Neural-Hardware Architecture Search

Yujun Lin, Driss Hafdi, Kuan Wang, Zhijian Liu, Song Han
MIT
Cambridge, MA 02139
{yujunlin, songhan}@mit.edu

Abstract

Neural architecture and hardware architecture co-design is an effective way to enable specialization and acceleration for deep neural networks (DNNs). The design space and its exploration methodology impact efficiency and productivity. However, it is difficult to exhaust such an enormous design space by rule-based heuristics. To tackle this problem, we propose a machine learning based design and optimization methodology of a neural network accelerator. It includes the evolution strategy based hardware architecture search and one-shot hyper-net based quantized neural architecture search. Such machine learning based co-design can compose highly matched neural-hardware architectures and rival the best human-designed architectures by 1.3× speedup and 1.6× energy savings without loss of ResNet-50 accuracy under the same chip area budget.

1 Introduction

The resurgence of neural networks and the demand for energy efficiency give rise to specialized deep learning accelerators. Together with the neural network architecture, they provide a very large design space on the hardware (*e.g.* array dimensions, buffer size), compiler (*e.g.* loop tiling strategy, tiling order, fusion methods) and neural network (*e.g.* kernel size, #channels, #layers and precision). These design parameters tightly interact with each other as illustrated in Table 1. It is important to co-design the neural-hardware architecture by considering all the correlations. A perfectly matched neural architecture and hardware architecture will improve the utilization of the compute array and on-chip memories, maximizing the efficiency and performance.

However, the design space is too large to be exhausted by brute force or rule-based heuristics. Motivated by AutoML [14, 9, 10, 1, 6], we propose a machine learning based method to design and optimize AI accelerators. The design flow automatically searches for the optimal *hyper-parameters* that configure the neural network and the accelerator, and brings tightly matched neural-hardware architectures that efficiently work together. We demonstrate that our learning-based architecture search performs better than conventional grid search under the same frequency and area budget.

2 Approach

Jointly optimizing both neural and hardware architecture leads naturally a chicken or egg problem: *i.e.*, which should we configure first, the neural architecture or the hardware architecture? Given the accelerator's design cycle is much longer than neural networks, the accelerator should generalize to a suite of neural networks. Therefore, we choose to first search the hardware architecture by sampling a suite of neural architectures as a benchmark, and find the hardware architecture that achieves the best performance on the benchmarks (Section 2.1). Next, we fix the hardware architecture and design a fully specialized neural architecture that runs fast on the accelerator (Section 2.2). Figure 1 presents an overview of our methods automatically optimizing hardware and neural architectures.

33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

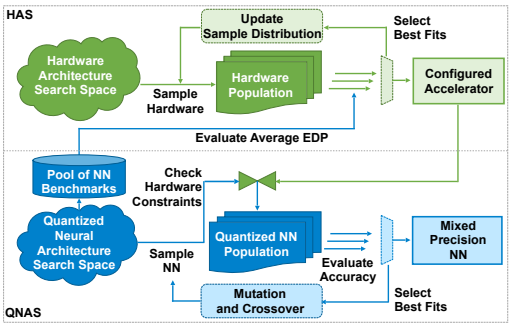

Figure 1: Neural-hardware architecture search.

Hardware Architecture	Quantized Neural Architecture Search (QNAS) Space					
Search (HAS) Space	Input Channels	Output Channels	Kernel Size	Feature Map Size	Input Bits	Output Bits
Array Height	✓				✓	
Array Width		✓				✓
IBUF Size	✓			✓	✓	
WBUF Size	✓	✓	✓		✓	✓
OBUF Size		✓		✓		✓
Fusion Styles	✓	✓			✓	✓

Table 1: The complicated correlation between neural and hardware architectures. We leverage machine learning to perform joint optimization.

2.1 Hardware Architecture Search (HAS)

We elaborate the HAS technique in three dimensions: search space, machine learning based search strategy and baseline configuration.

Search Space. The hardware architecture search space defines the types of neural network accelerator that will be explored and optimized. The base architecture that defines our search space is an in-house designed mixed-precision systolic-array-like architecture, MPA. MPA exposes five design knobs: #rows and #columns of the array size, input/weight/output buffer size. We choose these knobs as they are general to a wide range of neural network accelerators. Our method is not limited by these knobs. The array size ranges from 2×4 to 16×64 in each dimension, the input/output buffer bank size ranges from 128B to 4096B with interval of 4B, and the weight buffer bank size ranges from 128B to 256B with interval of 4B. The whole space contains 10^{10} possible configurations.

