Guided Evolution for Neural Architecture Search

Vasco Lopes
NOVA Lincs, Universidade da Beira Interior
vasco.lopes@ubi.pt

Miguel Santos
NOVA Lincs, Universidade da Beira Interior
miguel.santos@ubi.pt

Bruno Degardin
Universidade da Beira Interior
bruno.degardin@ubi.pt

Luís A. Alexandre
NOVA LINCS, Universidade da Beira Interior
luis.alexandre@ubi.pt

Abstract

Neural Architecture Search (NAS) methods have been successfully applied to image tasks with excellent results. However, NAS methods are often complex and tend to converge to local minima as soon as generated architectures seem to yield good results. In this paper, we propose G-EA, a novel approach for guided evolutionary NAS. The rationale behind G-EA, is to explore the search space by generating and evaluating several architectures in each generation at initialization stage using a zero-proxy estimator, where only the highest-scoring network is trained and kept for the next generation. This evaluation at initialization stage allows continuous extraction of knowledge from the search space without increasing computation, thus allowing the search to be efficiently guided. Moreover, G-EA forces exploitation of the most performant networks by descendant generation while at the same time forcing exploration by parent mutation and by favouring younger architectures to the detriment of older ones. Experimental results demonstrate the effectiveness of the proposed method, showing that G-EA achieves state-of-the-art results in NAS-Bench-201 search space in CIFAR-10, CIFAR-100 and ImageNet16-120, with mean accuracies of 93.98%, 72.12% and 45.94% respectively.

1 Introduction

Convolutional Neural Networks (CNNs) have been extensively applied with success to a panoply of tasks, from image classification [5, 13], to semantic segmentation [12], text analysis [4], amongst many others [16]. Their inherent capability of feature extraction allows CNNs to be easily applied and transferred to different problems. Over the years, several brilliantly and carefully designed architectures have incrementally out-performed the state-of-the-art by proposing novel components and mechanisms, such as skip and residual connections, faster and less size intensive operations and attention mechanisms [17, 28, 14, 15, 3, 29, 9]. However, designing tailor-made highly performant CNNs for a given task is extremely difficult, as the required design choices intrinsic to the architectures, layer combination and training requires extensive architecture engineering. Thus, there is a growing interest in Neural Architecture Search (NAS) to automate architecture engineering and design.

NAS has successfully been successfully applied in the task of designing different types of neural network's architectures [32], specially for image and text problems [11, 32]. These methods are commonly composed of three components, being the first the search space, which specifies the possible operations to be sampled and their connections, ultimately defining the type of architectures that the search method can generate. The second component is the search method, which represents the approach used to explore the search space and generate architectures. The most common approaches are reinforcement learning, evolutionary strategies and gradient-based methods, which commonly

35th Conference on Neural Information Processing Systems (NeurIPS 2021), Sydney, Australia.

work by updating a controller to sample more efficient architectures based on the performance of the generated models. Finally, the performance estimation strategy defines how the generated architectures are evaluated. Thus, the goal of a NAS method is to, based on the search method, efficiently search a large set of possible networks to find an optimal architecture for a given problem. Despite the 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 [38, 19, 39]. To mitigate this, interesting approaches focus on a cell-based design, where NAS methods design small cells that are replicated through an outer-skeleton, thus alleviating the complexity of the search space [26, 38, 39, 2]. More, 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 [31, 11] or by proposing one-shot methods, where the weights of the generated models are inherited [20, 23, 33]. Searching through extensive search spaces is highly complex, even when there is some prior knowledge about the space. It has been shown that some of the most prominent NAS methods fail to generalise to new datasets due to converging extremely fast to local minima [8, 34]. 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, thus requiring no training [21, 22]. 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 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 networks, and an exploration of the search space by conducting mutations. More, we solve the problem of conducting full evaluation of the generated networks 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 network 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-201 search space in CIFAR-10, CIFAR-100 and ImageNet16-120.

