SPARSE CNN ARCHITECTURE SEARCH (SCAS)

Yeshwanth V [*], *Ankur Deshwal* [*], *Sundeep Krishnadasan* [*], *Seungwon Lee* [†], *Joonho Song* [†]

[*] Samsung R&D Institute India-Bangalore;
[†] Samsung Advanced Institute of Technology (SAIT) Korea
{yesh.v, a.deshwal, sundeep.k, seungw.lee, joonho71.song}@samsung.com

ABSTRACT

Advent of deep neural networks has revolutionized Computer Vision. However, designing of such models with high accuracy and low computation requirements is a difficult task and needs extensive human expertise. Recent advances in Neural Architecture Search use various methods like Deep Reinforcement Learning, Evolutionary methods, Gradient Descent, HyperNetworks etc. to automatically generate neural networks with high level of accuracy. However, large size of such generated models limit their practical use. Recent findings about *lottery ticket hypothesis* suggest the existence of sparse subnetworks (*winning tickets*) which can reach the accuracy comparable to that of original dense network. In this paper, we present a method for leveraging redundancies inherent to deep Convolutional Neural Networks (CNN) to guide the generation of sparse CNN models (to find the architectures with winning tickets) without significant loss in accuracy. We evaluate our proposed method with different NAS methods on CIFAR-10, CIFAR-100 and MNIST datasets. Our results show a reduction ranging from $2\times$ to $12\times$ in terms of model size and $2\times$ to $19\times$ in terms of number of MAC operations with less than 1% drop in accuracy.

***Index Terms*—** Computer Vision, AutoML, Neural Architecture Search, Sparse
'

1. INTRODUCTION

Convolutional Neural Networks (CNN) have facilitated the automated representation learning as compared to extracting hand crafted features from raw input data [1]. However, designing the state of art neural network architectures requires substantial human expertise. Recent advancement of a set of techniques, termed as Neural Architecture Search (NAS), aims to automate the designing of complex neural networks for a given dataset and a cost function [2].

NAS is a rapidly evolving field. Different optimization techniques e.g. reinforcement learning, gradient descent, hypernetworks, evolutionary methods etc. have been used to automatically generate the models with near state of the art results on the vision and text datasets. However, little attention has been paid to study the size of models generated. A lot of impressive results are gained at the cost of models which are order of magnitude larger than the hand crafted versions. In order to facilitate rapid deployment of neural networks in mobile and embedded devices it is very important to automate the design neural network architectures that are compact and perform efficiently while inferencing.

Neural Architecture Search. MetaQNN [3] is one of the initial works in field of NAS. It uses Q-learning with an ϵ-greedy exploration strategy and experience replay for searching of neural network. Authors of [4] propose the use of policy gradient method with Recurrent Neural Network (RNN), termed as controller, to encode the policy. It samples models, termed as child models, and uses their validation accuracy after training to scale the gradients for policy improvement. The work presented above suffered from extremely high computational and memory resource requirements e.g. [4] used 450 GPUs for 3-4 days (i.e. 32,400-43,200 GPU hours) to reach the reported level of accuracy.

To counter the problem of high resource requirements, ENAS [5] uses parameter sharing of the candidate models sampled by the controller RNN in policy gradient based RL setting. ENAS brings down the resource requirements for network architecture search by 3 order of magnitude. Alternative methods to RL were previously proposed for NAS. Work presented in [6] applies HyperNet [7] to train an auxiliary model which dynamically generates weights of a main model with variable architecture. The search space is limited in this case as compared to the RL based techniques. In [8], authors use continuous relaxation representation of connections between convolution blocks. It then uses gradient descent for architecture search. Recent work including MnasNet [9], RENA [10] use multiple approaches to achieve state of the art architectures with computation constraints.

N2N Learning [11] take alternative approach to generation of Neural Architectures from scratch using network distillation. Work presented in [12] uses RL to predict the amount of fine grained sparsity possible in each layer of an input network. Although the methods [11] and [12] focus on optimizing computation and energy requirements, these techniques miss the opportunity of generating models from scratch. This limits their applicability only to the cases where state of the art deep

978-1-7281-1331-9/20/$31.00

learning models already exist.

Sparse [13] presents a multi objective BO approach for generating neural network with sparsity. Our method can be applied to multiple optimization techniques as demonstrated in this paper.

