International Journal of Computer Vision (2021) 129:501–516
https://doi.org/10.1007/s11263-020-01379-y

Binarized Neural Architecture Search for Efficient Object Recognition

Hanlin Chen[1] · Li'an Zhuo[1] · Baochang Zhang[1,2] · Xiawu Zheng[3] · Jianzhuang Liu[4] · Rongrong Ji[3] · David Doermann[5] · Guodong Guo[6,7]

Received: 19 December 2019 / Accepted: 28 August 2020 / Published online: 1 October 2020

Abstract
Traditional neural architecture search (NAS) has a significant impact in computer vision by automatically designing network architectures for various tasks. In this paper, binarized neural architecture search (BNAS), with a search space of binarized convolutions, is introduced to produce extremely compressed models to reduce huge computational cost on embedded devices for edge computing. The BNAS calculation is more challenging than NAS due to the learning inefficiency caused by optimization requirements and the huge architecture space, and the performance loss when handling the wild data in various computing applications. To address these issues, we introduce operation space reduction and channel sampling into BNAS to significantly reduce the cost of searching. This is accomplished through a performance-based strategy that is robust to wild data, which is further used to abandon less potential operations. Furthermore, we introduce the upper confidence bound to solve 1-bit BNAS. Two optimization methods for binarized neural networks are used to validate the effectiveness of our BNAS. Extensive experiments demonstrate that the proposed BNAS achieves a comparable performance to NAS on both CIFAR and ImageNet databases. An accuracy of 96.53% vs. 97.22% is achieved on the CIFAR-10 dataset, but with a significantly compressed model, and a 40% faster search than the state-of-the-art PC-DARTS. On the wild face recognition task, our binarized models achieve a performance similar to their corresponding full-precision models.

Keywords Neural architecture search (NAS) · Binarized network · Object recognition · Edge computing

1 Introduction

Efficient computing has become one of the hottest topics both in academy and industry. It will be vital for the 5G networks by providing hardware-friendly and efficient solutions for practical and wild applications (Mao et al. 2017). Edge computing is about computing resources that are closer to the end user. This makes applications faster and users friendly (Chen and Ran 2019). It enables mobile or embedded devices to provide real-time intelligent analysis of big data, which can reduce the pressure on the cloud computing center and improve the availability (Han et al. 2019). However, edge

Communicated by Cha Zhang.

✉ Baochang Zhang
bczhang@buaa.edu.cn

Hanlin Chen
hlchen@buaa.edu.cn

Li'an Zhuo
lianzhuo@buaa.edu.cn

Xiawu Zheng
zhengxiawu@buaa.edu.cn

Jianzhuang Liu
jz.liu@siat.ac.cn

Rongrong Ji
rrji@buaa.edu.cn

David Doermann
doermann@buffalo.edu

Guodong Guo
guoguodong01@baidu.com

1 Beihang University, Beijing, China

2 Shenzhen Academy of Aerospace Technology, Shenzhen 100083, China

3 Xiamen University, Xiamen, Fujian, China

4 Shenzhen Institutes of Advanced Technology, Shenzhen, China

5 University at Buffalo, Buffalo, NY, USA

6 Institute of Deep Learning, Baidu Research, Beijing, China

7 National Engineering Laboratory for Deep Learning Technology and Application, Beijing, China

computing is still challenged by its limited computational ability, memory and storage and severe performance loss, making the models for edge computing inefficient for feature calculation and inference (Li et al. 2019).

One possible solution for efficient edge computing can be achieved based on compressed deep models, which mainly fall into three lines: network pruning, knowledge distillation and model quantization. Network pruning (Han et al. 2015) aims to remove network connections with less significance, and knowledge distillation (Hinton et al. 2015) introduces a teacher-student model, which uses the soft targets generated by the teacher model to guide the student model with much smaller model size, to achieve knowledge transfer. Differently, model quantization (Courbariaux et al. 2016) calculates neural networks with low-bit weights and activations to compress a model in a more efficient way, which is also orthogonal to the other two. The binarized model is widely considered as one of the most efficient ways to perform computing on embedded devices with an extremely less computational cost. Binarized filters have been used in traditional convolutional neural networks (CNNs) to compress deep models (Rastegari etal. 2016; Courbariaux et al. 2016, 2015; Juefei-Xu et al. 2017), showing up to 58-time speedup and 32-time memory saving. Juefei-Xu et al. (2017), the XNOR network is presented where both the weights and inputs attached to the convolution are approximated with binary values. This results in an efficient implementation of convolutional operations by reconstructing the unbinarized filters with a single scaling factor. Zhuang et al. (2018) introduces 2 $\sim$ 4-bit quantization based on a two-stage approach to quantize the weights and activations, which significantly improves the efficiency and performance of quantized models. Furthermore, WAGE (Wu et l. 2018) is proposed to discretize both the training and inference processes, and it quantizes not only weights and activations, but also gradients and errors. Gu et al. (2019), a projection convolutional neural network (PCNN) is proposed to realize binarized neural networks (BNNs) based on a simple back propagation algorithm. In our previous work (Zhao et al. 2019), we propose a novel approach, called Bayesian optimized 1-bit CNNs (denoted as BONNs), taking the advantage of Bayesian learning to significantly improve the performance of extreme 1-bit CNNs. There are also other practices in Tang et al. (2017), Alizadeh et al. (2018), and Ding et al. (2019) with improvements over previous works. Binarized models show the advantages on computational cost reduction and memory saving, but they unfortunately suffer from performance loss when handling wild data in practical applications. The main reasons are twofold. On the one hand, there is still a gap between low-bit weights/activations and full-precision weights/activations on feature representation, which should be investigated from new perspectives. On the other hand, traditional binarized networks are based on the neural architecture manually designed for full-precision networks, which means that binarized architecture design remains largely unexplored.

Traditional neural architecture search (NAS) has attracted great attention with a remarkable performance in various deep learning tasks. Impressive results have been shown for reinforcement learning (RL) based methods (Zoph et al. 2018; Zoph and Le 2016), for example, which train and evaluate more than 20,000 neural networks across 500 GPUs over 4 days. Recent methods like differentiable architecture search (DARTS) reduce the search time by formulating the task in a differentiable manner (Liu et al. 2019). DARTS relaxes the search space to be continuous, so that the architecture can be optimized with respect to its validation set performance by gradient descent, which provides a fast solution for effective network architecture search. To reduce the redundancy in the network space, partially-connected DARTS (PC-DARTS) was recently introduced to perform a more efficient search without compromising the performance of DARTS (Xu et al. 2019).

Although DARTS or its variants has a smaller model size than traditional light models, the searched network still suffers from an inefficient inference process due to the complicated architectures generated by multiple stacked full-precision convolution operations. Consequently, the searched network for embedded device is still computationally expensive and inefficient. At the same time, the existing gradient-based approaches select operations without a meaningful guidance. Not only is the search process inefficient, but also the selected operation might exhibit significant vulnerability to model attacks based on gradient information (Goodfellow et al. 2014; Madry et al. 2017), also for the wild data. Clearly, these problems require further exploration to overcome these challenges.

To address these above challenges, we transfer the NAS to a binarized neural architecture search (BNAS), by exploring the advantages of binarized neural networks (BNNs) on memory saving and computational cost reduction. In our BNAS framework as shown in Fig. 1, we use PC-DARTS as a warm-up step, which is followed by the performance-based method to improve the robustness of the resulting BNNs for the wild data. In addition, based on the observation that the early optimal operation is not necessarily the optimal one in the end, and the worst operation in the early stage usually has a worse performance at the end (Zheng et al. 2019). We exploit the advantages of both PC-DARTS and performance evaluation to prune the operation space. This means that the operations we finally reserve are certainly a near an optimal solution. On the other hand, with the operation pruning process, the search space becomes smaller and smaller, leading to an efficient search process. We show that the BNNs obtained by BNAS can outperform conventional BNN models by a large margin. It is a significant contribution in the

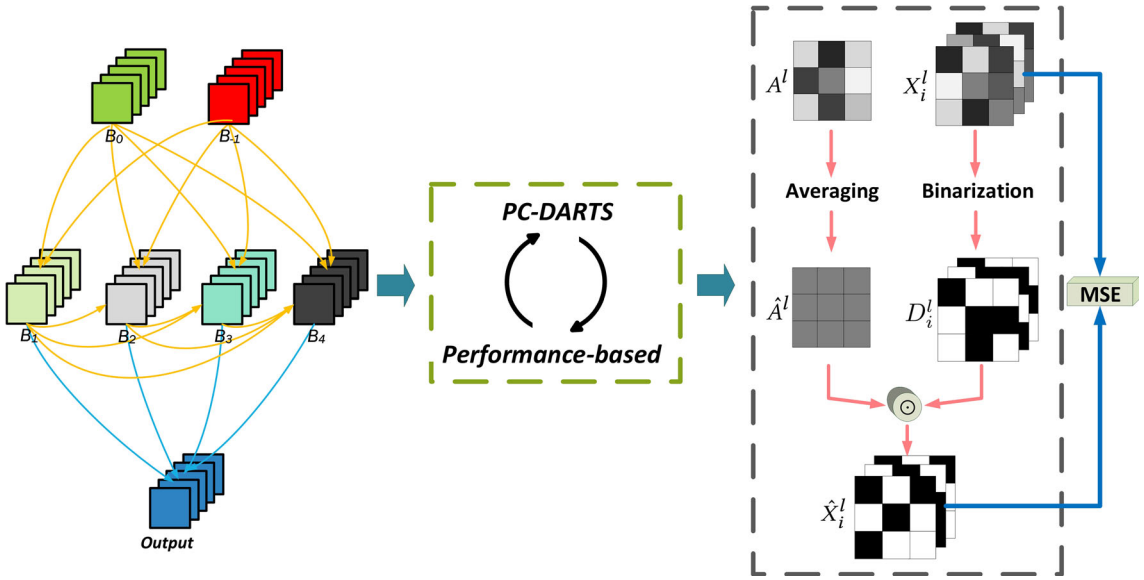

Fig. 1 The overall framework of the proposed binarized neural architecture search (BNAS). In BNAS, the search cell is a fully connected directed acyclic graph with four nodes, which is calculated based on PC-DARTS and a performance-based method. We also reformulate the optimization of binarization of CNNs in the same framework

field of BNNs, considering that the performance of conventional BNNs are not yet comparable with their corresponding full-precision models in terms of accuracy. To further validate the performance of our method, we also implement 1-bit BNAS in the same framework. Differently from BNNs (only kernels are binarized), 1-bit CNNs suffer from poor performance evaluation problem for binarized operations with binarzied activations in the beginning due to the insufficient training. We assume BNAS as a multi-armed bandit problem and introduce an exploration term based on the upper confidence bound (UCB) (Auer et al. 2002) to improve the search performance. The exploration term is used to handle the exploration-exploitation dilemma in the multi-armed bandit problem. We lead a new performance measure based on UCB by considering both the performance evaluation and number of trial for operation pruning in the same framework, which means that the operation is ultimately abandoned only when it is sufficiently evaluated.

