Multi-Objective Neural Architecture Search for Light-Weight Model

Nannan Li[1] Yaran Chen[1] Zixiang Ding[1] Dongbin Zhao[1] Zhonghua Pang[2] Ruisheng Qin[2]
[1]*State Key Laboratory of Management and Control for Complex Systems,*
Institute of Automation, Chinese Academy of Sciences, Beijing 100190, China
University of Chinese Academy of Sciences, Beijing 101408, China
linannan2017@ia.ac.cn chenyaran2013@ia.ac.cn dingzixiang2018@ia.ac.cn dongbin.zhao@ia.ac.cn
[2]*Key Laboratory of Fieldbus Technology and Automation of Beijing*
North China University of Technology, Beijing 100144, China
zhpang@ncut.edu.cn

***Abstract*—Neural architecture search (NAS) has achieved superior performance in visual tasks by automatically designing an effective neural network architecture. In recent years, deep neural networks are increasingly applied to resource-constrained devices. As a result, in addition to the model performance, model size is another very important factor that requires to consider when designing powerful neural network architectures. Therefore, we propose the multi-objective neural architecture search for light-weight model and name it Light-weight NAS. On one hand, the Light-weight NAS introduces Multiply-ACcumulate (MAC) into the optimize objective to get the architecture with fewer parameters. On the other hand, we simplify the search space and adopt weight sharing to make the search process more efficient. Experimental results indicate that the searched architecture can perform competitive classification accuracy with few parameters on the image classification task, while using less computation cost than the most existing multi-objective NAS approaches.**

***Keywords*—Neural architecture search, light-weight, multi-objective, reinforcement learning, image classification.**

I. Introduction

Because of the feature engineering process is automated, deep learning has demonstrated impressive results in lots of areas. For instance, object detection [1], image classification [2], [3], game artificial intelligence [4], [5], intelligent transportation system, etc. Especially image classification, as one of the fastest growing area, the presentation of many Convolution Neural Networks (CNN) is one of the main factors for this success. Such as AlexNet [6], ResNet [7], DenseNet [8], etc. However, for the desire to deploy these CNN models on resource-constrained devices, some high-performance models with fewer parameters are designed, such as CondenseNet [9], MobileNets [10], ShuffleNet [11], etc. All of these models achieve superior performance in computer vision tasks, and have been widely applied in the image processing area.

However, the design of the network architecture with good performance mainly depends on expert experience, and can be very time-consuming. To solve the problem, NAS begins to be widely researched as a hotspot of automatic machine learning. Through the process of automated architecture engineering, NAS has demonstrated prominent performance in image classification and language models [12]–[15]. Concurrently, some multi-objective neural architecture has also made impressive success for the goal of implementing the high-performance model applications on resource-constrained devices. For example, NSGA-Net [16], an outstanding work that achieves the excellent results in CIFAR. However, its computation cost takes 4 days. It can be seen that the work is very computing-resource intensive and time-consuming. So that it is demanding to make multi-objective neural network architecture search more efficient. At present, one of the most common methods to improve search efficiency is weight sharing. Combining multi-objective and weight sharing, we can search for the smaller model efficiently.

In this paper, we present the Light-weight neural architecture search. As shown in Fig.1, our work is divided into two stages, search stage and final stage. Search stage is to search for the optimal architecture while final stage is to construct the final model. In search stage, we construct a super network that contains all the candidate operations and connections. Then we use controller network LSTM to sample the child network (i.e. sequence code) from the super network. Child network is the subnetwork of super network. In Fig. 2, we show a mini example of the relationship of super network and child network. Of course, The super network is larger than the example in Fig. 2. To optimize the sampled child network, we train the child network and controller network iteratively in search stage. The sampled child network is trained to update the weight w of the super network. Concurrently, the controller network is trained using the policy gradient algorithm with the precision and MAC of the child network as the reward. In final stage, we construct the final model through changing the depth and width of the child network searched in the search stage. And we retrain the final model in final stage. Our method can get the high-performance network architecture with fewer parameters and only need 0.8 GPU-days on single NVIDIA Tesla P100. On CIFAR-10, our model reaches 3.30% test

This work is supported by Youth research fund of the state key laboratory of complex systems management and control No. 20190213 and No. GJHZ1849 International Partnership Program of Chinese Academy of Sciences, and No FA2018111061SOW12 Program of the Huawei Technologies Co Ltd, and Beijing University High-Level Talent Cross-Training Project (Practical Training Plan).

