CNASV: A Convolutional Neural Architecture Search-Train Prototype for Computer Vision Task

Tianbao Zhou(✉), Yu Weng, and Guosheng Yang

College of Information Engineering, Minzu University of China, Beijing 100081, China
tianbaochou@163.com

Abstract. Neural Architecture Search (NAS) has become more and more prevalent in the field of deep learning in the past two years. Existing works often focus on image classification, and few works recently extend NAS to another computer vision task, such as semantic image segmentation. The semantic image segmentation is essentially a dense prediction for each pixel on whole image. Therefore, we choose the same basic primitive operations to build the search space for the two computer vision task respectively. Searching good neural network architectures and then training them from scratch is a regular procedure for NAS. In this paper, we design a prototype system that deploy search module and train module to collaborate with each other. Follow the former research, we initialize over-parameterized cells architecture and then transform to the continuous relaxation of the architecture to derive the good subnetwork by gradient descent. Our system can support any differential search algorithm, such as one-shot, DARTS or ProxylessNAS. We illustrate the effectiveness of our chosen primitive operations in the image classification and ability to transfer these operations to build search space for semantic image segmentation.

Keywords: CNASV · Image classification · Image segmentation

1 Introduction

How to automatically design a good neural network architecture for dataset on hand? We may fit the pre-trained network model to custom dataset which denoted transfer learning. However, a more nature way is to customize a network architecture to the dataset from different fields. Neural architecture search (NAS), a subfield of AutoML, has proposed to solve this problem. Image classification is the start point for NAS to show it's power on searching neural network architectures exceed human-designed architecture. NAS can be categorized three components: search space, search algorithm and the model evaluation [9]. The search space defines what architectures can be found during the search process. Incorporating prior knowledge about properties well-suited for task can reduce

X. Wang et al. (Eds.): CollaborateCom 2019, LNICST 292, pp. 380–393, 2019.
https://doi.org/10.1007/978-3-030-30146-0_26

the size of the search space and simplify the search. For example, in image classification, the search space including the selection of primitive operations at each search step and the prior backbone architecture used to define outer network.

The search strategy details how to explore the search space. The objective of NAS is typically to find architectures that have high evaluated performance on unseen data (e.g. split training datasets into training and validation, and search architecture on training but evaluated by validation) [9]. Recent works have introduced many effective search algorithm, including reinforcement learning (RL) [2,5,20,32,33], evolutionary algorithms [22,28–31], Bayesian optimization [14,15], and gradient based algorithms [11,21,27].

Reinforcement learning methods usually encode whole network architecture as RNN sequence or cell architecture and update RNN weights to derive next candidate architecture by Q-Learning or another update strategy. Besides the RL algorithm, evolutionary algorithms represented by the genetic algorithm (GA) describe the architecture as a gene string, and then perform crossover and mutation. Each string represents a network architecture, and by training and putting it on the validation set, those with good evaluation results are more likely to be retained. Bayesian optimization is a good method to optimize the parameters of the machine learning algorithm model. In recent years, some scholars have used it to speed up the evaluation of the performance of currently searched which improving the search progress. The previous methods are all based on the discrete search space. Recently, Liu et al. extend the works of Grathwohl et al. [11,27] by proposing a continuous relaxation of the search space to enable gradient-based optimization [21], denoted DARTS. The authors initialize an over-parameterized network, similar to densenet [13] but replace fix a single operation o_i (e.g. convolution) to calculated as a specific layer with mixing N operation from a set of operations $o_1, o_2, \cdots o_N$. More specifically, given a layer input x, the layer output y is computed $y = MixO(x) = \sum_{i=1}^{N} w_i o_i(x), w_{i \geq 0}$, where the w_i indicates how important does o_i contribute to the layer output. Cai et al. [6] propose ProxylessNAS which introduces a binary gate to reduce the N candidate path to two at each update.

Model evaluation is a vital step to tell the search algorithm whether current candidate architecture is good or not and decide whether to keep it at next update. The traditional way to evaluate a network performance is train from scratch, but cost too much time. Model performance prediction is a natural idea to speed up the process of searching network architecture. Klein et al. design a Bayesian Neural Network to predict the learning curve of the network searched and early terminated the worse network if the curve is bad [17]. Baker et al. use an additional hand-designed features on the basis of Klein et al. to predict the learning curve in the v-SVR (Sequential regression model) [3]. Liu et al. choose a LSTM network as the surrogate predictor. Each time the network predicts the performance of the model, it selects the k best performance network architectures and train them from scratch to get the real performance, and then updates the surrogate predictor parameters [19]. Peephole encodes the layer of the network architecture into vectors and put it into a LSTM surrogate function.

