Efficient Guided Evolution for Neural Architecture Search

Vasco Lopes
NOVA LINCS, Universidade da Beira Interior
Covilhã, Portugal
vasco.lopes@ubi.pt

Miguel Santos
NOVA LINCS, Universidade da Beira Interior
Covilhã, Portugal

Bruno Degardin
DeepNeuronic, Universidade da Beira Interior
Covilhã, Portugal

Luís A. Alexandre
NOVA LINCS, Universidade da Beira Interior
Covilhã, Portugal

ABSTRACT

Neural Architecture Search methods have been successfully applied to image tasks with excellent results. However, NAS methods are often complex and tend to quickly converge for local minimas. In this paper, we propose G-EA, a novel approach for guided NAS. G-EA guides the evolution by exploring the search space by generating and evaluating several architectures in each generation at initialisation stage using a zero-proxy estimator, where only the highest-scoring architecture is trained and kept for the next generation. By generating several off-springs from an existing architecture at each generation, G-EA continuously extracts knowledge about the search space without added complexity. More, G-EA forces exploitation of the most performant architectures by descendant generation while at the same time forcing exploration by parent mutation and favouring younger architectures to the detriment of older ones. Experimental results demonstrate the effectiveness of the proposed method. Results show that G-EA achieves state-of-the-art results in NAS-Bench-101 and in all NAS-Bench-201 search space data sets: CIFAR-10, CIFAR-100 and ImageNet16-120, with mean accuracies of 93.99%, 72.62% and 46.04% respectively.

CCS CONCEPTS

• **Computing methodologies** → **Computer vision**; *Neural networks*; Machine learning algorithms; Search methodologies.

KEYWORDS

Neural Architecture Search, Optimization, Evolution

ACM Reference Format:
Vasco Lopes, Miguel Santos, Bruno Degardin, and Luís A. Alexandre. 2022. Efficient Guided Evolution for Neural Architecture Search. In *Genetic and Evolutionary Computation Conference (GECCO '22), July 9–13, 2022, Boston, MA, USA*. ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3520304.3528936

1 INTRODUCTION

Convolutional Neural Networks (CNNs) have been extensively applied with success to a panoply of tasks with unprecedented results, such as image classification [5, 11] and text analysis [4]. Their inherent capability of feature extraction allows CNNs to be easily applied and transferred to different problems. Over the years, several carefully designed architectures have incrementally outperformed the state-of-the-art by proposing novel components and training mechanisms [3, 9, 12, 22]. However, designing tailor-made highly performant CNNs for a given problem is a grueling endeavour. Design choices intrinsic to the architectures, layer combination and training require extensive architecture engineering, which is heavily dependant on human expertize and trial and error. Thus, a logical step was to automate the architecture engineering and design, creating a growing interest in Neural Architecture Search (NAS).

NAS has been successfully applied in designing architectures for image and text problems [10, 16, 18, 24]. Despite excellent results obtain by prominent NAS methods, the computational cost of most approaches is high, which in some cases can be in the order of months of GPU computation [14, 31, 32]. To mitigate this, several performance estimation strategies have been proposed to reduce the time constraint of NAS methods, by mainly conducting low-fidelity estimates, learning curve extrapolations, statistical approaches [10, 17, 19, 23] or by proposing one-shot methods, where the weights of the generated models are inherited [15, 20, 25]. However, searching through high-dimensional search spaces is highly complex, even when there is some prior knowledge about the space. Most prominent NAS methods fail to generalise to new data sets due to fast convergence to local minima [8, 26], thus hindering the search and method's applicability. The most reliable approach to obtain information about the search space while searching is to fully train generated architectures and optimise the search based on the most performant ones. However, this is costly, and results are highly dependant on the training schemes and initialisation setups. Therefore, zero-proxy estimators present an attractive solution, where statistics are drawn from the generated architectures to score them at initialisation stage [17, 19]. These methods are time efficient and capable of performing good correlations between the score and respective accuracies when the architectures are trained.