978-1-7281-4094-0/19/$31.00 ©2019 IEEE

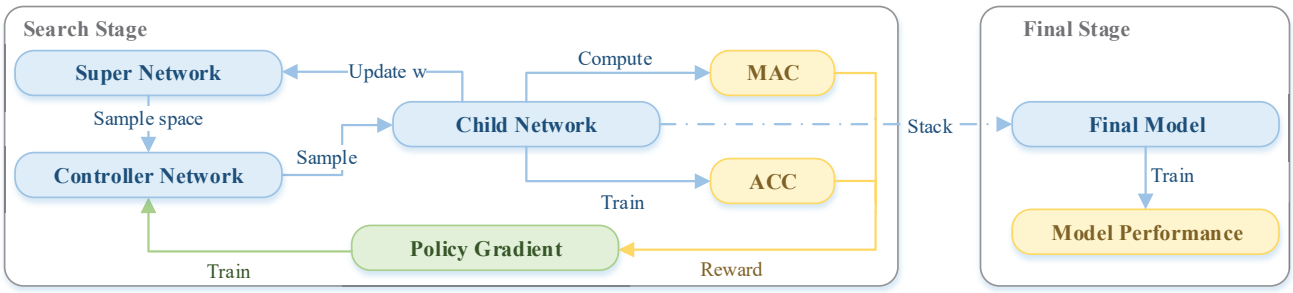

Fig. 1. The overview of the work: Our work establishes a controller network, and use it to sample and train the child network to get the performance to train the controller network. Then we stack the child network to get the final model.

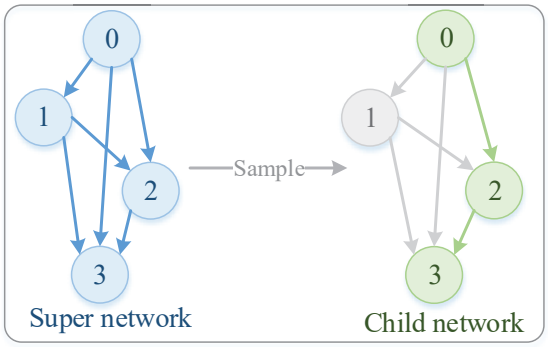

Fig. 2. The mini example of the super network and child network. The green nodes and connections are activated and trained in child model.

error with only 2.3M parameters. On CIFAR-100, our model achieves 18.04% test error with only 2.4M parameters. Our contributions are summarized as follows:

- **Reward Function:** Considering both the MAC and accuracy as the reward function, we optimize them by the policy gradient, which guides the controller network to sample the lighter architecture with competitive performance.
- **Simplified Search Space:** Due to the limited computing-resource, we redesign the search space of NAS under the guidance of expert experience. And by combining the search space and the weight sharing, our work greatly accelerates the performance-evaluation process.
- **Transferable Architecture:** The architecture searched on CIFAR10 can be transferred to CIFAR-100 and can also achieve competitive performance but with smaller parameters.

II. Related Works

We first review the related works on NAS, especially ENAS [17]. Then we outline the main works about multi-objective NAS.

Neural Architecture Search: Recently, NAS has obtained plentiful achievements, especially in the image classification task. And most of the models searched by these works beyond the human-designed models. The current mainstream search strategy mainly includes evolutionary algorithm (EA) and reinforcement learning (RL). There are plentiful representative works of EA-based NAS. The variants of NEAT [18], CoDeepNEAT [19] and rtNEAT [20]. Besides, Google proposes the large scale evolution in [21] and AmoebaNet [22] that achieves state-of-the-art results on CIFAR-10 and ImageNet. RL-based NAS also have many remarkable results. MetaQNN [23] search for optimal architecture through an ϵ-greedy Q-learning strategy with experience replay. BlockQNN [24] extends above work. In [12], Google uses the policy gradient algorithm to search for the entire network. And there are many extension of it, such as NASNet [13], ENAS [17], etc. Of course, other search algorithms (e.g. gradient-based algorithms) also achieve outstanding performance in NAS. For example, DARTS [25], NAO [26], SMASH [27] and so on.