The search process of our BNAS consists of two steps. One is the operation potential ordering based on partially-connected DARTS (PC-DARTS) (Xu et al. 2019) which also serves as a baseline for our BNAS. It is further improved with a second operation reduction step guided by a performance-based strategy. In the operation reduction step, we prune one operation at each iteration from one-half of the operations with less potential as calculated by PC-DARTS. As such, the optimization of the two steps becomes faster and faster because the search space is reduced due to the operation pruning. We can take advantage of the differential framework of DARTS where the search and performance evaluation are in the same setting. We also enrich the search strategy of DARTS. Not only is the gradient used to determine which operation is better, but the proposed performance evaluation is included for further reduction of the search space. The contributions of our paper include:

- BNAS is developed based on a new search algorithm which solves the BNNs and 1-bit CNNs optimization and architecture search in a unified framework. The 1-bit CNNs are obtained by incorporating the bandit strategy into BNAS, which can better evaluate the operation based on UCB.
- The search space is greatly reduced through a performance-based strategy used to abandon operations with less potential, which improves the search efficiency by 40%.
- Extensive experiments demonstrate that the proposed algorithm achieves much better performance than other light models on wild face recognition, CIFAR-10 and ImageNet.

This submission is an extension of our conference paper (Chen et al. 2020) by including: (1) extending our binarized models to 1-bit models, which are more challenging than BNNs; In addition, the 1-bit CNNs are achieved based on the bandit strategy, which can better evaluate the operation based on UCB; (2) adding more details about optimization of binarized models; (3) adding more experiments to sufficiently validate the performance of our methods, such as new experiments on wild face recognition, and results of 1-bit BNAS on all the datasets.

2 Related Work

In this section, we introduce the most related works on network quantization and NAS (DARTS). For the network quantization, both state-of-the-art BNNs and 1-bit CNNs are briefly introduced. We also described the PC-DARTS method, which are combined with binarized models, leading to a much better performance on object recognition tasks.

2.1 Neural Networks Quantization

To the best of our knowledge, Courbariaux et al. (2016) is the first attempt to binarize both the weights and activations of convolution layers in CNNs. It works well in maintaining the classification accuracy on small datasets like CIFAR-10 and CIFAR-100 (Krizhevsky et al. 2014), which is however less effective when being applied on large datasets like ImageNet (Rastegari etal. 2016; Deng et al. 2009). Instead of binarizing the kernel weights into ± 1, the work in Rastegari etal. (2016) adds a layer-wise scalar α_l to reconstruct the binarized kernels and proves that the mean absolute value (MAV) of each layer is the optimal value for α_l. Inspired by using a scalar to reconstruct binarized kernels, HQRQ (Li et al. 2017) adopts a high-order binarization scheme to achieve more accurate approximation while preserving the advantage of binary operation. In order to alleviate the degradation in prediction accuracy, ABC-Net (Lin et al. 2017) adopts multiple binary weights and activations to approximate full-precision weights. Leng et al. (2018) decoupled the continuous parameters from the discrete constraints of network using ADMM, which therefore achieves extremely low bit rates. Recently, Bi-real Net (Liu et al. 2018a) explores a new variant of residual structure to preserve the real activations before the sign function, with a tight approximation to the derivative of the non-differentiable sign function. McDonnell (2018) applied a warm-restart learning-rate schedule to quantize network weights into 1-bit, which achieves about $9 \sim 99\%$ of peak performance on CIFAR.

Quantizing kernel weights and activations to binary values is an extreme case of neural network quantization, which is prone to unacceptable accuracy degradation. Accordingly, sufficient attention has been paid to quantize DCNNs with more than 1 bit. Specifically, ternary weights are introduced to reduce the quantization error in TWN (Li and Liu 2016). DoReFa-Net (Zhou et al. 2016) exploits convolution kernels with low bit-width parameters and gradients to accelerate both the training and inference. TTQ (Zhu et al. 2017) uses two full-precision scaling coefficients to quantize the weights to ternary values. Zhuang et al. (2018) presented a $2 \sim 4$-bit quantization scheme using a two-stage approach to alternately quantize the weights and activations, which provides an optimal tradeoff among memory, efficiency and performance. Furthermore, WAGE (Wu et l. 2018) is proposed to discretize both the training and inference processes, where not only weights and activations but also gradients and errors are quantized. Other practices are shown in Tang et al. (2017), Alizadeh et al. (2018) and Ding et al. (2019) with improvements over previous works.

Despite the excellent efficiency, existing 1-bit CNNs suffer from its limited representation capability, leading to an inevitable performance loss on the object recognition tasks. Our previous works (Gu et al. 2019; Zhao et al. 2019) have significantly improved the performance of state-of-the-art 1-bit CNNs. However, the performance are still baffled by their manually designed architectures, and this paper exploits the BNAS method to further enhance the capability of BNNs, aiming to significantly reduce the gap to their full-precision counterparts.

2.2 Neural Architecture Search

Thanks to the rapid development of deep learning, significant gains in performance have been realized in a wide range of computer vision tasks, most of which are manually designed network architectures (Krizhevsky et al. 2012; Simonyan and Zisserman 2014; He et al. 2016; Huang et al. 2017). Recently, the new approach called neural architecture search (NAS) has been attracting increased attention. The goal is to find automatic ways of designing neural architectures to replace conventional hand-crafted ones. Existing NAS approaches need to explore a very large search space and can be roughly divided into three type of approaches: evolution-based, reinforcement-learning-based and one-shot-based.

In order to implement the architecture search within a short period of time, researchers try to reduce the cost of evaluating each searched candidate. Early efforts include sharing weights between searched and newly generated networks (Cai et al. 2018). Later, this method was generalized into a more elegant framework named one-shot architecture search (Brock et al. 2017; Cai et al. 2018; Liu et al. 2019; Pham et al. 2018; Xie et al. 2018; Zheng et al. 2019, ?). In these approaches, an over-parameterized network or super network covering all candidate operations is trained only once, and the final architecture is obtained by sampling from this super network. For example, Brock et al. (2017) trained the over-parameterized network using a HyperNet (Ha et al. 2016), and Pham et al. (2018) proposed to share parameters among child models to avoid retraining each candidate from scratch. DARTS (Liu et al. 2019) introduces a differentiable framework and thus combines the search and evaluation stages into one. Despite its simplicity, researchers have found some of its drawbacks and proposed a few improved approaches over DARTS (Xie et al. 2018; Chen et al. 2019). PDARTS (Chen et al. 2019) presents an efficient algorithm which allows the depth of searched architectures to grow gradually during the training procedure, with a significantly reduced search time.

ProxylessNAS (Cai et al. 2018) adopted the differentiable framework and proposed to search architectures on the target task instead of adopting the conventional proxy-based framework.

Unlike previous methods, the calculation of BNAS is more challenging due to the learning inefficiency and huge architecture search space, we implement BNAS based on combination of PC-DARTS and new performance measures. We prune one operation at each iteration from one-half of the operations with smaller weights calculated by PC-DARTS, and thus the search becomes faster and faster in the optimization. As such, BNAS shows stronger robustness to wild data than DARTS with gradient-based search strategy.

2.3 Bandit Problem

In probability theory, the multi-armed bandit problem is a problem in which a decision must be made among competing choices in a way that maximizes their expected gain. Each choice's properties are only partially known at any given time, and may become better understood as time passes or observed after the choices. The selection and following observations provide information useful in future choices. The aim is to minimize the distance from the optimal solution with the shortest time. A lot of breakthroughs have been made for the bandit problem for constructing the optimal selection policies with fastest rate of convergence (Lai and Robbins 1985).

Bandit optimization is commonly used to exemplify the exploration-exploit- ation trade-off dilemma to avoid an explosive traversal space and speed up optimal convergence. The upper confidence bound applied to trees (UCT) was propoesd as a bandit based Monte Carlo planning (Kocsis and Szepesvari 2006). It is also exploited to improve classical reinforcement learning methods such as Q-learning (Even-Dar et al. 2006) and state–action–reward–state–action (SARSA) (Tokic and Palm 2011). AlphaGo (Silver et al. 2017) modifies the original UCB multi-armed bandit policy by approximately predicting good arms at the start of a sequence of multi-armed bandit trials, which is called PUCB (predictor of upper confidence bounded) to balance the result of simulation and its uncertainty.

Objective functions for the multi-armed bandit problem tend to take one of two flavors: (1) best arm identification (or pure exploration) in which one is interested in identifying the arm with the highest average payoff, and (2) exploration-versus-exploitation in which one tries to maximize the cumulative payoff over time (Bubeck and Cesa-Bianchi 2012). Many optimization problems are studied in non-stochastic setting as the pull of each arm without the i.i.d. assumption (Neu 2015; Li et al. 2017; Jamieson and Talwalkar 2015). Relatedly, hyperband (Li et al. 2017) solves the pure-exploration bandit problem in the fixed budget setting without making parametric assumptions and achieves the state-of-the-art for the hyperparameter optimization. It extends the Successive Halving Algorithm (Jamieson and Talwalkar 2015) which evaluates and throws out the worst half until one remains. We share the similar idea of resources allocation with hyperband and formulate our BNAS as an exploration-versus-exploitation problem where the sampling and abandoning are based on UCB.

