EFFICIENT DECOUPLED NEURAL ARCHITECTURE SEARCH
BY STRUCTURE AND OPERATION SAMPLING

Heung-Chang Lee∗† Do-Guk Kim∗†
BigData & AI Lab.
Hana Institute of Technology, Hana TI
Seoul, Korea

Bohyung Han
Dept. of Electrical and Computer Engineering
Seoul National University
Seoul, Korea

ABSTRACT
We propose a novel neural architecture search algorithm via reinforcement learning by decoupling structure and operation search. Our approach samples candidate models from the multinomial distribution over the policy vectors. The proposed technique improves the efficiency of architecture search significantly compared to the existing methods while achieving competitive classification accuracy and model compactness. Our policy vectors are easily interpretable throughout the training procedure, which allows analyzing the search progress and the identified architectures. Note that, on the contrary, the black-box characteristics of the conventional methods based on RNN controllers hamper understanding training progress in terms of policy parameter updates. Our experiments demonstrate the outstanding performance of our approach compared to the state-of-the-art techniques with a fraction of search cost.

Index Terms— Neural Architecture Search, Automated Machine Learning, Computer Vision, Deep Learning

1. INTRODUCTION
Designing deep neural network architectures often requires various task-specific domain knowledge, and it is challenging to achieve the state-of-the-art accuracy by manual tuning without the custom information. Consequently, the automatic search for network architectures becomes an active research problem. Several neural architecture search (NAS) techniques achieve the state-of-the-art performances on the standard benchmark datasets [1, 2]. However, NAS methods inherently suffer from high computational cost due to their huge search spaces and frequent validation processes. Another critical drawback of the most existing methods is that it is extremely difficult to understand their search progress since decision making typically relies on the hidden state representations.

We propose an efficient decoupled neural architecture search (EDNAS) algorithm based on reinforcement learning (RL). Contrary to the conventional RL-based NAS methods, which employ an RNN controller to sample candidate architectures from the search space, we use the policy vectors for decoupled sampling from structure and operation search spaces. The decoupled sampling strategy enables us to reduce search cost significantly and analyze the architecture search procedure in a straightforward way. The resulting architecture achieves competitive performance compared to the output models from the state-of-the-art NAS techniques.

We claim the following contributions in this paper:
• We propose an RL-based NAS technique, which learns the policy vectors to samples candidate models by decoupling structure and operation search spaces of a network.
• The proposed sampling strategy relies on the fully observable policy vectors over the two independent search spaces, which facilitates to analyze the neural architecture search progress and understand the learned models.
• Our algorithm achieves the competitive performances on various benchmark datasets including CIFAR-10 and ImageNet with a fraction of computational cost.

2. RELATED WORK
We categorize the existing NAS methods into three groups: RL-based [3, 1, 4, 5, 6, 7, 8], evolutionary algorithm (EA)-based [9, 10, 11, 12], and gradient-based [13, 14, 15, 16, 8].

RL-based architecture search techniques are originally proposed in [3], where the RNN controller is used to search for whole models. NASNet [1] follows the optimization framework of [3], but the construction of the network is based on the cells discovered by the RNN controller. The search space reduction to a cell is proven to improve not only efficiency but also accuracy compared to the unrestricted exploration [1, 17, 4]. ENAS [4] aims to further improve efficiency by weight sharing. It achieves a remarkable reduction in search cost compared to NASNet with competitive accuracy.

The most representative work of EA-based approach is [9], where a CNN is evolved from a trivially simple architecture. This technique is extended to evolving the convolu-
EDNAS is an RL-based NAS approach with reduced complexity and observable search mechanism while maintaining accuracy in target tasks. Although it is straightforward to apply the proposed technique to RNNs, we focus on search for CNNs in this paper.

### 3. METHODOLOGY

EDNAS finds the neural networks given by a stack of cells as in [1, 4]. There are two sorts of cells: normal and reduction cells. A normal cell has the same input and output sizes while the reduction cell has an output feature map with a half of input width and height. By controlling the number of normal and reduction cells, EDNAS adjusts the size of the whole network. The architecture of a cell is defined by a directed acyclic graph (DAG), where a node and an edge represent the local and reduction cells, EDNAS adjusts the size of the whole network structure with differentiable weights [13], encoding network architectures using feature vectors in a continuous space [14], and adopting the concrete distribution over network architectures [16].

