AutoML for Architecting Efficient and Specialized Neural Networks

Han Cai, Ji Lin, Yujun Lin, Zhijian Liu, Kuan Wang, Tianzhe Wang, Ligeng Zhu, and Song Han
Massachusetts Institute of Technology

***Abstract*—Efficient deep learning inference requires algorithm and hardware codesign to enable specialization: we usually need to change the algorithm to reduce memory footprint and improve energy efficiency. However, the extra degree of freedom from the neural architecture design makes the design space much larger: it is not only about designing the hardware architecture but also codesigning the neural architecture to fit the hardware architecture. It is difficult for human engineers to exhaust the design space by heuristics. We propose design automation techniques for architecting efficient neural networks given a target hardware platform. We investigate automatically designing specialized and fast models, auto channel pruning, and auto mixed-precision quantization. We demonstrate that such learning-based, automated design achieves superior performance and efficiency than the rule-based human design. Moreover, we shorten the design cycle by 200× than previous work, so that we can afford to design *specialized* neural network models for different hardware platforms.**

Algorithm and hardware codesign plays an important role in efficient deep learning computing. Unlike treating the neural network as a black box, there is plenty of room in the neural architecture design that can improve the inference efficiency of deep learning, given fixed target hardware. The benefit usually comes from memory saving and locality improvement. For example, model compression techniques[1] including pruning and quantization can drastically reduce the memory footprint and save energy consumption. Another

Digital Object Identifier 10.1109/MM.2019.2953153
Date of publication 12 November 2019; date of current version 14 January 2020.

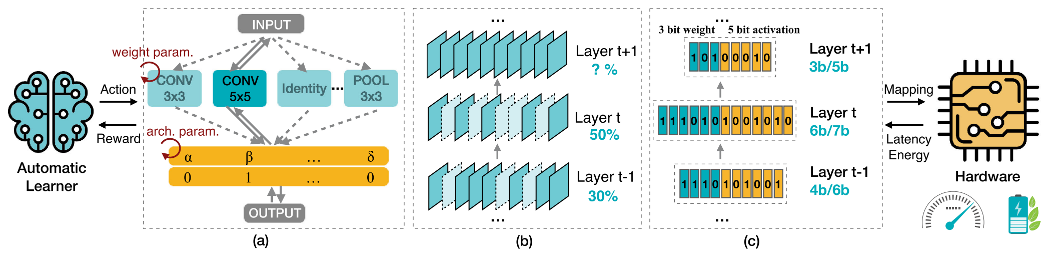

Figure 1. Design automation for (a) auto model specialization; (b) auto channel pruning; and (c) auto mixed-precision quantization.

example is small model design. MobileNet[2] has only 4.8 MB of model size, which can fit on-chip SRAM and improve the locality.

However, efficient model design and compression have a large design space. Many different neural network architectures can lead to similar accuracy, but drastically different hardware efficiency. This is difficult to exhaust by rule-based heuristics since there is a shortage of deep learning and hardware experts to hand-tune each model to make it run fast. It is demanding to systematically study how to design efficient neural architecture given a specific hardware architecture. We propose hardware-centric AutoML techniques that can automatically design neural networks that are hardware efficient for inference.[3–5] Such joint optimization is systematic and can transfer well between tasks. It requires less engineering effort while designing better neural networks at a low cost.

We explore three aspects of neural network design automation (Figure 1): auto design specialized model, auto channel pruning, and auto mixed-precision quantization. Each aspect is summarized as follows.

There is plenty of specialized hardware for neural networks, but little research has been done for specialized neural network architecture for a given hardware architecture (the *reverse* specialization). There are several advantages for a specialized model: it can fully utilize the parallelism of the target hardware [e.g., fitting the channel size with the processing element array (PE) size]. Besides, a specialized model can fully utilize the on-chip buffer and improve locality and reuse. Specialization can also match the model's computation intensity with the hardware's roofline model. However, designing a specialized neural network architecture used to be difficult. First, there are limited heuristics. Second, the computation cost used to be prohibitive: even searching a model on CIFAR-10 data set takes 10^4 GPU hours.[6,7] Third, it requires to directly take hardware feedback into the optimization process since low FLOPs do not directly translate to high hardware efficiency.[7] We cut the search cost by two orders of magnitude (actually more than that, since we directly search on ImageNet), which enables us to design specialized models on the target task and target hardware. On the mobile phone, our searched model[3] runs 1.8× faster than the best human-designed model.[8]

