Large-Scale Evolution of Image Classifiers

Esteban Real [1] **Sherry Moore** [1] **Andrew Selle** [1] **Saurabh Saxena** [1]
Yutaka Leon Suematsu [2] **Jie Tan** [1] **Quoc V. Le** [1] **Alexey Kurakin** [1]

Abstract

Neural networks have proven effective at solving difficult problems but designing their architectures can be challenging, even for image classification problems alone. Our goal is to minimize human participation, so we employ evolutionary algorithms to discover such networks automatically. Despite significant computational requirements, we show that it is now possible to evolve models with accuracies within the range of those published in the last year. Specifically, we employ simple evolutionary techniques at unprecedented scales to discover models for the CIFAR-10 and CIFAR-100 datasets, starting from trivial initial conditions and reaching accuracies of 94.6% (95.6% for ensemble) and 77.0%, respectively. To do this, we use novel and intuitive mutation operators that navigate large search spaces; we stress that no human participation is required once evolution starts and that the output is a fully-trained model. Throughout this work, we place special emphasis on the repeatability of results, the variability in the outcomes and the computational requirements.

1. Introduction

Neural networks can successfully perform difficult tasks where large amounts of training data are available (He et al., 2015; Weyand et al., 2016; Silver et al., 2016; Wu et al., 2016). Discovering neural network architectures, however, remains a laborious task. Even within the specific problem of image classification, the state of the art was attained through many years of focused investigation by hundreds of researchers (Krizhevsky et al. (2012); Simonyan & Zisserman (2014); Szegedy et al. (2015); He et al. (2016); Huang et al. (2016a), among many others). It is therefore not surprising that in recent years, techniques to automatically discover these architectures have been gaining popularity (Bergstra & Bengio, 2012; Snoek et al., 2012; Han et al., 2015; Baker et al., 2016; Zoph & Le, 2016). One of the earliest such "neuro-discovery" methods was *neuro-evolution* (Miller et al., 1989; Stanley & Miikkulainen, 2002; Stanley, 2007; Bayer et al., 2009; Stanley et al., 2009; Breuel & Shafait, 2010; Pugh & Stanley, 2013; Kim & Rigazio, 2015; Zaremba, 2015; Fernando et al., 2016; Morse & Stanley, 2016). Despite the promising results, the deep learning community generally perceives evolutionary algorithms to be incapable of matching the accuracies of hand-designed models (Verbancsics & Harguess, 2013; Baker et al., 2016; Zoph & Le, 2016). In this paper, we show that it is possible to evolve such competitive models today, given enough computational power.

We used slightly-modified known evolutionary algorithms and scaled up the computation to unprecedented levels, as far as we know. This, together with a set of novel and intuitive mutation operators, allowed us to reach competitive accuracies on the CIFAR-10 dataset. This dataset was chosen because it requires large networks to reach high accuracies, thus presenting a computational challenge. We also took a small first step toward generalization and evolved networks on the CIFAR-100 dataset. In transitioning from CIFAR-10 to CIFAR-100, we did not modify any aspect or parameter of our algorithm. Our typical neuro-evolution outcome on CIFAR-10 had a test accuracy with $\mu = 94.1\%$, $\sigma = 0.4\%$ @ 9×10^{19} FLOPs, and our top model (by validation accuracy) had a test accuracy of 94.6% @ 4×10^{20} FLOPs. Ensembling the validation-top 2 models from each population reaches a test accuracy of 95.6%, at no additional training cost. On CIFAR-100, our single experiment resulted in a test accuracy of 77.0% @ 2×10^{20} FLOPs. As far as we know, these are the most accurate results obtained on these datasets by automated discovery methods that start from trivial initial conditions.

Throughout this study, we placed special emphasis on the simplicity of the algorithm. In particular, it is a "one-shot" technique, producing a fully trained neural network requiring no post-processing. It also has few impactful meta-parameters (*i.e.* parameters not optimized by the algorithm). Starting out with poor-performing models with

[1]Google Brain, Mountain View, California, USA [2]Google Research, Mountain View, California, USA. Correspondence to: Esteban Real <ereal@google.com>.

Proceedings of the 34^{th} International Conference on Machine Learning, Sydney, Australia, PMLR 70, 2017.

Table 1. Comparison with single-model hand-designed architectures. The "C10+" and "C100+" columns indicate the test accuracy on the data-augmented CIFAR-10 and CIFAR-100 datasets, respectively. The "Reachable?" column denotes whether the given hand-designed model lies within our search space. An entry of "–" indicates that no value was reported. The † indicates a result reported by Huang et al. (2016b) instead of the original author. Much of this table was based on that presented in Huang et al. (2016a).