#### 3.1. Overview

EDNAS finds the neural networks given by a stack of cells as in [1, 4]. There are two sorts of cells: normal and reduction cells. A normal cell has the same input and output sizes while the reduction cell has an output feature map with a half of input width and height. By controlling the number of normal and reduction cells, EDNAS adjusts the size of the whole network. The architecture of a cell is defined by a directed acyclic graph (DAG), where a node and an edge represent the local and reduction cells, EDNAS adjusts the size of the whole network.

The structure search is given by identifying a subgraph of the initial graph, which is a complete DAG. We sample a candidate architecture and a set of operations for validation, where the sampling is controlled by the policy vector updated based on the feedback from the validation results. A convolutional cell has two input nodes, which have no incoming edges in the DAG and correspond to the outputs of two previous cells. Each node takes the outputs of two preceding nodes as its inputs, and applies operations to individual inputs.

We consider seven operations, which include $3 \times 3$ and $5 \times 5$ separable convolutions, $3 \times 3$ and $5 \times 5$ dilated separable convolutions, $3 \times 3$ max pooling, $3 \times 3$ average pooling, and the identity function. The output of a convolutional cell is given by the depth-wise concatenation of the feature maps in all hidden nodes.

### 3.2. Architecture Search Process in EDNAS

The objective of our algorithm, EDNAS, is to maximize the expected reward of sampled architectures, which is given by

$$\max_{\theta} \mathbb{E}_{P(m|\theta)}[R],$$

where $\theta$ is the policy parameter and $m$ is the sample model based on the current policy. While the RNN controller manages policies in the conventional RL-based methods ($\theta = \theta_c$), EDNAS employs the policy vectors ($\theta = \theta_e$) to search for the optimal architecture.

EDNAS decouples the structure and operation search, which are performed based on separate policy vectors. There are two kinds of policy vectors in EDNAS; one is for non-input nodes and the other is for edges. All non-input nodes in the DAG are associated with a $c_i$-dimensional policy vector, where $c_i$ is the number of incoming edge combinations to the $i$-th node. The policy vector of node $n_i$, $p_{n_i}$, is given by

$$p_{n_i} = [e_1, ..., e_{c_i}]^T, \quad c_i = \left(\frac{e_i}{r}\right),$$

where $e_i$ is the number of incoming edges to $n_i$, and $r$ is the number of the selected edges. The policy vector of edge $e$ is a $k$-dimensional vector, where $k$ is the number of operations:

$$p_e = [w_1, ..., w_k]^T.$$
Based on these policy vectors, we perform architecture sampling as follows. First, we search for the overall structure of the network by sampling edges from the DAG. To this end, the softmax function is applied to $p_{n_i}$ for its normalization. Then, an input edge combination of each node is sampled from the multinomial distribution given by $\text{softmax}(p_{n_i})$. After that, we optimize the operation corresponding to each selected edge. The operation of each selected edge is determined by drawing a sample from the multinomial distribution defined by $\text{softmax}(p_v)$, which is similar to the structure search step.

The policy vectors for the structure and operation search are observable in our framework. Therefore, we can analyze the training progress of EDNAS based on the statistics of architecture samples and the visualized policy vectors during training. For example, it is possible to see which combinations of edges are selected and which operations are preferred at each iteration or over many epochs.

Figure 1 illustrates an example of convolutional cell architecture sampling. There are two input nodes in the DAG and non-input nodes in our convolutional cell receive two inputs from preceding nodes, i.e., $r = 2$. For each non-input node, the feature maps resulting from the operations derive the output of the node by the element-wise summation. In the structure search step, an input edge combination is selected for each non-input node using the multinomial distribution of policy vector. In the example, the edges heading to $n_3$ from $n_0$ and $n_2$ are selected as the input edge combination of $n_3$.

In operation search step, a specific operation for a selected edge is determined. We obtain a sampled architecture after both steps are completed, as shown in the rightmost graph in Figure 1. We search for normal and reduction cells based on the policy vectors defined separately.

### 3.3. Training Process and Deriving Architectures

Training EDNAS consists of a two-step process: child model training and policy vector training. We alternate the two steps, where the child model is learned using the training data while the policy vectors are trained with the validation set.

For training the child model, we use stochastic gradient descent (SGD) on the shared parameters of the DAG. During the child model training step, the parameters of the policy vectors are fixed. We sample an architecture in each iteration, compute a gradient, and update the model using SGD.

