Practical Network Blocks Design with Q-Learning

Zhao Zhong[1]*, Junjie Yan[2],Cheng-Lin Liu[1]

[1]*National Laboratory of Pattern Recognition,Institute of Automation, Chinese Academy of Sciences*
[2] *SenseTime Group Limited*
Email: {zhao.zhong, liucl}@nlpr.ia.ac.cn, yanjunjie@sensetime.com

Abstract

Convolutional neural network provides an end-to-end solution to train many computer vision tasks and has gained great successes. However, the design of network architectures usually relies heavily on expert knowledge and is hand-crafted. In this paper, we provide a solution to automatically and efficiently design high performance network architectures. To reduce the search space of network design, we focus on constructing network blocks, which can be stacked to generate the whole network. Blocks are generated through an agent, which is trained with Q-learning to maximize the expected accuracy of the searching blocks on the learning task. Distributed asynchronous framework and early stop strategy are used to accelerate the training process. Our experimental results demonstrate that the network architectures designed by our approach perform competitively compared with hand-crafted state-of-the-art networks. We trained the Q-learning on CIFAR-100, and evaluated on CIFAR10 and ImageNet, the designed block structure achieved 3.60% error on CIFAR-10 and competitive result on ImageNet. The Q-learning process can be efficiently trained only on 32 GPUs in 3 days.

Introduction

Convolutional neural network (CNN) (LeCun et al. 1989) have become the first choice for most computer vision tasks in the past few years. It first achieved successes in image classification (Krizhevsky, Sutskever, and Hinton 2012; Simonyan and Zisserman 2014; Szegedy et al. 2015a; He et al. 2016a), and then be applied to object detection (Girshick 2015; Ren et al. 2015), semantic segmentation (Long, Shelhamer, and Darrell 2015; Chen et al. 2016) and tracking (Nam and Han 2016; Bertinetto et al. 2016).

One key problem in convolutional neural network design is in finding better network architecture. The importance of network architecture has been manifested by the increasing performance due to the improvement of network architecture in the past years. For example, the ILSVRC ImageNet classification top-5 error rate was decreased from 16.4% to 3.57% progressively, from AlexNet to Inception and Inception. While a main advantage of convolutional neural network is that it makes traditional computer vision solutions end-to-end, the network architecture itself is hand-crafted. While the hand-crafted network architecture heavily relies

---

expert knowledge and intuition, in this paper we want to explore an alternative solution that can design network architectures automatically from data and transfer the designed architecture to different datasets.

Modern neural networks can have hundreds of layers and each layer can have many options in layer type and parameters, which makes the network design space huge. To exploit the network design space more efficiently, we only design blocks in network architecture, instead of the whole network. Actually, most CNN architectures can be viewed as the stack of several basic sub-structures usually called 'block'. The blocks are applied repeatedly to build a deep network. For example, the popular CNN models like VGG (Simonyan and Zisserman 2014), Inception (Szegedy et al. 2015a; Szegedy et al. 2015b) and Resnet (He et al. 2016a) all have their own unique blocks. Due to the block-constructive architecture, these networks have powerful generalization ability can transfer to different datasets and application domains.

In this paper, we propose a novel distributed asynchronous Q-learning framework we called BlockQNN to automatic generate block for convolutional networks (see Figure 1). The framework contains an agent, a block list controller and many child training environments. The agent samples layers of network block structures sequentially as actions in reinforcement learning by Q-value it learns, then the block list controller stores a batch of blocks and assign training task to different child environments. The environment constructs networks based on block structure codes it gets from the list controller, and train the network on classification task with early stop strategy for speeding up. With all blocks finished training, block list controller backward the early stop accuracy with redefined reward to the agent for updating Q-value. we also use experience replay (Lin 1993) and epsilon-greedy strategy (Mnih et al. 2015) to help the agent choose higher performing blocks.

Our experiments show that our framework can get good block structure starting from random exploration without any human knowledge to achieve a state-of-the-art performance of error rate 3.60% on CIFAR-10 dataset, and the learned block structure can be transferred to other big datasets easily. On ImageNet task, the model gives competitive results compared with other human designed state-of-the-art models. More importantly, we only need 32 GPUs

in 3 days for searching process that is affordable for most laboratories, which is much less than the Google brain's work (Zoph and Le 2016) which used 800 GPUs.

The primary contributions of this work can be summarized as follows.

- We propose a novel automatic block search strategy to design convolutional neural networks. It explores the network space efficiently and the designed block structures can transfer to other datasets and tasks easily.

- We prove that the automatically designed network blocks work even better than networks designed by human from many expert knowledges.

- We find that distributed asynchronous Q-learning with mini-batch strategy can work as well as non-batch Q-learning. We also find an early stop strategy for speeding up the training process with redefined Q-learning reward to correct the early stop accuracy. The two strategies ensure that the efficient training with 32 GPUs in 3 days.

Related Work

In early research on automating neural network architecture design, most works relied on the genetic algorithm or other evolutionary algorithms (Schaffer, Whitley, and Eshelman 1992; Stanley and Miikkulainen 2002; Stanley, D'Ambrosio, and Gauci 2009; Suganuma, Shirakawa, and Nagao 2017). There are some other works which focus on automatic selection of network architectures (Saxena and Verbeek 2016; Domhan, Springenberg, and Hutter 2015; Miconi 2016). These works can find suitable network architectures but do not perform competitively compared with hand-crafted networks. The recent work MetaQNN (Baker et al. 2016) and Neural Architecture Search (NAS) (Zoph and Le 2016) with reinforcement learning reported surprisingly good results and can beat the state-of-the-art hand-crafted networks. However, the network designed by reinforcement agent is constrained to particular dataset, and can't transfer to different input size datasets or tasks very well. Hence, this approach can hardly be applied to network design for large scaled problems like ImageNet classification task.

