Practical Block-wise Neural Network Architecture Generation

Zhao Zhong[1,3]*, Junjie Yan[2], Wei Wu[2], Jing Shao[2], Cheng-Lin Liu[1,3,4]

[1]National Laboratory of Pattern Recognition, Institute of Automation, Chinese Academy of Sciences
[2] SenseTime Research Institute   [3] University of Chinese Academy of Sciences
[4] CAS Center for Excellence of Brain Science and Intelligence Technology
Email: {zhao.zhong, liucl}@nlpr.ia.ac.cn, {yanjunjie, wuwei, shaojing}@sensetime.com

Abstract

Convolutional neural networks have gained a remarkable success in computer vision. However, most usable network architectures are hand-crafted and usually require expertise and elaborate design. In this paper, we provide a block-wise network generation pipeline called BlockQNN which automatically builds high-performance networks using the Q-Learning paradigm with epsilon-greedy exploration strategy. The optimal network block is constructed by the learning agent which is trained sequentially to choose component layers. We stack the block to construct the whole auto-generated network. To accelerate the generation process, we also propose a distributed asynchronous framework and an early stop strategy. The block-wise generation brings unique advantages: (1) it performs competitive results in comparison to the hand-crafted state-of-the-art networks on image classification, additionally, the best network generated by BlockQNN achieves 3.54% top-1 error rate on CIFAR-10 which beats all existing auto-generate networks. (2) in the meanwhile, it offers tremendous reduction of the search space in designing networks which only spends 3 days with 32 GPUs, and (3) moreover, it has strong generalizability that the network built on CIFAR also performs well on a larger-scale ImageNet dataset.

1. Introduction

During the last decades, Convolutional Neural Networks (CNNs) have shown remarkable potentials almost in every field in the computer vision society [17]. For example, thanks to the network evolution from AlexNet [16], VGG [25], Inception [30] to ResNet [10], the top-5 performance on ImageNet challenge steadily increases from 83.6% to 96.43%. However, as the performance gain usually requires an increasing network capacity, a high-performance network architecture generally possesses a

tremendous number of possible configurations about the number of layers, hyperparameters in each layer and type of each layer. It is hence infeasible for manually exhaustive search, and the design of successful hand-crafted networks heavily rely on expert knowledge and experience. Therefore, constructing the network in a smart and automatic manner remains an open problem.

Although some recent works have attempted computer-aided or automated network design [2, 37], there are several challenges still unsolved: (1) Modern neural networks always consist of hundreds of convolutional layers, each of which has numerous options in type and hyperparameters. It makes a huge search space and heavy computational costs for network generation. (2) One typically designed network is usually limited on a specific dataset or task, and thus is hard to transfer to other tasks or generalize to another dataset with different input data sizes. In this paper, we provide a solution to the aforementioned challenges by a novel fast Q-learning framework, called *BlockQNN*, to automatically design the network architecture, as shown in Fig. 1.

Particularly, to make the network generation efficient and generalizable, we introduce the `block`-wise network generation, *i.e.*, we construct the network architecture as a flexible stack of personalized blocks rather tedious per-layer network piling. Indeed, most modern CNN architectures such as Inception [30, 14, 31] and ResNet Series [10, 11] are assembled as the stack of basic `block` structures. For example, the inception and residual blocks shown in Fig. 1 are repeatedly concatenated to construct the entire network. With such kind of `block`-wise network architecture, the generated network owns a powerful generalization to other task domains or different datasets.

In comparison to previous methods like NAS [37] and MetaQNN [2], as depicted in Fig. 1, we present a more readily and elegant model generation method that specifically designed for block-wise generation. Motivated by the unsupervised reinforcement learning paradigm, we employ the well-known Q-learning [33] with experience replay [19] and epsilon-greedy strategy [21] to effectively and

---

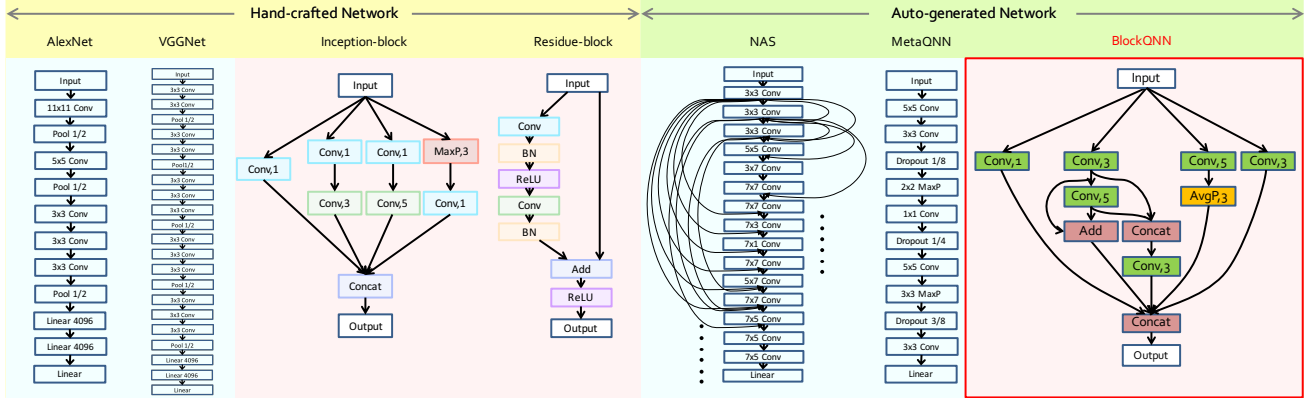*The work was done when the first author interns at SenseTime.

