Trainless Model Performance Estimation for Neural Architecture Search

Ekaterina Gracheva\textsuperscript{a,b}

\textsuperscript{a}International Center for Materials Nanoarchitectonics, National Institute for Materials Science, 1-2-1 Sengen, Tsukuba, Ibaraki, 305-0047 Japan
\textsuperscript{b}University of Tsukuba, 1-1-1 Tennodai, Tsukuba, Ibaraki, 305-8577 Japan

Abstract

Neural architecture search has become an indispensable part of the deep learning field. Modern methods allow to find out the best performing architectures for a task, or to build a network from scratch, but they usually require a tremendous amount of training. In this paper we present a simple method, allowing to discover a suitable architecture for a task based on its untrained performance. We introduce the metric score as the relative standard deviation of the untrained accuracy, which is the standard deviation divided by the mean. Statistics for each neural architecture are calculated over multiple initialisations with different seeds on a single batch of data. An architecture with the lowest metric score value has on average an accuracy of 91.90 ± 2.27, 64.08 ± 5.63 and 38.76 ± 6.62 for CIFAR-10, CIFAR-100 and a downscaled version of ImageNet, respectively. The results show that a good architecture should be stable against initialisations before training. The procedure takes about 190 s for CIFAR and 133.9 s for ImageNet, on a batch of 256 images and 100 initialisations.

Keywords: Neural architecture search, Trainless architecture search, Deep learning, Machine learning

1. Introduction

Since the beginning of the boom in the field of artificial intelligence, there has been continuous increase in data complexity and quantity, neural architecture designs, as well as yet increasing choice of powerful hardware. All these factors render neural architecture building process complex. Given an extremely large number of parameters to be tuned, it can also be extremely slow, and decisions are often based on trial and error method. Neural architecture search (NAS) is therefore a way to automatise and accelerate the decision taking process, shifting this task from humans to machines. Not surprisingly, NAS has now become one of the most popular topics among the deep learning society.

Email address: gracheva.ekaterina@nims.go.jp (Ekaterina Gracheva)
The first attempts to find the most suitable network structure were done through evolutionary algorithms \cite{1, 2, 3, 4}. There, several architectures are mutated in various ways (e.g. adding or removing a layer, changing activation function, etc.), and the resulting offsprings are evaluated through training. The best performing ones are added to the population for the next step, and so on. This method has been used since back in the 1990’s \cite{5} and shows one of the best performances until now \cite{6}.

Similarly, Bayesian optimisation \cite{7} is used to predict the best performing architecture out of many by training a subset of architectures \cite{8}. This method has shown a few state-of-the-art performances in the period between 2013 and 2016 \cite{9, 10, 11}.

In 2016 Zoch et al. \cite{12} proposed to use the reinforcement learning to build neural architectures from scratch. There, a so-called controller neural network is trained to build a child-network — the network to be used for the final training and prediction. This method, though fruitful, demands tremendous amount of child-model training and is extremely lengthy. Several related works show significant acceleration of the process by reducing the search space \cite{13} or introducing weight sharing \cite{14}. An extensive overview of the NAS methods has been recently done by Thomas Elsken et al. \cite{15}.

The common point of all the above mentioned NAS algorithms is that at some point they all require model training. Not only that means longer search time, but also a higher uncertainty, since model training brings extra parameters to be tuned, such as batch size \((N_{BS})\) or learning rate. Therefore, finding a way to estimate model’s trained performance without training is very desirable.

As a step towards trainless NAS, in 2018 Istrate et al. \cite{16} have introduced a small LSTM-based model, that allows to predict architecture’s performance without training it on the data of interest. This model predicts an architecture’s potential for a given data complexity. This data is taken from a so-called lifelong database of experiments. The straightaway restriction of this method is that there should already exist some data of a similar complexity within the database, and the available networks are limited to already existing ones (focused on image classification). Moreover, with time the overall procedure might lead to a bias, i.e., a most often predicted architecture in the beginning will have yet more chance to be output in future, thus ”locking” it at the top position.

Another trainless NAS approach has been proposed by Mellor et al. \cite{17} in 2020. The authors assess the neural architecture’s potential by passing a single minibatch of the data through a network forwards and backwards — one single time. Based on the results of the backpropagation, they measure the correlation between calculated gradients associated with the input layer. First, the correlation matrix Σ is built, then the final score is computed as

\[
S = -\sum_{i=1}^{n} \left[ \log(\sigma_{J,i} + k) + (\sigma_{J,i} + k)^{-1} \right],
\]

where \(\sigma_{J,i}\) are the eigenvalues of \(\Sigma\), and \(k\) is a small constant added for numerical stability \((k = 1e^{-5})\). It’s worth noting that the choice of the final score’s shape
is not clearly explained. The tests are performed using the NAS-Bench-201 benchmark database [13], containing extensive information on 15625 generated neural architectures, trained on the three datasets: CIFAR-10, CIFAR-100 [19] and a lower resolution version of ImageNet (named ImageNet16-120 with 16x16 pixels per image [20]).

