HAQ: Hardware-Aware Automated Quantization With Mixed Precision

Citation: Wang, Kuan et al. "HAQ: Hardware-Aware Automated Quantization With Mixed Precision." Paper in the Proceedings of the 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition, Long Beach CA, 16-20 June 2019, IEEE © 2019 The Author(s)

As Published: 10.1109/CVPR.2019.00881

Publisher: Institute of Electrical and Electronics Engineers (IEEE)

Persistent URL: https://hdl.handle.net/1721.1/129522

Version: Original manuscript: author's manuscript prior to formal peer review

HAQ: Hardware-Aware Automated Quantization with Mixed Precision

Kuan Wang*, Zhijian Liu*, Yujun Lin*, Ji Lin, and Song Han
{kuanwang, zhijian, yujunlin, jilin, songhan}@mit.edu
Massachusetts Institute of Technology

Abstract

Model quantization is a widely used technique to compress and accelerate deep neural network (DNN) inference. Emergent DNN hardware accelerators begin to support mixed precision *(1-8 bits) to further improve the computation efficiency, 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, which is both time-consuming and sub-optimal. There are plenty of specialized hardware for neural networks, but little research has been done for specialized neural network optimization for a particular hardware architecture. Conventional quantization algorithm ignores the different hardware architectures and quantizes all the layers in a uniform way. In this paper, we introduce the* ***H****ardware-Aware* ***A****utomated* ***Q****uantization (HAQ) framework which leverages the reinforcement learning to automatically determine the quantization policy, and we take the hardware accelerator's feedback in the design loop. Rather than relying on proxy signals such as FLOPs and model size, we employ a hardware simulator to generate direct feedback signals (latency and energy) to the RL agent. Compared with conventional methods, our framework is fully automated and can specialize the quantization policy for different neural network architectures and hardware architectures. Our framework effectively reduced 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 bits) quantization. Our framework reveals that the optimal policies on different hardware architectures (*i.e.*, edge and cloud architectures) under different resource constraints (*i.e.*, latency, energy and model size) are drastically different. We interpreted the implication of different quantization policies, which offer insights for both neural network architecture design and hardware architecture design.*

∗ indicates equal contributions.

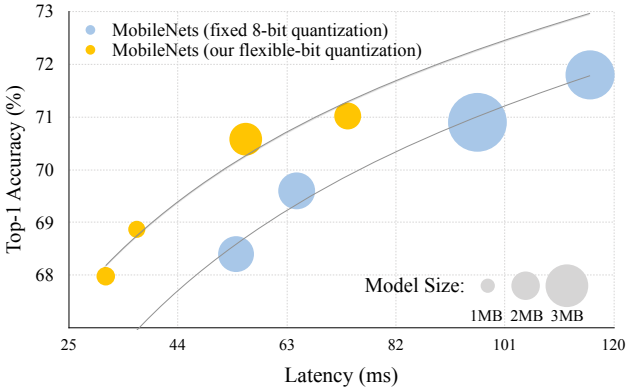

Figure 1: We need **mixed precision** for different layers. We quantize MobileNets [12] to different number of bits (both weights and activations), and it lies on a better pareto curve (yellow) than fixed bit quantization (blue). The reason is that different layers have different redundancy and have different arithmetic intensity (OPs/byte) on the hardware, which advocates for using mixed precision for different layers.

1. Introduction

In many real-time machine learning applications (such as robotics, autonomous driving, and mobile VR/AR), deep neural networks is strictly constrained by the latency, energy, and model size. In order to improve the hardware efficiency, many researchers have proposed to quantize the weights and activations to low precision [8, 18, 34].

Conventional quantization methods use the same number of bits for all layers [2, 14], but as different layers have different redundancy and behave differently on the hardware (computation bounded or memory bounded), it is necessary to use *mixed precision* for different layers (as shown in Figure 1). This flexibility was originally not supported by chip vendors until recently the hardware manufacturers started to implement this feature: Apple released the A12 Bionic chip that supports mixed precision for the neural network inference [6]; NVIDIA recently introduced the Turing GPU architecture that supports 1-bit, 4-bit, 8-bit and 16-bit arithmetic operations [21]; Imagination launched a flexible neural network IP that supports per-layer bitwidth adjustment for

	Inference latency on		
	HW1	**HW2**	**HW3**
Best Q. policy for **HW1**	**16.29** ms	85.24 ms	117.44 ms
Best Q. policy for **HW2**	19.95 ms	**64.29** ms	108.64 ms
Best Q. policy for **HW3**	19.94 ms	66.15 ms	**99.68** ms

Table 1: Inference latency of MobileNet-V1 [12] on three hardware architectures under different quantization policies. The quantization policy that is optimized for one hardware is not optimal for the other. This suggests we need a **specialized** quantization solution for different hardware architectures. (HW1: BitFusion [25], HW2: BISMO [26] edge accelerator, HW3: BISMO cloud accelerator, batch = 16).

both weights and activations [13]. Besides industry, recently academia also works on the bit-level flexible hardware design: BISMO [26] proposed the bit-serial multiplier to support multiplications of 1 to 8 bits; BitFusion [25] supports multiplications of 2, 4, 8 and 16 bits in a spatial manner.