3 Binarized Neural Architecture Search

In this section, we first describe the search space in a general form, where the computation procedure for an architecture (or a cell in it) is represented as a directed acyclic graph. We then describe binarized optimaization for BNAS and review the baseline PC-DARTS (Xu et al. 2019), which is used as warm-up for our method. Then an operation sampling and a performance-based search strategy are proposed to effectively reduce the search space. Our BNAS framework is shown in Fig. 2 and additional details of it are described in the rest of this section. Finally, we reformulate the optimization of BNNs in a unified framework.

3.1 Search Space

Following Zoph and Le (2016), Zoph et al. (2018), Liu et al. (2019) and Real et al. (2019), we search for a computation cell as the building block of the final architecture. A network consists of a pre-defined number of cells (Zoph and Le 2016), which can be either normal cells or reduction cells. Each cell takes the outputs of the two previous cells as input. A cell is a fully-connected directed acyclic graph (DAG) of M nodes, *i.e.*, $\{B_1, B_2, \ldots, B_M\}$, as illustrated in Fig. 3a. Each node B_i takes its dependent nodes as input, and generates an output through a sum operation $B_j = \sum_{i<j} o^{(i,j)}(B_i)$. Here each node is a specific tensor (*e.g.,* a feature map in convolutional neural networks) and each directed edge (i, j) between B_i and B_j denotes an operation $o^{(i,j)}(\cdot)$, which is sampled from $\mathcal{O}^{(i,j)} = \{o_1^{(i,j)}, \ldots, o_K^{(i,j)}\}$. Note that the constraint $i < j$ ensures there are no cycles in a cell. Each cell takes the outputs of two dependent cells as input, and we define the two input nodes of a cell as B_{-1} and B_0 for simplicity. Following Liu et al. (2019), the set of the operations $\mathcal{O}$ consists of $K = 8$ operations. They include 3×3 max pooling, no connection (zero), 3×3 average pooling, skip connection (identity), 3×3 dilated convolution with rate 2, 5×5 dilated convolution with rate 2, 3×3 depth-wise separable convolution, and 5×5 depth-wise separable convolution, as illustrated in Fig. 3b. The search space of a cell is constructed by the operations of all the edges, denoted as $\{\mathcal{O}^{(i,j)}\}$.

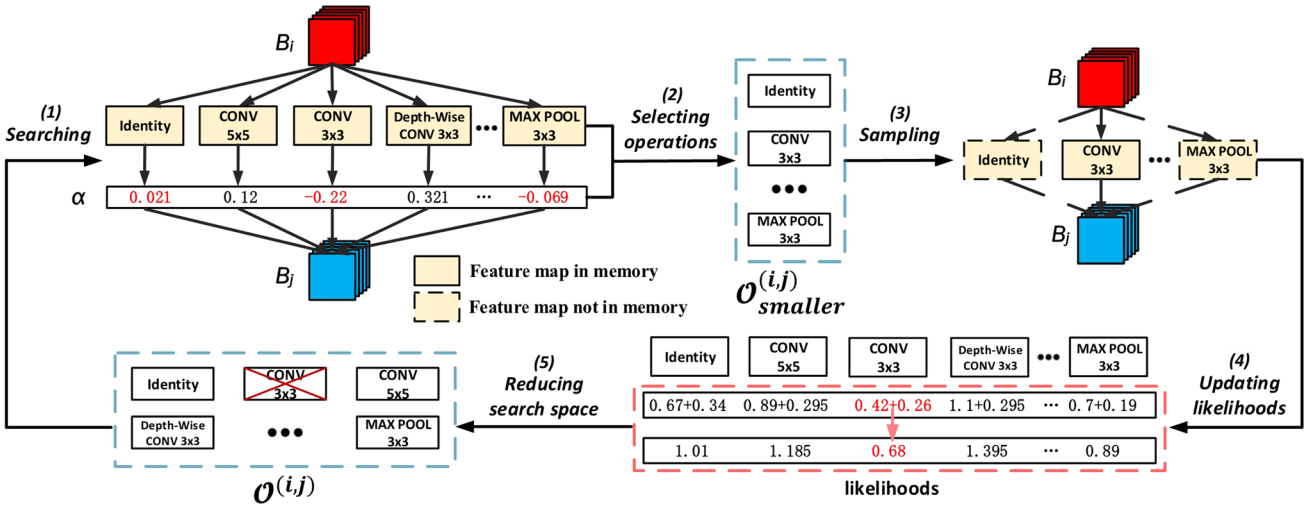

Fig. 2 The main steps of our BNAS: (1) search an architecture based on $\mathcal{O}^{(i,j)}$ using PC-DARTS. (2) Select half the operations with less potential from $\mathcal{O}^{(i,j)}$ for each edge, resulting in $\mathcal{O}^{(i,j)}_{smaller}$. (3) Select an architecture by sampling (without replacement) one operation from $\mathcal{O}^{(i,j)}_{smaller}$ for every edge, and then train the selected architecture. (4) Update the operation selection likelihood $s(o_k^{(i,j)})$ based on the accuracy obtained from the selected architecture on the validation data. (5) Abandon the operation with the minimal selection likelihood from the search space $\{\mathcal{O}^{(i,j)}\}$ for every edge

Unlike conventional convolutions, our BNAS is achieved by transforming all the convolutions in $\mathcal{O}$ to binarized convolutions. We denote the full-precision and binarized kernels as X and $\hat{X}$ respectively. A convolution operation in $\mathcal{O}$ is represented as $B_j = B_i \otimes \hat{X}$ as shown in Fig. 3b, where $\otimes$ denotes convolution. To build BNAS, one key step is how to binarize the kernels from X to $\hat{X}$, which can be implemented based on state-of-the-art BNNs, such as XNOR or PCNN. As we know, the optimization of BNNs is more challenging than that of conventional CNNs (Gu et al. 2019; Rastegari etal. 2016), which adds an additional burden to NAS. To solve it, we introduce channel sampling and operation space reduction into differentiable NAS to significantly reduce the cost of GPU hours, leading to an efficient BNAS.

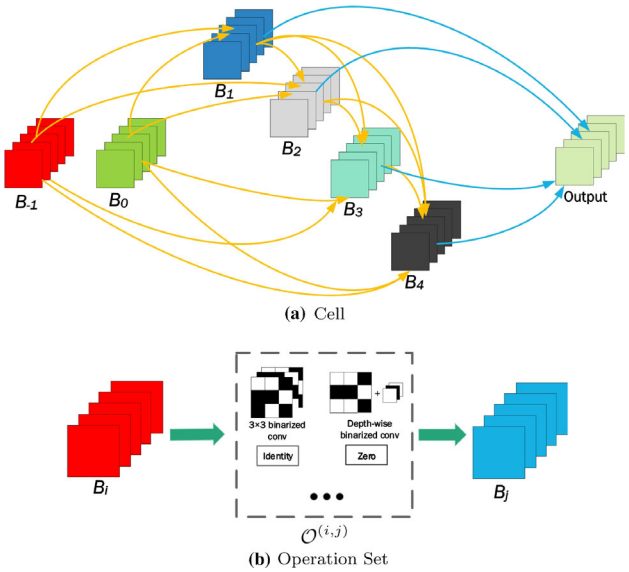

Fig. 3 (**a**) A cell contains 7 nodes, two input nodes B_{-1} and B_0, four intermediate nodes B_1, B_2, B_3, B_4 that apply sampled operations on the input nodes and upper nodes, and an output node that concatenates the outputs of the four intermediate nodes. (**b**) The set of operations $\mathcal{O}^{(i,j)}$ between B_i and B_j, including binarized convolutions

3.2 Binarized Optimization for BNAS

The inference process of a BNN model is based on the binarized kernels, which means that the kernels must be binarized in the forward step (corresponding to the inference) during training. Contrary to the forward process, during back propagation, the resulting kernels are not necessary to be binarized and can be full-precision.

In order to achieve binarized weights, we first divide each convolutional kernel into two parts (amplitude and direction), and formulate the current binarized methods in a unified framework. In addition to Table 1, we elaborate D, A and $\hat{A}$: D_i^l are the directions of the full-precision kernels X_i^l of the l^{th} convolutional layer, $l \in \{1, \ldots, N\}$; A^l shared by all D_i^l represents the amplitude of the l^{th} convolutional layer; $\hat{A}^l$ and A^l are of the same size and all the elements of $\hat{A}^l$ are equal to the average of the elements of A^l. In the forward pass, $\hat{A}^l$ is used instead of the full-precision A^l. In this case, $\hat{A}^l$ can be considered as a scalar. The full-precision A^l is only used for back propagation during training. Noted that our formulation can represent both XNOR based on scalar, and also simplified PCNN (Gu et al. 2019) whose scalar is learnable

Table 1 A brief description of the main notations used in Sect. 3.2

X: full-precision kernel	$\hat{X}$: binarized kernel	A: amplitude matrix
F: feature map	D: X's direction	$\hat{A}$: generated from A
i: kernel index	g: input feature map index	h: output feature map index
S: number of examples	l: layer index	M: number of facial landmarks

as a projection matrix. We represent $\hat{X}$ by the amplitude and direction as

$$\hat{X} = \hat{A} \odot D, \tag{1}$$

where $\odot$ denotes the element-wise multiplication between matrices. We then define an amplitude loss function to reconstruct the full-precision kernels as

$$L_{\hat{A}} = \frac{\theta}{2} \sum_{i,l} \|X_i^l - \hat{X}_i^l\|^2 = \frac{\theta}{2} \sum_{i,l} \|X_i^l - \hat{A}^l \odot D_i^l\|^2, \tag{2}$$

where $D_i^l = sign(X_i^l)$ represents the binarized kernel. X_i^l is the full-precision model which is updated during the back propagation process in PCNNs, while $\hat{A}^l$ is calculated based on a closed-form solution in XNOR. The element-wise multiplication combines the binarized kernels and the amplitude matrices to approximate the full-precision kernels. The final loss function is defined by considering

$$L_S = \frac{1}{2S} \sum_s \|\hat{Y}_s - Y_s\|_2^2, \tag{3}$$

where $\hat{Y}_s$ is the label of the s^{th} example; Y_s is the corresponding classification results. Finally, the overall loss function L is applied to supervise the training of BNAS in the back propagation as

$$L = L_S + L_{\hat{A}}. \tag{4}$$

The binarized optimization is used to optimize the neural architecture search, leading to our binarized neural architecture search (BNAS). To this end, we use partially-connected DARTS (PC-DARTS) to achieve operation potential ordering, which serves as a warm-up step for our BNAS. Denote by L_{train} and L_{val} the training loss and the validation loss, respectively. Both losses are determined by not only the architecture α but also the binarized weights $\hat{X}$ in the network. The goal for the warm-up step is to find $\hat{X}^*$ and α^* that minimize the validation loss $L_{val}(\hat{X}^*, \alpha^*)$, where the weights $\hat{X}^*$ associated with the architecture are obtained by minimizing the training loss $\hat{X}^* = \arg\min_{\hat{X}} L_{train}(\hat{X}, \alpha^*)$.