In the policy vector training step, the model parameters are fixed and the policy vectors are updated to maximize the expected reward. We use Adam optimizer [20], and the gradient is computed by REINFORCE [21], as shown below:

$$\nabla_{\theta_v} \log P(m; \theta_v)(R - b),$$

where $P(m; \theta_v)$ is the probability of the model $m$ sampled based on the policy vectors $\theta_v$, and $b$ is a moving average of rewards. We compute the reward $R$ based on the validation accuracy, which encourages the policy vector to sample an architecture with high generalization performance.

The training process is repeated until the training epoch reaches the maximum number. When deriving the final architecture, we sample a predefined number of models—100 in our experiment—and compute their rewards on a single minibatch. Then, the model that receives the best reward is selected as the final architecture discovered by EDNAS.

### 4. EXPERIMENTS

We tested the performance of our algorithm on CIFAR-10 and ImageNet. Refer to our project page\footnote{https://github.com/logue311/EDNAS} for the source code, implementation details, experiment setup, and full results.

#### 4.1. Cell Search with CIFAR-10

Table 1 summarizes the results. Although the manual search techniques achieve the state-of-the-art accuracy, their model sizes are much larger than the ones given by automatic search methods. The NAS methods without weight sharing suffer from huge search cost while the approaches with weight sharing have a reasonable balance between cost and accuracy. Note that EDNAS achieves competitive accuracy to the techniques with weight sharing in terms of accuracy and model size, but it is substantially faster than other architecture search methods.

To further analyze the search procedure, we present the statistics of the sampled architectures. Specifically, we illustrate the cumulative distributions of the sampled edges and the

| Method | Search Cost (GPU days) | Params (M) | Test Error (%) |
|--------|------------------------|------------|----------------|
| DenseNet [18] | - | 25.6 | 3.46 |
| DenseNet + cutout [18] | - | 26.2 | 2.56 |
| NASNet-A + cutout [1] | 1800 | 3.3 | 2.65 |
| AmoebaNet-A + cutout [10] | 3150 | 3.2 | 3.34 |
| AmoebaNet-B + cutout [10] | 3150 | 3.1 | 2.55 |
| PNAS [6] | 225 | 3.1 | 3.41 |
| NAONet + cutout [14] | 200 | 128 | 2.11 |
| ENAS + cutout [4] | 0.5 | 4.6 | 2.89 |
| ENAS + cutout\† [4] | 0.6 | 3.2 | 3.32 |
| DARTS (first) + cutout [13] | 0.38 | 2.9 | 2.94 |
| DARTS (first) + cutout\† [13] | 0.32 | 2.8 | 3.05 |
| DARTS (second) + cutout [13] | 1 | 3.4 | 2.83 |
| NAONet-WS [14] | 0.3 | 2.5 | 3.53 |
| GHN + cutout [19] | 0.84 | 5.7 | 2.84 |
| DSO-NAS + cutout [15] | 1 | 3.0 | 2.84 |
| Random | 0.27 | 3.4 | 3.91 |
| EDNAS + cutout | 0.28 | 3.7 | 2.84 |
operations in the normal cells over every 50 epoch. Figure 2(a) shows that $e_{0,3}$ and $e_{1,3}$ are selected more frequently at the later stage of training while the sampling ratio of $e_{2,3}$ drops consistently over time. In general, the edges from input nodes are preferred to the ones from the hidden nodes. On the other hand, when we observe the operation sampling patterns in the normal cell, we can see the clear tendency of individual operations; the frequency of pooling (max, avg) and identity (id) operations decreases gradually while the separable convolutions and dilated convolutions with a relatively large kernel size ($e.g., 5 \times 5$) are sampled more frequently at the later stage of the optimization process. It implies that the searched models attempt to extract the high-level information from the inputs to improve accuracy.

Figure 3 illustrates the derived architectures. The characteristics of the two models are different in the sense that the normal cell has many parallel operations, which coincides with the tendency illustrated in Figure 2(a) while the operations in the reduction cell tend to be serial.

4.2. Cell Search with ImageNet

Table 2 presents the performance of EDNAS on ImageNet dataset in comparison to other methods. Most of the existing approaches identify the best architecture on CIFAR-10 dataset and use the same model for the evaluation on ImageNet after fine-tuning. This is mainly because their search costs are prohibitively high and it is almost impossible to apply their algorithms to the large-scale datasets directly. On the contrary, EDNAS, and DSO-NAS [15] as well, are fast enough to explore the search space directly even on the ImageNet dataset. The performance of EDNAS is as competitive as DSO-NAS in terms of model size and accuracy, but the search cost of EDNAS is substantially smaller than DSO-NAS.