This function can predict performance based only on the previous network architecture, without additional training [8].

Since Zoph et al. proposed the RL-based NAS method obtained competitive performance on the CIFAR-10 and Penn Treebank benchmarks. Many researchers have focused on improving search methods to speed up the search process. As we mentioned before, lots of search algorithms have successfully reduce the search time from 800 GPUs for three to four weeks to 1 GPUs several days even one days. Weight-sharing is a major factor to accelerate the entire search procedure for these search algorithms. In DARTS, One-Shot and ProxylessNAS the mixed operation is actually a weight sharing trick, which avoids retraining the currently generated network architecture by sharing the sub-network architecture's weights in the search process.

Image classification lays a good foundation for the development of NAS. A logical next step is extending to another computer vision task, such as semantic image segmentation and object detection. A few work recently applied NAS to image segmentation. Chen et al. [7] first introduce NAS to solve image segmentation. The authors show that even with random search on constructing a recursive search space, the architecture search outperforms human-invented architectures and achieves better performance on many segmentation datasets. However, this work does not use one-shot searching, which focused on search a small Atrous Spatial Pyramid Pooling (ASPP) module called DPC (similar as decoder) and fix the pre-trained backbone (modified Xception) as encoder. Liu et al. [18] propose Auto-DeepLab: a general-purpose network level search space, and jointly search across two-level hierarchy (network level and cell level architecture). The authors indicate that search space includes various existing designs such as DeepLabv3, Conv-Deconv and Stacked Hourglass. However, the search space of Auto-DeepLab does not include U-Like architectures (eg. U-net), which are the most famous architectures in the field of medical image segmentation.

In this paper, we attempt to find a set of primitive operations for computer vision problems, such as image classification and image segmentation, and then we build two different search space for image classification and semantic image segmentation. After that, we design a prototype of search-train system using NAS for image classification and semantic image classification. Our system supports any type of differential architecture search algorithms to search on our search space. In summary, our contributions are as follows:

1. We build two different search space for image classification and semantic image segmentation.
2. We design a prototype of search-train system for automatically search good architecture on specific dataset and train that model.
3. We speed up the forward passing of cell-based architecture search on our system by parallel non-topology order nodes in our cell architecture.

2 The NAS Methods

2.1 Search Space

CNN Architecture Representation. A directed acyclic graph (DAG) is used to represent the network topology architecture, in which each node h_i represents input image or a feature map and each edge e_{ij} is associated with an operation (e.g. convolution operation, a pooling operation and a skip connection) between node h_i and node h_j. When the generation method of the DAG is unrestricted, its network architecture space will be very large, which will bring great challenges to the present search algorithms. Therefore, we use cell-based architecture [33]. When determining the best cell architecture, we can stack the cells into a deeper network on the backbone network (we will describe below). In other words, the architecture of cell is shared by entire network.

Primitive Operation Sets. How to choose suitable primitive operations? We have investigated the popular CNN architecture and the former NAS that has great success on image classification, and choose the primitive operations as Fig. 1 shown.

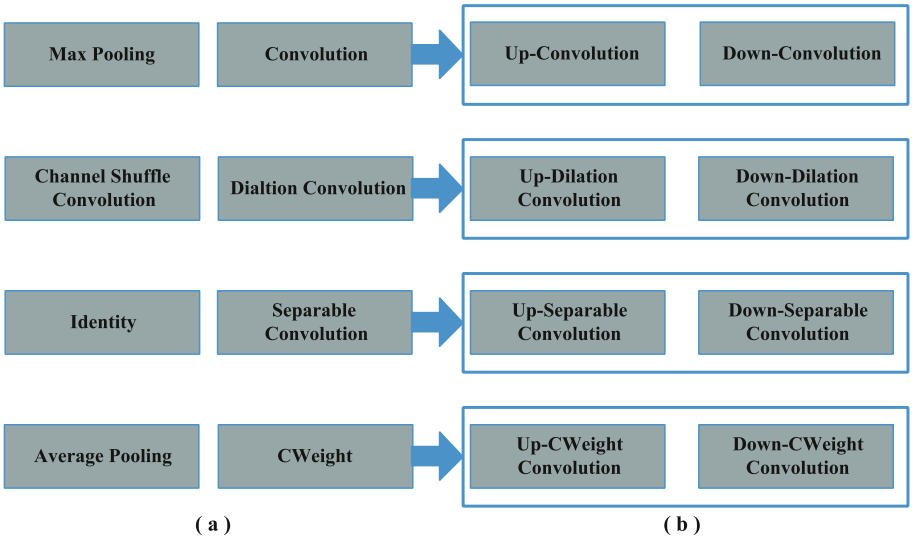

Fig. 1. (a) The basic primitive operations for image classification and image segmentation (b) the up and down operations derived from four basic primitive operations. CWeight operation indicates squeeze-and-excitation operation [12].

