Regularized Evolution for Image Classifier Architecture Search

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

April 30, 2018

Abstract

The effort devoted to hand-crafting image classifiers has motivated the use of architecture search to discover them automatically. Although evolutionary algorithms have been repeatedly applied to architecture search, the architectures thus discovered have remained inferior to human-crafted ones. Here we show for the first time that artificially-evolved architectures can match or surpass human-crafted and RL-designed image classifiers. In particular, our models—named *AmoebaNets*—achieved a state-of-the-art accuracy of 97.87% on CIFAR-10 and top-1 accuracy of 83.1% on ImageNet. Among mobile-size models, an AmoebaNet with only 5.1M parameters also achieved a state-of-the-art top-1 accuracy of 75.1% on ImageNet. We also compared this method against strong baselines. Finally, we performed platform-aware architecture search with evolution to find a model that trains quickly on Google Cloud TPUs. This method produced an AmoebaNet that won the Stanford DAWNBench competition for lowest ImageNet training cost.

1. Introduction

Until recently, most state-of-the-art (SOTA) convolutional architectures have been manually designed by human experts (Krizhevsky et al., 2012; Ciregan et al., 2012; Lin et al., 2014; Sermanet et al., 2014; He et al., 2014; Chatfield et al., 2014; Simonyan & Zisserman, 2015; Karpathy et al., 2014; Szegedy et al., 2015; Donahue et al., 2015; Jaderberg et al., 2015; He et al., 2016; Zagoruyko & Komodakis, 2016; Larsson et al., 2017; Huang et al., 2017b; Szegedy et al., 2016; 2017; Xie et al., 2017; Chen et al., 2017; Zhang et al., 2017b; Hu et al., 2018). To speed up the process, researchers have looked into automated methods to discover new models (Stanley & Miikkulainen, 2002; Stanley et al., 2009; Andrychowicz et al., 2016; Real et al., 2017; Miikkulainen et al., 2017; Liu et al., 2018; Zoph et al., 2018; Liu et al., 2017; Zoph & Le, 2016; Baker et al., 2017a; Zhong et al., 2018; Cai et al., 2018; Suganuma et al., 2017; Pham et al., 2018). These methods are now collectively known as *architecture search algorithms*. The traditional approach to architecture search is *neuro-evolution of topologies* (Angeline et al., 1994; Stanley & Miikkulainen, 2002; Floreano et al., 2008; Stanley et al., 2009; Fernando et al., 2016; Xie & Yuille, 2017). Improved hardware now allows scaling up these evolutionary algorithms, producing high-quality image classification models (Real et al., 2017; Miikkulainen et al., 2017; Liu et al., 2018). Yet, the evolved architectures have not reached the accuracy and performance of those directly designed by experts.

In this paper, we use a slightly-modified variant of a standard evolutionary algorithm for architecture search. Each architecture is iteratively trained and selected based on how well it performs on a validation set. The best architectures at every iteration are allowed to mutate to generate new ones, keeping a population of models that improves over time. In our implementation, the architectures age and die off from the population, even if they performed well. This aging effect regularizes standard evolution and improves the outcome. We applied this algorithm to the *NASNet search space* (Zoph et al., 2018) and found better architectures than those obtained in that study, which used a reinforcement-learning (RL) algorithm. In particular, the model we discovered—named *AmoebaNet-C*—achieves a top-1 accuracy of 83.1% on ImageNet, matching the current SOTA, and also sets a new SOTA for small models that can fit on a mobile device.[2]

We also applied the evolutionary algorithm to find architectures that train well on a particular hardware platform. For this purpose, we evolved directly on Google Cloud TPUs. The search produced AmoebaNet-B, the new single-model SOTA on CIFAR-10, with an accuracy of 97.87%, reducing

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

[2]Since the writing of this paper, a new study has set a new SOTA on CIFAR-10 and ImageNet by introducing a new data-augmentation technique; they still used AmoebaNet to reach their ImageNet result (Cubuk et al., 2018).

the error from the previous leading result by 8%. We then ran evolution on ImageNet directly and manually extrapolated the changes the evolutionary process was making, to produce AmoebaNet-D. An implementation of AmoebaNet-D won the Stanford DAWNBench competition for the lowest ImageNet training cost (Coleman et al., 2017). In particular, that implementation cost $49.30 to reach 93% top-5 accuracy. This is 16% better than the second-best model in the competition, which was ResNet and which trained on the same hardware[3].

2. Related Work

In recent years, progress has been made in automating the process of searching for good architectures, especially for image classification. In fact, some recent ImageNet SOTA results had been obtained by using learning-based approaches (Zoph et al., 2018; Liu et al., 2017), after a series of studies exploring these methods (Zoph & Le, 2016; Baker et al., 2017a; Zoph et al., 2018; Zhong et al., 2018; Cai et al., 2018).

Some approaches stand out due to their efficiency (Zhong et al., 2018; Suganuma et al., 2017; Pham et al., 2018). This is important because reducing compute cost is one of the main challenges of architecture search. This efficiency, however, may not be entirely due to their algorithm—a smaller search space can help significantly, for example. Platform-aware optimization is a viable approach (Dong et al., 2018). Also, while some got very close to the SOTA (Zhong et al., 2018), actually reaching it might require much more compute power, as it did in some studies (Zoph et al., 2018; Liu et al., 2017). 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., 2017), hypernets (Brock et al., 2018), accuracy prediction (Baker et al., 2017b; Klein et al., 2017; Domhan et al., 2017), warm-starting and ensembling (Feurer et al., 2015), parallelization, reward shaping and early stopping (Zhong et al., 2018) or Net2Net transformations (Cai et al., 2018). 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, showing benefits.

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., 2018). 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. The version we use is regularized, natural, and permits garbage collection.

Other than through evolution and RL, architecture search was also explored with cascade-correlation (Fahlman & Lebiere, 1990), boosting (Cortes et al., 2017; Huang et al., 2017a), hill-climbing (Elsken et al., 2017), MCTS (Negrinho & Gordon, 2017), SMBO (Mendoza et al., 2016; Liu et al., 2017), 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

3.1. Search Space

All experiments use the *NASNet search space* which has previously been used to design high-quality models with an RL algorithm in *the baseline study*, Zoph et al. (2018). The goal is to discover 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. The resulting models have two hyper-parameters that control their size and impact their accuracy: convolution channel depth (F) and cell stacking depth (N). We only used these parameters only to trade accuracy against size, so knowledge of their existence suffices here and we refer the reader to the baseline study for other details.

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. (More details in the Supplementary Material, Section S-1.1; examples in Figure 2).