Our work is also related to the hyper-parameter optimization (Bergstra et al. 2011) , meta-learning (Vilalta and Drissi 2002) and learning to learn methods (Hochreiter, Younger, and Conwell 2001; Andrychowicz et al. 2016). The goal for these methods is to use meta-data for improving the performance of existing learning algorithms. For examples, to learn the learning rate of optimization methods or the number of hidden layers in network. In this paper, we focus on learning good blocks' topology for deep neural networks to improve classification task.

The block design conception follows the modern modular convolutional network such as VGG (Simonyan and Zisserman 2014), Inception (Szegedy et al. 2015a; Szegedy et al. 2015b) and Resnet (He et al. 2016a). They all achieve state-of-the-art performance on classification tasks and become the basal part in other task at that time. The VGG uses only 3x3 convolutional layers stacked on top of each other in increasing depth as a simple block and reduce the volume size
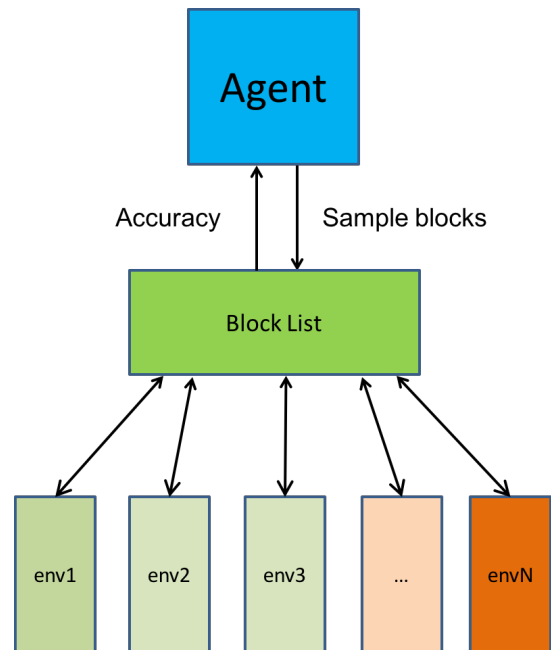

Figure 1: The distributed asynchronous Q-learning framework. It contains three parts: agent, block list controller and child environment. The N child environments train different block-based network in parallel.

by using max pooling. The inception network uses a multi-level feature extractor strategy by computing 1x1, 3x3, and 5x5 convolutions within the same module of the network to construct the block structure. The Resnet uses blocks with short cut connection that make it easy for network layers to represent the identity mapping, thus, the Resnet can be stacked very deeply with a lot of layers. The blocks generated by our approach have some similar features with the modern hand-crafted convolutional network, for example, some blocks contain short cut connection, but our blocks are all generated automatically without expert knowledge.

Our approach is motivated by the recently proposed MetaQNN (Baker et al. 2016), which uses a Q-learning (Watkins and Dayan 1992) based meta-modeling to search architecture configurations. Although MetaQNN can yield good performance on small datasets such as CIFAR-10, CIFAR-100 (Krizhevsky and Hinton 2009) in a small search space, the direct use of MetaQNN for architecture design on big datasets like ImageNet (Deng et al. 2009) is computationally expensive and the designed architecture is hard to generalize, because they search the whole neural network architecture space directly without block design. Instead, our approach is aimed to design convolution block structure by efficient search using a novel distributed asynchronous Q-learning framework and other acceleration strategy.

A few weeks ago, in preparing the paper, Google proposed their second version of NAS (Zoph et al. 2017). In this paper, they also design blocks and use more sophisticated layers. But their work still need 450 GPUs, which is not affordable for many researchers. They follow the design con-
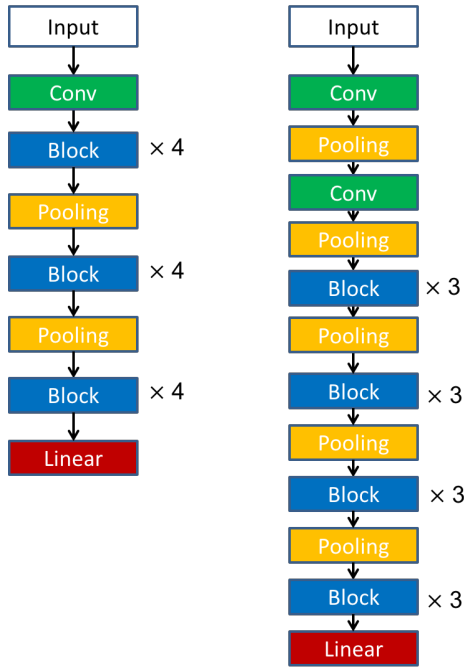
Figure 2: Network architectures we used for classification. They only consist of normal convolution layers, pooling layers and stacked blocks. The left one is for CIFAR-10 and right one for ImageNet task.

cept of RNN cell's instead of modern CNN blocks, and the RNN controller is used to recursively predict the structure parameters for repeating 5 times to construct one CNN cell but we use Q-learning to design the complete CNN blocks directly with connection codes.

The Proposed Method

Our method is based on Q-learning, which consists of an agent, states and a set of actions per state. The agent generates block structure by selecting the action with the highest value in each state. In our work, state represents the current layer in block, and action represents next layer the agent chosen. The last action in block is always terminal layer. Each state or action is defined as a tuple of structure codes, for example {layer number, layer type, kernel size, connection1, connection2} in our framework. The training environment receives the network architecture sampled from agent and then train the network on target dataset and backward the validation accuracy as the reward to agent for updating. With finite iteration, the agent can get an optimal policy to generate block structure.

Distributed Asynchronous Framework

The first problem we met is that it is very time consuming to complete searching process, usually take more than a week. Most of the time is spent in training environment, so we consider using multi-machine to accelerate training. Original MetaQNN only sampled one network architecture

at a time for update, but we need many networks training in parallel for multi-machine framework, thus we sampled 64 network codes as a minibatch, and when the batch is finished training, the agent update 64 times successively.