We can see from Fig. 1(b) that when the sliding step (stride value) greater than 1, the convolution operation can halve the dimension of feature map or double the dimension (we simply set stride as 2), the former denoted 'Down'-Convolution and the later called 'Up'-Convolution. This indicates that the down

operation and up operation can be derived from the same base operation. In contrast, different from the primitive operations in image classification, the 'up'-version of some operations make no sense (e.g. the identity operation) and the 'up'-version of pooling operations (e.g. the average pooling and max pooling) do not exist. In our work, based on these primitive operations, we design three types of primitive operation set: Normal POs, Down POs and Up POs. In accordance with it, the three types of cell-based over-parameterized architecture are formed. As shown in the Fig. 2, the Normal POs and Down POs form NormalC and DownC, Up POs constitute UpC. All the operations is 3×3.

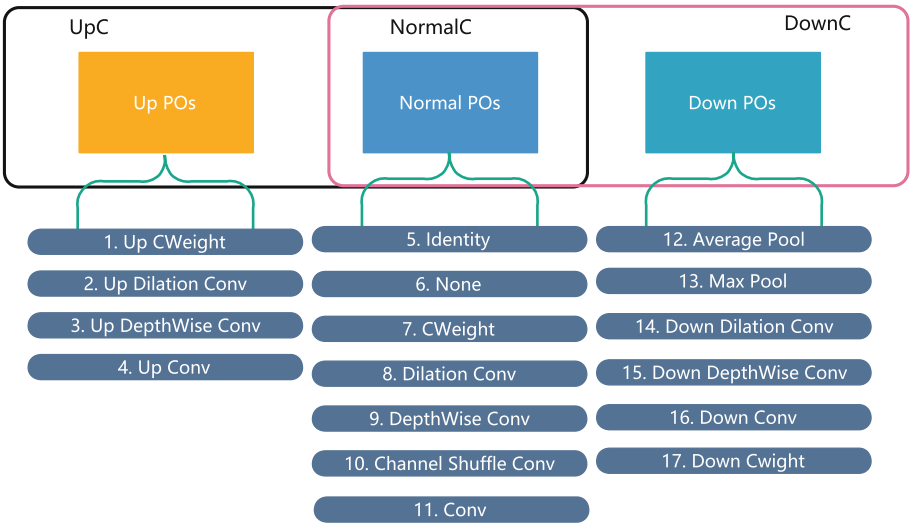

Fig. 2. The three types of primitive operation sets and in accordance with it, the three types of cell-based over-parameterized architecture. NormalC indicates the cell outputs the same size of input feature maps. DownC represents the cell halves the dimension of inputs, but UpC doubles.

Backbone Network. For image classification, We follow Zoph et al. [32] that define a minimum architecture called cell (as shown in Fig. 3(a)), which has two input nodes: the input of the k^{th} cell, denoted $cell_k$, comes from the output of the cell $k-1$ and $k-2$. During searching, we stack the cells into shadow network, but when finish searching, we stack more cells into deep network. Inspired by the success of encoder-decoder network in semantic image segmentation, we use an encoder-decoder architecture as our backbone (Fig. 3(b)) for semantic image segmentation. Different from the image classification, the input of the k^{th} cell either comes from the output of the cell $k-1$ (FCN-like networks) or $k-1$ and $k-2$ in the encoder parts, but the decoder parts are $L-k+1$, where $L = \#DownC + \#UpC$ is total number of cells (U-Like networks).

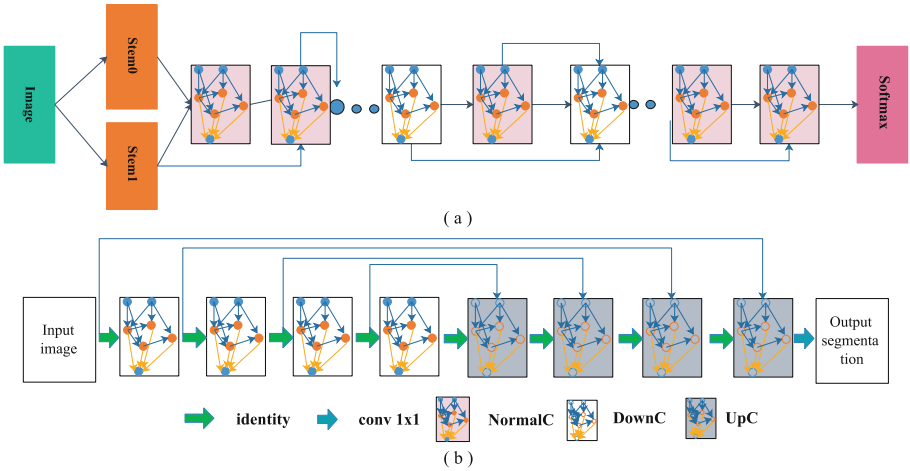

Fig. 3. (a) The backbone network used for image classification. The Stem0 and Stem1 is a Conv-ReLU-BN with stride 2 for reduce the image dimensation (b) The backbone network used for semantic image segmentation.