After designing a specialized model, compression and pruning is an effective technique to further reduce the memory footprint.[1,9] Conventional model compression techniques rely on handcrafted heuristics and *rule-based* policies that require domain experts to explore the large design space. We propose an automated design flow that leverages reinforcement learning to give the best model compression policy. This *learning-based* compression policy outperforms the conventional *rule-based* compression policy by having a higher compression ratio, better preserving the accuracy, and freeing human labor. We applied this automated, push-the-button compression pipeline to MobileNet and achieved 1.81× speedup of measured inference latency on an Android phone and 1.43× speedup on the Titan XP GPU, with only 0.1% loss of ImageNet top-1 accuracy.

The last step is automatic mixed-precision quantization. Emergent DNN hardware accelerators begin to support *flexible bitwidth* (1–8 b), which raises a great challenge to find the optimal bitwidth for each layer: it requires domain experts to explore the vast design space trading off among accuracy, latency, energy, and model size. The conventional quantization algorithm ignores the different hardware architectures and quantizes all the layers in a uniform way. We introduce the automated design flow of model quantization, and we take the hardware accelerator's feedback in the design loop. Our framework can specialize the quantization policy for different hardware architectures. It can effectively reduce the latency by 1.4–1.95$\times$ and the energy consumption by 1.9$\times$ with negligible loss of accuracy compared with the fixed bitwidth (8 b) quantization.

AUTOMATED MODEL SPECIALIZATION

In order to fully utilize the hardware resource, we propose to search a specialized convolutional neural network (CNN) architecture for the given hardware. The model is compact and runs fast. We start with a large design space [Figure 1(a)] that includes many candidate operators (e.g., 3 $\times$ 3 Conv, 5 $\times$ 5 Conv, etc.) to *learn* which is the best one by gradient descent, rather than hand-picking with rule-based heuristics. Instead of just learning the weight parameter, we jointly learn the architecture parameter [shown in red in Figure 1(a)]. The architecture parameter is the probability of choosing each operator. The search space for each block i consists of many choices:

In order to fully utilize the hardware resource, we propose to search a specialized convolutional neural network (CNN) architecture for the given hardware. The model is compact and runs fast.

- ConvOp: mobile inverted bottleneck conv[8] with various kernel sizes ($\{3 \times 3, 5 \times 5, 7 \times 7\}$ in our experiments) and expansion ratios ($\{3, 6\}$ in our experiments);
- ZeroOp: if ZeroOp is chosen at ith block, it means the block is skipped.

Therefore, in our experiments, the total number of possible architectures is

$$[\underbrace{(3 \times 2)}_{\texttt{ConvOp}} + \underbrace{1}_{\texttt{ZeroOp}}]^N = 7^N,$$

where N is the number of blocks (21 in our experiments).

Given the vast design space, it is infeasible to rely on domain experts to manually design the CNN model for each hardware platform. So we need to employ automatic architecture design techniques.

However, early reinforcement learning-based[6,7] neural architecture search (NAS) methods are very expensive to run (e.g., 10^4 GPU hours) since they need to iteratively sample an architecture, train it from scratch, and update the meta-controller. It typically requires tens of thousands of networks to be trained to find a good neural network architecture.

We adopt a different approach to improve the efficiency of model specialization.[3] We first build a super network that comprises all candidate architectures. Concretely, it has a similar structure to a CNN model in the design space except that each specific operation is replaced with a mixed operation that has n parallel operators, and we introduce an architecture parameter α_i to each operator to learn which operators are redundant and thereby can be pruned. Finally, only one operator remains within each block, and we retrain this model from scratch on the target task for deployment.

In the forward step, to save GPU memory, we allow only one candidate operator to actively reside in the GPU memory. This is achieved by hard-thresholding the probability of each candidate operator to either 0 or 1. As such, the output of a mixed operation is given as

$$x_l = \sum_i g_i o_i(x_{l-1}) \qquad (1)$$

where g_i is sampled according to the multinomial distribution derived from the architecture parameters, i.e., $\{p_i = \mathrm{softmax}(\alpha_i; \alpha) = \exp(\alpha_i)/\sum_i \exp(\alpha_i)\}$.

