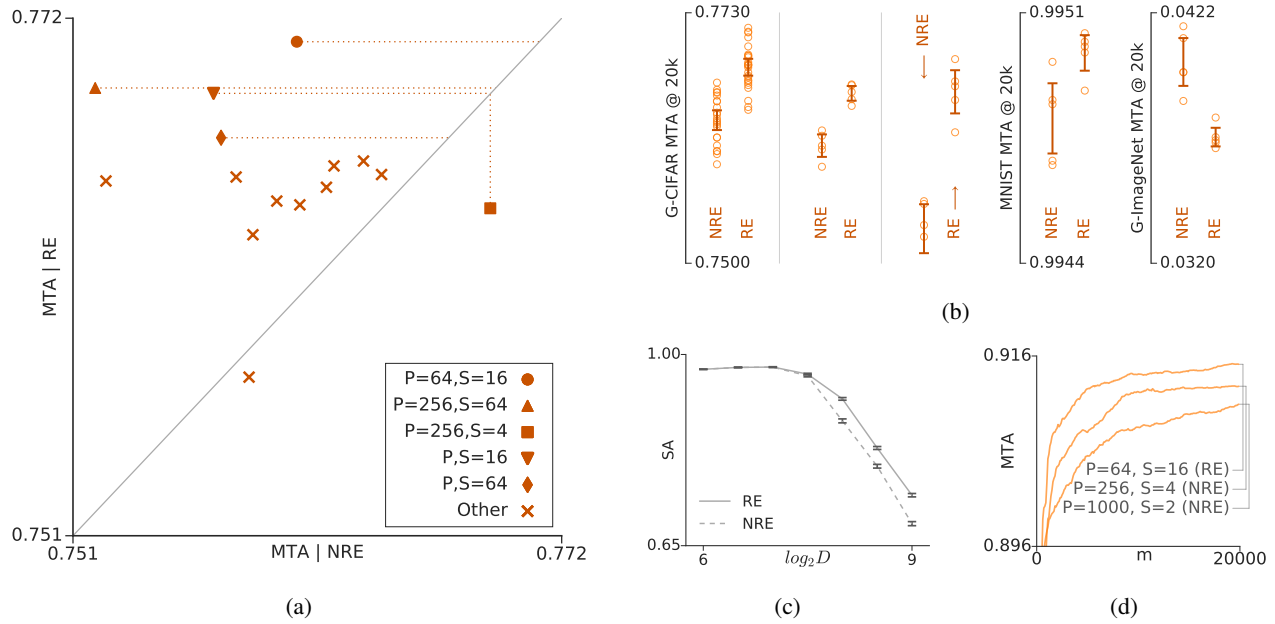

Figure 1. Regularized *vs*. non-regularized evolution (Section 4.1). **(a)** A comparison of non-regularized (*NRE*) and regularized evolution (*RE*) under different meta-parameters through small-scale experiments on the G-CIFAR dataset. Each marker represents a choice of the meta-parameters, namely a population size (P) and a sample size (S). The best are indicated in the legend and the remaining ones are labelled "other". For each P and S combination, we plot the the quality of the models obtained in one *RE* and in one *NRE* experiment along the horizontal and vertical axes, respectively. The quality is measured by the mean testing accuracy (*MTA*) of the top 100 models found (selected by validation accuracy). The fact that most points are above the $y = x$ line (solid, gray) suggests that regularization improves model quality for generic meta-parameters. Moreover, *RE* reaches the highest accuracy (circle marker, vertical axis)—the meta-parameters for this experiment are selected for *RE* throughout the rest of this figure. Analogously, the meta-parameters that produced the highest *NRE* accuracy (square marker, horizontal axis) are selected for *NRE*. **(b)** A comparison of *NRE* and *RE* under 5 different contexts, spanning different datasets and search spaces: G-CIFAR/SP-I, G-CIFAR/SP-II, G-CIFAR/SP-III, MNIST/SP-I and G-ImageNet/SP-I, shown from left to right. For each context, we show the final *MTA* of a few *NRE* and a few *RE* experiments (circles) in adjacent columns. We superpose $\pm 2\,SEM$ error bars, where *SEM* denotes the standard error of the mean. The first context contains many repeats with identical meta-parameters and their *MTA* values seem normally distributed (Shapiro–Wilks test). Under this normality assumption, the error bars represent 95% confidence intervals. All experiments use the meta-parameters optimized in (a). **(c)** Simulation results. The graph summarizes thousands of evolutionary search simulations (Section 3.5). The vertical axis measures the simulated accuracy (*SA*) and the horizontal axis the dimensionality (D) of the problem, a measure of its difficulty. For each D, we optimized the meta-parameters for *NRE* and *RE* independently. To do this, we carried out 100 simulations for each meta-parameter combination and averaged the outcomes. We plot here the optima found, together with $\pm 2\,SEM$ error bars. The graph shows that in this elementary simulated scenario, *RE* is never worse and is significantly better for larger D (note the broad range of the vertical axis). **(d)** Three large-scale experiments on the CIFAR-10 dataset. From top to bottom: an *RE* experiment with the best *RE* meta-parameters from (a), an analogous *NRE* experiment and an *NRE* experiment with the meta-parameters used in a previous study (Real et al., 2017). These accuracy values are not meaningful in absolute terms, as the models need to be augmented to larger size to reach their maximum accuracy (Section 3.4).

better by far. It generally achieved higher accuracy for an arbitrary choice of meta-parameters. Such robustness is desirable for the computationally demanding experiments below, where we cannot afford many runs to optimize the meta-parameters. In addition to being more robust, *RE* also achieved the best accuracy overall. These experiments used the SP-I search space on G-CIFAR (Section 3.3). As a second test for robustness, we swapped the dataset or the search space to produce 5 different contexts. In each, we ran several repeats of evolutionary search using *NRE* and *RE* (Figure 1b). Under 4 of the 5 contexts, *RE* resulted in statistically-significant higher accuracy at the end of the runs, on average. The exception was the G-ImageNet search space, where the experiments were extremely short due to the compute demands of training on so much data using only CPUs. Interestingly, in the two contexts where the search space was bigger (SP-II and SP-III), *all RE* runs did better than *all NRE* runs.

