Runtime Network Routing for Efficient Image Classification

Yongming Rao, *Student Member, IEEE,* Jiwen Lu, *Senior Member, IEEE,* Ji Lin, *Student Member, IEEE,* and Jie Zhou, *Senior Member, IEEE*

Abstract—In this paper, we propose a generic Runtime Network Routing (RNR) framework for efficient image classification, which selects an optimal path inside the network. Unlike existing static neural network acceleration methods, our method preserves the full ability of the original large network and conducts dynamic routing at runtime according to the input image and current feature maps. The routing is performed in a bottom-up, layer-by-layer manner, where we model it as a Markov decision process and use reinforcement learning for training. The agent determines the estimated reward of each sub-path and conducts routing conditioned on different samples, where a faster path is taken when the image is easier for the task. Since the ability of network is fully preserved, the balance point is easily adjustable according to the available resources. We test our method on both multi-path residual networks and incremental convolutional channel pruning, and show that RNR consistently outperforms static methods at the same computation complexity on both the CIFAR and ImageNet datasets. Our method can also be applied to off-the-shelf neural network structures and easily extended to other application scenarios.

Index Terms—Deep network compression, image classification, efficient inference model, reinforcement learning, deep learning

1 Introduction

Deep neural networks have been proven to be effective in various areas. Despite the great success, the capability of deep neural networks comes at the cost of huge computational burdens and large power consumption, which is a big challenge for real-time deployments, especially for embedded systems. To address this, several neural pruning methods have been proposed recently [14], [15], [16], [34], [56] to reduce the parameters of convolutional neural networks, which achieve competitive or even slightly better performance. However, these works mainly focus on reducing the number of network weights, which have limited effects on speeding up the computation. More specifically, fully connected layers are proven to be more redundant and contribute more to the overall pruning rate, while convolutional layers are the most computationally dense part of the network. Furthermore, such pruning strategy usually leads to an irregular network structure, *i.e.* with part of sparsity in convolution kernels, which requires a special algorithm for speeding up and is hard to harvest actual computational savings. A surprisingly effective approach to trade accuracy for the size and the speed is to simply reduce the number of channels in each convolutional layer. For example, Changpinyo *et al.* [9] proposed a method to speed up the network by deactivating connections between filters in convolutional layers, achieving a better tradeoff between the accuracy and the speed.

All these methods produce a static model for all the input images. However, it is obvious that some of the input samples are easier for recognition, which can be recognized by simpler and faster models. Some other samples are more difficult, which require more computational resources to achieve the expected output. For example, compared to distinguish female faces from man faces, it is much easier to distinguish background images from faces (see section 4.2.2). This property is not well exploited in previous efficient neural frameworks, where input samples are usually treated equally, and the compression procedure is conducted over the whole dataset. Since some of the weights are lost during the pruning process, the network will lose the ability for some hard tasks forever. We argue that preserving the whole ability of the network and routing between different paths of the neural network dynamically according to the input image is desirable to achieve better tradeoff between the speed and the accuracy compared to static efficient methods, which will also not harm the upper bound ability of the network. Moreover, since the functionality of different network components is separated among input samples, each component serves a simpler and more concentrated functionality, thus requires smaller complexity.

In this paper, we propose a Runtime Network Routing (RNR) framework for deep neural network compression, which selects an optimal path inside the network for compression. Fig. 1 shows four different types of network models to illustrate the basic idea of our framework. Unlike existing static neural network acceleration methods which employ the structures as shown in Fig. 1(a) and Fig. 1(b), our method preserves the full ability of the original large network and conducts dynamic routing at runtime, according to the input image and current feature maps, as shown in Fig. 1(c) and Fig. 1(d). More specifically, the routing is performed in a bottom-up, layer-by-layer manner, which we model it as a Markov decision process (MDP) and train an

- *The authors are with the Department of Automation, Tsinghua University, State Key Lab of Intelligent Technologies and Systems, and Beijing National Research Center for Information Science and Technology (BNRist), Beijing, 100084, China. Email: raoyongming95@gmail.com; lujiwen@tsinghua.edu.cn; linj14@mails.tsinghua.edu.cn; jzhou@tsinghua.edu.cn.*
- *Partial of this work was presented in [39].*

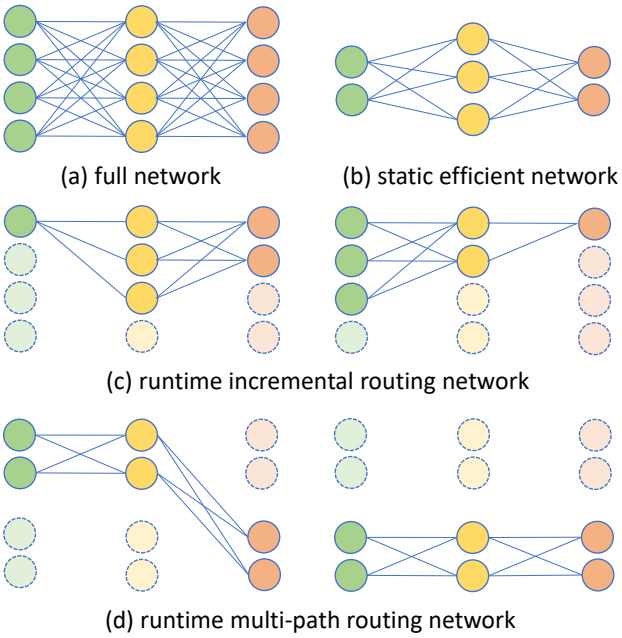

Fig. 1. Four different types of models. (a) is the full neural network, where all the parameters are preserved. (b) is a static efficient model obtained by pruning some of the connections or directly reducing the parameter size. (c) is a runtime incremental routing network. It has the same number of parameters as the full model, while only part of the network is routed according to the input image in an incremental way. (d) is a multi-path network, where there are several smaller branches. The network routes between one of the branches to obtain higher accuracy-calculation tradeoff. (c) and (d) are the models used in our RNR framework.

agent with reinforcement learning to learn the best policy for routing. The agent determines the estimated reward of each sub-path and conducts routing conditioned on different samples, where a faster path is taken when the image is easier for the task. Since the ability of network is fully preserved, the balance point is easily adjustable according to the available resources. We tested our method on both the multi-path residual network and the incremental convolutional channel pruning. Experimental results on both the CIFAR [30] and ImageNet [52] datasets show that our framework successfully learns to allocate different amount of computational resources for different input images, and achieves much better performance at the same cost.

The contributions of this work are summarized as follows:

1) We propose a runtime incremental routing network that dynamically routes between output channels of convolutional layers. The framework selects an incremental number of feature map channels to accelerate the inference for simple images.
2) We propose a runtime multi-path routing network, where each block contains several small convolutional residual branches. By routing in one specific branch, the network learns to obtain competitive performance at much lower computational costs.
3) We conduct experiments on both the CIFAR [30] and ImageNet [52] datasets, and our experimental results demonstrate that our method consistently outperforms static methods at the same computation complexity.

2 Related Work

In this section, we briefly review four related topics: 1) CNN-based image classification, 2) deep network pruning, 3) deep reinforcement learning, and 4) dynamic network.

2.1 CNN-based Image Classification

Convolutional neural networks (CNN) have dominated many computer vision tasks since AlexNet [31] won the ImageNet Challenge ILSVRC 2012 [52] by large margins. The network has been growing deeper and more complicated to achieve higher accuracy [19], [54], [60], [63]. VGG-16 [54] is among the first attempts to train a very large and deep model. Inception [63] uses a specially designed module to improve the parameter efficiency. ResNet enables training of models deeper than 100 layers with residual connections. While the network is getting deeper and heavier, it brings considerable computational burdens during inference. There has been rising interest in building small and efficient networks [20], [25], [50], [65]. Wang [65] propose to accelerate the inference of CNN by matrix factorization of weights. SqueezeNet [25] presents a specially designed Fire module to achieve the AlexNet-level accuracy with 50x fewer parameters. Recently, MobileNet [20] utilizes the depthwise separable convolutions and gains state-of-the-art results among lightweight models.

2.2 Deep Network Pruning

There has been several works focusing on deep network pruning, which is a valid way to reduce the network complexity. For example, Hanson and Pratt [16] introduced hyperbolic and exponential biases to the pruning objective. Damage [34] and Surgeon [17] pruned the networks with second-order derivatives of the objective. Han *et al.* [14], [15] iteratively pruned near-zero weights to obtain a pruned network with no loss of accuracy. Some other works exploited more complicated regularizers. For example, [36], [67] introduced structured sparsity regularizers on the network weights, [46] put them to the hidden units. [21] pruned neurons based on the network output. Anwar *et al.* [2] considered channel-wise and kernel-wise sparsity, and proposed to use particle filters to decide the importance of connections and paths. Another aspect focuses on deactivating some subsets of connections inside a fixed network architecture. LeCun *et al.* [33] removed connections between the first two convolutional feature maps in a uniform manner. Depth multiplier method was proposed in [20] to reduce the number of filters in each convolutional layer by a factor in a uniform manner. These methods produced a static model for all the samples, failing to exploit the different property of input images. Moreover, most of them produced irregular network structures after pruning, which makes it hard to harvest actual computational savings directly.