In the backward step, we update the weight parameters of active operators using standard gradient descent. Since the architecture parameters are not directly involved in the computational graph (1), we use the gradient w.r.t. binary gates to update the corresponding architecture parameters:

$$\frac{\partial L}{\partial \alpha_i} = \sum_{j=1} \frac{\partial L}{\partial p_j}\frac{\partial p_j}{\partial \alpha_i} \approx \sum_{j=1} \frac{\partial L}{\partial g_j}\frac{\partial p_j}{\partial \alpha_i}.$$

In order to specialize the model for hardware, we need to take the latency running on the hardware as a design reward. However, measuring the inference latency on-device has the following problems: 1) being slow due to the measurement

Model	Top-1	Top-5	GPU Latency
MobileNet-V2 [8]	72.0	91.0	6.1 ms
ResNet-34 [10]	73.3	91.4	8.0 ms
NASNet-A [7]	74.0	91.3	38.3 ms
MnasNet [11]	74.0	91.8	6.1 ms
Specialized model for GPU	**75.1**	**92.5**	**5.1 ms**

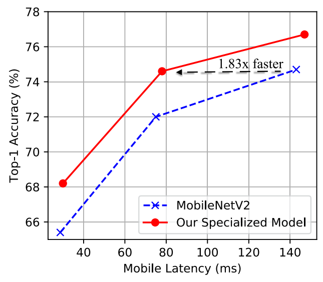

Figure 2. Left: ImageNet Accuracy (%) and GPU latency (Tesla V100). Our specialized model outperforms previous work. Right: AI automatically designed specialized model consistently outperforms human designed MobileNetV2 under various latency settings.

time; 2) high variance due to different battery condition, and thermal throttling; and 3) latency is nondifferentiable and cannot be directly optimized. To address these, we present our latency prediction model and hardware-aware loss.

To build the latency model, we precompute the latency of all possible operators in the architecture space. We query the lookup table during the searching process. The overall latency of ith block is the weighted sum of the latency of each operator.

Then, we combine the latency and training loss (e.g. cross-entropy loss) using the following formula:

$$\mathcal{L} = \mathcal{L}_{\text{CE}} \times \lambda \log \left(\frac{\mathbb{E}[\text{LAT}]}{\text{LAT}_{\text{ref}}} \right)^{\beta} \quad (2)$$

where $\mathcal{L}$ is the loss function, $\mathcal{L}_{\text{CE}}$ is the cross-entropy loss, λ and β are hyperparameters controlling the accuracy–latency tradeoff, and LAT_{ref} is the target latency. Note that our formulation not only provides a fast estimation of the searched model but also makes the search process fully differentiable.

We demonstrate the effectiveness of our model specialization on ImageNet data set with CPU (Xeon E5-2640 v4), GPU (Tesla V100), and mobile phone (Google Pixel-1). We first search for a specialized CNN model for the mobile phone [Figure 2 (b)]. Compared to MobileNet-V2 (the state-of-the-art human engineered architecture), our model improves the top-1 accuracy by 2.6% while maintaining a similar latency. Under the same level of top-1 accuracy (around 74.6%), MobileNet-V2 has 143-ms latency while ours has only 78 ms (1.83× faster). Compared with the state-of-the-art auto designed model, MnasNet,[10] our model can achieve 0.6% higher top-1 accuracy with slightly lower mobile latency. More remarkably, we reduced the search cost by 200×, from 40 000 GPU hours to only 200 GPU hours.

Figure 2(a) reports the speedup on GPU. Our method achieved superior performances compared to both human-designed and automatically searched architectures. Compared to general-purpose models, our specialized model improves the top-1 accuracy by 1.1%–3.1% while being 1.2×–7.5× faster. It is essential to learn *specialized* neural networks to cater for different hardware.