However, a very missing part is how to **determine the bitwidth of both weights and activations for each layer on different hardware accelerators**. This is a vast design space: with M different neural network models, each with N layers, on H different hardware platforms, there are in total $\mathcal{O}(H \times M \times 8^{2N})$* possible solutions. For a widely used ResNet-50 [9] model, the size of the search space is about 8^{100}, which is even larger than the number of particles in the universe. Conventional methods require domain experts (with knowledge of both machine learning and hardware architecture) to explore the huge design space smartly with rule-based heuristics, such as: we should retain more bits in the first layer which extracts low level features and in the last layer which computes the final outputs; also, we should use more bits in the convolution layers than in the fully-connected layers because empirically, the convolution layers are more sensitive. As the neural network becomes deeper, the search space increases exponentially, which makes it infeasible to rely on hand-crafted strategies. Therefore, these *rule-based* quantization policies are usually sub-optimal, and they cannot generalize from one model to another. In this paper, we would like to *automate* this exploration process by a *learning-based* framework.

Another challenge is how to optimize the latency and the energy consumption of a given model on the hardware. A widely adopted approach is to rely on some proxy signals (*e.g.*, FLOPs, number of memory references) [12, 24]. However, as different hardware behaves very differently, the performance of a model on the hardware cannot always be accurately reflected by these proxy signals. Therefore, it is important to directly *involve the hardware architecture's performance feedback into the design loop*. Also, as demonstrated in Table 1, 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.

To this end, we propose the **H**ardware-Aware **A**utomated **Q**uantization (HAQ) framework that leverages reinforcement learning to automatically predict the quantization policy given the hardware's feedback. The RL agent decides the bitwidth of a given neural network in a layer-wise manner. For each layer, the agent receives the layer configuration and statistics as observation, and it then outputs the action which is the bitwidth of weights and activations. We then leverage the hardware accelerator as the environment to obtain the *direct feedback from hardware* to guide the RL agent to satisfy the resource constraints. After all layers are quantized, we finetune the quantized model for one more epoch, and feed the validation accuracy after short-term retraining as the reward signal to our RL agent. During the exploration, we leverage the deep deterministic policy gradient (DDPG) [17] to supervise our RL agent. We also studied the quantization policy on multiple hardware architectures: both cloud and edge neural network accelerators, with spatial or temporal multi-precision design.

The contribution of this paper has four aspects:

1. **Automation**: We propose an automated framework for quantization, which does not require domain experts and rule-based heuristics. It frees the human labor from exploring the vast search space of choosing bitwidths.

2. **Hardware-Aware**: Our framework involves the hardware architecture into the loop so that it can directly reduce the latency, energy and storage on the target hardware instead of relying on proxy signals.

3. **Specialization**: For different hardware architectures, our framework can offer a specialized quantization policy that's exactly tailored for the target hardware architecture to optimize latency and energy.

4. **Design Insights**: We interpreted the different quantization polices learned for different hardware architectures. Taking both computation and memory access into account, the interpretation offers insights on both neural network architecture and hardware architecture design.

2. Related Work

Quantization. There have been extensive explorations on compressing and accelerating deep neural networks using quantization. Han *et al*. [8] quantized the network weights to reduce the model size by rule-based strategies: *e.g.*, they used human heuristics to determine the bitwidths for convolution and fully-connected layers. Courbariaux *et al*. [4]

*Assuming the bitwidth is 1 to 8 for both weights and activations.

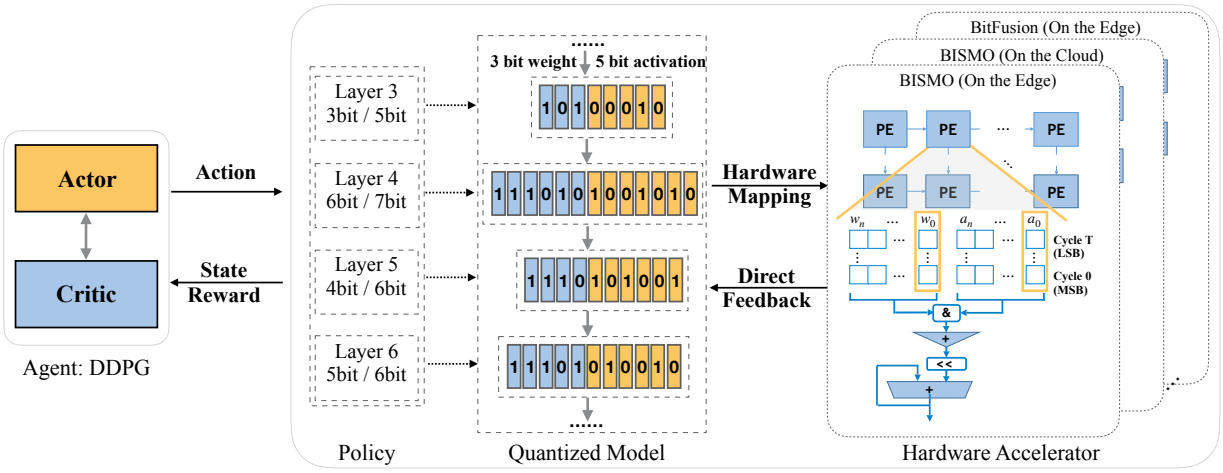

Figure 2: An overview of our **H**ardware-Aware **A**utomated **Q**uantization (HAQ) framework. We leverage the reinforcement learning to automatically search over the huge quantization design space with hardware in the loop. The agent propose an optimal bitwidth allocation policy given the amount of computation resources (*i.e.*, latency, power, and model size). Our RL agent integrates the hardware accelerator into the exploration loop so that it can obtain the direct feedback from the hardware, instead of relying on indirect proxy signals.

binarized the network weights into $\{-1, +1\}$; Rastegari *et al*. [23] and Zhou *et al*. [32] binarized each convolution filter into $\{-w, +w\}$; Zhu *et al*. [34] mapped the network weights into $\{-w_{\mathrm{N}}, 0, +w_{\mathrm{P}}\}$ using two bits; Zhou *et al*. [33] used one bit for network weights and two bits for activations; Jacob *et al*. [14] made use of 8-bit integers for both weights and activations. We refer the reader to the survey paper by Krishnamoorthi *et al*. [16] for a more detailed overview. These conventional quantization methods either simply assign the same number of bits to all layers or require domain experts to determine the bitwidths for different layers, while our framework automates this design process, and our *learning-based* policy outperforms *rule-based* strategies.