Our contributions can be summarized as:

- We propose a novel guided NAS method based on evolutionary strategies and zero-proxy estimation to generate image classifier architectures - Convolutional Neural Networks.
- We empirically show that guided mechanisms can be used to improve the generated models performance without compromising time efficiency. Also, we detail the algorithm, emphasizing the accessible transferability of the guiding mechanism.
- We achieve state-of-the-art results in the NAS-Bench-201 search space, in all datasets: 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 sample more efficient ones [38]. 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 [37, 11, 32]. ENAS [23], showed that RL could be used to perform NAS in a reasonable time-frame by training a controller to discover architectures through optimal subgraph search within a large computational graph, requiring only a few computational days. DARTS, proposed the use of gradient-based approaches to generate architectures by performing a continuous relaxation of the parameters using a bi-level gradient optimization, resulting in the generation of competitive networks in a few GPU days [20]. These methods served as basis for follow-up weight-sharing NAS methods and one-shot models [2, 7, 18, 6, 36, 33]. Evolutionary strategies are also a common approach for NAS, which takes inspiration from biologic systems in order to generate architectures through a set of mutation operations. NEAT was the first evolutionary method to evolve simple neural networks [27], which served as base and inspiration for methods

Algorithm 1 Guided Evolution

$population \leftarrow$ empty queue ▷ Population.
$history \leftarrow \varnothing$ ▷ Models history.
while $|population| < C$ **do** ▷ Initialize population.
  $model.arch \leftarrow$ RANDOMARCHITECTURE()
  $model.accuracy \leftarrow$ ZEROPROXY($model.arch$)
  add $model$ to right of $population$ ▷ Add model to the end, forcing age
drop the $C - (C - P)$ worst individuals from $population$
for $model \in population$ **do**
  $model.accuracy \leftarrow$ TRAINANDEVAL($model.arch$)
  add $model$ to history
while $|history| < C$ **do** ▷ Evolve for C cycles.
  $sample \leftarrow \varnothing$ ▷ Parent candidates.
  while $|sample| < S$ **do**
    $candidate \leftarrow$ random model from $population$ ▷ Sampling with replacement.
    add $candidate$ to $sample$
  $parent \leftarrow$ highest-accuracy model in $sample$
  $generation \leftarrow$ empty list ▷ Population.
  while $|generation| < P$ **do**
    $child.arch \leftarrow$ MUTATE($parent.arch$)
    $child.accuracy \leftarrow$ ZEROPROXY($model.arch$)
    add $child$ to $generation$
  $top\_child \leftarrow$ highest-performant model in $generation$
  $top\_child.accuracy \leftarrow$ TRAINANDEVAL($model.arch$)
  add top_child to right of $population$
  add top_child to $history$
  remove $dead$ from left of $population$ ▷ Oldest model.
  discard $dead$
return highest-accuracy model in $history$ ▷ Most performant model.

that evolve deeper architectures where parent architectures have their parameters mutated to force evolution towards better performances [25, 10]. REA, is one of the most prominent approaches, in which the evolutionary strategy evolves architectures through operation and hidden states mutations, and also employs a tournament selection that favours younger architectures [24].

Guiding mechanisms have been proposed to improve NAS. PNAS introduced a consortium learning to the search, where the design of architectures is gradual, based on the evaluation of increasingly larger networks [19]. This approach allowed the method to be progressively guided through the search space, avoiding the need to train bad networks due to the estimation of the performance by a predictor network. However, this method still required immense computation. NPENAS guides an evolutionary search by proposing two predictors: a graph-based uncertainty estimation network and a performance predictor. NPENAS achieves a mean accuracy on NAS-Bench-201 CIFAR-10 of 91.07% [30]. In [1], the authors evaluate the similarity of the internal activations of generated architectures against a known one, e.g., ResNet, via representational similarity analysis to obtain knowledge regarding the search. [35] proposes the use of landmark architecture's evaluation to regularize the ranking of child networks in super-net settings, thus guiding the search towards a better ranking correlation between stand-alone networks and the super-net ranking.