To verify that these findings hold at scale, we ran three experiments using the baseline study's conditions. Figure 1d shows that *RE* performed better here too. Taken together, simulations and architecture evolution experiments provide

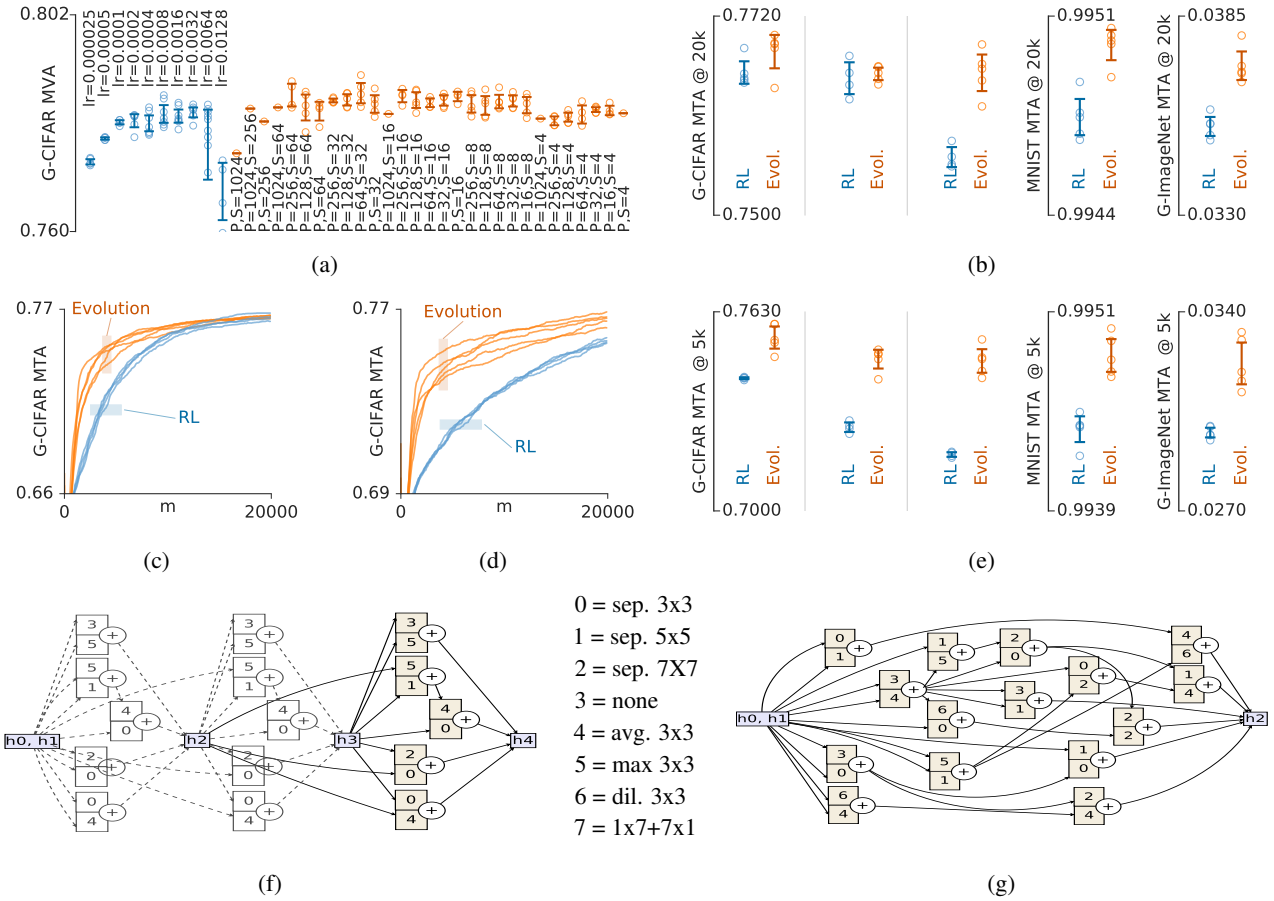

Figure 2. Evolution *vs*. RL at small-compute scale in different contexts (Section 4.2). Plots show repeated evolution (orange) and RL (blue) experiments side-by-side. **(a)** Summary of hyper-parameter optimization experiments on G-CIFAR. We swept the learning rate (*lr*) for RL (left) and the population size (*P*) and sample size (*S*) for evolution (right). We ran 5 experiments (circles) for each scenario. The vertical axis measures the mean validation accuracy (*MVA*) of the top 100 models in an experiment. Superposed on the raw data are ± 2 *SEM* error bars. From these results, we selected best meta-parameters to use in the remainder of this figure. **(b)** We assessed robustness by running the same experiments in 5 different contexts, spanning different datasets and search spaces: G-CIFAR/SP-I, G-CIFAR/SP-II, G-CIFAR/SP-III, MNIST/SP-I and G-ImageNet/SP-I, shown from left to right. These experiments ran to 20k models. The vertical axis measures the mean testing accuracy (*MTA*) of the top 100 models (selected by validation accuracy). **(c)** and **(d)** show a detailed view of the progress of the experiments in the G-CIFAR/SP-II and G-CIFAR/SP-III contexts, respectively. The horizontal axes indicate the number of models (*m*) produced as the experiment progresses. **(e)** Resource-constrained settings may require stopping experiments early. At 5k models, evolution performs better than RL in all 5 contexts. **(f)** and **(g)** show a stack of normal cells of the best model found for G-CIFAR in the SP-I and SP-III search spaces, respectively (see Section 3.1). The "h" labels hidden states. The ops ("avg 3x3", *etc*.) are listed in full form in Section S1.1. Data flows from left to right. See the baseline study for a detailed description of these diagrams. In (f), N=3, so the cell is replicated three times; *i.e*. the left two-thirds of the diagram (grayed out) are constrained to mirror the right third. This is in contrast with the vastly larger SP-III search space of (g), where a bigger, unconstrained construct without replication is explored.

evidence that *RE* is desirable. We will, therefore, use this method for comparison against RL in the rest of this study.

4.2. Evolution *vs*. RL at Small-Compute Scale

We first optimized the meta-parameters for RL and for evolution by running small-scale experiments (Section 3.3) with each algorithm, repeatedly, under each condition. Figure 2a shows that neither approach was very sensitive. Still, this was a necessary step to ensure both RL and evolution are treated fairly. We then compared both algorithms in 5 different contexts by swapping the dataset or the search space (Figure 2b). Evolution is either better than or equal to RL, with statistical significance. The best contexts for evolution and for RL are shown in more detail in Figures 2c and 2d, respectively. They show the progress of 5 repeats of each algorithm. The initial speed of evolution is striking, especially in the largest search space (SP-III). Figures 2f

and 2g illustrate the top architectures from SP-I and SP-III, respectively. Regardless of context, Figure 2e indicates that accuracy under evolution increases significantly faster than RL at the initial stage. This stage was not accelerated by higher RL learning rates.

