DELVING INTO CHANNELS: EXPLORING HYPERPARAMETER SPACE OF CHANNEL BIT WIDTHS WITH LINEAR COMPLEXITY

Anonymous authors
Paper under double-blind review

ABSTRACT

Allocating different bit widths to different channels and quantizing them independently bring higher quantization precision and accuracy. Most of prior works use equal bit width to quantize all layers or channels, which is sub-optimal. On the other hand, it is very challenging to explore the hyperparameter space of channel bit widths, as the search space increases exponentially with the number of channels, which could be tens of thousand in a deep neural network. In this paper, we address an important problem of efficiently exploring the hyperparameter space of channel bit widths. We formulate the quantization of deep neural networks as a rate-distortion optimization problem, and present an ultra-fast algorithm to search the bit allocation of channels. Our approach has only linear time complexity and can find the optimal bit allocation within a few minutes on CPU. In addition, we provide an effective way to improve the performance on target hardware platforms. We restrict the bit rate (size) of each layer to allow as many weights and activations as possible to be stored on-chip, and incorporate hardware-aware constraints into our objective function. The hardware-aware constraints do not cause additional overhead to optimization, and have very positive impact on hardware performance. Experimental results show that our approach achieves state-of-the-art quantization results on four deep neural networks, ResNet-18, ResNet-34, ResNet-50, and MobileNet-v2, on ImageNet. Hardware simulation results demonstrate that our approach is able to bring up to 3.5× and 3.0× speedup on two deep-learning accelerators, TPU and Eyeriss, respectively.

1 INTRODUCTION

Deep Learning (LeCun et al., 2015) has become the de-facto technique in Computer Vision, Natural Language Processing, Speech Recognition, and many other fields. However, the high accuracy of deep neural networks (Krizhevsky et al., 2012) comes at the cost of high computational complexity. Due to the large model size and huge computational cost, deploying deep neural networks on mobile devices is very challenging, especially on tiny devices. It is therefore important to make deep neural networks smaller and faster through model compression (Han et al., 2016), to deploy deep neural networks on resource-limited devices.

Quantization (Han et al., 2016) is one of the standard techniques for neural network compression. One problem existed in prior works is that they typically use equal bit width to quantize weights and activations of all layers, which is sub-optimal because weights and activations in different layers react differently on quantization. They should be treated independently and quantized with un-equal bit widths. Moreover, most of prior works only consider the model size and accuracy in their methods and do not consider the system-level performance when deploying quantized models on hardware platforms. As has been illustrated (Wang et al., 2019), a well quantized network can not guarantee superior performance on hardware platforms. To address these two issues, the recently proposed mixed-precision quantization methods assign un-equal bit widths across layers, and optimize the hardware metrics directly in the quantization mechanism. For example, HAQ (Wang et al., 2019) proposed a reinforcement learning method to learn the bit widths of weights and activations across layers and minimize latency and energy in objective function directly.

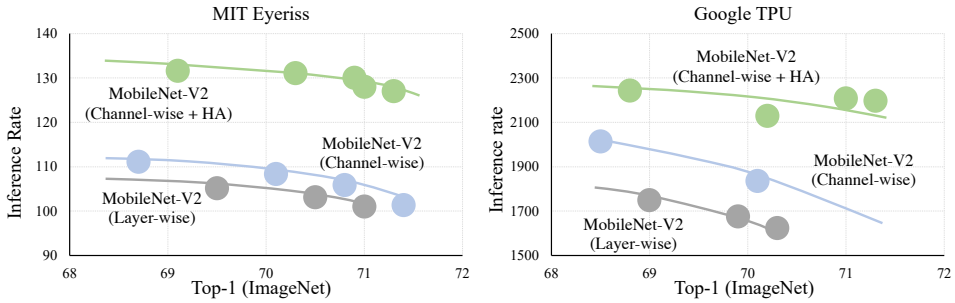

Figure 1: Channel-wise bit allocation plus hardware-aware constraints (HA) achieves the best performance on Eyeriss and TPU. Channel-wise bit allocation outperforms layer-wise bit allocation because of higher quantization precision. Inference Rate: number of images per second.