Efficient Neural Architecture Search: The key idea of the ENAS [17] is that it presents the weight sharing which greatly speeds up the architecture search process. In NAS, the weights of network architecture are all randomly initialized and trained from scratch. Different from previous work, ENAS shares these parameters among all architectures. The trained weights of the previous network are retained, and the repeated nodes of new network don't need to resume training. Such that ENAS can quickly search for a set of architectures with high performance. For consideration of speeding up the search process, we adopt the weight sharing to our works.

Multi-Objective Neural Architecture Search: Multi-objective NAS is also mainly based on EA and RL. Multi-objective optimization algorithm based on genetic algorithm (e.g. NSGA) optimizes multiple targets simultaneously to obtain Pareto optimal solutions. And EA-based multi-objective NAS get the decent performance, such as NSGA-Net [16], LEMONADE [28], etc. Versus the EA-based multi-objective NAS, RL-based multi-objective NAS tends to transform the multi-objective optimization to the single-objective optimization problem through incorporating different objects into the reward function. For example, MnasNet [29], RENA [30], etc. There are also some works that use other algorithms.

III. Methods

In this section, we detail our approach. Generally speaking, multi-objective NAS aims to get the Pareto optimal architec-

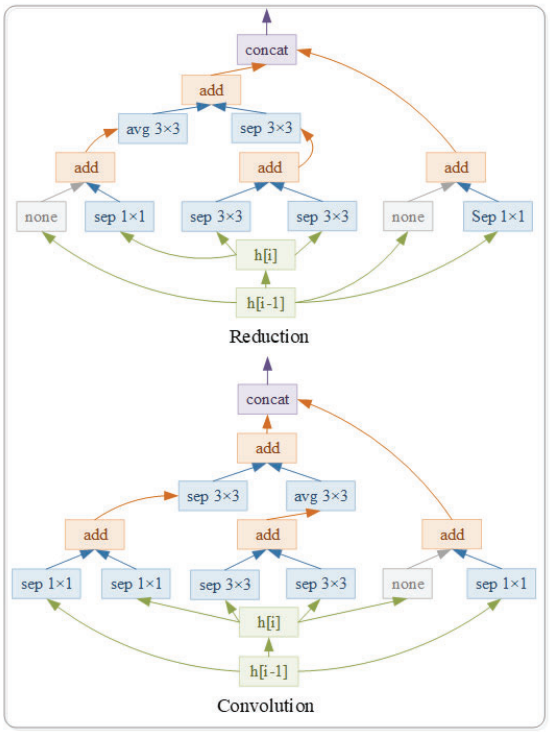

Fig. 3. The example architecture of convolution block and reduction block.

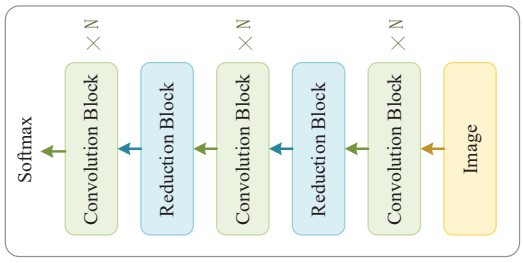

Fig. 4. The form of stacking the convolution block and reduction block.

tures in a certain search space with a defined search strategy. Such that the Light-weight NAS is organized around three main factors: search space, search procedure, and reward function design.

A. Search Space

The search space is critical in NAS. And it determines the paradigm of the searched architecture. Thus the design of search space is mainly up to the target task. And the relatively large search space on the image classification task is proposed by NASNet [13], which consists of thirteen operations. Such a large search space leads to the difficulty of convergence and time-consuming. Indeed, NASNet spends 1800 GPU-days, which is partly caused by its search space.

Therefore, in order to shorten the search time, we redesign the search space. Considering that our network is geared towards mobile devices and the experience from the human-designed networks, we design a search space with only five candidate operations. The specific operations are shown as below:

- 1×1 depthwise-separable convolution
- 3×3 depthwise-separable convolution
- 3×3 average pooling
- 3×3 max pooling
- identity

Inspired by the models like ResNet [7] and DenseNet [8], which just select 1 and 3 as convolution kernel size. Thus we discard the convolution operation with large kernel size, and only choose the 1×1 and 3×3 convolution operations. Besides, in order to reduce model parameters, we only retain the depthwise-separable convolution operations. Though the search space is very disadvantageous in terms of architectural diversity, it is good for searching architectures with few-parameters and high-precision.