This paper proposes G-EA, an evolutionary NAS method that leverages zero-proxy estimation to efficiently guide the search. By using an evolutionary strategy where operations can be mutated and younger architectures are prefered, G-EA forces an exploitation of the most performant architectures, and an exploration of the search space by performing mutations. More, we solve the problem

GECCO '22 Companion, July 9–13, 2022, Boston, MA, USA

ACM ISBN 978-1-4503-9268-6/22/07.
https://doi.org/10.1145/3520304.3528936

of conducting full evaluation of the generated architectures to obtain knowledge about the search space by generating several architectures in each generation, where all are evaluated at initialisation stage using a zero-proxy estimator and only the highest scoring architecture is trained and kept for the next generation. By doing so, G-EA is capable of continuously extracting knowledge about the search space without compromising the search, resulting in state-of-the-art results in NAS-Bench-101 and NAS-Bench-201 search space data sets: CIFAR-10, CIFAR-100 and ImageNet16-120. The code is publicly available at **https://github.com/VascoLopes/GEA**.

Our contributions can be summarized as:

- We propose a guided NAS method based on evolutionary strategies and zero-proxy estimation to generate convolutional neural networks, empirically showing that guided mechanisms can be used without compromising time efficiency nor the generated models performance.
- We achieve state-of-the-art results in NAS-Bench-101 search space and all data sets from NAS-Bench-201 search space: CIFAR-10, CIFAR-100 and ImageNet16-120.

2 RELATED WORK

NAS was initially proposed as a Reinforcement Learning (RL) problem, where a controller is trained based on the generated architecture's performances to incrementally sample more efficient ones [31]. Follow-up approaches focused on improving the overall performance, and the computation required to frame NAS as a RL problem by proposing the use of different learning strategies, distributed computing, and novel incremental sampling strategies [10, 20, 24, 30]. DARTS [15], proposed the use of gradients to generate architectures by performing continuous relaxation of the parameters using a bi-level gradient optimization. These methods served as basis for follow-up weight-sharing NAS methods and one-shot models [2, 6, 7, 13, 29]. Differently, REA proposed the to frame NAS as a evolutionary computation problem [21], where architectures are evolved through operation and hidden states mutations. However, most evolutionary computation methods are computationally heavy, and quickly converge to non-optimal minimas. To improve latter, guiding mechanisms have been proposed. PNAS introduced consortium learning to the search, where the design of architectures is gradual, based on the evaluation of increasingly larger networks [14]. NPENAS guides an evolutionary search by using a graph-based uncertainty estimation network and a performance predictor. In [1], the authors evaluate the similarity of the internal activations of the generated architectures against a known one, e.g., ResNet, via representational similarity analysis. [28] proposes the use of landmark architecture's evaluation to regularize the ranking of child architectures in super-net settings.

3 PROPOSED METHOD

3.1 Search Method

G-EA starts by randomly generating C architectures from the search space of possible architectures, $\mathcal{A}$. The architectures that belong to the search space have equal probabilities of being randomly sampled. Sampled architectures are then evaluated using a zero-proxy estimator that scores the architectures at initialisation stage, without requiring any training (the zero-proxy estimation mechanism is detailed in section 3.2). Then, from the C scored architectures, only the top P scoring ones are added to the population and trained to extract their fitness, f. The fitness, f, is the validation accuracy after a partial train (few epochs). By scoring C architecture at initialisation stage, G-EA acquires knowledge regarding the search space, which is then exploited by selecting the top performant architectures, thus guiding the upcoming search by weeding out bad architectures.