This implies a bilevel optimization problem with α as the upper-level variable and $\hat{X}$ as the lower-level variable:

$$\begin{aligned} &\arg\min_{\alpha} L_{val}(\hat{X}^*, \alpha) \\ &\quad s.t.\ \hat{X}^* = \arg\min_{\hat{X}} L_{train}(\hat{X}, \alpha). \end{aligned} \tag{5}$$

To better understand our method, we also review the core idea of PC-DARTS, which can take advantage of partial channel connections to improve memory efficiency. Taking the connection from B_i to B_j for example, this involves defining a channel sampling mask $S^{(i,j)}$, which assigns 1 to selected channels and 0 to masked ones. The selected channels are sent to a mixed computation of $|\mathcal{O}^{(i,j)}|$ operations, while the masked ones bypass these operations. They are directly copied to the output, which is formulated as

$$\begin{aligned} &f^{(i,j)}(B_i, S^{(i,j)}) \\ &= \sum_{o_k^{i,j} \in \mathcal{O}^{(i,j)}} \frac{exp\{\alpha_{o_k^{(i,j)}}\}}{\sum_{o_{k'}^{(i,j)} \in \mathcal{O}^{(i,j)}} exp\{\alpha_{o_{k'}^{(i,j)}}\}} \cdot o_k^{(i,j)}(S^{(i,j)} * B_i) \\ &\quad + (1 - S^{(i,j)}) * B_i, \end{aligned} \tag{6}$$

where $S^{(i,j)} * B_i$ and $(1 - S^{(i,j)}) * B_i$ denote the selected and masked channels, respectively, and $\alpha_{o_k^{(i,j)}}$ is the parameter of operation $o_k^{(i,j)}$ between B_i and B_j.

PC-DARTS sets the proportion of selected channels to $1/C$ by regarding C as a hyper-parameter. In this case, the computation cost can also be reduced by C times. However, the size of the whole search space is $2 \times K^{|\mathcal{E}_{\mathcal{M}}|}$, where $\mathcal{E}_{\mathcal{M}}$ is the set of possible edges with M intermediate nodes in the fully-connected DAG, and the "2" comes from the two types of cells. In our case with $M = 4$, together with the two input nodes, the total number of cell structures in the search space is $2 \times 8^{2+3+4+5} = 2 \times 8^{14}$. This is an extremely large space to search for a binarized neural architectures which need more time than a full-precision NAS. Therefore, efficient optimization strategies for BNAS are required.

3.3 Performance-Based Strategy for BNAS

Reinforcement learning is inefficient in the architecture search due to the delayed rewards in network training, i.e.,

the evaluation of a structure is usually done after the network training converges. On the other hand, we can perform the evaluation of a cell when training the network. Inspired by Ying et al. (2019), we use a performance-based strategy to boost the search efficiency by a large margin. Ying et al. (2019) did a series of experiments showing that in the early stage of training, the validation accuracy ranking of different network architectures is not a reliable indicator of the final architecture quality. However, we observe that the experiment results actually suggest a nice property that if an architecture performs badly in the beginning of training, there is little hope that it can be part of the final optimal model. As the training progresses, this observation shows less uncertainty. Based on this observation, we derive a simple yet effective operation abandoning process. During training, along with the increasing epochs, we progressively abandon the worst performing operation in each edge.

To this end, we reduce the search space $\{\mathcal{O}^{(i,j)}\}$ after the warm-up step achieved by PC-DARTS to increase search efficiency. According to $\{\alpha_{o_k^{(i,j)}}\}$, we can select half of the operations with less potential from $\mathcal{O}^{(i,j)}$ for each edge, resulting in $\mathcal{O}^{(i,j)}_{smaller}$. After that, we randomly sample one operation from the $K/2$ operations in $\mathcal{O}^{(i,j)}_{smaller}$ for every edge, then obtain the validation accuracy by training the sampled network for one epoch, and finally assign this accuracy to all the sampled operations. These three steps are performed $K/2$ times by sampling without replacement, leading to each operation having exactly one accuracy for every edge.

We repeat it T times. Thus each operation for every edge has T accuracies $\{y^{(i,j)}_{k,1}, y^{(i,j)}_{k,2}, \ldots, y^{(i,j)}_{k,T}\}$. Then we define the selection likelihood of the k^{th} operation in $\mathcal{O}^{(i,j)}_{smaller}$ for each edge as

$$s_{smaller}(o_k^{(i,j)}) = \frac{exp\{\bar{y}_k^{(i,j)}\}}{\sum_m exp\{\bar{y}_m^{(i,j)}\}}, \tag{7}$$

where $\bar{y}_k^{(i,j)} = \frac{1}{T}\sum_t y_{k,t}^{(i,j)}$. And the selection likelihoods of the other operations not in $\mathcal{O}^{(i,j)}_{smaller}$ are defined as

$$\begin{aligned} s_{larger}(o_k^{(i,j)}) &= \tfrac{1}{2}(\max_{o_k^{(i,j)}}\{s_{smaller}(o_k^{(i,j)})\} \\ &\quad + \tfrac{1}{\lceil K/2\rceil}\textstyle\sum_{o_k^{(i,j)}} s_{smaller}(o_k^{(i,j)})), \end{aligned} \tag{8}$$

where $\lceil K/2\rceil$ denotes the smallest integer $\geq K/2$. The reason to use it is because K can be an odd integer during iteration in the proposed Algorithm 1. Equation (8) is an estimation for the rest operations using a value balanced between the maximum and average of $s_{smaller}(o_k^{(i,j)})$. Then, $s(o_k^{(i,j)})$ is updated by

$$\begin{aligned} s(o_k^{(i,j)}) \leftarrow \frac{1}{2}s(o_k^{(i,j)}) + q_k^{(i,j)} s_{smaller}(o_k^{(i,j)}) \\ + (1 - q_k^{(i,j)}) s_{larger}(o_k^{(i,j)}), \end{aligned} \tag{9}$$

where $q_k^{(i,j)}$ is a mask, which is 1 for the operations in $\mathcal{O}^{(i,j)}_{smaller}$ and 0 for the others.

When searching for BNAS, we do not use PC-DARTS as warm-up for the consideration of efficiency because quantizing feature maps is slower. Hence, $\mathcal{O}^{(i,j)}_{smaller}$ is $\mathcal{O}^{(i,j)}$. Also, we introduce an exploration term into Eq. (9) based on bandit (Auer et al. 2002). In machine learning, the multi-armed bandit problem is a classic reinforcement learning problem that exemplifies the exploration-exploitation trade-off dilemma: shall we stick to an arm that gave high reward so far (exploitation) or rather probe other arms further (exploration)? The Upper Confidence Bound (UCB) is widely used for dealing with the exploration-exploitation dilemma in the multi-armed bandit problem. Then, with the above analysis, Eq. (9) becomes

$$s(o_k^{(i,j)}) \leftarrow s(o_k^{(i,j)}) + \delta * \sqrt{\frac{2\log N}{n_{k,t}^{(i,j)}}} \tag{10}$$

where N is the total number of samples, $n_{k,t}^{(i,j)}$ refers to the number of times the k^{th} operation of edge (i, j) has been selected, and t is the index of the epoch. The first item in Eq. (10) is the value term which favors the operations that look good historically and the second is the exploration term which allows operations to get an exploration bonus that grows with $\log N$. And in this work $\delta = 2$ is used to balance value term and exploration term. We also test other values, which achieve a littler worse results. In that, 1-bit convolutions which behave badly in sufficient trials are prone to be abandoned.

Finally, we abandon the operation with the minimal selection likelihood for each edge. Such that the search space size is significantly reduced from $2 \times |\mathcal{O}^{(i,j)}|^{14}$ to $2 \times (|\mathcal{O}^{(i,j)}| - 1)^{14}$. We have

$$\mathcal{O}^{(i,j)} \leftarrow \mathcal{O}^{(i,j)} - \{\arg\min_{o_k^{(i,j)}} s(o_k^{(i,j)})\}. \tag{11}$$

The optimal structure is obtained when there is only one operation left in each edge. Our performance-based search algorithm is presented in Algorithm 1. Note that in line 1, PC-DARTS is performed for L epochs as the warm-up to find an initial architecture, and line 14 is used to update the architecture parameters $\alpha_{o_k^{(i,j)}}$ for all the edges due to the reduction of the search space $\{\mathcal{O}^{(i,j)}\}$.