AutoML. Many researchers aimed to improve the performance of deep neural networks by searching the network architectures: Zoph *et al*. [35] proposed the Neural Architecture Search (NAS) to explore and design the transformable network building blocks, and their network architecture outperforms several human designed networks; Liu *et al*. [19] introduced the Progressive NAS to accelerate the architecture search by 5× using sequential model-based optimization; Pham *et al*. [22] introduced the Efficient NAS to speed up the exploration by 1000× using parameter sharing; Cai *et al*. [1] introduced the path-level network transformation to effectively search the tree-structured architecture space. Motivated by these AutoML frameworks, He *et al*. [10] leveraged the reinforcement learning to automatically prune the convolution channels. Our framework further explores the automated quantization for network weights and activations, and it takes the hardware architectures into consideration.

Efficient Models. To facilitate the efficient deployment, researchers designed hardware-friendly approaches to slim neural network models. For instance, the coarse-grained channel pruning methods [11, 20] prune away the entire channel of convolution kernels to achieve speedup. Recently, researchers have explicitly optimized for various aspects of hardware properties, including the inference latency and energy: Yang *et al*. [30] proposed the energy-aware pruning to directly optimize the energy consumption of neural networks; Yang *et al*. [31] reduced the inference time of neural networks on the mobile devices through a lookup table. Nevertheless, these methods are still rule-based and mostly focus on pruning. Our framework automates the quantization process by taking hardware-specific metric as direct rewards using a learning based method.

3. Approach

We model the quantization task as a reinforcement learning problem (Figure 2). We use the actor-critic model with DDPG agent to give the action: bits for each layer. We collect hardware counters as constraints, together with accuracy as rewards to search the optimal quantization policy. We have three hardware environments that covers edge and cloud, spatial and temporal architectures for mixed-precision accelerator. Below describes the details of the RL formulation.

3.1. Observation (State Space)

Our agent processes the neural network in a layer-wise manner. For each layer, our agent takes two steps: one for weights, and one for activations. In this paper, we introduce

a ten-dimensional feature vector O_k as our observation:

If the k^{th} layer is a convolution layer, the state O_k is

$$O_k = (k, c_{\text{in}}, c_{\text{out}}, s_{\text{kernel}}, s_{\text{stride}}, s_{\text{feat}}, n_{\text{params}}, i_{\text{dw}}, i_{\text{w/a}}, a_{k-1}), \quad (1)$$

where k is the layer index, c_{in} is #input channels, c_{out} is #output channels, s_{kernel} is kernel size, s_{stride} is the stride, s_{feat} is the input feature map size, n_{params} is #parameters, i_{dw} is a binary indicator for depthwise convolution, $i_{\text{w/a}}$ is a binary indicator for weight/activation, and a_{k-1} is the action from the last time step.

If the k^{th} layer is a fully-connected layer, the state O_k is

$$O_k = (k, h_{\text{in}}, h_{\text{out}}, 1, 0, s_{\text{feat}}, n_{\text{params}}, 0, i_{\text{w/a}}, a_{k-1}), \quad (2)$$

where k is the layer index, h_{in} is #input hidden units, h_{out} is #output hidden units, s_{feat} is the size of input feature vector, n_{params} is #parameters, $i_{\text{w/a}}$ is a binary indicator for weight/activation, and a_{k-1} is the action from the last step.

For each dimension in the observation vector O_k, we normalize it into $[0, 1]$ to make them in the same scale.

3.2. Action Space

We use a *continuous* action space to determine the bitwidth. The reason that we do not use a *discrete* action space is because it loses the relative order: *e.g.*, 2-bit quantization is more aggressive than 4-bit and even more than 8-bit. At the k^{th} time step, we take the continuous action a_k (which is in the range of $[0, 1]$), and round it into the discrete bitwidth value b_k:

$$b_k = \text{round}(b_{\text{min}} - 0.5 + a_k \times (b_{\text{max}} - b_{\text{min}} + 1)), \quad (3)$$

where b_{min} and b_{max} denote the min and max bitwidth (in our experiments, we set b_{min} to 2 and b_{max} to 8).

Resource Constraints. In real-world applications, we have limited computation budgets (*i.e.*, latency, energy, and model size). We would like to find the quantization policy with the best performance given the 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 accelerator, which we will discuss in Section 3.3. 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.

3.3. Direct Feedback from Hardware Accelerators

An intuitive feedback to our RL agent can be FLOPs or the model size. However, as these proxy signals are indirect, they are not equal to the performance (*i.e.*, latency, energy consumption) on the hardware. Cache locality, number of kernel calls, memory bandwidth all matters. Proxy feedback can not model these hardware functionality to find the specialized strategies (see Table 1).

Instead, we use direct latency and energy feedback from the hardware accelerator as resource constraints, which enables our RL agent to determine the bitwidth allocation policy from the subtle differences between different layers: *e.g.*, vanilla convolution has more data reuse and better locality, while depthwise convolution [3] has less reuse and worse locality, which makes it memory bounded. Such difference impacts the optimal quantization policy.