Once the initial population is defined, the evolution takes place for C cycles. At each iteration, the first step is to randomly and uniformly sample S architectures from the population. Then, the architecture with the highest fitness score, f, from the pool of S architectures is selected to be the parent of the next generation (cycle). To generate new architectures, G-EA performs a mutation over the parent architecture. The mutation works by randomly changing one operation of the architecture by another from the pool of operations. P new architectures are generated at each cycle by performing operation mutations over the selected parent, which are then scored using the zero-proxy estimator. The highest-scoring architecture is kept and added to the population after evaluating its fitness. By generating and evaluating P architectures, the search method can find which is the best direction to evolve the parent through the search space. This allows the method to be guided through a complex space without jeopardizing the time required to perform the evolution or the search method's complexity. When the new architecture is added to the population, a regularization mechanism takes place, where the oldest architecture is removed and discarded, thus forcing exploration of the search space by favouring younger architectures that represent new settings evolved by prior acquired knowledge.

Inherently, higher P values represent a higher degree of exploration of the search space, while higher S values represent higher exploitation by increasing the probability of the best architectures in the population being selected as parents for the next generation.

3.2 Zero-proxy Estimator

The goal of scoring architectures at initialisation stage is to provide knowledge about the search space, thus allowing guiding the search to optimal settings. For this, we use a zero-proxy estimator based on Jacobian covariance. This allows us to quickly evaluate if an architecture is good without requiring any training, thus allowing the selection of a generated architecture to be added to the population with more confidence that the search is being correctly guided to good spaces. To do this, we can define a linear mapping, $w_i = f(\mathbf{x}_i)$, which maps the input $\mathbf{x}_i \in \mathbb{R}^D$, through the network, $f(\mathbf{x}_i)$, where $\mathbf{x}_i$ represents an image that belongs to a batch $\mathbf{X}$, and D is the input dimension [17]. Then, the Jacobian of the linear map can be computed using: $\mathbf{J}_i = \frac{\partial f(\mathbf{x}_i)}{\partial \mathbf{x}_i}$.

This allows us to evaluate the architecture's behaviour for different images by calculating the Jacobian $\mathbf{w}_i$ for different data points, $f(\mathbf{x}_i)$, of a single batch $\mathbf{X}$, $i \in 1, \cdots, N$:

$$\mathbf{J} = \begin{pmatrix} \frac{\partial f(\mathbf{x}_1)}{\partial \mathbf{x}_1} & \frac{\partial f(\mathbf{x}_2)}{\partial \mathbf{x}_2} & \cdots & \frac{\partial f(\mathbf{x}_N)}{\partial \mathbf{x}_N} \end{pmatrix}^\top \tag{1}$$

$\mathbf{J}$ then contains information about the architecture's output with respect to the input for several images. We can split this into classes and evaluate how an architecture models complex functions at initialisation stage and its effect on images that belong to the same

Table 1: Mean test accuracy (%) and standard deviation across 50 runs in NAS-Bench-101 CIFAR-10 data set.

Method	Search Time (s)	Mean Test Accuracy (%)
RS	N/A	90.38±5.51
REA	26676.49	93.12±0.48
G-EA (ours)	30128.32	**93.99±0.25**

class. To do that, we split $\mathbf{J}$ into several sets, where each set, $\mathbf{M}_k$, contains all $\mathbf{J}_i$ that belong to the same class k. Then, we can calculate a per-class correlation matrix, $\Sigma_{\mathbf{M}_k}$, using the obtained sets, $\mathbf{M}_k$, where $k = 1, ...K$.

Individual correlation matrices provide information about how a single architecture treats images for each class. However, different correlation matrices might yield different sizes, as the number of images per class differ. To be able to compare different correlation matrices, they are individually evaluated:

$$\mathbf{E}_k = \begin{cases} \sum_{i=1}^{N}\sum_{j=1}^{N} log(|(\Sigma_{\mathbf{M}k})_{i,j}| + t), & \text{if } K \leq \tau \\ \frac{\sum_{i=1}^{N}\sum_{j=1}^{N} log(|(\Sigma_{\mathbf{M}c})_{i,j}|+t)}{||\Sigma_{\mathbf{M}k}||}, & \text{otherwise} \end{cases} \tag{2}$$

where t is a small-constant with the value of 1×10^{-5}, and K is the number of classes in batch $\mathbf{X}$, and $||.||$ represents the size of the set X.