STUDY	PARAMS.	C10+	C100+	REACHABLE?
MAXOUT (GOODFELLOW ET AL., 2013)	–	90.7%	61.4%	NO
NETWORK IN NETWORK (LIN ET AL., 2013)	–	91.2%	–	NO
ALL-CNN (SPRINGENBERG ET AL., 2014)	1.3 M	92.8%	66.3%	YES
DEEPLY SUPERVISED (LEE ET AL., 2015)	–	92.0%	65.4%	NO
HIGHWAY (SRIVASTAVA ET AL., 2015)	2.3 M	92.3%	67.6%	NO
RESNET (HE ET AL., 2016)	1.7 M	93.4%	72.8%†	YES
EVOLUTION (OURS)	5.4 M 40.4 M	94.6%	 77.0%	N/A
WIDE RESNET 28-10 (ZAGORUYKO & KOMODAKIS, 2016)	36.5 M	96.0%	80.0%	YES
WIDE RESNET 40-10+D/O (ZAGORUYKO & KOMODAKIS, 2016)	50.7 M	96.2%	81.7%	NO
DENSENET (HUANG ET AL., 2016A)	25.6 M	96.7%	82.8%	NO

no convolutions, the algorithm must evolve complex convolutional neural networks while navigating a fairly unrestricted search space: no fixed depth, arbitrary skip connections, and numerical parameters that have few restrictions on the values they can take. We also paid close attention to result reporting. Namely, we present the variability in our results in addition to the top value, we account for researcher degrees of freedom (Simmons et al., 2011), we study the dependence on the meta-parameters, and we disclose the amount of computation necessary to reach the main results. We are hopeful that our explicit discussion of computation cost could spark more study of efficient model search and training. Studying model performance normalized by computational investment allows consideration of economic concepts like opportunity cost.

2. Related Work

Neuro-evolution dates back many years (Miller et al., 1989), originally being used only to evolve the weights of a fixed architecture. Stanley & Miikkulainen (2002) showed that it was advantageous to simultaneously evolve the architecture using the *NEAT algorithm*. NEAT has three kinds of mutations: (i) modify a weight, (ii) add a connection between existing nodes, or (iii) insert a node while splitting an existing connection. It also has a mechanism for *recombining* two models into one and a strategy to promote diversity known as *fitness sharing* (Goldberg et al., 1987). Evolutionary algorithms represent the models using an encoding that is convenient for their purpose—analogous to nature's DNA. NEAT uses a *direct encoding*: every node and every connection is stored in the DNA. The alternative paradigm, *indirect encoding*, has been the subject of much neuro-evolution research (Gruau, 1993; Stanley et al., 2009; Pugh & Stanley, 2013; Kim & Rigazio, 2015; Fernando et al., 2016). For example, the *CPPN* (Stanley, 2007; Stanley et al., 2009) allows for the evolution of repeating features at different scales. Also, Kim & Rigazio (2015) use an indirect encoding to improve the convolution filters in an initially highly-optimized fixed architecture.

Research on weight evolution is still ongoing (Morse & Stanley, 2016) but the broader machine learning community defaults to back-propagation for optimizing neural network weights (Rumelhart et al., 1988). Back-propagation and evolution can be combined as in Stanley et al. (2009), where only the structure is evolved. Their algorithm follows an alternation of architectural mutations and weight back-propagation. Similarly, Breuel & Shafait (2010) use this approach for hyper-parameter search. Fernando et al. (2016) also use back-propagation, allowing the trained weights to be *inherited* through the structural modifications.

The above studies create neural networks that are small in comparison to the typical modern architectures used for image classification (He et al., 2016; Huang et al., 2016a). Their focus is on the encoding or the efficiency of the evolutionary process, but not on the scale. When it comes to images, some neuro-evolution results reach the computational scale required to succeed on the MNIST dataset (LeCun et al., 1998). Yet, modern classifiers are often tested on realistic images, such as those in the CIFAR datasets (Krizhevsky & Hinton, 2009), which are much more challenging. These datasets require large models to achieve high accuracy.

Non-evolutionary neuro-discovery methods have been more successful at tackling realistic image data. Snoek et al. (2012) used Bayesian optimization to tune 9 hyper-parameters for a fixed-depth architecture, reach-

Table 2. Comparison with automatically discovered architectures. The "C10+" and "C100+" contain the test accuracy on the data-augmented CIFAR-10 and CIFAR-100 datasets, respectively. An entry of "–" indicates that the information was not reported or is not known to us. For Zoph & Le (2016), we quote the result with the most similar search space to ours, as well as their best result. Please refer to Table 1 for hand-designed results, including the state of the art. "Discrete params." means that the parameters can be picked from a handful of values only (*e.g.* strides $\in \{1, 2, 4\}$).

STUDY	STARTING POINT	CONSTRAINTS	POST-PROCESSING	PARAMS.	C10+	C100+
BAYESIAN (SNOEK ET AL., 2012)	3 LAYERS	FIXED ARCHITECTURE, NO SKIPS	NONE	–	90.5%	–
Q-LEARNING (BAKER ET AL., 2016)	–	DISCRETE PARAMS., MAX. NUM. LAYERS, NO SKIPS	TUNE, RETRAIN	11.2 M	93.1%	72.9%
RL (ZOPH & LE, 2016)	20 LAYERS, 50% SKIPS	DISCRETE PARAMS., EXACTLY 20 LAYERS	SMALL GRID SEARCH, RETRAIN	2.5 M	94.0%	–
RL (ZOPH & LE, 2016)	39 LAYERS, 2 POOL LAYERS AT 13 AND 26, 50% SKIPS	DISCRETE PARAMS., EXACTLY 39 LAYERS, 2 POOL LAYERS AT 13 AND 26	ADD MORE FILTERS, SMALL GRID SEARCH, RETRAIN	37.0 M	96.4%	–
EVOLUTION (OURS)	SINGLE LAYER, ZERO CONVS.	POWER-OF-2 STRIDES	NONE	5.4 M 40.4 M ENSEMB.	94.6% 95.6%	77.0%

ing a new state of the art at the time. Zoph & Le (2016) used reinforcement learning on a deeper fixed-length architecture. In their approach, a neural network—the "discoverer"—constructs a convolutional neural network—the "discovered"—one layer at a time. In addition to tuning layer parameters, they add and remove skip connections. This, together with some manual post-processing, gets them very close to the (current) state of the art. (Additionally, they surpassed the state of the art on a sequence-to-sequence problem.) Baker et al. (2016) use Q-learning to also discover a network one layer at a time, but in their approach, the number of layers is decided by the discoverer. This is a desirable feature, as it would allow a system to construct shallow or deep solutions, as may be the requirements of the dataset at hand. Different datasets would not require specially tuning the algorithm. Comparisons among these methods are difficult because they explore very different search spaces and have very different initial conditions (Table 2).