Pruning. Pruning is a popular method for removing redundancies as well as to improve generalization in deep neural networks. After the advent of CNNs in modern applications in embedded devices there is a growing interest in optimizing CNN models to fit in limited resource environments. Pruning redundant connections is one of the go to methods in Model Compression and Acceleration of neural networks. Work presented in [14] and [15] use an iterative threshold based pruning and retraining to learn the weights and simultaneously prune the unimportant weights. Authors of [16] go for a more structured way of pruning by pruning at filter level and group level respectively.

Lottery Ticket Hypothesis. [17] provides a new perspective on how a large dense network has sparse sub networks which if found can be trained efficiently to reach the same accuracy as the large dense model would reach in same number of training iterations. A pruning approach is used to discover such sparse subnetworks. Further, the lottery ticket hypothesis is extended into a conjecture that dense networks have more possible subnetworks from which a winning ticket can be discovered hence can be trained easier than training the sparse network.

Contributions. In this paper, we explore the scope of generating high performing sparse neural networks i.e. to find architectures with winning tickets, using multiple NAS techniques by introducing constraints to favor sparse architectures. We evaluate our method with distinct approaches for NAS introduced in [5], [6] and [8].

Our contribution can be summarized as follows:

- We present a novel technique to guide NAS methods to generate models with high parameter sparsity (to find the architectures with winning tickets) while retaining the accuracy at par with the state of the art models.
- We study the relative effectiveness of popular NAS techniques on CIFAR-10, CIFAR-100 and MNIST datasets.
- We evaluate the robustness of our method by extensive experimentation with NAS techniques that are based on orthogonal approaches. i.e. ENAS(using Deep Reinforcement Learning), DARTS(using Gradient Descent), and SMASH(using HyperNetworks).

Parameter sparsity leads to proportional improvement of memory requirements as well as performance on AI accelerators/hardware architectures which can take advantage of such sparsity. We note that while our focus is on CNN due to its central role in computer vision, the technique is generic enough and can be extended to other types of deep neural networks. Also, as our method is generic enough to work along with NAS techniques based on different principles, it can be extended to other NAS methods with minimal effort.

2. SPARSE CNN ARCHITECTURE SEARCH

In this section, we discuss our methodology of modifying different NAS techniques to generate sparse neural network architectures. General idea is to introduce sparsity in the models being searched thus directing the search process towards sparse yet high performing architectures. The flow of our method is illustrated in Figure 1.

Fig. 1. Key components of SCAS.

The NAS controller node hosts the NAS method. In case of ENAS, this is an RNN controller. In case of DARTS, this is the weights of the architecture encoding. In case of SMASH, this is the HyperNet that generates the weights for child networks. At each training step of the controller, child models are sampled and given to the trainer node. As the trainer trains the child models, it is interleaved with a pruner which introduces sparsity by pruning the weights up to a preset level. A feedback is provided to the NAS controller based on which it learns to generate better architectures. In ENAS this is done by updating the RNN controller using REINFORCE algorithm [18] with validation accuracy of the child models as reward. In case of DARTS, the weights encoding the architecture are updated using gradient descent in order to maximize the validation accuracy of the child model. In case of SMASH, as the weights are generated by the HyperNet, they are not trained in the trainer. The pruner prunes the generated weights up to the specified target sparsity and the training error of this pruned child model is back propagated to the HyperNet.

In sections 2.1, 2.2 and 2.3, we present the implementation details for our methodology applied to ENAS [5], DARTS [8], and SMASH [6] NAS techniques respectively. Finally, in section 2.4 we outline the general pruning method used as an interleaving subroutine in each of the NAS techniques.

2.1. ENAS

In this section, we explain our proposed changes that directs ENAS [5] to search for sparse networks.

ENAS is reinforcement learning based NAS method. ENAS consists of an LSTM network setup in auto-regressive mode. Each episode generates a CNN model, which is evaluated and the accuracy level achieved is used as a reward to

update LSTM network to generate better models.There are two methods of framing the search used by ENAS, introduced below (details available at [5]). We present results with both the methods.

(a) Macro search space: In this case, designing the entire network is considered as the RL task and search space for RL agent includes the configuration of the layers of generated network.

(b) Micro search space: In case of micro search, objective is to design a computational cell. ENAS generates two types of cell named convolution cell and reduction cells. To generate the full network, 'N' convolution cells are followed by one reduction cell.