Mellor et al. [17] have shown that their score allows to find one of the best neural architectures among many with consistent success. The average test accuracies of selected networks for CIFAR-10, CIFAR-100 and ImageNet16-120 are 91.61 ± 1.71%, 66.56 ± 3.28% and 36.37 ± 6.97%, respectively, when choosing among 100 randomly picked architectures, with statistics reported over 500 runs. This compares nicely to the best performing NAS algorithm, REA [6], which achieves, respectively, 93.92 ± 0.30%, 71.84 ± 0.99% and 45.54 ± 1.03% accuracies for the same datasets based on 3 runs. There is also a random baseline given, that is when the architecture is chosen at random, which is of 86.61 ± 13.46%, 60.83 ± 12.58% and 33.13 ± 9.66% for CIFAR-10, CIFAR-100 and ImageNet16-120, respectively.

From another hand, there are a few papers, indicating that the best trained neural architecture also shows a better untrained accuracy. For example, the work of the UBER team [21] mentions that the best final architecture shows nearly 40% accuracy on MNIST dataset at initialisation. David Ha and Adam Gaier [22] have presented a NAS algorithm which builds an architecture based on the untrained score. Their score is taking into account both the number of parameters contained within a model, which they seek to minimise, and the mean accuracy, which is being maximised. The mean accuracy is computed over several initialisations of the child model using a set of constant weights (single value for all the weights). They report that the resulting model achieves 82.0% ± 18.7% on MNIST data with random weights at initialisation, and over 90% when the weights are fixed to the best performing constant ones.

Inspired by these findings, in the present paper we investigate how an architecture’s performance upon training depends on its performance before training, based on statistics over several initialisations. The aim of this study is to develop a scoring metric for entirely trainless NAS.

Our work can be divided into two parts. First, we have conducted an extensive MNIST study to find the optimal scoring metric. For this, we have trained a range of neural architectures with multiple seeds, to see which combination of untrained metrics results in the best correlations with trained performance. link 1

The second part is to examine the generality of the obtained metric. Since Mellor et al. [17] provide a working code to reproduce the results presented in their paper, we use this opportunity to see if the metric, found in the first part of our study, can be successfully applied to more complex neural structures and

---

1Fully reproducible code with data is available on GitHub at https://github.com/egracheva/TrainlessNAS_MNIST
The paper is structured as follows: Section 2 details the search spaces, datasets and training schemes used for the scoring metric search (2.1) and application (2.2). Section 3.1 presents the selected scoring metric, while in Section 3.2 we provide the results of the experiments with CIFAR and ImageNet. We discuss the results in Section 4. Conclusions and future improvements are proposed in Section 5.

2. Materials and methods

2.1. MNIST dataset processing and training

2.1.1. Search space

First of all, we explore correlations between some of the untrained performance metrics and the resulting trained accuracy (evaluated on the test set). For this purpose we use the MNIST [23] dataset, which is a renowned dataset containing images of handwritten digits from 0 to 9. In order to reduce the uncertainty brought by complex neural structures (initialisation, activation, etc.), the search region is limited to fully connected neural networks consisting of 2 hidden layers. The number of units in each hidden layer is set to be one of the 12 values \([8, 16, 24, 32, 56, 64, 96, 128, 256, 512, 1024, 2048]\), making a total of 144 architectures.

2.1.2. Dataset

Since MNIST is a fairly easy-to-fit dataset, the prediction accuracy is quickly saturated with growing network complexity, making it harder to distinguish between trained performances of different networks. To see the difference clearer, we reduced the training set from the original 55000 data points to as little as 200 (20 data points per class), thus making the prediction task harder. This also allowed to radically decrease the training time. Note that both the validation and test sets are entirely preserved, containing 5000 data points each. No data augmentation is applied.

2.1.3. Training scheme

Every neural network is initialised and trained with 100 different seeds between 0 and 99, and 6 learning rates \([0.0001, 0.0003, 0.001, 0.003, 0.01, 0.03]\) (600 trainings per architecture, 86400 trainings overall). The batch size is fixed to 50, which we found showing the best results for a wide range of architectures within the search space. The models are built with Keras [24] and Tensorflow [25] and trained for 200 epochs using 3 NVIDIA Titan V GPUs. Weights are initialised using the He uniform initialiser [26], which is used together with ReLU activation function [27] for hidden layers and Adam optimiser with default decay rates (0.9 and 0.99 for the first and second moments, respectively).