Tangentially, there has also been neuro-evolution work on LSTM structure (Bayer et al., 2009; Zaremba, 2015), but this is beyond the scope of this paper. Also related to this work is that of Saxena & Verbeek (2016), who embed convolutions with different parameters into a species of "super-network" with many parallel paths. Their algorithm then selects and ensembles paths in the super-network. Finally, canonical approaches to hyper-parameter search are *grid search* (used in Zagoruyko & Komodakis (2016), for example) and *random search*, the latter being the better of the two (Bergstra & Bengio, 2012).

Our approach builds on previous work, with some important differences. We explore large model-architecture search spaces starting with basic initial conditions to avoid priming the system with information about known good strategies for the specific dataset at hand. Our encoding is different from the neuro-evolution methods mentioned above: we use a simplified graph as our DNA, which is transformed to a full neural network graph for training and evaluation (Section 3). Some of the mutations acting on this DNA are reminiscent of NEAT. However, instead of single nodes, one mutation can insert whole *layers—i.e.* tens to hundreds of nodes at a time. We also allow for these layers to be removed, so that the evolutionary process can simplify an architecture in addition to complexifying it. Layer parameters are also mutable, but we do not prescribe a small set of possible values to choose from, to allow for a larger search space. We do not use fitness sharing. We report additional results using recombination, but for the most part, we used mutation only. On the other hand, we do use back-propagation to optimize the weights, which can be inherited across mutations. Together with a learning rate mutation, this allows the exploration of the space of learning rate schedules, yielding fully trained models at the end of the evolutionary process (Section 3). Tables 1 and 2 compare our approach with hand-designed architectures and with other neuro-discovery techniques, respectively.

3. Methods

3.1. Evolutionary Algorithm

To automatically search for high-performing neural network architectures, we evolve a *population* of models. Each model—or *individual*—is a trained architecture. The model's accuracy on a separate validation dataset is a measure of the individual's quality or *fitness*. During each evolutionary step, a computer—a *worker*—chooses two individuals at random from this population and compares their fitnesses. The worst of the pair is immediately removed from the population—it is *killed*. The best of the pair is selected to be a *parent*, that is, to undergo *reproduction*. By this we mean that the worker creates a copy of the parent and modifies this copy by applying a *mutation*, as described below. We will refer to this modified copy as the *child*. After the worker creates the child, it trains this child, evaluates it on the validation set, and puts it back into the population. The child then becomes *alive—i.e.* free to act as a parent. Our scheme, therefore, uses repeated pairwise competitions of random individuals, which makes it an example of *tournament selection* (Goldberg & Deb, 1991). Using pairwise comparisons instead of whole population operations prevents workers from idling when they finish early. Code and more detail about the methods described below can be found in Supplementary Section S1.

Using this strategy to search large spaces of complex image models requires considerable computation. To achieve scale, we developed a massively-parallel, lock-free infrastructure. Many workers operate asynchronously on different computers. They do not communicate directly with each other. Instead, they use a shared file-system, where the population is stored. The file-system contains directories that represent the individuals. Operations on these individuals, such as the killing of one, are represented as atomic renames on the directory[2]. Occasionally, a worker may concurrently modify the individual another worker is operating on. In this case, the affected worker simply gives up and tries again. The *population size* is 1000 individuals, unless otherwise stated. The number of workers is always $\frac{1}{4}$ of the population size. To allow for long run-times with a limited amount of space, dead individuals' directories are frequently garbage-collected.

3.2. Encoding and Mutations

Individual architectures are encoded as a graph that we refer to as the *DNA*. In this graph, the vertices represent rank-3 tensors or *activations*. As is standard for a convolutional network, two of the dimensions of the tensor represent the spatial coordinates of the image and the third is a number of channels. Activation functions are applied at the vertices and can be either (i) batch-normalization (Ioffe & Szegedy, 2015) with rectified linear units (*ReLUs*) or (ii) plain linear units. The graph's edges represent identity connections or convolutions and contain the mutable numerical parameters defining the convolution's properties. When multiple edges are incident on a vertex, their spatial scales or numbers of channels may not coincide. However, the vertex must have a single size and number of channels for its activations. The inconsistent inputs must be resolved. Resolution is done by choosing one of the incoming edges as the primary one. We pick this primary edge to be the one that is not a skip connection. The activations coming from the non-primary edges are reshaped through zeroth-order interpolation in the case of the size and through truncation/padding in the case of the number of channels, as in He et al. (2016). In addition to the graph, the learning-rate value is also stored in the DNA.