2.3 Deep Reinforcement Learning

Reinforcement learning [40] aims to enable the agent to decide the behavior from its experiences. Unlike conventional machine learning methods, reinforcement learning is conducted through the reward signals of actions. Deep reinforcement learning [44] is a combination of deep learning and reinforcement learning, which has been widely used in recent years. For examples, Mnih *et al.* [44] combined reinforcement learning with CNN and achieved the human-level performance in the Atari game. Caicedo *et al.* [8] introduced reinforcement learning for active object localization. Zhang *et al.* [68] employed reinforcement learning for vision control in robotics. Reinforcement learning has also been adopted for feature selection to build a fast classifier [4], [18], [28]. [4] proposed an algorithm to build sparse decision DAGs (directed acyclic graphs) from a list of base classifiers such as AdaBoost. A coach network is adopted for imitation learning in [18] to build a cost-sensitive dynamic feature selection framework. Karayev *et al.* [28] proposed a dynamic, closed-loop policy that infers the contents of the image for timely object recognition, aiming to decide which detector to deploy next, which is learned from execution traces using reinforcement learning.

2.4 Dynamic Network

Dynamic network structures and executions have been studied in previous works [7], [11], [37], [47], [53], [57], [58]. Some input-dependent execution methods rely on a predefined strategy. Cascaded methods [35], [37], [57], [58] employ manually-selected thresholds to control execution. Dynamic capacity Network [1] used a specially designed method to compute a saliency map for control execution. Other conditional computation methods activate parts of a network under a learned policy. Bengio *et al.* [6] introduced Stochastic Times Smooth neurons as gaters for conditional computation within a deep neural network, producing a sparse binary gater to be computed as a function of the input. Bengio *et al.* [5] selectively activated output of a fully-connected neural network, according to a control policy parameterized as the sigmoid of an affine transformation from last activation. Liu *et al.* [41] proposed Dynamic Deep Neural Networks (D2NN), a feed-forward deep neural network that allows selective execution with self-defined topology, where the control policy is learned using single step reinforcement learning. In their work, control node is a part of network structure, thus special network design is needed, which cannot be directly applied to prevalent network architectures such as VGG [54] and ResNet [19]. One work that is very related to the framework in [55], which learns a sequential decision process over the filters of a convolutional neural network (CNN). However, the method aims to improve the overall accuracy after the feature map is calculated, which cannot be utilized for acceleration. It also uses a gradient-free evolutionary algorithm, while we use the gradient-based method. There have been some efforts on learning an input-dependent model for path planning in neural network [11], [53]. Denoyer *et al.* [11] proposed an end-to-end learning framework to simultaneously learn successive representations and paths, where an extension of policy gradient method is introduced. Shazeer1 *et al.* [53] successfully applied the idea of mixtures of experts in nature language processing applications with very large models. The method employs a gating network to select experts from thousands of candidates, which significantly accelerates large language model and achieve state-of-the-art performance. Apart from efficient inference, dynamic network is also adopted during training, such that information will be passed through fewer connections, providing faster training and stronger regularizer. Stochastic depth [22] randomly drops a subset of layers and bypasses them with the identity function during training. A random forward/backward training technology for two-path residual network [13] is proposed for deep neural network regularization, where hidden features are fed to random branches of residual block.

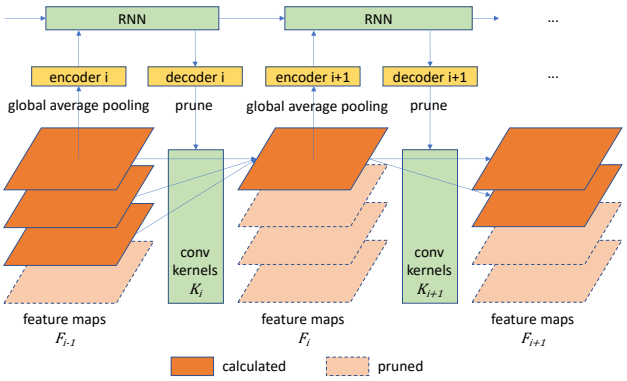

Fig. 2. The overall framework of our RNP. RNP consists of two sub-networks: the backbone CNN network and the decision network. The convolution kernels of backbone CNN network are dynamically pruned according to the output Q-value of the decision network, conditioned on the state forming from the last calculated feature maps.

3 Approach

In this section, we detail the proposed Runtime Network Routing (RNR) framework. We first present the Runtime Neural Pruning (RNP) method for adaptive channel pruning, which is built on the methodology of bottom-up and layer-by-layer Markov decision in an incremental manner. Then we show how to employ this idea to build efficient multi-path dynamic network architectures.

3.1 Runtime Neural Pruning

Fig. 2 shows the overall framework of our RNP method. Specifically, RNP consists of two sub-networks, the backbone CNN network and the decision network, which decides how to prune the convolution kernels conditioned on the input image and current feature maps. Since convolutional layers are the most computationally dense layers in a CNN, we focus on the pruning of convolutional layers in this work, leaving fully connected layers as a classifier.

3.1.1 Bottom-up Runtime Pruning

We denote the backbone CNN with m convolutional layers as C, with convolutional layers denoted as $C_1; C_2; :::; C_m$, whose kernels are $K_1; K_2; :::; K_m$, respectively, with the size of $n_i \quad n_{i\ 1} \quad k \quad k; i = 1; 2; :::; m$, where $k, n_{i\ 1}$ and

n_i are kernel size, the numbers of input and output channels of $\mathbf{K}_i$ respectively. These convolution layers produce feature maps $\mathbf{F}_i \in \mathbb{R}^{n_i \times H \times W}; i = 1; 2; :::; m$ by applying corresponding filters $\mathbf{K}_i^j \in \mathbb{R}^{n_{i-1} \times k \times k}; j = 1; 2; :::n_i$ on the input n_{i-1} channels, where one output channel is generated by one filter. Given feature maps $\mathbf{F}_i, i = 1; 2; :::; m - 1$, our method is designed to find and prune the redundant output feature maps in $\mathbf{F}_{i+1}$ and corresponding convolutional filters in $\mathbf{K}_{i+1}$ to reduce computation and achieve maximum performance simultaneously. Note that here we focus on filter-level pruning instead of arbitrary kernel-pruning to achieve actual speed up during inference, so our method prunes certain filters and related kernels at the same time.

Taking the i-th layer as an example, we denote our goal as the following objective [1]:

$$\min_{\mathbf{K}_{i+1},h} \mathbb{E}_{\mathbf{F}_i}[L_{cls}(\text{conv}(\mathbf{F}_i; \mathbf{K}_{i+1}[h(\mathbf{F}_i)])) + L_{pnt}(fh(\mathbf{F}_k)g_{k=1}^{i})]; \tag{1}$$

where L_{cls} is the loss of the classification task, L_{pnt} is the penalty term that reflects the computional cost of the i-th layer, which also represents the tradeoff between the speed and the accuracy, $h(\mathbf{F}_i)$ is the conditional pruning unit that produces a list of indexes of selected output filters according to the input feature map, $K[\,]$ is the indexing operation for filter pruning and $\text{conv}(x_1; x_2)$ is the convolutional operation for the input feature map x_1 and kernel x_2. Note that our framework infers through standard convolutional layers after filter-level pruning, which can be easily boosted by utilizing GPU-accelerated neural network library such as cuDNN [10].

To solve the optimization problem in (1), we divide the whole problem into two sub-problems of $f\mathbf{K}g$ and h: 1) optimizing backbone CNN with given filter selections and 2) find optimal filter selections conditioned on the input image with given CNN , and adopt an alternate training strategy to solve each sub-problem independently with the neural network optimizer such as RMSprop [64].

For an input sample and a m-layer CNN, we need to decide which filters should be pruned for each layer, in other words, there are totally m decisions of pruning to be made. In the proposed framework, we employ a recurrent network to model the decision procedure and produce these decisions. In order to learn this network, a straightforward idea is to using the optimized decisions under certain penalties to supervise the decision network. However, for a backbone CNN with m layers, the time complexity of collecting the supervised signal is $O(\prod_{i=1}^{m} n_m)$, which is NP-hard and unacceptable for prevalent very deep architecture such as VGG [54] and ResNet [3]. To simplify the training problem, we employ the following two strategies: 1) modeling the network pruning as a Markov decision process (MDP) [49] and train the decision network by reinforcement learning, and 2) redefining the procedure of pruning to reduce the number of possible decisions, which is described as below.

1. Since the first convolution layer has a small number of input channels that contributes little to the total computational cost and it is difficult to determine the importance of a filter according to the input image, we do not apply our pruning method on this layer.

3.1.2 Layer-by-layer Markov Decision Process

The decision network consists of an encoder-RNN-decoder structure, where the encoder E_i embeds the feature map $\mathbf{F}_i$ into fixed-length code, RNN R aggregates codes from previous stages, and the decoder D_i outputs the Q-value of each action. We formulate key elements in Markov decision process (MDP) based on the decision network to adopt deep Q-learning in our RNP framework as follows.