3.2. Regularized Evolution for Architecture Search

The evolutionary algorithm is a variant of the standard tournament selection method (Goldberg & Deb, 1991). In our

[3]The Stanford DAWNBench public leaderboard can be found at https://dawn.cs.stanford.edu/benchmark/#imagenet-train-cost

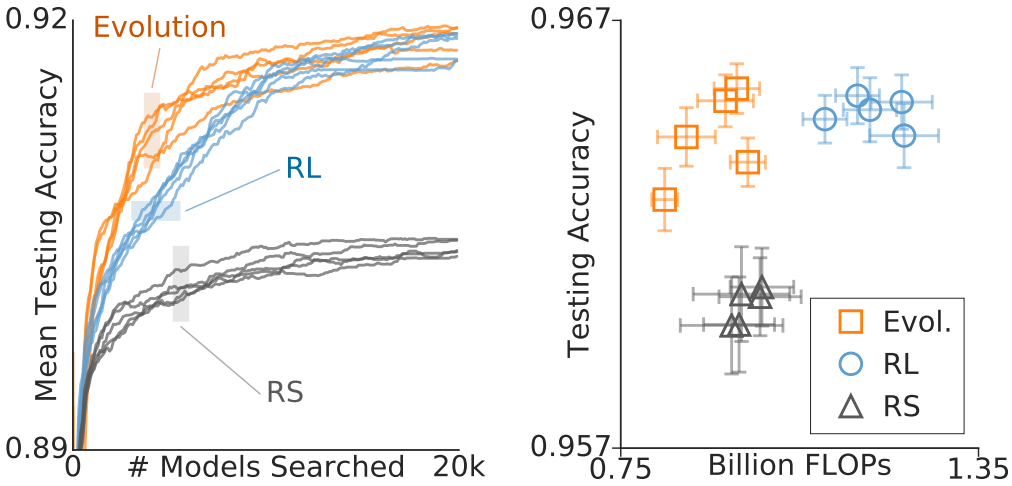

Figure 1. A comparison between evolution, RL, and RS for architecture search. LEFT: Time-course of 5 identical large-scale experiments for each algorithm, showing accuracy before augmentation. Note this plot does not show FLOPs, making evolution and RL look similar toward the end. RIGHT: Results of 5 identical architecture-search experiments for each algorithm, on CIFAR-10. Each marker corresponds to the top models from one experiment, after augmentation. (See also Appendix A).

implementation, a population of P trained models is improved in iterations. At each iteration, 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 oldest model in the population is discarded (*i.e.* the first to have been trained). This is very similar to the implementation in (Real et al., 2017). The only difference is that they discard the worst individual in the sample instead. For RL experiments, we employed the algorithm in the baseline study without changes. (Details in Supplementary Material, Section S-1.2)

4. Experimental Setup

We ran controlled comparisons at scale, ensuring identical conditions for evolution, RL and random search (RS). In particular, all methods used *the same* computer code for network construction, training and evaluation. Experiments always searched on the CIFAR-10 dataset (Krizhevsky & Hinton, 2009). Each experiment ran on 450 K40 GPUs for approximately 7 days.

As in the baseline study, we first performed architecture search over small models (*i.e.* small N and F). We then used the *model augmentation* trick (Zoph et al., 2018) by which we take an architecture discovered by the search (*e.g.* the output of an evolutionary experiment) and turn it into a full-size, accurate model. To accomplish this, we enlarged the model 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 (Krizhevsky & Hinton, 2009; Deng et al., 2009). For consistency, we followed the same procedure as in the baseline study as far as we knew. (More details can be found in the Supplementary Material, Sections S-1.5 and S-1.6).

5. Results

In the following we will establish a comparison between our method against two competitive baselines, reinforcement learning (RL) and random search (RS). We will then provide the key result of our paper: the application of regularized evolution to the ImageNet image classification dataset. Finally, we will show that our method can be used with great flexibility to design models for a particular hardware platform such as Google Cloud TPU. Figure 2 contains diagrams of the architectures evolved.

5.1. Comparison With RL and RS Baselines

The current SOTA for image classifier architecture search had been obtained by RL in the baseline study (Zoph et al., 2018) (recently matched by another learning-based technique (Liu et al., 2017)). Here we seek to compare our evolutionary approach against their RL algorithm. We performed large-scale side-by-side architecture-search experiments on CIFAR-10. We first optimized the hyper-parameters of the two approaches independently (details in Supplementary Material, Section S-2). Then we ran 5 repeats of each of the two algorithms—and also of random search (RS). Figure 1 (left) shows the accuracy as the experiment progresses,

Table 1. ImageNet classification results for AmoebaNets compared to hand-designs (top) and other methods (middle). Evolution-based methods are marked with a *.

Model	# Params	# Multiply-adds	Top-1/Top-5 Accuracy (%)
Inception V3 (Szegedy et al., 2016)	23.8M	5.72B	78.8 / 94.4
Xception (Chollet, 2017)	22.8M	8.37B	79.0 / 94.5
Inception ResNet V2 (Szegedy et al., 2017)	55.8M	13.2B	80.4 / 95.3
ResNeXt-101 (64x4d) (Xie et al., 2017)	83.6M	31.5B	80.9 / 95.6
PolyNet (Zhang et al., 2017b)	92.0M	34.7B	81.3 / 95.8
Dual-Path-Net-131 (Chen et al., 2017)	79.5M	32.0B	81.5 / 95.8
Squeeze-Excite-Net (Hu et al., 2018)	145.8M	42.3B	82.7 / 96.2
Squeeze-Excite-Net v2 (Hu et al., 2018)	–	–	**83.1 / 96.4**
GeNet-2 (Xie & Yuille, 2017)*	156M	–	72.1 / 90.4
Block-QNN-B (N=3) (Zhong et al., 2018)*	–	–	75.7 / 92.6
Hierarchical (2, 64) (Liu et al., 2018)*	64M	–	79.7 / 94.8
NASNet-A (N=6, F=168) (Zoph et al., 2018)	88.9M	23.8B	82.7 / 96.2
PNASNet-5 (N=4, F=216) (Liu et al., 2017)	86.1M	25.0B	82.9 / 96.2
AmoebaNet-C (N=6, F=168)*	85.5M	22.5B	82.7 / 96.1
AmoebaNet-C (N=6, F=228)*	155.3M	41.1B	**83.1** / 96.3

Table 2. ImageNet classification results in the *mobile* setting. Notation is as in Table 1.