3.4. Quantization

We linearly quantize the weights and activations of each layer using the action a_k given by our agent, as linearly quantized model only needs fixed point arithmetic unit which is more efficient to implement on the hardware.

Specifically, for each weight value w in the k^{th} layer, we first truncate it into the range of $[-c, c]$, and we then quantize it linearly into a_k bits:

$$\text{quantize}(w, a_k, c) = \text{round}(\text{clamp}(w, c)/s) \times s, \quad (4)$$

where $\text{clamp}(\cdot, x)$ is to truncate the values into $[-x, x]$, and the scaling factor s is defined as $s = c/(2^{a_k - 1} - 1)$. In this paper, we choose the value of c by finding the optimal value x that minimizes the KL-divergence between the original weight distribution W_k and the quantized weight distribution $\text{quantize}(W_k, a_k, x)$:

$$c = \arg\min_x \mathcal{D}_{\text{KL}}(W_k \mid\mid \text{quantize}(W_k, a_k, x)), \quad (5)$$

where $\mathcal{D}_{\text{KL}}(\cdot \mid\mid \cdot)$ is the KL-divergence that characterizes the distance between two distributions. As for activations, we quantize the values similarly except that we truncate them into the range of $[0, c]$, not $[-c, c]$ since the activation values (which are the outputs of the ReLU layers) are non-negative.

3.5. Reward Function

After quantization, we retrain the quantized model for one more epoch to recover the performance. As we have already imposed the resource constraints (latency, energy) by limiting the action space (Section 3.2), we define our reward function $\mathcal{R}$ to be only related to the accuracy:

$$\mathcal{R} = \lambda \times (\text{acc}_{\text{quant}} - \text{acc}_{\text{origin}}), \quad (6)$$

where $\text{acc}_{\text{origin}}$ is the top-1 classification accuracy of the full-precision model on the training set, $\text{acc}_{\text{quant}}$ is the accuracy of the quantized model after finetuning, and λ is a scaling factor which is set to 0.1 in our experiments.

3.6. Agent

For the RL agent, we leverage the deep deterministic policy gradient (DDPG) [17], which is an off-policy actor-critic algorithm for continuous control problem. In our environment, one step means that our agent makes an action to decide the number of bits assigned to the weights or activations of a specific layer, while one episode is composed of multiple steps, where our RL agent makes actions to all layers. We apply a variant form of the Bellman's Equation, where each transition in an episode is defined as $T_k = (O_k, a_k, \mathcal{R}, O_{k+1})$. During exploration, the Q-function is computed as

$$\hat{Q}_k = \mathcal{R}_k - \mathcal{B} + \gamma \times Q(O_{k+1}, w(O_{k+1}) \mid \theta^Q), \quad (7)$$

and the loss function can be approximated by

$$\mathcal{L} = \frac{1}{N_\text{s}} \sum_{k=1}^{N_\text{s}} (\hat{Q}_k - Q(O_k, a_k \mid \theta^Q))^2, \quad (8)$$

where N_s denotes the number of steps in this episode, and the baseline $\mathcal{B}$ is defined as an exponential moving average of all previous rewards in order to reduce the variance of the gradient estimation. The discount factor γ is set to 1 since we assume that the action made for each layer should contribute equally to the final result. Moreover, as the number of steps is always finite (bounded by the number of layers), the sum of the rewards will not explode.

3.7. Implementation Details

In this section, we present the implementation details about RL exploration and finetuning quantized models.

Agent. The DDPG agent consists of an actor network and a critic network. Both using the same network architecture: they take the state vector and the action from the last time step as inputs and feed them into two separate fully-connected layers with hidden sizes of 400. After that, we add the two hidden vectors together and go through another two fully-connected layers with hidden sizes of $\{300, 1\}$. As for the actor network, we use an additional sigmoid function to project the output into the range of $[0, 1]$.

Exploration. Optimization of the DDPG agent is carried out using ADAM [15] with $\beta_1 = 0.9$ and $\beta_2 = 0.999$. We use a fixed learning rate of 10^{-4} for the actor network and 10^{-3} for the critic network. During exploration, we employ the following stochastic process of the noise:

$$w'(O_k) \sim \mathcal{N}_\text{trunc}(w(O_k \mid \theta_k^w), \sigma^2, 0, 1), \quad (9)$$

where $\mathcal{N}_\text{trunc}(\mu, \sigma, a, b)$ is the truncated normal distribution, and w is the model weights. The noise σ is initialized as 0.5, and after each episode, the noise is decayed exponentially with a decay rate of 0.99.

	Hardware	Batch	PE Array	AXI port	Block RAM
Edge	Zynq-7020	1	8×8	4×64b	140×36Kb
Cloud	VU9P	16	16×16	4×256b	2160×36Kb

Table 2: The configurations of edge and cloud accelerators.

Finetuning. During exploration, we finetune the quantized model for one epoch to help recover the performance (using SGD with a fixed learning rate of 10^{-3} and momentum of 0.9). We randomly select 100 categories from ImageNet [5] to accelerate the model finetuning during exploration. After exploration, we quantize the model with our best policy and finetune it on the full dataset.

4. Experiments

We conduct extensive experiments to demonstrate the consistent effectiveness of our framework for multiple objectives: *latency*, *energy*, and *model size*.

Datasets and Models. Our experiments are performed on the ImageNet [5] dataset. As our focus is on more efficient models, we extensively study the quantization of MobileNet-V1 [12] and MobileNet-V2 [24]. Both MobileNets are inspired from the depthwise separable convolutions [3] and replace the regular convolutions with the *pointwise* and *depthwise* convolutions: MobileNet-V1 stacks multiple "*depthwise – pointwise*" blocks repeatedly; while MobileNet-V2 uses the "*pointwise – depthwise – pointwise*" blocks as its basic building primitives.