State: Given feature map $\mathbf{F}_i$, we first extract a dense feature embedding $p_{\mathbf{F}_i}$ with global average pooling, as commonly conducted in [12], [51], whose length is n_i. Since the number of channels for different convolutional layers are different, the length of $p_{\mathbf{F}_i}$ varies. To address this, we use the encoder E_i (a fully connected layer) to project the pooled feature into a fixed-length embedding $E_i(p_{\mathbf{F}_i})$. $E_i(p_{\mathbf{F}_i})$ from different layers are associated in a bottom-up way with a RNN structure, which produces a latent code $R(E_i(p_{\mathbf{F}_i}))$, regarded as embedded state information for reinforcement learning. The decoder (also a fully connected layer) produces the Q-value for decision.

Action: The actions for each pruning are defined in an incremental way. For the convolution kernel $\mathbf{K}_i$ with n_i output channels, we determine which output channels to be calculated and which to be pruned. To simplify the process, we group the output feature maps into k sets, denoted as $\mathbf{F}_1^0; \mathbf{F}_2^0; :::; \mathbf{F}_k^0$. One extreme case is $k = n_i$, where one single output channel forms a set. The actions $a_1; a_2; :::; a_k$ are defined as follows: taking actions a_i yields calculating the feature map groups $\mathbf{F}_1^0; \mathbf{F}_2^0; :::; \mathbf{F}_i^0; i = 1; 2; :::; k$. Hence the feature map groups with lower index are calculated more, and the higher indexed feature map groups are calculated only when the sample is difficult enough. Specially, the first feature map group is always calculated, which we mention it as base feature map group. Since we do not have state information for the first convolutional layer, it is not pruned, with totally $m - 1$ actions to take.

While the definitions of actions are rather simple, one can easily extend the definition for more complicated network structures. Like Inception [61] and ResNet [3], we define the action based on unit of a single block by sharing pruning rate inside the block, which is more scalable and can avoid considering about the sophisticated structures.

Reward: The reward of each action taken at the t-th step with the action a_i is defined as:

$$r_t(a_i) = \begin{cases} L_{cls} + (i - 1)\, p; & t = m - 1; \\ (i - 1)\, p; & t < m - 1 \end{cases} \tag{2}$$

where the agent gets penalty according to computational complexity after every decision and gets rewards according to classification loss when inference terminates, and p is a negative penalty that can be manually set. The reward was set according to the loss for the original task. We took the negative loss L_{cls} as the final reward so that if a task is completed better, the final reward of the chain will be higher, *i.e.*, closer to 0. is a hyper-parameter to rescale L_{cls} into a proper range, since L_{cls} varies a lot for different network structures and different tasks. Taking actions that calculate more feature maps, *i.e.*, with higher i, will bring higher penalty due to more computations. For $t = 1; :::; m - 2$, the reward is only about the computation penalty, while

Algorithm 1 : Runtime neural pruning for solving optimization problem (1)

Input: training set with labels fXg
Output: backbone CNN C, decision network D

initialize: train C in normal way or initialize C with pre-trained model
for i $1; 2; :::; M$ **do**
  // train decision network
  for j $1; 2; :::; N_1$ **do**
    Sample random minibatch from fXg
    Forward and sample -greedy actions $fs_t; a_tg$
    Compute corresponding rewards fr_tg
    Backward Q values for each stage and generate r L_{re}
    Update using r L_{re}
  end for
  // fine-tune backbone CNN
  for k $1; 2; :::; N_2$ **do**
    Sample random minibatch from fXg
    Forward and calculate L_{cls} after runtime pruning by D
    Backward and generate $r_C L_{cls}$
    Update C using $r_C L_{cls}$
  end for
end for
return C and D

at the last step, the chain will get a final reward of L_{cls} to assess how well the pruned network completes the task.

The key step of the Markov decision model is to decide the best action at certain state. In other words, it is to find the optimal decision policy. By introducing the Q-learning method [44], [66], we define $Q(a_i; s_t)$ as the expectation value of taking action a_i at state s_t. So the policy is defined as $= \text{argmax}_{a_i} Q(a_i; s_t)$.

Therefore, the optimal action-value function can be written as:

$$Q(s_t; a_i) = \max \mathrm{E}[r_t + r_{t+1} + {}^2 r_{t+2} + :::j \ ]; \quad (3)$$

where is the discount factor in Q-learning, providing a tradeoff between the immediate reward and the prediction of future rewards. We use the decision network to approximate the expected Q-value $Q(s_t; a_i)$, with all the decoders sharing parameters and outputting a k-length vector, each representing the Q of corresponding action. If the estimation is optimal, we will have $Q(s_t; a_i) = Q(s_t; a_i)$ exactly.

According to the Bellman equation [3], we adopt the squared mean error (MSE) as a criterion for training to keep decision network self-consistent. Hence, we rewrite the objective for sub-problem of h in optimization problem 1 as:

$$\min L_{re} = \mathrm{E}[r(s_t; a_i) + \max_{a_i} Q(s_{t+1}; a_i) \quad Q(s_t; a_i)]^2; \quad (4)$$

where is the weights of decision network. In our proposed framework, a series of states are created for an given input image. And the training is conducted using -greedy strategy that selects actions following with probability and select random actions with probability 1 , while inference is conducted greedily. The backbone CNN network and decision network is trained alternately.

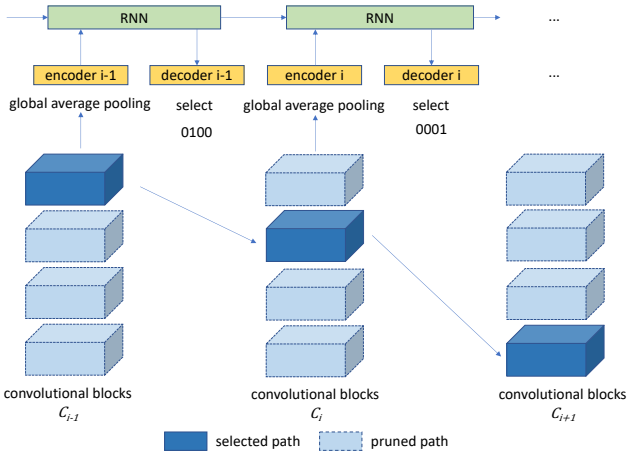

Fig. 3. Overall framework of our runtime routing for multi-path network. A decision network is designed to decide which path to take in a multi-path CNN conditioned on the output of previous layer.

Algorithm 1 details the training procedure of the proposed method, where alternate training is performed for M iterations and we update decision network N_1 times and backbone CNN N_2 times in each iteration.

It is worth noticing that during the training of agent, we manually set a fixed penalty for different actions and reach a balance status. While during deployment, we can adjust the penalty by compensating the output Q of each action with relative penalties accordingly to switch between different balance point of accuracy and computation costs, since penalty is input-independent. Thus one single model can be deployed to different systems according to the available resources.

3.2 Runtime Routing for Multi-Path Network

While RNP is designed for pruning neural network at runtime, our network routing method can be extended to a more generic framework that selects optimal path inside the network during inference conditioned on the input image. Therefore, we design a new method to build efficient multi-path dynamic network based on network ensemble.

Network ensemble is a widely used technique that has been proven to be effective in many computer vision tasks such as image classification [19], [62], object detection [24], [38] and semantic segmentation [38]. Prediction ensemble used in [19], [38], [62] can be regarded as a special case of multi-path CNN that each path of CNN infers independently. Except network ensemble, multi-path CNN is also used as a regularization method to boost classification performance of deep neural networks. A random forward/backward training technology for two-path residual network [13] is proposed for deep neural network regularization, where hidden features are fed to random branches of residual block and gradients updates parameters of random previous branches. In this work, we formulate the framework of multi-path CNN and propose a runtime network routing method to select an optimal path inside the multi-path, which extends our RNP method from channel-wise incremental routing to network path dynamic routing scenario.

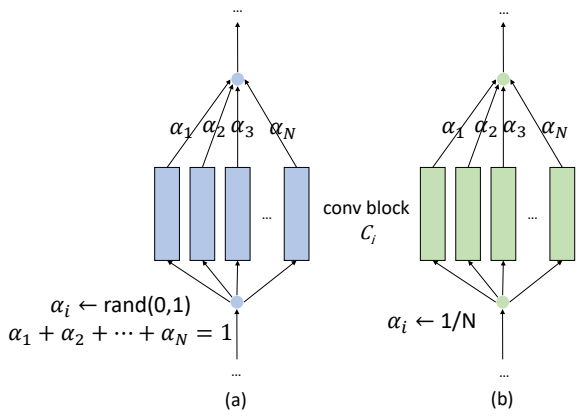

Fig. 4. Different phases of multi-path CNN. (a) is the training phase and (b) is the testing phase.

The overall framework of our runtime routing for multi-path network is shown in Fig. 3. Similar with the above-mentioned RNP method, a decision network is also designed to decide which path to take in a multi-path CNN conditioned on the output of the previous layer. The multi-path CNN network can be a simple extension of any kinds of CNN structure. To achieve a better trade-off between the speed and the accuracy, the widths of convolutional layers in multi-path network are reduced accordingly for different path number.