Model	# Params	# Multiply-adds	Top-1/Top-5 Accuracy (%)
MobileNetV1 (Howard et al., 2017)	4.2M	575M	70.6 / 89.5
ShuffleNet (2x) (Zhang et al., 2017a)	4.4M	524M	70.9 / 89.8
MobileNetV2 (1.4) (Sandler et al., 2018)	6.9M	585M	74.7 / –
NASNet-A (N=4, F=44) (Zoph et al., 2018)	5.3M	564M	74.0 / 91.3
PNASNet-5 (N=3, F=54) (Liu et al., 2017)	5.1M	588M	74.2 / 91.9
AmoebaNet-C (N=4, F=44)*	5.1M	535M	75.1 / 92.1
AmoebaNet-C (N=4, F=50)*	6.4M	570M	75.7 / 92.4

highlighting that evolution is consistently faster at the earlier stages. It is important to note that this graph does not present the computation requirements of the models. When these are included, it is clear that evolution produces more efficient architectures, as can be seen in Figure 1 (right), which compares the augmented models. Evolved architectures have higher accuracy (and similar FLOPs) than those obtained with RS, and lower FLOPs (and similar accuracy) than those obtained with RL. (More controlled comparisons in alternate datasets and modified search spaces can be found in Appendix A.)

5.2. ImageNet Results

We selected a high-quality model from the evolution experiments in Section 5.1 based on its CIFAR-10 performance after augmentation (selection details in Supplementary Material, Section S-1.5). We name it AmoebaNet-C (Figure 2). When retrained on ImageNet, this model attained SOTA performance for top-1 validation accuracy (Table 1).

The models that reach such high accuracy, evolved or not, tend to be very large. Their parameter counts are well above 100 million. It therefore becomes interesting to compare how well the models do when they are constrained to fewer parameters instead. A natural constraint is to pick models that fit on a mobile phone (Howard et al., 2017; Zhang et al., 2017a). AmoebaNet-C sets a new SOTA in that regime (Table 2).

5.3. Evolving Models with Low-Cost Training

The number of parameters is only an approximation to neural-network efficiency. Others are number of connections, total RAM required, and training time, for example. A good overall efficiency metric may be to simply consider the monetary cost of training a neural network to a given accuracy. Google Cloud introduced a hardware designed to efficiently train neural networks, their TPUv2 chips. We evolved on this hardware at a larger compute scale on CIFAR-10, while exploring a bigger search space: the cells could contain ops from an increased set of choices (including various types of convolutions and pooling). Other than this, the same search and augmentation process as above was used. We highlight that our selection process was done across experiments (*i.e. without* testing the top model

Table 3. CIFAR-10 results for AmoebaNets compared to hand-designs (top) and other methods (middle). Evolution-based methods are marked with a *.

Model	# Params	Test Error (%)
DenseNet-BC (k = 24) (Huang et al., 2017b)	15.3 M	3.62
ResNeXt-29, 16x64d (Xie et al., 2017)	68.1 M	3.58
DenseNet-BC (L=100, k=40) (Huang et al., 2017b)	25.6 M	3.46
Shake-Shake 26 2x96d (Gastaldi, 2017) + cutout (DeVries & Taylor, 2017)	26.2 M	2.56
PyramidNet + Shakedrop (Han et al., 2016; Yamada et al., 2018)	26.0 M	2.31
Evolving DNN (Miikkulainen et al., 2017)*	–	7.30
MetaQNN (top model) (Baker et al., 2017a)	–	6.92
CGP-CNN (ResSet) (Suganuma et al., 2017)*	1.68 M	5.98
Large Scale Evolution (Real et al., 2017)*	5.4 M	5.40
EAS (Cai et al., 2018)	23.4 M	4.23
SMASHv2 (Brock et al., 2018)	16 M	4.03
Hierarchical (2, 64) (Liu et al., 2018)*	15.7 M	3.75 ± 0.12
Block-QNN-A, N=4 (Zhong et al., 2018)	–	3.60
PNASNet-5 (N=3, F=48) (Liu et al., 2017)	3.2 M	3.41 ± 0.09
NASNet-A (N=6, F=32) (Zoph et al., 2018)	3.3 M	3.41
NASNet-A (N=6, F=32) (Zoph et al., 2018) + cutout	3.3 M	2.65
NASNet-A (N=7, F=96) (Zoph et al., 2018) + cutout	27.6 M	2.40
AmoebaNet-B (N=6, F=36)*	2.8 M	3.37 ± 0.04
AmoebaNet-B (N=6, F=112)*	26.7 M	3.04 ± 0.04
AmoebaNet-B (N=6, F=128)*	34.9 M	2.98 ± 0.05
AmoebaNet-B (N=6, F=36) + cutout *	2.8 M	2.55 ± 0.05
AmoebaNet-B (N=6, F=112) + cutout*	26.7 M	2.21 ± 0.04
AmoebaNet-B (N=6, F=128) + cutout*	34.9 M	$\mathbf{2.13 \pm 0.04}$

in *each* experiment), so as to not incur in overfitting. We name the resulting model AmoebaNet-B (Figure 2). More details can be found in the Supplementary Material, Section S-1.4.

AmoebaNet-B sets a new SOTA for test accuracy on CIFAR-10, even though the evolution process had never seen the CIFAR-10 test set and even though we report an average over training runs (of course, our top result is better than the average we report). As can be seen in Table 3, the test error attained is 8% smaller than the previous SOTA. Even though the augmented model has 34.9 M parameters, it is very efficient: it trained in less than 24 hours on one machine.

We then continued to evolve on TPU, starting with AmoebaNet-B, and progressively increasing the training time for each model. Now evolution was performed directly on ImageNet, but withholding the validation set from the algorithm, so as to not overfit. From this, we observed that evolution was starting to replace dilated convolutions with regular convolutions. We therefore decided to manually replace all convolutions in the model this way, extrapolating the evolutionary process. The result was AmoebaNet-D (Figure 2).

An implementation of AmoebaNet-D was submitted to the lowest-ImageNet-training-cost category in the recent Stanford DAWNBench competition. The authors of the submission first trained it several times and selected a worse-than-median model, also to avoid overfitting. This model won the category with a 16% margin over the second best submission.