4.1. Latency-Constrained Quantization

We first evaluate our framework under *latency* constraints on two representative hardware architectures: spatial and temporal architectures for multi-precision CNN. We show that it's beneficial to have specialized quantization policies for different hardware architectures. We systematically interpret the policy given by AI to guide future human designs.

Temporal Architecture. Bit-Serial Matrix Multiplication Overlay (BISMO) proposed by Yaman *et al*. [26] is a classic temporal design of neural network accelerator on FPGA. It introduces bit-serial multipliers which are fed with one-bit digits from 256 weights and corresponding activations in parallel at one time and accumulates their partial products by shifting over time.

Spatial Architecture. BitFusion architecture proposed by Hardik *et al*. [25] is a state-of-the-art spatial ASIC design for neural network accelerator. It employs a 2D systolic array of Fusion Units which spatially sum the shifted partial products of two-bit elements from weights and activations.

		Edge Accelerator						Cloud Accelerator					
		MobileNet-V1			MobileNet-V2			MobileNet-V1			MobileNet-V2		
	Bitwidths	Acc.-1	Acc.-5	Latency	Acc.-1	Acc.-5	Latency	Acc.-1	Acc.-5	Latency	Acc.-1	Acc.-5	Latency
PACT [2]	4 bits	62.44	84.19	45.45 ms	61.39	83.72	52.15 ms	62.44	84.19	57.49 ms	61.39	83.72	74.46 ms
Ours	*flexible*	**67.40**	**87.90**	45.51 ms	**66.99**	**87.33**	52.12 ms	**65.33**	**86.60**	57.40 ms	**67.01**	**87.46**	73.97 ms
PACT [2]	5 bits	67.00	87.65	57.75 ms	68.84	88.58	66.94 ms	67.00	87.65	77.52 ms	68.84	88.58	99.43 ms
Ours	*flexible*	**70.58**	**89.77**	57.70 ms	**70.90**	**89.91**	66.92 ms	**69.97**	**89.37**	77.49 ms	**69.45**	**88.94**	99.07 ms
PACT [2]	6 bits	70.46	89.59	70.67 ms	71.25	90.00	82.49 ms	70.46	89.59	99.86 ms	71.25	90.00	127.07 ms
Ours	*flexible*	**71.20**	**90.19**	70.35 ms	**71.89**	**90.36**	82.34 ms	**71.20**	**90.08**	99.66 ms	**71.85**	**90.24**	127.03 ms
Original	8 bits	70.82	89.85	96.20 ms	71.81	90.25	115.84 ms	70.82	89.85	151.09 ms	71.81	90.25	189.82 ms

Table 3: Latency-constrained quantization on BISMO (edge accelerator and cloud accelerator) on ImageNet. Our framework can reduce the latency by **1.4**× to **1.95**× with negligible loss of accuracy compared with the fixed bitwidth (8 bits) quantization.

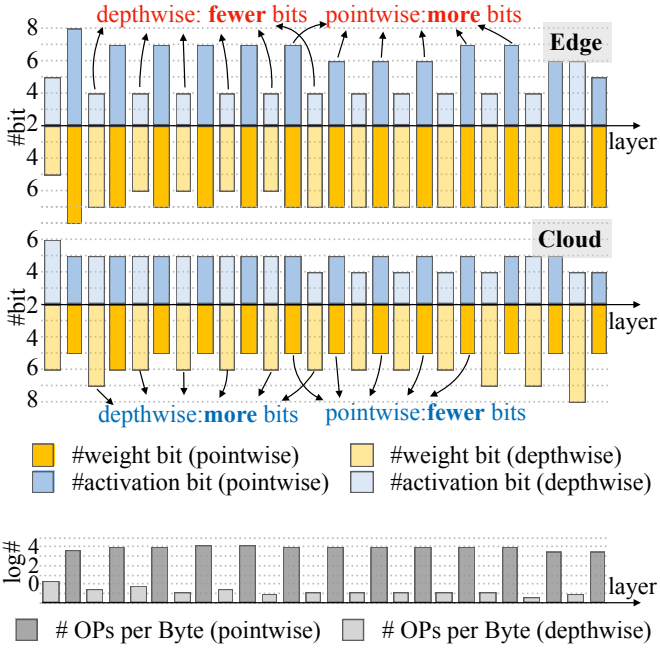

Figure 3: Quantization policy under latency constraints for MobileNet-V1. On edge accelerator, our RL agent allocates *less* activation bits to the depthwise convolutions, which echos that the depthwise convolutions are memory bounded and the activations dominates the memory access. On cloud accelerator, our agent allocates *more* bits to the depthwise convolutions and allocates *less* bits to the pointwise convolutions, as cloud device has more memory bandwidth and high parallelism, the network appears to be computation bounded.

4.1.1 Quantization policy for BISMO Architecture

Inferencing neural networks on edge devices and cloud severs can be quite different: batch size, memory bandwidth, peak FLOPs, *etc*. We use Xilinx Zynq-7020 FPGA [29] as our edge device and Xilinx VU9P [28] as our cloud device. Table 2 shows our experiment configurations on these two platforms along with their available resources.