4.3. Evolution *vs*. RL at Large-Compute Scale

We performed large-scale architecture search experiments (Section 3.3) to compare evolution and RL side-by-side. As above, we started by optimizing each approach. The 2-hyper-parameter space of evolution is too large to explore in detail, so we only tried a handful of trial-and-error runs, informed by the smaller-scale results above. We then chose the best set of conditions found (Section S1.5). For RL, we were more thorough: we took all parameters from the baseline study and then fine-tuned the learning rate. This was done by sweeping until we saw the accuracy decline at both extremes (Section S1.5). The RL optimum was consistent with that found at the small scale. With the hyper-parameters thus obtained, we ran evolution, RL and random search (RS) experiments. We repeated each experiment exactly 5 times and we present all the results in Figures 3a–c. They show that under the baseline study's conditions:

- Evolution and RL do equally well on accuracy;
- Both are significantly better than RS; and
- Evolution is faster, as we saw above.

We then took the top models from each experiment and augmented them (Section 3.4). Figure 3d shows the testing accuracy and FLOPs of the resulting full-size models. Augmentation adds noise, but the relative accuracy between evolution, RL and RS is roughly preserved. Random search is still the worst with high confidence. Evolved models exhibit a slight increase in accuracy variance and much lower FLOPs than those obtained with RL (see also Section 5).

4.4. Beyond Controlled Comparisons

We selected the best model from all the evolution runs from Section 4.3 and nickname it *AmoebaNet-A*. To avoid overfitting, this model was selected by validation accuracy within and across experiments—and was the only model selected. By adjusting N and F, we can trade more parameters for lower testing error (Table 1). Under the same experimental conditions, the baseline study obtained NASNet-A. The table indicates that on CIFAR-10 AmoebaNet-A exhibits lower error while matching parameters and fewer parameters while matching error. It also reaches the current state of the art on ImageNet (Table 2).

Having completed the controlled comparison, we concentrated resources on dedicated evolution experiments, exploring the larger SP-II search space with TPUv2 chips (Section S1.3). After the augmentation procedure (Section 3.4), we selected the top model by validation accuracy (again, within and across experiments). We nickname it *AmoebaNet-B* (model diagram in Section S2). This model sets a new state of the art on the CIFAR-10 dataset (Table 1). Note that training time was not a constraint: even the largest configuration with 34.9 M parameters took less than 24 hours to train to completion on TPU. We forewent training larger models having observed diminishing returns.

We found that another model from Section 4.3, *AmoebaNet-C*, had better accuracy with very few parameters. It achieves the state of the art for mobile-sized and full-sized ImageNet

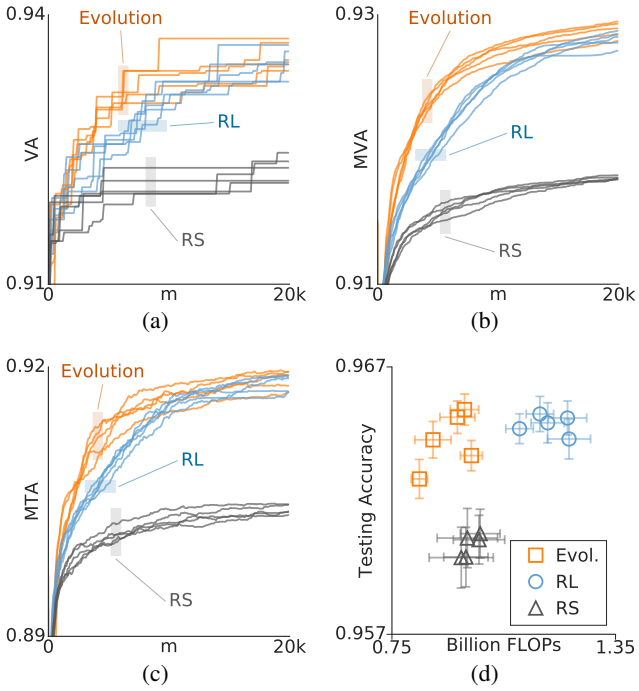

Figure 3. Evolution *vs*. RL at large-compute scale under the conditions of the baseline study (Section 4.3). We show results from (regularized) evolution (orange), RL (blue) and random search (RS, black) experiments. Except in (d), all horizontal axes measure experiment progress in terms of number of models generated (m), which can be approximately thought of as "experiment run time". All vertical axes show various measures of model quality. Each curve shows the improvement in the accuracy of the models generated throughout one experiment. **(a)**, **(b)** and **(c)** show the progress of 5 identical experiments for each of the three algorithms. The evolution and RL experiments used the best meta-parameters found (see Section 4.3). The vertical axis measures the top validation accuracy (*VA*) by model m in (a), the *MVA* in (b) and the mean testing accuracy (*MTA*) of the top 100 models seen by model m selected by validation accuracy in (c). The testing accuracy had been hidden from the algorithm and the researchers until the plotting of (c). **(d)** Model accuracy and complexity for top equally augmented models, showing the true potential of the architecture search experiments just presented. Each marker corresponds to the average within an experiment. The error bars are ± 2 SEM.

Table 1. CIFAR-10 results. We compare hand-designs† (top section), other architecture search results† (middle section) and our best evolved model (bottom section). "+c/o" indicates use of cutout (DeVries & Taylor, 2017). "Params" is the number of free parameters. We report our model's test error as $\mu \pm 2 \times$ SEM. NASNets, PNASNets and AmoebaNets are reported as "XXNet (N, F)". Evolution-based methods are marked with a *.

Model	Params	Test Error (%)
DenseNet-BC (k = 24)	15.3 M	3.62
ResNeXt-29, 16x64d	68.1 M	3.58
DenseNet-BC (L=100, k=40)	25.6 M	3.46
Shake-Shake 26 2x96d + c/o	26.2 M	2.56
PyramidNet + Shakedrop	26.0 M	2.31
Evolving DNN*	–	7.30
MetaQNN (top model)	–	6.92
CGP-CNN (ResSet)*	1.68 M	5.98
Large Scale Evolution*	5.4 M	5.40
EAS	23.4 M	4.23
SMASHv2	16 M	4.03
Hierarchical (2, 64)*	15.7 M	3.75 ± 0.12
Block-QNN-A, N=4	–	3.60
PNASNet-5 (3, 48)	3.2 M	3.41 ± 0.09
NASNet-A (6, 32)	3.3 M	3.41
NASNet-A (6, 32) + c/o	3.3 M	2.65
NASNet-A (7, 96) + c/o	27.6 M	2.40
AmoebaNet-A (6, 32)*	2.6 M	3.40 ± 0.08
AmoebaNet-B (6, 36)*	2.8 M	3.37 ± 0.04
AmoebaNet-A (6, 36)*	3.2 M	3.34 ± 0.06
AmoebaNet-B (6, 80)*	13.7 M	3.04 ± 0.09
AmoebaNet-B (6, 112)*	26.7 M	3.04 ± 0.04
AmoebaNet-B (6, 128)*	34.9 M	2.98 ± 0.05
AmoebaNet-B (6, 36) + c/o*	2.8 M	2.55 ± 0.05
AmoebaNet-B (6, 80) + c/o*	13.7 M	2.31 ± 0.05
AmoebaNet-B (6, 112) + c/o*	26.7 M	2.21 ± 0.04
AmoebaNet-B (6, 128) + c/o*	34.9 M	$\mathbf{2.13 \pm 0.04}$

(Table 3) and (Table 2). We do not include this model in the CIFAR-10 table because it was selected by CIFAR-10 test accuracy explicitly with the purpose of retraining and testing on ImageNet.