B. Search Procedure

The search strategy adopted by Light-weight NAS is policy gradient algorithm. We optimize the architectures through updating the hyperparameters of controller network, which samples the child network in aforementioned candidate operations. Bellow, we introduce entire search procedure through three aspects: controller network, policy gradient algorithm, and weight sharing.

1) Controller network: The architecture is encoded as sequences, and the LSTM is good at generating the string with different length. Thus we choose the LSTM structure as the controller network. The output of the controller network will be as the input of the controller network in next step to predict the new sequence code (i.e. new architecture). The controller network samples architectures via generating the sequence code that represents two types of the blocks. One is the convolution block with operation (convolution, pooling) stride as 1, while the other is the reduction block with operation stride as 2 and double channel number for reducing the size of images (i.e. the feature map). The example architecture is shown in Fig. 3. Then we stack convolution block and reduction block in turns to construct the child network, as shown in Fig. 4. The number of the convolution block N determines the depth of the network. And the performance of child network is used to guide the training of controller network to sample the architectures with better performance.

2) Policy gradient algorithm: In view of the correlation between the child network and controller network LSTM, we choose the policy gradient algorithm, REINFORCE [31], to optimize hyperparameters of the LSTM with the performance of child network as reward. The specific steps are as follows:

Step 1: The controller network samples the child network by generating the sequence code. And the sequence code can be viewed as a list of action $a_{1:L}$.

Step 2: Training the sampled child network to update the weights w of the super network and get reward r that

consists of model classification ACCuracy (*ACC*) and the model *MAC*, defined by:

$$r = ACC - \frac{MAC}{T},$$

where *T* is the constraint factor on the *MAC*. The *ACC* is the classification accuracy of the child network on CIFAR-10. The *MAC* of depthwise-separable convolution is calculated as follows:

$$MAC = C_{in} \times H \times W \times (K \times K + C_{out}),$$

where *K* is the kernel size of filter, *H/W* is the height/width of the output feature and C_{in}, C_{out} is the channels number of input and output. We don't take *MAC* of fully connected layer into account because that each sampled child network has the same fully connected layer as the output layer.

Step 3: Updating the hyperparameters θ of the controller network using the policy gradient algorithm. To be specific, with *r* as reward, the expected reward $J(\theta)$ with respect to the LSTM is maximized by using the REINFORCE rule to compute the policy gradient $\nabla_\theta J(\theta)$. $J(\theta)$ can be computed as follows:

$$J(\theta) = E_{\pi(a_{1:L};\theta)}[r(m;w)],$$

where $r(m;w)$ means r of the model *m* with weight parameters w, $\pi(a_{1:L};\theta)$ is the probability distribution that LSTM predicts the model *m* with parameters θ. $a_{1:L}$ is a list of action, i.e. sampled child network. And the gradient $\nabla_\theta J(\theta)$ can be described as follows:

$$\nabla_\theta J(\theta) = \sum_{l=1}^{L} E_{\pi(a_{1:L};\theta)}[\nabla_\theta \log \pi(a_l \mid a_{l-1:1};\theta) r(m;w)].$$

Step 4: Repeating Step 1 to 4 until the stop condition (e.g. max epochs of the algorithm) is met.

3) Weight sharing: With an eye to improve the search efficiency, we adopt the weight sharing proposed by ENAS. The specific principle of acceleration is the trained weights of the previous child network are retained, and the repeatedly sampled nodes of new child network share the retained weights instead of resume training. Just because of this, search efficiency gets a great breakthrough.

C. Network Training

As shown in Fig.1, there are two phases in our work, the search stage and final stage. The search stage is the core of the experiment to get the optimal architecture and the final stage is to verify the performance of final model. And it is worth noting that we use a proxy in the search stage with the purpose of efficiency. Concretely speaking, there are some points need to be explained in train process of child network and final model. *i)* The child network used in search stage is small for reducing the train time, while the final model is deeper and wider. Both of them are the stack of the architecture which is the decoding of sequence code predicted by the LSTM. Differences between them are the depth and width of the network. Specifically, we change them through adjusting the number of the convolution block N and number of channels. *ii)* The model for CIFAR-100 is searched on CIFAR-10. This proves the generalization of the light-weight model we searched.