2.2 Search Algorithm

In this section, we frist describe the way to construct an over-parameterized network [4,6,18,21]. After that we introduce two differential architecture search method into our work: DARTS and ProxylessNAS, and describe the smility and difference between them below.

Over-Parameterized Cell Architecture. Given a cell architecture $C(e_1, \cdots, e_E)$ where e_i represents a certain edge in the DAG. Let $O = o_i$ be a set of operations in the above with N candidate operations. Instead of setting each edge associates with definite operation, we set each edge to be a mixed operation that has N parallel paths (As shown in Fig. 4(a), the green arrows indicate the output of cell which simply the concatenation of the blocks' output tensors $\sum_{i=1}^{M} h_i$, where M is the number of intermediate Nodes), denoted as $MixO$. Therefore, the over-parameterized cell architecture can be expressed as $C(e_1 = MixO_1, \cdots, e_E = MixO_E)$. The output of a mixed operation $MixO$ is defined based on the output of its N paths:

$$MixO(x) = \sum_{(i=1)}^{N} w_i o_i(x). \tag{1}$$

As shown in Eq. 1, w_i represents the weight of o_i, in One-Shot [4] is constant value 1, but in DARTS [21] is calculated by applying softmax to N real-valued architecture parameters $\{\alpha_i\} : e^{\alpha_i} / \sum_j e^{\alpha_j}$. The initial value of α_i is $1/N$.

In the above, the ouput feature maps of all N paths is calculated when all operations are loaded into GPU memory. Because the output of each edge is

a mixed operation for N candidate primitive operations. As such, [21] and [4] roughly need N times GPU memory compared to training a compact model (the model stacked by searched cells). However, training the compact model only use one path.

Cai et al. use binary gate for learning binarized path instead of N paths [6]. The difference between DARTS and binary gate method (denoted Proxyless-NAS) is that the former update all of the architecture parameters by gradient descent at each step, but the latter only update one of them. On one hand, when updating network weight parameters, we need first fix the architecture parameters and randomly sample a binary gate for each batch of input data. Then the weight parameters of active paths are updated via standard gradients descent on the training dataset. On the other hand, when training architecture parameters, the network weight parameters are frozen, then we reset the binary gates and update the architecture parameters on the validation set (the details see the paper). These two update steps are performed in an alternative manner. Once the training of architecture parameters is done, we need derive the our cell-based architecture by pruning redundant paths. In this work, we simply choose the path with the k ($k = 2$, for our works) highest path weight (Fig. 4(b)). In this way, the memory requirement is reduced to the same level of training a compact model. Since only considers two paths for updating at each update step, the trained-level of operation not on current two paths being much lower than the operation on (it is unfair to compare the contribution of a well-trained operation and another insufficiently trained operation to the output.). Therefore, we need more iterations for updating util all of the operations is well-trained and a extra time will cost at moving feature map not in GPU memory to GPU.

2.3 Parallel the Operations Calculation

As we mentioned before, a cell is a DAG consisting of an ordered sequence of M nodes. Therefore, the $Node_i$ always be produced before $Node_j$. However, the $Node_i$ and $Node_j$ may not exist data correlation ($Node_j$ can only be created after $Node_i$ has done). It means we can parallel output $Node_i$ and $Node_j$. For example, in Fig. 4(b), $Node_1$ and $Node_2$ has not data correlation, and we can produce $Node_1$ and $Node_2$ simultaneously. In our implementation, before training a network architecture searched, we first seperate each topology-path and parallel compute them between cells. On this way, we can accelerate the efficiency of our network.