3.2.1 Multi-Path CNN

For ease of implementation, we assume that multi-path CNN consists of several CNN with the same structure in this work. Let $\mathsf{C} = [C^1; C^2; :::; C^N]$ be a multi-path CNN that has N branches of CNN, where $C^1; C^2; :::; C^N$ are N CNNs that have the same network architecture. C^n has m convolutional blocks $C_1^n; C_2^n; :::; C_m^n$. Taking the i-th block in multi-path CNN as an example, for input hidden features $F_{i\ 1}$, the output of i-th block can be written as:

$$F_i = \ (C_i^1(F_{i\ 1}); C_i^2(F_{i\ 1}); :::; C_i^N(F_{i\ 1})); \qquad (5)$$

where $\ $ is an aggregation function that takes outputs of M branches in i-th block as input and produce an aggregated feature map that has the summarized information from different branch. In this work, we employ a linear interpolation operator to aggregate the outputs of M branches as follows:

$$F_i = \ (F_i^1; F_i^2; :::; F_i^N) = \ _1F_i^1 + \ _2F_i^2 + ::: + \ _NF_i^N; \quad (6)$$

and

$$_1; \ _2; :::; \ _N > 0; \ _1 + \ _2 + ::: + \ _N = 1: \qquad (7)$$

During training, we set $f\ _ig$ as generated random real numbers for each mini-batch following [13], and we set $\ _i$ to $\frac{1}{N}$ for all $1 \quad i \quad N$ during testing, which is illustrated in Fig. 4.

3.2.2 Bottom-up Runtime Branch Routing

After training, we get a multi-path CNN $\mathsf{C} = [C^1; C^2; :::; C^N]$. Inference through all the M branches and calculate mean could obtain high performance as mentioned in [13]. However, it would cost great computational burdens. Our goal is to inference through one single branch in each block while maintain the highest possible performance, which will cut the computation by M times.

We denote our goal as the following objective:

$$\min_{a_1;a_2;:::;a_n} \mathrm{E}[L_{cls}(C_m^{a_N}(:::(C_2^{a_2}(C_1^{a_1}(X)))))]; \qquad (8)$$

$$a_i = h(F_{i\ 1}) = C_i^{a_i}{}_1^1(:::(C_2^{a_2}(C_1^{a_1}(X)))) \qquad (9)$$

where L_{cls} is the loss of the classification task, X is the input image to be classified. $h(F_i)$ is the conditional pruning unit that produces a list of indexes of selected kernels according to input feature map, and here we used global average pooling for state extraction.

Same as above RNP framework, we model the network pruning as a Markov decision process and train the decision network by reinforcement learning. One of the major difference is that we used Policy Gradient [59] for training, since all the actions requires the same computations, and the only difference is the final output loss. The reward setting is related to the whole trajectory, which is suitable with Policy Gradient algorithm.

Since the **state** is the same as RNP, which is the embedded vectors encoded by global average pooling operation, encoder layer and RNN in decision network, here we focus on the formulation of **action** and **reward**.

Action: The actions are defined based on stages of network. At each stage, we choose one of the identical branches for inference. For a block with m branches, there are m actions $a_1; a_2; :::; a_m$ in total. Taking a_i yields inference through branch i, while other branches are not calculated.

The definitions of actions are rather simple but extremely effective. It does not care about the inner-block structure of CNNs, which could be very complicated, like Inception [61] and ResNet [3]. As long as several branches with same structure are present, we can apply our RNR framework to accelerate the inference.

Reward: Since all the m branches share the same structure and computational complexity, the reward function only contains classification loss L_{cls}. We define the reward function over the whole trajectory T as

$$r(T) = \ L_{cls} \qquad (10)$$

The reward was set according to the loss for the original task. We took the negative loss $\ L_{cls}$ as the final reward so that if a task is completed better, the final reward of the trajectory will be higher, *i.e.*, closer to 0.

The key step of the Markov decision model is to decide the best action at certain state. In other words, it is to find the optimal decision policy. By introducing the Policy Gradient method, we define $Q(a_i; s_t)$ as the expectation value of taking action a_i at state s_t. So the policy is defined as $= \mathrm{argmax}_{a_i}\mathrm{Policy}(a_i; s_t) = \mathrm{argmax}_{a_i} \ (a_i js; \ )$.

The general idea behind the Policy Gradient method is to increase the possibility of favorable actions while reducing the unfavorable ones. The parameters in decision network can be learned as follows:

$$+ \quad r \ \log \ \ (s_t; a_t)r: \qquad (11)$$

However, it can be inefficient to train the decision network following Equation 11 in practice, since the variance of the

Algorithm 2 : Runtime network routing for solving optimization problem (8)

Input: training set with labels $\{X\}$
Output: Multi-path CNN C, decision network D

initialize: train C according to Section 3.2.1.
for $i = 1, 2, \ldots, M$ **do**
  // train decision network Fix multi-path CNN C
  for $j = 1, 2, \ldots, N_1$ **do**
    Sample random minibatch from $\{X\}$
    Forward and sample actions with distribution $h(F_i)$ layer-by-layer
    Compute corresponding rewards of trajectory $\{r_t = r(T)\}$
    Update using Equation 12
  end for
  // fine-tune backbone CNN Fix decision network D
  for $k = 1, 2, \ldots, N_2$ **do**
    Sample random minibatch from $\{X\}$
    Forward and calculate L_{cls} after runtime network routing by D
    Backward and generate $r_C L_{cls}$
    Update C using $r_C L_{cls}$
  end for
end for
return C and D

estimated gradient can be large and leads to unstable training. Therefore, the policy gradient method can be further generalized to compute the reward associated with an action value relative to a reference reward or **baseline** b. During training, we adopt the following update equations

$$+ \quad r \quad \log \quad (s_t; a_t)(r \quad b) \tag{12}$$

where is the weights of decision network, is the update rate, r is the reward. The baseline b can be any function that dose not depend on action, because:

$$\begin{aligned} E_{s_t;a_t}[b\, r \quad \log \quad (s_t; a_t)] &= b\, r \quad E_{s_t;a_t}[\log \quad (s_t; a_t)] \\ &= b\, r \quad 1 \\ &= 0. \end{aligned} \tag{13}$$

(13) shows that adding the baseline b will not change the expectation of estimated gradient, but it can reduce the variance of gradient estimate.

In our implementation, we use the reward of the whole trajectory, i.e. $r = r(T) = \quad L_{cls}$. b is a baseline for reward r, and we use moving average to update b as:

$$b \qquad b + (1 \qquad) \quad r \tag{14}$$

In our experiments, we set $= 0.99$.

Algorithm 2 details the training procedure of the proposed method, where alternate training is performed for M iterations and we update decision network N_1 times and backbone CNN N_2 times in each iteration.

4 Experiments

We conducted image classification experiments to evaluate our RNP and RNR on three widely used object image datasets including CIFAR-10, CIFAR-100 [30] and ILSVRC2012 [52].

For the RNP method, we trained a network of four convolutional layers with 3 3 kernels and reported experimental results on the CIFAR-10. For CIFAR-100, we used the VGG-16 network for evaluation. For ILSVRC2012, we report the results of both VGG-16 and ResNet-50. For results on the CIFAR dataset, we compared the results obtained by our RNP and naive channel reduction methods. For results on the ILSVRC2012 dataset, we compared the results achieved by our RNP with recent state-of-the-art network pruning methods.

For the RNR method, we trained three different ResNet architectures: ResNet-18, ResNet-34, ResNet-50, and reported the results on both CIFAR and ILSVRC2012 datasets with several different settings.

The details of networks used in our experiments are summarized in Table 1.

4.1 Datasets and Protocols

We evaluated the proposed RNP and RNR method on three widely used datasets. Here we give a brief description of these datasets.

CIFAR-10 & CIFAR-100: The CIFAR-10 and CIFAR-100 dataset consist of 60000 32 32 colour images. The CIFAR-10 dataset has 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images. The test set contains exactly 1000 randomly-selected images from each class. The training set contain the remaining images in random order. The CIFAR-100 dataset is an extension of the CIFAR-10, which has 100 classes containing 600 images each. There are 500 training images and 100 testing images per class.

ILSVRC2012: ImageNet is a very large dataset that contains over 15 million labeled high-resolution images belonging to roughly 22,000 categories. The images were collected from the web and labeled by human labelers. The ImageNet Large-Scale Visual Recognition Challenge(ILSVRC) is a part of Pascal Visual Object Challenge, which is an influential competition held annually. ILSVRC is a relatively small subset of ImageNet that has 1000 categories and roughly 1000 images in each category. In the ILSVRC dataset, there are roughly 1.2 million training images, 50,000 validation images, and 150,000 testing images. Since the test set labels are not available for ILSVRC2012, we evaluated our method on the validation set of ILSVRC2012 following previous works, which is not used during training or hyper-parameter searching.

4.2 Experiments on RNP

In this section, we present the detailed information about our evaluations on proposed RNP method, including 1) implementation details, 2) intuitive experiments, and 3) results on three datasets.

4.2.1 Implementation Details

We trained RNP in an alternative manner, where the backbone CNN network and the decision network were trained iteratively. To make the training converge faster, we first re-rank filters in pre-trained model according to their l_1-norm

TABLE 1
The architecture of models used in our experiments. For ResNet, we show that the building blocks in brackets with the numbers of blocks stacked following [19], and downing sampling is performed in the first convolutional layer of stage 3, stage 4 and stage 5 with a stride of 2.