A child is similar but not identical to the parent because of the action of a mutation. In each reproduction event, the worker picks a mutation at random from a predetermined set. The set contains the following mutations:

- ALTER-LEARNING-RATE (sampling details below).
- IDENTITY (effectively means "keep training").
- RESET-WEIGHTS (sampled as in He et al. (2015), for example).
- INSERT-CONVOLUTION (inserts a convolution at a random location in the "convolutional backbone", as in Figure 1. The inserted convolution has 3×3 filters, strides of 1 or 2 at random, number of channels same as input. May apply batch-normalization and ReLU activation or none at random).
- REMOVE-CONVOLUTION.
- ALTER-STRIDE (only powers of 2 are allowed).
- ALTER-NUMBER-OF-CHANNELS (of random conv.).
- FILTER-SIZE (horizontal or vertical at random, on random convolution, odd values only).
- INSERT-ONE-TO-ONE (inserts a one-to-one/identity connection, analogous to insert-convolution mutation).
- ADD-SKIP (identity between random layers).
- REMOVE-SKIP (removes random skip).

These specific mutations were chosen for their similarity to the actions that a human designer may take when improving an architecture. This may clear the way for hybrid evolutionary–hand-design methods in the future. The probabilities for the mutations were not tuned in any way.

A mutation that acts on a numerical parameter chooses the new value at random around the existing value. All sampling is from uniform distributions. For example, a mutation acting on a convolution with 10 output channels will

[2]The use of the file-name string to contain key information about the individual was inspired by Breuel & Shafait (2010), and it speeds up disk access enormously. In our case, the file name contains the *state* of the individual (*alive*, *dead*, *training*, *etc*.).

result in a convolution having between 5 and 20 output channels (that is, half to twice the original value). All values within the range are possible. As a result, the models are not constrained to a number of filters that is known to work well. The same is true for all other parameters, yielding a "dense" search space. In the case of the strides, this applies to the log-base-2 of the value, to allow for activation shapes to match more easily[3]. In principle, there is also no upper limit to any of the parameters. All model depths are attainable, for example. Up to hardware constraints, the search space is unbounded. The dense and unbounded nature of the parameters result in the exploration of a truly large set of possible architectures.

3.3. Initial Conditions

Every evolution *experiment* begins with a population of simple individuals, all with a learning rate of 0.1. They are all very bad performers. Each initial individual constitutes just a single-layer model with no convolutions. This conscious choice of poor initial conditions forces evolution to make the discoveries by itself. The experimenter contributes mostly through the choice of mutations that demarcate a search space. Altogether, the use of poor initial conditions and a large search space limits the experimenter's impact. In other words, it prevents the experimenter from "rigging" the experiment to succeed.

3.4. Training and Validation

Training and validation is done on the CIFAR-10 dataset. This dataset consists of 50,000 training examples and 10,000 test examples, all of which are 32 x 32 color images labeled with 1 of 10 common object classes (Krizhevsky & Hinton, 2009). 5,000 of the training examples are held out in a validation set. The remaining 45,000 examples constitute our actual training set. The training set is augmented as in He et al. (2016). The CIFAR-100 dataset has the same number of dimensions, colors and examples as CIFAR-10, but uses 100 classes, making it much more challenging.

Training is done with *TensorFlow* (Abadi et al., 2016), using SGD with a momentum of 0.9 (Sutskever et al., 2013), a batch size of 50, and a weight decay of 0.0001. Each training runs for 25,600 steps, a value chosen to be brief enough so that each individual could be trained in a few seconds to a few hours, depending on model size. The loss function is the cross-entropy. Once training is complete, a single evaluation on the validation set provides the accuracy to use as the individual's fitness. Ensembling was done by majority voting during the testing evaluation. The models used in the ensemble were selected by validation accuracy.

[3]For integer DNA parameters, we actually store and mutate a floating-point value. This allows multiple small mutations to have a cumulative effect in spite of integer round-off.

3.5. Computation cost

To estimate computation costs, we identified the basic TensorFlow (*TF*) operations used by our model training and validation, like convolutions, generic matrix multiplications, *etc*. For each of these TF operations, we estimated the theoretical number of floating-point operations (*FLOPs*) required. This resulted in a map from TF operation to FLOPs, which is valid for all our experiments.

For each individual within an evolution experiment, we compute the total FLOPs incurred by the TF operations in its architecture over one batch of examples, both during its training (F_t FLOPs) and during its validation (F_v FLOPs). Then we assign to the individual the cost $F_t N_t + F_v N_v$, where N_t and N_v are the number of training and validation batches, respectively. The cost of the experiment is then the sum of the costs of all its individuals.

We intend our FLOPs measurement as a coarse estimate only. We do not take into account input/output, data preprocessing, TF graph building or memory-copying operations. Some of these unaccounted operations take place once per training run or once per step and some have a component that is constant in the model size (such as disk-access latency or input data cropping). We therefore expect the estimate to be more useful for large architectures (for example, those with many convolutions).

3.6. Weight Inheritance

We need architectures that are trained to completion within an evolution experiment. If this does not happen, we are forced to retrain the best model at the end, possibly having to explore its hyper-parameters. Such extra exploration tends to depend on the details of the model being retrained. On the other hand, 25,600 steps are not enough to fully train each individual. Training a large model to completion is prohibitively slow for evolution. To resolve this dilemma, we allow the children to inherit the parents' weights whenever possible. Namely, if a layer has matching shapes, the weights are preserved. Consequently, some mutations preserve all the weights (like the identity or learning-rate mutations), some preserve none (the weight-resetting mutation), and most preserve some but not all. An example of the latter is the filter-size mutation: only the filters of the convolution being mutated will be discarded.