5. CONCLUSION

We presented a novel neural architecture search algorithm, referred to as EDNAS, which decouples the structure and operation search by applying the separate policy vectors. Since the policy vectors are fully observable, the architecture search process is more interpretable in EDNAS. The experimental results demonstrate that the proposed algorithm is competitive in terms of accuracy and model size even with significantly faster learning speed.
6. REFERENCES

[1] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc 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.

[2] Liang-Chieh Chen, Maxwell Collins, Yukun Zhu, George Papandreou, Barret Zoph, Florian Schroff, Hartwig Adam, and Jon Shlens, “Searching for efficient multi-scale architectures for dense image prediction,” in Advances in Neural Information Processing Systems, 2018, pp. 8713–8724.

[3] Barret Zoph and Quoc V Le, “Neural architecture search with reinforcement learning,” Proceedings of the International Conference on Learning Representations, 2017.

[4] Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean, “Efficient neural architecture search via parameter sharing,” International Conference on Machine Learning, 2018.

[5] Zhao Zhong, Junjie Yan, Wei Wu, Jing Shao, and Cheng-Lin Liu, “Practical block-wise neural network architecture generation,” in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018.

[6] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy, “Progressive neural architecture search,” in Proceedings of the European Conference on Computer Vision (ECCV), 2018, pp. 19–34.

[7] Juan-Manuel Pérez-Rúa, Moez Baccouche, and Stephane Pateux, “Efficient progressive neural architecture search,” British Machine Vision Conference, 2018.

[8] Han Cai, Ligeng Zhu, and Song Han, “Proxylessnas: Direct neural architecture search on target task and hardware,” International Conference on Learning Representations, 2019.

[9] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc Le, and Alex Kurakin, “Large-scale evolution of image classifiers,” International Conference on Machine Learning, 2017.

[10] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le, “Regularized evolution for image classifier architecture search,” in Proceedings of the AAAI Conference on Artificial Intelligence, 2019, vol. 33, pp. 4780–4789.

[11] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter, “Efficient multi-objective neural architecture search via lamarckian evolution,” International Conference on Learning Representations, 2019.

[12] David So, Chen Liang, and Quoc Le, “The evolved transformer,” International Conference on Machine Learning, 2019.

[13] Hanxiao Liu, Karen Simonyan, and Yiming Yang, “Darts: Differentiable architecture search,” International Conference on Learning Representations, 2019.

[14] Renqian Luo, Fei Tian, Tao Qin, Enhong Chen, and Tie-Yan Liu, “Neural architecture optimization,” in Advances in Neural Information Processing Systems, 2018.

[15] Xinbang Zhang, Zehao Huang, and Naiyan Wang, “You only search once: Single shot neural architecture search via direct sparse optimization,” arXiv preprint arXiv:1811.01567, 2018.

[16] Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin, “Snas: stochastic neural architecture search,” International Conference on Learning Representations, 2019.

[17] Gabriel Bender, Pieter-Jan Kindermans, Barret Zoph, Vijay Vasudevan, and Quoc Le, “Understanding and simplifying one-shot architecture search,” in International Conference on Machine Learning, 2018, pp. 549–558.

[18] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger, “Densely connected convolutional networks,” in 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). IEEE, 2017, pp. 2261–2269.

[19] Chris Zhang, Mengye Ren, and Raquel Urtasun, “Graph hypernetworks for neural architecture search,” International Conference on Learning Representations, 2019.

[20] Diederik P Kingma and Jimmy Ba, “Adam: A method for stochastic optimization,” arXiv preprint arXiv:1412.6980, 2014.

[21] Ronald J Williams, “Simple statistical gradient-following algorithms for connectionist reinforcement learning,” Machine learning, vol. 8, no. 3-4, pp. 229–256, 1992.

[22] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich, “Going deeper with convolutions,” in Proceedings of the IEEE conference on computer vision and pattern recognition, 2015, pp. 1–9.

[23] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam, “Mobilenets: Efficient convolutional neural networks for mobile vision applications,” arXiv preprint arXiv:1704.04861, 2017.