In our work, we propose a distributed asynchronous framework and minibatch strategy to speed up the learning of agent. The agent sampled a batch of structure codes at a time and store in a list controller, N child environments share the structure codes. Each environment trained in parallel, and when the minibatch is finished, list controller backward results of that minibatch to update the agent's weights. It's like a simplified version of parameter-server (Dean et al. 2012; Li et al. 2013). With this framework, we can train our model in multi-machine and multi-GPUs, in this work we use 32 GPUs, it's 10 times faster than use only 2 GPUs . This framework is shown in Figure 1.

The modern CNN design rule is to stack same neural block with different weights and feature map to construct the network. Following this rule, we constrict the searching space only in block level. So, we can consist different network use a series of block to deal with arbitrary inputs and different task.

Our strategy to construct a complete network is very simple, we just stack the block straightforwardly. All blocks have the same feature maps size between the pooling layers, and after pooling layer with a stride of two, feature map will be reduced by two but the block weights will be doubled. Figure 2 shows our network for CIFAR and ImageNet. Because the input image size is different for these two datasets, we set more pooling layer with a stride of two in the bottom layers in ImageNet network. More important, we can change the repeat times N of blocks for different demands, and even place the block in nonlinear manner.

Block Design

Architectural codes in previous works can only describe plain nets (that simply stack layers) that resulting in low performance. Powerful network like ResNet and Inception have shortcut connections or multi-branch connection in their block, it's more complex than plain nets. Unlike MetaQNN, our structure codes include connection parameters which can perform complexity neural network architectural like modern models. Our block structure codes are summarized in Table 1, it contains six different types of layers: convolution, max pooling, average pooling, identity, elemental add and concat. Only Computation layers have the kernel size parameters. Connection parameters means the in-layer's number, it should be less than the number of current layer. Only elemental add and concat layer have two connection parameters, and with the identity layer, we can transform multi-branch connection to two-branch connection, so our structure codes can describe nearly any network block topology.

In our work, we start block with the identity layer, and end with a terminal layer. All layers without out-connection are concatenated together to provide the final output. In elemental add layers, if two in-layers have different channel, we use 1x1 convolutions to match dimensions just like (He et al. 2016a). There is no down sampling operation in block.

Table 1: Neural Block's structure Code Space. K is the number that less than the number of current layer.

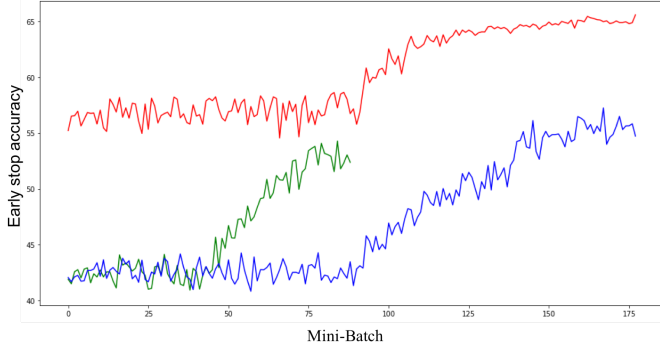| Layer Type	Convolution	Max Pooling	Average Pooling	Identity	Elemental Add	Concat	Terminal
Kernel Size	1, 3, 5	1, 3	1, 3	0	0	0	0
Connection 1	K	K	K	K	K	K	K
Connection 2	0	0	0	0	K	K	0

Figure 3: Q-learning result with different experiment setting on CIFAR-100. The blue line means the searching with ReLU and batch normalization layer code for 178 minibatch and the red line is searching with small pre-activation cell for 178 minibatch. The green one means searching with ReLU and batch normalization layer code for 89 minibatch.
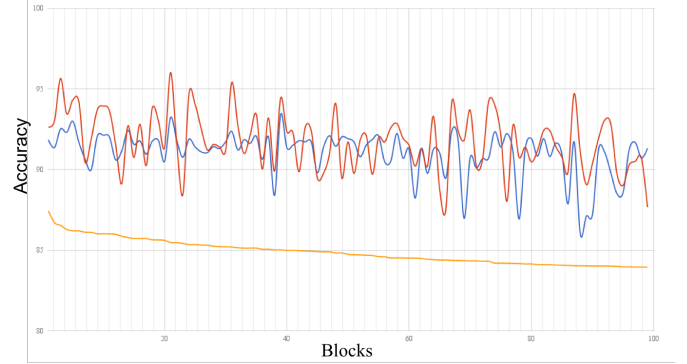

Figure 4: Top-100 blocks searching result on CIFAR-10 with ReLU and batch normalization layer code. The yellow line represents the early stop result and blue one is the train until convergence result. The red line is the redefined reward according to formula 1.

In experiment, we set the layers up to 23, which ensures the enough block structure space we can search.

All convolution layers in block is a small pre-activation(He et al. 2016b) cell that contain: ReLU, conventional convolution and batch normalization (Ioffe and Szegedy 2015). In the early experiment, we search ReLU and batch normalization directly, but it cost a low performance. As shown in Figure 3, the blue line is the searching with ReLU and batch normalization layer code, and red line is searching with small pre-activation cell, other experiment settings are same, we can find that there is a big gap between blue and red start from random exploration period. Because the block with searching with ReLU and batch normalization layer directly is more random than searching with pre-activation cell, so it's more likely to get "bad" blocks and need more searching space to find a good block structure. With the pre-activation cell, we can get better initialization for Q-learning and generate good block structure in limited update.

Training Speed Up

The time is the biggest problem in this work as said above. With distributed asynchronous framework, we can train the agent on multi-machine and multi-GPUs, but we have limited computing resources with only 32 GPUs for this experiment. Child environment take a lot of time to train network until convergence, but what we need is a reward to distinguish good or bad block instead of the exact result.