\footnote{The modified code can be found on GitHub at \url{https://github.com/egracheva/TrainlessNAS_NAS201Bench}}
The final weights are based on the epoch with the best validation accuracy after a burn-in period of 50 epochs. Ignoring the first quarter of the training process is based on experience, since the validation loss of small noisy data tends to have random spikes in the beginning of the training, leading to faulty results.

The pseudocode for the MNIST training process is given in Algorithm 1.

Algorithm 1 MNIST training pseudocode

```
Load the data
Split data on train/val/test sets
for cat in categories do
  ▷ creating reduced train set
    Randomly pick 20 points from the original train set
end for
for numits_layer,1 in [8, 16, ..., 2048] do
  for numits_layer,2 in [8, 16, ..., 2048] do
    for lr in [0.0001, 0.0003, 0.001, 0.003, 0.01, 0.03] do
      for seed in range(N_init) do
        ▷ untrained accuracy
          Build initial model
          Assess the untrained accuracy $U_i$
        ▷ trained accuracy
          Train the model
          Select the final weights based on the best validation accuracy
          Compute test set accuracy $T_i$
      end for
    end for
  end for
end for
Once the training is complete, only the best performing learning rate for each architecture is selected for the comparison. This is done to ensure that neural architectures are compared in a fair way, each showing its best performance. Afterwards, mean untrained error $\mu_U$, mean trained error $\mu_T$, together with their respective standard deviations ($\sigma_U$, $\sigma_T$) are calculated.
```

\[
\mu_T = \frac{1}{N} \sum_{i=1}^{N_{init}} T_i, \quad \sigma_T = \sqrt{\frac{1}{N_{init}} \sum_{i=1}^{N_{init}} (T_i - \mu_T)^2} \\
\mu_U = \frac{1}{N} \sum_{i=1}^{N_{init}} U_i, \quad \sigma_U = \sqrt{\frac{1}{N_{init}} \sum_{i=1}^{N_{init}} (U_i - \mu_U)^2}
\]
Table 1: A summary over the datasets used in this paper: number of classes, image resolution and splitting schemes (in thousands) for reduced MNIST, CIFAR-10, CIFAR-100 and ImageNet-16-120.

| Dataset          | Classes | Resolution | Train/val/test (K) |
|------------------|---------|------------|--------------------|
| Reduced MNIST    | 10      | 28x28      | 0.2/5/5            |
| CIFAR-10         | 10      | 32x32x3    | 25/25/10           |
| CIFAR-100        | 100     | 32x32x3    | 50/5/5             |
| ImageNet16-120   | 120     | 16x16x2    | 151.7/3/3          |

2.2. CIFAR-10, CIFAR-100 and ImageNet

2.2.1. Search space: NAS-Bench-201

To test more complex geometries on challenging datasets, we used a modified version of the code published by Mellor et al. \cite{17} together with their paper\cite{3} To check validity of their NAS search metric, authors use the NAS-Bench-201 search space. It is a set of architectures with a fixed skeleton, consisting of convolution layer and three stacks of cells, connected by a residual block. Each cell is a densely-connected directed acyclic graph with 4 nodes, 5 possible operations and no limits on the number of edges, providing a total of 15,625 architectures.

2.2.2. Datasets

Each of these architectures is trained on three major datasets: CIFAR-10, CIFAR-100 and ImageNet. Since the original CIFAR datasets do not contain a validation set, the NAS-Bench-201 authors created it by splitting the original data. In case of CIFAR-10, the training set is split into halves to form the validation set, leaving the test set unchanged; for CIFAR-100, the test set is split in halves to form the validation set and the new test set. For the sake of computational tractability, a simplified version of ImageNet is used. All images are down-scaled to 16x16 pixels, with 120 classes kept, forming a new ImageNet16-120 dataset. Data augmentation is used for all datasets; augmentation schemes differ slightly between CIFAR and ImageNet due to the difference between input images’ sizes.

An overview on all the data used in present work is given in Table 1.

2.2.3. Training

The training is done using up to 3 different seeds, and with the same fixed set of hyperparameters for each dataset. The authors use stochastic gradient descent with Nesterov momentum, $N_{BS} = 256$, learning rate between 0.1 and 0 with cosine annealing and weight decay of 0.0005. Architectures are trained for 200 epochs.