Algorithm 1: Performance-Based Search

Input: Training data, Validation data, Searching hyper-graph: $\mathcal{G}$, $K = 8$, $T = 3$, $V = 1$, $L = 5$, $s(o_k^{(i,j)}) = 0$ for all edges;
Output: Optimal structure α;

```
Search an architecture for L epochs based on O^(i,j) using PC-DARTS;
while (K > 1) do
    Select O^(i,j)_smaller consisting of ⌈K/2⌉ operations with smallest α_{o_k^(i,j)} from O^(i,j) for every edge;
    for t = 1, ..., T epoch do
        O'^(i,j)_smaller ← O^(i,j)_smaller;
        for e = 1, ..., ⌈K/2⌉ epoch do
            Select an architecture by sampling (without replacement) one operation from O'^(i,j)_smaller for every edge;
            Train the selected architecture and get the accuracy on the validation data;
            Assign this accuracy to all the sampled operations;
        end
    end
    Update s(o_k^(i,j)) using Eq. 9;
    if 1 bit then
        Update s(o_k^(i,j)) using Eq. 10;
    end
    Update the search space {O^(i,j)} using Eq. 11;
    Search the architecture for V epochs based on O^(i,j) using PC-DARTS;
    K = K − 1;
end
```

3.4 Gradient Update for BNAS

In BNAS, $\hat{X}^l$ in the l^{th} layer are used to calculated the output feature maps F^{l+1} as

$$F^{l+1} = ACconv(F^l, \hat{X}^l), \tag{12}$$

where $ACconv$ denotes the designed amplitude convolution operation in Eq. (13). In ACconv, the channels of the output feature maps are generated as follows

$$F_h^{l+1} = \sum_{i,g} F_g^l \otimes \hat{X}_i^l, \tag{13}$$

where $\otimes$ denotes the convolution operation; F_h^{l+1} is the h^{th} feature map in the $(l+1)^{th}$ convolutional layer; F_g^l denotes the g^{th} feature map in the l^{th} convolutional layer. Note that the kernels of BNAS are binarized, while for 1-bit BNAS, both the kernels and the activations are binarized. Similar to the previous work (Rastegari etal. 2016; Liu et al. 2018a; Gu et al. 2019), the 1-bit BNAS is obtained via binarizing the kernels and activations simultaneously. In addition, we replace ReLU with PReLU to reserve negative elements generated by 1-bit convolution.

In BNAS, what need to be learned and updated are the full-precision kernels X_i and amplitude matrices A. The kernels and the matrices are jointly learned. In each convolutional layer, BNAS update the full-precision kernels and then the amplitude matrices. In what follows, the layer index l is omitted for simplicity.

We denote δ_{X_i} as the gradient of the full-precision kernel X_i, and have

$$\delta_{X_i} = \frac{\partial L}{\partial X_i} = \frac{\partial L_S}{\partial X_i} + \frac{\partial L_{\hat{A}}}{\partial X_i}, \tag{14}$$

$$X_i \leftarrow X_i - \eta_1 \delta_{X_i}, \tag{15}$$

where η_1 is a learning rate. We then have

$$\begin{aligned} \frac{\partial L_S}{\partial X_i} &= \frac{\partial L_S}{\partial \hat{X}_i} \cdot \frac{\partial \hat{X}_i}{\partial X_i} \\ &= \frac{\partial L_S}{\partial \hat{X}_i} \cdot \hat{A} \cdot \mathbb{1}, \end{aligned} \tag{16}$$

$$\frac{\partial L_{\hat{A}}}{\partial X_i} = \theta \cdot (X_i - \hat{A} \odot D_i), \tag{17}$$

where X_i is the full-precision convolutional kernel corresponding to D_i, and $\mathbb{1}$ is the indicator function (Rastegari etal. 2016) widely used to estimate the gradient of non-differentiable function.

After updating X, we update the amplitude matrix A. Let δ_A be the gradient of A. According to Eq. 4, we have

$$\delta_A = \frac{\partial L}{\partial A} = \frac{\partial L_S}{\partial A} + \frac{\partial L_{\hat{A}}}{\partial A}, \tag{18}$$

$$A \leftarrow |A - \eta_2 \delta_A|, \tag{19}$$

where η_2 is another learning rate. Note that the amplitudes are always set to be non-negative. We then have

$$\frac{\partial L_S}{\partial A} = \sum_i \frac{\partial L_S}{\partial \hat{X}_i} \cdot \frac{\partial \hat{X}_i}{\partial \hat{A}} \cdot \frac{\partial \hat{A}}{\partial A} = \sum_i \frac{\partial L_S}{\partial \hat{X}_i} \cdot D_i, \tag{20}$$

$$\frac{\partial L_{\hat{A}}}{\partial A} = \frac{\partial L_{\hat{A}}}{\partial \hat{A}} \cdot \frac{\partial \hat{A}}{\partial A} = -\theta \cdot (X_i - \hat{A} \odot D_i) \cdot D_i, \tag{21}$$

where $\frac{\partial \hat{A}}{\partial A}$ is set to 1 for easy implementation of the algorithm. Note that $\hat{A}$ and A are respectively used in the forward pass and the back propagation in an asynchronous manner. The above derivations show that BNAS is learnable with the new BP algorithm.

4 Experiments

In this section, we compare our BNAS with state-of-the-art NAS methods, and also validate two BNAS models based

on XNOR (Rastegari etal. 2016) and PCNN (Gu et al. 2019). The 1-bit BNAS models are also included in our experiments to further validate our methods.

4.1 Experiment Protocol

4.1.1 Datasets

CIFAR-10 CIFAR-10 (Krizhevsky et al. 2014) is a natural image classification dataset, which is composed of a training set and a test set, with 50,000 and 10,000 32×32 color images, respectively. These images span 10 different classes, including airplanes, automobiles, birds, cats, deer, dogs, frogs, horses, ships and trucks.

ILSVRC12 ImageNet ILSVRC12 ImageNet object classification dataset (Russakovsky et al. 2015) is more diverse and challenging. It contains 1.2 million training images, and 50,000 validation images, across 1000 classes.

CASIA-WebFace CASIA-WebFace (Dong et al. 2014) is a face image dataset collected from over ten thousand different individuals, containing nearly half a million facial images. Note that compared to other private datasets used in DeepFace (Taigman et al. 2014) (4M), VGGFace (Parkhi et al. 2015) (2M) and FaceNet (Schroff et al. 2015) (200M), our training data contains just 490K images and is more challenging.

LFW The labeled faces in the wild (LFW) dataset (Huang et al. 2008) has 5749 celebrities and collected 13,323 photos of them from web. The photos are organized into 10 splits, each of which contain 6000 images. Celebrities in Frontal-Profile (CFP) (Sengupta et al. 2016) consists of 7000 images of 500 subjects. The dataset contains 5000 images in frontal view and 2000 images in extreme profile to evaluate the performance on coping with the pose variation. The data is divided into 10 splits, each containing an equal number of frontal-frontal and frontal-profile comparisons.

AgeDB AgeDB (Moschoglou et al. 2017) includes 16,488 images of various famous people. The images are categorized to 568 distinct subjects according to their identity, age and gender attributes.

4.1.2 Train and Search Details

In these experiments, we first search neural architectures on an over-parameterized network on CIFAR-10, and then evaluate the best architecture with a stacked deeper network on the same dataset. Then we further perform experiments to search architectures directly on ImageNet. We run the experiment multiple times and find that the resulting architectures only show slight variation in performance, which demonstrates the stability of the proposed method.

We use the same datasets and evaluation metrics as existing NAS works (Liu et al. 2019; Cai et al. 2018; Zoph et al. 2018; Liu et al. 2018b). First, most experiments are conducted on CIFAR-10 (Krizhevsky et al. 2009), and the color intensities of all images are normalized to $[-1, +1]$. During architecture search, the 50K training samples of CIFAR-10 is divided into two subsets of equal size, one for training the network weights and the other for searching the architecture hyper-parameters. When reducing the search space, we randomly select 5K images from the training set as a validation set (used in line 8 of Algorithm 1). Specially for 1-bit BNAS, we replace ReLU with PReLU to avoid the disappearance of negative numbers generated by 1-bit convolution, and the bandit strategy is introduced to solve the insufficient training problem caused by the binarization of both kernels and activations. To further show the efficiency of our method, we also search architecture on ImageNet directly.

In the search process, we consider a total of 6 cells in the network, where the reduction cell is inserted in the second and the fourth layers, and the others are normal cells. There are $M = 4$ intermediate nodes in each cell. Our experiments follow PC-DARTS. We set the hyper-parameter C in PC-DARTS to 2 for CIFAR-10 so only $1/2$ features are sampled for each edge. The batch size is set to 128 during the search of an architecture for $L = 5$ epochs based on $\mathcal{O}^{(i,j)}$ (line 1 in Algorithm 1). Note that for $5 \leq L \leq 10$, a larger L has little effect on the final performance, but costs more search time as shown in Table 3. We freeze the network hyper-parameters such as α, and only allow the network parameters such as filter weights to be tuned in the first 3 epochs. Then in the next 2 epochs, we train both the network hyper-parameters and the network parameters. This is to provide an initialization for the network parameters and thus alleviates the drawback of parameterized operations compared with free parameter operations. We also set $T = 3$ (line 4 in Algorithm 1) and $V = 1$ (line 14), so the network is trained less than 60 epochs, with a larger batch size of 400 (due to few operation samplings) during reducing the search space. The initial number of channels is 16. We use SGD with momentum to optimize the network weights, with an initial learning rate of 0.025 (annealed down to zero following a cosine schedule), a momentum of 0.9, and a weight decay of 5×10^{-4}. The learning rate for finding the hyper-parameters is set to 0.01. When we search architecture directly on ImageNet, we use the same parameters with searching on CIFAR-10 except that initial learning rate is set to 0.05

After search, in the architecture evaluation step, our experimental setting is similar to Liu et al. (2019), Zoph et al. (2018), and Pham et al. (2018). A larger network of 20 cells (18 normal cells and 2 reduction cells) is trained on CIFAR-10 for 600 epochs with a batch size of 96 and an additional

Table 2 Test error rates for human-designed full-precision networks, human-designed binarized networks, full-precision networks obtained by NAS, and networks obtained by our BNAS on CIFAR-10