3 A Prototype of Search-Train System

In this section, We design a search-training prototype for image classification and semantic image segmentation (which can also be used to other computer vision tasks, such as object detection). It is useful when user need find a good network architecture for new image dataset without any high-performance equipment at hand and it is easy to use. Combing with binary gate search algorithm [6]

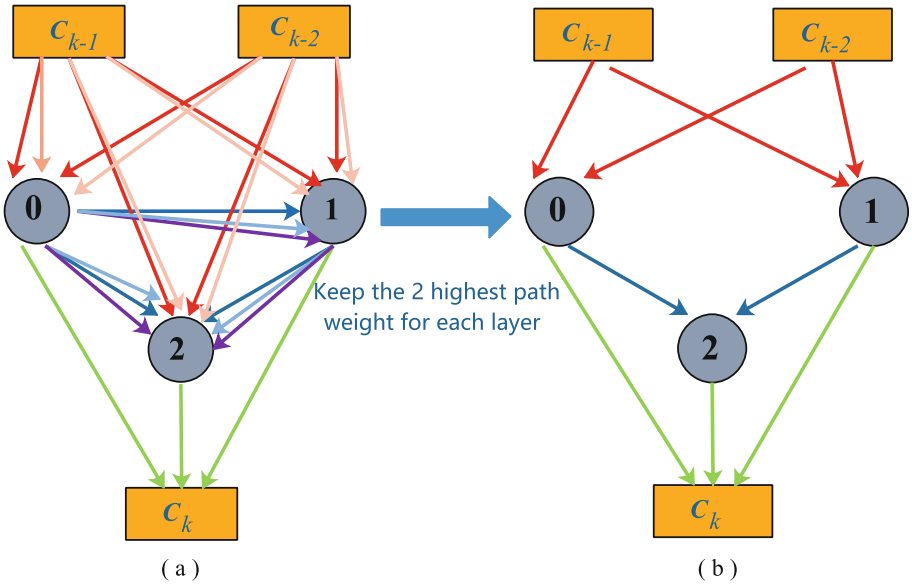

Fig. 4. (a) The over-parameterized architecture (b) Choose the path with the two highest path weight.

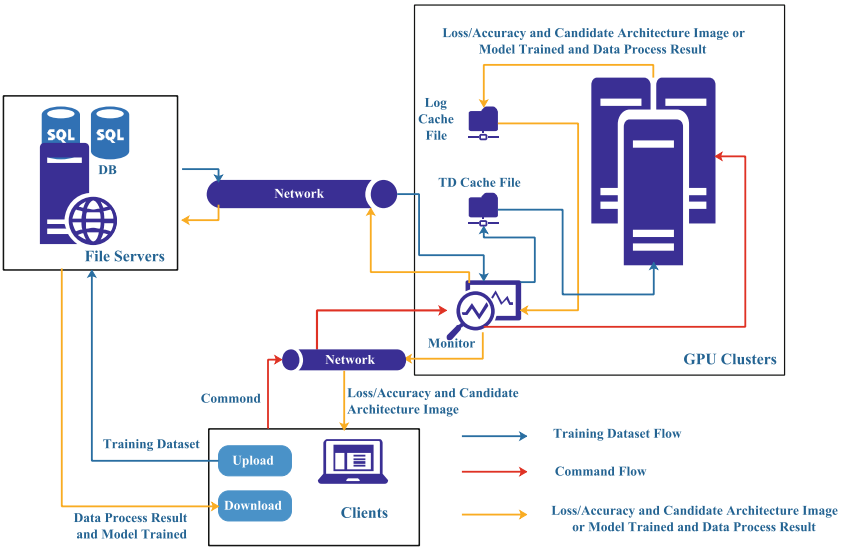

Fig. 5. A prototype of search-train system, notice that the Clients can be web clients or desktop clients (such as Qt clients).

and our prototype of search-train system, we can build a one-shot search-train system for all of the computer vision and deploy it in distributed cloud (Fig. 5).

The Search-Train system is composed of Clients, File Servers and GPU Clusters. The Clients can be web clients or desktop clients. Its main role is to upload the training dataset to the File servers, observe the search states or training states (such as loss, accuracy and candidate architecture), and interact with the server to control the search or training schedule and download both the final trained-model and prediction results. The File servers can be any one of popular distributed file system, such as Hadoop [1], FastDFS [25] and OpenAFS [23]. The GPU Clusters have four components: Log Cache File, TD Cache File, Monitor and GPUs. Both of Log Cache File and TD Cache File is on the CPU memory implemented by the queue. Log Cache File store search or training states. For example, the train/validation loss, performance and current best cell architecture. TD Cache File implement asynchronous loading of training data set for the NAS procedure.

Monitor plays a coordinator role, which maintains the status of Log Cache File and TD Cache File. After the NAS procedure reads a batch size of training data from TD Cache File, the monitor put the next batch size of training data from File servers to TD Cache File for next load. On the other hand, when the training is done, the monitor will send results and trained-model to File servers and notifies Clients to download. Similarly, the monitor gets the data from Log Cache File and send it to Clients. In addition, the Monitor also manages the status of NAS procedure, including new a NAS procedure to waiting queue, close the error or stopped NAS procedure to recycle the GPU resources and assign free GPUs to the top of waiting NAS procedure. If we use DARTS update strategy, we need clearly two steps, one for searching best cell architectures another for training network stacked by cells searched. However, when we use ProxylessNAS update strategy, the two steps can be merged into one step.