IV. Experiments

This section demonstrates the availability of the Light-weight NAS on two benchmark datasets (CIFAR-10 and CIFAR-100) for image classification task.

A. Datasets

CIFAR-10: The CIFAR-10 dataset contains 10 types of RGB images with size as 32 × 32. And there are 60,000 images, the training images and validation images are 50,000 and 10,000 respectively. We also use some pre-processing on the data, e.g. randomly flipping, centrally padding and randomly cropping.

CIFAR-100: The CIFAR-100 dataset contains 100 types of images with number as 60,000, and training images and validation images are 50,000 and 10,000 respectively.

B. Details

The experiment contains two phases. In search stage, we set N as 2. The output channel number of first convolution block is set to 20, and it doubles when processing the reduction block. The constraint factor *T* is set to 5×10^6. Moreover, we use Nesterov momentum [32] train the weight parameters of the super network, and we use the cosine schedule change the learning rate. The max learning rate is set to 0.05 while the min is 0.001. The epoch and batch size are set to 310 and 160 respectively. The hyperparameters θ of LSTM are trained by using Adam optimizer with learning rate set to 0.00035. In final stage, the experimental settings are consistent for CIFAR-10 and CIFAR-100, N is set to 5, output channel number is set to 36, the epoch is set to 630 and 200, respectively. Other settings are consistent with the training of child network in search stage.

C. Results

Fig. 5 shows the influence of constraint factor *T* on search results, precision and *MAC* of the child network. As shown in Fig. 5 (a), the *MAC* of the architecture with *MAC* limit is obviously smaller than architecture without *MAC* limit. we can see that when there is no limit, *MAC* is on the rise in initial to get higher accuracy and then gradually stabilizes. After adding restrictions on the *MAC*, the trend of *MAC* is inverse to meet the limit. Fig. 5 (b) demonstrates the *ACC* comparison. The sudden decrease of *ACC* arises from the learning rate schedule, cosine schedule. The architecture with *MAC* limit have lower *ACC* by reason of the smaller *MAC*. Moreover, the training of searched architecture doesn't completely converge so as to reduce the search cost. In conclusion, the results are comforting. Not only the MAC can be limited to a smaller value, but also the *ACC* can achieve a competitive value.