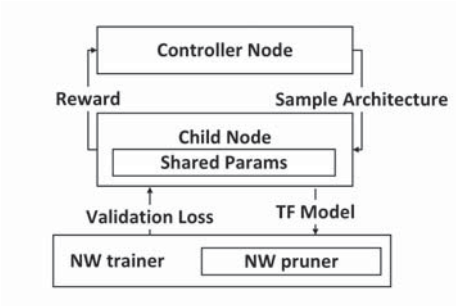


Fig. 2. SCAS architecture using ENAS

Algorithm 1 SCAS on ENAS

1: Initialize controller network θ and shared parameters for child network ω
2: **loop**
3: Fix controller's policy $\pi(\theta)$ and sample m networks
4: Create computation graph using shared parameters ω
5: **for** an epoch of training data **do**
6: Update weights ω (using Monte Carlo estimate)
7: Prune the child model up to the target sparsity s.
8: **end for**
9: Fix pruned weights $\omega(s)$
10: Update $\pi(\theta)$ to maximize the expected reward $E_{m\sim\pi(\theta)}[R(m;\omega(s))]$
11: **end loop**
12: Sample architectures from the trained controller network
13: Train them from scratch along with a sparsity constraint

In the subsequent text, we describe our proposed changes to ENAS to generate sparse models (Algorithm 1). Figure 2 shows our setup. As in ENAS, controller node encapsulates the policy in the form of LSTM model. To start an iteration, controller node samples m CNN network architectures and passes them to the child node in textual form. In RL terminology, this constitutes m $'rollouts'$. The child node then creates computation graphs for the sampled architectures. The shared parameters of the sampled models are trained by network trainer. It then prunes all networks according to the target pruning level. The models are then retrained on a subset of dataset in order to regain the drop in accuracy due to pruning. Cross-entropy loss, $L(m_{\mathrm{i}}, \omega)$ for each model m_{i} is calculated on validation dataset, where ω represents the shared parameters of the models.

The gradient calculated above is provided as the signal to LSTM to update it's parameters θ using REINFORCE [18] algorithm. To reduce the variation due to noisy gradients, we use moving average baseline, which is a standard practice to stabilize and accelerate training in REINFORCE. The method is summarized in algorithm 1.

2.2. DARTS

DARTS [8] implements NAS by representing the architecture in continuous space and use gradient descent to perform a joint optimization of the architecture and its weights.

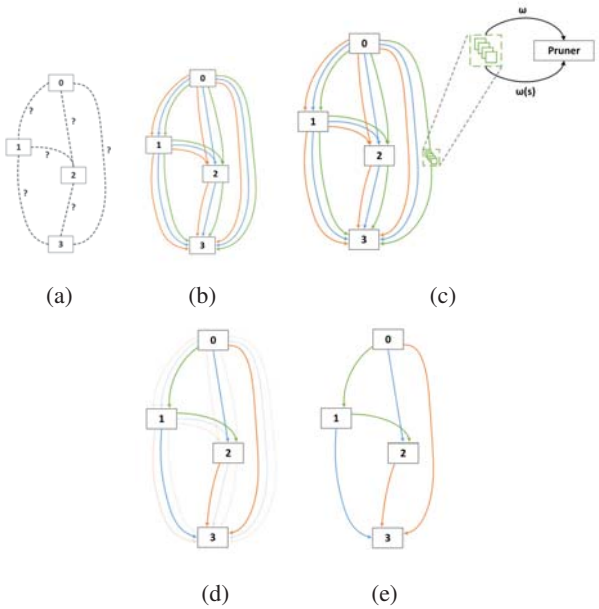


Fig. 3. SCAS architecture using DARTS: (a) Operations on the edges are initially unknown. (b) Continuous relaxation of search space by replacing every edge with a mixed operation. (c) A pruning subroutine is interleaved with the network weight optimization method. (d) Joint optimization of ω and α. (e) Extracting the final architecture from the learned α parameters (Best viewed in color).

The architecture encoding α and the weights within all the operations ω are jointly learned as a bi-level optimization problem with α as upper-level variable minimizing validation loss L_{val} and ω as the lower-level variable minimizing the training loss L_{train} (ref equation (3), (4) in [8]). To drive DARTS to generate high performing sparse networks, we in-