Although noticeable improvement has been obtained by mixed-precision quantization, the layer-wise bit allocation scheme is still sub-optimal, since all channels in a CONV layer are quantized with equal bit width. In fact, different channels react very distinctively to quantization. Higher precision can be obtained if allocating un-equal bit widths to channels. However, the challenge is that the hyperparameter space of channel bit widths increases exponentially with the number of channels. Given N channels and C bit widths, the search complexity is $O(C^N)$, where in a deep neural network, N can be tens of thousand or even more. Such huge search space could make it unaffordable for a heuristic search method, like reinforcement learning (Sutton & Barto, 2018), to find solution within limited time.

In this paper, we propose a new approach to efficiently explore the hyperparameter space of channel bit widths. We apply the classic coding theories (Taubman & Marcellin, 2002), and formulate the quantization of weights and activations as a rate-distortion optimization problem. Since the output distortion is highly related to accuracy, by minimizing the output distortion induced by quantization, our approach is able to well maintain the accuracy at very low bit widths. We then search the optimal bit allocation across channels in a rate-distortion optimized manner. Through utilizing the additivity property of output distortion, we present an ultra-fast algorithm with linear time complexity to find the optimal channel-wise bit allocation, by using Lagrangian formulation. Our algorithm only costs a few minutes on CPU for a deep neural network.

What's more, we present an alternative way to improve the system-level performance when deploying quantized networks on target hardware platforms. Prior works typically optimize the hardware metrics of a whole network directly, and need real-time feedback from simulators in their learning procedure, which could cause additional overhead to optimization. Instead, our approach improves hardware performance by restricting the size of each individual layer, and does not require feedback from simulators. Our key insight is that the volume of weights and activations in some layers is particularly significant, which exceeds the capacity of on-chip memory. As a result, these layers significantly prolong the inference time due to the necessity of slow data access to off-chip memory. We thus constrain the size of these large layers to ensure that all variables can be stored on-chip.

To our best knowledge, only one prior work, AutoQ (Lou et al., 2020), finds channel-wise bit allocation, and optimizes the performance on hardware platforms simultaneously. AutoQ employs reinforcement learning to solve the bit allocation problem, which is time-consuming, and could fall into a local optimum in their heuristic search method. Our approach adopts Lagrangian formulation for fast optimization, and is able to find global optimum solution in a rate-distortion optimized manner. We summarize the main contributions of our paper as following:

- We formulate the quantization of deep neural networks as a rate-distortion optimization problem, and optimize the channel-wise bit allocation for higher accuracy. We present an ultra-fast algorithm with linear time complexity to efficiently explore the hyperparameter space of channel bit widths.
- We present a simple yet effective way to improve the performance on target hardware platforms, through restricting the size of each individual layer and incorporating hardware-

aware constraints into our objective function. The hardware-aware constraints can be integrated seamlessly, without causing additional overhead to optimization.

- Our approach achieves state-of-the-art results on various deep neural networks on ImageNet. Hardware simulation results demonstrate that our approach is able to bring considerable speedups for deep neural networks deployed on hardware platforms.

2 APPROACH

We utilize classic coding theories (Taubman & Marcellin, 2002), and formulate the quantization of deep neural networks as a rate-distortion optimization problem. The differentiability of input-output relationships for the layers of neural networks allows us to relate output distortion to the bit rate (size) of quantized weights and activations. We discuss the formulation of our approach and its optimization in this section.

2.1 FORMULATION

Let f denote a deep neural network. Given an input $\mathbf{I}$, we denote $\boldsymbol{y}$ as the output of f, i.e. $\boldsymbol{y} = f(\mathbf{I})$. When performing quantization on weights and activations, a modified output vector $\widehat{\boldsymbol{y}}$ would be received. The output distortion is measured by the distance between $\boldsymbol{y}$ and $\widehat{\boldsymbol{y}}$, which is defined as

$$\delta = \|\boldsymbol{y} - \widehat{\boldsymbol{y}}\|_2^2 \tag{1}$$

Here Euclidean distance $\|.\|_2$ is adopted. Our approach allocates different quantization bit widths to different weight channels and layer activations. We aim to minimize the output distortion under the constraint of bit rate. Given the bit rate constraint r, the rate-distortion optimization problem is formulated as

