Joint Neural Architecture Search and Quantization

Yukang Chen[1,2], Gaofeng Meng[1,2], Qian Zhang[3]
Xinbang Zhang[1,2], Liangchen Song[3], Shiming Xiang[1,2], Chunhong Pan[1,2]
[1]*National Laboratory of Pattern Recognition, Institute of Automation, Chinese Academy of Sciences*
[2]*University of Chinese Academy of Sciences* [3]*Horizon Robotics*
{*yukang.chen, gfmeng, xinbang.zhang, smxiang, chpan*}@*nlpr.ia.ac.cn*
{*qian01.zhang, liangchen.song*}@*horizon.ai*

Abstract

Designing neural architectures is a fundamental step in deep learning applications. As a partner technique, model compression on neural networks has been widely investigated to gear the needs that the deep learning algorithms could be run with the limited computation resources on mobile devices. Currently, both the tasks of architecture design and model compression require expertise tricks and tedious trials. In this paper, we integrate these two tasks into one unified framework, which enables the joint architecture search with quantization (compression) policies for neural networks. This method is named JASQ. Here our goal is to automatically find a compact neural network model with high performance that is suitable for mobile devices. Technically, a multi-objective evolutionary search algorithm is introduced to search the models under the balance between model size and performance accuracy.

In experiments, we find that our approach outperforms the methods that search only for architectures or only for quantization policies. 1) Specifically, given existing networks, our approach can provide them with learning-based quantization policies, and outperforms their 2 bits, 4 bits, 8 bits, and 16 bits counterparts. It can yield higher accuracies than the float models, for example, over 1.02% higher accuracy on MobileNet-v1. 2) What is more, under the balance between model size and performance accuracy, two models are obtained with joint search of architectures and quantization policies: a high-accuracy model and a small model, JASQNet and JASQNet-Small that achieves 2.97% error rate with 0.9 MB on CIFAR-10.

1. Introduction

Deep convolutional neural networks have successfully revolutionized various challenging tasks, e.g., image classification [12, 16, 31], object detection [28] and semantic segmentation [3]. Benefited from its great representation power, CNNs have released human experts from laborious feature engineering with end-to-end learning paradigms. However, another exhausting task appears, i.e., neural architecture design that also requires endless trails and errors. For further liberation of human labours, many neural architecture search (NAS) methods [35, 27] have been proposed and proven to be capable of yielding high-performance models. But the technique of NAS alone is far from real-world AI applications.

As networks usually need to be deployed on devices with limited resources, model compression techniques are also indispensable. In contrast to NAS that is considered at the topological level, model compression aims to refine the neural nodes of a given network with sparse connections or weighting-parameter quantization. However, computation strategies also need elaborate design. Taking quantization for example, conventional quantization policies often compress all layers to the same level. Actually each layer has different redundancy, it is wise to determine a suitable quantization bit for each layer. However, quantization choices also involve a large search space and designing mutual heuristics would make human burden heavier.

In this paper, we make a further step for the liberation of human labours and propose to integrate architecture search and quantization policy into a unified framework for neural networks (JASQ). A Pareto optimal model [5] is constructed in the evolutionary algorithm to achieve good trade-offs between accuracy and model size. By adjusting the multi-objective function, our search strategy can output suitable models for different accuracy or model size demands. During search, a population of models are first initialized and then evolved in iterations according to their fitness. Fig. 1 shows the evolutionary framework of our method. Our method brings the following advantages:

- *Effectiveness* Our method can jointly search for neural architectures and quantization policies. The re-
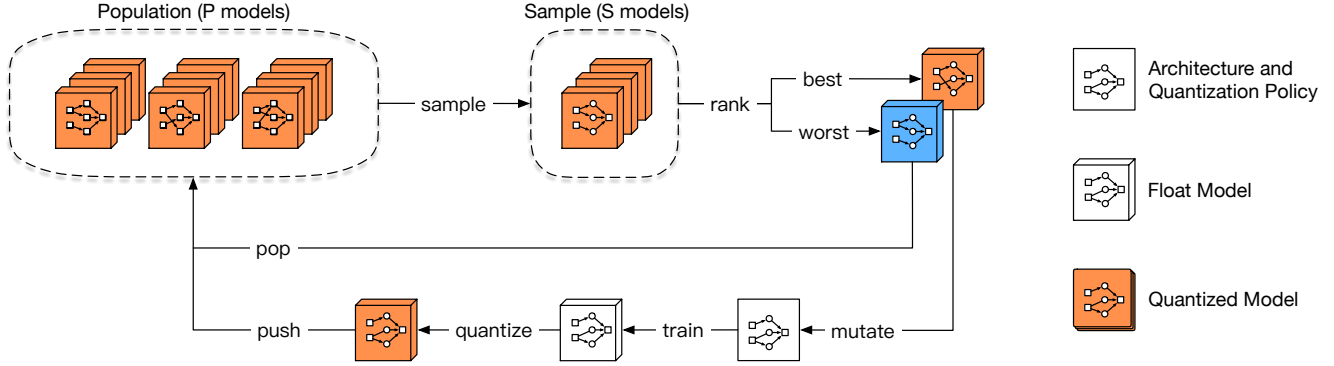
Figure 1. The evolutionary algorithm framework for our joint search method. Each individual in the population is evaluated with the accuracy and model size of the quantized model. When architectures are fixed during search, the method could provide existing networks with quantization policies.

sulting models, i.e., JASQNet and JASQNet-Small, achieve competitive accuracy to state-of-the-art methods [12, 16, 35] and have relatively small model size. For existing architectures, e.g., ResNet [12], DenseNet [16] and MobileNets [15, 29], our quantized models can outperform their 2/4/8/16 bits counterparts and even achieve higher accuracies than float models on ImageNet.

- *Flexibility*   In our evolutionary search method, a multi-objective function is adopted as illustrated in Fig. 3 and Eq. (1). By adjusting $\mathcal{T}_\mathcal{S}$ in the objective function, we obtain models with different accuracy and size balances. JASQNet has a comparable accuracy to ResNet34 [12] but much less model size. JASQNet-Small has a similar model size to SqueezeNet [17] but much better accuracy (65.90% vs 58.09%).

- *Efficiency*   We need only 1 GPU across 3 days to accomplish the joint search of architectures and quantization policies. Given hand-craft networks, their quantization policies can be automatically found in a few hours on ImageNet.