6. Discussion

The comparison study (Figure 1, left) suggests 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.*). Algorithm speed would be more important when exploring larger spaces, where reaching the optimum may require more compute than is available. In that regime, evolution might shine. The speed of individual models produced is also relevant. Figure 1 (right) demonstrates that evolved models are faster (lower FLOPs). 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

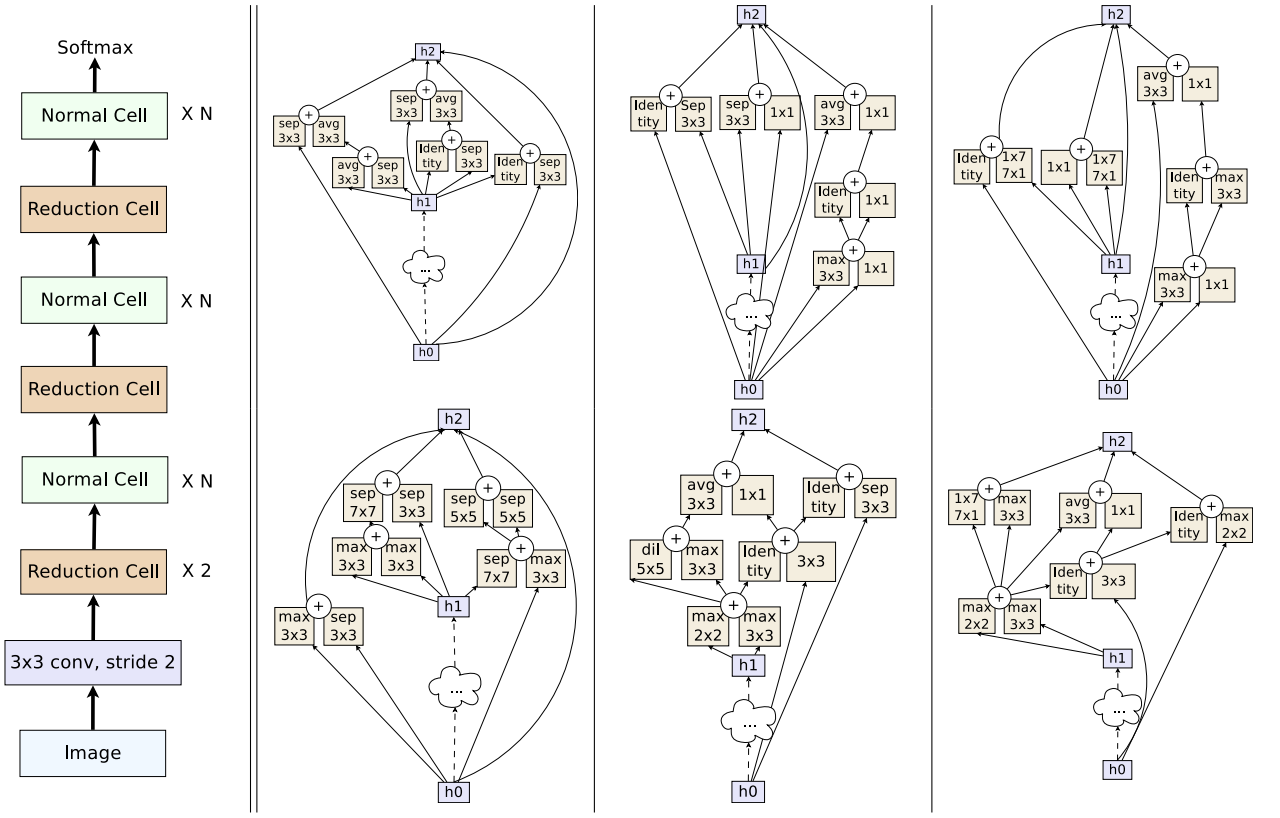

Figure 2. Architectures of overall model and cells. From left to right: outline of the overall model (Zoph et al., 2018) and diagrams for the cell architectures discovered by evolution: AmoebaNet-C, AmoebaNet-B, and AmoebaNet-D. The three normal cells are on the top row and the three reduction cells are on the bottom row.

this speculation is beyond the scope of this paper.

Another desirable feature of evolutionary algorithms is their simplicity. The implementation used here can be described in one sentence: "keep a population of size N, at each step kill the oldest and copy-mutate the best of K random individuals". Note that only two meta-parameters are involved. This is in contrast to learning-based approaches, which require a controller—such as an LSTM—that must be trained in the correct meta-parameter regime (learning rate, learning-rate decay, batching). If RL is used, additional meta-parameters may be necessary to control the greediness, possible replay buffer, *etc*. On the other hand, learning-based approaches have the ability to generalize, which could be extremely powerful. This power comes at some cost: learning must train for a while to find patterns before taking advantage of them. Evolution, however, can in principle start from an arbitrary initial condition (such as a good hand-crafted model) and evolve right away—it does not matter how a point in the search space was reached. This is what allowed us to start from AmoebaNet-B in order to discover AmoebaNet-D.

Regularization—*i.e*. removing the oldest individual instead of the worst one—seemed advantageous in additional small-scale experiments, shown in Figure 3, and presented in more detail in Appendix B. These were carried out on CPU instead of GPU, and using a gray-scale version of CIFAR-10, to reduce compute requirements. We can speculate that regularization may help by navigate the training noise in evolution experiments, as follows. Under regularized evolution, all models have a short lifespan. Yet, populations improve over longer timescales. 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, *i.e*. 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. However, 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 benefits from regularization is conjecture. We leave its verification to future work.

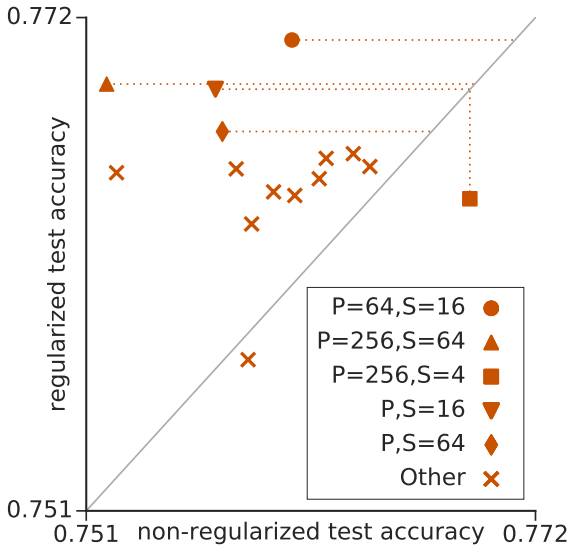

Figure 3. Comparison between our regularized implementation and the non-regularized one from (Real et al., 2017), for different population sizes (P) and sample sizes (S), showing that regularization tends to be beneficial (most markers are above the $y = x$ line). (See also Appendix B).