Figure 1. The proposed **BlockQNN** (right in red box) compared with the hand-crafted networks marked in yellow and the existing auto-generated networks in green. Automatically generating the `plain` networks [2, 37] marked in blue need large computational costs on searching optimal layer types and hyperparameters for each single layer, while the `block`-wise network heavily reduces the cost to search structures only for one block. The entire network is then constructed by stacking the generated blocks. Similar `block` concept has been demonstrated its superiority in hand-crafted networks, such as inception-block and residue-block marked in red.

efficiently search the optimal block structure. The network block is constructed by the learning agent which is trained sequentiality to choose component layers. Afterwards we stack the block to construct the whole auto-generated network. Moreover, we propose an early stop strategy to enable efficient search with fast convergence. A novel reward function is designed to ensure the accuracy of the early stopped network having positive correlation with the converged network. We can pick up good blocks in reduced training time using this property. With this acceleration strategy, we can construct a Q-learning agent to learn the optimal block-wise network structures for a given task with limited resources (*e.g.* few GPUs or short time period). The generated architectures are thus succinct and have powerful generalization ability compared to the networks generated by the other automatic network generation methods.

The proposed `block`-wise network generation brings a few advantages as follows:

- *Effective*. The automatically generated networks present comparable performances to those of hand-crafted networks with human expertise. The proposed method is also superior to the existing works and achieves a state-of-the-art performance on CIFAR-10 with 3.54% error rate.
- *Efficient*. We are the first to consider `block`-wise setup in automatic network generation. Companied with the proposed early stop strategy, the proposed method results in a fast search process. The network generation for CIFAR task reaches convergence with only 32 GPUs in 3 days, which is much more efficient than that by NAS [37] with 800 GPUs in 28 days.
- *Transferable*. It offers surprisingly superior transferable ability that the network generated for CIFAR can be transferred to ImageNet with little modification but still achieve outstanding performance.

2. Related Work

Early works, from 1980s, have made efforts on automating neural network design which often searched good architecture by the genetic algorithm or other evolutionary algorithms [24, 27, 26, 28, 23, 7, 34]. Nevertheless, these works, to our best knowledge, cannot perform competitively compared with hand-crafted networks. Recent works, *i.e.* Neural Architecture Search (NAS) [37] and MetaQNN [2], adopted reinforcement learning to automatically search a good network architecture. Although they can yield good performance on small datasets such as CIFAR-10, CIFAR-100, the direct use of MetaQNN or NAS for architecture design on big datasets like ImageNet [6] is computationally expensive via searching in a huge space. Besides, the network generated by this kind of methods is task-specific or dataset-specific, that is, it cannot been well transferred to other tasks nor datasets with different input data sizes. For example, the network designed for CIFAR-10 cannot been generalized to ImageNet.

Instead, our approach is aimed to design network block architecture by an efficient search method with a distributed asynchronous Q-learning framework as well as an early-stop strategy. The block design conception follows the modern convolutional neural networks such as Inception [30, 14, 31] and Resnet [10, 11]. The inception-based networks construct the `inception blocks` via a hand-crafted multi-level feature extractor strategy by computing 1×1, 3×3, and 5×5 convolutions, while the Resnet uses `residue blocks` with shortcut connection to make it easier to represent the identity mapping which allows a very deep network. The `blocks` automatically generated by our approach have similar structures such as some blocks contain short cut connections and inception-like multi-branch combination. We will discuss the details in Section 5.1.

Name	Index	Type	Kernel Size	Pred1	Pred2
Convolution	**T**	1	1, 3, 5	**K**	0
Max Pooling	**T**	2	1, 3	**K**	0
Average Pooling	**T**	3	1, 3	**K**	0
Identity	**T**	4	0	**K**	0
Elemental Add	**T**	5	0	**K**	**K**
Concat	**T**	6	0	**K**	**K**
Terminal	**T**	7	0	0	0

Table 1. Network Structure Code Space. The space contains seven types of commonly used layers. Layer index stands for the position of the current layer in a block, the range of the parameters is set to be $\mathbf{T} = \{1, 2, 3, ... \text{max layer index}\}$. Three kinds of kernel sizes are considered for convolution layer and two sizes for pooling layer. Pred1 and Pred2 refer to the predecessor parameters which is used to represent the index of layers predecessor, the allowed range is $\mathbf{K} = \{1, 2, ..., \text{current layer index} - 1\}$

Codes = [(1,4,0,0,0), (2,1,1,1,0), (3,1,3,2,0), (4,1,1,1,0), (5,1,5,4,0), (6,6,0,3,5), (7,2,3,1,0), (8,1,1,7,0), (9,6,0,6,8), (10,7,0,0,0)]