We implement our NAS procedure in Pytorch [24] and use ring-allreduce technology [26] (e.g. Horovod) to distribute searching and training models.

4 Experiments

In this section, we will first describe the details of implementing search process on image classification and semantic image segmentation. After that, we will show the architecture DAG we searched and the performance after training on some image datasets. Note that in our experiments, we follow the DARTS algorithm to search our cell architecture.

4.1 Implement Searching Architecture

Image Classification. We search the network architecture searched on CIFAR-10 dataset. We keep half of the training data as the validation set, and a small network obtained by stacking 6 cells is trained for 50 epochs with batch size 64, and we use the classification error rate on the validation set as the performance

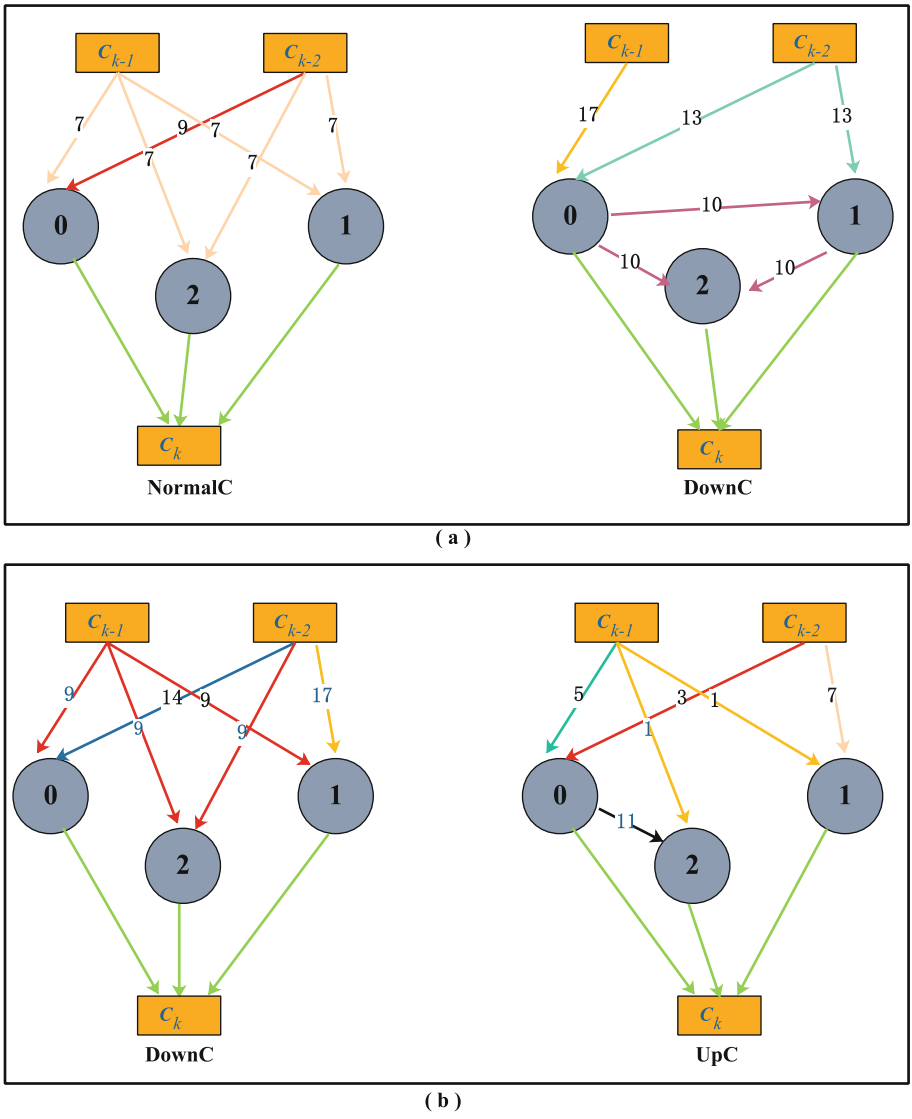

Fig. 6. (a) The Down Cell and Normal Cell architectures searched on CIFAR 10 for image classification (b) The Down Cell anb Up Cell architectures on PASCAL VOC2012 for semantic image segmentation. The operation represented by the specific Node numbering can be found in Fig. 2

of the cells we searched. Cells located at 1/3 and 2/3 of the total depth of the network are Down Cells.