Table 2. ImageNet classification results. We compare hand-designs† (top section), other architecture search results† (middle section) and our model (bottom section). "Params" is the number of free parameters. "×+" means number of multiply-adds. "1/5-Acc" refers to the top-1 and top-5 test accuracy. NASNets, PNASNets and AmoebaNets are reported as "XXNet (N, F)". Evolution-based methods are marked with a *.

Model	Params	×+	1/5-Acc (%)
Inception V3	23.8M	5.72B	78.8 / 94.4
Xception	22.8M	8.37B	79.0 / 94.5
Inception ResNet V2	55.8M	13.2B	80.4 / 95.3
ResNeXt-101 (64x4d)	83.6M	31.5B	80.9 / 95.6
PolyNet	92.0M	34.7B	81.3 / 95.8
Dual-Path-Net-131	79.5M	32.0B	81.5 / 95.8
Squeeze-Excite-Net	145.8M	42.3B	82.7 / 96.2
GeNet-2*	156M	–	72.1 / 90.4
Block-QNN-B (N=3)*	–	–	75.7 / 92.6
Hierarchical (2, 64)*	64M	–	79.7 / 94.8
PNASNet-5 (4, 216)	86.1M	25.0B	82.9 / 96.1
NASNet-A (6, 168)	88.9M	23.8B	82.7 / 96.2
AmoebaNet-B (6, 190)*	84.0M	22.3B	82.3 / 96.1
AmoebaNet-A (6, 190)*	86.7M	23.1B	82.8 / 96.1
AmoebaNet-A (6, 204)*	99.6M	26.2B	82.8 / 96.2
AmoebaNet-C (6, 228)*	155.3M	41.1B	**83.1 / 96.3**

Table 3. ImageNet classification results in the *mobile* setting. We compare hand-designs† (top section), other architecture search results† (middle section) and our model (bottom section). Notation is as in Table 2.

Model	Params	×+	1/5-Acc (%)
MobileNetV1	4.2M	575M	70.6 / 89.5
ShuffleNet (2x)	4.4M	524M	70.9 / 89.8
MobileNetV2 (1.4)	6.9M	585M	74.7 / –
NASNet-A (4, 44)	5.3M	564M	74.0 / 91.3
PNASNet-5 (3, 54)	5.1M	588M	74.2 / 91.9
AmoebaNet-B (3, 62)*	5.3M	555M	74.0 / 91.5
AmoebaNet-A (4, 50)*	5.1M	555M	74.5 / 92.0
AmoebaNet-C (4, 44)*	5.1M	535M	75.1 / 92.1
AmoebaNet-C (4, 50)*	6.4M	570M	75.7 / 92.4

5. Discussion

We employed different metrics to assess experiment progress and outcome. The validation accuracy (*VA*) is the actual reward seen by the algorithms (*e.g.* Figure 3a). It is, therefore, a natural choice to assess algorithm performance. However, it suffers from significant uncertainty due to neural network training noise. Averaging over the top models yields a more robust quantity, the *mean validation accuracy* (*MVA*, *e.g.* Figure 3b). The *MVA*, still has a drawback: it may be deceiving in search-space regions where the models are prone to large generalization error. These regions "look good" to the algorithm but are less conducive to finding high-quality classifiers. This is more a search *space* issue than a search *algorithm* issue. Still, in practice, we want to know how well the models generalize. For this we can instead analyze the *mean testing accuracy* (*MTA*),

†Table references: Huang et al. (2016); Xie et al. (2017); Gastaldi (2017); Han et al. (2016); Miikkulainen et al. (2017); Baker et al. (2016); Suganuma et al. (2017); Real et al. (2017); Cai et al. (2017); Brock et al. (2017); Liu et al. (2017b); Zhong et al. (2017); Liu et al. (2017a); Yamada et al. (2018); Szegedy et al. (2016); Chollet (2016); Szegedy et al. (2017); Zhang et al. (2017b); Chen et al. (2017); Hu et al. (2017); Xie & Yuille (2017); Zhong et al. (2017); Zoph et al. (2017); Howard et al. (2017); Zhang et al. (2017a); Sandler et al. (2018).

a metric not probed during the search process (*e.g*. Figure 3c). Note that the *MTA* still averages models selected by *validation* accuracy. These various metrics all support the conclusions of this study

The large-scale experiment progress plots (Figure 3) suggest that both RL and evolution are approaching a common accuracy asymptote. This raises the question of which algorithm gets there faster. The plots indicate that RL reaches half-maximum accuracy in roughly twice the time. We abstain, nevertheless, from further quantifying this effect since it depends strongly on how speed is measured (the number of models necessary to reach accuracy a depends on a; the natural choice of $a = a_{max}/2$ may be too low to be informative; *etc*.). Note from the variance in the figures that the relative speed factor would also become very noisy if measured from the non-averaged *VA* curves, even noisier if experiment repeats had not been performed. Algorithm speed is more important when exploring larger spaces, where reaching the optimum requires more compute than is available (*e.g*. Figures 2c and 2g). In this regime, evolution may shine.

The size of the search space deserves more consideration. Large spaces can have the advantage of requiring less expert input (Real et al., 2017), while small spaces can reach better results sooner (Liu et al., 2017b;a) because they can be constructed to exclude bad models. Consequently, in such smaller spaces, it is likely harder to distinguish between search algorithms (Liu et al. (2017b); also see overlapping error bars in Liu et al. (2017a)). The well-crafted space of Zoph et al. (2017) provided an appropriate compromise for our work. Different regimes could be important elsewhere, depending on goals, resources and expertise.