$$\min \delta = \|\boldsymbol{y} - \widehat{\boldsymbol{y}}\|_2^2 \;\; s.t. \sum_{i=1}^{l} \sum_{j=1}^{n_i} R_{i,j}^w + \sum_{i=1}^{l} R_i^a \leq r, \tag{2}$$

where $R_{i,j}^w$ denotes the bit rate of weight channel j in layer i, R_i^a denotes the bit-rate of activations in layer i, n_i denotes the number of channels in layer i, and l denotes the total number of layers. Specifically, $R_{i,j}^w$ equals to the quantization bit width of channel j in layer i, $B_{i,j}^w$, multiplied by the number of weights in that channel; R_i^a equals to the quantization bit width of activations in layer i, B_i^a, multiplied by the number of activations in that layer.

We notice that output distortion is highly related to network accuracy. By minimizing output distortion induced by quantization, our approach is able to well maintain the accuracy at very high compression ratio. We provide an analysis regarding the relationship between output distortion and accuracy in appendix.

2.2 OPTIMIZING CHANNEL-WISE BIT ALLOCATION

We explore the additivity property of output distortion when performing quantization on weight channels and layer activations, and find that the additivity property holds, similar to the observation made in (Zhou et al., 2018; Zhe et al., 2019). Utilizing the additivity property, we develop an efficient Lagrangian formulation method to solve the bit allocation problem.

Let $\delta_{i,j}^w$ and δ_i^a denote the output distortion caused by quantizing an individual weight channel and an individual layer activation, respectively. The output distortion δ, caused by quantizing all weight channels and layer activations, equals the sum of output distortion due to the quantization of each individual part

$$\delta = \sum_{i=1}^{l} \sum_{j=1}^{n_i} \delta_{i,j}^w + \sum_{i=1}^{l} \delta_i^a \tag{3}$$

Equation (3) can be derived mathematically by linearizing the output distortion using Taylor series expansion with the assumption that the neural network is continuously differentiable and quantization errors can be considered as small deviations. The detailed mathematical derivation of Equation (3) is provided in appendix.

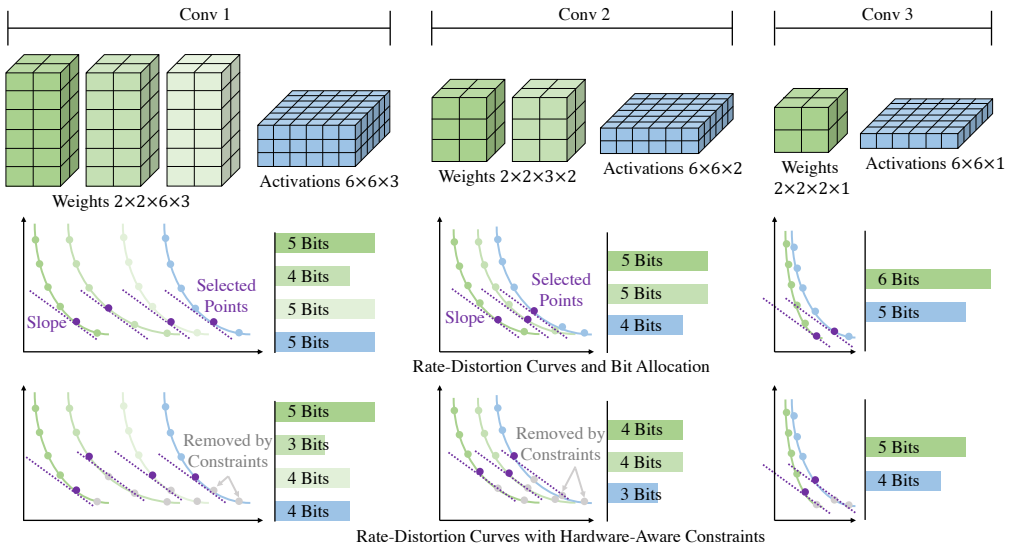

Figure 2: Examples of finding optimal bit allocation w/ and w/o hardware-aware constraints.