Semantic Image Segmentation. We search our image segmentation network architecture on PASCAL VOC 2012 dataset [10]. Similiar to Image Classification,

Table 1. Comparison with Darts on CIFAR-10

Model	Model size (million)	Accuracy	Evaluation time
Darts	3.16	4.74	18 h
Ours	**2.12**	**4.96**	9 h

We randomly keep half of the training data as the validation set, and stack 3 DownC and UpC into U-like backbone. When we use DARTS search strategy, the batch size is 2 and the architecture search optimization is conducted for a total of 120 epochs. A batch size can be 8 when we use binary gate update strategy, but a much 200 epochs is needed.

When learning network weight w, we follow DARTS use SGD optimizer with momentum 0.95, cosine learning rate that decays from 0.025 to 0.01, and weight decay 0.0003 [21]. When learning the architecture, we use Adam optimizer [16] with learning rate 0.0003 and weight decay 0.0001. The cell architectures are show in Fig. 6. We can that the node 0 and node 1 in Normal Cell can be calculated in parallel (Fig. 6(a)). All of intermediate nodes in Down Cell or node 0 and node 1 in Up Cell also can be calculated simultaneously (Fig. 6(b)).

4.2 Performance on Some Datasets

We evaluate our two network architectures on CIFAR10 and Camvid dataset. The error rate is selected as the evaluation metric for image classification and Mean Intersection over Union (mIoU) for semantic image segmentation. We will describe the train details below.

Evaluate on CIFAR10 Dataset

We use a large network of 20 cells for training over 200 epochs with a batch size of 64. Other hyperparameters remain the same as the ones used for the architecture search, similar to Darts. As the Table 1 shown, our performance of our classification network is comparable to DARTS but with halves parameters size and much more efficient.

Evaluate on CamVid Dataset

From the Table 2, we can see that the origin U-net has a much bad performance in Camvid dataset but with larger parameters. Which reveals that the more network parameters may not improve network performance. Our network achieve a comparable performance with FC-DenseNet103 but a much less parameters and 3 time faster owing to our parallel computation technology. It is worth to noticing that the FC-DenseNets cost over 2 times GPU memory than U-Net and Ours network.

Table 2. Results on CamVid dataset.

Model	Model size (million)	mIoU	Evaluation time
U-Net	31.4	54.3	15 h
FC-Densenet56	1.5	58.9	2 day-12 h
FC-Densenet67	3.5	65.8	2 day-21 h
FC-Densenet103	9.4	**66.9**	3 day-10 h
Ours	**1.09**	66.1	18 h

5 Conclusion

In this paper, we select eight basic primitive operations for both image classification and image segmentation. Moreover, we create three types of primitive operation set base on them. We search our cell-based architectures on different backbone networks for image classification and image segmentation respectively. To implement our experiments, we design a prototype called CNASV for search good architectures and train them in a shot. Owing to our parallel calculation of operations cell architectures, our networks are more efficient than other comparison networks. In the future, we will integrated binary gate for allowing use more batch size to accelerate prune over-parameterized network and improve each module in our system.

References

1. Apache Hadoop. https://hadoop.apache.org/
2. Baker, B., Gupta, O., Naik, N., Raskar, R.: Designing neural network architectures using reinforcement learning. In: International Conference on Learning Representations (ICLR), pp. 1–18 (2017). https://doi.org/10.1080/0305215042000274942. http://arxiv.org/abs/1611.02167
3. Baker, B., Gupta, O., Raskar, R., Naik, N.: Accelerating neural architecture search using performance prediction. In: ICLR Workshop, vol. 2, pp. 1–7 (2018). http://metalearning.ml/papers/metalearn17_baker.pdf
4. Bender, G., Kindermans, P.J., Zoph, B., Vasudevan, V., Le, Q.: Understanding and simplifying one-shot architecture search. In: 35th International Conference on Machine Learning (ICML), vol. 80, pp. 549–558 (2018). https://doi.org/10.1109/TDEI.2009.5211872. http://proceedings.mlr.press/v80/bender18a.html
5. Cai, H., Chen, T., Zhang, W., Yu, Y., Wang, J.: Efficient architecture search by network transformation. In: AAAI (2018). http://arxiv.org/abs/1707.04873
6. Cai, H., Zhu, L., Han, S.: ProxylessNAS: direct neural architecture search on target task and hardware. In: 2019 International Conference on Learning Representations (ICLR) (2019)
7. Chen, L.C., et al.: Searching for efficient multi-scale architectures for dense image prediction. In: NeurIPS (2018)
8. Deng, B., Lin, D., Yan, J., Lin, D.: Peephole: predicting network performance before training (2017). http://arxiv.org/abs/1712.03351