	4-layer CNN	VGG-16 CIFAR	VGG-16 ImageNet	ResNet-f18,34g	ResNet-50
stage 1		3 3, 64 3 3, 64 maxpool	3 3, 64 3 3, 64 maxpool	3 3, 64	7 7, 64, stride 2 maxpool
stage 2	3 3, 32	3 3, 128 3 3, 128 maxpool	3 3, 128 3 3, 128 maxpool	$\begin{bmatrix} 3 & 3; 64 \\ 3 & 3; 64 \end{bmatrix}$ $f2; 3g$	$\begin{bmatrix} 1 & 1; 64 \\ 3 & 3; 64 \\ 1 & 1; 256 \end{bmatrix}$ 3
stage 3	3 3, 32 maxpool	3 3, 256 3 3, 256 3 3, 256 maxpool	3 3, 256 3 3, 256 3 3, 256 maxpool	$\begin{bmatrix} 3 & 3; 128 \\ 3 & 3; 128 \end{bmatrix}$ $f2; 4g$	$\begin{bmatrix} 1 & 1; 128 \\ 3 & 3; 128 \\ 1 & 1; 512 \end{bmatrix}$ 4
stage 4	3 3, 64 maxpool	3 3, 512 3 3, 512 3 3, 512 maxpool	3 3, 512 3 3, 512 3 3, 512 maxpool	$\begin{bmatrix} 3 & 3; 256 \\ 3 & 3; 256 \end{bmatrix}$ $f2; 6g$	$\begin{bmatrix} 1 & 1; 256 \\ 3 & 3; 256 \\ 1 & 1; 1024 \end{bmatrix}$ 6
stage 5	3 3, 64 maxpool	3 3, 512 3 3, 512 3 3, 512 maxpool	3 3, 512 3 3, 512 3 3, 512 maxpool	$\begin{bmatrix} 3 & 3; 256 \\ 3 & 3; 256 \end{bmatrix}$ $f2; 3g$	$\begin{bmatrix} 1 & 1; 512 \\ 3 & 3; 512 \\ 1 & 1; 2048 \end{bmatrix}$ 3
	classifier				
# Parameters	0.6M	15M	138M	f11M, 21Mg	26M
FLOPs	1:7 10^7	3:1 10^8	1:5 10^{10}	f5:5 10^8, 1:1 10^9g	3:8 10^9

for networks without batch normalization layers as [36] or scaling factors in batch normalization layers as [42] in the descending order before we assign each filter to a group. The backbone CNN is then initialized with random pruning, where we selected random number of filter groups uniformly for each convolution layer in the forward pass, i.e., decisions were randomly made. During training, we fixed the CNN parameters and trained the decision network, regarding the backbone CNN as an environment, where the agent can take actions and get corresponding rewards. We fixed the decision network and fine-tuned the backbone CNN following the policy of the decision network, which helps CNN specialize in a specific task. The initialization was trained using SGD, with an initial learning rate 0:01, decay by a factor of 10 after 120, 160 epochs, with totally 200 epochs in total. For CIFAR-10 and CIFAR-100 datasets, a standard data augmentation method that consists of random cropping with padding and random horizontal flipping was applied. For the ILSVRC2012 dataset, we randomly cropped the resized 256 256 images to 224 224 as input and random horizontal flipping was also used during training. The other training progress was conducted using RMSprop [64] with the learning rate of 1e-6. For the -greedy strategy, the hyper-parameter was annealed linearly from 1.0 to 0.1 in the beginning and fixed at 0.1 thereafter.

For most experiments, we set the number of convolutional group to $k = 4$, which is a trade-off between the performance and the complicity. Increasing k will enable more possible pruning combinations, while at the same time making it harder for reinforcement learning with an enlarged action space. Since the action is taken conditioned on the current feature map, the first convolutional layer is not pruned, where we have totally m 1 decisions to make, forming a decision sequence. During the training, we set the penalty for extra feature map calculation as $p =$ 0:1. During inference, we compensated the output of decision network to $Q(a_i; s)$ $(i$ 1) p to achieve different speed-up solutions by adjusting the value of p. The scale factor was set such that the average L_{cls} is approximately 1 to make the relative difference more significant. In our implementation, we set the to $1=\hat{L}_{cls}$ where $\hat{L}_{cls}$ is the average classification loss over the past n epochs (e.g., $n = 10$) and update every n epochs to make the training process stable. For experiments with the VGG-16 model, we defined the actions based on unit of a single block by sharing pruning rate inside the block as mentioned in Section 3.1.2 to simplify implementation and accelerate convergence. For experiments with the ResNet-50 model, we pruned the first and the second convolution layers in a residual block with the same pruning ratio and did not prune the output feature maps in the last convolution layer to keep the consistency between the projected mapping and the residual mapping.

For vanilla baseline methods comparison on the CIFAR datasets, we evaluated the performance of normal neural network with the same computations. More specifically, we calculated the average number of multiplications of every convolution layer and rounded it up to the nearest number of channels sharing same computations, which resulted in an identical network topology with reduced convolutional channels. We trained the vanilla baseline network with the SGD until convergence for comparison. All our experiments

were implemented using the modified Caffe toolbox [27].

4.2.2 Intuitive Experiments

To have an intuitive understanding of our framework, we first conducted a simple experiment to show the effectiveness and undergoing logic of our RNP. We considered a 3-category classification problem, consisting of male faces, female faces and background samples. It is intuitive to think that separating male faces from female faces is a much more difficult task than separating faces from background, needing more detailed attention, so more resources should be allocated to face images than background images. In other words, a good tradeoff for RNP is to prune the neural network more when dealing with background images and keep more convolutional channels when inputting a face image.

To validate this idea, we constructed a 3-category dataset by using the Labeled Faces in the Wild [23] dataset, which we referred to as LFW-T. More specifically, we randomly cropped 3000 images for both male and female faces, and also 3000 background images randomly cropped from LFW. We used the attributes from [32] as labels for male and female faces. All these images were resized to 32 32 pixels. We held out 2000 images for testing and the remaining for training. For this experiment, we designed a 3-layer convolutional network with two fully connected layers. All convolutional kernels are 3 3 and with 32, 32, 64 output channels respectively. We followed the same training protocol as mentioned above with $p = 0.1$, and focused on the difference between different classes.

The original network achieved 91.1% recognition accuracy. By adjusting the penalty, we managed to get a certain point of accuracy-computation tradeoff, where computations (multiplications) were reduced by a factor of 2, while obtaining even slightly higher accuracy of 91.75%. We looked into the average computations of different classes by counting multiplications of convolutional layers. The results were shown in Fig. 5. For the whole network, RNP allocated more computations on faces images than background images, at approximately a ratio of 2, which clearly demonstrates the effectiveness of RNP. However, since the first convolutional layers and fully connected layers were not pruned, to get the absolute ratio of pruning rate, we also studied the pruning of a certain convolutional layer. In this case, we selected the last convolutional layer `conv3`. The results are shown on the right figure of Fig. 5. We see that for this certain layer, computations for face images are almost 5 times of background images. The differences in computations show that RNP is able to find the relative difficulty of different tasks and exploit such property to prune the neural network accordingly.

4.2.3 Results

CIFAR-10 & CIFAR-100: For CIFAR-10 and CIFAR-100, we used a four-layer convolutional network and the VGG-16 network for experiments, respectively. The goal of these two experiments is to compare our RNP with vanilla baseline network, where the number of convolutional layers was reduced directly from the beginning. The fully connected layers of standard VGG-16 are too redundant for CIFAR-100, so we eliminated one of the fully connected layer and set the inner dimension as 512. The modified VGG-16 model was easier to converge and actually slightly outperformed the original model on CIFAR-100. The results are shown in Fig. 6. We see that for vanilla baseline method, the accuracy suffered from a stiff drop when computations savings were than 2.5 times. While our RNP consistently outperformed the baseline model, and achieved competitive performance even with a very large computation saving rate.

ILSVRC2012: We compared our RNP with recent state-of-the-art network pruning methods [26], [36], [43], [45], [69] on the ImageNet dataset using the VGG-16 model, including recent filter pruning methods [36], [43], [45]. We evaluated the *top-5 error* using single-view testing on ILSVRC2012-val set and trained RNP model using ILSVRC2012-train set. The view was the center 224 224 region cropped from the resized images whose shorter side is 256 by following [69]. RNP was fine-tuned based on the public available model [2] which achieves 10.1% top-5 error on ILSVRC2012-val set. Results are shown in Table 2, where *speed-up* is the theoretical speed-up ratio computed by the complexity. For filter pruning method in [36], we set the same pruning rate for convolution layers in the same stage according to layer sensitivity and measure the top-5 error after 20 epoch re-training as described in their paper. We see that RNP achieves similar performance with a relatively small speed-up ratio with other methods and outperforms other methods by a significant margin with a large speed-up ratio. We further conducted our experiments on larger ratio (10) and found RNP only suffered slight drops (1.31% compared to 5), far beyond others' results on 5 setting.