As for comparison, we adopt the PACT [2] as our baseline, which uses the same number of bits for all layers except for the first layer which extracts the low level features, they use 8 bits for both weights and activations as it has fewer parameters and is very sensitive to errors. We follow a similar setup for the first layer (8 bits), and explore the bitwidth allocation policy for all the other layers. Under the same latency, HAQ consistently achieved better accuracy than the baseline on both the cloud and the edge (Table 3). With similar accuracy, HAQ can reduce the latency by 1.4× to 1.95× compared with the baseline.

Interpreting the quantization policy. Our agent gave quite different quantization policy for edge and cloud accelerators (Figure 3). For the activations, the depthwise convolution layers are assigned less bitwidth than the pointwise layers on the edge; while on the cloud device, the bitwidth of these two types of layers are similar. For weights, the bitwidth of these types of layers are nearly the same on the edge; while on the cloud, the depthwise convolution layers got more bitwidth than the pointwise convolution layers.

We explain the difference of quantization policy between edge and cloud by the roofline model [27]. Many previous works use FLOPs or BitOPs as metrics to measure computation complexity. However, they are not able to directly reflect the latency, since there are many other factors influencing the hardware performance, such as memory access cost and degree of parallelism [24, 20]. Taking computation and memory access into account, the roofline model assumes that applications are either computation-bound or memory bandwidth-bound, if not fitting in on-chip caches, depending on their operation intensity. Operation intensity is measured as operations (MACs in neural networks) per byte accessed. A lower operation intensity indicates suffering more from the memory access.

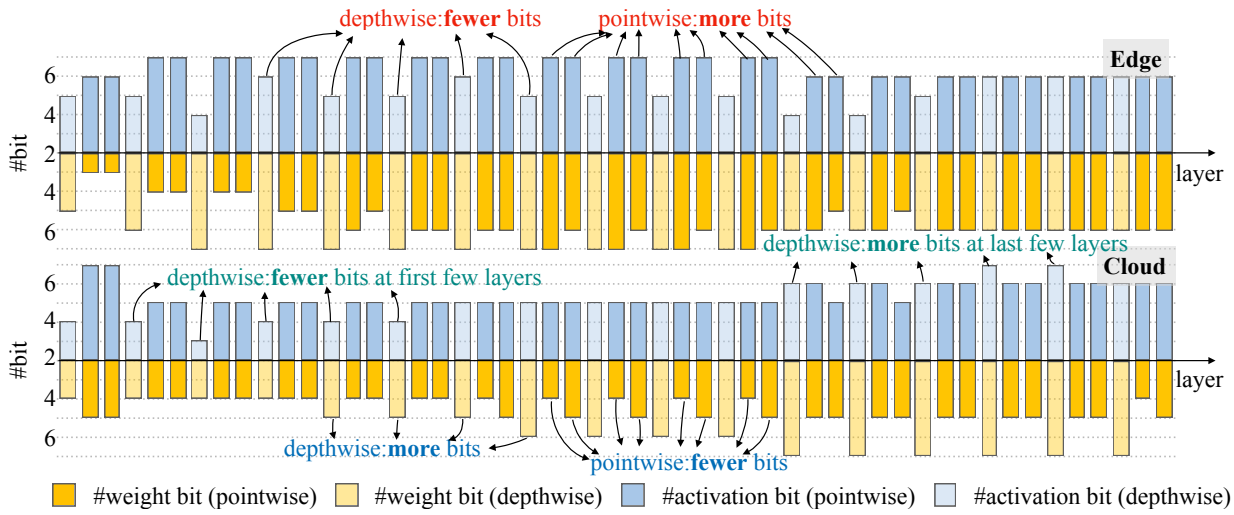

Figure 4: Quantization policy under latency constraints for MobileNet-V2 on BISMO. Similar to Figure 3, depthwise layer is assigned with fewer bits on the edge accelerator, and pointwise layer is assigned with fewer bits on the cloud accelerator.

	Weights	Activations	Acc.-1	Acc.-5	Latency
PACT [2]	4 bits	4 bits	62.44	84.19	7.86 ms
Ours	*flexible*	*flexible*	**67.45**	**87.85**	7.86 ms
PACT [2]	6 bits	4 bits	67.51	87.84	11.10 ms
Ours	*flexible*	*flexible*	**70.40**	**89.69**	11.09 ms
PACT [2]	6 bits	6 bits	70.46	89.59	19.99 ms
Ours	*flexible*	*flexible*	**70.90**	**89.95**	19.98 ms
Original	8 bits	8 bits	70.82	89.85	20.08 ms

Table 4: Latency-constrained quantization on BitFusion (MobileNet-V1 on ImageNet). Our framework can reduce the latency by **2**× with almost no loss of accuracy compared with the fixed bitwidth (8 bits) quantization.

	Weights	Activations	Acc.-1	Acc.-5	Energy
PACT [2]	4 bits	4 bits	62.44	84.19	13.47 mJ
Ours	*flexible*	*flexible*	**64.78**	**85.85**	13.69 mJ
PACT [2]	6 bits	4 bits	67.51	87.84	16.57 mJ
Ours	*flexible*	*flexible*	**70.37**	**89.40**	16.30 mJ
PACT [2]	6 bits	6 bits	70.46	89.59	26.80 mJ
Ours	*flexible*	*flexible*	**70.90**	**89.73**	26.67 mJ
Original	8 bits	8 bits	70.82	89.95	31.03 mJ

Table 5: Energy-constrained quantization on BitFusion (MobileNet-V1 on ImageNet). Our framework reduces the power consumption by **2**× with nearly no loss of accuracy compared with the fixed bitwidth quantization.