Our automated design flow designed CNN architectures that were long dismissed as being too inefficient—but in fact, they are very efficient. For instance, engineers have essentially stopped using 7 × 7 filters, because they are computationally more expensive than multiple, smaller filters (one 7 × 7 layer has the same receptive field than three 3 × 3 layers, but bears 49 weights rather than 27). However, our AI-designed model found that using a 7 × 7 filter is very efficient on GPUs. That is because GPUs have high parallelization, and invoking a large kernel call is more efficient than invoking multiple small kernel calls. This design choice goes against previous human thinking. The larger the search space, the more unknown things you can find. You do not know if something will be better than the past human experience. Let the automated design tool figure it out (kicking neural network design automation into high gear).[*]

AUTOMATED CHANNEL PRUNING

Pruning is widely used in model compression and acceleration. It is very important to find the optimal sparsity for each layer during pruning. Pruning too much will hurt accuracy; pruning not enough will not achieve high compression ratio. This used to be manually determined in previous studies.[1] Our goal is to automatically

[*]Available at: http://news.mit.edu/2019/convolutional-neural-network-automation-0321

Table 1. AMC speeds up MobileNet. On Google Pixel-1 CPU, AMC achieves 1.95× measured speedup with batch size one, while saving the memory by 34%. On NVIDIA Titan XP GPU, AMC achieves 1.53× speedup with batch size of 50.

	Million	Top-1	Top-5	GPU		Android		
	MAC	Acc.	Acc.	Latency	Speed	Latency	Speed	Memory
100% MobileNet	569	70.6%	89.5%	0.46 ms	2191 fps	123.3 ms	8.1 fps	20.1 MB
75% MobileNet	325	68.4%	88.2%	0.34 ms	2944 fps	72.3 ms	13.8 fps	14.8 MB
AMC (50% FLOPs)	285	70.5%	89.3%	0.32 ms	**3127 fps** (**1.43**×)	68.3 ms	**14.6 fps** (**1.81**×)	14.3 MB
AMC (50% Latency)	272	70.2%	89.2%	0.30 ms	**3350 fps** (**1.53**×)	63.3 ms	**16.0 fps** (**1.95**×)	13.2 MB

find out the effective *sparsity* for each layer. We train a reinforcement learning agent to predict the best sparsity for a given hardware resource.[4] We evaluate the accuracy and FLOPs after pruning. Then, we update the agent by encouraging smaller, faster, and more accurate models.

Our automatic model compression (AMC) leverages reinforcement learning to efficiently search the pruning ratio [Figure 1(b)]. The reinforcement learning agent receives an embedding state s_t of layer L_t from the environment and then outputs a sparsity ratio as action a_t. The layer is compressed with a_t (rounded to the nearest feasible fraction). We calculate the average magnitude of weight tensor for each input channel and remove the input channels with least magnitude. Then, the agent moves to the next layer L_{t+1}, and receives state s_{t+1}. After finishing the final layer L_T, the reward accuracy is evaluated on the validation set and returned to the agent.

With our framework, we are able to push the expert-tuned limit of fine-grained model pruning. For ResNet-50 on ImageNet, we can push the compression ratio from 3.4× to 5× without loss of accuracy. With further investigation, we find that AMC automatically learns to prune 3 × 3 convolution kernels harder than 1 × 1 kernels, which is similar to human heuristics since the latter is less redundant.

We applied AMC to MobileNet[2] and observed significant speedup both on the GPU and the mobile phone (Table 1). Since MobiletNet is already very compact, it is convincing to compress this model. For a 0.5× FLOPs setting, we achieve 1.81× speedup on a Google Pixel 1 phone. For a 0.5× FLOPs setting, we accurately achieve 1.95× speedup, which is very close to actual 2× target, showing that AMC can directly optimize inference time and achieve an accurate speedup ratio. On GPUs, we also achieve up to 1.5× speedup, which is less than mobile phone but still significant on an already very compact model. The less speedup is because a GPU has a higher degree of parallelism than a mobile phone.