We then apply Lagrangian formulation (Shoham & Gersho, 1988) to solve objective function (2). The Lagrangian cost function of (2) is defined as

$$\mathcal{J} = \delta - \lambda \cdot \Big(\sum_{i=1}^{l} \sum_{j=1}^{n_i} R_{i,j}^{w} + \sum_{i=1}^{l} R_i^a - r \Big), \tag{4}$$

in which λ decides the trade-off between bit rate and output distortion. Setting the partial derivations of $\mathcal{J}$ to zero with respect to each $R_{i,j}^{w}$ and R_i^a and utilizing the additivity property in (3), we obtain the optimal condition

$$\frac{\partial \delta_{1,1}^{w}}{\partial R_{1,1}^{w}} = ... = \frac{\partial \delta_{1,n_1}^{w}}{\partial R_{1,n_1}^{w}} = \frac{\partial \delta_{1}^{a}}{\partial R_{1}^{a}} = ... = \frac{\partial \delta_{l,1}^{w}}{\partial R_{l,1}^{w}} = ... = \frac{\partial \delta_{l,n_l}^{w}}{\partial R_{l,n_l}^{w}} = \frac{\partial \delta_{l}^{a}}{\partial R_{l}^{a}} = \lambda \tag{5}$$

Equation (5) expresses that the slopes of all rate-distortion curves (output distortion versus bit rate functions) should be equal to obtain optimal bit allocation with minimal output distortion. According to (5), we are able to solve objective function (2) efficiently by enumerating slope λ and then choosing the points on rate-distortion curves with slope equal to λ as solution.

The algorithm works as follows. Before optimization, we quantize each weight channel and layer activation with different bit widths and calculate the output distortion caused by quantization to generate the rate-distortion curve for each weight channel and layer activation. After that, we assign a real value to λ, and select the point with slope equal to λ on each curve. The selected points on all curves correspond to a group of solution for bit allocation. In practice, we explore multiple values for λ until the network bit rate exceeds constraint R. We random select 50 images from training dataset to calculate output distortion caused by quantization. Given the number of λ evaluated, t, and the total number of quantization bit widths, b, the time complexity of optimization is $O((l + \sum_{i=1}^{l} n_i) \cdot t \cdot b)$. The algorithm is ultra-fast with only linear time complexity. It is able to find the answer in a few minutes on an Intel i7-6600U CPU with 2.6 GHz.

2.3 Choice of Quantizer

We adopt uniform quantizer for weights and activations. The quantization step size Δ is defined as a value of a power of 2, ranging from 2^{-16} to 2^{0}, where the one with minimal quantization error is selected. We clip all weights by $(-2^{b-1} \cdot \Delta, (2^{b-1} - 1) \cdot \Delta)$ and all activations by $(0, (2^{b} - 1))$, in which b is the quantization bit width. Note that our approach is compatible with any quantization techniques, including both uniform quantizer and non-uniform quantizer (e.g., K-Means (Gersho & Gray, 1991)). Since the focus of this paper is not the design of quantizer, we only evaluate

uniform quantizer in our approach. It is worth mentioning that applying a non-uniform quantizer could further improve the accuracy, but non-uniform quantizers are more complicated for computation, and require additional resources (e.g., look-up tables) for implementation. Similar as prior mixed-precision methods (Wang et al., 2019; Lou et al., 2020; Wu et al., 2019), we employ uniform quantizer in our paper, which is more hardware-friendly and is straightforward for implementation.

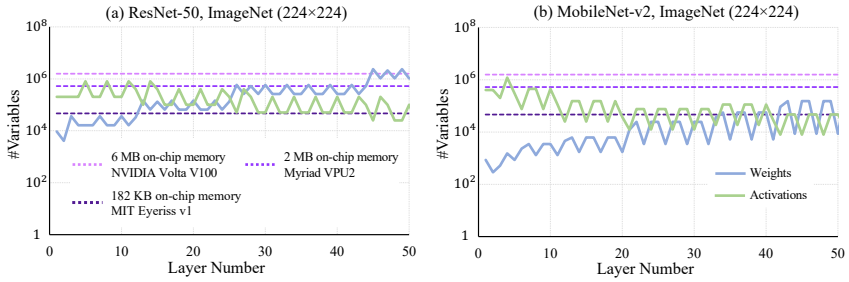

Figure 3: Number of weights and activations of layers and on-chip capacity on hardware platforms.

