MSNet: Structural Wired Neural Architecture Search for Internet of Things

Hsin-Pai Cheng[1], Tunhou Zhang[1], Yukun Yang[1], Feng Yan[2], Harris Teague[3], Yiran Chen[1], and Hai Li[1]

[1]ECE Department, Duke University, Durham, NC 27708
[2]CSE Department, University of Nevada, Reno, NV 89557
[3]Qualcomm AI Research, 5775 Morehouse Drive, San Diego, CA 92121

Abstract

The prosperity of Internet of Things (IoT) calls for efficient ways of designing extremely compact yet accurate DNN models. Both the cell-based neural architecture search methods and the recently proposed graph based methods fall short in finding high quality IoT models due to the search flexibility, accuracy density, and node dependency limitations. In this paper, we propose a new graph-based neural architecture search methodology MSNAS for crafting highly compact yet accurate models for IoT devices. MSNAS supports flexible search space and can accumulate learned knowledge in a meta-graph to increase accuracy density. By adopting structural wiring architecture, MSNAS reduces the dependency between nodes, which allows more compact models without sacrificing accuracy. The preliminary experimental results on IoT applications demonstrate that the MSNet crafted by MSNAS outperforms MobileNetV2 and MnasNet by 3.0% in accuracy, with 20% less peak memory consumption and similar Multi-Adds.

1. Introduction

With the rising of the Internet of Things (IoT), efficiently performing Deep Neural Network (DNN) tasks in embedded systems such as microcontrollers has become ever important. However, directly deploying DNN models or even its mobile version onto IoT devices is not feasible [1] due to the extremely limited computation power, on-chip memory (100-320KB SRAM) and flash (256KB-1MB). A recent study shows that the computational cost of a neural network to be deployed in IoT devices should be less than 60M multiple-adds (MACs), which is challenging to be achieved by existing DNN models [1].

Neural Architecture Search (NAS) has been proved as a promising way for designing DNN model architecture, especially in crafting compact models [2].In general, there are two different NAS approaches to craft DNN models – cell-based approaches that stacking found cells to automate the depth discovery of neural architectures, and graph-based approaches that formulate the overall architecture as a graph and optimize the flow of information to generate better neural architectures.

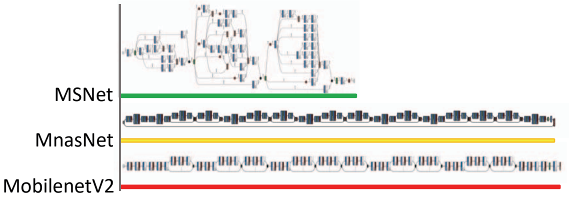

Figure 1. The structure comparison between MSNet generated by our MSNAS algorithm, MobileNetV2, and MnasNet. Our structural wiring provide better feature extraction and more adaptable to edge devices. Under the same hardware constraint, MSNet achieved 93.5% accuracy on visual wake words dataset while MnasNet and MobileNetV2 are 90% and 90.1%.

Mobile-size models found by cell-based NAS like MnasNet [2] achieves high accuracy, low computation cost, and low latency. However, when searching even smaller models or transforming large/mobile models (e.g., using width multiplier) on IoT devices [1], such approach usually achieves poor accuracy due to the redundancy of stacked cells that results in low accuracy density [3] and pre-defined cell structure that makes the search space less flexible [4]. For graph-based architecture, e.g., *random wiring* [4] relaxes the constraints of repetitive cells by exploring more diverse connectivity patterns. By allowing more flexible interconnection between nodes, random wiring achieves competitive performance for large models [4]. However, these models are difficult to be transformed to fit microcontrollers or mobile devices due to the strong dependency of each node (i.e., difficult to perform width multiplier nor pruning). In addition, models found by random wiring may induce large activation memory on SRAM [1, 4].

To overcome the above limitations of existing NAS ap-

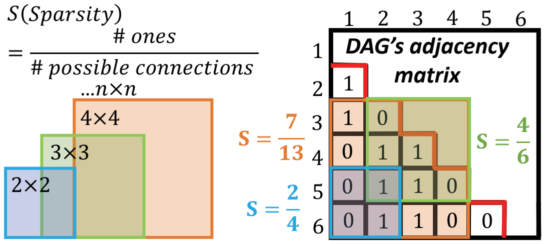

Figure 2. Visualization of constraints on both local connectivity and global connectivity. Smaller kernels represent local connectivity constraints; larger kernels represent global constraints.

proaches, we propose a new graph-based NAS methodology called MSNAS (Meta-knowledge based Structural wiring Neural Architecture Search). MSNAS introduces two key improvements over existing graph-based NAS, which makes it very capable in crafting models for IoT devices. First, by introducing meta-graph, which accumulates learned knowledge, MSNAS can explore a much more flexible search space while achieving high accuracy density search and good hardware adaptability. Second, MSNAS adopts structural wiring architecture, which reduces the dependency between nodes so that it can preserve high accuracy when transforming to small models, a significant improvement over the existing cell-based approaches and random wiring method. In addition, MSNAS also supports multi-objective neural architecture search and facilitates adding and changing objectives through changing the update rules in meta-graph.