Architecture	Test error (%)	# Params (M)	W	A	Search cost (GPU days)	Search method
ResNet-18 (He et al. 2016)	3.53	11.1	32	32	–	Manual
WRN-22 (Zagoruyko and Komodakis 2016)	4.25	4.33	32	32	–	Manual
DenseNet (Huang et al. 2017)	4.77	1.0	32	32	–	Manual
SENet (Hu et al. 2018)	4.05	11.2	32	32	–	Manual
NASNet-A (Zoph et al. 2018)	2.65	3.3	32	32	1800	RL
AmoebaNet-A (Real et al. 2019)	3.34	3.2	32	32	3150	Evolution
PNAS (Liu et al. 2018b)	3.41	3.2	32	32	225	SMBO
ENAS (Pham et al. 2018)	2.89	4.6	32	32	0.5	RL
Path-level NAS (Cai et al. 2018)	3.64	3.2	32	32	8.3	RL
DARTS(first order) (Liu et al. 2019)	2.94	3.1	32	32	1.5	Gradient-based
DARTS(second order) (Liu et al. 2019)	2.83	3.4	32	32	4	Gradient-based
PC-DARTS	2.78	3.5	32	32	0.15	Gradient-based
BNAS (full-precision)	2.84	3.3	32	32	0.08	Performance-based
Network in McDonnell (2018)	6.13	4.30	1	32	–	Manual
ResNet-18 (XNOR)	6.69	11.17	1	32	–	Manual
ResNet-18 (PCNN)	5.63	11.17	1	32	–	Manual
WRN22 (PCNN) (Gu et al. 2019)	5.69	4.29	1	32	–	Manual
PC-DARTS*	4.86	3.638	1	32	0.15	Gradient-based
PC-DARTS	4.88	3.1	1	32	0.18	Gradient-based
BNAS (XNOR)	5.71	2.3	1	32	0.104	Performance-based
BNAS (XNOR, larger)	4.88	3.5	1	32	0.104	Performance-based
BNAS	3.94	2.6	1	32	0.09375	Performance-based
BNAS†	4.01	2.7	1	32	0.094	Performance-based
BNAS (larger)	3.47	4.6	1	32	0.09375	Performance-based
ResNet-18 (PCNN) (Liu et al. 2019)	14.5	0.59	1	1	–	Manual
WRN22 (XNOR) (Zhao et al. 2019)	11.48	4.33	1	1	–	Manual
WRN22 (PCNN) (Liu et al. 2019)	8.38	2.4	1	1	-	Manual
PC-DARTS	8.94	4.2	1	1	0.21	Gradient-based
BNAS	8.3	4.6	1	1	0.112	Performance-based
BNAS†	6.72	4.7	1	1	0.113	Performance-based

'W' and 'A' refer to the weight and activation bitwidth respectively. For fair comparison, we select the architectures by NAS with similar parameters (< 5M). In addition, we also train an optimal architecture in a larger setting, *i.e.,* with more initial channels (44 in XNOR or 48 in PCNN). † Indicate that BNAS is performed based on Eq. (10), which is also the same case in the following experiments. $*$ Indicate that the result is tested by the quantized NAS architecture obtained by PC-DARTS

Table 3 With different L, the accuracy and search cost of BNAS based on PCNN on CIFAR10 dataset

Model	L				
	3	5	7	9	11
Accuracy (%)	95.8	96.06	95.94	96.01	96.03
Search cost	0.0664	0.09375	0.1109	0.1321	0.1687

regularization cutout (DeVries and Taylor 2017). The initial number of channels is 36. We use the SGD optimizer with an initial learning rate of 0.025 (annealed down to zero following a cosine schedule without restart), a momentum of 0.9, a weight decay of 3×10^{-4} and a gradient clipping at 5. When stacking the cells to evaluate on ImageNet, the evaluation stage follows that of DARTS, which starts with three convolution layers of stride 2 to reduce the input image resolution from 224×224 to 28×28. 14 cells (12 normal cells and 2 reduction cells) are stacked after these three layers, with the initial channel number being 64. The network is trained from scratch for 250 epochs using a batch size of 512. We use the SGD optimizer with a momentum of 0.9, an initial learning rate of 0.05 (decayed down to zero following a cosine schedule), and a weight decay of 3×10^{-5}. Additional enhancements are adopted including label smoothing and an

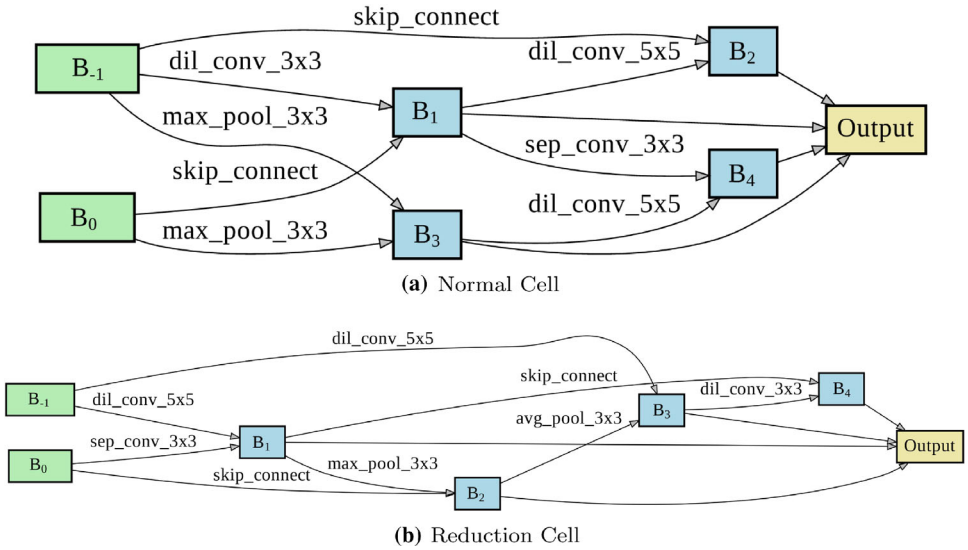

Fig. 4 Detailed structures of the best cells discovered on CIFAR-10 using BNAS based on XNOR. In the normal cell, the stride of the operations on 2 input nodes is 1, and in the reduction cell, the stride is 2

auxiliary loss tower during training. All the experiments and models are implemented in PyTorch (Paszke et al. 2017).

4.2 Results on CIFAR-10

We compare our method with both manually designed networks and networks searched by NAS. The manually designed networks include ResNet (He et al. 2016), Wide ResNet (WRN) (Zagoruyko and Komodakis 2016), DenseNet (Huang et al. 2017) and SENet (Hu et al. 2018). For the networks obtained by NAS, we classify them according to different search methods, such as RL (NASNet Zoph et al. 2018, ENAS Pham et al. 2018, and Path-level NAS Cai et al. 2018), evolutional algorithms (AmoebaNet Real et al. 2019), Sequential Model Based Optimization (SMBO) (PNAS Liu et al. 2018b), and gradient-based methods (DARTS Liu et al. 2019 and PC-DARTS Xu et al. 2019).

The results for different architectures on CIFAR-10 are summarized in Table 2. Using BNAS, we search for two binarized networks based on XNOR (Rastegari etal. 2016) and PCNN (Gu et al. 2019). In addition, we also train a larger XNOR variant with 44 initial channels and a larger PCNN variant with 48 initial channels. We can see that the test errors of the binarized networks obtained by our BNAS are comparable to or smaller than those of the full-precision human designed networks, and are significantly smaller than those of the other binarized networks.

Compared with the full-precision networks obtained by other NAS methods, the binarized networks by our BNAS have comparable test errors but with much more compressed models. Note that the numbers of parameters of all these searched networks are less than 5M, but the binarized networks only need 1 bit to save one parameter, while the full-precision networks need 32 bits. For 1-bit BNAS, as shown in Table 2, the UCB improves it by 1.58%, which

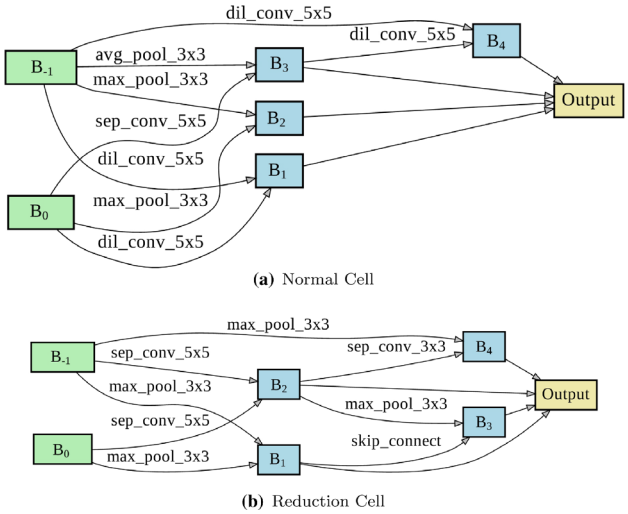

Fig. 5 Detailed structures of the best cells discovered on CIFAR-10 using BNAS based on PCNN. In the normal cell, the stride of the operations on 2 input nodes is 1, and in the reduction cell, the stride is 2

validates the effectiveness of our method. Also, we observe that up to 1.66% accuracy improvement is gained with 1-bit BNAS. In terms of search efficiency, compared with the previous fastest PC-DARTS, our BNAS is 40% faster (tested on our platform (NVIDIA GTX TITAN Xp). We attribute our superior results to the proposed way of solving the problem with the novel scheme of search space reduction. As illustrated in Figs. 4 and 5, compared with NAS, the architectures of BNAS prefer larger receptive fields. It also results in more pooling operations, most of which can increase the nonlinear representation ability of BNNs.

Our BNAS method can also be used to search full-precision networks. In Table 2, BNAS (full-precision) and PC-DARTS perform equally well, but BNAS is 47% faster.

Table 4 Comparison with the state-of-the-art image classification methods on ImageNet

Architecture	Accuracy (%)		Params (M)	W	A	Search cost (GPU days)	Search method
	Top1	Top5					
ResNet-18 (Gu et al. 2019)	69.3	89.2	11.17	32	32	–	Manual
MobileNetV1 (Howard et al. 2017)	70.6	89.5	4.2	32	32	-	Manual
NASNet-A (Zoph et al. 2018)	74.0	91.6	5.3	32	32	1800	RL
AmoebaNet-A (Real et al. 2019)	74.5	92.0	5.1	32	32	3150	Evolution
AmoebaNet-C (Real et al. 2019)	75.7	92.4	6.4	32	32	3150	Evolution
PNAS (Liu et al. 2018b)	74.2	91.9	5.1	32	32	225	SMBO
DARTS (Liu et al. 2019)	73.1	91.0	4.9	32	32	4	Gradient-based
PC-DARTS (Xu et al. 2019)	75.8	92.7	5.3	32	32	3.8	Gradient-based
ResNet-18 (PCNN) (Gu et al. 2019)	63.5	85.1	11.17	1	32	–	Manual
BNAS	71.3	90.3	6.2	1	32	2.6	Performance-based
ResNet-18 (Bi-Real) (Liu et al. 2018a)	56.4	79.5	11.17	1	1	–	Manual
ResNet-18 (BONN) (Zhao et al. 2019)	59.3	81.6	11.17	1	1	–	Manual
ResNet-18 (PCNN) (Gu et al. 2019)	57.3	80.0	11.17	1	1	–	Manual
BNAS	64.3	86.1	6.4	1	1	3.2	Performance-based

'W' and 'A' refer to the weight and activation bitwidth respectively. BNAS and PC-DARTS are obtained directly by NAS and BNAS on ImageNet, others are searched on CIFAR-10 and then directly transferred to ImageNet

Both the binarized methods XNOR and PCNN in our BNAS perform well, which shows the generalization of BNAS. Figures 4 and 5 show the best cells searched by BNAS based on XNOR and PCNN, respectively.