\footnote{The code can be found on GitHub at \url{https://github.com/BayesWatch/nas-without-training}}
2.2.4. Experimental scheme

Here we mainly use the implementation from Mellor et al. [17] with only modifications related to scoring metric. The idea is to find out how well does a particular scoring metric select one architecture among many. In order to obtain statistically significant information, selection process is run $N_{\text{runs}} = 500$ times, each time choosing $N_a$ architectures at random (among 15,625 available). Each architecture is initialised $N_{\text{init}}$ times, in order to access the mean $\mu_U$ and standard deviation $\sigma_U$ of the untrained performance. The batch data used for the accuracy computation is fixed for every individual run, so that all the architectures are compared in the most fair way, and the uncertainty of data choice has minimum influence.

The pseudocode for this part of study can be written as follows:

**Algorithm 2** $\sigma_r$ tests on NAS-Bench-201

```
for run in range($N_{\text{runs}}$) do
    Randomly select $N_{BS}$ images from the training dataset
    Randomly select $N_a$ architectures from the whole space > arches
    for arch in arches do
        for seed in range($N_{\text{init}}$) do
            Initialise the arch with the seed
            Forward propagate selected $N_{BS}$ images
            Compute untrained accuracy $U_i$
        end for
        Compute mean $\mu_U$, standard deviation $\sigma_U$ for untrained accuracies over initialisations
        $\mu_U = \frac{1}{N} \sum_{i=1}^{N_{\text{init}}} U_i$, $\sigma_U = \sqrt{\frac{\sum_{i=1}^{N_{\text{init}}} (U_i - \mu_U)^2}{N_{\text{init}}}}$
    end for
    Compute the score
    $\sigma_r = \frac{\sigma_U}{\mu_U}$ (1)
end for
Select the architecture with the minimum score value ($\sigma_r > 0$)
Retrieve trained accuracy $T$ for the selected architecture from the database
end for
Average trained accuracies of selected architectures over $N_{\text{runs}}$
$\mu_T = \sum_{j=1}^{N_{\text{runs}}} T_j$ (2)
```

Filtering out the scores equal to zero is necessary for the random architectures containing no meaningful layers (for example, architectures consisting
of skip-connection layers only). These architectures, naturally, show random accuracy with no deviation ($\sigma_U = 0$).

3. Results

3.1. Scoring metric search with MNIST

The aim of the MNIST-related experiments is to find the best combination of untrained accuracy statistics over several architecture initialisations, in order to predict the best performing architecture. The existing machine learning literature gives hints that the best trained architecture also shows high untrained performance. We expect, thus, to see some tendency between mean untrained $\mu_U$ and mean trained $\mu_T$ performances. It turned out, however, that there is no clear correlation, as shown on Figure 1a. Instead, surprisingly, the mean trained accuracy $\mu_T$ seems to be related to the untrained standard deviation $\sigma_U$: even though there is no linear correlation, the lowest $\sigma_U$ values belong to architectures from the top performance range (Figure 1b).

![Figure 1: (a) Mean untrained accuracy $\mu_U$ (a) and standard deviation of untrained accuracy $\sigma_U$ (b) plot against mean trained accuracy $\mu_T$, all three computed over $N_{init} = 100$ initialisations. One point stands for one architecture. The colours represent the logarithm of the number of parameters for a given architecture.](image)

We have also recognised that a higher mean $\mu_U$ corresponds to a higher standard deviation $\sigma_U$ (Figure 2a). It can be due to the fact that if one dataset contains proportionally larger values than the other, it will also have proportionally larger mean and standard deviation. So comparing standard deviations alone may bias towards the networks that show overall low untrained accuracies $U_i$. To compensate for this effect, we normalise the standard deviation $\sigma_U$ by the mean $\mu_U$:

$$\sigma_r = \frac{\sigma_U}{\mu_U}$$

The resulting parameter $\sigma_r$ is known in statistics as the relative standard deviation, or the coefficient of variation.
After the normalisation, there is no correlation with the mean untrained accuracy $\mu_U$ (Figure 2a). When plotting the relative standard deviation $\sigma_r$ against the trained accuracy, visual pattern becomes yet more clear: selecting the architectures with low $\sigma_r$ leads to high trained accuracy $\mu_T$ (Figure 3). We conclude, therefore, that $\sigma_r$ might serve as a scoring metric for NAS.