7. Conclusion

We have shown for the first time that neural-network architecture evolution can match and surpass hand-designs. After a controlled comparison against an RL architecture-search baseline, we transferred a high-quality model evolved on CIFAR-10—AmoebaNet-C—to ImageNet, where it attained state-of-the-art (SOTA) top-1 accuracy and set a new SOTA for mobile-sized models. We then evolved on Google Cloud TPUs at scale to produce AmoebaNet-B, a model that sets a new SOTA for CIFAR-10 after training for less than 24 hours in one machine. Finally, we further evolved with longer training times and manually extrapolated the process to find AmoebaNet-D, which won the Stanford DAWN-Bench competition for lowest training cost on ImageNet.

This study is an empirical step toward asserting the usefulness of evolutionary algorithms for image classifier architecture search, and draws from a long history of research on the subject. We hope that future work will show achievements with more generality and will further elucidate the relationship of evolutionary and learning-based approaches to expose the merits of each. 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 their 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; Chris Ying for his work open-sourcing and helping tune AmoebaNet models and for his help with specialized hardware, Barret Zoph and Vijay Vasudevan for help with the code and experiments used in Zoph et al. (2018), 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, and Ruoming Pang for helpful discussions; and the larger Google Brain team.

Appendix A: Evolution *vs*. Reinforcement Learning

A-1. Motivation

In this appendix, we will extend the comparison between evolution and reinforcement learning (RL) from Section 5.1. As described in Sections 1 and 2, evolutionary algorithms and RL have been applied recently to the field of architecture search. Yet, comparison is difficult because studies tend to use novel search spaces, 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[4] and different hardware platforms that affect algorithm run-time[5]. Accounting for all these factors, we will compare the two approaches in a variety of image classification contexts. To achieve statistical confidence, we will present repeated experiments without sampling bias.

A-2. Setup

All evolution and RL experiments used the NASNet search space design (Zoph et al., 2018). Within this design, we define three concrete search spaces that differ in the value of C and in the number of ops allowed (see Section 3.1). In order of increasing size, we will refer to them as SP-I (*e.g*. Figure A-1f), SP-II, and SP-III (*e.g*. Figure A-1g). SP-I is the exact variant used in the baseline study and in most of the main text of this paper. SP-II has more allowed ops (more types of convolutions, for example). SP-III allows for larger tree structures within the cells (details in Supplementary Material, Section S-3).

The evolutionary algorithm is the same as that in the main text. The RL algorithm is the one used in 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.

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 experiments in this appendix were performed at a smaller compute scale than in the main text, to reduce resource usage: we used gray-scale versions of popular datasets ("G-Imagenet" instead of ImageNet; Supplementary Material, Section S-1.3), we ran on CPU instead of GPU and trained relatively small models for a shorter time. Where unstated, the experiments ran on SP-I and G-CIFAR.

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

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

A-3. Findings

We first optimized the meta-parameters for evolution and for RL by running experiments with each algorithm, repeatedly, under each condition. Figure A-1a shows that neither approach was very sensitive. Still, this was a necessary step to ensure both methods are treated fairly. We then compared the algorithms in 5 different contexts by swapping the dataset or the search space (Figure A-1b). 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 A-1c and A-1d, 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 A-1f and A-1g illustrate the top architectures from SP-I and SP-III, respectively. Regardless of context, Figure A-1e indicates that accuracy under evolution increases significantly faster than RL at the initial stage. This stage was not accelerated by higher RL learning rates.

A-4. Outcome

The main text provides a comparison between algorithms for image classifier architecture search in the context of the SP-I search space on CIFAR-10, at scale. This appendix extends those results, varying the dataset and the search space by running many small experiments. These controlled comparisons were done on search spaces designed for RL and included experiments with the same parameters as the original study. The outcome confirms the conclusions of the main text. Taken together, results show that evolution matches or surpasses RL in the final outcome and reaches that outcome faster. This is the first controlled comparison between evolution and RL in the context of image classifier architecture search