In this work, we propose a guided evolutionary method that is inspired by the findings that show that evolving architectures is an efficient approach for NAS, and that zero-proxy estimators provide a reasonably good and extremely fast scoring of untrained networks [24, 21, 31, 32]. By coupling a zero-proxy estimator as a guiding mechanism to the search method, we force further exploitation of settings that are favourable to the architectures being generated, and, at the same time, also allows the exploration of the search space efficiently, by evaluating thousands of networks, providing information to guide the search.

3 Proposed Method

The goal of NAS algorithms is to find an optimal architecture a^* from the space of architectures $\mathcal{A}$, $a^* \in \mathcal{A}$, that maximizes an objective function $\mathcal{O}$. In this paper, we propose G-EA, which frames NAS as an optimization problem where an evolutionary strategy evolves architectures $a \in \mathcal{A}$ based on mutations and guided evolution.

In the following sections, we detail G-EA and the zero-proxy estimator leveraged to create the guiding mechanism.

3.1 Search Method

G-EA is summarised in Algorithm 1. In detail, 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 networks, only the top P scoring architectures are added to the population and trained to extract their fitness, f, which is the validation accuracy. By scoring C networks 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, 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. An example of a mutation using the NAS-Bench-201 search space is visually represented in Fig. 1. 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 network is kept and added to the population after evaluating its fitness. By evaluating P architectures, the search method can find which are the best directions to evolve the parent in the 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, 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.

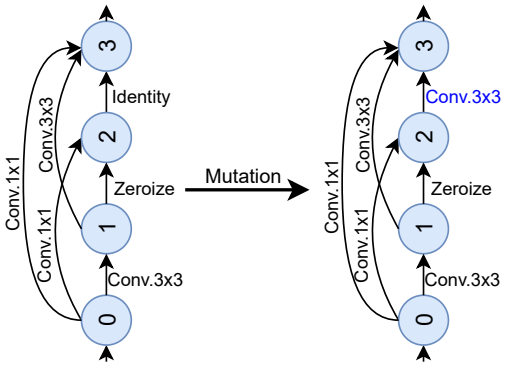


Figure 1: Representation of mutating an operation using NAS-Bench-201 search space.

3.2 Zero-proxy Estimator

To score networks at initialisation stage to aid in the guiding mechanism of the evolution, we use a zero-proxy estimator based on Jacobian covariance. This allows us to quickly evaluate if a network is good without requiring any training, thus allowing the selection of a generated network 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 [21]. Then, the Jacobian of the linear map can be

computed using:

$$\mathbf{J}_i = \frac{\partial f(\mathbf{x}_i)}{\partial \mathbf{x}_i} \tag{1}$$

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

$$\mathbf{J} = \begin{pmatrix} \frac{\partial f(\mathbf{x}_1)}{\partial \mathbf{x}_1} & \frac{\partial f(\mathbf{x}_2)}{\partial \mathbf{x}_2} & \cdots & \frac{\partial f(\mathbf{x}_N)}{\partial \mathbf{x}_N} \end{pmatrix}^\top \tag{2}$$

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

$$\mathbf{E}_k = \begin{cases} \sum_{i=1}^{N} \sum_{j=1}^{N} log(|(\mathbf{\Sigma}_{\mathbf{M}k})_{i,j}| + t), & \text{if } K \leq \tau \\ \frac{\sum_{i=1}^{N} \sum_{j=1}^{N} log(|(\mathbf{\Sigma}_{\mathbf{M}c})_{i,j}| + t)}{||\mathbf{\Sigma}_{\mathbf{M}k}||}, & \text{otherwise} \end{cases} \tag{3}$$

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:

$$z = \begin{cases} \sum_{w=1}^{K} |\mathbf{e}_w|, & \text{if } K \leq \tau \\ \frac{\sum_{i=1}^{K} \sum_{j=i+1}^{K} |\mathbf{e}_i - \mathbf{e}_j|}{||\mathbf{e}||}, & \text{otherwise} \end{cases} \tag{4}$$