2. Related Work

2.1. Neural Architecture Search

Techniques in automatically designing network [35, 24, 27] have attracted increasing research interests. Current works usually fall into one of two categories: reinforcement learning (RL) and evolutionary algorithm (EA). In terms of RL-based methods, NAS [34] abstracts networks into variable-length strings and uses a reinforcement controller to determine models sequentially. NASNet [35] follows this search algorithm, but adopts cell-wise search space to save computational resources. In terms of EA-based methods, AmoebaNet [27] shows that a common evolutionary algo-

rithm without any controller can also achieve comparable results and even surpass RL-based methods.

In addition to RL and EA, some other methods have also been applied. DARTS [20] introduces a gradient-based method where they formulate the originally discrete search space into continuous parameters. PNAS [19] uses a sequential model-based optimization (SMBO) strategy to search architectures in order of increasing complexity. Other methods including MCTS [23], boosting [4] and hill-climbing [9] have also shown their potentials. Most methods mentioned above have produced networks that outperforms classical hand-crafted models. However, only neural architectures can not satisfy the demands of real-world applications. Thus, we propose a more convenient approach to provide complete schemes for deep learning practitioners.

2.2. Model Compression

Model compression has received increasing attention. This technique can effectively execute deep models in resource-constrained environments, such as mobile or embedded devices. A few practical methods are proposed and put into effect. Network pruning conducts channel-level compressions for CNN models [21, 11]. Distillation has been introduced recently [14, 2] that transfers the behaviour of a given model to the smaller student structure. In addition, some special convolution structures are also applied in mobile size devices, such as separable depthwise convolution [15] and 1 x 3 then 3 x 1 factorized convolution [31]. To reduce the redundancy of the fully connected layer, some methods propose to factorize its weights into truncated pieces [7, 32].

Quantization is also a significant branch of model compression and widely used in real applications [25, 33, 26]. Quantization can effectively reduce model size and thus save storage space and communication cost. Previous works tend to use a uniform precision for the whole network regardless of the different redundancy for each layer. De-
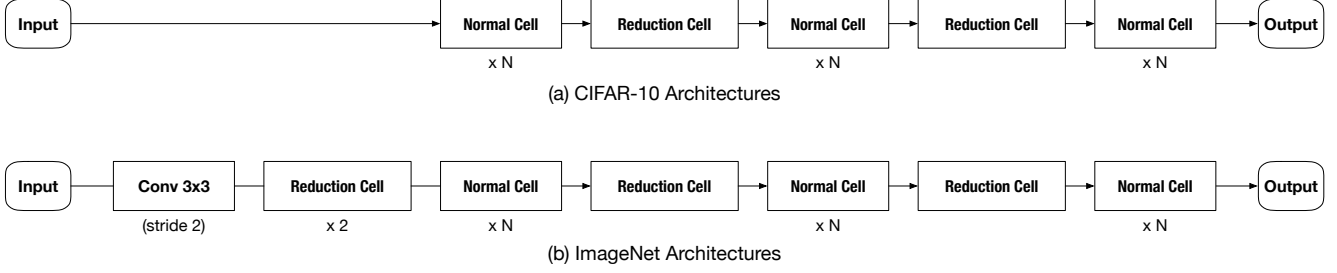
Figure 2. Architectures for CIFAR-10 and ImageNet. The image size in ImageNet (224x224) is much larger than that in CIFAR-10 (32x32). So there are additional reduction cells and convolution 3x3 with stride 2 in ImageNet architectures to downsample feature maps.

termining mixed precisions for different layers seems more promising. Actually mixed precision storage and computation have been widely supported by most hardware platforms, e.g., CPUs and FPGAs. However, because each model has tens or hundreds of layers, it is tedious to conduct this job by human experts. In this work, we combine the search of quantization policies with neural architecture search. Determining a quantization bit for a convolution layer is similar to choosing its kernel size. It is easy to implement this method based on previous NAS works.

3. Methods

Neural architecture design and model compression are both essential steps in deep learning applications, especially when we face mobile devices that have limited computation resources. However, both of them are time-consuming if conducted by human experts. In this work, we joint search of neural architectures and quantization policies in a unified framework. Compared with only searching for architectures, we evolve both architectures and quantization policies and use the validation accuracies of quantized models as fitnesses. Fig. 1 illustrates our framework.

3.1. Problem Definition

A quantized model Θ can be constructed by its neural network architecture $\mathscr{A}$ and its quantization policy $\mathscr{P}$. After the model is quantized, we can obtain its validation accuracy $\alpha(\Theta)$ and its model size $\mathcal{S}(\Theta)$. In this paper, we define the search problem as a multi-objective problem. The Pareto optimal model [5] is famous for solving multi-objective problems and we define our search problem into maximizing the objective function $\mathcal{F}(\Theta)$ as follow:

$$\max_{\Theta} \mathcal{F}(\Theta) = \max_{\Theta} \alpha(\Theta) \cdot \left[\frac{\mathcal{S}(\Theta)}{\mathcal{T}_{\mathcal{S}}} \right]^{\gamma} \qquad (1)$$

where $\mathcal{T}_{\mathcal{S}}$ is the target for the model size and γ in the formulation above is defined as follow:

$$\gamma = \begin{cases} 0, & \text{if } \mathcal{S}(\Theta) \leq \mathcal{T}_{\mathcal{S}} \\ -1, & \text{otherwise} \end{cases} \qquad (2)$$
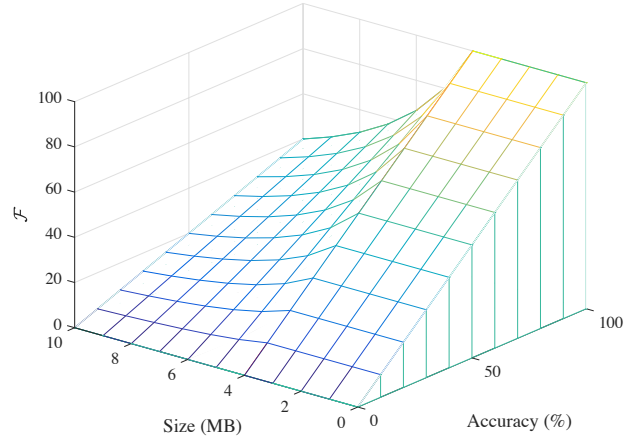

Figure 3. Multi-objective Function. Take $\mathcal{T}_{\mathcal{S}} = 4$ MB for example. When size is less than $\mathcal{T}_{\mathcal{S}}$, $\mathcal{F}$ depends only on accuracy. Otherwise, $\mathcal{F}$ sharply decreases as punishment.