3.7. Reporting Methodology

To avoid over-fitting, neither the evolutionary algorithm nor the neural network training ever see the testing set. Each time we refer to "the best model", we mean the model with the highest validation accuracy. However, we always report the test accuracy. This applies not only to the choice of the best individual within an experiment, but also *to the choice*

of the best experiment. Moreover, we only include experiments that we managed to reproduce, unless explicitly noted. Any statistical analysis was fully decided upon before seeing the results of the experiment reported, to avoid tailoring our analysis to our experimental data (Simmons et al., 2011).

4. Experiments and Results

We want to answer the following questions:

- Can a simple one-shot evolutionary process start from trivial initial conditions and yield fully trained models that rival hand-designed architectures?
- What are the variability in outcomes, the parallelizability, and the computation cost of the method?
- Can an algorithm designed iterating on CIFAR-10 be applied, without any changes at all, to CIFAR-100 and still produce competitive models?

We used the algorithm in Section 3 to perform several experiments. Each experiment evolves a population in a few days, typified by the example in Figure 1. The figure also contains examples of the architectures discovered, which turn out to be surprisingly simple. Evolution attempts skip connections but frequently rejects them.

To get a sense of the variability in outcomes, we repeated the experiment 5 times. Across all 5 experiment runs, the best model by validation accuracy has a testing accuracy of $94.6\,\%$. Not all experiments reach the same accuracy, but they get close ($\mu = 94.1\%$, $\sigma = 0.4$). Fine differences in the experiment outcome may be somewhat distinguishable by validation accuracy (correlation coefficient $= 0.894$). The total amount of computation *across all 5 experiments* was 4×10^{20} FLOPs (or 9×10^{19} FLOPs on average per experiment). Each experiment was distributed over 250 parallel workers (Section 3.1). Figure 2 shows the progress of the experiments in detail.

As a control, we disabled the selection mechanism, thereby reproducing and killing random individuals. This is the form of random search that is most compatible with our infrastructure. The probability distributions for the parameters are implicitly determined by the mutations. This control only achieves an accuracy of $87.3\,\%$ in the same amount of run time on the same hardware (Figure 2). The total amount of computation was 2×10^{17} FLOPs. The low FLOP count is a consequence of random search generating many small, inadequate models that train quickly but consume roughly constant amounts of setup time (not included in the FLOP count). We attempted to minimize this overhead by avoiding unnecessary disk access operations, to no avail: too much overhead remains spent on a combination of neural network setup, data augmentation, and training step initialization.

We also ran a partial control where the weight-inheritance mechanism is disabled. This run also results in a lower accuracy ($92.2\,\%$) in the same amount of time (Figure 2), using 9×10^{19} FLOPs. This shows that weight inheritance is important in the process.

Finally, we applied our neuro-evolution algorithm, without any changes and with the same meta-parameters, to CIFAR-100. Our only experiment reached an accuracy of $77.0\,\%$, using 2×10^{20} FLOPs. We did not attempt other datasets. Table 1 shows that both the CIFAR-10 and CIFAR-100 results are competitive with modern hand-designed networks.

5. Analysis

Meta-parameters. We observe that populations evolve until they plateau at some local optimum (Figure 2). The fitness (*i.e*. validation accuracy) value at this optimum varies between experiments (Figure 2, inset). Since not all experiments reach the highest possible value, some populations are getting "trapped" at inferior local optima. This entrapment is affected by two important meta-parameters (*i.e*. parameters that are not optimized by the algorithm). These are the population size and the number of training steps per individual. Below we discuss them and consider their relationship to local optima.

Effect of population size. Larger populations explore the space of models more thoroughly, and this helps reach better optima (Figure 3, left). Note, in particular, that a population of size 2 can get trapped at very low fitness values. Some intuition about this can be gained by considering the fate of a *super-fit* individual, *i.e*. an individual such that any one architectural mutation reduces its fitness (even though a sequence of many mutations may improve it). In the case of a population of size 2, if the super-fit individual wins once, it will win every time. After the first win, it will produce a child that is one mutation away. By definition of super-fit, therefore, this child is inferior[4]. Consequently, in the next round of tournament selection, the super-fit individual competes against its child and wins again. This cycle repeats forever and the population is trapped. Even if a sequence of two mutations would allow for an "escape" from the local optimum, such a sequence can never take place. This is only a rough argument to heuristically suggest why a population of size 2 is easily trapped. More generally, Figure 3 (left) empirically demonstrates a benefit from an increase in population size. Theoretical analyses of this dependence are quite complex and assume very specific models of population dynamics; often larger populations are better at handling local optima, at least beyond a size threshold (Weinreich & Chao (2005) and references

[4]Except after identity or learning rate mutations, but these produce a child with the same architecture as the parent.