Finally, an architecture is scored based on the individual evaluations of the correlation matrices by:

$$z = \begin{cases} \sum_{w=1}^{K} |\mathbf{e}_w|, & \text{if } K \leq \tau \\ \frac{\sum_{i=1}^{K}\sum_{j=i+1}^{K} |\mathbf{e}_i - \mathbf{e}_j|}{||\mathbf{e}||}, & \text{otherwise} \end{cases} \tag{3}$$

where $\mathbf{e}$ is a vector that contains all the correlation matrices' scores. The final score is dependant on the number of classes present in $\mathbf{X}$, as data sets with a higher number of classes commonly have more noise, which is mitigated by conducting a normalized pair-wise difference. In our experiments, we empirically defined $\tau = 100$, based on the search space and data sets used.

We can then use z to rank the generated architectures, providing an efficient mechanism of differentiating between bad and good architectures. Thus allowing the search to be guided towards better settings without compromising the search cost.

4 EXPERIMENTS

4.1 Search Spaces

To evaluate the effectiveness of the proposed NAS algorithm, we utilise two different search spaces: NAS-Bench-101 [27] and NAS-Bench-201 benchmarks [8], with 423,624 and 15,625 architectures respectively. These benchmarks were designed to have tractable NAS search spaces with metadata for the training of thousands of architectures within those search spaces.

4.2 Results and Discussion

First, we evaluate the proposed method on NAS-Bench-101. For this, we fixed $P/S/C = 10/5/200$, following standard settings used and assessed by prior works [8, 21], and directly compare it against random search (RS) and REA. In Table 1 we present this comparison in terms of search cost, in seconds, and mean test accuracy and standard deviation, calculated from executing G-EA and REA 50 times. From the results, it is clear that G-EA outperforms REA and heavily improves when compared against RS. G-EA is highly efficient, requiring only approximately 0.3 GPU days to execute each run. The results show that the guiding mechanism can improve the search, promoting regions that yield better architectures.

In Table 2 we further compare G-EA using $P/S/C = 10/5/200$ against other state-of-the-art methods on the NAS-Bench-201 search space, using as evaluation metrics the mean accuracy, standard deviation, and search time in seconds, across the 3 data sets. G-EA consistently outperforms both weight sharing and non-weight sharing NAS methods, achieving state-of-the-art results in all three data sets. Even though G-EA evaluates $C \times P$ architectures with the zero-proxy estimator and further evaluates C architectures by partially training them, it requires a similar search time as REA under the same settings and considerably less than most weight sharing methods. Lower standard deviation also indicates that G-EA is precise and capable of generating high performant architectures, which is especially valid in ImageNet16-120, a data set with low-resolution images and high levels of noise, in which G-EA considerably outperforms existing NAS methods.

The obtained results show that an evolutionary strategy, coupled with a mechanism to quickly evaluate architectures to guide the search, can achieve state-of-the-art results while still having competitive search times. Despite the complexity of search spaces and severe difficulty in obtaining their global information, the results shed insights that guiding mechanisms powered by scoring architectures at initialisation stages give the advantage of acquiring preliminary information regarding which direction the search should evolve. Therefore, G-EA can converge to better results by avoiding local minimas, while still being efficient in terms of the time required by the search method.

5 CONCLUSIONS

This paper proposes G-EA, a guided evolution strategy for neural architecture search by leveraging zero-proxy estimation of untrained architectures. G-EA forces exploitation of the most performant architectures by descendant generation and an exploration of the search space by conducting mutations. G-EA guides the evolution by generating several architectures in each generation and having them evaluated at initialisation stage using a zero-proxy estimator, thus being capable of continuously extracting knowledge about the search space without compromising the search, resulting in state-of-the-art results in NAS-Bench-101 and NAS-Bench-201 search spaces.