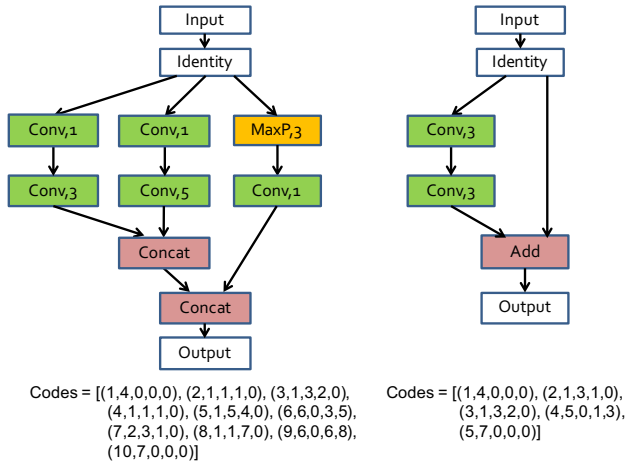Codes = [(1,4,0,0,0), (2,1,3,1,0), (3,1,3,2,0), (4,5,0,1,3), (5,7,0,0,0)]

Figure 2. Representative block exemplars with their Network structure codes (NSC) respectively: the block with multi-branch connections (left) and the block with shortcut connections (right).

Another bunch of related works include hyper-parameter optimization [3], meta-learning [32] and learning to learn methods [12, 1]. However, the goal of these works is to use meta-data to improve the performance of the existing algorithms, such as finding the optimal learning rate of optimization methods or the optimal number of hidden layers to construct the network. In this paper, we focus on learning the entire topological architecture of network blocks to improve the performance.

3. Methodology

3.1. Convolutional Neural Network Blocks

The modern CNNs, *e.g.* Inception and Resnet, are designed by stacking several `blocks` each of which shares similar structure but with different weights and filter numbers to construct the network. With the block-wise design, the network can not only achieves high performance but
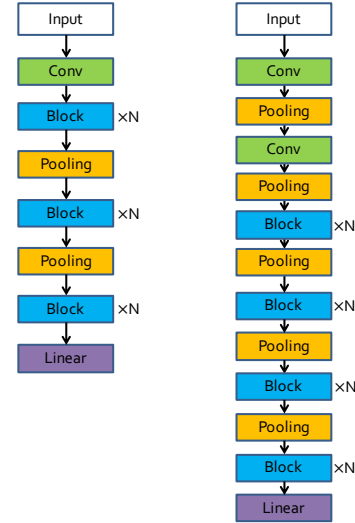

Figure 3. Auto-generated networks on CIFAR-10 (left) and ImageNet (right). Each network starts with a few convolution layers to learn low-level features, and followed by multiple repeated `blocks` with several pooling layers inserted to downsample.

also has powerful generalization ability to different datasets and tasks. Unlike previous research on automating neural network design which generate the entire network directly, we aim at designing the `block` structure.

As a CNN contains a feed-forward computation procedure, we represent it by a directed acyclic graph (DAG), where each node corresponds to a layer in the CNN while directed edges stand for data flow from one layer to another. To turn such a graph into a uniform representation, we propose a novel layer representation called **Network Structure Code** (NSC), as shown in Table 2. Each block is then depicted by a set of 5-D NSC vectors. In NSC, the first three numbers stand for the *layer index*, *operation type* and *kernel size*. The last two are *predecessor parameters* which refer to the position of a layer's predecessor layer in structure codes. The second predecessor (Pred2) is set for the layer owns two predecessors, and for the layer with only one predecessor, Pred2 will be set to zero. This design is motivated by the current powerful hand-crafted networks like Inception and Resnet which own their special block structures. This kind of block structure shares similar properties such as containing more complex connections, *e.g.* shortcut connections or multi-branch connections, than the simple connections of the plain network like AlexNet. Thus, the proposed NSC can encode complexity architectures as shown in Fig. 2. In addition, all of the layer without successor in the block are concatenated together to provide the final output. Note that each convolution operation, same as the declaration in Resnet [11], refers to a **Pre-activation Convolutional Cell** (PCC) with three components, *i.e. ReLU*, *Convolution* and *Batch Normalization*. This results in a smaller
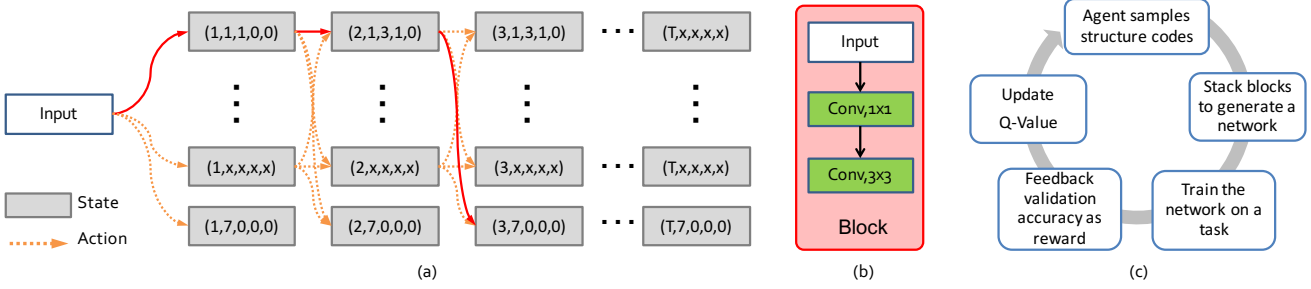
Figure 4. Q-learning process illustration. (a) The state transition process by different action choices. The block structure in (b) is generated by the red solid line in (a). (c) The flow chart of the Q-learning procedure.

searching space than that with three components separate search, and hence with the PCC, we can get better initialization for searching and generating optimal block structure with a quick training process.