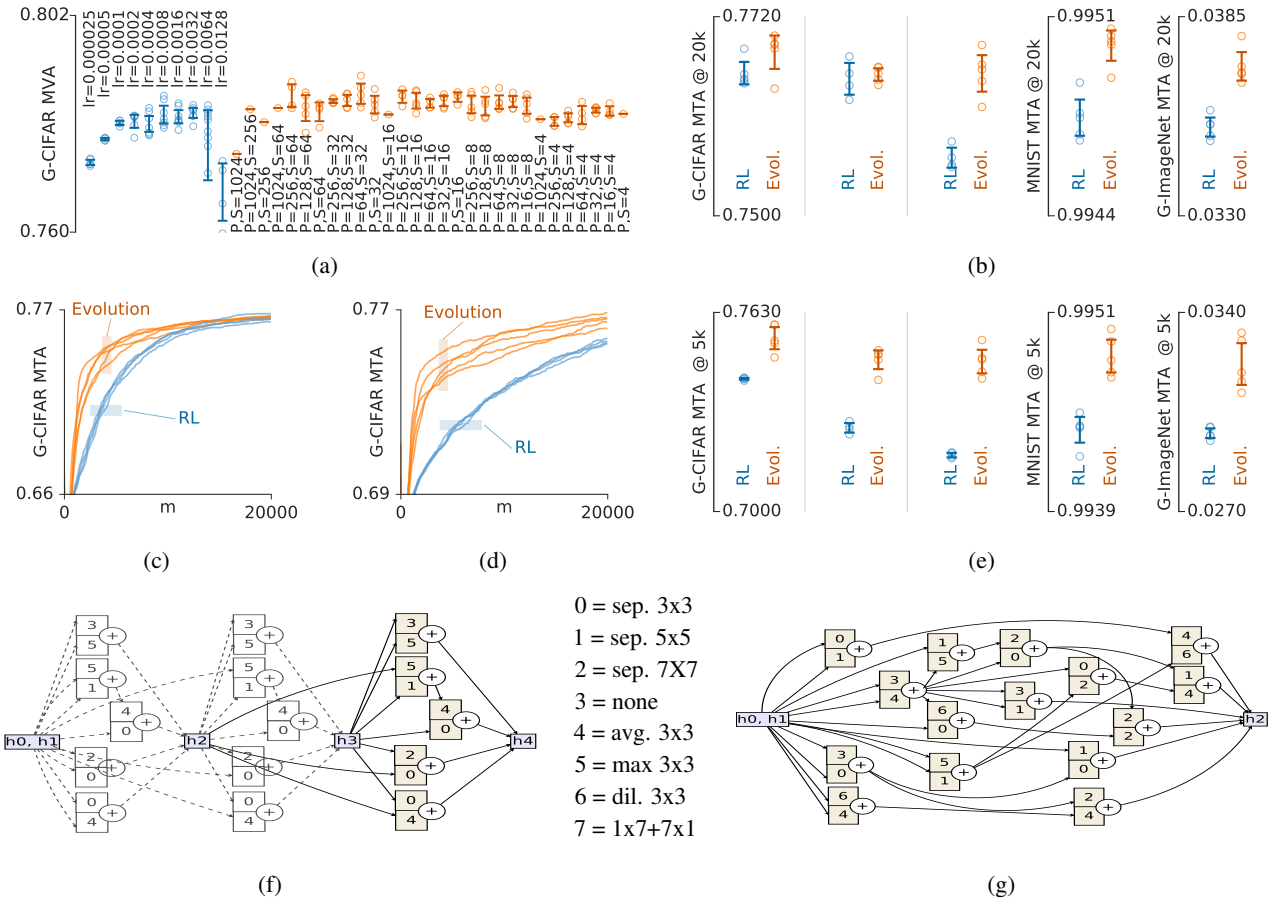

Figure A-1. Evolution *vs*. RL in different contexts (Section A-3). Plots show repeated evolution (orange) and RL (blue) experiments side-by-side. **(a)** Summary of hyper-parameter optimization experiments on G-CIFAR. We swept the learning rate (*lr*) for RL (left) and the population size (*P*) and 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 $\pm 2\,SEM$ 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. The "h" labels hidden states. The ops ("avg 3x3", *etc*.) are listed in full form in Section S-1.1. 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.

Appendix B: Regularized *vs*. Non-regularized Evolution

B-1. Motivation

In this appendix, we will extend the comparison between regularized evolution (*RE*) and non-regularized evolution (*NRE*) from Figure 3. As was described in Section 3.2, the evolutionary algorithm used in this paper keeps the population size constant by always removing the oldest individual whenever a new individual is added; we will refer to this algorithm as *RE*. A recent paper used a similar method but kept the population size constant by removing the worst individual in each tournament (Real et al., 2017); we will refer to that algorithm as *NRE*. This appendix will show how these two algorithms compare in a variety of contexts.

B-2. Setup

The search spaces and datasets were the same as in Appendix A, Section A-2.

B-3. Findings

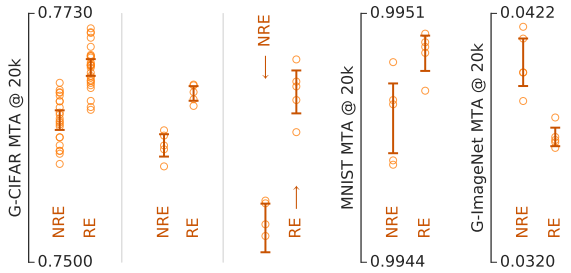

Figure B-1. A comparison of *NRE* and *RE* under 5 different contexts, spanning different datasets and search spaces: G-CIFAR/SP-I, G-CIFAR/SP-II, G-CIFAR/SP-III, MNIST/SP-I and G-ImageNet/SP-I, shown from left to right. For each context, we show the final *MTA* of a few *NRE* and a few *RE* experiments (circles) in adjacent columns. We superpose ± 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.

We performed experiments in 5 different search space–dataset contexts. In each context, we ran several repeats of evolutionary search using *NRE* and *RE* (Figure B-1). 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.

Additionally, we performed three experiments comparing *RE* and *NRE* at scale, under the same conditions as in the main text. The results, which can be seen in Figure B-2, provide some verification that the results from smaller CPU experiments in the previous paragraph generalize to the large-compute regime.

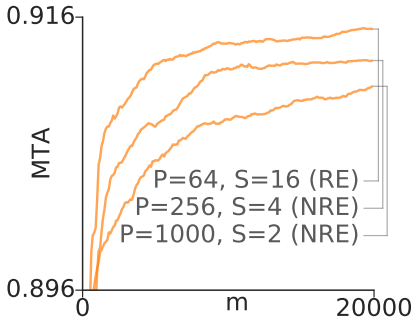

Figure B-2. A comparison of *RE* and *NRE* at scale. These experiments use the same conditions as the main text (including dataset, search space, resources and duration). From top to bottom: an *RE* experiment with the best *RE* meta-parameters from Figure 3), an analogous *NRE* experiment and an *NRE* experiment with the meta-parameters used in Real et al. (2017). These accuracy values are not meaningful in absolute terms, as the models need to be augmented to reach their maximum accuracy (Section 4).

B-4. Outcome

Main text Section 6 suggested that removing the oldest individual from the population has a regularizing effect[6] by preventing over-fitting through the introduction of new information to the search process: successful architectures must remain good after retraining. Main text Figure 3 showed that *RE* tends to perform better than *NRE* across various parameters for a fixed search space–dataset context. This appendix extends those results to show that the conclusion holds across various contexts. Such robustness is desirable for computationally demanding architecture search experiments, where we cannot always afford many runs to optimize the meta-parameters.

[6] https://en.wikipedia.org/wiki/Regularization_(mathematics)

References

Andrychowicz, Marcin, Denil, Misha, Gomez, Sergio, Hoffman, Matthew W, Pfau, David, Schaul, Tom, and de Freitas, Nando. Learning to learn by gradient descent by gradient descent. In *Advances in Neural Information Processing Systems*, pp. 3981–3989, 2016.

Angeline, Peter J, Saunders, Gregory M, and Pollack, Jordan B. An evolutionary algorithm that constructs recurrent neural networks. *IEEE transactions on Neural Networks*, 5(1):54–65, 1994.

Baker, Bowen, Gupta, Otkrist, Naik, Nikhil, and Raskar, Ramesh. Designing neural network architectures using reinforcement learning. In *International Conference on Learning Representations*, 2017a.

Baker, Bowen, Gupta, Otkrist, Raskar, Ramesh, and Naik, Nikhil. Accelerating neural architecture search using performance prediction. *International Conference on Learning Representations, Workshop Track*, 2017b.

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. In *International Conference on Learning Representations*, 2018.

Cai, Han, Chen, Tianyao, Zhang, Weinan, Yu, Yong, and Wang, Jun. Efficient architecture search by network transformation. In *AAAI Conference on Artificial Intelligence*, 2018.

Chatfield, Ken, Simonyan, Karen, Vedaldi, Andrea, and Zisserman, Andrew. Return of the devil in the details: Delving deep into convolutional nets. *British Machine Vision Conference*, 2014.

Chen, Yunpeng, Li, Jianan, Xiao, Huaxin, Jin, Xiaojie, Yan, Shuicheng, and Feng, Jiashi. Dual path networks. In *Advances in Neural Information Processing Systems*, pp. 4470–4478, 2017.