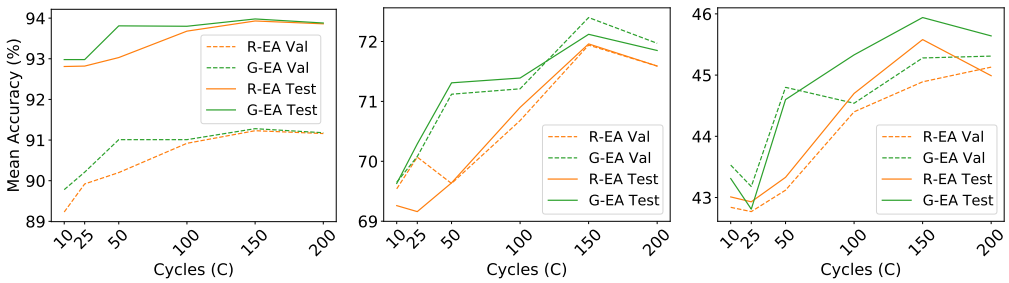
where $\mathbf{e}$ is a vector that contains all the correlation matrices' scores. The final score is dependant on the number of classes present in $\mathbf{X}$, as data sets with a higher number of classes commonly have more noise, which is mitigated by conducting a normalized pair-wise difference.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

We can then use z to rank generated architectures, providing an efficient mechanism of differentiating between bad and good architectures.

4 Experiments

4.1 Search Space

To evaluate the proposed method, we used the NAS-Bench-201 tabular benchmark [8]. NAS-Bench-201 fixes the search space as a cell-based design with 5 operations: zeroize, skip connection, 1×1 convolution, 3×3 convolution, and 3×3 average pooling layer. The cell design comprises six edges and four nodes, where an edge represents a possible operation through two nodes. By fixing the cell size and the operation pool, there are $5^6 = 15625$ possible cells in this search space. To form entire networks, the cells are replicated in an outer-defined skeleton. More, NAS-Bench-201 provides information regarding the training and performance of all possible networks in the search space in three datasets: CIFAR-10, CIFAR-100 and ImageNet16-120, thus allowing a quick prototyping and a controlled setting that allows different NAS methods to be fairly compared, as they are forced to use the search space, training procedures and hyper-parameters.



(a) CIFAR-10 (b) CIFAR-100 (c) ImageNet16-120

Figure 2: Mean accuracy over 10 runs of the proposed method, G-EA, and direct comparison with R-EA for different cycles (C) across CIFAR-10, CIFAR-100 and ImageNet16-120 data sets.

4.2 Results and Discussion

First, we evaluate the importance of the number of generations/cycles, C. This parameter inherently defines the time required for the search procedure. Higher C values will take longer to finish. More, C defines the number of architectures that are evaluated - C^P architectures (P per cycle) are generated and evaluated using the zero-proxy estimation method to provide information about the search space, from which, C architectures (1 per cycle) are selected and trained. The results from this experiment can be seen in Fig 2, where a direct comparison with R-EA is also provided. The results are expressed as the mean accuracy over 10 runs, obtained by the best architecture found by each method both in the validation and test sets. In this experiment, the P/S used to allow a fair comparison was set to $P/S = 5/2$, following the common settings used and extensively evaluated by prior works [8, 24]. For our proposed method, G-EA, P value means that at any given time of the search, the population is equal to 5 architectures, and that from the pool of parents, S, 2 architectures are sampled with replacement in order to elect the parent of the generated architectures at a given cycle. Denote that the sampled parent generates P architectures through mutation per cycle, which are evaluated using the zero-proxy estimator, wherein the top scoring architecture is selected to integrate the population. By selecting $S > 1$ architectures to have the opportunity of being a parent, we are leveraging the intrinsic exploitation characteristics of the evolutionary strategy, while by generating P architectures, we are forcing an exploitation that guides the search more effectively.

From Fig. 2, it is possible to see that across all datasets, G-EA consistently outperforms R-EA, and is capable of converging to better results even with a low C. These results demonstrate that by providing a guided mechanism, the search method convergences more quickly to regions of the search space that contain better architectures, peaking at $C = 150$ in these settings. Based on these results, in Table 1 we further compare G-EA using $P/S/C = 5/2/150$ 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. Moreover, G-EA is extremely efficient in terms of search time, requiring only 0.2 GPU days to complete the search. Even though G-EA evaluates C^P architectures with the zero-proxy estimator and further evaluates C architectures by 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. This is specially 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 evolutionary strategies 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 achieved results shed insights that guiding mechanisms powered by scoring architectures at initialisation stages give us the advantage of acquiring preliminary information regarding which direction should the search evolve to. Therefore, G-EA is capable of avoiding local minima and quickly converge to better results while still being capable of improving the time required by the search method.