Our preliminary experimental evaluation demonstrates that the model, MSNet, found by MSNAS outperforms state-of-the-art hardware-aware NAS works. Under the same IoT device constrained, MSNet achieved 93.5% accuracy, surpassing MobileNet-V2's 90.1%, and MNasNet's 90.0%. Compared to MobileNetV1 (400K parameters), MSNet achieved up to 4.8% performance gain with only 231K parameters. MSNet has only 200KB peak memory consumption, 20% (50KB) less than MobileNetV2 and MnasNet.

2. Meta-knowledge based Structural Wiring

We propose MSNAS (Meta-knowledge based Structural wiring Neural Architecture Search), a new structural wired neural architecture search method that enables quickly discovering IoT friendly models. MSNAS abstracts the architecture of DNNs into several directed acyclic graphs (DAGs) connected by downsampling modules. All of the DAGs are independent and each DAG is a sampled subgraph from a complete DAG. The target DNN architecture is divided into several stages represented by their respective DAGs. A down-sampling module is employed to connect DAGs in adjacent stages.

Each node of a DAG represents a node operation o and produces an output tensor x. The node operation can be any valid operations in DNN architectures, parameterized by weight parameter W. For Convolutional Neural Networks, each node can choose the operation from either 1×1 Convolution or 3×3 Depthwise Separable Convolution. Each Convolution operation uses a conv-bn-relu triplet. Each edge $e \in \mathcal{E}$ of a DAG represents the flow of tensors between its two connected nodes, and we use w to denote the probability of connection of this edge. For example, for two nodes u, v, edge $e_{u \to v} = (u, v, w_{uv}) \in \mathcal{E}$ represents the flow of the tensor from node u to node v, and the probability of the connection is given by w_{uv}. These probabilities of edge connections are used to generate optimal architectures after the graph propagation process is completed. The output tensor is computed by concatenating all of the input nodes within the last dimension and performing the corresponding node operation:

$$x_v = o_v(\{u : e_{u \to v \in \mathcal{E}}\}; W_v) \tag{1}$$

By using filter concatenation, MSNAS-family architectures are able to gather knowledge from different parts of DNNs to improve the flow of information between layers like DenseNets [5].

Search Objective. Unlike existing NAS works which target at finding the best neural architecture, MSNAS targets at finding a wide range of neural architectures which can fit a wide range of resource budgets. MSNAS aims to explore a family of highly representative architectures by learning the edge connection probabilities of each DAG (edge connection weights) using a combined search metric with consideration of both performance metrics (e.g., accuracy) and IoT metrics (e.g., peak memory usage, MAC count, number of parameters, etc.). For each candidate $\mathcal{A}$, the search metric $\mathcal{M}$ for evaluation is defined as:

$$\mathcal{M}(\mathcal{A}) = Perf(w^*(\mathcal{A}), \mathcal{A}, \mathcal{D}) - \cdot IoT(\mathcal{A}) \tag{2}$$

Where *Perf()* denotes the performance metrics of current candidate architecture given optimal weight parameters validated on the proxy dataset [1]. *IoT()* is dependent on both the resource constraints and the complexity of the graph. This term is used to penalize extreme resource consumption. $\mathcal{D}$ represents the validation dataset. $w^*(A)$ represents the optimal weight parameters which are defined as:

$$w^*(\mathcal{A}) = \arg\min_{w} \mathcal{L}_{train}(w, \mathcal{A}) \tag{3}$$

Structural wiring. We define the metric of searching a hardware-aware structural wiring architecture as follows:

$$\begin{aligned} IoT(A) =& \lambda_a \| \cdot \sum_{i}^{n} k_i \circledast adj(\mathcal{A}) \|_0 \\ &+ \lambda_b \cdot MACs + \lambda_c \cdot param \end{aligned} \tag{4}$$

[1] We randomly select 5,000 samples from the entire CIFAR-10 dataset as our proxy dataset.