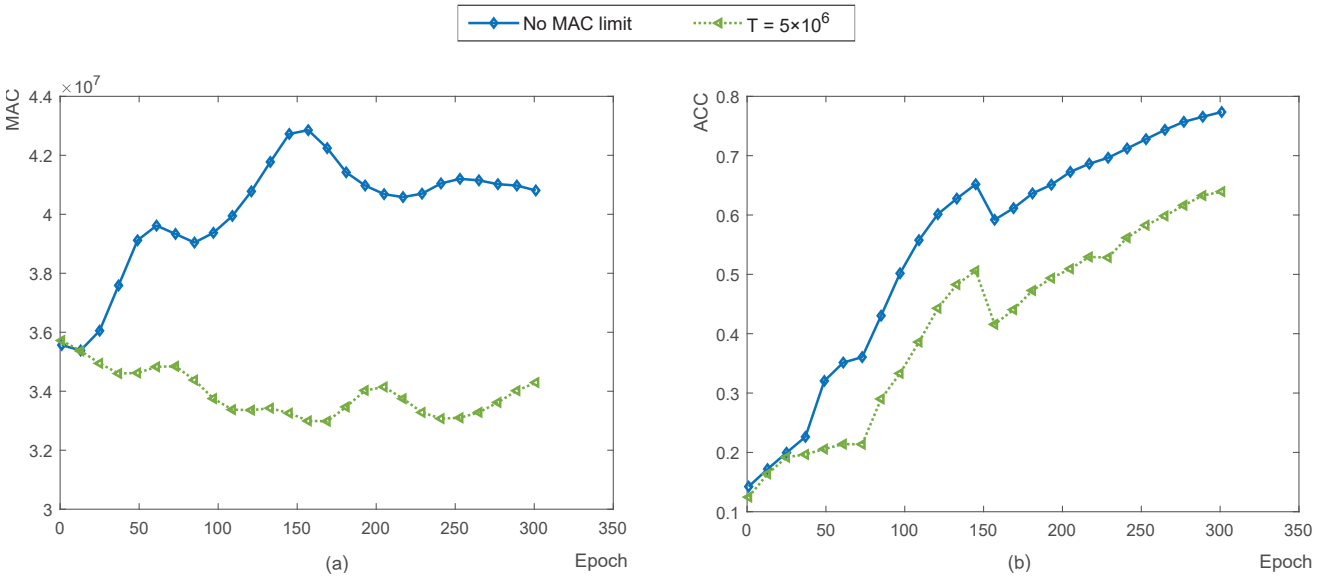

Fig. 5. Comparison results of search stage with and without *MAC* limit: (a) The *MAC* comparison. (b) The *ACC* comparison.

TABLE I and II summarize the objectives (e.g. test error, parameters) of our work and other approaches on CIFAR-10 and CIFAR-100 respectively. In these two tables, the first block presents the results of models that designed by human. And the second block presents the performance of the other multi-objective NAS. As shown in Table II, Light-weight NAS can get competitive results. On CIFAR-10, our model reaches 3.30% test error with 2.3M parameters. On CIFAR-100, our model achieves 18.04% test error with only 2.4M parameters. Besides, our approach only costs 0.8 GPU-days to search the architecture better than most NAS works. Our model performance can outperform manual-design networks and most works of NAS. And the model can be applied in the resource-constrained device for its light-weight. We can get that appropriate design of search space improves both of the search efficiency and model performance, while the restriction of the *MAC* greatly reduce the model parameters.

TABLE I
COMPARISON OF CLASSIFICATION ACCURACY (%) ON CIFAR-10(C-10).

Architecture	**C-10(%)**	**Params(M)**	**Search Cost** (GPU-Days)
ResNet [7]	6.61	**1.7**	-
Wide ResNet [33]	4.17	36.5	-
DenseNet-BC [8]	3.46	25.6	-
CondenseNet-182 [9]	3.76	4.2	-
NASNet-A [13]	3.41	3.3	1800
NASNet-A+cutout [13]	**2.65**	3.3	1800
AmoebaNet-A+cutout [22]	3.34	3.2	3150
PNAS [14]	3.41	3.2	225
Block-QNN-S [24]	3.30	6.1	96
DARTS+cutout [25]	2.94	2.9	1.5
ENAS [17]	3.54	4.6	**0.5**
ENAS+cutout [17]	2.89	4.6	**0.5**
NSGA-Net [16]	2.75	3.3	4
RENA [30]	3.98	2.2	-
Light-weight NAS (ours)	3.30	2.3	0.8

TABLE II
COMPARISON OF CLASSIFICATION ACCURACY (%) ON CIFAR-100(C-100).

Architecture	**C-100(%)**	**Params(M)**
Wide ResNet [33]	22.07	11.0
CondenseNet-182 [9]	18.47	4.2
NASNet-A [13]	19.70	3.3
NASNet-A+cutout [13]	**16.58**	3.3
PNAS [14]	19.53	3.2
SMASHv1 [27]	22.07	4.6
Block-QNN-S [24]	20.65	6.1
NSGA-Net [16]	20.74	3.3
Light-weight NAS (ours)	18.04	**2.4**

V. CONCLUSION

This paper presents the Light-weight NAS, which can efficiently search for the excellent model with the fewer parameters by policy gradient. Our main ideas are incorporating *MAC* into the reward function and improving a simplified search space for light-weight model with best trade-offs between model parameters and performance. We show that the Light-weight NAS can obtain the excellent models that perform better than many existing approaches on two benchmark datasets. In the future, we will consider searching the most suitable architecture of the model with different size.

REFERENCES

[1] Y. Chen, D. Zhao, L. Lv, and Q. Zhang, "Multi-task learning for dangerous object detection in autonomous driving," *Information Sciences*, vol. 432, pp. 559–571, 2018.

[2] L. Lv, D. Zhao, and K. Shao, "Deep sparse representation-based mid-level visual elements discovery in fine-grained classification," *Soft Computing*, vol. 23, no. 18, pp. 8711–8722, 2019.

[3] D. Zhao, Y. Chen, and L. Lv, "Deep reinforcement learning with visual attention for vehicle classification," *IEEE Transactions on Cognitive and Developmental Systems*, vol. 9, no. 4, pp. 356–367, 2016.