Table 1: 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, 21, 22].

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†	18577	91.23±0.29	93.93±0.31	71.94±1.24	71.96±1.19	44.89±1.00	45.58±1.01
G-EA (ours)†	18567	**91.28±0.12**	**93.98±0.18**	**72.40±0.41**	**72.12±0.35**	**45.28±0.68**	**45.94±0.71**

† Results of 10 runs using the same settings: $P/S/C = 5/2/150$, using a single 1080Ti GPU.

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 networks by descendant generation and an exploration of the search space by conducting mutations. G-EA guides the evolution by exploring the search space by generating several architectures in each generation and having them evaluated at initialisation stage using a zero-proxy estimator, where only the highest-scoring network is trained and kept for the next generation. By generating several architectures from an existing architecture from the population at each generation, 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-201 search space in CIFAR-10, CIFAR-100 and ImageNet16-120, with mean accuracies of 93.98%, 72.12% and 45.94% respectively.

The simplicity of our approach allows it to easily be extended, where the search method is further improved by incorporating new regularisation and mutation mechanisms. Also, the components that compose the guiding mechanism can easily be transferred to other evolutionary algorithms, allowing existing NAS evolutionary methods to be further improved.

Acknowledgments and Disclosure of Funding

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

References

[1] Bashivan, P., Tensen, M., DiCarlo, J.J.: Teacher guided architecture search. In: Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) (October 2019)

[2] Carlucci, F.M., Esperança, P.M., Singh, M., Gabillon, V., Yang, A., Xu, H., Chen, Z., Wang, J.: Manas: Multi-agent neural architecture search. arXiv preprint arXiv:1909.01051 (2019)

[3] Chollet, F.: Xception: Deep learning with depthwise separable convolutions. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 1251–1258 (2017)

[4] Conneau, A., Schwenk, H., Barrault, L., LeCun, Y.: Very deep convolutional networks for text classification. In: Lapata, M., Blunsom, P., Koller, A. (eds.) 15th Conference of the European Chapter of the Association for Computational Linguistics, EACL. Association for Computational Linguistics (2017)

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

[6] Dong, X., Yang, Y.: One-Shot Neural Architecture Search via Self-Evaluated Template Network. In: ICCV. IEEE (2019)

[7] Dong, X., Yang, Y.: Searching for a robust neural architecture in four gpu hours. In: CVPR. pp. 1761–1770 (2019)

[8] Dong, X., Yang, Y.: NAS-Bench-201: Extending the Scope of Reproducible Neural Architecture Search. In: ICLR (2020)

[9] Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., Houlsby, N.: An image is worth 16x16 words: Transformers for image recognition at scale. In: 9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021 (2021)

[10] Elsken, T., Metzen, J.H., Hutter, F.: Efficient multi-objective neural architecture search via lamarckian evolution. In: ICLR (2019)

[11] Elsken, T., Metzen, J.H., Hutter, F.: Neural architecture search: A survey. Journal of Machine Learning Research (2019)

[12] Garcia-Garcia, A., Orts-Escolano, S., Oprea, S., Villena-Martinez, V., Martinez-Gonzalez, P., Garcia-Rodriguez, J.: A survey on deep learning techniques for image and video semantic segmentation. Applied Soft Computing **70**, 41–65 (2018)

[13] Goodfellow, I., Bengio, Y., Courville, A., Bengio, Y.: Deep learning, vol. 1. MIT press Cambridge (2016)

[14] He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: CVPR (2016)

[15] Huang, G., Liu, Z., van der Maaten, L., Weinberger, K.Q.: Densely connected convolutional networks. In: CVPR (2017)

[16] Khan, A., Sohail, A., Zahoora, U., Qureshi, A.S.: A survey of the recent architectures of deep convolutional neural networks. Artificial Intelligence Review (2020)