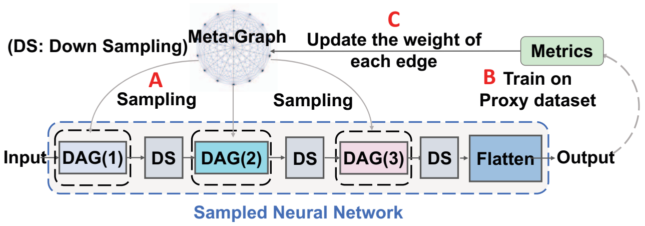

Figure 3. Overview diagram of the search process. To form a sampled DNN, we subsample multiple DAGs from the complete DAG. After several training epochs with the proxy training set, we use the search metrics (such as latency and accuracy) to update the complete DAG.

The Strong interconnection between nodes may induce activation buffer overflow in SRAM, so we regulate the local and global complexity of a sampled neural architecture by L_0-norm. To feed this information as a penalty term in the search process, we use different sizes of kernel matrices convolve with the adjacency matrices of candidate architecture, $\mathcal{A}$, to collect local and global sparsity as visualized in Figure 2. Here k_i is the kernel matrix. adj$(\cdot)$ denotes the adjacency matrix of a given DAG. MAC and $param$ represents the Multi-Adds and number of parameters in the entire neural architecture, which are measured in Millions. $\lambda_a, \lambda_b, \lambda_c$ are adjustable penalty terms. As indicated in Eq. 4, our approach aims to search for IoT-friendly neural architectures while applying the constraint of both local connectivity and global connectivity during the search process.

2.1. IoT device friendly Search Workflow

The 3-phase search workflow is visualized in Figure 3. In phase A, we sample each DAG to obtain a sampled neural network. In phase B, we train the sampled neural network on proxy dataset and get the feedback metrics according to Eq. 2. We use the validation accuracy on the proxy dataset as the performance metrics and the multi-objective penalty term in Eq. 4 as the IoT metrics. Finally, in phase C, we update the connection weights for all the DAGs according to the sampled architectures and feedback metrics in Eq. 5.

$$e^{k,t}_{w(i,j)} = \begin{cases} \frac{e^{k,t-1}_{w(i,j)}}{Z_t}\exp[\alpha(\eta' - \beta)] & \text{If } e^k_{i\to j} \in \mathcal{E}_k \\ \frac{e^{k,t-1}_{w(i,j)}}{Z_t} & \text{Otherwise} \end{cases} \tag{5}$$

Note that only the weights of edges chosen in the sampled neural networks are updated. Empirically, we apply exponential function in the update process to boost the training speed. We also use a scale factor α to adjust the rate of the update. The updated weights are normalized for a valid probability representation by Z_t.

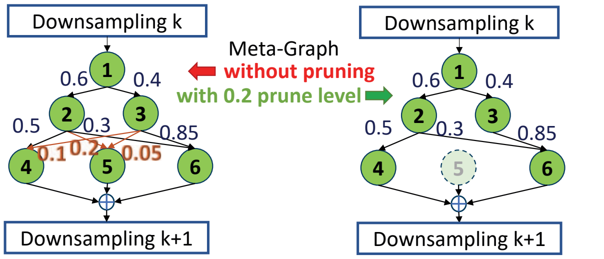

Figure 4. Visualization of structure-level pruning. The pruning achieves a trade-off between model size and model performance within a given hardware budget.

2.2. Post-searching Hardware Adaptation

Existing works such as MobileNets [6, 7] uses width multiplier to thinner the models and down-scale the channel depth to remove the redundant channels. However, such approaches could not identify redundant operations and thus not very efficient. We propose a new structure-level pruning method to explore compact architectures according to the resource constraints. Specifically, the connection weights that are below the given level are pruned while the connections weights larger than the level are kept. Figure 4 is a visualization of structure-level pruning process in a pretrained meta-graph after knowledge accumulation in MSNAS.

3. Experimental Evaluation

Experiment Setup. We configure our target meta-graph to have 3 coarse-grained stages. In each stage, we use a complete DAG with 30 nodes to generate candidate neural architectures during the search workflow. For downsampling modules connecting these DAGs from different stages, we use Max Pooling Layers with both pooling size and strides set to 2. During architecture search, hyperparameters of the searching algorithm need to be configured carefully. α is set to 0.9, β is the moving average of historic performance with initial value 0.4. λ_a is set to 0.1, λ_b, λ_c is set to 1 to penalize hardware cost with respect to the IoT metric function. Thanks to MSNAS's ability to efficiently accumulating knowledge through evaluation of candidate architectures, we trained the Meta-Graph for 1000 iterations before the post-searching hardware adaptation. Because of the small number of iterations, the training process takes only 8 GPU hours on an NVIDIA GTX 1080 GPU, which is significantly faster than existing NAS methods such as MNasNet and Randomly Wired Neural Networks. Previous works apply width multiplier to adapt to smaller devices. MnasNet can redo the hardware-aware search to adapt to different devices. On the contrary, MSNAS does not need to redo the searching to adapt to different devices. The metagraph of MSNAS is able to generate a wide range of architectures for different tasks. We evaluate the adaptation ability of MSNAS using CIFAR-10, Visual Wake Words, ImageNet.