Based on the above defined `blocks`, we construct the complete network by stacking these block structures sequentially which turn a common plain network into its counterpart block version. Two representative auto-generated networks on CIFAR and ImageNet tasks are shown in Fig. 3. There is no down-sampling operation within each block. We perform down-sampling directly by the pooling layer. If the size of feature map is halved by pooling operation, the block's weights will be doubled. The architecture for ImageNet contains more pooling layers than that for CIFAR because of their different input sizes, *i.e.* 224×224 for ImageNet and 32×32 for CIFAR. More importantly, the `blocks` can be repeated any N times to fulfill different demands, and even place the blocks in other manner, such as inserting the block into the Network-in-Network [20] framework or setting short cut connection between different blocks.

3.2. Designing Network Blocks With Q-Learning

Albeit we squeeze the search space of the entire network design by focusing on constructing network `blocks`, there is still a large amount of possible structures to seek. Therefore, we employ reinforcement learning rather than random sampling for automatic design. Our method is based on Q-learning, a kind of reinforcement learning, which concerns how an agent ought to take actions so as to maximize the cumulative reward. The Q-learning model consists of an *agent*, *states* and a set of *actions*.

In this paper, the *state* $s \in S$ represents the status of the current layer which is defined as a Network Structure Code (NSC) claimed in Section 3.1, *i.e.* 5-D vector {layer index, layer type, kernel size, pred1, pred2}. The *action* $a \in A$ is the decision for the next successive layer. Thanks to the defined NSC set with a limited number of choices, both the *state* and *action* space are thus finite and discrete to ensure a relatively small searching space. The state tran-

sition process $(s_t, a(s_t)) \rightarrow (s_{t+1})$ is shown in Fig. 4(a), where t refers to the current layer. The block example in Fig. 4(b) is generated by the red solid lines in Fig. 4(a). The learning agent is given the task of sequentially picking NSC of a block. The structure of block can be considered as an action selection trajectory $\tau_{a_{1:T}}$, *i.e.* a sequence of NSCs. We model the layer selection process as a Markov Decision Process with the assumption that a well-performing layer in one block should also perform well in another block [2]. To find the optimal architecture, we ask our agent to maximize its expected *reward* over all possible trajectories, denoted by R_τ,

$$R_\tau = \mathbb{E}_{P(\tau_{a_{1:T}})}[\mathbb{R}], \tag{1}$$

where the $\mathbb{R}$ is the cumulative reward. For this maximization problem, it is usually to use recursive Bellman Equation to optimality. Given a state $s_t \in S$, and subsequent action $a \in A(s_t)$, we define the maximum total expected reward to be $Q^*(s_t, a)$ which is known as Q-value of state-action pair. The recursive Bellman Equation then can be written as

$$Q^*(s_t, a) = \mathbb{E}_{s_{t+1}|s_t,a}[\mathbb{E}_{r|s_t,a,s_{t+1}}[r|s_t, a, s_{t+1}] + \gamma \max_{a' \in A(s_{t+1}))} Q^*(s_{t+1}, a')]. \tag{2}$$

An empirical assumption to solve the above quantity is to formulate it as an iterative update:

$$Q(s_T, a) = 0 \tag{3}$$

$$Q(s_{T-1}, a_T) = (1 - \alpha)Q(s_{T-1}, a_T) + \alpha r_T \tag{4}$$

$$Q(s_t, a) = (1 - \alpha)Q(s_t, a) + \alpha[r_t + \gamma \max_{a'} Q(s_{t+1}, a')], t \in \{1, 2, ...T - 2\}, \tag{5}$$

where α is the learning rate which determines how the newly acquired information overrides the old information, γ is the discount factor which measures the importance of future rewards. r_t denotes the intermediate reward observed for the current state s_t and s_T refers to final state, *i.e.* terminal layers. r_T is the validation accuracy of corresponding
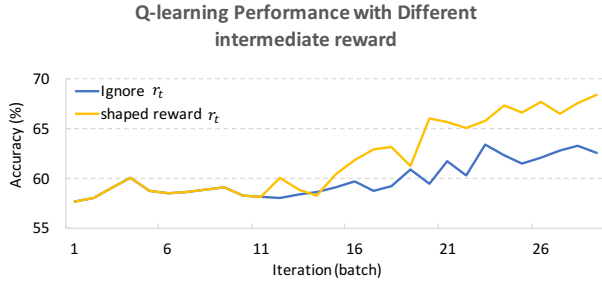
Figure 5. Comparison results of Q-learning with and without the shaped intermediate reward r_t. By taking our shaped reward, the learning process convergent faster than that without shaped reward start from the same exploration.

network trained convergence on training set for a_T, *i.e.* action to final state. Since the reward r_t cannot be explicitly measured in our task, we use reward shaping [22] to speed up training. The shaped intermediate reward is defined as:

$$r_t = \frac{r_T}{T}. \tag{6}$$

Previous works [2] ignore these rewards in the iterative process, *i.e.* set them to zero, which may cause a slow convergence in the beginning. This is known as the temporal credit assignment problem which makes RL time consuming [29]. In this case, the Q-value of s_T is much higher than others in early stage of training and thus leads the agent prefer to stop searching at the very beginning, *i.e.* tend to build small block with fewer layers. We show a comparison result in Fig. 5, the learning process of the agent with our shaped reward r_t convergent much faster than previous method.