It means that if the model size meets the target, we simply use accuracy as the objective function. It degrades to a single objective problem. Otherwise, the objective value is penalized sharply to discourage the excessive model size. We visualize the multi-objective function in Fig. 3.

The search task is converted into finding a neural architecture $\mathscr{A}$ and a quantization policy $\mathscr{P}$ to construct an optimal model $\Theta = \{\mathscr{A}, \mathscr{P}\}$ that maximizes the objective Eq. (1). In experiments, we first show the effectiveness of the learned quantization policies by fixing the network architecture $\mathscr{A}$ as classical hand-crafted networks. After that, the whole search space is explored as described in Section 3.2.

3.2. Search Space

Our search space can be partitioned into neural architecture search space and quantization search space, $\mathbb{S} = \{\mathbb{S}_{\mathscr{A}}, \mathbb{S}_{\mathscr{P}}\}$. In this section, we first introduce them respectively and then summarize our total search space in details.

For neural architecture search space $\mathbb{S}_{\mathscr{A}}$, we follow the NASNet search space [35]. This search space has been

widely used by many well-known methods [24, 27, 19, 20] and thus it is fair for comparison. This cell-wise search space consists of two kinds of Inception-like modules, called the *normal cells* and the *reduction cells*. When taking in a feature map as input, the *normal cells* return a feature map of the same dimension. The *reduction cells* return a feature map with its height and width reduced by a factor of two. These cells are stacked in certain patterns for CIFAR-10 and ImageNet respectively as shown in Fig. 2. The resulting architecture is determined by the normal cell structure and the reduction cell structure, the first convolution channels (F) and cell stacking number (N). Only the structure of the cells are altered during search. Each cell is a directed acyclic graph consisting of combinations. A single combination takes two inputs and applies an operation to each of them. Therefore, each combination can be specified by two inputs and two operations, $\{i_1, i_2, o_1, o_2\}$. The combination output is the addition of them and all combination outputs are concatenated as the cell output.

For quantization policy $\mathbb{S}_{\mathscr{P}}$, we aim to find optimal quantization bit for each cell. As shown in Fig. 2, there are $k = 3 \cdot N + 2$ cells in the CIFAR-10 architecture. Thus, the problem is convert into searching for a string of bits for these cells $\mathscr{P} = \{b_1, b_2, ..., b_k\}$.

In our implementation, we conduct search with a string of code to represent our total search space $\mathbb{S}$. As the neural architecture is determined by the normal cell and the reduction cell, each model is specified by the normal cell structure and the reduction cell structure, $\mathbb{S}_{\mathscr{A}} = \{\mathscr{A}_{\text{nom}}, \mathscr{A}_{\text{rec}}\}$. As mentioned above, the normal cell structure contains $k = 3 \cdot N + 2$ combinations, that is, $\mathscr{A}_{\text{nom}} = \{C_1, C_2, ..., C_k\}_{\text{nom}}$ and the reduction cell structure is same. A combination is specified by two inputs and two operations, which is presented as $C_j = \{i_1, i_2, o_1, o_2\}_j$. The choices of architecture operations o and quantization levels b are shown below:

• *Architecture*: 3x3 separable conv, 5x5 separable conv, 3x3 avg pooling, 3x3 max pooling, zero, identity.

• *Quantization*: 4 bit, 8 bit, 16 bit.

Assuming there are $\#\mathbb{S}_{\mathscr{A}}$ possible architectures and $\#\mathbb{S}_{\mathscr{P}}$ possible compression heuristics respectively, the total complexity of our search space is $\#\mathbb{S}_{\mathscr{A}} \cdot \#\mathbb{S}_{\mathscr{P}}$. In experiments, we search on CIFAR-10 and the cell stacking number ($\mathbb{N}$) is 6. As in Fig. 2, there are $6 \times 3 + 2 = 20$ cells in each model and $\#\mathbb{S}_{\mathscr{P}}$ equals to $3^{20} = 3.5 \times 10^9$. For architecture search space, all our comparison methods and our approach follow. NASNet [35]. Thus, our total search space is 3.5×10^9 times large as that of comparison methods.

3.3. Search Strategy

We employ a classical evolutionary algorithm, *tournament selection* [10]. A population of models $\mathbf{P}$ is first initialized randomly. For any model Θ, we need to optimize its

---

Algorithm 1: Search Strategy

> **input** : population size #P, sample size #S,
> training set D_{train}, validation set D_{val},
> max num epochs #E
> **output:** a population of models $\mathbf{P}$

1   $\mathbf{P}^{(0)} \leftarrow$ `initialize(#E)`
2   **for** *i=1:#E* **do**
3      $\mathbf{S}^{(i)} \leftarrow$ `sample(`$\mathbf{P}^{(i-1)}$`, #S)`
4      $\Theta_{\text{best}}, \Theta_{\text{worst}} \leftarrow$ `select(`$\mathbf{S}^{(i)}$`)`
5      $\mathscr{A}_{\text{mut}} \leftarrow$ `mutate(`$\mathscr{A}_{\text{best}}$`)`
6      $\mathscr{P}_{\text{mut}} \leftarrow$ `mutate(`$\mathscr{P}_{\text{best}}$`)`
7      $\Theta_{\text{mut}} \leftarrow$ `train(`D_{train}`, `$\mathscr{A}_{\text{mut}}$`)`
8      $\mathcal{S}(\Theta_{\text{mut}}) \leftarrow$ `quantize(`$\Theta_{\text{mut}}, \mathscr{P}_{\text{mut}}$`)`
9      $\alpha(\Theta_{\text{mut}}) \leftarrow$ `test(`$\Theta_{\text{mut}}, D_{val}$`)`
10     $\mathcal{F}(\Theta_{\text{mut}}) \leftarrow$ `Eq.(1)(`$\alpha(\Theta_{\text{mut}}), \mathcal{S}(\Theta_{\text{mut}})$`)`
11     $\mathbf{P}^{(i-1)} \leftarrow$ `push(`$\mathbf{P}^{(i-1)}, \Theta_{\text{mut}}$`)`
12     $\mathbf{P}^{(i)} \leftarrow$ `pop(`$\mathbf{P}^{(i-1)}, \Theta_{\text{worst}}$`)`
13 **end**

---

architecture $\mathscr{A}$ and quantization policy $\mathscr{P}$. Each individual model Θ of $\mathbf{P}$ is first trained on the training set D_{train}, quantized as its compression strategy and then evaluated on the validation set D_{val}. Combined with its model size $\mathcal{S}(\Theta)$, its fitness $\mathcal{F}(\Theta)$ is computed as Eq. (1). At each evolutionary step, a subset $\mathbf{S}$ is randomly sampled from $\mathbf{P}$. According to their fitnesses, we can select the best individual Θ_{best} and the worst individual Θ_{worst} among $\mathbf{S}$. Θ_{worst} is then excluded from $\mathbf{P}$ and Θ_{best} becomes a parent and produces a child Θ_{mut} with *mutation*. Θ_{mut} is then trained, quantized and evaluated to measure its fitness $\mathcal{F}(\Theta)$. Afterwards Θ_{mut} is pushed into $\mathbf{P}$. This scheme actually keeps repeating competitions of random samples in iterations. The procedure is formulated in Algorithm 1.