Once the architecture search phase is complete, the top resulting models were all *augmented* equally (Section 3.4). The accuracy of these augmented models may not mirror perfectly that of their search-phase counterparts. This introduces randomness, making the accuracy comparison in Figure 3d less indicative of algorithm performance than that in Figure 3c. FLOPs, however, are an intrinsic property of the architecture and Figure 3d demonstrates that evolved models are leaner. We speculate that regularized asynchronous evolution may be reducing the FLOPs because it is indirectly optimizing for speed—fast models may do well because they "reproduce" quickly even if they lack the very high accuracy of their slower peers. Verifying this speculation is beyond the scope of this paper.

Regularization was advantageous in both simulations and neural-network architecture evolution experiments (Section 4.1, Figure 1). The simulations were constructed to be as simple as possible while still modeling the noisy evaluation present in neural networks. We can therefore speculate that regularization may help navigate this noise, as follows. Under regularized evolution, all models have a short lifespan. Yet, populations improve over longer timescales (Figures 1d, 2c,d, 3a–c). This requires that its surviving lineages remain good through the generations. This, in turn, demands that the inherited architectures retrain well (since we always train from scratch, the weights are not heritable). On the other hand, *non-regularized* tournament selection allows models to live infinitely long, so a population can improve simply by accumulating high-accuracy models. Unfortunately, these models may have reached their high accuracy by luck during the noisy training process. In summary, only the regularized form requires that the architectures remain good after they are retrained. Whether this mechanism is responsible for the observed superiority of regularization is conjecture. We leave its verification to future work.

6. Conclusion

We have shown for the first time that neural-network architecture evolution can produce state-of-the-art image classifiers. We (i) employed a regularized evolutionary algorithm and demonstrated through controlled comparisons that regularization is directly responsible for a significant performance improvement over a recently used tournament selection variant. Utilizing the search space from an existing RL baseline study, (ii) we performed the first rigorous comparison of evolution and RL for image classifier search. We found that regularized evolution had faster convergence speed and obtained equal or better accuracy across a variety of contexts without need for re-tuning parameters. Finally, (iii) we concentrated compute resources to explore a larger search space using TPUs. Regularized evolution experiments yielded *AmoebaNets*, novel architectures that achieve state-of-the-art results for CIFAR-10, mobile-sized ImageNet and ImageNet.

This study is only a first empirical step in illuminating the relationship between evolution and RL in this particular context. We hope that future work will generalize this comparison to expose the merits of each of the two approaches. We also hope that future work will further reduce the compute cost of architecture search. On the other hand, in an economy of scale, repeated use of efficient models discovered may vastly exceed the compute cost of the discovery, thus justifying it regardless. Undoubtedly, the maximization of the final accuracy and the optimization of the search process are both worth exploring.

Acknowledgements

We wish to thank Megan Kacholia, Vincent Vanhoucke, Xiaoqiang Zheng and especially Jeff Dean for their support and valuable input; Barret Zoph and Vijay Vasudevan for help with the code and experiments used in Zoph et al. (2017),

as well as Jianwei Xie, Jacques Pienaar, Derek Murray, Gabriel Bender, Golnaz Ghiasi, Saurabh Saxena and Jie Tan for other coding contributions; Jacques Pienaar, Luke Metz, Chris Ying and Andrew Selle for manuscript comments, all the above and Patrick Nguyen, Samy Bengio, Geoffrey Hinton, Risto Miikkulainen, Yifeng Lu, David Dohan, David So, David Ha, Vishy Tirumalashetty, Yoram Singer, Chris Ying and Ruoming Pang for helpful discussions; and the larger Google Brain team.

References

Baker, Bowen, Gupta, Otkrist, Naik, Nikhil, and Raskar, Ramesh. Designing neural network architectures using reinforcement learning. *arXiv preprint arXiv:1611.02167*, 2016.

Baker, Bowen, Gupta, Otkrist, Raskar, Ramesh, and Naik, Nikhil. Accelerating neural architecture search using performance prediction. *CoRR, abs/1705.10823*, 2017.

Bergstra, James and Bengio, Yoshua. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012.

Brock, Andrew, Lim, Theodore, Ritchie, James M, and Weston, Nick. Smash: one-shot model architecture search through hypernetworks. *arXiv preprint arXiv:1708.05344*, 2017.

Cai, Han, Chen, Tianyao, Zhang, Weinan, Yu, Yong, and Wang, Jun. Reinforcement learning for architecture search by network transformation. *arXiv preprint arXiv:1707.04873*, 2017.

Chen, Yunpeng, Li, Jianan, Xiao, Huaxin, Jin, Xiaojie, Yan, Shuicheng, and Feng, Jiashi. Dual path networks. In *NIPS*, pp. 4470–4478, 2017.

Chollet, François. Xception: Deep learning with depthwise separable convolutions. *arXiv preprint*, 2016.

Ciregan, Dan, Meier, Ueli, and Schmidhuber, Jürgen. Multi-column deep neural networks for image classification. In *CVPR*, pp. 3642–3649. IEEE, 2012.

Cortes, Corinna, Gonzalvo, Xavi, Kuznetsov, Vitaly, Mohri, Mehryar, and Yang, Scott. Adanet: Adaptive structural learning of artificial neural networks. *arXiv preprint arXiv:1607.01097*, 2016.

Deng, Jia, Dong, Wei, Socher, Richard, Li, Li-Jia, Li, Kai, and Fei-Fei, Li. Imagenet: A large-scale hierarchical image database. In *CVPR*, pp. 248–255. IEEE, 2009.

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

Elsken, Thomas, Metzen, Jan-Hendrik, and Hutter, Frank. Simple and efficient architecture search for convolutional neural networks. *arXiv preprint arXiv:1711.04528*, 2017.

Fahlman, Scott E and Lebiere, Christian. The cascade-correlation learning architecture. In *NIPS*, pp. 524–532, 1990.

Fernando, Chrisantha, Banarse, Dylan, Blundell, Charles, Zwols, Yori, Ha, David, Rusu, Andrei A, Pritzel, Alexander, and Wierstra, Daan. Pathnet: Evolution channels gradient descent in super neural networks. *arXiv preprint arXiv:1701.08734*, 2017.

Feurer, Matthias, Klein, Aaron, Eggensperger, Katharina, Springenberg, Jost, Blum, Manuel, and Hutter, Frank. Efficient and robust automated machine learning. In *NIPS*, pp. 2962–2970, 2015.

Floreano, Dario, Dürr, Peter, and Mattiussi, Claudio. Neuroevolution: from architectures to learning. *Evolutionary Intelligence*, 1(1):47–62, 2008.

Gastaldi, Xavier. Shake-shake regularization. *arXiv preprint arXiv:1705.07485*, 2017.