We summarize the learning procedure in Fig. 4(c). The agent first samples a set of structure codes to build the block architecture, based on which the entire network is constructed by stacking these blocks sequentially. We then train the generated network on a certain task, and the validation accuracy is regarded as the reward to update the Q-value. Afterwards, the agent picks another set of structure codes to get a better block structure.

3.3. Early Stop Strategy

Introducing `block`-wise generation indeed increases the efficiency. However, it is still time consuming to complete the search process. To further accelerate the learning process, we introduce an early stop strategy. As we all know, early stopping training process might result in a poor accuracy. Fig. 6 shows an example, where the early-stop accuracy in yellow line is much lower than the final accuracy in orange line, which means that some good blocks unfortunately perform worse than bad blocks when stop training early. In the meanwhile, we notice that the FLOPs and density of the corresponding blocks have a negative correlation
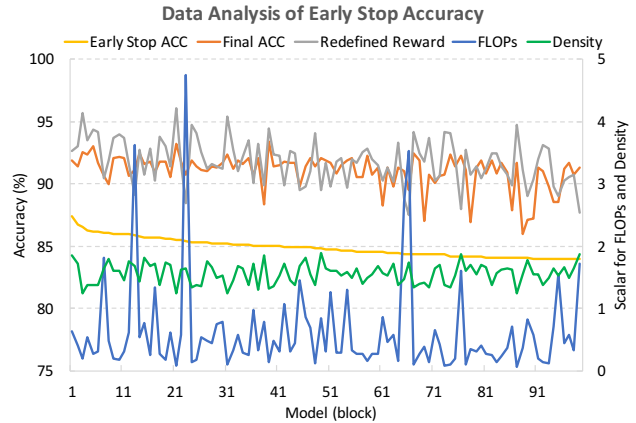

Figure 6. The performance of early stop training is poorer than the final accuracy of a complete training. With the help of FLOPs and Density, it squeezes the gap between the redefined reward function and the final accuracy.

with the final accuracy. Thus, we redefine the reward function as

$$reward = \text{ACC}_{\text{EarlyStop}} - \mu \log(\text{FLOPs}) \\ -\rho \log(\text{Density}), \tag{7}$$

where FLOPs [8] refer to an estimation of computational complexity of the block, and Density is the edge number divided by the dot number in DAG of the block. There are two hyperparameters, μ and ρ, to balance the weights of FLOPs and Density. With the redefined reward function, the reward is more relevant to the final accuracy.

With this early stop strategy and small searching space of network blocks, it just costs 3 days to complete the searching process with only 32 GPUs, which is superior to that of [37], spends 28 days with 800 GPUs to achieve the same performance.

4. Framework and Training Details

4.1. Distributed Asynchronous Framework

To speed up the learning of agent, we use a distributed asynchronous framework as illustrated in Fig. 7. It consists of three parts: master node, controller node and compute nodes. The agent first samples a batch of block structures in master node. Afterwards, we store them in a controller node which uses the block structures to build the entire networks and allocates these networks to compute nodes. It can be regarded as a simplified parameter-server [5, 18]. Specifically, the network is trained in parallel on each of compute nodes and returns the validation accuracy as reward by controller nodes to update agent. With this framework, we can generate network efficiently on multiple machines with multiple GPUs.
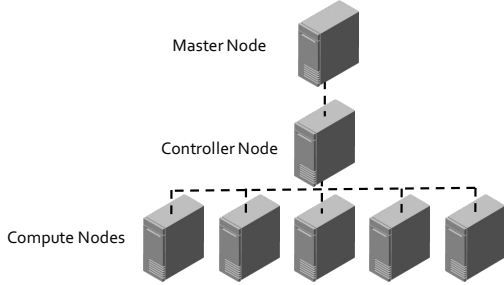
Figure 7. The distributed asynchronous framework. It contains three parts: master node, controller node and compute nodes.

ϵ	1.0	0.9	0.8	0.7	0.6	0.5	0.4	0.3	0.2	0.1
Iters	95	7	7	7	10	10	10	10	10	12

Table 2. Epsilon Schedules. The number of iteration the agent trains at each epsilon(ϵ) state.

4.2. Training Details

Epsilon-greedy Strategy. The agent is trained using Q-learning with experience replay [19] and epsilon-greedy strategy [21]. With epsilon-greedy strategy, the random action is taken with probability ϵ and the greedy action is chosen with probability $1 - \epsilon$. We decrease epsilon from 1.0 to 0.1 following the epsilon schedule as shown in Table 2 such that the agent can transform smoothly from exploration to exploitation. We find that the result goes better with a longer exploration, since the searching scope would become larger and the agent can see more block structures in the random exploration period.

Experience Replay. Following [2], we employ a replay memory to store the validation accuracy and block description after each iteration. Within a given interval, *i.e.* each training iteration, the agent samples 64 blocks with their corresponding validation accuracies from the memory and updates Q-value 64 times.

BlockQNN Generation.

In the Q-learning update process, the learning rate α is set to 0.01 and the discount factor γ is 1. We set the hyperparameters μ and ρ in the redefined reward function as 1 and 8, respectively. The agent samples 64 sets of NSC vectors at a time to compose a mini-batch and the maximum layer index for a block is set to 23. We train the agent with 178 iterations, *i.e.* sampling $11,392$ blocks in total.