AUTOMATED MIXED-PRECISION QUANTIZATION

Conventional quantization methods quantize each layer of the model to the same precision. Mixed-precision quantization is more flexible but suffer from a huge design space that is difficult to explore. Meanwhile, the quantization solution optimized on one hardware might not be optimal on the other, which raises the demand for *specialized* policies for different hardware architectures and further increase the design space. Assuming that the bitwidth is between 1 and 8 for both weights and activations, then each layer has 8^2 choices. If we have M different neural network models, each with N layers, on H different hardware platforms, there are in total $O(H \times M \times 8^{2N})$ possible solutions. Rather than using rule-based heuristics, we propose an automated design flow to quantize different layers with mixed precision. Our hardware-aware automatic quantization (HAQ)[5] models the quantization task as a reinforcement learning problem. We use the actor-critic model to give the quantization policy (#bits per layer) [Figure 1(c)], and use FPGAs to provide energy and latency cost because FPGAs well support mixed precision. The goal is not only high accuracy but also low energy and low latency.

An intuitive reward can be FLOPs or the model size. However, these proxy signals are indirect. They do not translate to latency or energy

Table 2. Latency-constrained quantization on the edge and cloud accelerator on ImageNet.

		Edge accelerator		Cloud accelerator	
	Bitwidths	Top-1	Latency	Top-1	Latency
PACT	4 b	62.44	45.45 ms	61.39	52.15 ms
Ours	*flexible*	**67.40**	45.51 ms	**66.99**	52.12 ms
PACT	5 b	67.00	57.75 ms	68.84	66.94 ms
Ours	*flexible*	**70.58**	57.70 ms	**70.90**	66.92 ms
PACT	6 b	70.46	70.67 ms	71.25	82.49 ms
Ours	*flexible*	**71.20**	70.35 ms	**71.89**	82.34 ms
Original	8 b	70.82	96.20 ms	71.81	115.84 ms

improvement. Cache locality, number of kernel calls, memory bandwidth all matter. Instead, we use direct latency and energy feedback from the hardware simulator. Such feedback enables our RL agent to learn the hardware characteristics for different layers, e.g., vanilla convolution has more data reuse and locality, while depthwise convolution has less reuse and worse locality, which makes it memory bounded.

In real-world applications, we have limited resource budgets (i.e., latency, energy, and model size). We would like to find the quantization policy with the best performance given the resource constraint. We encourage our agent to meet the computation budget by limiting the action space. After our RL agent gives actions $\{a_k\}$ to all layers, we measure the amount of resources that will be used by the quantized model. The feedback is directly obtained from the hardware simulator. If the current policy exceeds our resource budget (on latency, energy or model size), we will sequentially decrease the bitwidth of each layer until the constraint is finally satisfied.

We applied HAQ to three different hardware architectures to show the importance of specialized quantization policy. Inferencing on edge devices and cloud severs can be quite different, since 1) the batch size on the cloud servers are larger and 2) the edge devices are usually limited- to low-computation resources and memory bandwidth. We use embedded FPGA Xilinx Zynq-7020 as our edge device, and server FPGA Xilinx VU9P as our cloud device to compare the specialized quantization policies.

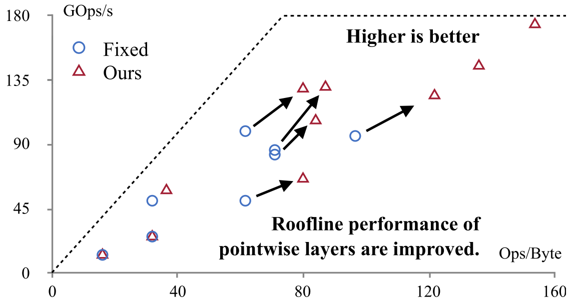

Figure 3. HAQ improves the roofline performance of pointwise layers in MobileNet-V1.

Compared to fixed 8-b quantization (PACT[11]), our automated quantization consistently achieved better accuracy under the same latency (see Table 2). With similar accuracy, HAQ can reduce the latency by 1.4×–1.95× compared with the baseline.

Our agent gave specialized quantization policy for edge and cloud accelerators. The policy is quite different on different hardware. For the activations, the depthwise convolution layers are assigned much less bitwidth than the pointwise layers on the edge device; however, on the cloud device, the bitwidths of these two types of layers are similar to each other. For weights, the bitwidths of these types of layers are nearly the same on the edge; however, on the cloud, the depthwise convolution layers are assigned much more bitwidth than the pointwise convolution layers.