To deal with the trouble, we propose early stop strategy to speed up training process. Every child environment only train the generated network for a fixed 12 epochs on CIFAR-100 dataset during the architecture search process. It can save a lot of time compared with training for complete process. The reason why we searching on CIFAR-100 dataset is that we believe the result from CIFAR-100 is more discriminatory to find good block compare to CIFAR-10, which the result is very closed.

There is new trouble that this method will cause some mistakes because the early stop result is not the exact accuracy which can make an agent learn wrong block structures. So, we redefine the reward function.

We sampled 50 block structures from top-200, because we only concerned the good one in top blocks, to find a relation between real and early stop accuracy. Thus, we trained these 50 networks until convergence to get the exact result first, and we list FLOPs, parameters, dot, edge and density of the block structures. We discover that the FLOPs (He and Sun 2015) and density of the block structures' topology (edge divided by dot) are inversely proportional to the final exact accuracy, so we revised the validation accuracy reward with the formula 1.

$$reward = accuracy - \frac{1}{2}\log(FLOPs) - \frac{1}{2}\log(density) \quad (1)$$

As shown in Figure 4, the yellow line is the early stop result and blue line is the exact result of top 100 blocks in one experiment, we can find there are many mistakes that some
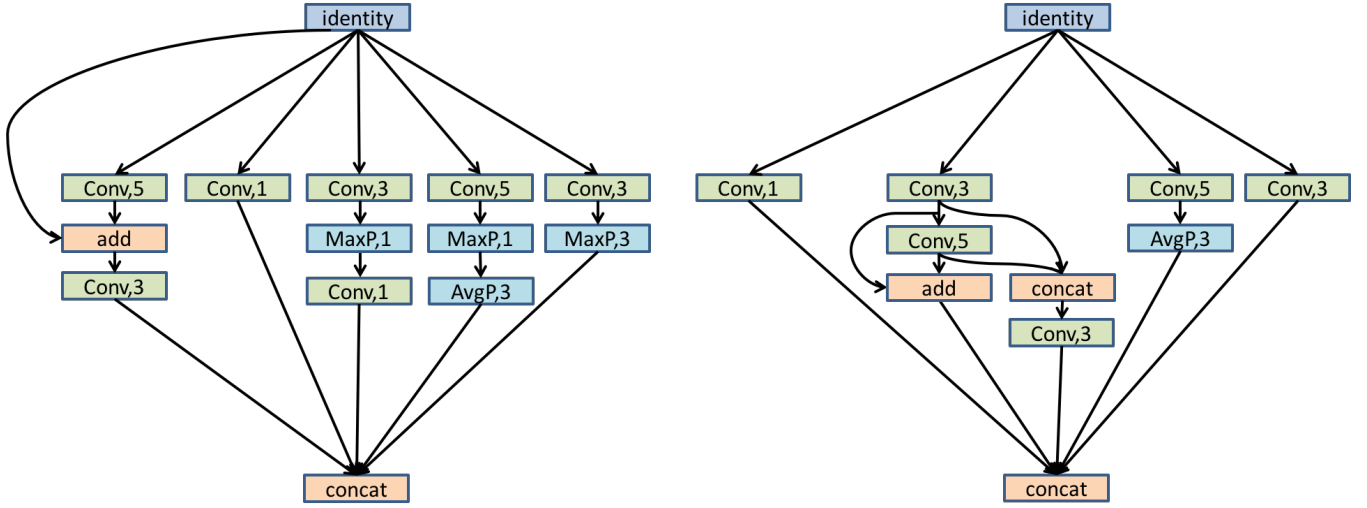
Figure 5: Topology of the Top-2 block structures searching on CIFAR-10 dataset.We call them Block-QNN-A and Block-QNN-B.

good blocks perform worse than bad blocks in early stop result line. The red line is the redefined reward formula 1, it's much more relevant to the blue line than yellow. The early stop strategy and new reward can help nearly 30 times faster than training until convergence in searching procedure.

With distributed asynchronous Q-learning framework, block structure codes and early stop strategy, we just cost 3 days to complete the searing process with only 32 GPUs. This is a huge benefit, since Google has used 800 GPUs to finish this task.

Experiments and Results

In this section, we describe our experiments with BlockQNN using the framework and search space described above to generate a good block structure. All searching experiments are conduct on CIFAR-100 dataset for Classification network. The agent was trained using Q-learning with experience replay and epsilon-greedy strategy.

We can get several candidate block structures generated by agent after searching. In our work, we choose the top 100 architectures to train until convergence on CIFAR-10 to verify the best architecture. Figure 5 shows the top-2 performing block from the CIFAR-10 verify process. We call them block-QNN-A and block-QNN-B. With the block structure, we can construct the network easily, the only hyper-parameters we need to decide is the repeating times of blocks. In subsection, we will show details of the searching experiment, verify experiment and transfer experiment on ImageNet task respectively.

Experiment Details

In the Q-learning updates process for searching blocks, the Q-learning rate is set to 0.01 and the minibatch size is 64. The agent samples 64 block structure codes at a time to compose a minibatch. Additionally, we find the iterative update
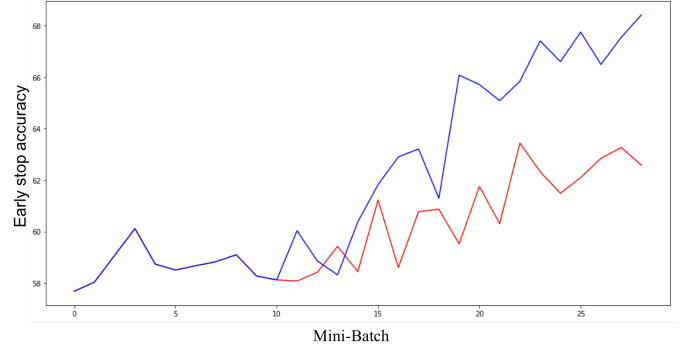

Figure 6: Q-learning result with different version of Bellmans Equation approximation. The red line is the original one Meta-QNN used, the blue line is based on Equation 2 in our work.

approximation of Bellmans Equation (Watkins 1989) Meta-QNN used doesn't work well in our experiment, maybe the different coding strategy or searching space caused. So, we do a little modification on the Equation.

$$Q_{t+1}(s_i, u) = (1 - \alpha)Q_t(s_i, u) +$$
$$\alpha[accuracy + max_{u' \in U(s_j)}Q_t(s_j, u'))] \qquad (2)$$

Equation 2 is a variant form, the only difference is that we add the accuracy reward at every layer's update, it can create faster convergence in our experiment. As shown in Figure 6, the red line is the result with original formula and blue one is Equation 2, two experiment start with same setting and initialization blocks, we can find that blue one Convergence faster than red one.