During the block searching phase, the compute nodes train each generated network for a fixed 12 epochs on CIFAR-100 using the early top strategy as described in Section 3.3. CIFAR-100 contains $60,000$ samples with 100 classes which are divided into training and test set with the ratio of $5 : 1$. We train the network without any data augmentation procedure. The batch size is set to 256. We use Adam optimizer [15] with $\beta_1 = 0.9$, $\beta_2 = 0.999$,

$\varepsilon = 10^{-8}$. The initial learning rate is set to 0.001 and is reduced with a factor of 0.2 every 5 epochs. All weights are initialized as in [9]. If the training result after the first epoch is worse than the random guess, we reduce the learning rate by a factor of 0.4 and restart training, with a maximum of 3 times for restart-operations.

After obtaining one optimal block structure, we build the whole network with stacked blocks and train the network until converging to get the validation accuracy as the criterion to pick the best network. In this phase, we augment data with randomly cropping the images with size of 32×32 and horizontal flipping. 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 in the 150-th and 225-th epoch. The batch size is set to 128 and all weights are initialized with MSRA initialization [9].

Transferable BlockQNN. We also evaluate the transferability of the best auto-generated block structure searched on CIFAR-100 to a smaller dataset, CIFAR-10, with only 10 classes and a larger dataset, ImageNet, containing 1.2M images with 1000 classes. All the experimental settings are the same as those on the CIFAR-100 stated above. The training is conducted with a mini-batch size of 256 where each image has data augmentation of randomly cropping and flipping, and is optimized with SGD strategy. The initial learning rate, weight decay and momentum are set as 0.1, 0.0001 and 0.9, respectively. We divide the learning rate by 10 twice, at the 30-th and 60-th epochs. The network is trained with a total of 90 epochs. We evaluate the accuracy on the test images with center crop.

Our framework is implemented under the PyTorch scientific computing platform. We use the CUDA backend and 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.

5. Results

5.1. Block Searching Analysis

Fig. 8(a) provides early stop accuracies over 178 batches on CIFAR-100, each of which is averaged over 64 auto-generated block-wise network candidates within in each mini-batch. After random exploration, the early stop accuracy grows steadily till converges. The mean accuracy within the period of random exploration is 56% while finally achieves 65% in the last stage with $\epsilon = 0.1$. We choose top-100 block candidates and train their respective networks to verify the best block structure. We show top-2 block structures in Fig. 8(b-c), denoted as **Block-QNN-A** and **Block-QNN-B**. As shown in Fig. 8(a), both top-2 blocks are found in the final stage of the Q-learning pro-
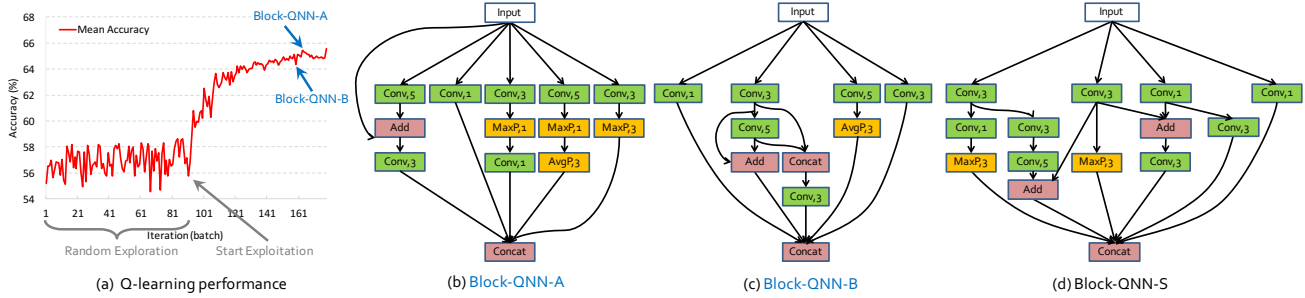
Figure 8. (a) Q-learning performance on CIFAR-100. The accuracy goes up with the epsilon decrease and the top models are all found in the final stage, show that our agent can learn to generate better block structures instead of random searching. (b-c) Topology of the Top-2 block structures generated by our approach. We call them Block-QNN-A and Block-QNN-B. (d) Topology of the best block structures generated with limited parameters, named Block-QNN-S.
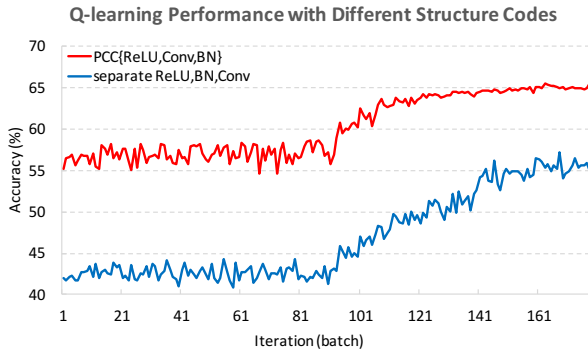

Figure 9. Q-learning result with different NSC on CIFAR-100. The red line refers to searching with PCC, *i.e.* combination of ReLU, Conv and BN. The blue stands for separate searching with ReLU, BN and Conv. The red line is better than blue from the beginning with a big gap.

cess, which proves the effectiveness of the proposed method in searching optimal block structures rather than randomly searching a large amount of models. Furthermore, we observe that the generated blocks share similar properties with those state-of-the-art hand-crafted networks. For example, Block-QNN-A and Block-QNN-B contain short-cut connections and multi-branch structures which have been manually designed in residual-based and inception-based networks. Compared to other auto-generated methods, the networks generated by our approach are more elegant and can automatically and effectively reveal the beneficial properties for optimal network structure.