We interpret the quantization policy's difference between edge and cloud by the roofline model. Operation intensity is defined as operations (MACs in neural networks) per DRAM byte accessed. A lower operation intensity indicates that the model suffers more from the memory access. On the edge accelerator, which has much less memory bandwidth, our RL agent allocates *fewer* activation bits to the depthwise convolutions since the activations dominate the memory access. On the cloud accelerator, which has more memory bandwidth, our agent allocates *more* bits to the depthwise convolutions and allocates *fewer* bits to the pointwise convolutions to prevent it from being computation bounded. Figure 3 shows the roofline model before and after HAQ. HAQ gives more reasonable policy to allocate the bits for each layer and pushes all the points to the upper-right corner that is more efficient.

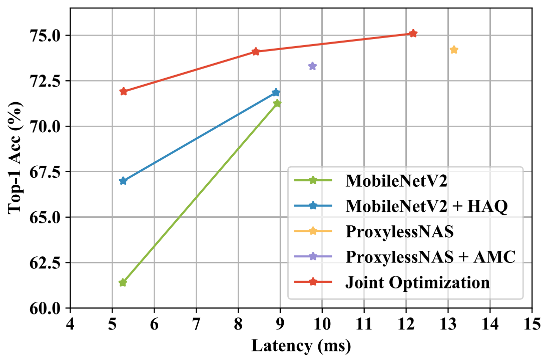

Figure 4. Combining all of the techniques together further improves the accuracy–latency tradeoff by a significant margin on BitFusion.

Finally, we integrate all of these techniques together, including auto model specialization, auto channel pruning, and auto mixed-precision quantization. Figure 4 shows the results on the BitFusion accelerator, where we can observe significant improvements over the baselines including ProxylessNAS (with 8-b quantization), ProxylessNAS + AMC (with 8-b quantization), MobileNetV2 (with 4-b/6-b quantization), and MobileNetV2 + HAQ (mixed-precision quantization with different target latency).

CONCLUSION

In this article, we presented design automation techniques for efficient deep learning computing. There is plenty of room at the algorithm level to improve the hardware efficiency, but the large design space makes it difficult to be exhausted by human, and the conventional AutoML techniques are not hardware efficient for both search and inference. We covered three aspects of design automation: specialized model design, compression and pruning, mixed-precision quantization. Such learning-based design automation outperformed rule-based heuristics. Our framework reveals that the optimal design policies on different hardware architectures are drastically different; therefore, specialization is important. We interpreted those design policies and believe the insights will inspire the future software and hardware codesign for efficient deep learning computing.

Our framework reveals that the optimal design policies on different hardware architectures are drastically different; therefore, specialization is important.

REFERENCES

1. S. Han, H. Mao, and W. J. Dally, “Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding,” in *Proc. Int. Conf. Learn. Representations*, 2016.
2. A. G. Howard *et al.*, “Mobilenets: Efficient convolutional neural networks for mobile vision applications,” 2017, *arXiv:1704.04861*. [Online]. Available: https://arxiv.org/abs/1704.04861
3. H. Cai, L. Zhu, and S. Han, “ProxylessNAS: Direct neural architecture search on target task and hardware,” in *Proc. Int. Conf. Learn. Representations*, 2019.
4. Y. He, J. Lin, Z. Liu, H. Wang, L.-J. Li, and S. Han, “AMC: AutoML for model compression and acceleration on mobile devices,” in *Proc. Eur. Conf. Comput. Vis.*, 2018, pp. 784–800.
5. K. Wang, Z. Liu, Y. Lin, J. Lin, and S. Han, “HAQ: Hardware-aware automated quantization with mixed precision,” in *Proc. Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 8612–8620.
6. B. Zoph and Q. V. Le, “Neural architecture search with reinforcement learning,” in *Proc. Int. Conf. Learn. Representations*, 2017.
7. B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, “Learning transferable architectures for scalable image recognition,” in *Proc. Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 8697–8710.
8. M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, “Mobilenetv2: Inverted residuals and linear bottlenecks,” in *Proc. Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 4510–4520.
9. J. Yu, A. Lukefahr, D. Palframan, G. Dasika, R. Das, and S. Mahlke, “Scalpel: Customizing DNN pruning to the underlying hardware parallelism,” *ACM SIGARCH Comput. Archit. News*, vol. 45, no. 2, pp. 548–560, 2017.
10. M. Tan, B. Chen, R. Pang, V. Vasudevan, and Q. V. Le, “Mnasnet: Platform-aware neural architecture search for mobile,” in *Proc. Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 2820–2828.
11. J. Choi, Z. Wang, S. Venkataramani, P. I.-J. Chuang, V. Srinivasan, and K. Gopalakrishnan, “PACT: Parameterized Clipping Activation for Quantized Neural Networks,” 2018, *arXiv:1805.06085*. [Online]. Available: https://arxiv.org/abs/1805.06085