Goldberg, David E and Deb, Kalyanmoy. A comparative analysis of selection schemes used in genetic algorithms. *Foundations of genetic algorithms*, 1:69–93, 1991.

Han, Dongyoon, Kim, Jiwhan, and Kim, Junmo. Deep pyramidal residual networks. *arXiv preprint arXiv:1610.02915*, 2016.

He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian. Deep residual learning for image recognition. In *CVPR*, pp. 770–778, 2016.

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

Hu, Jie, Shen, Li, and Sun, Gang. Squeeze-and-excitation networks. *arXiv preprint arXiv:1709.01507*, 2017.

Huang, Furong, Ash, Jordan, Langford, John, and Schapire, Robert. Learning deep resnet blocks sequentially using boosting theory. *arXiv preprint arXiv:1706.04964*, 2017.

Huang, Gao, Liu, Zhuang, Weinberger, Kilian Q, and van der Maaten, Laurens. Densely connected convolutional networks. *arXiv preprint arXiv:1608.06993*, 2016.

Jaderberg, Max, Dalibard, Valentin, Osindero, Simon, Czarnecki, Wojciech M, Donahue, Jeff, Razavi, Ali, Vinyals, Oriol, Green, Tim, Dunning, Iain, Simonyan, Karen, et al. Population based training of neural networks. *arXiv preprint arXiv:1711.09846*, 2017.

Krizhevsky, Alex and Hinton, Geoffrey. Learning multiple layers of features from tiny images. *Master's thesis, Dept. of Computer Science, U. of Toronto*, 2009.

Liu, Chenxi, Zoph, Barret, Shlens, Jonathon, Hua, Wei, Li, Li-Jia, Fei-Fei, Li, Yuille, Alan, Huang, Jonathan, and Murphy, Kevin. Progressive neural architecture search. *arXiv preprint arXiv:1712.00559*, 2017a.

Liu, Hanxiao, Simonyan, Karen, Vinyals, Oriol, Fernando, Chrisantha, and Kavukcuoglu, Koray. Hierarchical representations for efficient architecture search. *arXiv preprint arXiv:1711.00436*, 2017b.

Mendoza, Hector, Klein, Aaron, Feurer, Matthias, Springenberg, Jost Tobias, and Hutter, Frank. Towards automatically-tuned neural networks. In *Workshop on Automatic Machine Learning*, pp. 58–65, 2016.

Miikkulainen, Risto, Liang, Jason, Meyerson, Elliot, Rawal, Aditya, Fink, Dan, Francon, Olivier, Raju, Bala, Navruzyan, Arshak, Duffy, Nigel, and Hodjat, Babak. Evolving deep neural networks. *arXiv preprint arXiv:1703.00548*, 2017.

Negrinho, Renato and Gordon, Geoff. Deeparchitect: Automatically designing and training deep architectures. *arXiv preprint arXiv:1704.08792*, 2017.

Real, Esteban, Moore, Sherry, Selle, Andrew, Saxena, Saurabh, Suematsu, Yutaka Leon, Le, Quoc, and Kurakin, Alex. Large-scale evolution of image classifiers. *arXiv preprint arXiv:1703.01041*, 2017.

Sandler, Mark, Howard, Andrew, Zhu, Menglong, Zhmoginov, Andrey, and Chen, Liang-Chieh. Inverted residuals and linear bottlenecks: [...]. *arXiv preprint arXiv:1801.04381*, 2018.

Saxena, Shreyas and Verbeek, Jakob. Convolutional neural fabrics. In *NIPS*, pp. 4053–4061, 2016.

Srivastava, Nitish, Hinton, Geoffrey, Krizhevsky, Alex, Sutskever, Ilya, and Salakhutdinov, Ruslan. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1): 1929–1958, 2014.

Stanley, Kenneth O and Miikkulainen, Risto. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2):99–127, 2002.

Stanley, Kenneth O, D'Ambrosio, David B, and Gauci, Jason. A hypercube-based encoding for evolving large-scale neural networks. *Artificial life*, 15(2):185–212, 2009.

Such, Felipe Petroski, Madhavan, Vashisht, Conti, Edoardo, Lehman, Joel, Stanley, Kenneth O, and Clune, Jeff. Deep neuroevolution: Genetic algorithms [...]. *arXiv preprint arXiv:1712.06567*, 2017.

Suganuma, Masanori, Shirakawa, Shinichi, and Nagao, Tomoharu. A genetic programming approach to designing convolutional neural network architectures. *arXiv preprint arXiv:1704.00764*, 2017.

Szegedy, Christian, Vanhoucke, Vincent, Ioffe, Sergey, Shlens, Jon, and Wojna, Zbigniew. Rethinking the inception architecture for computer vision. In *CVPR*, pp. 2818–2826, 2016.

Szegedy, Christian, Ioffe, Sergey, Vanhoucke, Vincent, and Alemi, Alexander A. Inception-v4, inception-resnet and the impact of residual connections on learning. In *AAAI*, volume 4, pp. 12, 2017.

Wan, Li, Zeiler, Matthew, Zhang, Sixin, Le Cun, Yann, and Fergus, Rob. Regularization of neural networks using dropconnect. In *ICML*, pp. 1058–1066, 2013.

Xie, Lingxi and Yuille, Alan. Genetic cnn. *arXiv preprint arXiv:1703.01513*, 2017.

Xie, Saining, Girshick, Ross, Dollár, Piotr, Tu, Zhuowen, and He, Kaiming. Aggregated residual transformations for deep neural networks. In *CVPR*, pp. 5987–5995. IEEE, 2017.

Yamada, Yoshihiro, Iwamura, Masakazu, and Kise, Koichi. Shakedrop regularization. *https://openreview.net/forum?id=S1NHaMW0b*, 2018.

Zagoruyko, Sergey and Komodakis, Nikos. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

Zhang, Xiangyu, Zhou, Xinyu, Lin, Mengxiao, and Sun, Jian. Shufflenet: An extremely efficient convolutional neural network for mobile devices. *arXiv preprint arXiv:1707.01083*, 2017a.

Zhang, Xingcheng, Li, Zhizhong, Loy, Chen Change, and Lin, Dahua. Polynet: A pursuit of structural diversity in very deep networks. In *CVPR*, pp. 3900–3908. IEEE, 2017b.

Zhong, Zhao, Yan, Junjie, and Liu, Cheng-Lin. Practical network blocks design with q-learning. *arXiv preprint arXiv:1708.05552*, 2017.

Zoph, Barret and Le, Quoc V. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.

Zoph, Barret, Vasudevan, Vijay, Shlens, Jonathon, and Le, Quoc V. Learning transferable architectures for scalable image recognition. *arXiv preprint arXiv:1707.07012*, 2017.