To squeeze the searching space, as stated in Section 3.1, we define a Pre-activation Convolutional Cell (PCC) consists of three components, *i.e.* ReLU, convolution and batch normalization (BN). We show the superiority of the PCC, searching a combination of three components, in Fig. 9, compared to the separate search of each component. Searching the three components separately is more likely to

Method	Depth	Para	C-10	C-100
VGG [25]	-		7.25	-
ResNet [10]	110	1.7M	6.61	-
Wide ResNet [36]	28	36.5M	4.17	20.5
ResNet (pre-activation) [11]	1001	10.2M	4.62	22.71
DenseNet (k = 12) [13]	40	1.0M	5.24	24.42
DenseNet (k = 12) [13]	100	7.0M	4.10	20.20
DenseNet (k = 24) [13]	100	27.2M	3.74	19.25
DenseNet-BC (k = 40) [13]	190	25.6M	3.46	17.18
MetaQNN (ensemble) [2]	-	-	7.32	-
MetaQNN (top model) [2]	-	11.2M	6.92	27.14
NAS v1 [37]	15	4.2M	5.50	-
NAS v2 [37]	20	2.5M	6.01	-
NAS v3 [37]	39	7.1M	4.47	-
NAS v3 more filters [37]	39	37.4M	3.65	-
Block-QNN-A, N=4	25	-	3.60	18.64
Block-QNN-B, N=4	37	-	3.80	18.72
Block-QNN-S, N=2	19	6.1M	4.38	20.65
Block-QNN-S more filters	22	39.8M	3.54	18.06

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

generate "bad" blocks and also needs more searching space and time to pursue "good" blocks.

5.2. Results on CIFAR

Due to the small size of images (*i.e.* 32×32) in CIFAR, we set block stack number as $N = 4$. We compare our generated best architectures with the state-of-the-art hand-crafted networks or auto-generated networks in Table 3.

Comparison with hand-crafted networks - It shows that our Block-QNN networks outperform most hand-crafted networks. The DenseNet-BC [13] uses additional 1×1 convolutions in each composite function and compressive tran-

sition layer to reduce parameters and improve performance, which is not adopted in our design. Our performance can be further improved by using this prior knowledge.

Comparison with auto-generated networks - Our approach achieves a significant improvement to the MetaQNN [2], and even better than NAS's best model (*i.e. NASv3 more filters*) [37] proposed by Google brain which needs an expensive costs on time and GPU resources. As shown in Table 4, NAS trains the whole system on **800 GPUs** in **28 days** while we only need **32 GPUs** in **3 days** to get state-of-the-art performance.

Transfer block from CIFAR-100 to CIFAR-10 - We transfer the top blocks learned from CIFAR-100 to CIFAR-10 dataset, all experiment settings are the same. As shown in Table 3, the blocks can also achieve state-of-the-art results on CIFAR-10 dataset with 3.60% error rate that proved Block-QNN networks have powerful transferable ability.

Analysis on network parameters - The networks generated by our method might be complex with a large amount of parameters since we do not add any constraints during training. We further conduct an experiment on searching networks with limited parameters and adaptive block numbers. We set the maximal parameter number as 10M and obtain an optimal block (*i.e.* **Block-QNN-S**) which outperforms NASv3 with less parameters, as shown in Fig. 8(d). In addition, when involving more filters in each convolutional layer (*e.g.* from [32,64,128] to [80,160,320]), we can achieve even better result (3.54%).

5.3. Transfer to ImageNet

To demonstrate the generalizability of our approach, we transfer the block structure learned from CIFAR to ImageNet dataset.

For the ImageNet task, we set block repeat number $N = 3$ and add more down sampling operation before blocks, the filters for convolution layers in different level blocks are [64,128,256,512]. We use the best blocks structure learned from CIFAR-100 directly without any fine-tuning, and the generated network initialized with MSRA initialization as same as above. The experimental results are shown in Table 5. The network generated by our framework can get competitive result compared with other human designed models. The recently proposed methods such as Xception [4] and ResNext [35] use special depth-wise convolution operation to reduce their 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.

As far as we known, most previous works of automatic network generation did not report competitive result on large scale image classification datasets. With the con-

Method	Best Model on CIFAR10	GPUs	Time(days)
MetaQNN [2]	6.92	10	10
NAS [37]	3.65	800	28
Our approach	3.54	32	3

Table 4. The required computing resource and time of our approach compare with other automatic designing network methods.