We decrease epsilon from 1.0 to 0.1, and we have tried different epsilon schedules and found that with a longer ex-

Table 2: Epsilon Schedules. The number of minibatch sampled by agent at each epsilon state.

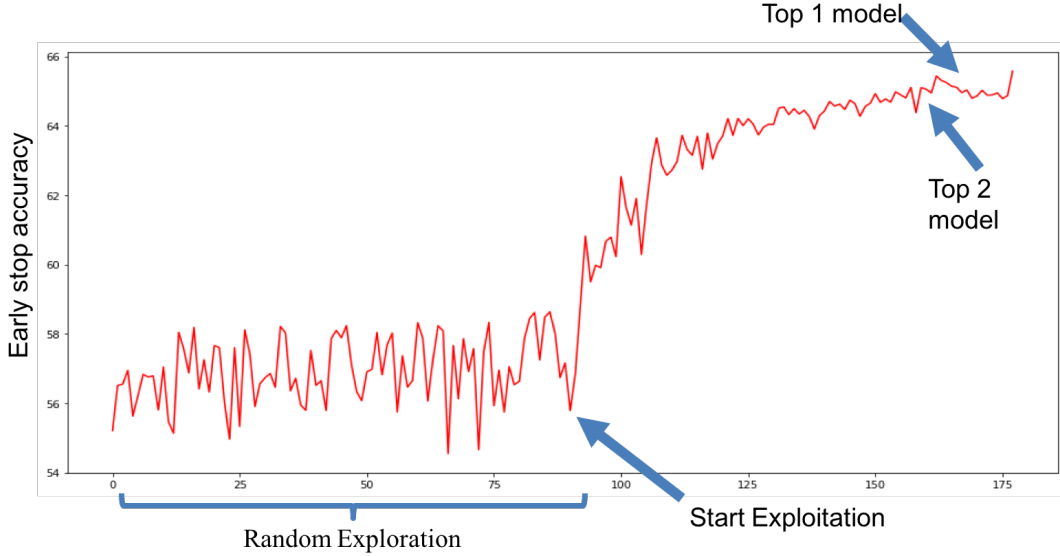| epsilon	1.0	0.9	0.8	0.7	0.6	0.5	0.4	0.3	0.2	0.1
Minibatch Trained	95	7	7	7	10	10	10	10	10	12

Figure 7: Q-learning Result.The accuracy goes up with the epsilon decrease and the top models are all found in the final stage, show that our agent can learn to search better block structures instead of random searching.

ploration and exploitation process, the result will get better as shown in Figure 3, green line with a shorter exploration and exploitation process but blue with longer, any other setting is same. This is because the searching space will be larger and the agent can see more block structures in the random exploration period.

Table 2 shows that the number of unique minibatch we trained at different epsilon. We maintain a replay memory same as (Baker et al. 2016) , after each minibatch is sampled and trained, the agent randomly samples 128 blocks from the memory dictionary and applies the Q-value update for 64 times.

During the block searching phase, the child environment trained each block topology for a fixed 12 epochs on CIFAR-100 as described above. CIFAR-100 is a dataset has 50,000 training samples and 10,000 testing samples in 100 classes. We trained without any data augmentation procedure. The batch size was set to 256 for saving times.

We use Adam optimizer (Kingma and Ba 2014) with $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\varepsilon = 10^{-8}$, and the initial learning rate was set to 0.001. If the model failed to perform better than a random predictor after the first epoch, we reduced the learning rate by a factor of 0.4 and restarted training, with a maximum of 3 restarts. For models that started learning we reduced the learning rate by a factor of 0.2 every 5 epochs. All weights were initialized as in (He et al. 2015). Our model is implemented under the pytorch scientific computing platform. We use the CUDA backend and use the cuDnn accelerated library in our implementation for high-performance GPU acceleration. Our experiments are carried out on 32 NVIDIA TitanX GPUs and took about 3 days to complete searching.

Block Searching Analysis

In Figure 7, we plot the mean prediction of early stop accuracy over 64 models (every minibatch) for CIFAR-100 searching experiments. After random exploration, the early stop accuracy begins to grow slowly, and achieves convergence in the end. The mean accuracy of models in random exploration stage is 56% and in stage of epsilon = 0.1 at the last the mean accuracy is nearly 65%.

As the Figure 7 shows, the Top models are all found in the final stage of the Q-learning process. It proves that our framework learns the way to generate better block structures rather than random searching a lot of models.