Chollet, François. Xception: Deep learning with depthwise separable convolutions. *Conference on Computer Vision and Pattern Recognition*, 2017.

Ciregan, Dan, Meier, Ueli, and Schmidhuber, Jürgen. Multi-column deep neural networks for image classification. In *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3642–3649. IEEE, 2012.

Coleman, Cody, Narayanan, Deepak, Kang, Daniel, Zhao, Tian, Zhang, Jian, Nardi, Luigi, Bailis, Peter, Olukotun, Kunle, Ré, Chris, and Zaharia, Matei. Dawnbench: An end-to-end deep learning benchmark and competition. *Training*, 100(101):102, 2017.

Cortes, Corinna, Gonzalvo, Xavi, Kuznetsov, Vitaly, Mohri, Mehryar, and Yang, Scott. Adanet: Adaptive structural learning of artificial neural networks. In *International Conference on Machine Learning*, 2017.

Cubuk, Ekin D, Zoph, Barret, Mane, Dandelion, Vasudevan, Vijay, and Le, Quoc V. Autoaugment: Learning augmentation policies from data. *arXiv preprint arXiv:1805.09501*, 2018.

Deng, Jia, Dong, Wei, Socher, Richard, Li, Li-Jia, Li, Kai, and Fei-Fei, Li. Imagenet: A large-scale hierarchical image database. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2009.

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

Domhan, Tobias, Springenberg, Jost Tobias, and Hutter, Frank. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2017.

Donahue, Jeffrey, Anne Hendricks, Lisa, Guadarrama, Sergio, Rohrbach, Marcus, Venugopalan, Subhashini, Saenko, Kate, and Darrell, Trevor. Long-term recurrent convolutional networks for visual recognition and description. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2625–2634, 2015.

Dong, Jin-Dong, Cheng, An-Chieh, Juan, Da-Cheng, Wei, Wei, and Sun, Min. Ppp-net: Platform-aware progressive search for pareto-optimal neural architectures. *International Conference on Learning Representations, Workshop Track*, 2018.

Elsken, Thomas, Metzen, Jan-Hendrik, and Hutter, Frank. Simple and efficient architecture search for convolutional neural networks. *International Conference on Learning Representations, Workshop Track*, 2017.

Fahlman, Scott E and Lebiere, Christian. The cascade-correlation learning architecture. In *Advances in Neural Information Processing Systems*, pp. 524–532, 1990.

Fernando, Chrisantha, Banarse, Dylan, Reynolds, Malcolm, Besse, Frederic, Pfau, David, Jaderberg, Max, Lanctot, Marc, and Wierstra, Daan. Convolution by evolution: Differentiable pattern producing networks. In *GECCO*, pp. 109–116. ACM, 2016.

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 *Advances in Neural Information Processing Systems*, 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. *International Conference on Learning Representations, Workshop Track, and 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. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian. Spatial pyramid pooling in deep convolutional networks for visual recognition. In *european conference on computer vision*, pp. 346–361. Springer, 2014.

He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, 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. *IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

Huang, Furong, Ash, Jordan, Langford, John, and Schapire, Robert. Learning deep resnet blocks sequentially using boosting theory. *Advances on Neural Information Processing Systems*, 2017a.

Huang, Gao, Liu, Zhuang, Weinberger, Kilian Q, and van der Maaten, Laurens. Densely connected convolutional networks. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2017b.

Jaderberg, Max, Simonyan, Karen, Zisserman, Andrew, et al. Spatial transformer networks. In *Advances in neural information processing systems*, pp. 2017–2025, 2015.

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.

Karpathy, Andrej, Toderici, George, Shetty, Sanketh, Leung, Thomas, Sukthankar, Rahul, and Fei-Fei, Li. Large-scale video classification with convolutional neural networks. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 1725–1732, 2014.

Klein, Aaron, Falkner, Stefan, Springenberg, Jost Tobias, and Hutter, Frank. Learning curve prediction with bayesian neural networks. *International Conference on Learning Representations*, 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.

Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey E. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, 2012.

Larsson, Gustav, Maire, Michael, and Shakhnarovich, Gregory. Fractalnet: Ultra-deep neural networks without residuals. *International Conference on Learning Representations*, 2017.

Lin, Min, Chen, Qiang, and Yan, Shuicheng. Network in network. *International Conference on Learning Representations*, 2014.

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*, 2017.

Liu, Hanxiao, Simonyan, Karen, Vinyals, Oriol, Fernando, Chrisantha, and Kavukcuoglu, Koray. Hierarchical representations for efficient architecture search. In *International Conference on Learning Representations*, 2018.

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

Pham, Hieu, Guan, Melody Y., Zoph, Barret, Le, Quoc V., and Dean, Jeff. Faster discovery of neural architectures by searching for paths in a large model. *International Conference on Learning Representations, Workshop Track*, 2018.

Real, Esteban, Moore, Sherry, Selle, Andrew, Saxena, Saurabh, Suematsu, Yutaka Leon, Le, Quoc, and Kurakin, Alex. Large-scale evolution of image classifiers. In *International Conference on Machine Learning*, 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 *Advances in Neural Information Processing Systems*, 2016.

Sermanet, Pierre, Eigen, David, Zhang, Xiang, Mathieu, Michaël, Fergus, Rob, and LeCun, Yann. Overfeat: Integrated recognition, localization and detection using convolutional networks. *International Conference on Learning Representations*, 2014.

Simmons, Joseph P, Nelson, Leif D, and Simonsohn, Uri. False-positive psychology: Undisclosed flexibility in data collection and analysis allows presenting anything as significant. *Psychological Science*, 22(11):1359–1366, 2011.

Simonyan, Karen and Zisserman, Andrew. Very deep convolutional networks for large-scale image recognition. *International Conference on Learning Representations*, 2015.

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. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 2017.

Szegedy, Christian, Liu, Wei, Jia, Yangqing, Sermanet, Pierre, Reed, Scott, Anguelov, Dragomir, Erhan, Dumitru, Vanhoucke, Vincent, and Rabinovich, Andrew. Going deeper with convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

Szegedy, Christian, Vanhoucke, Vincent, Ioffe, Sergey, Shlens, Jon, and Wojna, Zbigniew. Rethinking the inception architecture for computer vision. In *IEEE Conference on Computer Vision and Pattern Recognition*, 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 Conference on Artificial Intelligence*, volume 4, pp. 12, 2017.

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

Xie, Lingxi and Yuille, Alan. Genetic CNN. In *International Conference on Computer Vision*, 2017.