4.2.4 Analysis

Analysis of Feature Maps: Since we define the actions in an incremental way, the convolutional channels of lower index are calculated more (a special case is the base network that is always calculated). The convolutional groups with higher index are increments to the lower-indexed ones, so the functions of different convolution groups might be similar to "low-frequency" and "high-frequency" filters. We visualized different functions of convolutional groups by calculating average feature maps produced by each convolutional group. Specially, we took CIFAR-10 as an example and visualized the feature maps of `conv2` with $k = 4$. The results are shown in Fig. 7.

From the figure, we see that the base convolutional groups have highest activations to the input images, which can well describe the overall appearance of the object. While convolutional groups with higher index have sparse activations, which can be considered as a compensation to the base convolutional groups. So the undergoing logic of RNP is to judge when it is necessary to compensate the base convolutional groups with higher ones: if tasks are easy, RNP will prune the high-order feature maps for speed, otherwise bring in more computations to pursue accuracy.

Runtime Analysis: One advantage of our RNP is its convenience for deployment, which makes it easy to harvest actual computational time savings. Therefore, we measured the actual runtime under GPU acceleration, where we measured the actual inference time and top1/top-5 classification

2. http://www.robots.ox.ac.uk/~vgg/research/very_deep/

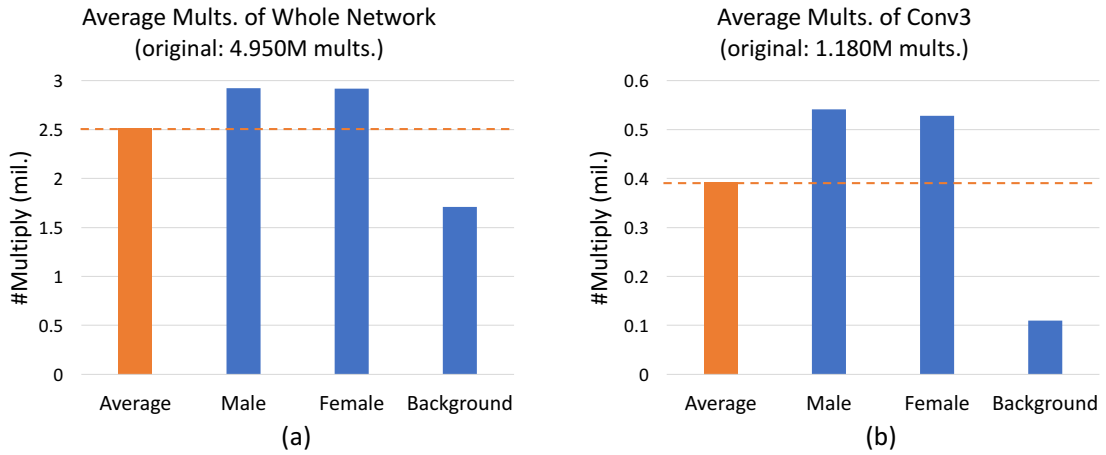

Fig. 5. The average multiplication numbers of different classes in our intuitive experiment. We show the computation numbers for both the whole network (on the left) and the fully pruned convolutional layer conv3 (on the right). The results show that RNP succeeds to focus more on faces images by preserving more convolutional channels while prunes the network more when dealing with background images, reaching a good tradeoff between accuracy and speed.

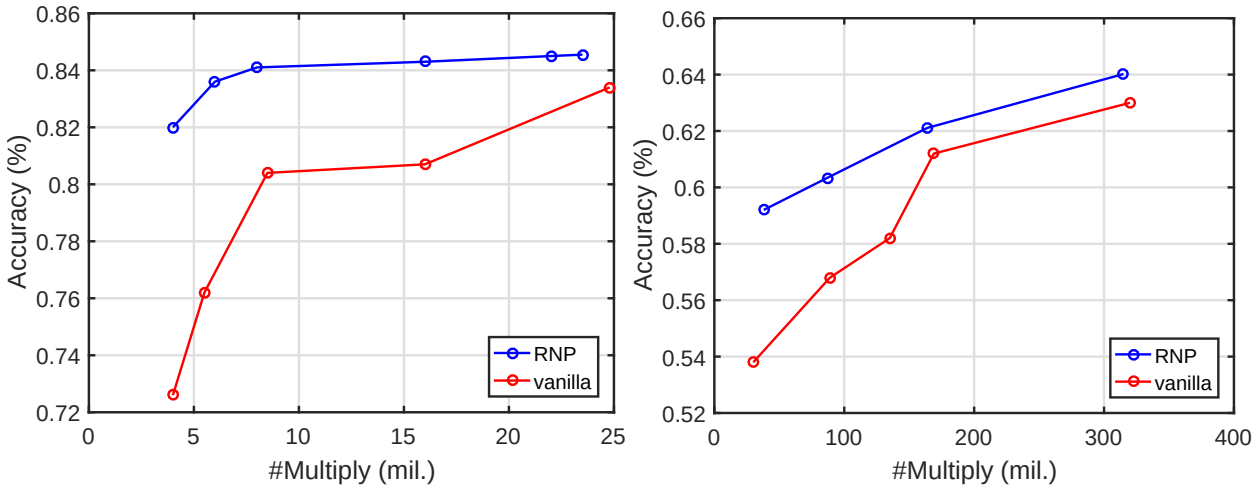

Fig. 6. The results on CIFAR-10 (on the left) and CIFAR-100 (on the right). For vanilla curve, the rightmost point is the full model and the leftmost is the $\frac{1}{4}$ model. RNP outperforms naive channel reduction models consistently by a very large margin.

accuracy for VGG-16 and ResNet-50 on the ILSVRC2012-val set. Inference time were measured on a Titan X (Pascal) GPU with batch size 64. Table 3 shows the GPU inference time of different settings. We see that our RNP generalizes well on GPU.

4.3 Experiments on RNR

In this section, we first present the detailed information about our implementations. Then we show the experimental results of proposed RNR method. Lastly, we further analyzed the computational time of RNR.

4.3.1 Implementation Details

Similar with the RNP method, we trained our RNR in an alternative manner by following the descriptions in **Algorithm 2**. To accelerate the training procedure of decision network and backbone multi-path CNN, we conducted 100 epochs hard selection training before RNR training, where $f_i g$ are randomly generated as one-hot vectors. The multi-path CNN backbone model was trained using SGD, with a learning rate of 0.1 which is decreased by 2 every 25 epochs, with 300 epochs standard training and 100 epochs hard selection training. The decision network was trained using Adam optimizer [29] with a learning rate of 1e-5. We set the initial b value as training set accuracy. Other training details are as in Section 4.2.1. Our experiments for RNR were implemented using open source software PyTorch [48].

4.3.2 Results

We evaluated our proposed RNR method on ResNet-18 and ResNet-34 networks. Our proposed multi-path CNN and runtime routing method were compared in serveral different settings, including different numbers of branches (1, 2 or 4), different numbers of channel (1/2 or 1/4 of original model were denoted as "ResNet 1 ", "ResNet 1/2 " and "ResNet 1/4 " respectively). Furthermore, we compared our proposed method with 1-branch models that have the same computational complexity as our models. The detailed information about our experiments is described as follows.

Comparisons with Original Models: Experimental results with the ResNet-18 network are summarized in Ta-

TABLE 2
Comparisons of increase of top-5 error on ILSVRC2012-val (%) for VGG-16 with recent state-of-the-art methods, where we used 10.1% top-5 error baseline as the reference.

Speed-up (in FLOPs)	3	4	5	10
Jaderberg *et al.* [26] ([69]'s implementation)	2.3	9.7	29.7	-
Asymmetric [69]	-	3.84	-	-
Filter pruning [36] (our implementation)	3.2	8.6	14.6	-
Taylor expansion [45]	2.3	4.8	-	-
ThiNet [43]	1.98	-	-	7.94
Ours	2.32	3.23	3.58	4.89

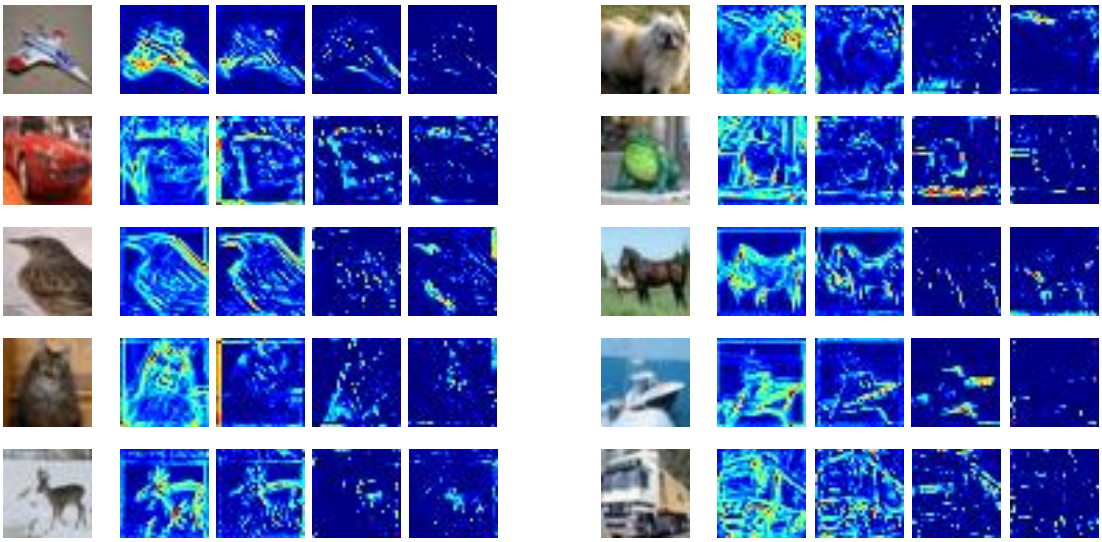

Fig. 7. Visualization of the original images and the feature maps of four convolutional groups, respectively. The presented feature maps are the average of corresponding convolutional groups.