Specially, *mutation* is conducted to the neural architecture $\mathscr{A}$ and the quantization policy $\mathscr{P}$ respectively in each iteration. For neural architecture $\mathscr{A}$, we make mutations to each combination in the cells, that is to randomly choose one from $\{i_1, i_2, o_1, o_2\}$, and then replace it with a random substitute. For quantization policy $\mathscr{P}$, *mutation* is to randomly pick one from $\{b_1, b_2, ..., b_k\}$ and reset it as a random choice of quantization bits.

3.4. Quantization Details

In this section, we introduce the quantization process in details. Given a weight vector ω and the quantization bit b, the quantization process can be formulated as follow:

$$\hat{w} = \mathcal{L}^{-1}(Q(\mathcal{L}(w), b)) \tag{3}$$

where $\mathcal{L}(w) = \frac{w - \mu}{\nu}$ is a linear scaling function [13] that normalizes arbitrary vectors into the range [0,1] and $\mathcal{L}^{-1}$ is the inverse function. Specially, as the whole parameter vector usually has a huge dimension, magnitude imbalance

Table 1. The results of quantization policy search for existing networks on ImageNet. Here we compare to 8 bits models and float models. Numbers in brackets are Acc increase and Size compression ratio compared to float models.

	Ours		8 bits		Float	
	Acc/%	Size/MB	Acc/%	Size/MB	Acc/%	Size/MB
ResNet18 [12]	**70.02 (+0.26)**	**7.21 (6.49x)**	69.64 (-0.12)	11.47 (4.08x)	69.76	46.76
ResNet34 [12]	**73.77 (+0.46)**	**11.92 (7.31x)**	73.23 (-0.08)	21.32 (4.09x)	73.31	87.19
ResNet50 [12]	**76.39 (+0.26)**	**14.91 (6.86x)**	76.15 (+0.02)	24.74 (4.13x)	76.13	102.23
ResNet101 [12]	**78.13 (+0.76)**	**31.54 (5.65x)**	77.27 (-0.10)	43.19 (4.12x)	77.37	178.20
ResNet152 [12]	**78.86 (+0.55)**	**46.63 (5.16x)**	78.30 (-0.01)	58.38 (4.12x)	78.31	240.77
DenseNet-121 [16]	**74.56 (+0.12)**	**6.15 (5.19x)**	74.44 (+0.00)	7.65 (4.17x)	74.44	31.92
DenseNet-169 [16]	**76.39 (+0.79)**	**11.89 (4.76x)**	75.45 (-0.15)	13.54 (4.18x)	75.60	56.60
DenseNet-201 [16]	**77.06 (+0.16)**	**17.24 (4.64x)**	76.92 (+0.02)	19.09 (4.19x)	76.90	80.06
MobileNet-v1* [15]	**70.59 (+1.02)**	4.10 (4.12x)	68.77 (-0.80)	**4.05 (4.18x)**	69.57	16.93
MobileNet-v2* [29]	**72.19 (+0.38)**	4.25 (3.30x)	68.06 (-3.75)	**3.45 (4.06x)**	71.81	14.02
SqueezeNet [17]	**60.01 (+1.92)**	1.22 (1.93x)	57.93 (-0.16)	**1.20 (1.96x)**	58.09	2.35

* MobileNet-v1 and MobileNet-v2 are implemented and trained by ourselves. The pre-trained models of other networks are officially provided by Pytorch.

might push most elements in the vector to zero. This would result in an extremely harm precision. To address this issue, we adopt the bucketing technique [1], that is, the scaling function is applied separately to a fixed length of consecutive values. The length is the bucket size k.

In Eq.(3), Q is the actual quantization function that only accepts values in [0,1]. For a certain element w_i and the quantization bit b, this process is shown as below:

$$Q(w_i, b) = \frac{\lfloor w_i 2^b \rfloor}{2^b} + \frac{\xi_i}{2^b} \qquad (4)$$

This function assigns the scaled value w_i to the closest quantization point and ξ_i is the rounding function as follow.

$$\xi_i = \begin{cases} 1, & \text{if } w_i 2^b - \lfloor w_i 2^b \rfloor > 0.5 \\ 0, & \text{otherwise} \end{cases} \qquad (5)$$

Given a certain weight vector of size N and the size of full precision weight f (usually 32 bits), full precision requires fN bits in total to store this vector. As we use b bits per weight and two scaling parameter α and β for each budget, the quantied vector needs $bN + 2\frac{fN}{k}$ bits in total. Thus, the compressed ratio is $\frac{kf}{kb+2f}$ for this weight vector.

4. Experimental Results

In this section, we first apply our approach to existing networks and show the compression results on ImageNet. After that, we introduce the joint search results.

4.1. Quantization on Fixed Architecture

Our method can be flexibly applied on any existing networks for providing quantization policies. In this section, we report the quantization results of some classical networks on ImageNet [6]. These state-of-the-art networks include a series of ResNet [12], DensenNet [16] and some mobile size networks, e.g., MobileNet-v1 [15], MobileNet-v2 [29] and SqueezeNet [17]. For all ResNets [12],

DenseNets [16] and SqueezeNet [17], we obtain their pre-trained float models from torchvision.models class of Py-Torch. Because MobileNet-v1 [15] and MobileNet-v2 [29] models are not provided by official PyTorch, we implement and train them from scratch to get these two float models. Table 1 presents the performance of our quantization policies on the state-of-the-art networks. In the $Acc/\%$ columns, the numbers in the brackets mean the accuracy increase or decrease after compression. In the *Params/M*, the numbers in the brackets mean the compression ratio.

It is worth to note that our method can effectively improve the accuracy and compress the model size. Taking ResNet18 [12] for example, the model generated by our method has 70.02% accuracy that is 0.26% higher than the float model. Our compressed ResNet18 has 7.21M parameters while the float model has 46.76M parameters that is 6.49 times as ours. For all these ResNets [12] and DenseNets [16], our method can generate models that are more accurate and smaller than both 8 bits and float models. For the mobile size networks, MobileNet-v1 [15] MobileNet-v2 [29] and SqueezeNet [17], ours are slightly larger than 8 bits models, but much more accurate than both the 8 bits and the float models.