Another point that we would like to mention, is the architecture performance as a function of the number of parameters. Figure 4 shows mean trained performance $\mu_T$ as a function of the number of parameters with the fixed number of units in the first (a) or the second (b) layer. One can see that trained accuracy is mostly saturated with about $10^{2.4}$ parameters (increasing the number of parameters doesn’t give big improvement). The same saturation effect is seen when fixing the number of units in one of the layers: increasing the number of units in the other layer brings the accuracy value to a maximum and then decreases. Since increasing the number of parameters is computationally expensive, we want a metric that is able to indicate a good architecture regardless the number of parameters contained within it. It is, therefore, important to note that the $\sigma_r$ metric does not have correlation with the number of parameters (Figure 5).
Figure 4: Number of parameters plot against the architectures mean trained performance $\mu_T$, computed over $N_{\text{init}} = 100$ initialisations. One point represents one architecture. The colours represent the number of units in the first layer (a) and the second layer (b).

Figure 5: The number of parameters plot against the scoring metric $\sigma_r$, or the relative standard deviation of the untrained performance, computed over $N_{\text{init}} = 100$ initialisations. One point stands for one architecture. The colours represent the number of units in the first layer (a) and the second layer (b).

3.2. Testing the scoring metric $\sigma_r$

The results of the $\sigma_r$ performance with CIFAR-10, CIFAR-100 and ImageNet16-120 are given in the Table 2. We present our results based on 100 initialisations ($N_{\text{init}} = 100$), for the $N_{BS} = 256$, both used by Mellor et al. 17 and during the NAS-Bench-201 training. We also provide the two sample t-test p-values to see the statistical significance of differences between our results and those of Mellor et al. 17 (values of p-value $< 0.05$ mean that results are statistically different, otherwise, they are considered similar). Comparisons are made both with the best performing $N_a$ and with $N_a$ fixed to 100 architectures.

The effects of number of iterations and number of selected architectures are shown on figures 6 and 7 respectively, on an example of CIFAR-10. The number of picked architectures considerably increases the overall performance, since there is more chance to have a good architecture. The number of iterations improves the precision of the method. Similar plots for CIFAR-100 and
et al. on CIFAR-10 and ImageNet-16-120, still being worse on CIFAR-100. The accuracies are statistically comparable to Mellor’s best reported accuracies across both datasets.

Performing architecture is expected to be stable against weight initializations (low \(\sigma\)) and training methods (REINFORCE vs. BOHB). Table 2 shows the comparison of trainless \(\sigma_r\) metric performance against existing NAS algorithms. Our method outperforms Mellor et al. on CIFAR-10/CIFAR-100/ImageNet16-120 format. Finally, the two sample t-test p-values are provided for two cases: when comparing best performing \(N_u\) (bold), and with a fixed \(N_u=100\).

ImageNet16-120 can be found in Appendix (Figures A.8, A.9, A.10, A.11).

4. Discussion

The main aim of this study is to find a metric allowing to distinguish the best performing architecture among many based on untrained performance. According to our analysis, based on the reduced MNIST dataset, the coefficient of variation of untrained accuracy \(\sigma_u\) might be one. This means, that the best performing architecture is expected to be stable against weight initialisations (low standard deviation \(\sigma_U\)), as well as to demonstrate high untrained performance (high \(\mu_U\)).

When testing the \(\sigma_r\) metric on CIFAR and ImageNet, our results are clearly above random, which proves the efficiency of the metric. Comparing our results against those from Mellor et al. on CIFAR and ImageNet, similar overall performances are observed. The test results achieved by our algorithm on CIFAR-10 and ImageNet-16-120 are statistically comparable to Mellor’s best reported accuracies (91.78 ± 1.45 vs 91.90 ± 2.27), while CIFAR-100 gives a slightly worse prediction results on our side. If we compare performances with all equals, for the number of sampled architectures \(N_u = 100\), our method outperforms Mellor et al. on CIFAR-10 and ImageNet16-120, still being worse on CIFAR-100. The fact that our algorithm is statistically above the random baseline means that it does indeed point towards the most suitable architectures.

As it was mentioned above, our algorithm involves two extra hyperparameters, which may be considered as a disadvantage. The first one is the batch size...
Figure 6: Comparison of the relative standard deviation $\sigma_r$ performance against mean trained accuracy $\mu_T$ for CIFAR-10 dataset for different number of selected architectures $N_a \in [10, 25, 50, 100, 1000, 5000]$. Statistics are computed over $N_{init} = 100$ initialisations. One point stands for one architecture. The colours represent the logarithm of the total number of trainable parameters.