Han Cai is currently working toward the Ph.D. degree with the Electrical Engineering and Computer Sciences Department, Massachusetts Institute of Technology, Cambridge, MA, USA, under the supervision of Prof. Song Han. He received the M.Eng. degree in computer science from Shanghai Jiao Tong University, Shanghai, China. His research mainly focuses on efficient deep learning and AutoML. Contact him at hancai@mit.edu.

Ji Lin is currently working toward the Ph.D. degree with the Electrical Engineering and Computer Sciences Department, Massachusetts Institute of Technology, Cambridge, MA, USA, under the supervision of Prof. Song Han. He received the B.Eng. degree in electronic engineering from Tsinghua University, Beijing, China. His research mainly focuses on efficient deep learning and its applications. Contact him at jilin@mit.edu.

Yujun Lin is currently working toward the Ph.D. degree with the Electrical Engineering and Computer Sciences Department, Massachusetts Institute of Technology, Cambridge, MA, USA, under the supervision of Prof. Song Han. He received the B.Eng. degree in electronic engineering from Tsinghua University, Beijing, China. His research mainly focuses on efficient deep learning acceleration and machine learning- assisted hardware optimization. Contact him at yujunlin@mit.edu.

Zhijian Liu is currently working toward the Ph.D. degree with the Electrical Engineering and Computer Sciences Department, Massachusetts Institute of Technology, Cambridge, MA, USA, under the supervision of Prof. Song Han. He received the B.Eng. degree in computer science from Shanghai Jiao Tong University, Shanghai, China. His research mainly focuses on efficient and hardware-friendly machine learning and its applications in vision and language. Contact him at zhijian@mit.edu.

Kuan Wang is currently a Research Assistant with HAN Lab, Massachusetts Institute of Technology, Cambridge, MA, USA, under the supervision of Prof. Song Han. He is an undergraduate student with Tsinghua University, Shanghai, China. His research interests focuses on the intersection of computer vision, deep learning, and efficient hardware architecture. He is a student member of the IEEE. Contact him at kuanwang@mit.edu.

Tianzhe Wang is currently a Research Assistant with HAN Lab, Massachusetts Institute of Technology, Cambridge, MA, USA, under the supervision of Prof. Song Han. His research focuses on efficient and automated deep learning and their applications in vision, speech, and robotics. He is an undergraduate student of ACM Class in Zhiyuan College (Honored Program), Shanghai Jiao Tong University, Shanghai, China. Contact him at usedtobe@mit.edu.

Ligeng Zhu is currently a Research Assistant with HAN Lab, Massachusetts Institute of Technology, Cambridge, MA, USA, under the supervision of Prof. Song Han. His research focuses on efficient machine learning and computer vision. He received the B.Sc. and B.Eng. degrees in computer science from Simon Fraser University, Burnaby, BC, Canada, and Zhejiang University, Hangzhou, China, respectively. Contact him at ligeng@mit.edu.

Song Han is currently an Assistant Professor with the Electrical Engineering and Computer Sciences Department, Massachusetts Institute of Technology, Cambridge, MA, USA. His research focuses on efficient deep learning computing at the intersection between machine learning and computer architecture. He proposed "Deep Compression" and the efficient hardware implementation "EIE Accelerator" that impacted the industry. He received the B.S. degree in electrical engineering from Tsinghua University, Shanghai, China, and the Ph.D. degree in electrical engineering from Stanford University, Stanford, CA, USA. His work received the best paper award in ICLR'16 and FPGA'17. He is listed by *MIT Technology Review's* 35 Innovators Under 35. Contact him at songhan@mit.edu.