We also use PC-DARTS to perform a binarized architecture search based on PCNN on CIFAR10, resulting in a network denoted as PC-DARTS (PCNN). Compared with PC-DARTS (PCNN), BNAS achieves a better performance (95.12% vs. 96.06% in test accuracy) with less search time (0.18 vs. 0.09375 GPU days). We also compare our 1-bit BNAS with PC-DARTS, and find that our method is better than PC-DARTS (93.28% vs. 90.06%) on CIFAR-10 and about twice as fast as PC-DARTS (0.113 vs. 0.21 GPU days). The reason for this may be because the performance based strategy can help find better operations for recognition.

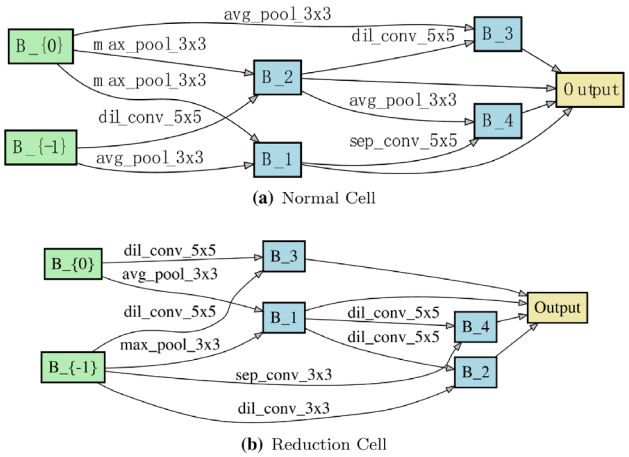

Fig. 6 Detailed structures of the best cells discovered on ImageNet using BNAS based on PCNN. In the normal cell, the stride of the operations on 2 input nodes is 1, and in the reduction cell, the stride is 2

4.3 Results on ImageNet

We further compare the state-of-the-art image classification methods on ImageNet. All the searched networks are obtained directly by NAS and BNAS on ImageNet by stacking the cells. Due to the large number of categories and data, ImageNet is more challenging than CIFAR-10 for binarized network. Different from the architecture settings for CIFAR-10, we do not binarize the first convolutional layer in depth-wise separable convolution and the preprocessing operations for 2 input nodes. Instead, we replace the concatenation with summation for the preprocessing operations and increase the number of channels for each cell. The benefits are more focusing on model compression with the state-of-the-art performance. From the results in Table 4, we have the following observations: (1) BNAS performs better than human-designed binarized networks (71.3% vs. 63.5%) and has far fewer parameters (6.1M vs. 11.17M). (2) BNAS has a performance similar to the human-designed full-precision light networks (71.3% vs. 70.6%), with a much more highly compressed model. (3) 1-bit BNAS achieves 5.0% accuracy improvement than the state-of-the-art human-designed 1-bit network, with fewer parameters. (4) Compared with the full-precision networks obtained by other NAS methods, BNAS has little performance drop, but is fastest in terms of search

Table 5 Test accuracies based on ResNet-18, ResNet-34, ResNet-50, ResNet-100 and BNAS on face recognition datasets

Architecture	Accuracy (%)			# Params (M)	W	A	Search cost (GPU days)	Search method
	LFW	CFP	AgeDB					
ResNet-18 (He et al. 2016)	98.68	92.33	90.23	24.02	32	32	–	Manual
ResNet-34 (He et al. 2016)	99.03	92.98	91.15	36.56	32	32	–	Manual
ResNet-50 (He et al. 2016)	99.07	93.73	91.58	45.46	32	32	–	Manual
ResNet-100 (He et al. 2016)	99.20	92.22	93.99	75.58	32	32	–	Manual
ResNet-18 (XNOR)	92.03	75.04	72.13	24.02	1	1	–	Manual
ResNet-18 (PCNN)	94.32	80.01	77.55	24.02	1	1	–	Manual
ResNet-34 (XNOR)	91.65	73.94	71.98	36.56	1	1	–	Manual
ResNet-34 (PCNN)	94.58	80.59	77.50	36.56	1	1	–	Manual
ResNet-50 (XNOR)	92.03	75.50	72.12	45.46	1	1	–	Manual
ResNet-100 (XNOR)	92.34	75.01	72.80	75.58	1	1	–	Manual
BNAS	98.57	92.46	89.03	10.224	1	32	0.717	Performance-based
BNAS	97.62	89.89	83.6	10.768	1	1	0.856	Performance-based

'W' and 'A' refer to the weight and activation bitwidth respectively. We train these models on the CASIA-WebFace dataset, but the test process are performed on the following datasets: LFW, CFP, AgeDB. On all the three test datasets, the results of BNAS consistently outperform the other methods

efficiency (0.09375 vs. 0.15 GPU days) and is a much more highly compressed model due to the binarization of the network. The above results show the excellent transferability of our BNAS method. Figure 6 shows the best cells searched by BNAS based on PCNN. They perform comparably to the full-precision networks obtained by NAS methods, but with highly compressed models.

4.4 Results on Face Recognition

In this section, we compare different kinds of ResNets with BNAS on face recognition task. Different kinds of ResNets are ResNet-18, ResNet34, ResNet-50 and ResNet-100 with kernel stage, 64-128-256-512 and each model has two FC layers. We directly search on CASIA-Webface for 17.2 h using one TITAN V GPU with 400 batch size, learning rate of 0.05. We use CASIA-Webface dataset for training and LFW, CFP, AgeDB datasets for testing. The setting of hyper-parameters is similar to the strategy of CIFAR experiments, despite the difference that the learning rate is 0.05 and the maximum epochs is set to 100. Note that the amount of parameters of ResNet is huge because we remove the pooling operation before FC layer following the face recognition code.[1] It makes the fully connected layer parameters large.

As demonstrated in Table 5, BNAS has a performance similar to the human-designed full-precision networks ResNet-18, with a much more highly compressed model. Also, 1-bit BNAS not only achieves the best test result among 1-bit CNNs but also has fewest parameters. On LFW, 1-bit BNAS has only 1.06% accuracy degradation compared to the results of the full-precision models ResNet-18, which verify the potential of 1-bit networks in practice.

[1] https://github.com/wujiyang/Face_Pytorch.

5 Conclusion

In this paper, we introduce BNAS (1-bit BNAS) for efficient object recognition, which is the first binarized neural architecture search algorithm. Our BNAS can effectively reduce the search time by pruning the search space in early training stages, which is faster than the previous most efficient search method PC-DARTS. We also introduce the bandit strategy into 1-bit BNAS, which can significantly improve the performance. The binarized networks searched by BNAS can achieve excellent accuracies on CIFAR-10, ImageNet, and wild face recognition. They perform comparably to the full-precision networks obtained by other NAS methods, but with much compressed models.

Acknowledgements The work was supported in part by National Natural Science Foundation of China under Grants 62076016 and 61672079. This work is supported by Shenzhen Science and Technology Program KQTD2016112515134654. Baochang Zhang is also with Shenzhen Academy of Aerospace Technology, Shenzhen 100083, China. Hanlin Chen and Li'an Zhuo have the same contributions to the paper.

References

Alizadeh, M., Fernández-Marqués, J., Lane, N. D., & Gal, Y. (2018). An empirical study of binary neural networks' optimisation. In *Proc. of ICLR*.

Auer, P., Cesa-Bianchi, N., & Fischer, P. (2002). Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, *47*(2–3), 235–256.