In addition, we also compare our results to other compression strategies in Fig. 4, including 2 bits, 4 bits and 16 bits. It shows the bi-objective frontiers obtained by our results and the corresponding 2/4/8/16 bits results. A clear improvement appears that our results have much higher accuracies than 2/4 bits models and are much smaller than 8/16 bits models of ResNets [12] and DenseNets [16]. For mobile size models, i.e., MobileNet-v1 [15], MobileNet-v2 [29] and SqueezeNet [17], our results are more accurate than all bits models.

4.2. Joint Architecture Search and Quantization

The joint search is conducted on CIFAR-10 to obtain the normal cell structure $\mathscr{A}_{\text{nom}}$, the reduction cell structure
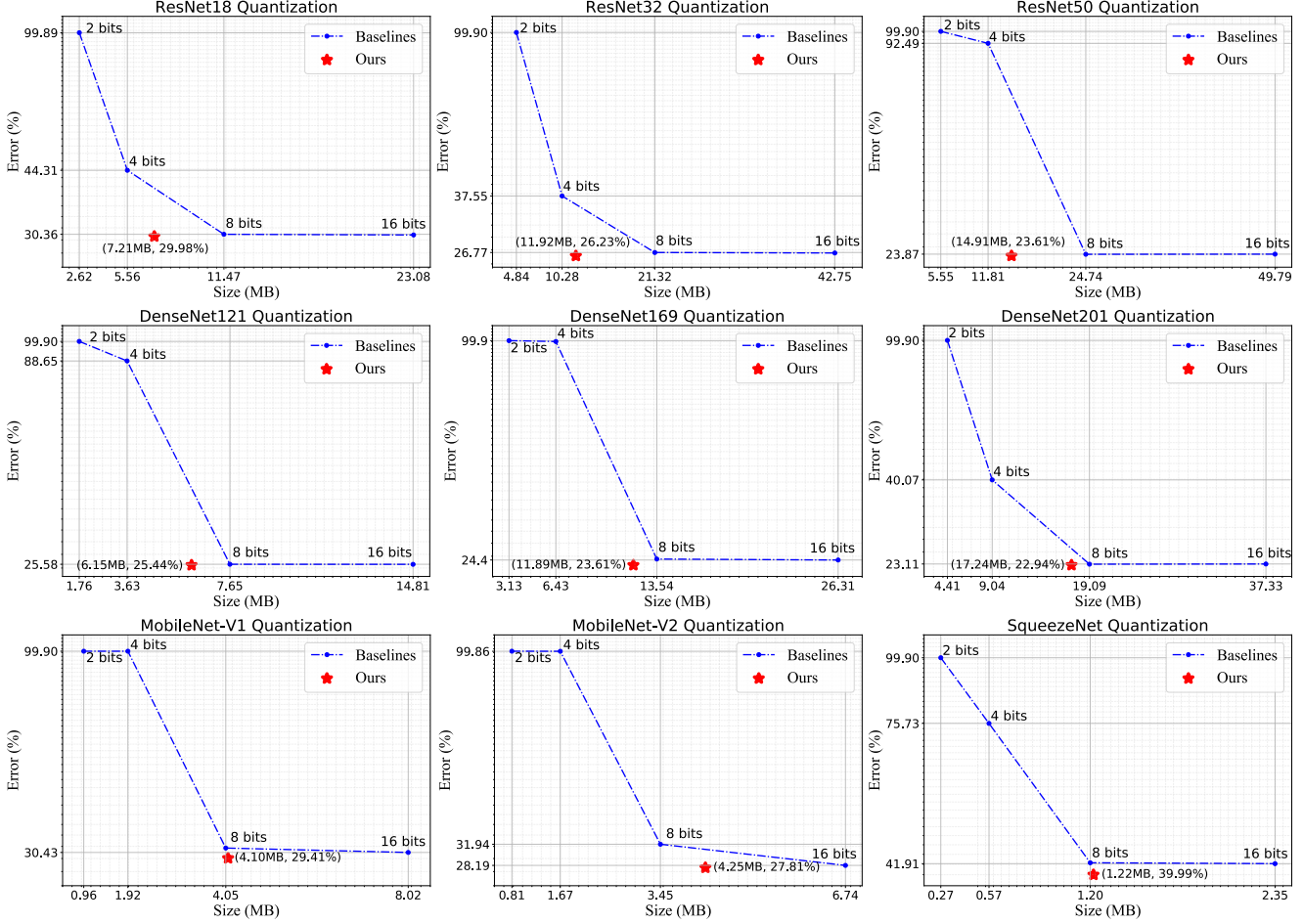
Figure 4. The results of quantization policy search for existing networks on ImageNet. Here we compare to 2bits, 4 bits, 8 bits and 16 bits models. The points of Ours are clearly under the Baselines. Models quantized by our policies have better accuracies than others.

$\mathscr{A}_{\text{rec}}$ and the quantization policy $\mathscr{P}$. After search, we retrain CIFAR-10 and ImageNet float models from scratch. CIFAR-10 results are obtained by quantizing the float models with the search quantization policy $\mathscr{P}$. As ImageNet architectures have additional cells and layers, it is unable to directly apply $\mathscr{P}$ on ImageNet float models. Thus we use $\mathscr{P}$ to initialize an evolution population to search ImageNet quantization policies as in Section 4.1.

In Table 2, we compare the performance of ours to other state-of-the-art methods that search only for neural architectures. Note that all methods listed in Table 2 use NASNet [35] architecture search space. JASQNet is obtained with $\mathcal{T}_{\mathcal{S}}$ set as 3 MB during search and JASQNet-Small is obtained with $\mathcal{T}_{S}$ set as 1 MB during search. Ours-Small(float) and JASQNet (float) are the float models before the searched quantization policies applied to them.

For the model JASQNet, it achieves competitive accuracies and relatively small model size to other comparison methods. On CIFAR-10, only NASNet-A [35] and AmoebaNet-B [27] have clearly higher accuracies than

JASQNet. But their search costs are hundreds of larger times than ours. On CIFAR-10, the model size of JASQNet is more than 4 times small as the size of other comparison models. On ImageNet, the accuracy of JASQNet is competitive to others and the model size of JASQNet is also 4 times or so small as that of other comparison models.