Figure 7: Comparison of the relative standard deviation $\sigma_r$ performance against mean trained accuracy $\mu_T$ for CIFAR-10 dataset. Statistics are computed over varying number of initialisations $N_{init} \in [3, 5, 10, 25, 50, 100]$. Number of architectures $N_a = 1000$. One point stands for one architecture. The colours represent the logarithm of the total number of trainable parameters.
size: there are significant deviations on the prediction power (with different optimal $N_{BS}$ for each dataset, see Table A.3). The second is the number of initialisations. Also, the fact that our method requires multiple initialisations of each tested architecture leads to a significantly slower performance compared to Mellor et al. (running time grows linearly with the number of sampled architectures). Yet, comparing to the methods that require training, the absolute performance speed remains high (tens to hundreds of seconds). Moreover, $\sigma_r$ metric is easy to interpret and to implement, making the results more confident.

Prediction accuracy improves with the number of sampled architectures (for any batch size). This is a natural consequence of the fact that the chance of having a well performing architecture among many architectures is higher than among few (which is confirmed by random selection tests, see Table 2). Note that in the work of Mellor et al. increasing the number of sampled architectures does not improve the result, which is counterintuitive. While this could be a statistical artefact for CIFAR data, for ImageNet the difference between $N_a = 10$ and $N_a = 100$ is statistically significant (p-value of $5.8e^{-7}$). Which means that in their case, the number of sampled architectures $N_a$ should be considered as a hyperparameter.

Finally, both methods are comparable in the sense, that they rather filter out bad architectures, than choosing the best one.

However, it is important to note, that the NAS-Bench-201 benchmark uses a fixed set of hyperparameters for training each of the architectures. There is a chance that the architecture that is reported to be the best by our algorithm would actually be the best, if trained with the right hyperparameters. We plan to check this fact, as well as to investigate the effect of the batch size, in our future work.

5. Conclusions

In this work we present an entirely trainless method of neural architecture search, based on the relative standard deviation of the untrained accuracy $\sigma_r = \sigma/\mu$. Our metric allows to find a good architecture among thousands in less than 200 seconds. While this might be slower than the other trainless method developed by Mellor et al. [17], we consider that our metric is transparent, easy to implement and interpret, and shows expected behaviour with increasing number of architectures. We could achieve the best accuracies of $91.90 \pm 2.27$, $64.08 \pm 5.63$ and $38.76 \pm 6.62$ for CIFAR-10, CIFAR-100 and a downscaled version of ImageNet, respectively, when using the default batch size of 256. These values are statistically above the random baseline. Results mean that a good architecture should show high stability against initialisations. $\sigma_r$ is a batch size dependent metric, and we plan to pay more attention to this point in our future studies. At present, we would suggest to use the $\sigma_r$ metric as a pre-selection criterion.
6. Acknowledgement

We would like to express our deepest gratitude to Dr. Ayako Nakata and Dr. Guillaume Lambard for continuous support and constructive criticism of the manuscript. We also thank Dr. Keitaro Sodeyama for the financial support of the research.

References

[1] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. Le, A. Kurakin, Large-scale evolution of image classifiers, arXiv preprint arXiv:1703.01041 (2017).
[2] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, K. Kavukcuoglu, Hierarchical representations for efficient architecture search, arXiv preprint arXiv:1711.00436 (2017).
[3] M. Suganuma, S. Shirakawa, T. Nagao, A genetic programming approach to designing convolutional neural network architectures, in: Proceedings of the genetic and evolutionary computation conference, 2017, pp. 497–504.
[4] T. Elsken, J. H. Metzen, F. Hutter, Efficient multi-objective neural architecture search via lamarckian evolution, arXiv preprint arXiv:1804.09081 (2018).
[5] P. J. Angeline, G. M. Saunders, J. B. Pollack, An evolutionary algorithm that constructs recurrent neural networks, IEEE Transactions on Neural Networks 5 (1) (1994) 54–65.
[6] E. Real, A. Aggarwal, Y. Huang, Q. V. Le, Regularized evolution for image classifier architecture search, in: Proceedings of the aaai conference on artificial intelligence, Vol. 33, 2019, pp. 4780–4789.
[7] P. I. Frazier, A tutorial on bayesian optimization, arXiv preprint arXiv:1807.02811 (2018).
[8] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, N. De Freitas, Taking the human out of the loop: A review of bayesian optimization, Proceedings of the IEEE 104 (1) (2015) 148–175.
[9] J. Bergstra, D. Yamins, D. D. Cox, Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures, in: Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28, ICML’13, JMLR.org, 2013, p. I–115–I–123.
[10] T. Domhan, J. T. Springenberg, F. Hutter, Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves, in: Proceedings of the 24th International Conference on Artificial Intelligence, IJCAI’15, AAAI Press, 2015, p. 3460–3468.
[11] H. Mendoza, A. Klein, M. Feurer, J. T. Springenberg, F. Hutter, Towards automatically-tuned neural networks, in: Workshop on Automatic Machine Learning, 2016, pp. 58–65.