Algorithm 2 SCAS on DARTS

1: Create a mixed operation $\bar{o}^{(i,j)}$ parameterized by $\alpha^{(i,j)}$ for each edge (i, j)
2: **loop**
3: Update weights ω by freezing α.
$\omega \leftarrow \omega - \xi \nabla_{\omega} L_{train}(\omega, \alpha)$
4: Prune weights ω to a given target sparsity s.
5: Update architecture parameters α by freezing sparse weights $\omega(s)$ and descending $\nabla_{\alpha} L_{val}(\omega(s), \alpha)$
6: **end loop**
7: Replace $\bar{o}^{(i,j)}$ with most likely operation.
$\bar{o}^{(i,j)} = argmax_{o \in O} \alpha_o^{(i,j)}$ for each edge (i, j)

troduce sparsity in the weights of every operation. Let $\omega(s)$ be sparse weights where s denotes the level of sparsity introduced implying s percent of the weights ω are zeros. The final optimization problem reduces to:

$$\min_{\alpha} \; L_{val}(\omega^*(\alpha, \mathbf{s}), \alpha) \tag{1}$$

$$s.t. \; \omega^*(\alpha, \mathbf{s}) = argmin_{\omega} \; L_{train}(\omega(\mathbf{s}), \alpha) \tag{2}$$

This bi-level optimization is solved by an approximate iterative procedure where ω and α are optimized by alternating between the gradient descent steps in weight and architecture spaces respectively. At the end of search, the most likely operation is taken to build the final architecture. By introducing sparsity constraint on the weights we can optimize the architecture parameters α for the sparse weights $\omega(s)$ hence achieving sparse yet high performing models. The method for applying SCAS on optimization procedure of DARTS is illustrated in figure 3, and summarized in algorithm 2.

2.3. SMASH

SMASH [6] takes a different approach towards NAS. Instead of using a controller to generate architectures, it uses an auxiliary HyperNet that generates the weights for the model, based on the architecture that is given as input.

In order to generate sparse architectures, we propose interleaving SMASH method with a subroutine that prunes the weights generated by the HyperNet. Thus the evaluation of relative validation accuracy of candidate architectures is performed for the networks which perform well despite a sparsity constraint. The method is outlined in algorithm 3. In our proposed method, the HyperNet is trained to generate weights that show resilience to pruning. Consequently, the architectures which work better with sparsity constraint are scored higher when they are compared based on relative validation accuracy.

2.4. Pruning Method

We use the gradual pruning algorithm that prunes the smallest magnitude weights to achieve a preset level of network sparsity

Algorithm 3 SCAS on SMASH

Require: R_c - Space of all candidate architectures, s - Target sparsity
Ensure: c^* - Best performing model at given level of sparsity
1: Initialize HyperNet weights H
2: **loop**
3: Sample a random architecture c and minibatch x_i
4: Get architecture weights from HyperNet $\omega = H(c)$
5: Prune the weights to the given target sparsity
6: Get training error $E_t = f_c(\omega(s), x_i)$, and update H using backprop
7: **end loop**
8: **loop**
9: Sample random architecture c
10: Get architecture weights from HyperNet $\omega(s) = H(c)$
11: Prune the weights to the given target sparsity and get sparse weights $\omega(s)$
12: Evaluate error on validation set $E_v = f_c(\omega(s), x_v)$
13: **end loop**
14: Fix architecture with minimum relative validation error and train along with a target sparsity s

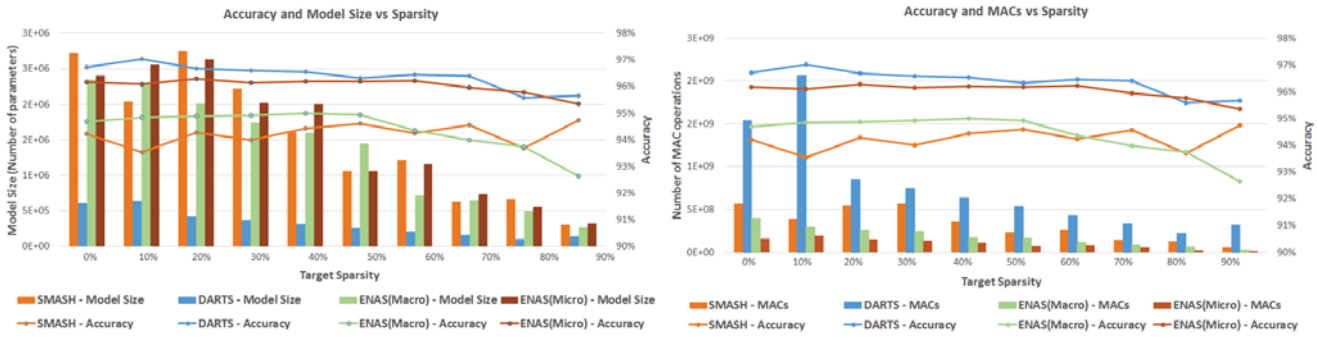
following the implementation of [19]. For each layer to be pruned, a binary mask variable of the same shape as the weight tensor is maintained. This mask specifies whether the corresponding weight is pruned or not. This enables the conditional blocking of the gradient flow during back-propagation. Hence, only the unpruned weights are updated. In case of ENAS, we follow the implementation presented in [19], which is an extension in Tensorflow library. For introducing pruning in DARTS and SMASH we use a custom implemented pruning subroutine in PyTorch.