For the model JASQNet-Small, its model size is 10 times small as the size of other comparison models on CIFAR-10. On ImageNet, its model size is 7 or 8 times small as others. Compared to SqueezeNet [17], the model with similar size (41.91% with 2.35 MB), its accuracy is much higher.

Compared to JASQNet (float) and JASQNet-Small (float), JASQNet and JASQNet-Small has a higher accuracy and smaller model size. It shows that our learned quantization policies are effective. Compared to other only searching for architecture methods, JASQNet (float) and JASQNet-Small (float) are not best. Because our search space is much larger that includes quantization choices and it is unfair to directly compare them with our float models.

It is worth to clarify #Params and Size in Table 2.

Table 2. Comparisons to Architecture Search on CIFAR-10 and 224 ImageNet.

	Search Cost		CIFAR-10			ImageNet		
	GPUs	Days	#Params/M	Size/MB	Error/%	#Params/M	Size/MB	Error/%
PNASNet-5 [19]	100	1.5	3.2	12.8	3.41 ± 0.09	5.1	20.4	25.8
NASNet-A* [35]	500	4	3.3	13.2	2.65	5.3	21.2	26.0
NASNet-B [35]	500	4	2.6	10.4	3.73	5.3	21.2	27.2
NASNet-C [35]	500	4	3.1	12.4	3.59	4.9	19.6	27.5
AmoebaNet-B* [27]	450	7	2.8	11.2	2.55 ± 0.05	5.3	21.2	26.0
ENAS* [24]	1	0.5	4.6	18.4	2.89	-	-	-
DARTS (1st order)* [20]	1	1.5	2.9	11.6	2.94	4.9	19.6	26.9
DARTS (2nd order)*[20]	1	4	3.4	13.6	2.83 ± 0.06	-	-	-
JASQNet (float)*	1	3	3.3	13.2	2.94	4.7	18.8	27.25
JASQNet*	1	3	3.3	**2.5**	2.90	4.7	**4.9**	27.22
JASQNet-Small (float)*	1	3	1.8	7.2	3.08	2.8	11.2	34.14
JASQNet-Small*	1	3	1.8	**0.9**	2.97	2.8	**2.5**	34.10

* Training with cutout [8] on CIFAR-10. All methods use NASNet [35] architecture search space.
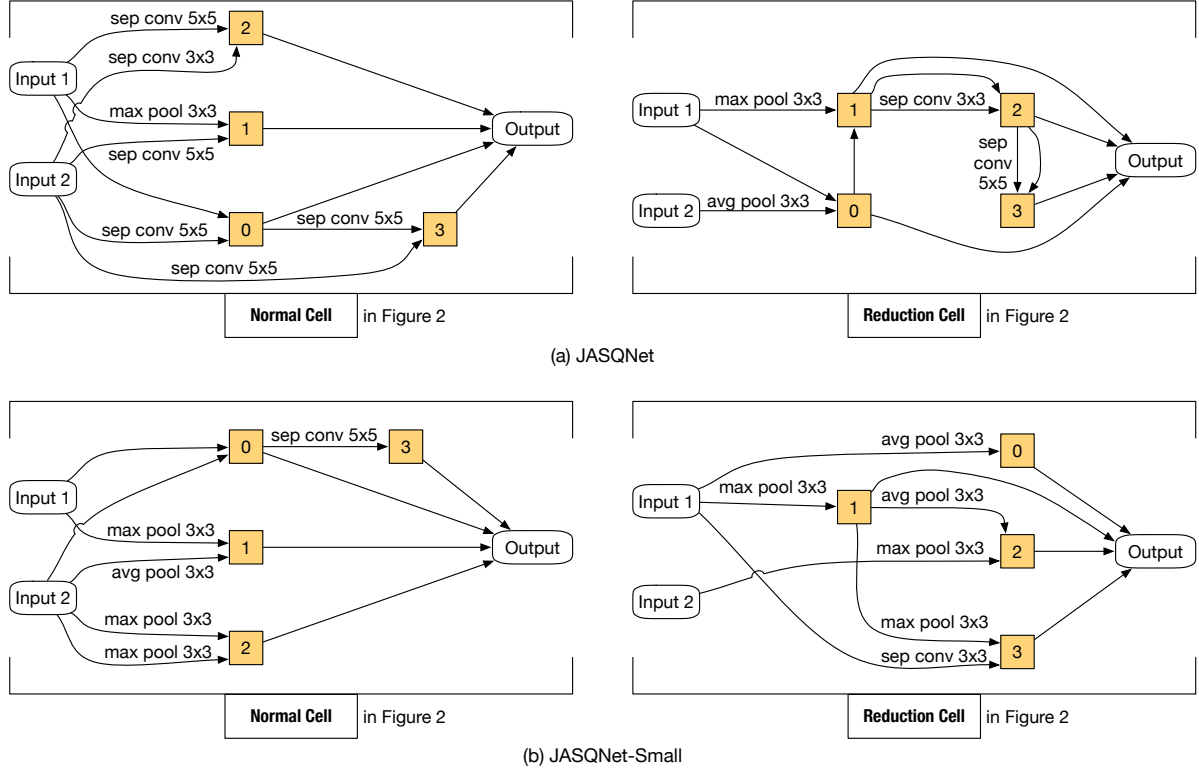

(a) JASQNet

(b) JASQNet-Small

Figure 5. JASQNet and JASQNet-Small

#Params means the number of free parameters and its unit is million (M). Size means model size for storage and its unit is MByte (MB). Quantization can reduce Size but not #Params. The result architectures are shown in Fig. 5.

### 4.3. Analyses

4.3.1 Search Process Details

Previous works [35, 24, 27, 20, 19] tend to search on small proxy networks and use wider and deeper networks in the final architecture evaluation. In Table 3, we list the depth and width of networks for search and networks for evaluation. N is the number of stacking cells in Fig. 2 and F is the initial convolution channels. Taking the width for example, DARTS [20] uses a network with initial channels 16 for search and evaluates on networks with initial channels 36. ENAS [24] searches on networks with initial channels 20 and evaluates on a network with initial channels 36.