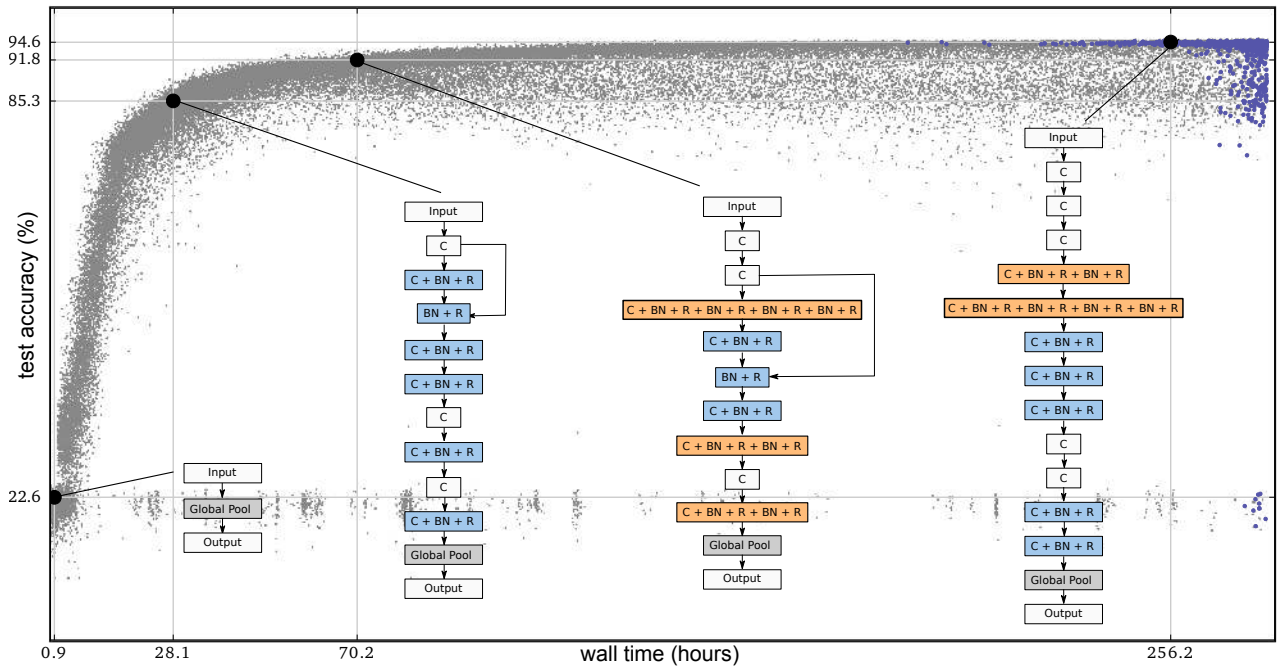

Figure 1. Progress of an evolution experiment. Each dot represents an individual in the population. Blue dots (darker, top-right) are alive. The rest have been killed. The four diagrams show examples of discovered architectures. These correspond to the best individual (rightmost) and three of its ancestors. The best individual was selected by its validation accuracy. Evolution sometimes stacks convolutions without any nonlinearity in between ("C", white background), which are mathematically equivalent to a single linear operation. Unlike typical hand-designed architectures, some convolutions are followed by more than one nonlinear function ("$C+BN+R+BN+R+...$", orange background).

therein).

Effect of number of training steps. The other meta-parameter is the number T of training steps for each individual. Accuracy increases with T (Figure 3, right). Larger T means an individual needs to undergo fewer identity mutations to reach a given level of training.

Escaping local optima. While we might increase population size or number of steps to prevent a trapped population from forming, we can also free an already trapped population. For example, increasing the *mutation rate* or resetting all the weights of a population (Figure 4) work well but are quite costly (more details in Supplementary Section S3).

Recombination. None of the results presented so far used recombination. However, we explored three forms of recombination in additional experiments. Following Tuson & Ross (1998), we attempted to evolve the mutation probability distribution too. On top of this, we employed a recombination strategy by which a child could inherit structure from one parent and mutation probabilities from another. The goal was to allow individuals that progressed well due to good mutation choices to quickly propagate such choices to others. In a separate experiment, we attempted recombining the trained weights from two parents in the hope that each parent may have learned different concepts from the training data. In a third experiment, we recombined structures so that the child fused the architectures of both parents side-by-side, generating wide models fast. While none of these approaches improved our recombination-free results, further study seems warranted.

6. Conclusion

In this paper we have shown that (i) neuro-evolution is capable of constructing large, accurate networks for two challenging and popular image classification benchmarks; (ii) neuro-evolution can do this starting from trivial initial conditions while searching a very large space; (iii) the process, once started, needs no experimenter participation; and (iv) the process yields fully trained models. Completely training models required weight inheritance (Sections 3.6). In contrast to reinforcement learning, evolution provides a natural framework for weight inheritance: mutations can be constructed to guarantee a large degree of similarity be-