The idea is to maintain a threshold th_l for each layer l. The weights whose magnitude is less than this threshold shall be pruned. As the training progresses, this threshold is increased or decreased depending upon the set target sparsity and the sparsity achieved. This is equivalent to the procedure described in [19] i.e. sort the weights in each layer based on their magnitude and in subsequent iterations gradually remove a fraction of weights in increasing order of their magnitudes until the target sparsity is achieved.

3. EXPERIMENTS AND RESULTS

We apply our proposed method to CIFAR-10, CIFAR-100 and MNIST datasets. These datasets are widely used in NAS literature and hence is helpful for benchmarking with published results. We explain the key insights based on the experiments on CIFAR-10 dataset and summarize the results on others.

We use the implementations of [5], [8], and [6] as our baseline. The default hyperparameters, input transformations, data augmentation and regularization techniques are unchanged from the original implementations by the corresponding au-



(a) (b)

Fig. 4. (a) Trend of Accuracy and Model Size vs Target Sparsity of different NAS models on CIFAR-10 dataset. (b) Trend of Accuracy and Multiply and Accumulate (MAC) operations vs Target Sparsity of different NAS models on CIFAR-10 dataset (Best viewed in color).

Table 1. The reduction in Model Size and MAC operations Achieved by SCAS with different NAS techniques on different datasets with <1% drop in accuracy. Baseline columns show the Model Size (in million parameters) and number of MAC operations (in millions) when no sparsity constraint is applied. Next columns show the best model found by SCAS with less than 1% drop in accuracy. Redn columns show the reduction achieved.

Dataset	**NAS method**	**Model Size**			**MAC Operations**		
		Baseline	<1% drop	Redn	Baseline	<1% drop	Redn
CIFAR10	ENAS-Macro	2.35M	0.50M	4.7x	395.35M	60.73M	6.5x
	ENAS-Micro	2.40M	0.33M	7.3x	162.84M	20.41M	8.0x
	DARTS	0.61M	0.16M	3.9x	1542.26M	329.70M	4.7x
	SMASH	2.72M	0.30M	9.1x	567.43M	57.90M	9.8x
CIFAR100	ENAS-Macro	2.76M	1.40M	2.0x	314.94M	181.92M	1.7x
	ENAS-Micro	3.05M	1.03M	3.0x	218.54M	77.98M	2.8x
	DARTS	0.36M	0.06M	5.6x	109.91M	57.77M	1.9x
	SMASH	3.14M	0.26M	12.3x	759.66M	115.56M	6.6x
MNIST	ENAS-Macro	2.48M	0.21M	11.6x	221.46M	23.36M	9.5x
	ENAS-Micro	2.84M	0.26M	11.1x	229.26M	11.89M	19.3x
	DARTS	0.62M	0.10M	6.4x	1620.15M	312.35M	5.2x
	SMASH	2.58M	0.24M	10.8x	658.34M	45.67M	14.4x

thors. To each of these implementations, we apply the pruning method mentioned earlier with different levels of target sparsity ranging from 0% to 90% at regular intervals of 10%.

Figure 4a demonstrates the trend in accuracy and model size of the best models found for CIFAR-10 dataset by different NAS methods as the sparsity constraint is increased from 0% to 90%. The key observation here is that the drop in accuracy is not as steep as the reduction in model size implying that we can automatically design high performing sparse neural network architectures using our method.

Among the different NAS methods, we observe that SCAS-DARTS gives the best performing models with much less parameters among the evaluated methods. On the other hand, if we examine the trend in number of Multiply and Accumulate (MAC) operations in figure 5b, we can observe that SCAS-DARTS is the most expensive among all the evaluated methods while SCAS-ENAS (micro) outperforms other methods. This can be attributed to the choice of search space by the NAS method e.g. while comparing two models with same sizes of convolution parameters, a model with bigger feature map requires more computation with same number of convolution parameters. The search space used in DARTS uses lesser reduction operations and thus results in bigger feature maps and more number of MAC operations in spite of having least number of parameters.