9. Elsken, T., Metzen, J.H., Hutter, F.: Neural architecture search: a survey. CoRR abs/1808.05377 (2018)
10. Everingham, M., Eslami, S.M.A., Van Gool, L., Williams, C.K.I., Winn, J., Zisserman, A.: The Pascal visual object classes challenge: a retrospective. Int. J. Comput. Vis. **111**(1), 98–136 (2015). https://doi.org/10.1007/s11263-014-0733-5
11. Grathwohl, W., Creager, E., Kamyar, S., Ghasemipour, S., Zemel, R.: Gradient-based optimization of neural network architecture. In: International Conference on Learning Representations (ICLR), pp. 1–6 (2018)
12. Hu, J., Shen, L., Sun, G.: Squeeze-and-excitation networks. In: 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 7132–7141 (2018)
13. Huang, G., Liu, Z., van der Maaten, L., Weinberger, K.Q.: Densely connected convolutional networks. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 2261–2269 (2017)
14. Jin, H., Song, Q., Hu, X.: Efficient neural architecture search with network morphism. CoRR abs/1806.10282 (2018)
15. Kandasamy, K., Neiswanger, W., Schneider, J., Poczos, B., Xing, E.: Neural architecture search with bayesian optimisation and optimal transport (2018). http://arxiv.org/abs/1802.07191
16. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. CoRR abs/1412.6980 (2014). http://arxiv.org/abs/1412.6980
17. Klein, A., Falkner, S., Springenberg, J.T., Hutter, F.: Learning curve prediction with bayesian neural networks. In: International Conference on Learning Representations (ICLR), pp. 1–16. MCMC (2017)
18. Liu, C., et al.: Auto-DeepLab: hierarchical neural architecture search for semantic image segmentation. CoRR abs/1901.02985 (2019)
19. Liu, C., et al.: Progressive neural architecture search. In: ECCV (2018)
20. Liu, H., Simonyan, K., Vinyals, O., Fernando, C., Kavukcuoglu, K.: Hierarchical representations for efficient architecture search. In: 2018 International Conference on Learning Representations (ICLR) (2018). https://openreview.net/forum?id=BJQRKzbA-
21. Liu, H., Simonyan, K., Yang, Y.: DARTS: differentiable architecture search. In: 2019 International Conference on Learning Representations (ICLR) (2019). https://openreview.net/forum?id=S1eYHoC5FX
22. Miikkulainen, R., et al.: Evolving deep neural networks. In: Kozma, R., Alippi, C., Choe, Y., Morabito, F.C. (eds.) Artificial Intelligence in the Age of Neural Networks and Brain Computing. Elsevier, Amsterdam (2018). http://nn.cs.utexas.edu/?miikkulainen:chapter18
23. Milicchio, F., Gehrke, W.A.: Distributed Services with OpenAFS: For Enterprise and Education, 1st edn. Springer, Heidelberg (2010). Incorporated
24. Paszke, A., et al.: Automatic differentiation in PyTorch. In: NIPS, pp. 1–4 (2017)
25. Qing, Y.: FastDFS is an open source high performance distributed file system (DFS) (2013). https://github.com/happyfish100/fastdfs/tree/master
26. Sergeev, A., Balso, M.D.: Horovod: fast and easy distributed deep learning in TensorFlow. CoRR abs/1802.05799 (2018). http://arxiv.org/abs/1802.05799
27. Shin, R., Packer, C., Song, D.: Differentiable neural network architecture search. In: International Conference on Learning Representations (ICLR), pp. 1–4 (2018). No. 2017
28. Stanley, K.O., D'Ambrosio, D.B., Gauci, J.: A hypercube-based encoding for evolving large-scale neural networks. Artif. Life 15(2), 185–212 (2009). https://doi.org/10.1162/artl.2009.15.2.15202. http://www.mitpressjournals.org/doi/10.1162/artl.2009.15.2.15202

29. Stanley, K.O., Miikkulainen, R.: Evolving neural networks through augmenting topologies. Evol. Comput. **10**(2), 99–127 (2002). https://doi.org/10.1162/106365602320169811
30. Elsken, T., Metzen, J.H., Hutter, F.: Simple and efficient architecture search for convolutional neural networks. In: 2018 International Conference on Learning Representations (ICLR) (2018). https://openreview.net/forum?id=SySaJ0xCZ
31. Xie, L., Yuille, A.L.: Genetic CNN. In: 2017 IEEE International Conference on Computer Vision (ICCV), pp. 1388–1397 (2017)
32. Zoph, B., Le, Q.V.: Neural architecture search with reinforcement learning. In: International Conference on Learning Representations (ICLR), pp. 1–16 (2017). https://doi.org/10.1016/j.knosys.2015.01.010
33. Zoph, B., Vasudevan, V., Shlens, J., Le, Q.V.: Learning transferable architectures for scalable image recognition. In: 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 8697–8710 (2018)