Evolutionary Search. We automate the hardware design exploration by turning it into an optimization problem. We apply evolution strategy [4] to find the best solution. In order to reduce both latency and energy, we choose the widely used metric Energy-Delay Product (EDP) as the objective. We use the geometric mean of EDP on a set of NN benchmarks normalized over the baseline as the final objective. At each evolution iteration, we first sample a set of candidates as population according to a multivariate normal distribution in $\mathbb{R}^5$ corresponding to the five parameters in our search space. These candidates are mapped to hardware configurations within given hardware area budget. After evaluating all the candidates, we update the sampling distribution based on the ranking of their EDP improvements. Specifically, we select top K solutions as the "parents" of next generation, and use their center to be the new mean of sampling distribution. We update the covariance matrix of the distribution to increase the likelihood of generating samples near parents following Equation 22 in [4]. The whole procedure is shown at the top of Figure 1.

Grid Search Baseline. In order to present a baseline design, we use grid search to find a MPA configuration with best energy-delay product on the same NN benchmarks. Since the design space is extremely large, we prune the search space following BitFusion open source code [11]. There are 250 possible configurations in total after pruning.

2.2 Quantized Neural Architecture Search (QNAS)

Researchers conventionally follow a *two-stage* pipeline to design the quantized neural network (QNN): the neural architecture is first designed without taking quantization into consideration [7, 1]; the bit width assignment for each NN layer is then searched [12]. However, this *separation* usually leads to a sub-optimal solution since the best neural architecture for floating-point might not be optimal after quantization. Another limitation is that, previous neural architecture design was guided by proxy rewards (*e.g.*, FLOPs) or latency measured on general-purpose hardware (*e.g.*, GPU, mobile CPU). As a result, the designed model is not *specialized* for the target accelerator and might not fully utilize the resources (*e.g.*, SRAM can hold 32 channels while the designed model only uses 16). To this end, we propose the quantized neural architecture search (QNAS) to perform the neural architecture and quantization policy search *at the same time* and provide specialized solution for the target hardware accelerator.

Design Space. Our quantized neural network is composed of multiple ResNet [5] blocks. We relax each block to have 10 choices with different kernel sizes (3 or 5) and numbers of channels ($0.125\times$, $0.25\times$, $0.5\times$, $1\times$ or $1.5\times$). For each conv layer within the block, the activation can be 4/6/8 bits,

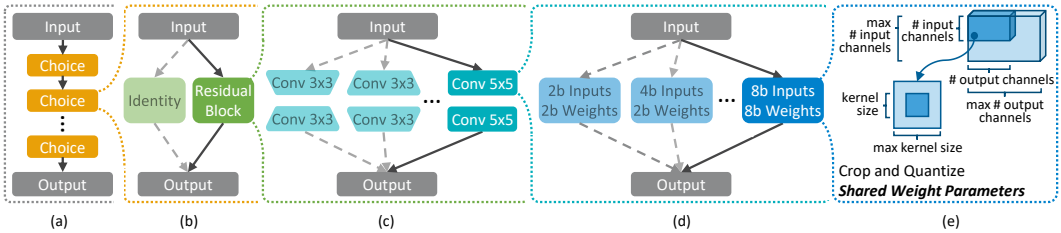

Figure 2: (a-d) hierarchically sample the neural architecture and quantization policy from the search space; (e) a floating-point weight parameter is shared among different convolution and quantization settings within the same layer.

and the weight can be 2/4/6 bits. For a model with 16 blocks (ResNet-50), the total design space has at least $10^{16} \times 9^{50}$ choices.

Evolutionary Search. We automate this exploration with a biologically-inspired evolutionary algorithm [3] (Figure 1). We first initialize the population with N randomly sampled QNNs. At each iteration, we evaluate all models in the population and select the k models with the highest accuracies (*i.e.*, the fittest individuals). The population for the next iteration is then generated with $(N/2)$ mutations and $(N/2)$ crossovers. For each mutation, we randomly pick one among the top-k candidates and alter each of its parameters (*e.g.*, kernel size, bit width) with a pre-defined probability; for each crossover, we select two from the top-k candidates and generate a new model by fusing them together randomly. Finally, the best model is obtained from the population of the last iteration. We set #iterations to 40 and the population size N to 100; therefore, there are 4000 different QNNs evaluated during search, each of which requires considerable training time (*e.g.*, 8 GPU days for DoReFa ResNet-50 [13]). It is computationally prohibitive to train each candidate from scratch.