[12] B. Zoph, Q. V. Le, Neural architecture search with reinforcement learning, arXiv preprint arXiv:1611.01578 (2016).

[13] B. Zoph, V. Vasudevan, J. Shlens, Q. V. Le, Learning transferable architectures for scalable image recognition, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2018, pp. 8697–8710.

[14] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, J. Dean, Efficient neural architecture search via parameter sharing, arXiv preprint arXiv:1802.03268 (2018).

[15] T. Elsken, J. Metzen, F. Hutter, Neural architecture search: A survey. arxiv 2018, arXiv preprint arXiv:1808.05377 (2018).

[16] R. Istrate, F. Scheidegger, G. Mariani, D. Nikolopoulos, C. Bekas, A. C. I. Malossi, Tapas: Train-less accuracy predictor for architecture search, in: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 33, 2019, pp. 3927–3934.

[17] J. Mellor, J. Turner, A. Storkey, E. J. Crowley, Neural architecture search without training, arXiv preprint arXiv:2006.04647v1 (2020).

[18] X. Dong, Y. Yang, Nas-bench-102: Extending the scope of reproducible neural architecture search, arXiv preprint arXiv:2001.00326 (2020).

[19] A. Krizhevsky, G. Hinton, et al., Learning multiple layers of features from tiny images, Master’s thesis, Department of Computer Science, University of Toronto (2009).

[20] P. Chrabaszcz, I. Loshchilov, F. Hutter, A downsampled variant of imagenet as an alternative to the cifar datasets, arXiv preprint arXiv:1707.08819 (2017).

[21] H. Zhou, J. Lan, R. Liu, J. Yosinski, Deconstructing lottery tickets: Zeros, signs, and the supermask, in: Advances in Neural Information Processing Systems, 2019, pp. 3597–3607.

[22] A. Gaier, D. Ha, Weight agnostic neural networks, in: Advances in Neural Information Processing Systems, 2019, pp. 5364–5378.

[23] Y. LeCun, C. Cortes, C. Burges, Mnist handwritten digit database, ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist 2 (2010).

[24] F. Chollet, et al., Keras, https://keras.io (2015).
[25] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, X. Zheng, [Tensorflow: Large-scale machine learning on heterogeneous systems](http://tensorflow.org/) software available from tensorflow.org (2015).

URL [http://tensorflow.org/](http://tensorflow.org/)

[26] K. He, X. Zhang, S. Ren, J. Sun, Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, in: Proceedings of the IEEE international conference on computer vision, 2015, pp. 1026–1034.

[27] V. Nair, G. E. Hinton, Rectified linear units improve restricted boltzmann machines, in: ICML, 2010, pp. 807–814.

[28] R. J. Williams, Simple statistical gradient-following algorithms for connectionist reinforcement learning, Machine learning 8 (3-4) (1992) 229–256.

[29] S. Falkner, A. Klein, F. Hutter, Bohb: Robust and efficient hyperparameter optimization at scale, arXiv preprint arXiv:1807.01774 (2018).
## Appendix A. Appendix

### Table A.3

#### CIFAR-10

| N  | N_{out} | Validation | Test | Validation | Test | Validation | Test | Validation | Test | Validation | Test | Batch size |
|----|---------|------------|------|------------|------|------------|------|------------|------|------------|------|------------|
| 10 | 100     | -1.08 ± 1.37 | 90.94 ± 1.88 | -1.08 ± 1.37 | 90.94 ± 1.88 | -1.08 ± 1.37 | 90.94 ± 1.88 | -1.08 ± 1.37 | 90.94 ± 1.88 | -1.08 ± 1.37 | 90.94 ± 1.88 | -1.08 ± 1.37 | 90.94 ± 1.88 |
| 25 | 100     | -1.08 ± 1.37 | 90.94 ± 1.88 | -1.08 ± 1.37 | 90.94 ± 1.88 | -1.08 ± 1.37 | 90.94 ± 1.88 | -1.08 ± 1.37 | 90.94 ± 1.88 | -1.08 ± 1.37 | 90.94 ± 1.88 | -1.08 ± 1.37 | 90.94 ± 1.88 |
| 100| 100     | -1.08 ± 1.37 | 90.94 ± 1.88 | -1.08 ± 1.37 | 90.94 ± 1.88 | -1.08 ± 1.37 | 90.94 ± 1.88 | -1.08 ± 1.37 | 90.94 ± 1.88 | -1.08 ± 1.37 | 90.94 ± 1.88 | -1.08 ± 1.37 | 90.94 ± 1.88 |