CIFAR10. We first train the meta-graph of MSNAS using the CIFAR-10 proxy training dataset. Then we use meta-graph to generate the neural architecture and train the architecture to convergence. To show the flexibility and adaptability of MSNAS, we use different pruning threshold to generate MSNets from MSNAS with consideration of different computational budget. Table 1 reports the validation accuracy of MSNet on CIFAR-10 dataset and demonstrates MSNet's ability to fit for a wide range of computational budgets. With different structure-level pruning threshold, MSNet is able to achieve 85%-90% accuracy with 3-36 Million Multi-addons. In addition, MSNet has a much higher MAC efficiency compared to existing works, which verifis that MSNet can well utilize the limited computational power on IoT devices.

Table 1. Performance and accuracy density of different versions of MSNet trained on CIFAR-10 in the format of MSNet-k, where k represents the level of structure-level pruning.

Model	Accuracy	#MACs
MSNet-0.65	85.92%	3M
MSNet-0.50	88.74%	5M
MSNet-0.40	89.09%	12M
MSNet-0.30	90.17%	36M

Adaptation to IoT devices. Visual wake words dataset is a common vision task [1] in microcontrollers. Table 2 shows performance on visual wake words dataset for both hand-crafted models and NAS-based models. For IoT devices, both the peak memory usage limit and parameter storage limit is 250KB. We can see that MSNet has reached the state-of-art validation accuracy with fewer parameters and less peak memory consumption than existing state-of-the-art architectures.

Table 2. Performance of MSNet on IoT tasks.

IoT model - Visual wake words				
Model	Acc	#param	MACs	Peak Memory
MSNet	**93.5%**	**231K**	**42M**	**200KB**
MobileNetV1	88.7%	208K	41M	220KB
MobileNetV2	90.1%	290K	54M	252KB
MnasNet	90.0%	400K	54M	255KB

Adaptation to ImageNet. We adapt the microcontroller model to ImageNet task by changing the size of the output layer to match the number of classes. We increase the depth of ImageNet model by stacking sub-modules (i.e., DAGs between downsampling layers) to form a deeper architecture. Table 3 shows the adaptation result of MSNet on the ImageNet-1k dataset. Compared to state-of-the-art architectures, MSNet offers competitive results within the 250KB peak memory constraint.

Table 3. Performance of MSNet on ImageNet-1k.

Mobile model - ImageNet				
Model	Acc	#param	MACs	Peak Memory
MSNet	**59.1%**	**1556K**	**58M**	**250KB**
MobileNetV2	58.2% [1]	1660K	43M	250KB
MnasNet	59.7% [1]	1700K	63M	250KB

Structure Transferability. MSNAS is capable of providing a range of flexible neural architectures by changing the structure-pruning level, thus it is possible for MSNAS to generate new deep neural networks for new tasks without further training. We select a wide range of image classification tasks and use the same meta-graph of MSNAS to directly generate architectures for them. Table 4 shows that the adapted models achieved good performance while the computational cost is only around 1 Million Multi-Adds. With the extremely low computational cost, MSNet is the best fit for deployment on IoT applications.

Table 4. Adapt MSNAS to new tasks without further training.

Task	#MACs	Accuracy
MNIST	0.23M	99.24%
SVHN	1.15M	94.95%
Fashion MNIST	0.86M	93.56%

References

[1] A. Chowdhery, P. Warden, J. Shlens, A. Howard, and R. Rhodes, "Visual wake words dataset," *arXiv preprint arXiv:1906.05721*, 2019. 1, 4

[2] M. Tan, B. Chen, R. Pang, V. Vasudevan, and Q. V. Le, "Mnasnet: Platform-aware neural architecture search for mobile," in *arXiv preprint arXiv:1807.11626*, 2018. 1

[3] S. Bianco, R. Cadene, L. Celona, and P. Napoletano, "Benchmark analysis of representative deep neural network architectures," *IEEE Access*, vol. 6, pp. 64270–64277, 2018. 1

[4] S. Xie, A. Kirillov, R. Girshick, and K. He, "Exploring randomly wired neural networks for image recognition," *arXiv preprint arXiv:1904.01569*, 2019. 1

[5] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4700–4708, 2017. 2

[6] 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," in *arXiv preprint arXiv:1704.04861*, 2017. 3

[7] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4510–4520, 2018. 3