Xie, Saining, Girshick, Ross, Dollár, Piotr, Tu, Zhuowen, and He, Kaiming. Aggregated residual transformations for deep neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

Yamada, Yoshihiro, Iwamura, Masakazu, and Kise, Koichi. Shakedrop regularization. *International Conference on Learning Representations, Workshop Track*, 2018.

Zagoruyko, Sergey and Komodakis, Nikos. Wide residual networks. In *British Machine Vision Conference*, 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 *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3900–3908. IEEE, 2017b.

Zhong, Zhao, Yan, Junjie, and Liu, Cheng-Lin. Practical network blocks design with q-learning. In *AAAI Conference on Artificial Intelligence*, 2018.

Zoph, Barret and Le, Quoc V. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations*, 2016.

Zoph, Barret, Vasudevan, Vijay, Shlens, Jonathon, and Le, Quoc V. Learning transferable architectures for scalable image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

Supplementary Material

S-1. Detailed Methods

S-1.1. Search Space Details

Section 3.1 introduced the search space in outline. In the language presented in Section 3, the space 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.). To find AmoebaNet-B, 19 possible ops were allowed (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).

S-1.2. Architecture Search Algorithm Details

For all three algorithms—evolution, reinforcement learning (RL) and random search (RS)—the initial models produced have random architectures. That is, the initial models have cells that are constructed with uniformly random ops, which draw from uniformly random inputs, subject to the constraint that no loops should be formed. For RS, all models are constructed this way throughout the entirety of the experiment.

S-1.3. 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 the rest constituted the training set. The standard validation set of 50k was used for testing.
- G-ImageNet: a grayscaled subset of ImageNet. The original images were averaged across channels and re-sized to 32x32. We generated a training set with 200k random images from the standard training set. We also held out 10k images from it for validation. The standard validation set of 50k was used for testing.

S-1.4. Experiment Setup Details

The following completes the description of experiments in Section 4. Each experiment ended when 20k models were trained. Each model trained for 25 epochs. $C = 5, N = 3$ and $F = 24$. To compare different approaches, it was important that the results be independent of the hardware type and network usage. To achieve such hardware-independence, a fixed number of training-steps-per-model and a fixed number of models-per-experiment were used, causing variable training times. Yet, the observed advantage of the evolutionary approach was not due to longer training times. In fact, the evolutionary approach ran faster experiments with *shorter* training times than RL—as may be guessed from the efficiency of the final models shown in Figure 1 (right).

Section 5.3 introduced a new setup for evolving on Google Cloud TPUs. This was the same as those in Section 4, except that 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). The experiment ran on 225 Google Cloud TPUs for 5 days and trained 27k models total. Each TPU supported 4 workers.

S-1.5. Augmented Model Selection

To compare augmented models side-by-side (Figure 1, right), 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 find AmoebaNet-C (Table 1), we selected the model with the best tradeoff between complexity and CIFAR-10 test accuracy among evolution models in Figure 1. Note that using test accuracy was reasonable because the architecture is selected for ImageNet transfer, and not for presenting results on CIFAR-10 (the model used for the CIFAR-10 table was selected by validation, as described in the next paragraph).

To find AmoebaNet-B (Table 3), we selected $K = 100$ models from the TPU experiment. 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 highest validation accuracy. We then re-augmented this model with the (N, F) values in Table 3. Note that we report all 3 sizes; as the size increases, the mean accuracy increases as expected, so there is little risk of overfitting due to evaluating 3 times (which is standard reporting methodology anyway). We trained each size 8 times on CIFAR-10 to measure the *mean* testing accuracy. Using the mean (as opposed to the maximum) ensures there is no overfitting from these 8 repeats.

S-1.6. Augmented Model Training

To train augmented models on CIFAR-10, we followed similar training, data augmentation, and evaluation procedures to those in (Zoph et al., 2018). We used one Google Cloud TPUv2 with batch size 128, SGD with momentum optimizer and momentum rate set to 0.9, L2 weight regularization with rate $5x10^{-4}$, initial learning rate of 0.024 with cosine decay to zero over 600 epochs, and ScheduledDropPath (Zoph et al., 2018) with a final keep-probability of 0.7. During training, the model used an auxiliary softmax classifier 2/3 down the network weighted by 0.5. To report validation accuracy, the held out validation was not included in the training. To report testing accuracy, the full training set was used.

To train augmented models on ImageNet we adjusted the size so that the number of trainable parameters in the model is comparable to others from the literature. For mobile size models, we used 224x224 input images and set the reduction size *F* to 44 and 54—to compare against PNASNet-5 (N=3, F=54) and MobileNetV2 (1.4), respectively. For large-size models, we used 331x331 input images and set the reduction size *F* to 168 (to compare against PNASNet-5 N4F216) and to 228 (the biggest model supported by P100 GPUs). We followed similar training, data augmentation, and evaluation procedures to those in (Szegedy et al., 2016). We used distributed synchronous SGD with 100 P100 GPUs. 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 softmax classifier weighted by 0.4. We applied dropout to the final softmax layer with probability 0.5. We apply ScheduledDropPath (Zoph et al., 2018) with a final keep-probability of 0.7. That is, we randomly drop out each connection edge in the cell architecture diagram with probability that increases linearly over the course of the training, reaching at the end a 0.7 chance that any edge will be kept. The learning rate started at 0.001 and decayed every 2 epochs with rate 0.97. Due to resource limitations, we applied the same hyper-parameters to all models we reported in tables 1 and 2 without further hyper-parameter tuning.

Where possible, we report our model's test error as $\mu \pm 2 \times$ SEM. This applies to the uncertainties in the tables and the error bars in the figures. We report inference number of parameters and FLOPs.

S-2. Baseline Comparison Addenda

A comparison between two given algorithms will be biased if the meta-parameters are not equally well optimized for both. We took steps to ensure that the meta-parameters for RL were at least as optimized as those for evolution. We started with the meta-parameters from the baseline study (under identical conditions) and fine-tuned them by sweeping the *lr* until we saw the accuracy decline at both extremes. 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 is the same value as that found in the smaller-scale experiments of Appendix A.

In order to avoid selection bias, the experiment repeats plotted in Figure 1 do not include the actual runs from the optimization stage described here. Only the meta-parameters found carry over. This was a decision made a priori.

S-3. Supplementary Appendix Material

Section A-2 performs experiments using 3 distinct 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: the same space as in the main text, already defined in Section S-1.1. See also Figure A-1f;
- 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 A-1g.

Each appendix 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. Each experiment used 450 CPU workers and lasted 2–5 days.