Experiment on CIFAR

The CIFAR-10 dataset consists of 60000 32x32 color images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images. We use the common data augmentation process that randomly crop 32x32 patches from padded images of size 40x40 and apply random horizontal flips for training. All models use the SGD optimizer with momentum rate set to 0.9 and weight decay set to 0.0005. We start with a learning rate of 0.1 and train the models for 300 epochs, reducing the learning rate at

Table 3: Block-QNN's results (error rate) compare with state-of-the-art methods on CIFAR-10 and CIFAR-100 dataset

Method	Depth	CIFAR-10	CIFAR-100
Network in Network (Lin, Chen, and Yan 2013)	-	8.81	35.68
Highway Network (Srivastava, Greff, and Schmidhuber 2015)	-	7.72	-
All-CNN (Springenberg et al. 2014)	-	7.25	33.71
VGGnet (Simonyan and Zisserman 2014)	-	7.25	-
ResNet (He et al. 2016a)	110	6.61	-
Wide ResNet (Zagoruyko and Komodakis 2016)	16	4.81	22.07
	28	4.17	20.5
ResNet (pre-activation) (He et al. 2016b)	164	5.46	24.33
	1001	4.62	22.71
DenseNet (L = 40, k = 12) (Huang et al. 2017)	40	5.24	24.42
DenseNet (L = 100, k = 12) (Huang et al. 2017)	100	4.10	20.20
DenseNet (L = 100, k = 24) (Huang et al. 2017)	100	3.74	19.25
DenseNet-BC (L = 10, k = 40) (Huang et al. 2017)	190	3.46	17.18
MetaQNN (ensemble) (Baker et al. 2016)	-	7.32	-
MetaQNN (top model) (Baker et al. 2016)	-	6.92	27.14
Neural Architecture Search v1 no stride or pooling (Zoph and Le 2016)	15	5.50	-
Neural Architecture Search v2 predicting strides (Zoph and Le 2016)	20	6.01	-
Neural Architecture Search v3 max pooling (Zoph and Le 2016)	39	4.47	-
Neural Architecture Search v3 max pooling + more filters (Zoph and Le 2016)	39	3.65	-
Block-QNN-A, N=4	24	3.60	18.64
Block-QNN-B, N=4	36	3.80	18.72

Table 4: Block-QNN's results compare with modern methods on ImageNet-1K Dataset

Method	input size	Depth	Top-1 error.(%)	Top-5 error.(%)
VGG (Simonyan and Zisserman 2014)	224x224	16	28.5	9.9
Inception V1 (Szegedy et al. 2015a)	224x224	22	27.8	10.1
Inception V2 (Ioffe and Szegedy 2015)	224x224	22	25.2	7.8
ResNet-50 (He et al. 2016b), our tested	224x224	50	24.7	7.7
ResNet-152 (He et al. 2016b)	224x224	152	23	6.7
Block-QNN-B, N=3	224x224	38	24.3	7.4

the 150-th and 225-th epoch. The batch size is set to 128 and all weights were initialized with MSRA initialization(He et al. 2015).

For the CIFAR-10 task, we set N=4. After top-100 models trained until convergence, we can find some good block structures. We compare our best architecture by automatic searching with different typical methods using the same dataset. The experimental results are shown in Table 3. It is shown that the searching models can achieve best performance which is very competitive to some of the best models design by human experts. The DenseNet-BC (Huang et al. 2017) model which achieves 3.46% error rate uses additional 1x1 convolutions in each composite function and

compressive transition layers to reduce parameters and improve performance, this strategy is not used in our framework, and our performance can be further improved by using it. Additionally, we do not introduce extra knowledge during the agent training procedure.

Our approach improves a lot compared with the original MetaQNN. More important, our best model is better than the NAS's best model NASv3 + more filters proposed by Google brain which training the whole system on 800 GPUs. We only need 32 GPUs to get the state of art performance in automatic designing network methods in 3 days.

We transfer the top blocks learned from CIFAR-10 to CIFAR-100 dataset, all experiment settings are same as

above. As summarized in Table 3, the blocks can also achieve state of art result on CIFAR-100 dataset that proved Block-QNN networks have powerful transfer learning ability.

Experiment on ImageNet

We transfer the block structure learned from CIFAR-10 to ImageNet dataset. ImageNet is a 1000-class image database for large scale image classification that consists of approximately 1.2M images.

We use SGD with a mini-batch size of 256 on 8 GPUs. The weight decay is 0.0001 and the momentum is 0.9. We start from a learning rate of 0.1, and divide it by 10 two times, at the 30-th and 60-th epochs. For the training images, we use a simple data augmentation that randomly cropped 224x224 patches from a resized image with its shorter side randomly sampled in [256, 480] with random horizontal flips. For the testing images, we evaluate the accuracy on single 224x224 center crop from an image shorter side resize to 256.

For the ImageNet task, we set N=3 and add more pooling operation before blocks. We use the best block structure learned from CIFAR-10 directly without any fine-tuning, and initialized with MSRA initialization same as above. The experimental results are shown in Table 4. The model generated by our framework can get competitive result compared with other human designed models. The recently proposed methods such as Xception (Chollet 2016) and ResNext(Xie et al. 2016) use special depth-wise convolution operation to reduce its total number of parameters and to improve performance, in our work, we do not use this new convolution operation, so it can't be compared fairly, and we will consider this in our future work to further improve the performance.

For automatic designing network methods, all previous works did not conduct the experiment on large scale image classification datasets. With the conception of block learning, we can transfer our architecture learned in small datasets to ImageNet task easily. In the future experiments, we will try the new models to improve the performance further.

Conclusion

In this paper, we show how to efficiently design high performance network blocks with Q-learning. we propose a distributed asynchronous Q-learning framework and early stop strategy focus on fast variable-length block structures searching. The result proves that we can automatic design block to construct good convolutional network for the classification task. Our Block-QNN networks outperform modern man-made networks as well as other automatic searching models. The best block which achieved a state-of-the-art performance in CIFAR-10 can transfer to the large-scale dataset ImageNet easily, and can get a competitive performance compared with best hand-crafted networks. We show that searching with the block design strategy can get more generalized network architecture. In the future, we will continue to improve the proposed framework from different aspects, such as using more powerful convolution layers and

making the searching process faster. We will also try to search blocks with limited parameters and FLOPs, and try to conduct experiments on other tasks such as detection or segmentation.