Train One, Get N for Free. Instead of training each candidate *separately*, we train an over-parameterized HyperNet *once*. It takes longer to train a HyperNet than a candidate model but the training time is paid only once and amortized by many subsequent models it produces. Specifically, HyperNet is composed of a stack of choice blocks (in Figure 2), which encodes the entire QNN design space as every candidate corresponds to a *path* in HyperNet. The weights of individual QNN are inherited from the HyperNet (without any further training). To train HyperNet, we randomly sample one path at each iteration (equivalent to sample one candidate model) and only update parameters along this sampled path. However, HyperNet requires more iterations to reach the convergence as only a small portion of parameters are updated at each iteration. To solve this, we let all ResNet blocks within the same choice block *share* one set of floating-point parameters so that the shared parameters will always be updated no matter which path is sampled. As in Figure 2e, the shared weight is cropped to match the selected #channels and kernel size, and then quantized to selected bit width at each iteration.

Hardware Feedback. Prior work [3] adopted proxy signals such as BitOps as their objective. However, they are not directly related to the hardware cost (*i.e.*, latency and energy). Instead, we directly use the hardware cost as a hard constraint to control the population generation. However, evaluating all sampled network on hardware is still time consuming. Fortunately, the total amount of distinct convolution settings are limited (less then 10^3 in our design space). Therefore, we build a lookup table for all possible convolutions in a HyperNet on a configured MPA accelerator. As layers are executed sequentially and each layer's latency is independent, we sum each layer's cost to get the total cost.

3 Evaluation

We evaluate the performance from two aspects: 1) performance of HAS, the machine learning-based hardware architecture search; 2) performance of QNAS, quantized neural architecture search that fits the HAS designed accelerator. We develop a cycle-accurate simulator modeling the execution time and memory access behaviour; the circuit is clocked at 500MHz following [11] and synthesized at 45nm technology node with Cadence Genus and TSMC standard-cell library, which provides the chip area and dynamic/static power; the energy consumption of on-chip buffers are modeled using CACTI [8]. When evaluating, we use the same area budgets as Eyeriss and BitFusion which is 1.1 mm^2 for compute units and 5.87 mm^2 for the whole chip. To measure the performance and energy consumption of Eyeriss and BitFusion, we use their open-source simulation infrastructure [2, 11].

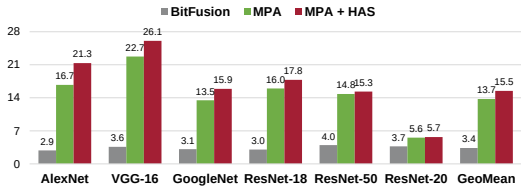

Figure 3: Speedup over Eyeriss.

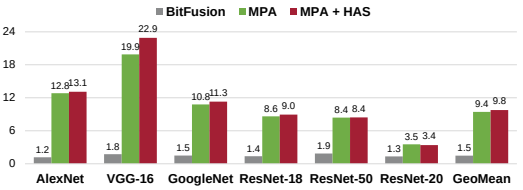

Figure 4: Energy Savings over Eyeriss.

	Grid Search	Random Search	Bayesian Optimization	HAS
# Samples	250	2048	2048	240
EDP Reduction	1.00×	1.05×	0.90×	1.16×

Table 2: Machine learning based hardware architecture search (HAS) achieves human performance optimizing the energy-delay product (EDP) with good sample efficiency (batch=16).

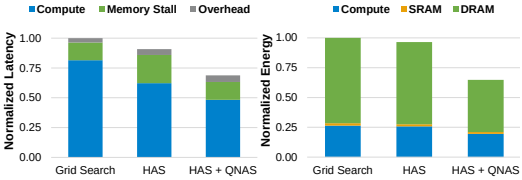

Figure 5: Normalized performance (ResNet18, batch=1).

Improvement from HAS. The red bar in Figure 3 and 4 show the performance and energy savings with HAS configuration. On average, it offers additional 1.1× speedup than the best human design (grid search) on the MPA search space with similar number of trials (Table 2). HAS configuration is perfectly in line with the correlations shown in Table 1. For MPA design, HAS decreased the input parallelism by half and doubled the output parallelism, which can improve the inputs reuse. The WBUF size is slightly increased from 128B to 144B, a multiple of the most common convolution kernel size (3×3). HAS also reduces the IBUF and OBUF size so that OBUF can exactly fit in the most common output feature map size (7×7) and IBUF can exactly fit in the corresponding input feature map size (15×15) when stride is 2. HAS configuration enables larger tiling size and thus increases the PE array utilization after data are ready. As illustrated in Figure 5, HAS spends more time on data transfer but saves more on computation, and therefore the total latency is reduced.