ACKNOWLEDGMENT

This work was supported by 'FCT - Fundação para a Ciência e Tecnologia' through the research grants '2020.04588.BD' and 'UI/BD/150765/2020', partially supported by NOVA LINCS (UIDB/04516/2020) with the financial support of FCT, through national funds and CENTRO-01-0247-FEDER-113023 - DeepNeuronic.

Table 2: Comparison of manually designed networks and several search methods evaluated using the NAS-Bench-201 benchmark. Performance is shown in terms of accuracy (%) with mean±std, on CIFAR-10, CIFAR-100 and ImageNet-16-120. Search times are the mean time required to search for cells in CIFAR-10. Search time includes the time taken to train networks as part of the process where applicable. Table adapted from [8, 17, 19].

Method	Search Time (s)	CIFAR-10		CIFAR-100		ImageNet-16-120	
		validation	test	validation	test	validation	test
Manually designed							
ResNet	-	90.83	93.97	70.42	70.86	44.53	43.63
Weight sharing							
RSPS	7587	84.16±1.69	87.66±1.69	59.00±4.60	58.33±4.34	31.56±3.28	31.14±3.88
DARTS-V1	10890	39.77±0.00	54.30±0.00	15.03±0.00	15.61±0.00	16.43±0.00	16.32±0.00
DARTS-V2	29902	39.77±0.00	54.30±0.00	15.03±0.00	15.61±0.00	16.43±0.00	16.32±0.00
GDAS	28926	90.00±0.21	93.51±0.13	71.14±0.27	70.61±0.26	41.70±1.26	41.84±0.90
SETN	31010	82.25±5.17	86.19±4.63	56.86±7.59	56.87±7.77	32.54±3.63	31.90±4.07
ENAS	13315	39.77±0.00	54.30±0.00	15.03±0.00	15.61±0.00	16.43±0.00	16.32±0.00
Non-weight sharing							
RS	12000	90.93±0.36	93.70±0.36	70.93±1.09	71.04±1.07	44.45±1.10	44.57±1.25
REINFORCE	12000	91.09±0.37	93.85±0.37	71.61±1.12	71.71±1.09	45.05±1.02	45.24±1.18
BOHB	12000	90.82±0.53	93.61±0.52	70.74±1.29	70.85±1.28	44.26±1.36	44.42±1.49
REA†	26070	91.22±0.25	93.97±0.31	72.36±1.07	72.14±0.86	45.09±0.92	45.55±1.02
G-EA (ours)†	26911	**91.26±0.20**	**93.99±0.23**	**72.62±0.77**	**72.36±0.66**	**45.97±0.72**	**46.04±0.67**

† Results of 25 runs using the same settings: $P/S/C = 10/5/200$, using a single 1080Ti GPU.

ACKNOWLEDGMENTS

REFERENCES

[1] Pouya Bashivan, Mark Tensen, and James J. DiCarlo. 2019. Teacher Guided Architecture Search. In *ICCV*.

[2] Fabio Maria Carlucci, Pedro M Esperança, Marco Singh, Victor Gabillon, Antoine Yang, Hang Xu, Zewei Chen, and Jun Wang. 2019. MANAS: Multi-agent neural architecture search. *arXiv preprint arXiv:1909.01051* (2019).

[3] François Chollet. 2017. Xception: Deep learning with depthwise separable convolutions. In *CVPR*. 1251–1258.

[4] Alexis Conneau, Holger Schwenk, Loïc Barrault, and Yann LeCun. 2017. Very Deep Convolutional Networks for Text Classification. In *EACL*.

[5] Li Deng, Dong Yu, et al. 2014. Deep learning: methods and applications. *Foundations and Trends in Signal Processing* (2014).

[6] Xuanyi Dong and Yi Yang. 2019. One-Shot Neural Architecture Search via Self-Evaluated Template Network. In *ICCV*.