[17] Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in Neural Information Processing Systems (2012)

[18] Li, L., Talwalkar, A.: Random search and reproducibility for neural architecture search. In: UAI. pp. 367–377. PMLR (2020)

[19] Liu, C., Zoph, B., Neumann, M., Shlens, J., Hua, W., Li, L.J., Fei-Fei, L., Yuille, A., Huang, J., Murphy, K.: Progressive neural architecture search. In: Proceedings of the European conference on computer vision (ECCV) (2018)

[20] Liu, H., Simonyan, K., Yang, Y.: DARTS: Differentiable Architecture Search. In: ICLR (2019)

[21] Lopes, V., Alirezazadeh, S., Alexandre, L.A.: EPE-NAS: Efficient Performance Estimation Without Training for Neural Architecture Search. In: International Conference on Artificial Neural Networks (ICANN) (2021)

[22] Mellor, J., Turner, J., Storkey, A.J., Crowley, E.J.: Neural Architecture Search without Training. In: ICML (2021)

[23] Pham, H., Guan, M., Zoph, B., Le, Q., Dean, J.: Efficient neural architecture search via parameters sharing. In: ICML (2018)

[24] Real, E., Aggarwal, A., Huang, Y., Le, Q.V.: Regularized Evolution for Image Classifier Architecture Search. In: AAAI. pp. 4780–4789. AAAI Press (2019)

[25] Real, E., Moore, S., Selle, A., Saxena, S., Suematsu, Y.L., Tan, J., Le, Q.V., Kurakin, A.: Large-scale evolution of image classifiers. In: Precup, D., Teh, Y.W. (eds.) Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017. Proceedings of Machine Learning Research, vol. 70, pp. 2902–2911. PMLR (2017)

[26] Shu, Y., Wang, W., Cai, S.: Understanding architectures learnt by cell-based neural architecture search. In: 8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020. OpenReview.net (2020)

[27] Stanley, K.O., Miikkulainen, R.: Evolving neural networks through augmenting topologies. Evolutionary computation **10**(2), 99–127 (2002)

[28] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S.E., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A.: Going deeper with convolutions. In: CVPR (2015)

[29] Tan, M., Le, Q.V.: Efficientnet: Rethinking model scaling for convolutional neural networks. In: Chaudhuri, K., Salakhutdinov, R. (eds.) ICML (2019)

[30] Wei, C., Niu, C., Tang, Y., Liang, J.: NPENAS: neural predictor guided evolution for neural architecture search. CoRR **abs/2003.12857** (2020)

[31] White, C., Zela, A., Ru, B., Liu, Y., Hutter, F.: How powerful are performance predictors in neural architecture search? CoRR **abs/2104.01177** (2021)

[32] Wistuba, M., Rawat, A., Pedapati, T.: A survey on neural architecture search. CoRR **abs/1905.01392** (2019)

[33] Xu, Y., Xie, L., Zhang, X., Chen, X., Qi, G., Tian, Q., Xiong, H.: PC-DARTS: partial channel connections for memory-efficient architecture search. In: 8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020. OpenReview.net (2020)

[34] Yang, A., Esperança, P.M., Carlucci, F.M.: Nas evaluation is frustratingly hard. In: ICLR (2020)

[35] Yu, K., Ranftl, R., Salzmann, M.: Landmark regularization: Ranking guided super-net training in neural architecture search. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 13723–13732 (June 2021)

[36] Zela, A., Elsken, T., Saikia, T., Marrakchi, Y., Brox, T., Hutter, F.: Understanding and robustifying differentiable architecture search. In: ICLR (2020)

[37] Zhong, Z., Yan, J., Wu, W., Shao, J., Liu, C.L.: Practical block-wise neural network architecture generation. In: CVPR (2018)

[38] Zoph, B., Le, Q.V.: Neural architecture search with reinforcement learning. In: ICLR (2017)

[39] Zoph, B., Vasudevan, V., Shlens, J., Le, Q.V.: Learning transferable architectures for scalable image recognition. CVPR (Jun 2018)