More Improvement from QNAS. Figure 6 shows the trade-offs between accuracy and latency in QNAS. Under the same latency constraint, QNAS finds quantized neural architecture that gives better model accuracy. On Cifar10 dataset, QNAS realizes additional 1.7× speedup and 1.7× energy savings with same accuracy compared to ResNet-20. On ImageNet dataset, QNAS provides additional 1.3× speedup and 1.7× energy savings over ResNet-18, 1.3× speedup and 1.5× energy savings over ResNet-50 without loss of accuracy. Combining with the energy breakdown in Figure 5, QNAS mostly reduces the DRAM's energy (by 25%). QNAS can find a specialized model that fit the hardware's buffer size and array size, which shows the effectiveness of neural-hardware architecture co-design.

4 Conclusion

Deploying deep neural networks on edge devices poses great challenges to both neural architecture and hardware architecture design. We propose an automated machine learning based architecture search including hardware architecture search HAS and quantized neural architecture search QNAS to find the closely matched neural-hardware architectures. We synthesis the RTL implementation

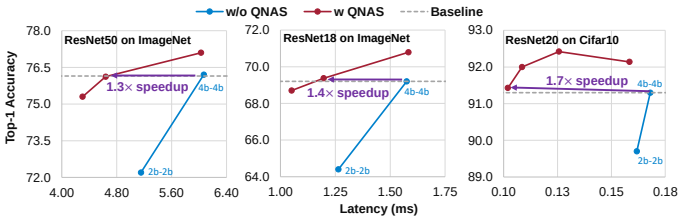

Figure 6: Accuracy/Latency benefit from QNAS (batch=1).

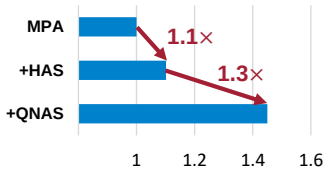

Figure 7: Speedup breakdown.

of MPA in 45nm technology node and evaluate the benefits with six real-world CNN benchmarks. Figure 7 break down the performance improvement from HAS and QNAS. On ImageNet, we offer 1.5×, 1.3× speedup and 1.7×, 1.6× energy savings over in-house desgin MPA on ResNet18 and ResNet50 while fully preserving the accuracy. Such neural-hardware architecture search finds different design choice than human and paves the way for automated algorithm and hardware co-design.

References

[1] Han Cai, Ligeng Zhu, and Song Han. ProxylessNAS: Direct neural architecture search on target task and hardware. *arXiv preprint arXiv:1812.00332*, 2018. 1, 2

[2] Mingyu Gao, Jing Pu, Xuan Yang, Mark Horowitz, and Christos Kozyrakis. Tetris: Scalable and efficient neural network acceleration with 3D memory. In *ACM SIGARCH Computer Architecture News*, volume 45, pages 751–764. ACM, 2017. 3

[3] Zichao Guo, Xiangyu Zhang, Haoyuan Mu, Wen Heng, Zechun Liu, Yichen Wei, and Jian Sun. Single path one-shot neural architecture search with uniform sampling. *arXiv preprint arXiv:1904.00420*, 2019. 3

[4] Nikolaus Hansen. The cma evolution strategy: a comparing review. In *Towards a new evolutionary computation*, pages 75–102. Springer, 2006. 2

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016. 2

[6] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. Amc: Automl for model compression and acceleration on mobile devices. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 784–800, 2018. 1

[7] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017. 2

[8] Sheng Li, Ke Chen, Jung Ho Ahn, Jay B Brockman, and Norman P Jouppi. CACTI-P: Architecture-level modeling for SRAM-based structures with advanced leakage reduction techniques. In *Proceedings of the International Conference on Computer-Aided Design*, pages 694–701. IEEE Press, 2011. 3

[9] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 19–34, 2018. 1

[10] Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. *arXiv preprint arXiv:1802.03268*, 2018. 1

[11] Hardik Sharma, Jongse Park, Naveen Suda, Liangzhen Lai, Benson Chau, Vikas Chandra, and Hadi Esmaeilzadeh. Bit Fusion: Bit-level dynamically composable architecture for accelerating deep neural networks. In *Proceedings of the 45th Annual International Symposium on Computer Architecture*, pages 764–775. IEEE Press, 2018. 2, 3

[12] Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. HAQ: Hardware-aware automated quantization with mixed precision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8612–8620, 2019. 2

[13] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160*, 2016. 3

[14] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016. 1