TABLE 3
The increase of top-1/top-5 error (%) and GPU inference time (ms) under different theoretical speed-up ratios on the ILSVRC2012-val set. The Inference time includes runtimes of both backbone CNN and decision network.

Speed-up solution	top-1/top-5 err.	Inference time
VGG-16 (1)	0/0	3.26 (1:0)
RNP-VGG-16 (3)	2.98/2.32	1.38 (2:3)
RNP-VGG-16 (4)	4.01/3.23	1.07 (3:0)
RNP-VGG-16 (5)	4.88/3.58	0.880 (3:7)
RNP-VGG-16 (10)	6.12/4.89	0.554 (5:9)
ResNet-50 (1)	0/0	2.54 (1:0)
RNP-ResNet-50 (2)	2.90/2.14	1.94 (1:31)
RNP-ResNet-50 (3)	5.21/3.66	1.68 (1:51)

ble 4. We see that multi-path can significantly improve the recognition performance for the original ResNet-18 network and achieves relatively high performance even when computations savings were more than 4 times. By applying the proposed runtime routing method, models can be further accelerated to 16 times, and RNR method can still achieve very competitive performance even with such large speed-up rate.

Comparisons with Vanilla Models: To demonstrate the effectiveness of the proposed method, we conducted ablation experiments where 1-branch models that have the same complexity were trained as the baselines of runtime routing models. Results are summarized in Table 5, where we evaluated our method for different speed-up solutions (4 and 16). We see that runtime routing models can consistently outperform vanilla 1-branch models. As the number of branch growing, the image classification accuracy grows as well. It can be observed that performance of our method can be improved through adding more branches, which will lead to larger model size, but importantly, constant computational cost.

4.3.3 Runtime Analysis

Compared to the RNP method, RNR model is composed of several relatively independent modules. Hence, our RNR method can harvest even more actual computational time savings. Table 6 and presents Table 7 the mean inference time measured on GPU on both CIFAR-100 and ILSVRC2012 datasets for different ResNet networks . We see that RNR

TABLE 4
Experimental results on the CIFAR-10 and CIFAR-100 dataset with ResNet-18 for our proposed RNR method. We denote the inference method that calculate means of M branchs as "mean" and runtime routing method as "routing".

Model	# Branch	Method	CIFAR-10 accuarcy (%)	CIFAR-100 accuarcy (%)	Speed Up (in FLOPs)
ResNet-18	1	baseline	94.72	77.12	1.0
ResNet-18 1	2	Mean	95.36	78.85	0.5
	2	Routing	94.81	78.05	1.0
	4	Mean	**95.65**	**79.41**	0.25
	4	Routing	94.98	78.42	1.0
ResNet-18 1/2	2	Mean	94.65	76.40	2.0
	2	Routing	94.38	74.89	4.0
	4	Mean	94.95	77.21	1.0
	4	Routing	94.51	75.23	4.0
ResNet-18 1/4	2	Mean	93.41	71.75	8.0
	2	Routing	92.87	69.01	16.0
	4	Mean	93.95	73.81	4.0
	4	Routing	93.14	70.87	16.0

TABLE 5
Comparisons of the image classification accuracy (%) on the CIFAR-10 and CIFAR-100 with vanilla speed-up solutions for ResNet-18 and ResNet-34. We denote the proposed runtime routing method as "routing".

Network	Speed-up (in FLOPs)	Method	# Branch	CIFAR-10 accuarcy (%)	CIFAR-100 accuarcy (%)
ResNet-18	1	basline	1	94.72	77.12
ResNet-18	4	vanilla	1	94.34	73.85
ResNet-18		routing	2	94.38	74.89
ResNet-18		routing	4	94.51	75.23
ResNet-18	16	vanilla	1	92.25	68.14
ResNet-18		routing	2	92.87	69.01
ResNet-18		routing	4	93.14	70.87
ResNet-34	1	basline	1	95.42	78.01
ResNet-34	4	vanilla	1	94.49	75.69
ResNet-34		routing	2	94.62	76.32
ResNet-34		routing	4	94.75	76.99
ResNet-34	16	vanilla	1	92.99	71.73
ResNet-34		routing	2	93.21	73.01
ResNet-34		routing	4	93.43	73.72

can easily obtain actual speed-up rates that are very close with theoretical speed-up rates with standard hardware environment and prevalent deep learning libraries.

5 Conclusion

In this paper, we have proposed a generic Runtime Network Routing (RNR) framework for deep neural network compression. Unlike existing static neural network acceleration methods, our RNR selects an optimal path inside the network and preserves the full ability of the original large network. To achieve the idea of dynamic routing, we have modeled the network compression problem as a bottom-up, layer-by-layer Markov decision process, and employed reinforcement learning to train the network. The decision determined the estimated reward of each sub-path and conducts routing conditioned on different samples, where a faster path was taken when the image is easier for the task. Since the ability of network can be fully preserved, the balance point is easily adjustable according to the available resources. We have evaluated our method on both the multi-path residual network and the incremental convolutional channel pruning, and shown that RNR consistently outperforms static methods at the same computation complexity

TABLE 6
The GPU inference time under different theoretical speed-up ratios on the CIFAR-100 dataset. The Inference time includes runtimes of both backbone CNN and decision network.

Speed-up solution	err. (%)	Inference time (ms)
ResNet-18 (1)	0	2.68 (1:0)
ResNet-18 (4)	1.89	0.88 (3:0)
ResNet-18 (16)	6.25	0.279 (9:6)
ResNet-34 (1)	0	4.58 (1:0)
ResNet-34 (4)	1.02	1.43 (3:2)
ResNet-34 (16)	4.29	0.449 (10:2)

TABLE 7
The increase of top-1/top-5 error (%) and GPU inference time (ms) under different theoretical speed-up ratios on the ILSVRC2012-val set. The Inference time includes runtimes of both backbone CNN and decision network.

Speed-up solution	top-1/top-5 err.	Inference time
ResNet-50 (1)	0/0	1.56 (1:0)
ResNet-50 (4)	6.23/4.44	0.47 (3:3)

on both the CIFAR and ImageNet datasest. How to apply our proposed method to other computer vision applications such as object detection and semantic segmentation seems to be an interesting future work as efficient inference model is also desirable.

ACKNOWLEDGMENTS

This work is supported in part by the National Key Research and Development Program of China under Grant 2017YFA0700802, and the National Natural Science Foundation of China under Grants U1713214, 61672306, 61572271, and 61527808, and Shenzhen fundamental research fund (subject arrangement) under Grant JCYJ20170412170602564.