#### CIFAR-100

| N  | N_{out} | Validation | Test | Validation | Test | Validation | Test | Validation | Test | Validation | Test | Batch size |
|----|---------|------------|------|------------|------|------------|------|------------|------|------------|------|------------|
| 10 | 100     | -1.08 ± 1.37 | 90.94 ± 1.88 | -1.08 ± 1.37 | 90.94 ± 1.88 | -1.08 ± 1.37 | 90.94 ± 1.88 | -1.08 ± 1.37 | 90.94 ± 1.88 | -1.08 ± 1.37 | 90.94 ± 1.88 | -1.08 ± 1.37 | 90.94 ± 1.88 |
| 25 | 100     | -1.08 ± 1.37 | 90.94 ± 1.88 | -1.08 ± 1.37 | 90.94 ± 1.88 | -1.08 ± 1.37 | 90.94 ± 1.88 | -1.08 ± 1.37 | 90.94 ± 1.88 | -1.08 ± 1.37 | 90.94 ± 1.88 | -1.08 ± 1.37 | 90.94 ± 1.88 |
| 100| 100     | -1.08 ± 1.37 | 90.94 ± 1.88 | -1.08 ± 1.37 | 90.94 ± 1.88 | -1.08 ± 1.37 | 90.94 ± 1.88 | -1.08 ± 1.37 | 90.94 ± 1.88 | -1.08 ± 1.37 | 90.94 ± 1.88 | -1.08 ± 1.37 | 90.94 ± 1.88 |

#### ImageNet-120

| N  | N_{out} | Validation | Test | Validation | Test | Validation | Test | Validation | Test | Validation | Test | Batch size |
|----|---------|------------|------|------------|------|------------|------|------------|------|------------|------|------------|
| 10 | 100     | -1.08 ± 1.37 | 90.94 ± 1.88 | -1.08 ± 1.37 | 90.94 ± 1.88 | -1.08 ± 1.37 | 90.94 ± 1.88 | -1.08 ± 1.37 | 90.94 ± 1.88 | -1.08 ± 1.37 | 90.94 ± 1.88 | -1.08 ± 1.37 | 90.94 ± 1.88 |
| 25 | 100     | -1.08 ± 1.37 | 90.94 ± 1.88 | -1.08 ± 1.37 | 90.94 ± 1.88 | -1.08 ± 1.37 | 90.94 ± 1.88 | -1.08 ± 1.37 | 90.94 ± 1.88 | -1.08 ± 1.37 | 90.94 ± 1.88 | -1.08 ± 1.37 | 90.94 ± 1.88 |
| 100| 100     | -1.08 ± 1.37 | 90.94 ± 1.88 | -1.08 ± 1.37 | 90.94 ± 1.88 | -1.08 ± 1.37 | 90.94 ± 1.88 | -1.08 ± 1.37 | 90.94 ± 1.88 | -1.08 ± 1.37 | 90.94 ± 1.88 | -1.08 ± 1.37 | 90.94 ± 1.88 |

### Notes
- N: Batch size
- N_{out}: Number of output units in the last layer
- Validation and Test: Accuracy (in percentage)
Figure A.8: Comparison of the relative standard deviation $\sigma_r$ performance against mean trained accuracy $\mu_T$ for CIFAR-100 dataset for different number of selected architectures $N_a \in \{10, 25, 50, 100, 1000, 5000\}$. Statistics are computed over $N_{init} = 100$ initialisations. One point represents one architecture. The colours represent the logarithm of the total number of trained parameters.

Figure A.9: Comparison of the relative standard deviation $\sigma_r$ performance against mean trained accuracy $\mu_T$ for CIFAR-100 dataset. Statistics are computed over varying number of initialisations $N_{init} \in \{3, 5, 10, 25, 50, 100\}$. One point stands for one architecture. The colours represent the logarithm of the total number of trained parameters.
Figure A.10: Comparison of the relative standard deviation $\sigma_r$ performance against mean trained accuracy $\mu_T$ for ImageNet16-120 dataset for different number of selected architectures $N_a \in [10, 25, 50, 100, 1000, 5000]$. Statistics are computed over $N_{init} = 100$ initialisations. One point represents one architecture. The colours represent the logarithm of the total number of trained parameters.

Figure A.11: Comparison of the relative standard deviation $\sigma_r$ performance against mean trained accuracy $\mu_T$ for ImageNet-16-120 dataset. Statistics are computed over varying number of initialisations $N_{init} \in [3, 5, 10, 25, 50, 100]$. One point stands for one architecture. The colours represent the logarithm of the total number of trained parameters.