Regularized Evolution for Image Classifier Architecture Search

Supplementary Material

S1. Detailed Methods

S1.1. Search Space Details

Section 3.1 introduced 3 search spaces, SP-I, SP-II and SP-III and described them in outline. In the language presented in Section 3, these can be defined as:

- SP-I: uses $C = 5$ and 8 possible ops (identity; 3x3, 5x5 and 7x7 separable (sep.) convolutions (convs.); 3x3 average (avg.) pool; 3x3 max pool; 3x3 dilated (dil.) sep. conv.; 1x7 then 7x1 conv.)—see Figure 2f,
- SP-II: uses $C = 5$ and 19 possible ops (identity; 1x1 and 3x3 convs.; 3x3, 5x5 and 7x7 sep. convs.; 2x2 and 3x3 avg. pools; 2x2 min pool.; 2x2 and 3x3 max pools; 3x3, 5x5 and 7x7 dil. sep. convs.; 1x3 then 3x1 conv.; 1x7 then 7x1 conv. 3x3 dil. conv. with rates 2, 4 and 6), and
- SP-III: uses $C = 15$ and 8 possible ops (same as SP-I)—see Figure 2g.

S1.2. Datasets

We used the following datasets:

- CIFAR-10 (Krizhevsky & Hinton, 2009): dataset with naturalistic images labeled with 1 of 10 common object classes. It has 50k training examples and 10k test examples, all of which are 32 x 32 color images. 5k of the training examples were held out in a validation set. The remaining 45k examples were used for training.
- G-CIFAR: a grayscaled version of CIFAR-10. The original images were each averaged across channels. Training, validation and testing set splits were preserved.
- MNIST: a handwritten black-and-white digit classification dataset. It has 60k and 10k testing examples. We held out 5k for validation. The labels are the digits 0-9.
- ImageNet (Deng et al., 2009): large set of naturalistic images, each labeled with one or more objects from among 1k classes. Contains a total of about 1.2M 331x331 examples. Of these, 50k were held out for validation and 50k for testing. The rest constituted the training set.
- G-ImageNet: a grayscaled subset of ImageNet (see Section 3.4). The original images were averaged across channels and re-sized to 32x32. We generated a training set with 200k images and a validation set with 10k images, both from the standard training set. We also generated a testing set with the 50k images from the standard validation set.

S1.3. Experiment Setup Details

Section 3.3 introduced 2 different compute scales. The following completes their descriptions.

Small-scale experiments. Each experiment ended when 20k models were trained (*i.e.* 20k sample complexity). Each model trained for 4, 4 or 1 epochs in either the G-CIFAR, MNIST or G-ImageNet datasets, respectively. In all cases, $C = 5$, $N = 3$ and $F = 8$. These settings were chosen to be as close as possible to the large-scale experiments below while running reasonably fast on CPU.

Large-scale experiments. Each experiment also ended when 20k models were trained. The search space was SP-I and the dataset was CIFAR-10 (see Section 3.4). Each model trained for 25 epochs. $C = 5$, $N = 3$ and $F = 24$.

Section 4.4 introduced a new setup for exploring the larger SP-II search space. The following completes its description.

Dedicated evolution experiments. Like the large-scale experiments, except that the search space was expanded to SP-II, the models were larger ($F = 32$) and the training was longer (50 epochs). By training larger models for more epochs, the search phase validation accuracy is more representative of the true validation accuracy when the model is scaled up in

parameters, i.e. a model evaluated with ($N = 3, F = 32$, 50 epochs) is likely to be better than ($N = 3, F = 24$, 25 epochs) at predicting performance of ($N = 6, F = 32$, 600 epochs). Each search step is now more expensive. The experiment ran on 900 TPUv2 chips for 5 days and trained 27k models total.

S1.4. Model Training During Search

All training details as in Zoph et al. (2017).

S1.5. Meta-Parameter Optimization

For simulations, see Figure 1c and Section S1.7.

For small-scale evolution experiments (Section 4.2), we swept both *P* and *S*. The values used are in Figure 2a.

Given the robustness to meta-parameters observed in Figure 2a, we deemed it sufficient to try only a few parameters to optimize large-scale evolution experiments (Section 4.3 and Figure 3). We tried: $P = 100, S = 25$; $P = 64, S = 16$; $P = 20, S = 20$; $P = 100, S = 50$; $P = 100, S = 2$.

For small-scale RL experiments (Section 4.2), we used the parameters from the baseline study and fine-tuned them by sweeping the learning rate (*lr*). The values used are in Figure 2a.

For large-scale RL experiments (Section 4.3), again we used the meta-parameters from the baseline study (under identical conditions) and fine-tuned them by sweeping the *lr*. We tried: $lr = 0.00003$, $lr = 0.00006$, $lr = 0.00012$, $lr = 0.0002$, $lr = 0.0004$, $lr = 0.0008$, $lr = 0.0016$, $lr = 0.0032$. The best *lr* was 0.0008, which matched the optimization done at small scale (Section 2a).

In order to avoid selection bias, the experiment repeats plotted in Figures 3a, 3b and 3c do not include the actual runs from the optimization stage, only the meta-parameters found. This was a decision made a priori.

S1.6. Augmented Model Training

To compare augmented models side-by-side (Figure 3d, Section 4.3), we selected from each evolution or RL experiment the top 20 models by validation accuracy. We augmented all of them equally by setting $N = 6$ and $F = 32$, as was done in the experiment that produced NASNet-A in the baseline study. Finally, we trained them on CIFAR-10 (details below).

To compare our best model with the baseline study's best model, NASNet-A, while matching experiment resources (last paragraph of Section 4.3), we augmented each of the top $K = 100$ models from each evolution run (hence 500 total models) with $N = 6$ and $F = 32$, and selected the best by validation fitness. We then retrained the model 8 times at various sizes to measure the mean testing error. We presented the two configurations that matched either the accuracy or number of parameters of NASNet-A.

For Section 4.4, Table 1, we selected from the experiment $K = 100$ models. To do this, we binned the models by their number of parameters to cover the range, using b bins. From each bin, we took the top K/b models by validation accuracy. We then augmented all models to $N = 6$ and $F = 32$ and picked the one with the top validation accuracy. We then re-augmented this model with the (N, F) values in Table 1. We trained each resulting size 8 times on CIFAR-10 to measure the mean testing accuracy.

For Section 4.4, Tables 2 and 3, the selection was already described in the main text.