[7] Xuanyi Dong and Yi Yang. 2019. Searching for a robust neural architecture in four gpu hours. In *CVPR*.

[8] Xuanyi Dong and Yi Yang. 2020. NAS-Bench-201: Extending the Scope of Reproducible Neural Architecture Search. In *ICLR*.

[9] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. 2021. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In *ICLR*.

[10] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. 2019. Neural Architecture Search: A Survey. *Journal of Machine Learning Research* (2019).

[11] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. 2016. *Deep learning*. Vol. 1. MIT press Cambridge.

[12] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. 2017. Densely Connected Convolutional Networks. In *CVPR*.

[13] Liam Li and Ameet Talwalkar. 2020. Random search and reproducibility for neural architecture search. In *UAI*.

[14] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. 2018. Progressive neural architecture search. In *ECCV*.

[15] Hanxiao Liu, Karen Simonyan, and Yiming Yang. 2019. DARTS: Differentiable Architecture Search. In *ICLR*.

[16] Vasco Lopes and Luís A Alexandre. 2020. Auto-classifier: A robust defect detector based on an automl head. In *ICONIP*.

[17] Vasco Lopes, Saeid Alirezazadeh, and Luís A Alexandre. 2021. EPE-NAS: Efficient Performance Estimation Without Training for Neural Architecture Search. In *ICANN*.

[18] Vasco Lopes, António Gaspar, Luís A Alexandre, and João Cordeiro. 2021. An AutoML-based Approach to Multimodal Image Sentiment Analysis. In *IJCNN*.

[19] Joseph Mellor, Jack Turner, Amos J. Storkey, and Elliot J. Crowley. 2021. Neural Architecture Search without Training. In *ICML*.

[20] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. 2018. Efficient Neural Architecture Search via Parameters Sharing. In *ICML*.

[21] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V. Le. 2019. Regularized Evolution for Image Classifier Architecture Search. In *AAAI*.

[22] Mingxing Tan and Quoc V. Le. 2019. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. In *ICML*.

[23] Colin White, Arber Zela, Binxin Ru, Yang Liu, and Frank Hutter. 2021. How Powerful are Performance Predictors in Neural Architecture Search?. In *NeurIPS*.

[24] Martin Wistuba, Ambrish Rawat, and Tejaswini Pedapati. 2019. A Survey on Neural Architecture Search. *CoRR* abs/1905.01392 (2019).

[25] Yuhui Xu, Lingxi Xie, Xiaopeng Zhang, Xin Chen, Guo-Jun Qi, Qi Tian, and Hongkai Xiong. 2020. PC-DARTS: Partial Channel Connections for Memory-Efficient Architecture Search. In *ICLR*.

[26] Antoine Yang, Pedro M. Esperança, and Fabio M. Carlucci. 2020. NAS evaluation is frustratingly hard. In *ICLR*.

[27] Chris Ying, Aaron Klein, Eric Christiansen, Esteban Real, Kevin Murphy, and Frank Hutter. 2019. NAS-Bench-101: Towards Reproducible Neural Architecture Search. In *ICML*.

[28] Kaicheng Yu, Rene Ranftl, and Mathieu Salzmann. 2021. Landmark Regularization: Ranking Guided Super-Net Training in Neural Architecture Search. In *CVPR*.

[29] Arber Zela, Thomas Elsken, Tonmoy Saikia, Yassine Marrakchi, Thomas Brox, and Frank Hutter. 2020. Understanding and Robustifying Differentiable Architecture Search. In *ICLR*.

[30] Zhao Zhong, Junjie Yan, Wei Wu, Jing Shao, and Cheng-Lin Liu. 2018. Practical block-wise neural network architecture generation. In *CVPR*.

[31] Barret Zoph and Quoc V. Le. 2017. Neural Architecture Search with Reinforcement Learning. In *ICLR*.

[32] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. 2018. Learning Transferable Architectures for Scalable Image Recognition. *CVPR* (2018).