References

[Andrychowicz et al. 2016] Andrychowicz, M.; Denil, M.; Gomez, S.; Hoffman, M. W.; Pfau, D.; Schaul, T.; and de Freitas, N. 2016. Learning to learn by gradient descent by gradient descent. In *Advances in Neural Information Processing Systems*, 3981–3989.

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

[Bergstra et al. 2011] Bergstra, J. S.; Bardenet, R.; Bengio, Y.; and Kégl, B. 2011. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems*, 2546–2554.

[Bertinetto et al. 2016] Bertinetto, L.; Valmadre, J.; Henriques, J. F.; Vedaldi, A.; and Torr, P. H. 2016. Fully-convolutional siamese networks for object tracking. In *European Conference on Computer Vision*, 850–865. Springer.

[Chen et al. 2016] Chen, L.-C.; Papandreou, G.; Kokkinos, I.; Murphy, K.; and Yuille, A. L. 2016. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *arXiv preprint arXiv:1606.00915*.

[Chollet 2016] Chollet, F. 2016. Xception: Deep learning with depthwise separable convolutions. *arXiv preprint arXiv:1610.02357*.

[Dean et al. 2012] Dean, J.; Corrado, G.; Monga, R.; Chen, K.; Devin, M.; Mao, M.; Senior, A.; Tucker, P.; Yang, K.; Le, Q. V.; et al. 2012. Large scale distributed deep networks. In *Advances in neural information processing systems*, 1223–1231.

[Deng et al. 2009] Deng, J.; Dong, W.; Socher, R.; Li, L.-J.; Li, K.; and Fei-Fei, L. 2009. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, 248–255. IEEE.

[Domhan, Springenberg, and Hutter 2015] Domhan, T.; Springenberg, J. T.; and Hutter, F. 2015. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *IJCAI*, 3460–3468.

[Girshick 2015] Girshick, R. 2015. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, 1440–1448.

[He and Sun 2015] He, K., and Sun, J. 2015. Convolutional neural networks at constrained time cost. In *Proceedings*

of the IEEE Conference on Computer Vision and Pattern Recognition, 5353–5360.

[He et al. 2015] He, K.; Zhang, X.; Ren, S.; and Sun, J. 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, 1026–1034.

[He et al. 2016a] He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016a. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778.

[He et al. 2016b] He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016b. Identity mappings in deep residual networks. In *European Conference on Computer Vision*, 630–645. Springer.

[Hochreiter, Younger, and Conwell 2001] Hochreiter, S.; Younger, A. S.; and Conwell, P. R. 2001. Learning to learn using gradient descent. In *International Conference on Artificial Neural Networks*, 87–94. Springer.

[Huang et al. 2017] Huang, G.; Liu, Z.; Weinberger, K. Q.; and van der Maaten, L. 2017. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*.

[Ioffe and Szegedy 2015] Ioffe, S., and Szegedy, C. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, 448–456.

[Kingma and Ba 2014] Kingma, D., and Ba, J. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

[Krizhevsky and Hinton 2009] Krizhevsky, A., and Hinton, G. 2009. Learning multiple layers of features from tiny images.

[Krizhevsky, Sutskever, and Hinton 2012] Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. Imagenet classification with deep convolutional neural networks. In *Proc. Advances in Neural Information Processing Systems*, 1097–1105.

[LeCun et al. 1989] LeCun, Y.; Boser, B.; Denker, J. S.; Henderson, D.; Howard, R. E.; Hubbard, W.; and Jackel, L. D. 1989. Backpropagation applied to handwritten zip code recognition. *Neural Computation* 1(4):541–551.

[Li et al. 2013] Li, M.; Zhou, L.; Yang, Z.; Li, A.; Xia, F.; Andersen, D. G.; and Smola, A. 2013. Parameter server for distributed machine learning. In *Big Learning NIPS Workshop*, volume 6, 2.

[Lin, Chen, and Yan 2013] Lin, M.; Chen, Q.; and Yan, S. 2013. Network in network. In *International Conference on Learning Representations*.

[Lin 1993] Lin, L.-J. 1993. Reinforcement learning for robots using neural networks. Technical report, Carnegie-Mellon Univ Pittsburgh PA School of Computer Science.

[Long, Shelhamer, and Darrell 2015] Long, J.; Shelhamer, E.; and Darrell, T. 2015. Fully convolutional networks for semantic segmentation. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, 3431–3440.

[Miconi 2016] Miconi, T. 2016. Neural networks with differentiable structure. *arXiv preprint arXiv:1606.06216*.

[Mnih et al. 2015] Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518(7540):529–533.

[Nam and Han 2016] Nam, H., and Han, B. 2016. Learning multi-domain convolutional neural networks for visual tracking. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 4293–4302.

[Ren et al. 2015] Ren, S.; He, K.; Girshick, R.; and Sun, J. 2015. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, 91–99.

[Saxena and Verbeek 2016] Saxena, S., and Verbeek, J. 2016. Convolutional neural fabrics. In *Advances in Neural Information Processing Systems*, 4053–4061.

[Schaffer, Whitley, and Eshelman 1992] Schaffer, J. D.; Whitley, D.; and Eshelman, L. J. 1992. Combinations of genetic algorithms and neural networks: A survey of the state of the art. In *Combinations of Genetic Algorithms and Neural Networks, 1992., COGANN-92. International Workshop on*, 1–37. IEEE.

[Simonyan and Zisserman 2014] Simonyan, K., and Zisserman, A. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

[Springenberg et al. 2014] Springenberg, J. T.; Dosovitskiy, A.; Brox, T.; and Riedmiller, M. 2014. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*.

[Srivastava, Greff, and Schmidhuber 2015] Srivastava, R. K.; Greff, K.; and Schmidhuber, J. 2015. Highway networks. *arXiv preprint arXiv:1505.00387*.