[4] K. Shao, D. Zhao, N. Li, and Y. Zhu, "Learning battles in vizdoom via deep reinforcement learning," in *2018 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 2018, pp. 1–4.

[5] K. Shao, Y. Zhu, and D. Zhao, "Starcraft micromanagement with reinforcement learning and curriculum transfer learning," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 3, no. 1, pp. 73–84, 2018.

[6] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[7] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[8] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4700–4708.

[9] G. Huang, S. Liu, L. Van der Maaten, and K. Q. Weinberger, "Condensenet: An efficient densenet using learned group convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 2752–2761.

[10] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[11] X. Zhang, X. Zhou, M. Lin, and J. Sun, "Shufflenet: An extremely efficient convolutional neural network for mobile devices," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 6848–6856.

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

[13] B. Zoph, V. Vasudevan, J. Shlens, and 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] C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy, "Progressive neural architecture search," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 19–34.

[15] Z. Ding, Y. Chen, N. Li, and D. Zhao, "Simplified space based neural architecture search," in *The 2019 IEEE Symposium Series on Computational Intelligence*, accepted.

[16] Z. Lu, I. Whalen, V. Boddeti, Y. Dhebar, K. Deb, E. Goodman, and W. Banzhaf, "Nsga-net: neural architecture search using multi-objective genetic algorithm," in *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM, 2019, pp. 419–427.

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

[18] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary computation*, vol. 10, no. 2, pp. 99–127, 2002.

[19] R. Miikkulainen, J. Liang, E. Meyerson, A. Rawal, D. Fink, O. Francon, B. Raju, H. Shahrzad, A. Navruzyan, N. Duffy *et al.*, "Evolving deep neural networks," in *Artificial Intelligence in the Age of Neural Networks and Brain Computing*. Elsevier, 2019, pp. 293–312.

[20] K. O. Stanley, B. D. Bryant, and R. Miikkulainen, "Real-time neuroevolution in the nero video game," *IEEE transactions on evolutionary computation*, vol. 9, no. 6, pp. 653–668, 2005.

[21] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. V. Le, and A. Kurakin, "Large-scale evolution of image classifiers," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 2902–2911.

[22] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," *arXiv preprint arXiv:1802.01548*, 2018.

[23] B. Baker, O. Gupta, N. Naik, and R. Raskar, "Designing neural network architectures using reinforcement learning," *arXiv preprint arXiv:1611.02167*, 2016.

[24] Z. Zhong, Z. Yang, B. Deng, J. Yan, W. Wu, J. Shao, and C.-L. Liu, "Blockqnn: Efficient block-wise neural network architecture generation," *arXiv preprint arXiv:1808.05584*, 2018.

[25] H. Liu, K. Simonyan, and Y. Yang, "Darts: Differentiable architecture search," *arXiv preprint arXiv:1806.09055*, 2018.

[26] R. Luo, F. Tian, T. Qin, E. Chen, and T.-Y. Liu, "Neural architecture optimization," in *Advances in neural information processing systems*, 2018, pp. 7816–7827.

[27] A. Brock, T. Lim, J. M. Ritchie, and N. Weston, "Smash: one-shot model architecture search through hypernetworks," *arXiv preprint arXiv:1708.05344*, 2017.

[28] T. Elsken, J. H. Metzen, and F. Hutter, "Efficient multi-objective neural architecture search via lamarckian evolution," *arXiv preprint arXiv:1804.09081*, 2018.

[29] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, "Mnasnet: Platform-aware neural architecture search for mobile," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 2820–2828.

[30] Y. Zhou, S. Ebrahimi, S. Ö. Arık, H. Yu, H. Liu, and G. Diamos, "Resource-efficient neural architect," *arXiv preprint arXiv:1806.07912*, 2018.

[31] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, no. 3-4, pp. 229–256, 1992.

[32] Y. E. Nesterov, "A method for solving the convex programming problem with convergence rate o (1/kˆ 2)," in *Dokl. akad. nauk Sssr*, vol. 269, 1983, pp. 543–547.

[33] S. Zagoruyko and N. Komodakis, "Wide residual networks," *arXiv preprint arXiv:1605.07146*, 2016.