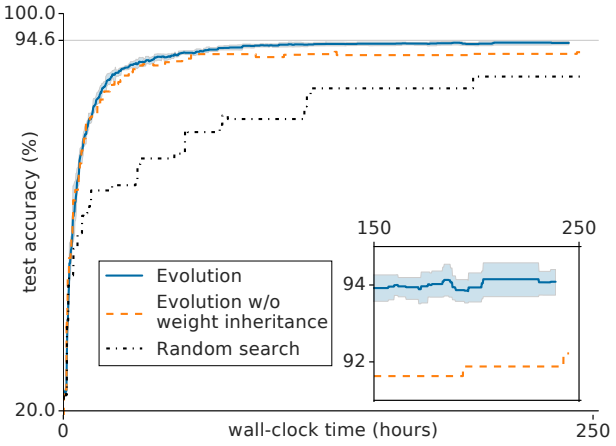

Figure 2. Repeatability of results and controls. In this plot, the vertical axis at wall-time t is defined as the test accuracy of the individual with the highest validation accuracy that became alive at or before t. The inset magnifies a portion of the main graph. The curves show the progress of various experiments, as follows. The top line (solid, blue) shows the mean test accuracy across 5 large-scale evolution experiments. The shaded area around this top line has a width of $\pm 2\sigma$ (clearer in inset). The next line down (dashed, orange, main graph and inset) represents a single experiment in which weight-inheritance was disabled, so every individual has to train from random weights. The lowest curve (dotted-dashed) is a random-search control. All experiments occupied the same amount and type of hardware. A small amount of noise in the generalization from the validation to the test set explains why the lines are not monotonically increasing. Note the narrow width of the $\pm 2\sigma$ area (main graph and inset), which shows that the high accuracies obtained in evolution experiments are repeatable.

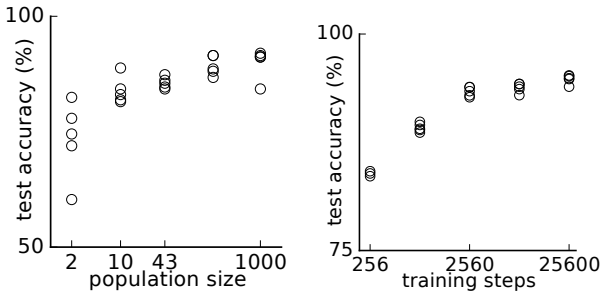

Figure 3. Dependence on meta-parameters. In both graphs, each circle represents the result of a full evolution experiment. Both vertical axes show the test accuracy for the individual with the highest validation accuracy at the end of the experiment. All populations evolved for the same total wall-clock time. There are 5 data points at each horizontal axis value. LEFT: effect of population size. To economize resources, in these experiments the number of individual training steps is only 2560. Note how the accuracy increases with population size. RIGHT: effect of number of training steps per individual. Note how the accuracy increases with more steps.

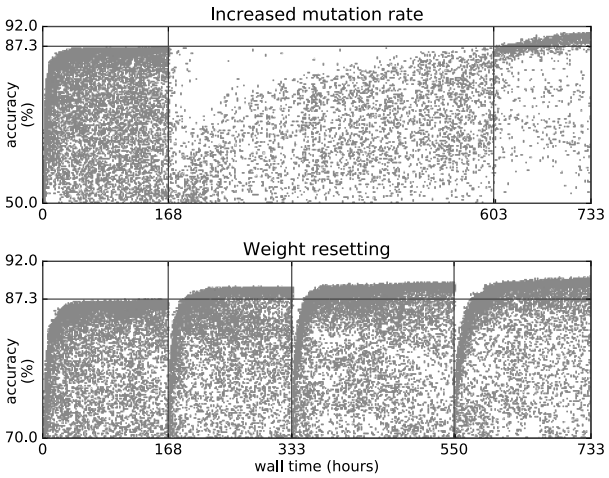

Figure 4. Escaping local optima in two experiments. We used smaller populations and fewer training steps per individual (2560) to make it more likely for a population to get trapped and to reduce resource usage. Each dot represents an individual. The vertical axis is the accuracy. TOP: example of a population of size 100 escaping a local optimum by using a period of increased mutation rate in the middle (Section 5). BOTTOM: example of a population of size 50 escaping a local optimum by means of three consecutive weight resetting events (Section 5). Details in Supplementary Section S3.

tween the original and mutated models—as we did. Evolution also has fewer tunable meta-parameters with a fairly predictable effect on the variance of the results, which can be made small.

While we did not focus on reducing computation costs, we hope that future algorithmic and hardware improvement will allow more economical implementation. In that case, evolution would become an appealing approach to neuro-discovery for reasons beyond the scope of this paper. For example, it "hits the ground running", improving on arbitrary initial models as soon as the experiment begins. The mutations used can implement recent advances in the field and can be introduced without having to restart an experiment. Furthermore, recombination can merge improvements developed by different individuals, even if they come from other populations. Moreover, it may be possible to combine neuro-evolution with other automatic architecture discovery methods.

Acknowledgements

We wish to thank Vincent Vanhoucke, Megan Kacholia, Rajat Monga, and especially Jeff Dean for their support and valuable input; Geoffrey Hinton, Samy Bengio, Thomas Breuel, Mark DePristo, Vishy Tirumalashetty, Martin Abadi, Noam Shazeer, Yoram Singer, Dumitru Erhan, Pierre Sermanet, Xiaoqiang Zheng, Shan Carter and Vijay Vasudevan for helpful discussions; Thomas Breuel, Xin Pan and Andy Davis for coding contributions; and the larger Google Brain team for help with *TensorFlow* and training vision models.

References

Abadi, Martín, Agarwal, Ashish, Barham, Paul, Brevdo, Eugene, Chen, Zhifeng, Citro, Craig, Corrado, Greg S, Davis, Andy, Dean, Jeffrey, Devin, Matthieu, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

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

Bayer, Justin, Wierstra, Daan, Togelius, Julian, and Schmidhuber, Jürgen. Evolving memory cell structures for sequence learning. In *International Conference on Artificial Neural Networks*, pp. 755–764. Springer, 2009.

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

Breuel, Thomas and Shafait, Faisal. Automlp: Simple, effective, fully automated learning rate and size adjustment. In *The Learning Workshop*. Utah, 2010.