The original purpose of searching on small proxy net-
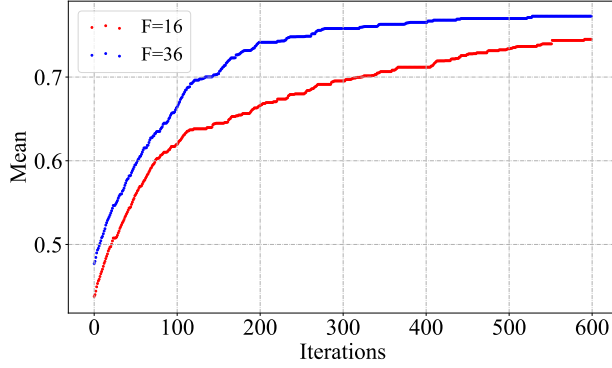
(a) Mean of Population Fitnesses
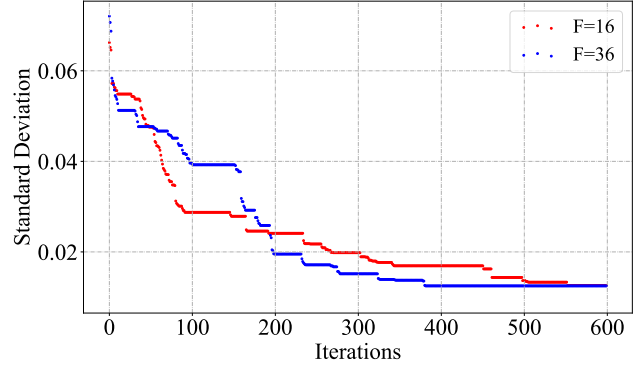
(b) Standard Deviation of Population Fitnesses

Figure 6. Ablation study on whether use small proxy networks for search.

Table 3. Depth and Width for Search and Evaluation on CIFAR-10.

	Search		Evaluation	
	F	N	F	N
PNASNet-5 [19]	24	2	48	3
NASNet [35]	32	2	32	6
AmoebaNet [27]	24	3	36	6
ENAS [24]*	20	2	36	5
DARTS [20]	16	2	36	2
JASQ	36	6	36	6

* This info is discovered in their released code but in not their paper.

works is to save time. But in our joint search experiments, we empirically find it is a bit harmful to search process. We make an ablation study on using small proxy networks as in Fig. 6. The blue line represents the experiment without small proxy networks, where the networks have the same width (F=36) and depth (N=6) to those for evaluation. The red line represents searching with small proxy networks (F=16 and N=6). We keep track of the most recent population during evolution. Fig. 6 (a) shows the highest average fitness of the population over time. Fig. 6 (b) shows the lowest standard deviation of the population fitnesses over time. Wider networks might lead to higher accuracies but it is clear that the blue line in Fig. 6 (a) converges faster than the red line. Standard deviation of the population fitnesses represents the convergence of evolution. Thus, Fig. 6 (b) also shows that searching without proxy networks leads to a faster convergence.

4.3.2 Comprehensive Comparison

Joint search performs better than only architecture search or only quantization search. JASQNet are better than only architecture search (blue squares) and only quantization search (red circles) as illustrated in Fig. 7. Models with too many parameters (DenseNets), are not shown in it. It shows that JASQNet reaches a better multi-objective position.

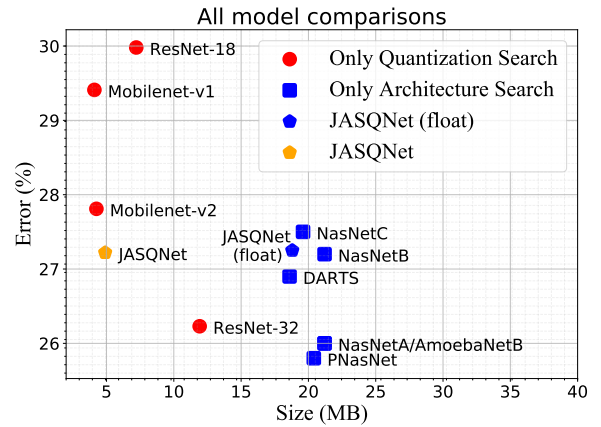In addition, as shown in results in Table 1, suitable quan-

Figure 7. Comparisons with only architecture search and only quantization search. The gap between JASQNet and JASQNet (float) shows the effectiveness of our quantization policy. JASQNet reaches a better balance point thant other models.

tization policies can improve accuracy and decrease model size simultaneously. No matter for existing networks quantization or joint search of architecture and quantization, our quantized models are more accurate than their float counterparts. In Fig. 7, we also depict JASQNet (float) as a blue pentagon. The gap between JASQNet and JASQNet (float) shows the effectiveness of our quantization policy. Their accuracies are almost same but JASQNet has much less model size.

As shown in Table 2, JASQNet (float) and JASQNet-Small (float) are not better than NASNet [35] or AmoebaNet [27]. The first reason is that joint search results in larger search space that might harm the quality of searched architectures. The second possible reason is that their search processes spend much more computation resources than ours.

5. Conclusion

Searching for both architectures and compression heuristics is a direct and convenient way for deep learning practitioners. To our best knowledge, this task has never been proposed in the literature. In this work, we propose to automatically design architectures and compress models. Our method can not only conduct joint search of architectures and quantization policies, but also provide quantization policies for existing networks. The models generated by our method, JASQNet and JASQNet-Small, achieve better trade-offs between accuracy and model size than only architecture search or only quantization search.

Appendix

1) CIFAR-10 Classification

Dataset   There are 50,000 training images and 10,000 test images in CIFAR-10. 5,000 images are partitioned from the training set as a validation set. We whiten all images with the channel mean subtraction and standard deviation division. 32 x 32 patches are cropped from images and padded to 40 x 40. Horizontal flip is also used. We use this preprocessing procedures for both search and evaluation.

Training   For fair comparisons, our training hyperparameters on CIFAR-10 are identical to those of DARTS [20]. The models for evaluation are trained for 600 epochs with batch size 96 on one GPU. The version of our GPUs is Titan-XP. The initial learning rate is 0.025 and annealed down to zero following a cosine schedule. We set the momentum rate as 0.9 and set weight decay as 3×10^{-4}. Following existing works [20, 35, 27], additional enhancements include cutout [8], path dropout of probability 0.3 and auxiliary towers with weight 0.4.

2) ImageNet Classification

Dataset   The original input images are first resized and their shorter sides are randomly sampled in [256, 480] for scale augmentation [30]. We then randomly crop images into 224×224 patches. We also conduct horizontal flip, mean pixel subtraction and the standard color augmentation. These are standard augumentations that proposed in Alexnet [18]. In addition, most augmentations are excluded in the last 20 epochs with the sole exception of the crop and flip for fine-tuning.