We now summarize the effectiveness of SCAS on different datasets (MNIST, CIFAR-10, CIFAR-100) in 1. Baseline metrics correspond to the best model found by the respective

NAS technique with default hyper parameters and no sparsity constraint. The next columns show the number of parameters and the number of MAC operations in the best model found with sparsity constraint that falls within less than 1% drop in validation accuracy.

We observe up-to ***11.6X*** and ***9.5X*** reductions in model size and MAC operations respectively compared with SCAS-ENAS with macro search space settings. Comparing with SCAS-ENAS with micro search space, ***11.1X*** and ***19.3X*** reductions were observed. Similarly, while comparing with SCAS-DARTS, improvement of ***6.4X*** and ***5.2X*** is observed. In case of SCAS-SMASH, ***10.8X*** and ***14.4X*** reductions in parameters and MACs is observed.

We observe while comparing the absolute model sizes in context of different datasets, SCAS-DARTS achieved smallest models consistently. Also, except the case of CIFAR-10, SCAS-DARTS also results in model with smallest computation requirements.

4. CONCLUSION

We presented a simple and effective technique for directing different NAS methods to generate considerably sparse CNN models (the architectures with winning tickets) with less than 1% drop in accuracy. We compared the effectiveness of our method along with different NAS methods on CIFAR-10, CIFAR-100 and MNIST datasets and presented our insights.

Automatic generation of optimized CNN models can help in widespread deployment of CNN models towards various applications in embedded devices including smart phones, watches and IoT devices.

5. REFERENCES

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds., pp. 1097–1105. Curran Associates, Inc., 2012.

[2] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter, "Neural architecture search: A survey," *arXiv preprint arXiv:1808.05377*, 2018.

[3] Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar, "Designing neural network architectures using reinforcement learning," *CoRR*, vol. abs/1611.02167, 2016.

[4] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le, "Learning transferable architectures for scalable image recognition," *CoRR*, vol. abs/1707.07012, 2017.

[5] Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, and Jeff Dean, "Efficient neural architecture search via parameter sharing," *CoRR*, vol. abs/1802.03268, 2018.

[6] Andrew Brock, Theodore Lim, James M. Ritchie, and Nick Weston, "SMASH: one-shot model architecture search through hypernetworks," *CoRR*, vol. abs/1708.05344, 2017.

[7] David Ha, Andrew M. Dai, and Quoc V. Le, "Hypernetworks," *CoRR*, vol. abs/1609.09106, 2016.

[8] Hanxiao Liu, Karen Simonyan, and Yiming Yang, "DARTS: differentiable architecture search," *CoRR*, vol. abs/1806.09055, 2018.

[9] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, and Quoc V. Le, "Mnasnet: Platform-aware neural architecture search for mobile," *CoRR*, vol. abs/1807.11626, 2018.

[10] Yanqi Zhou, Siavash Ebrahimi, Sercan Ömer Arik, Haonan Yu, Hairong Liu, and Greg Diamos, "Resource-efficient neural architect," *CoRR*, vol. abs/1806.07912, 2018.

[11] Anubhav Ashok, Nicholas Rhinehart, Fares Beainy, and Kris M. Kitani, "N2N learning: Network to network compression via policy gradient reinforcement learning," *CoRR*, vol. abs/1709.06030, 2017.

[12] Yihui He and Song Han, "ADC: automated deep compression and acceleration with reinforcement learning," *CoRR*, vol. abs/1802.03494, 2018.

[13] Igor Fedorov, Ryan P Adams, Matthew Mattina, and Paul Whatmough, "Sparse: Sparse architecture search for cnns on resource-constrained microcontrollers," in *Advances in Neural Information Processing Systems*, 2019, pp. 4978–4990.

[14] Song Han, Jeff Pool, John Tran, and William Dally, "Learning both weights and connections for efficient neural network," in *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds., pp. 1135–1143. Curran Associates, Inc., 2015.

[15] Song Han, Huizi Mao, and William J Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[16] Vadim Lebedev and Victor Lempitsky, "Fast convnets using group-wise brain damage," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 2554–2564.

[17] Jonathan Frankle and Michael Carbin, "The lottery ticket hypothesis: Training pruned neural networks," *CoRR*, vol. abs/1803.03635, 2018.

[18] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in neural information processing systems*, 2000, pp. 1057–1063.

[19] Michael H. Zhu and Suyog Gupta, "To prune, or not to prune: Exploring the efficacy of pruning for model compression," 2018.