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 *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference*, pp. 109–116. ACM, 2016.

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

Goldberg, David E, Richardson, Jon, et al. Genetic algorithms with sharing for multimodal function optimization. In *Genetic algorithms and their applications: Proceedings of the Second International Conference on Genetic Algorithms*, pp. 41–49. Hillsdale, NJ: Lawrence Erlbaum, 1987.

Goodfellow, Ian J, Warde-Farley, David, Mirza, Mehdi, Courville, Aaron C, and Bengio, Yoshua. Maxout networks. *International Conference on Machine Learning*, 28:1319–1327, 2013.

Gruau, Frederic. Genetic synthesis of modular neural networks. In *Proceedings of the 5th International Conference on Genetic Algorithms*, pp. 318–325. Morgan Kaufmann Publishers Inc., 1993.

Han, Song, Pool, Jeff, Tran, John, and Dally, William. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems*, pp. 1135–1143, 2015.

He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778, 2016.

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

Huang, Gao, Sun, Yu, Liu, Zhuang, Sedra, Daniel, and Weinberger, Kilian Q. Deep networks with stochastic depth. In *European Conference on Computer Vision*, pp. 646–661. Springer, 2016b.

Ioffe, Sergey and Szegedy, Christian. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

Kim, Minyoung and Rigazio, Luca. Deep clustered convolutional kernels. *arXiv preprint arXiv:1503.01824*, 2015.

Krizhevsky, Alex and Hinton, Geoffrey. Learning multiple layers of features from tiny images. 2009.

Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey E. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pp. 1097–1105, 2012.

LeCun, Yann, Cortes, Corinna, and Burges, Christopher JC. The mnist database of handwritten digits, 1998.

Lee, Chen-Yu, Xie, Saining, Gallagher, Patrick W, Zhang, Zhengyou, and Tu, Zhuowen. Deeply-supervised nets. In *AISTATS*, volume 2, pp. 5, 2015.

Lin, Min, Chen, Qiang, and Yan, Shuicheng. Network in network. *arXiv preprint arXiv:1312.4400*, 2013.

Miller, Geoffrey F, Todd, Peter M, and Hegde, Shailesh U. Designing neural networks using genetic algorithms. In *Proceedings of the third international conference on Genetic algorithms*, pp. 379–384. Morgan Kaufmann Publishers Inc., 1989.

Morse, Gregory and Stanley, Kenneth O. Simple evolutionary optimization can rival stochastic gradient descent in neural networks. In *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference*, pp. 477–484. ACM, 2016.

Pugh, Justin K and Stanley, Kenneth O. Evolving multimodal controllers with hyperneat. In *Proceedings of the 15th annual conference on Genetic and evolutionary computation*, pp. 735–742. ACM, 2013.

Rumelhart, David E, Hinton, Geoffrey E, and Williams, Ronald J. Learning representations by back-propagating errors. *Cognitive Modeling*, 5(3):1, 1988.

Saxena, Shreyas and Verbeek, Jakob. Convolutional neural fabrics. In *Advances In Neural Information Processing Systems*, pp. 4053–4061, 2016.

Silver, David, Huang, Aja, Maddison, Chris J, Guez, Arthur, Sifre, Laurent, Van Den Driessche, George, Schrittwieser, Julian, Antonoglou, Ioannis, Panneershelvam, Veda, Lanctot, Marc, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

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. *arXiv preprint arXiv:1409.1556*, 2014.

Snoek, Jasper, Larochelle, Hugo, and Adams, Ryan P. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pp. 2951–2959, 2012.

Springenberg, Jost Tobias, Dosovitskiy, Alexey, Brox, Thomas, and Riedmiller, Martin. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014.

Srivastava, Rupesh Kumar, Greff, Klaus, and Schmidhuber, Jürgen. Highway networks. *arXiv preprint arXiv:1505.00387*, 2015.

Stanley, Kenneth O. Compositional pattern producing networks: A novel abstraction of development. *Genetic programming and evolvable machines*, 8(2):131–162, 2007.

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.

Sutskever, Ilya, Martens, James, Dahl, George E, and Hinton, Geoffrey E. On the importance of initialization and momentum in deep learning. *ICML (3)*, 28:1139–1147, 2013.

Szegedy, Christian, Liu, Wei, Jia, Yangqing, Sermanet, Pierre, Reed, Scott, Anguelov, Dragomir, Erhan, Dumitru, Vanhoucke, Vincent, and Rabinovich, Andrew. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–9, 2015.

Tuson, Andrew and Ross, Peter. Adapting operator settings in genetic algorithms. *Evolutionary computation*, 6(2): 161–184, 1998.

Verbancsics, Phillip and Harguess, Josh. Generative neuroevolution for deep learning. *arXiv preprint arXiv:1312.5355*, 2013.

Weinreich, Daniel M and Chao, Lin. Rapid evolutionary escape by large populations from local fitness peaks is likely in nature. *Evolution*, 59(6):1175–1182, 2005.

Weyand, Tobias, Kostrikov, Ilya, and Philbin, James. Planet-photo geolocation with convolutional neural networks. In *European Conference on Computer Vision*, pp. 37–55. Springer, 2016.

Wu, Yonghui, Schuster, Mike, Chen, Zhifeng, Le, Quoc V., Norouzi, Mohammad, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

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

Zaremba, Wojciech. An empirical exploration of recurrent network architectures. 2015.

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