Training   Each model is trained for 200 epochs on 4 GPUs with batch size 256. We set the momentum rate as 0.9 and set weight decay as 4×10^{-5}. We also employ an auxiliary classifier located at $\frac{2}{3}$ of the maximum depth weighted by 0.4. The initial learning rate is 0.1. It later decays with a polynomial schedule.
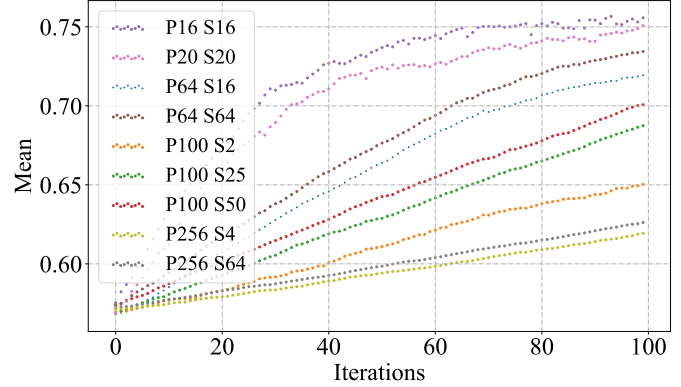

Figure 8. Hyper-parameter optimization experiments about population size and sample size. We conduct these experiments in a small scale by setting input filters F=16 and stacking cells number N=2. Each experiment runs for 100 iterations.

3) Quantization Process

Previous works [33, 22] do not quantize the first and last layers of ImageNet models to avoid severe accuracy harm. We also follow this convention on our ImageNet models and do not apply this constraint on CIFAR-10 models. Another detail is that we use Huffman encoding for quantized value representation to save additional space.

4) Search Process

The evolutionary search algorithm employed in this paper can be classified into tournament selection [10]. There are only two hype-parameters, population size #P and sample size #S. The hyper-parameter optimization process is illustrated in Figure 8. We conduct all these experiments with the same settings except #P and #S. For efficient comparison, these experiments runs in a small scale for only 100 iteration. The input filters F is set as 16 and the stacking cells number N is set as 2. This figure shows the mean fitness of models in the population over iterations. We pick the best one (#P = 16, #S = 16) from Figure 8 for the experiments in this paper. We also employ the parameter sharing technique for acceleration [24], that is, a set of parameters are shared among all individual models in the population.

References

[1] D. Alistarh, J. Li, R. Tomioka, and M. Vojnovic. QSGD: randomized quantization for communication-optimal stochastic gradient descent. *CoRR*, abs/1610.02132, 2016.

[2] J. Ba and R. Caruana. Do deep nets really need to be deep? In *NIPS*, pages 2654–2662, 2014.

[3] L. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE Trans. Pattern Anal. Mach. Intell.*, 40(4):834–848, 2018.

[4] C. Cortes, X. Gonzalvo, V. Kuznetsov, M. Mohri, and S. Yang. Adanet: Adaptive structural learning of artificial neural networks. In *ICML*, pages 874–883, 2017.

[5] K. Deb. Multi-objective optimization. In *Search methodologies*, pages 403–449. 2014.

[6] J. Deng, W. Dong, R. Socher, L. Li, K. Li, and F. Li. Imagenet: A large-scale hierarchical image database. In *CVPR*, pages 248–255, 2009.

[7] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *NIPS*, pages 1269–1277, 2014.

[8] T. Devries and G. W. Taylor. Improved regularization of convolutional neural networks with cutout. *CoRR*, abs/1708.04552, 2017.

[9] T. Elsken, J. H. Metzen, and F. Hutter. Simple and efficient architecture search for convolutional neural networks. *CoRR*, abs/1711.04528, 2017.

[10] D. E. Goldberg and K. Deb. A comparative analysis of selection schemes used in genetic algorithms. In *FGA*, pages 69–93. 1990.

[11] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. *CoRR*, abs/1510.00149, 2015.

[12] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016.

[13] Q. He, H. Wen, S. Zhou, Y. Wu, C. Yao, X. Zhou, and Y. Zou. Effective quantization methods for recurrent neural networks. *CoRR*, abs/1611.10176, 2016.

[14] G. E. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *CoRR*, abs/1503.02531, 2015.

[15] 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. *CoRR*, abs/1704.04861, 2017.

[16] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *CVPR*, pages 2261–2269, 2017.

[17] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally, and K. Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size. *CoRR*, abs/1602.07360, 2016.

[18] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, pages 1106–1114, 2012.

[19] C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L. Li, L. Fei-Fei, A. L. Yuille, J. Huang, and K. Murphy. Progressive neural architecture search. In *ECCV*, pages 19–35, 2018.

[20] H. Liu, K. Simonyan, and Y. Yang. DARTS: differentiable architecture search. *CoRR*, abs/1806.09055, 2018.

[21] J. Luo, J. Wu, and W. Lin. Thinet: A filter level pruning method for deep neural network compression. In *ICCV*, pages 5068–5076, 2017.

[22] A. K. Mishra, E. Nurvitadhi, J. J. Cook, and D. Marr. WRPN: wide reduced-precision networks. *CoRR*, abs/1709.01134, 2017.

[23] R. Negrinho and G. J. Gordon. Deeparchitect: Automatically designing and training deep architectures. *CoRR*, abs/1704.08792, 2017.

[24] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean. Efficient neural architecture search via parameter sharing. In *ICML*, pages 4092–4101, 2018.

[25] A. Polino, R. Pascanu, and D. Alistarh. Model compression via distillation and quantization. *CoRR*, abs/1802.05668, 2018.

[26] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. Xnornet: Imagenet classification using binary convolutional neural networks. In *ECCV*, pages 525–542, 2016.

[27] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le. Regularized evolution for image classifier architecture search. *CoRR*, abs/1802.01548, 2018.

[28] S. Ren, K. He, R. B. Girshick, and J. Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *IEEE Trans. Pattern Anal. Mach. Intell.*, 39(6):1137–1149, 2017.

[29] M. Sandler, A. G. Howard, M. Zhu, A. Zhmoginov, and L. Chen. Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation. *CoRR*, abs/1801.04381, 2018.

[30] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

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

[32] J. Xue, J. Li, and Y. Gong. Restructuring of deep neural network acoustic models with singular value decomposition. In *INTERSPEECH*, pages 2365–2369, 2013.

[33] C. Zhu, S. Han, H. Mao, and W. J. Dally. Trained ternary quantization. *CoRR*, abs/1612.01064, 2016.

[34] B. Zoph and Q. V. Le. Neural architecture search with reinforcement learning. *CoRR*, abs/1611.01578, 2016.

[35] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le. Learning transferable architectures for scalable image recognition. *CoRR*, abs/1707.07012, 2017.