REFERENCES

[1] A. Almahairi, N. Ballas, T. Cooijmans, Y. Zheng, H. Larochelle, and A. Courville, "Dynamic capacity networks," *ICML*, 2016.
[2] S. Anwar, K. Hwang, and W. Sung, "Structured pruning of deep convolutional neural networks," *JETC*, 2017.
[3] R. Bellman, "Dynamic programming and lagrange multipliers," *PNAS*, vol. 42, no. 10, pp. 767–769, 1956.
[4] D. Benbouzid, R. Busa-Fekete, and B. Kégl, "Fast classification using sparse decision dags," *ICML*, 2012.
[5] E. Bengio, P.-L. Bacon, J. Pineau, and D. Precup, "Conditional computation in neural networks for faster models," *ICLRW*, 2016.
[6] Y. Bengio, N. Léonard, and A. Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation," *arXiv preprint arXiv:1308.3432*, 2013.
[7] T. Bolukbasi, J. Wang, O. Dekel, and V. Saligrama, "Adaptive neural networks for fast test-time prediction," *ICML*, 2017.
[8] J. C. Caicedo and S. Lazebnik, "Active object localization with deep reinforcement learning," in *ICCV*, 2015, pp. 2488–2496.
[9] S. Changpinyo, M. Sandler, and A. Zhmoginov, "The power of sparsity in convolutional neural networks," *arXiv preprint arXiv:1702.06257*, 2017.
[10] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer, "cudnn: Efficient primitives for deep learning," *arXiv preprint arXiv:1410.0759*, 2014.
[11] L. Denoyer and P. Gallinari, "Deep sequential neural network," *NIPS Workshop*, 2014.
[12] M. Figurnov, M. D. Collins, Y. Zhu, L. Zhang, J. Huang, D. P. Vetrov, and R. Salakhutdinov, "Spatially adaptive computation time for residual networks." in *CVPR*, vol. 2, no. 3, 2017, p. 7.
[13] X. Gastaldi, "Shake-shake regularization," *arXiv preprint arXiv:1705.07485*, 2017.
[14] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *ICLR*, 2016.
[15] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *NIPS*, 2015, pp. 1135–1143.
[16] S. J. Hanson and L. Y. Pratt, "Comparing biases for minimal network construction with back-propagation," in *NIPS*, 1989, pp. 177–185.
[17] B. Hassibi, D. G. Stork *et al.*, "Second order derivatives for network pruning: Optimal brain surgeon," *NIPS*, pp. 164–164, 1993.
[18] H. He, J. Eisner, and H. Daume, "Imitation learning by coaching," in *NIPS*, 2012, pp. 3149–3157.
[19] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016, pp. 770–778.
[20] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.
[21] H. Hu, R. Peng, Y.-W. Tai, and C.-K. Tang, "Network trimming: A data-driven neuron pruning approach towards efficient deep architectures," *arXiv preprint arXiv:1607.03250*, 2016.
[22] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Q. Weinberger, "Deep networks with stochastic depth," in *ECCV*. Springer, 2016, pp. 646–661.
[23] G. B. Huang, M. Ramesh, T. Berg, and E. Learned-Miller, "Labeled faces in the wild: A database for studying face recognition in unconstrained environments," Technical Report 07-49, University of Massachusetts, Amherst, Tech. Rep., 2007.
[24] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama *et al.*, "Speed/accuracy trade-offs for modern convolutional object detectors," *CVPR*, 2017.
[25] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size," *arXiv preprint arXiv:1602.07360*, 2016.
[26] M. Jaderberg, A. Vedaldi, and A. Zisserman, "Speeding up convolutional neural networks with low rank expansions," *BMVC*, 2014.
[27] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," *ACMMM*, 2014.
[28] S. Karayev, T. Baumgartner, M. Fritz, and T. Darrell, "Timely object recognition," in *NIPS*, 2012, pp. 890–898.
[29] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *ICLR*, 2015.
[30] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Tech. Rep., 2009.
[31] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *NIPS*, 2012, pp. 1097–1105.
[32] N. Kumar, A. Berg, P. N. Belhumeur, and S. Nayar, "Describable visual attributes for face verification and image search," *PAMI*, vol. 33, no. 10, pp. 1962–1977, 2011.
[33] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.
[34] Y. LeCun, J. S. Denker, and S. A. Solla, "Optimal brain damage," in *NIPS*, 1990, pp. 598–605.
[35] S. Leroux, S. Bohez, E. De Coninck, T. Verbelen, B. Vankeirsbilck, P. Simoens, and B. Dhoedt, "The cascading neural network: building the internet of smart things," *Knowledge and Information Systems*, pp. 1–24, 2017.
[36] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," *ICLR*, 2017.
[37] H. Li, Z. Lin, X. Shen, J. Brandt, and G. Hua, "A convolutional neural network cascade for face detection," in *CVPR*, 2015, pp. 5325–5334.
[38] Y. Li, H. Qi, J. Dai, X. Ji, and Y. Wei, "Fully convolutional instance-aware semantic segmentation," *CVPR*, 2017.

[39] J. Lin, Y. Rao, J. Lu, and J. Zhou, "Runtime neural pruning," in *NIPS*, 2017, pp. 2178–2188.

[40] M. L. Littman, "Reinforcement learning improves behaviour from evaluative feedback," *Nature*, vol. 521, no. 7553, pp. 445–451, 2015.

[41] L. Liu and J. Deng, "Dynamic deep neural networks: Optimizing accuracy-efficiency trade-offs by selective execution," *AAAI*, 2018.

[42] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, "Learning efficient convolutional networks through network slimming," in *ICCV*. IEEE, 2017, pp. 2755–2763.

[43] J.-H. Luo, J. Wu, and W. Lin, "Thinet: A filter level pruning method for deep neural network compression," *CVPR*, 2017.

[44] 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*, vol. 518, no. 7540, pp. 529–533, 2015.

[45] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient inference," *ICLR*, 2017.

[46] K. Murray and D. Chiang, "Auto-sizing neural networks: With applications to n-gram language models," *EMNLP*, 2015.

[47] A. Odena, D. Lawson, and C. Olah, "Changing model behavior at test-time using reinforcement learning," *ICLRW*, 2017.

[48] A. Paszke, S. Gross, S. Chintala, and G. Chanan, "Pytorch: Tensors and dynamic neural networks in python with strong gpu acceleration," 2017.

[49] M. L. Puterman, *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[50] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *ECCV*. Springer, 2016, pp. 525–542.

[51] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *NIPS*, 2015, pp. 91–99.

[52] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *IJCV*, vol. 115, no. 3, pp. 211–252, 2015.

[53] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. Le, G. Hinton, and J. Dean, "Outrageously large neural networks: The sparsely-gated mixture-of-experts layer," *ICLR*, 2017.

[54] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *ICLR*, 2015.

[55] M. F. Stollenga, J. Masci, F. Gomez, and J. Schmidhuber, "Deep networks with internal selective attention through feedback connections," in *NIPS*, 2014, pp. 3545–3553.

[56] N. Ström, "Phoneme probability estimation with dynamic sparsely connected artificial neural networks," *The Free Speech Journal*, vol. 5, pp. 1–41, 1997.

[57] C. Sun, M. Paluri, R. Collobert, R. Nevatia, and L. Bourdev, "Pronet: Learning to propose object-specific boxes for cascaded neural networks," in *CVPR*, 2016, pp. 3485–3493.

[58] Y. Sun, X. Wang, and X. Tang, "Deep convolutional network cascade for facial point detection," in *CVPR*, 2013, pp. 3476–3483.

[59] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *NIPS*, 2000, pp. 1057–1063.

[60] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning." in *AAAI*, 2017, pp. 4278–4284.

[61] ——, "Inception-v4, inception-resnet and the impact of residual connections on learning." in *AAAI*, vol. 4, 2017, p. 12.

[62] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *CVPR*, 2015, pp. 1–9.

[63] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *CVPR*, 2016, pp. 2818–2826.

[64] T. Tieleman and G. Hinton, "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude," *COURSERA: Neural networks for machine learning*, vol. 4, no. 2, 2012.

[65] M. Wang, B. Liu, and H. Foroosh, "Factorized convolutional neural networks," *ICCVW*, 2017.

[66] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[67] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," in *NIPS*, 2016, pp. 2074–2082.

[68] F. Zhang, J. Leitner, M. Milford, B. Upcroft, and P. Corke, "Towards vision-based deep reinforcement learning for robotic motion control," *ACRA*, 2015.

[69] X. Zhang, J. Zou, K. He, and J. Sun, "Accelerating very deep convolutional networks for classification and detection," *PAMI*, vol. 38, no. 10, pp. 1943–1955, 2016.

Yongming Rao is a fourth-year undergraduate student at Tsinghua Univeristy, Beijing, China, advised by Dr. Jiwen Lu at the Department of Automation of Tsinghua University. His current research interests include computer vision and deep learning, where he has published 4 top computer vision conference papers such as ICCV'2017, NIPS'2017 and CVPR'2018. He was a recipient of the Sensetime Undergraduate Scholarship in 2017.

Jiwen Lu received the B.Eng. degree in mechanical engineering and the M.Eng. degree in electrical engineering from the Xi'an University of Technology, Xi'an, China, and the Ph.D. degree in electrical engineering from the Nanyang Technological University, Singapore, in 2003, 2006, and 2012, respectively. He is currently an Associate Professor with the Department of Automation, Tsinghua University, Beijing, China. His current research interests include computer vision, pattern recognition, and machine learning. He serves as an Associate Editor of the *IEEE Trans. on Circuits and Systems for Video Technology*, the *IEEE Trans. on Biometrics, Behavior, and Identity Sciences*, *Pattern Recognition*, and *Journal of Visual Communication and Image Representation*. He was a recipient of the National 1000 Young Talents Program of China in 2015 and the National Science Fund for Excellent Young Scholars in 2018, respectively. He is a senior member of the IEEE.

Ji Lin is a fourth-year undergraduate student at Tsinghua Univeristy, Beijing, China, advised by Dr. Jiwen Lu at the Department of Automation of Tsinghua University. His current research interests include computer vision and deep learning, where he has published 3 top computer vision conference papers such as CVPR'2017, ICCV'2017 and NIPS'2017.

Jie Zhou received the BS and MS degrees both from the Department of Mathematics, Nankai University, Tianjin, China, in 1990 and 1992, respectively, and the PhD degree from the Institute of Pattern Recognition and Artificial Intelligence, Huazhong University of Science and Technology (HUST), Wuhan, China, in 1995. From then to 1997, he served as a postdoctoral fellow in the Department of Automation, Tsinghua University, Beijing, China. Since 2003, he has been a full professor in the Department of Automation, Tsinghua University. His research interests include computer vision, pattern recognition, and image processing. He is an associate editor for the *IEEE Trans. on Pattern Analysis and Machine Intelligence* and three other journals. He received the National Outstanding Youth Foundation of China Award. He is a senior member of the IEEE.