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

[Stanley, D'Ambrosio, and Gauci 2009] Stanley, K. O.; D'Ambrosio, D. B.; and Gauci, J. 2009. A hypercube-based encoding for evolving large-scale neural networks. *Artificial life* 15(2):185–212.

[Suganuma, Shirakawa, and Nagao 2017] Suganuma, M.; Shirakawa, S.; and Nagao, T. 2017. A genetic programming approach to designing convolutional neural network architectures. *arXiv preprint arXiv:1704.00764*.

[Szegedy et al. 2015a] Szegedy, C.; Liu, W.; Jia, Y.; Sermanet, P.; Reed, S.; Anguelov, D.; Erhan, D.; Vanhoucke, V.; and Rabinovich, A. 2015a. Going deeper with convolutions. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, 1–9.

[Szegedy et al. 2015b] Szegedy, C.; Vanhoucke, V.; Ioffe, S.; Shlens, J.; and Wojna, Z. 2015b. Rethinking the inception architecture for computer vision. 2818–2826.

[Vilalta and Drissi 2002] Vilalta, R., and Drissi, Y. 2002. A

perspective view and survey of meta-learning. *Artificial Intelligence Review* 18(2):77–95.

[Watkins and Dayan 1992] Watkins, C. J., and Dayan, P. 1992. Q-learning. *Machine learning* 8(3-4):279–292.

[Watkins 1989] Watkins, C. J. C. H. 1989. *Learning from delayed rewards*. Ph.D. Dissertation, King's College, Cambridge.

[Xie et al. 2016] Xie, S.; Girshick, R.; Dollár, P.; Tu, Z.; and He, K. 2016. Aggregated residual transformations for deep neural networks. *arXiv preprint arXiv:1611.05431*.

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

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

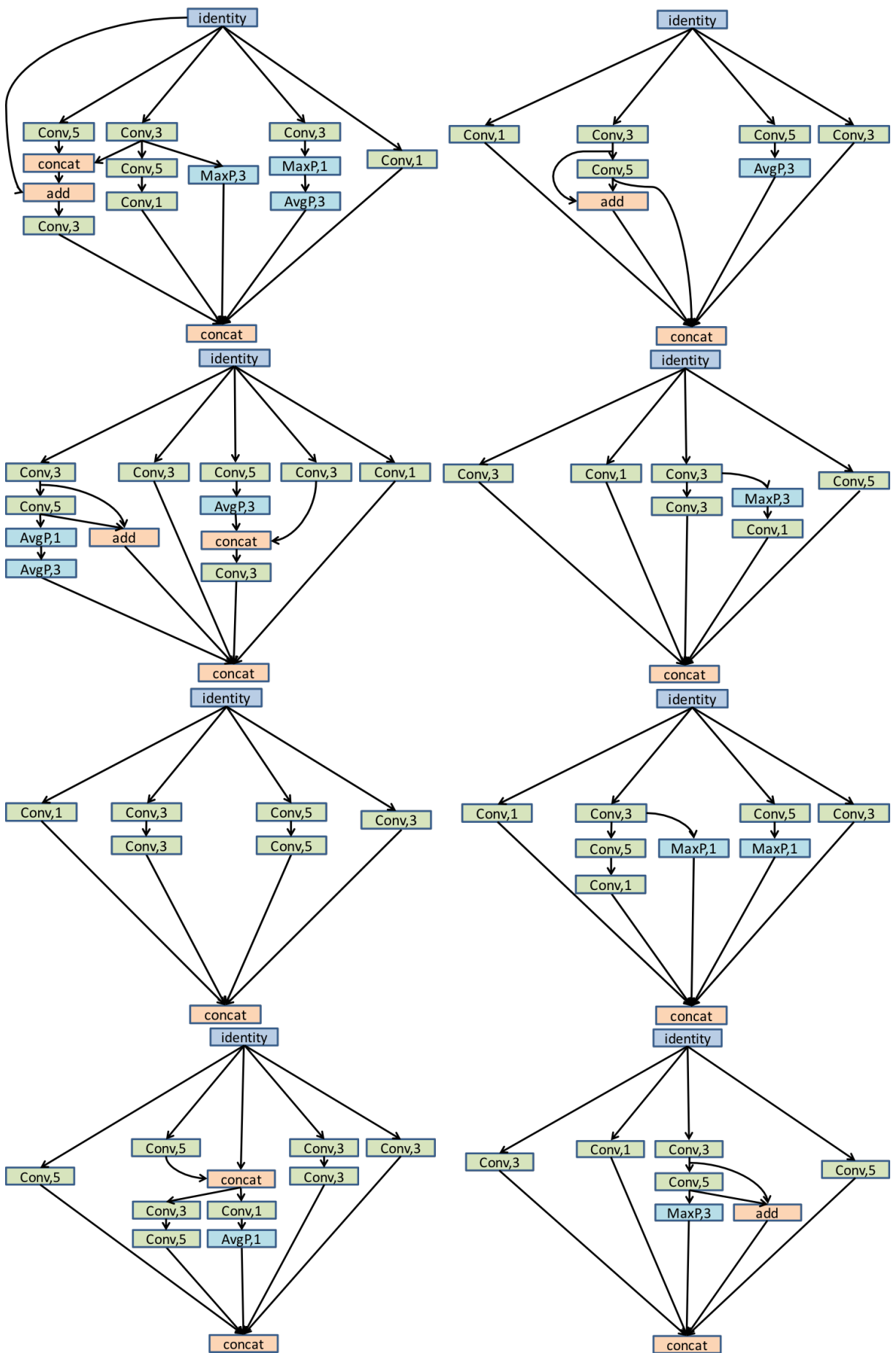[Zoph et al. 2017] Zoph, B.; Vasudevan, V.; Shlens, J.; and Le, Q. V. 2017. Learning transferable architectures for scalable image recognition. *arXiv preprint arXiv:1707.07012*.

Figure 8: Topology of other Block-QNN blocks.