The bottom of Figure 3 shows the operation intensities (OPs per Byte) of convolution layers in the MobileNet-V1. Depthwise convolution is memory bounded, and the pointwise convolution is computation bounded. Our experiments show that when running MobileNet-V1 on the edge devices with small batch size, its latency is dominated by the depthwise convolution layers. Since the feature maps take a major proportion in the memory of depthwise convolution layers, our agent gives the activations less bits. In contrast, when running MobileNet-V1 on the cloud with large batch size, our agent increases the bitwidth of depthwise convolution to preserve the accuracy at low memory overhead since depthwise convolution only takes a small proportion of the total weights. A similar phenomenon can be observed in Figure 4 on MobileNet-V2. Moreover, as the activation size in deeper layers gets smaller, they get assigned more bits.

4.1.2 Quantization policy for BitFusion Architecture

In order to demonstrate the effectiveness of our framework on different hardware architectures, we further compare our framework with PACT [2] under the latency constraints on the BitFusion [25] architecture (Table 4). Our framework performs much better than the hand-craft policy with the same latency. It can achieve almost no degradation of accuracy with only half of the latency used by the original MobileNet-V1 model (from **20.08** to **11.09** ms). Therefore, our framework is flexible to provide specialized quantization policy for different hardware platforms.

4.2. Energy-Constrained Quantization

We then evaluate our framework under the energy constraints. Similar to the latency-constrained experiments, we compare our framework with PACT [2] that uses fixed number of bits without hardware feedback. From Table 5, we can clearly see that our framework outperforms the rule-based baseline: it achieves much better performance while consuming similar amount of energy. In particular, our framework is able to achieve almost no loss of accuracy with nearly half of the energy consumption of the original MobileNet-V1 model (from **31.03** to **16.57** mJ), which suggests that mixed preci-

		MobileNet-V1			MobileNet-V2			ResNet-50		
	Weights	Acc.-1	Acc.-5	Model Size	Acc.-1	Acc.-5	Model Size	Acc.-1	Acc.-5	Model Size
Han *et al*. [8]	2 bits	37.62	64.31	1.09 MB	58.07	81.24	0.96 MB	68.95	88.68	6.32 MB
Ours	*flexible*	**57.14**	**81.87**	1.09 MB	**66.75**	**87.32**	0.95 MB	**70.63**	**89.93**	6.30 MB
Han *et al*. [8]	3 bits	65.93	86.85	1.60 MB	68.00	87.96	1.38 MB	75.10	92.33	9.36 MB
Ours	*flexible*	**67.66**	**88.21**	1.58 MB	**70.90**	**89.76**	1.38 MB	**75.30**	**92.45**	9.22 MB
Han *et al*. [8]	4 bits	71.14	89.84	2.10 MB	71.24	89.93	1.79 MB	**76.15**	92.88	12.40 MB
Ours	*flexible*	**71.74**	**90.36**	2.07 MB	**71.47**	**90.23**	1.79 MB	76.14	**92.89**	12.14 MB
Original	32 bits	70.90	89.90	16.14 MB	71.87	90.32	13.37 MB	76.15	92.86	97.49 MB

Table 6: Model size-constrained quantization on ImageNet. Compared with Deep Compression [7], our framework achieves higher accuracy under similar model size (especially under high compression ratio).

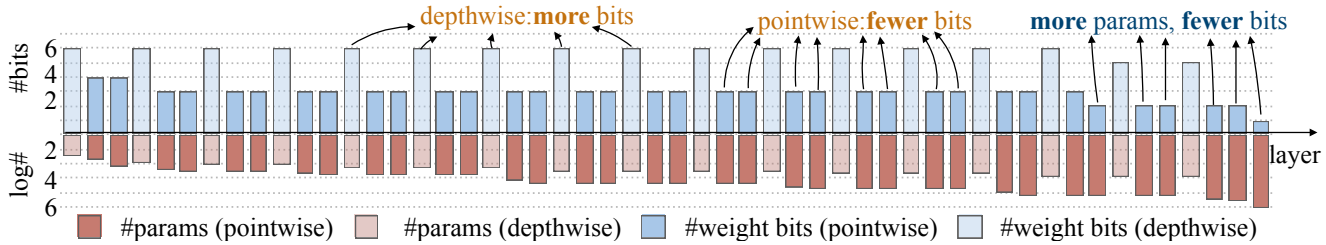

Figure 5: Quantization policy under model size constraints for MobileNet-V2. Our RL agent allocates *more* bits to the depthwise convolutions, since depthwise convolutions have *fewer* number of parameters.

sion with hardware-aware, specialized quantization policy can indeed help reduce the energy consumption.

4.3. Model Size-Constrained Quantization

Finally, we evaluate our framework under the model size constraints. Following Han *et al*. [8], we employ the k-means algorithm to quantize the values into k different centroids instead of using the linear quantization for compression, since k-means quantization can be more effective reducing the model size.

We compare our framework with Deep Compression [8] on MobileNets and ResNet-50. From Table 6, we can see that our framework performs much better than Deep Compression: it achieves higher accuracy with the same model size. For compact models like MobileNets, Deep Compression significantly degrades the performance especially under aggressive quantization, while our framework can preserve the accuracy much better. For instance, when Deep Compression quantizes the weights of MobileNet-V1 to 2 bits, the accuracy drops significantly from 70.90 to **37.62**; while our framework can still achieve **57.14** of accuracy with the same model size. The reason is our framework makes full use of the mixed precision by systematically searching the optimal quantization policy.