Method	Input Size	Depth	Top-1	Top-5
VGG [25]	224x224	16	28.5	9.90
Inception V1 [30]	224x224	22	27.8	10.10
Inception V2 [14]	224x224	22	25.2	7.80
ResNet-50 [11]	224x224	50	24.7	7.80
ResNet-152 [11]	224x224	152	23.0	6.70
Xception(our test) [4]	224x224	36	23.6	7.10
ResNext-101(64x4d) [35]	224x224	101	20.4	5.30
Block-QNN-B, N=3	224x224	38	24.3	7.40
Block-QNN-S, N=3	224x224	38	22.6	6.46

Table 5. Block-QNN's results (single-crop error rate) compare with modern methods on ImageNet-1K Dataset.

ception of block learning, we can transfer our architecture learned in small datasets to big dataset like ImageNet task easily. In the future experiments, we will try to apply the generated blocks in other tasks such as object detection and semantic segmentation.

6. Conclusion

In this paper, we show how to efficiently design high performance network blocks with Q-learning. We use a distributed asynchronous Q-learning framework and an early stop strategy focusing on fast block structures searching. We applied the framework to automatic block generation for constructing good convolutional network. Our Block-QNN networks outperform modern hand-crafted networks as well as other auto-generated networks in image classification tasks. The best block structure which achieves a state-of-the-art performance on CIFAR can be transfer to the large-scale dataset ImageNet easily, and also yield a competitive performance compared with best hand-crafted networks. We show that searching with the block design strategy can get more elegant and model explicable network architectures. 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 FLOPs and conduct experiments on other tasks such as detection or segmentation.

Acknowledgments

References

[1] M. Andrychowicz, M. Denil, S. Gomez, M. W. Hoffman, D. Pfau, T. Schaul, and N. de Freitas. Learning to learn by gradient descent by gradient descent. In *Advances in Neural Information Processing Systems*, pages 3981–3989, 2016. 3

[2] B. Baker, O. Gupta, N. Naik, and R. Raskar. Designing neural network architectures using reinforcement learning. In *6th International Conference on Learning Representations*, 2017. 1, 2, 4, 5, 6, 7, 8

[3] J. S. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems*, pages 2546–2554, 2011. 3

[4] F. Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017. 8

[5] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, A. Senior, P. Tucker, K. Yang, Q. V. Le, et al. Large scale distributed deep networks. In *Advances in neural information processing systems*, pages 1223–1231, 2012. 5

[6] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009. 2

[7] T. Domhan, J. T. Springenberg, and F. Hutter. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *IJCAI*, pages 3460–3468, 2015. 2

[8] K. He and J. Sun. Convolutional neural networks at constrained time cost. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5353–5360, 2015. 5

[9] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015. 6

[10] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 1, 2, 7

[11] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. In *European Conference on Computer Vision*, pages 630–645. Springer, 2016. 1, 2, 3, 7, 8

[12] S. Hochreiter, A. S. Younger, and P. R. Conwell. Learning to learn using gradient descent. In *International Conference on Artificial Neural Networks*, pages 87–94. Springer, 2001. 3

[13] G. Huang, Z. Liu, K. Q. Weinberger, and L. van der Maaten. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017. 7

[14] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456, 2015. 1, 2, 8

[15] D. Kingma and J. Ba. Adam: A method for stochastic optimization. In *3rd International Conference for Learning Representations*, 2015. 6

[16] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proc. Advances in Neural Information Processing Systems*, pages 1097–1105, 2012. 1

[17] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015. 1

[18] M. Li, L. Zhou, Z. Yang, A. Li, F. Xia, D. G. Andersen, and A. Smola. Parameter server for distributed machine learning. In *Big Learning NIPS Workshop*, volume 6, page 2, 2013. 5

[19] L.-J. Lin. Reinforcement learning for robots using neural networks. Technical report, Carnegie-Mellon Univ Pittsburgh PA School of Computer Science, 1993. 1, 6

[20] M. Lin, Q. Chen, and S. Yan. Network in network. In *International Conference on Learning Representations*, 2013. 4

[21] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015. 1, 6

[22] A. Y. Ng, D. Harada, and S. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, volume 99, pages 278–287, 1999. 5

[23] S. Saxena and J. Verbeek. Convolutional neural fabrics. In *Advances in Neural Information Processing Systems*, pages 4053–4061, 2016. 2

[24] J. D. Schaffer, D. Whitley, and L. J. Eshelman. 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*, pages 1–37. IEEE, 1992. 2

[25] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *3rd International Conference for Learning Representations*, 2015. 1, 7, 8

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

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

[28] M. Suganuma, S. Shirakawa, and T. Nagao. A genetic programming approach to designing convolutional neural network architectures. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 497–504, 2017. 2

[29] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998. 5

[30] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015. 1, 2, 8

[31] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2818–2826, 2016. 1, 2

[32] R. Vilalta and Y. Drissi. A perspective view and survey of meta-learning. *Artificial Intelligence Review*, 18(2):77–95, 2002. 3

[33] C. J. C. H. Watkins. *Learning from delayed rewards*. PhD thesis, King's College, Cambridge, 1989. 1

[34] L. Xie and A. Yuille. Genetic cnn. In *Proceedings of the International Conference on Computer Vision*, 2017. 2

[35] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He. Aggregated residual transformations for deep neural networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5987–5995. IEEE, 2017. 8

[36] S. Zagoruyko and N. Komodakis. Wide residual networks. In *British Machine Vision Conference*, 2016. 7

[37] B. Zoph and Q. V. Le. Neural architecture search with reinforcement learning. In *6th International Conference on Learning Representations*, 2017. 1, 2, 5, 7, 8