To train augmented models on CIFAR-10, we proceeded as in the baseline study, except setting batch size 128, an initial learning rate of 0.024 with cosine decay to zero over 600 epochs, and drop-connect probability of 0.7. When measuring the validation accuracy, the held out validation set was not included in the training set. When measuring the testing accuracy, we used the full training set. We stress that the testing accuracy had never been used until the evaluation of the final models on the given dataset presented in the text/table.

To train augmented models on ImageNet, we followed training, data augmentation, and evaluation procedures in (Szegedy et al., 2016). Input image size was 224x224 for mobile size models and 331x331 for large models. We used distributed synchronous SGD with 100 workers. We employed RMSProp optimizer with a decay of 0.9 and $\epsilon = 0.1$, L2 regularization with weight decay 4×10^{-5}, label smoothing with value 0.1 and an auxiliary head weighted by 0.4. We applied dropout to the final softmax layer with probability 0.5. Learning rate started at 0.001 and decayed every 2 epochs with rate 0.97.

S1.7. Simulation Details

The search space used is the set of vertices of a D-dimensional unit cube. A specific vertex is "analogous" to a neural network architecture in a real experiment. Training and evaluating a neural network yields a noisy accuracy. Likewise, the simulations assign a noisy *simulated accuracy* (*SA*) to each cube vertex. The *SA* is the fraction of coordinates that are zero, plus a small amount of Gaussian noise ($\mu = 0$, $\sigma = 0.01$, matching the observed noise for neural networks). Thus, the goal is to get close to the optimum, the origin. The sample complexity used was 10k. These simulations are helpful because they complete in milliseconds.

This optimization problem is a simplification of the evolutionary search for the minimum of a multi-dimensional integer-valued paraboloid with bounded support, where the mutations treat the values along each coordinate categorically. If we restrict the domain along each direction to the set $\{0, 1\}$, we are reduced to the unit cube described above. The paraboloid's value at all the cubes corners is just the number of coordinates that are not zero, *i.e*. the scenario above. We mention this connection because searching for the minimum of a paraboloid seems like a more natural choice for a trivial problem ("trivial" compared to architecture search). The simpler unit cube version, however, was chosen because it permits faster computation.

We stress that these simulations are not intended to truly mimic architecture search experiments over the space of neural networks. We used them only as a testing ground for evolving solutions in the presence of noisy evaluations.

S2. Best Evolved Architectures

Figure S1 shows the normal and reduction cells in AmoebaNet models. The "h" labels hidden states. The ops ("avg 3x3", *etc*.) are listed in full form in Section S1.1. Data flows from bottom to top. See the Zoph et al. (2017) for a detailed description of these diagrams and how they are stacked to form full models.

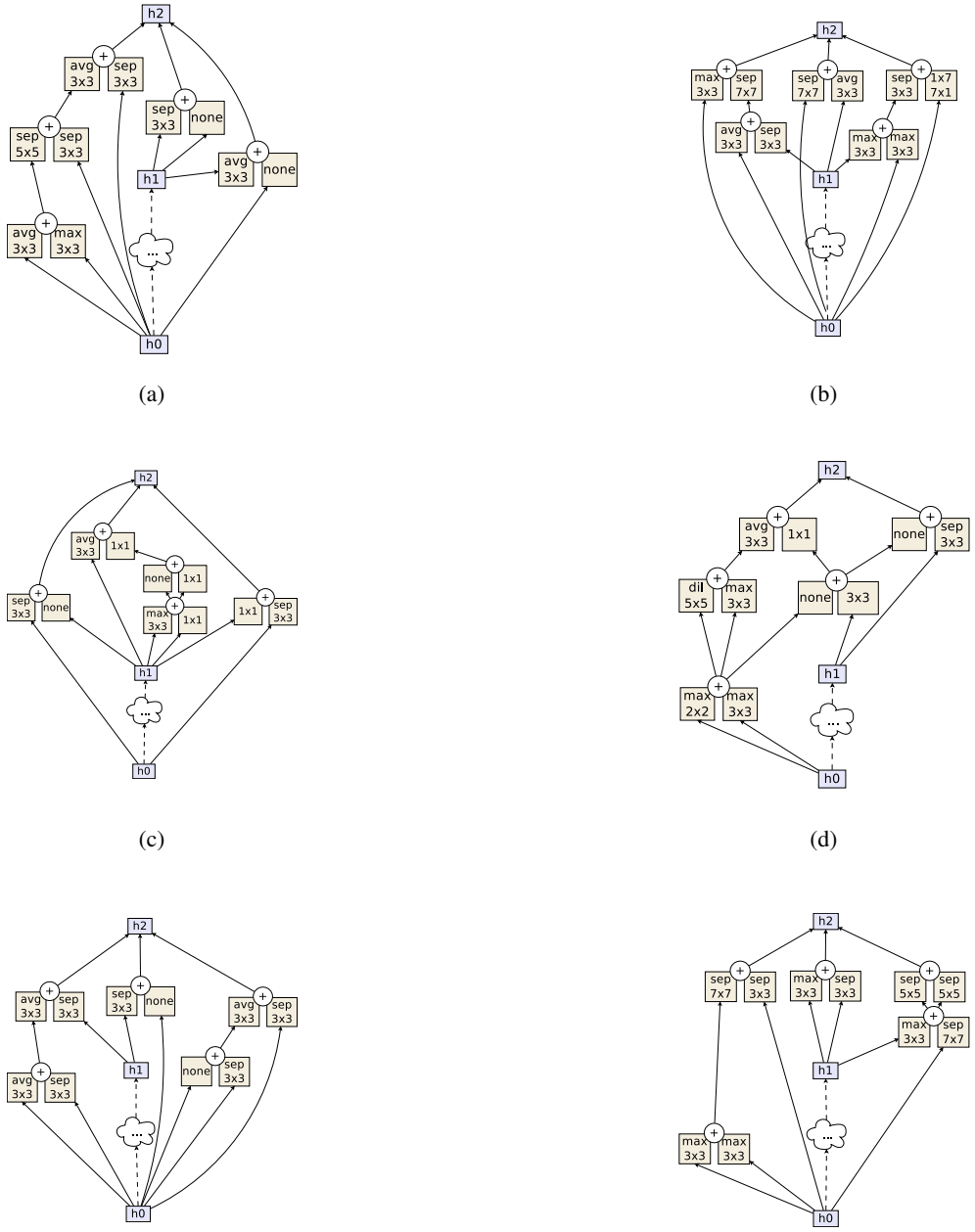

Figure S1. Basic AmoebaNet building blocks. AmoebaNet-A normal (a) and reduction (b) cells. AmoebaNet-B normal (c) and reduction (d) cells. AmoebaNet-C normal (e) and reduction (f) cells.