Discussions. In Figure 5, we visualize the bitwidth allocation strategy for MobileNet-V2. From this figure, we can observe that our framework assigns *more* bitwidths to the weights in depthwise convolution layers than pointwise convolution layers. Intuitively, this is because the number of parameters in the former is much smaller than the latter. Comparing Figure 4 and Figure 5, the policies are drastically different under different optimization objectives (**fewer** bitwiths for depthwise convolutions under *latency* optimization, **more** bitwidths for depthwise convolutions under *model size* optimization). Our framework succeeds in learning to adjust its bitwidth policy under different constraints.

5. Conclusion

In this paper, we propose **H**ardware-Aware **A**utomated **Q**uantization (HAQ), an automated framework for quantization which does not require any domain experts and rule-based heuristics. We provide a learning based method that can search the quantization policy with hardware feedback. Compared with indirect proxy signals, our framework can offer a specialized quantization solution for different hardware platforms. Extensive experiments demonstrate that our framework performs better than conventional rule-based approaches for multiple objectives: latency, energy and model size. Our framework reveals that the optimal policies on different hardware architectures are drastically different, and we interpreted the implication of those policies. We believe the insights will inspire the future software and hardware co-design for efficient deployment of deep neural networks.

Acknowledgements. We thank MIT Quest for Intelligence, MIT-IBM Watson AI Lab, Xilinx, Samsung, Intel, ARM, Qualcomm, and SONY for supporting this research. We thank Google Cloud and AWS Machine Learning Research Awards for providing the computation resource.

References

[1] Han Cai, Jiacheng Yang, Weinan Zhang, Song Han, and Yong Yu. Path-Level Network Transformation for Efficient Architecture Search. In *ICML*, 2018. 3

[2] Jungwook Choi, Zhuo Wang, Swagath Venkataramani, Pierce I-Jen Chuang, Vijayalakshmi Srinivasan, and Kailash Gopalakrishnan. PACT: Parameterized Clipping Activation for Quantized Neural Networks. *arXiv*, 2018. 1, 6, 7

[3] François Chollet. Xception - Deep Learning with Depthwise Separable Convolutions. In *CVPR*, 2017. 4, 5

[4] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1. *arXiv*, 2016. 2

[5] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Fei-Fei Li. ImageNet - A large-scale hierarchical image database. In *CVPR*, 2009. 5

[6] EENews. Apple describes 7nm a12 bionic chips, 2018. 1

[7] Song Han. *Efficient Methods and Hardware for Deep Learning*. PhD thesis, 2017. 8

[8] Song Han, Huizi Mao, and William Dally. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. In *ICLR*, 2016. 1, 2, 8

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *CVPR*, 2016. 2

[10] 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 *ECCV*, 2018. 3

[11] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *ICCV*, 2017. 3

[12] 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*, 2017. 1, 2, 5

[13] Imagination. Powervr neural network accelerator, 2018. 2

[14] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew G Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. In *CVPR*, 2018. 1, 3

[15] Diederik Kingma and Jimmy Ba. Adam - A Method for Stochastic Optimization. In *ICLR*, 2015. 5

[16] Raghuraman Krishnamoorthi. Quantizing deep convolutional networks for efficient inference - A whitepaper. *arXiv*, 2018. 3

[17] Timothy Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *ICLR*, 2016. 2, 5

[18] Ji Lin, Yongming Rao, Jiwen Lu, and Jie Zhou. Runtime Neural Pruning. In *NIPS*, 2017. 1

[19] 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 *ECCV*, 2018. 3

[20] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *ICCV*, 2017. 3, 6

[21] Nvidia. Nvidia tensor cores, 2018. 1

[22] Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. Efficient Neural Architecture Search via Parameter Sharing. In *ICML*, 2018. 3

[23] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. XNOR-Net - ImageNet Classification Using Binary Convolutional Neural Networks. In *ECCV*, 2016. 3

[24] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. MobileNetV2: Inverted Residuals and Linear Bottlenecks. In *CVPR*, 2018. 2, 5, 6

[25] 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 network. In *ISCA*, 2018. 2, 5, 7

[26] Yaman Umuroglu, Lahiru Rasnayake, and Magnus Sjalander. Bismo: A scalable bit-serial matrix multiplication overlay for reconfigurable computing. In *FPL*, 2018. 2, 5

[27] Samuel Williams, Andrew Waterman, and David Patterson. Roofline: an insightful visual performance model for multicore architectures. *Communications of the ACM*, 52(4):65–76, 2009. 6

[28] Xilinx. Ultrascale architecture and product data sheet: Overview, 2018. 6

[29] Xilinx. Zynq-7000 soc data sheet: Overview, 2018. 6

[30] Tien-Ju Yang, Yu-Hsin Chen, and Vivienne Sze. Designing energy-efficient convolutional neural networks using energy-aware pruning. *arXiv*, 2016. 3

[31] Tien-Ju Yang, Andrew Howard, Bo Chen, Xiao Zhang, Alec Go, Mark Sandler, Vivienne Sze, and Hartwig Adam. Netadapt: Platform-aware neural network adaptation for mobile applications. In *ECCV*, 2018. 3

[32] Aojun Zhou, Anbang Yao, Kuan Wang, and Yurong Chen. Explicit loss-error-aware quantization for low-bit deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9426–9435, 2018. 3

[33] Shuchang Zhou, Zekun Ni, Xinyu Zhou, He Wen, Yuxin Wu, and Yuheng Zou. DoReFa-Net - Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients. *arXiv*, 2016. 3

[34] Chenzhuo Zhu, Song Han, Huizi Mao, and William Dally. Trained Ternary Quantization. In *ICLR*, 2017. 1, 3

[35] Barret Zoph and Quoc V Le. Neural Architecture Search with Reinforcement Learning. In *ICLR*, 2017. 3