Brock, A., Lim, T., Ritchie, J. M., & Weston, N. (2017). Smash: One-shot model architecture search through hypernetworks. arXiv preprint arXiv:1708.05344.
Bubeck, S., & Cesa-Bianchi, N. (2012). Regret analysis of stochastic and nonstochastic multi-armed bandit problems. arXiv preprint arXiv:1204.5721.
Cai, H., Chen, T., Zhang, W., Yu, Y., & Wang, J. (2018). Efficient architecture search by network transformation. In *Proc. of AAAI*.
Cai, H., Yang, J., Zhang, W., Han, S., & Yu, Y. (2018). Path-level network transformation for efficient architecture search. arXiv preprint arXiv:1806.02639.
Cai, H., Zhu, L., & Han, S. (2018). ProxylessNAS: Direct neural architecture search on target task and hardware. In *Proc. of ICLR*.
Chen, J., & Ran, X. (2019). Deep learning with edge computing: A review. In *Proceedings of the IEEE*.
Chen, H., Zhuo, L., Zhang, B., Zheng, X., Liu, J., Doermann, D., & Ji, R. (2020). Binarized neural architecture search. In *Proc. of AAAI*.
Chen, X., Xie, L., Wu, J., & Tian, Q. (2019). Progressive differentiable architecture search: Bridging the depth gap between search and evaluation. In *Proc. of ICCV*.
Courbariaux, M., Bengio, Y., & David, J. P. (2015). BinaryConnect: Training deep neural networks with binary weights during propagations. In *Proc. of NIPS*.
Courbariaux, M., Hubara, I., Soudry, D., El-Yaniv, R., & Bengio, Y. (2016). Binarized neural networks: Training deep neural networks with weights and activations constrained to $+1$ or -1. arXiv preprint arXiv:1602.02830.
Deng, J., Dong, W., Socher, R., Li, L. J., Li, K., & Fei-Fei, L. (2009). ImageNet: A large-scale hierarchical image database. In *Proc. of CVPR*.
DeVries, T., & Taylor, G. W. (2017). Improved regularization of convolutional neural networks with cutout. arXiv preprint arXiv:1708.04552.
Ding, R., Chin, T. W., Liu, Z., & Marculescu, D. (2019). Regularizing activation distribution for training binarized deep networks. In *Proc. of CVPR*.
Dong, Y., Zhen, L., Liao, S., & Li, S. Z. (2014). Learning face representation from scratch. In *Computer science*.
Even-Dar, E., Mannor, S., & Mansour, Y. (2006). Action elimination and stopping conditions for the multi-armed bandit and reinforcement learning problems. *Machine Learning Research*, *7*, 1079–1105.
Goodfellow, I. J., Shlens, J., & Szegedy, C. (2014). Explaining and harnessing adversarial examples. arXiv preprint arXiv:1412.6572.
Gu, J., Li, C., Zhang, B., Han, J., Cao, X., Liu, J., & Doermann, D. (2019). Projection convolutional neural networks for 1-bit CNNs via discrete back propagation. In *Proc. of AAAI*.
Ha, D., Dai, A., & Le, Q. V. (2016). Hypernetworks. arXiv preprint arXiv:1609.09106.
Han, S., Pool, J., Tran, J., & Dally, W. (2015). Learning both weights and connections for efficient neural network. In *Proc. of NIPS*.
Han, Y., Wang, X., Leung, V., Niyato, D., Yan, X., & Chen, X. (2019). Convergence of edge computing and deep learning: A comprehensive survey. In *arXiv*.
He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proc. of CVPR*.
Hinton, G., Vinyals, O., & Dean, J. (2015). Distilling the knowledge in a neural network. In *Computer Science*.
Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., & Adam, H. (2017). MobileNets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861.
Hu, J., Shen, L., & Sun, G. (2018). Squeeze-and-excitation networks. In *Proc. of CVPR*.
Huang, G., Liu, Z., Van Der Maaten, L., & Weinberger, K. Q. (2017). Densely connected convolutional networks. In *Proc. of CVPR*.
Huang, G. B., Mattar, M., Berg, T., & Learned-Miller, E. (2008). Labeled faces in the wild: A database for studying face recognition in unconstrained environments.
Jamieson, K., & Talwalkar, A. (2015). Non-stochastic best arm identification and hyperparameter optimization. In *International conference on artificial intelligence and statistics*
Juefei-Xu, F., Naresh Boddeti, V., & Savvides, M. (2017). Local binary convolutional neural networks. In *Proc. of CVPR*.
Kocsis, L., & Szepesvari, C. (2006). Bandit based Monte-Carlo planning. In *Proc. of ECML*.
Krizhevsky, A., Hinton, G., et al. (2009). Learning multiple layers of features from tiny images. *Technical Report*.
Krizhevsky, A., Nair, V., & Hinton, G. (2014). The CIFAR-10 dataset. http://www.cs.toronto.edu/kriz/cifar.html
Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. In *Proc. of NIPS*.
Lai, T., & Robbins, H. (1985). Asymptotically efficient adaptive allocation rules. *Advances in Applied Mathematics*, *6*(1), 4–22.
Leng, C., Dou, Z., Li, H., Zhu, S., & Jin, R. (2018). Extremely low bit neural network: Squeeze the last bit out with ADMM. In *Proc. of AAAI*.
Li, E., Zeng, L., Zhou, Z., & Chen, X. (2019). Edge AI: On-demand accelerating deep neural network inference via edge computing. *IEEE Transactions on Wireless Communications*, *19*(1), 447–457.
Li, F., & Liu, B. (2016). Ternary weight networks. In *The 1st international workshop on efficient methods for deep neural networks*.
Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., & Talwalkar, A. (2017). Hyperband: A novel bandit-based approach to hyperparameter optimization. *The Journal of Machine Learning Research*, *18*(1), 6765–6816.
Li, Z., Ni, B., Zhang, W., Yang, X., & Gao, W. (2017). Performance guaranteed network acceleration via high-order residual quantization. In *Proc. of ICCV*.
Lin, X., Zhao, C., & Pan, W. (2017). Towards accurate binary convolutional neural network. In *Proc. of NIPS*.
Liu, C., Ding, W., Xia, X., Hu, Y., Zhang, B., Liu, J., Zhuang, B., & Guo, G. (2019). RBCN: Rectified binary convolutional networks for enhancing the performance of 1-bit DCNNs. In *Proc. of AAAI*.
Liu, C., Zoph, B., Neumann, M., Shlens, J., Hua, W., Li, L. J., Fei-Fei, L., Yuille, A., Huang, J., & Murphy, K. (2018). Progressive neural architecture search. In *Proc. of ECCV*.
Liu, H., Simonyan, K., & Yang, Y. (2019). DARTS: Differentiable architecture search. In *Proc. of ICLR*.
Liu, Z., Wu, B., Luo, W., Yang, X., Liu, W., & Cheng, K. T. (2018). Bi-real net: Enhancing the performance of 1-bit CNNs with improved representational capability and advanced training algorithm. In *Proc. of ECCV*.
Madry, A., Makelov, A., Schmidt, L., Tsipras, D., & Vladu, A. (2017). Towards deep learning models resistant to adversarial attacks. arXiv preprint arXiv:1706.06083.
Mao, Y., You, C., Zhang, J., Huang, K., & Letaief, K. B. (2017). Mobile edge computing: Survey and research outlook. arXiv preprint arXiv:1701.01090.
McDonnell, M. D. (2018). Training wide residual networks for deployment using a single bit for each weight. arXiv preprint arXiv:1802.08530.
Moschoglou, S., Papaioannou, A., Sagonas, C., Deng, J., Kotsia, I., & Zafeiriou, S. (2017). AgeDB: the first manually collected, in-the-wild age database. In *Proc. of CVPR workshops*.
Neu, G. (2015). Explore no more: Improved high-probability regret bounds for non-stochastic bandits. In *Proc. of NIPS*.
Parkhi, O. M., Vedaldi, A., Zisserman, A., et al. (2015). Deep face recognition. In *Proc. of BMVC*.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., & Lerer, A. (2017). Automatic differentiation in pyTorch. In *Proc. of NIPS*.

Pham, H., Guan, M. Y., Zoph, B., Le, Q. V., & Dean, J. (2018). Efficient neural architecture search via parameter sharing. arXiv preprint arXiv:1802.03268.

Rastegari, M., Ordonez, V., Redmon, J., & Farhadi, A. (2016). XNOR-Net: ImageNet classification using binary convolutional neural networks. In *Proc. of ECCV*.

Real, E., Aggarwal, A., Huang, Y., & Le, Q. V. (2019). Regularized evolution for image classifier architecture search. In *Proc. of AAAI*.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., et al. (2015). ImageNet large scale visual recognition challenge. *International Journal of Computer Vision*, *115*(3), 211–252.

Schroff, F., Kalenichenko, D., & Philbin, J. (2015). FaceNet: A unified embedding for face recognition and clustering. In *Proc. of CVPR*.

Sengupta, S., Chen, J. C., Castillo, C., Patel, V. M., Chellappa, R., & Jacobs, D. W. (2016). Frontal to profile face verification in the wild. In *Proc. of WACV*.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., et al. (2017). Mastering the game of go without human knowledge. *Nature*, *550*(7676), 354–359.

Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556.

Taigman, Y., Ming, Y., Ranzato, M., & Wolf, L. (2014). DeepFace: Closing the gap to human-level performance in face verification. In *Proc. of CVPR*.

Tang, W., Hua, G., & Wang, L. (2017). How to train a compact binary neural network with high accuracy? In *Proc. of AAAI*.

Tokic, M., & Palm, G. (2011). Value-difference based exploration: Adaptive control between epsilon-greedy and softmax. In *Annual conference on artificial intelligence*.

Wu, S., Li, G., Chen, F., & Shi, L. (2018). Training and inference with integers in deep neural networks. In *Proc. of ICLR*.

Xie, S., Zheng, H., Liu, C., & Lin, L. (2018). SNAS: Stochastic neural architecture search. arXiv preprint arXiv:1812.09926.

Xu, Y., Xie, L., Zhang, X., Chen, X., Qi, G.J., Tian, Q., & Xiong, H. (2019). Partial channel connections for memory-efficient differentiable architecture search. arXiv preprint arXiv:1907.05737.

Ying, C., Klein, A., Real, E., Christiansen, E., Murphy, K., & Hutter, F. (2019). NAS-bench-101: Towards reproducible neural architecture search. In *Proc. of ICML*.

Zagoruyko, S., & Komodakis, N. (2016). Wide residual networks. In *Proc. of BMVC*

Zhao, J., Gu, J., Jiang, X., Zhang, B., Jianzhuang, L., Guo, G., & Ji, R. (2019). Bayesian optimized 1-bit CNNs. In *Proc. of ICCV*.

Zheng, X., Ji, R., Tang, L., Wan, Y., Zhang, B., Wu, Y., Wu, Y., & Shao, L. (2019). Dynamic distribution pruning for efficient network architecture search. arXiv preprint arXiv:1905.13543.

Zheng, X., Ji, R., Tang, L., Zhang, B., Liu, J., & Tian, Q. (2019). Multinomial distribution learning for effective neural architecture search. In *Proc. of ICCV*.

Zhou, S., Wu, Y., Ni, Z., Zhou, X., Wen, H., & Zou, Y. (2016). DoReFa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. arXiv preprint arXiv:1606.06160.

Zhu, C., Han, S., Mao, H., & Dally, W. J. (2017). Trained ternary quantization. In *Proc. of ICLR*.

Zhuang, B., Shen, C., Tan, M., Liu, L., & Reid, I. (2018). Towards effective low-bitwidth convolutional neural networks. In *Proc. of CVPR*.

Zoph, B., & Le, Q. V. (2016). Neural architecture search with reinforcement learning. arXiv preprint arXiv:1611.01578.

Zoph, B., Vasudevan, V., Shlens, J., & Le, Q. V. (2018). Learning transferable architectures for scalable image recognition. In *Proc. of CVPR*.