2.4 Improving Performance on Hardware Platforms

We consider improving inference rate as a guide to the design of our quantization mechanism. Inference rate is defined as the maximum number of images that a neural network can process per unit time. Memory access, especially data movement from off-chip memory to on-chip memory, dominates inference time, rather than convolutional operations (Han et al., 2015; 2016). We thus aim to maintain as many weights and activations as possible stored on-chip, and avoid data movement from off-chip memory to improve inference speed.

Our key insight is that the volume of weights and activations in some layers is particularly significant. As a result, part of weights and activations can not be stored on-chip, which leads to lots of memory-access traffic to off-chip DRAM. Fig. 3 illustrates the number of parameters across layers and the on-chip memory capacity on different hardware platforms. As we can see, on-chip memory capacity is very limited, and the size of part of layers exceeds the capacity. To this end, we restrict the quantization bit widths in these large layers to make sure that the size of these layers is less than on-chip memory capacity. Specifically, for each layer i, we have an independent bit rate constraint,

$$\sum_{j=1}^{n_i} K_{i,j}^w \cdot B_{i,j}^w + K_i^a \cdot B_i^a \leq m_{on}, \tag{6}$$

in which $K_{i,j}^w$ denotes the number of weights of channel j in layer i, K_i^a denotes the number of activations in layer i, and m_{on} denotes the on-chip memory capacity. In practice, we relax (6) into two items, and incorporate them to objective function (2),

$$B_{i,j}^w \leq \frac{m_{on}}{\sum_j K_{i,j}^w + \frac{\beta}{1-\beta} \cdot K_i^a}, \quad B_i^a \leq \frac{\alpha \cdot m_{on}}{\frac{1-\beta}{\beta} \cdot \sum_j K_{i,j}^w + K_i^a}, \tag{7}$$

for all $1 \leq i \leq l$ and $1 \leq j \leq n_i$, where α and β are two hyper-parameters, ranging from 0 to 1. Incorporating constraints (7) into (2), we have the objective function with the bit rate of each weight channel and layer activation constrained to improve hardware performance

$$\begin{aligned} &\min \delta = \|\boldsymbol{y} - \widehat{\boldsymbol{y}}\|_2^2 \quad s.t. \sum_{i=1}^{l} \sum_{j=1}^{n_i} R_{i,j}^w + \sum_{i=1}^{l} R_i^a \leq r, \\ &B_{i,j}^w \leq \frac{m_{on}}{\sum_j K_{i,j}^w + \frac{\beta}{1-\beta} \cdot K_i^a}, \quad B_i^a \leq \frac{\alpha \cdot m_{on}}{\frac{1-\beta}{\beta} \cdot \sum_j K_{i,j}^w + K_i^a} \end{aligned} \tag{8}$$

Note that the optimization of (8) is the same as that of (2). The only difference is that in (8) we have different search range for quantization bit widths. In (2), the range is from 1 to b where $b = 16$ is

the maximal bit width, while in (8), it is from 1 to $\frac{m_{on}}{\sum_j K^w_{i,j} + \frac{\beta}{1-\beta} \cdot K^a_i}$ for weight channels, and is from 1 to $\frac{\alpha \cdot m_{on}}{\frac{1-\beta}{\beta} \cdot \sum_j K^w_{i,j} + K^a_i}$ for layer activations. Incorporating bit rate constraints into objective function does not increase the search time. Actually, it slightly decreases the time as it reduces the search range of bit widths. Fig. 2 illustrates examples of the optimization procedure with and without constraints (7). Examples of distributions of bit rate across layers, implementation details of optimization, and the pseudo-codes are provided in appendix.

2.5 DISCUSSION

Prior works (Wang et al., 2019; Lou et al., 2020) typically use hardware simulators to guide the design of quantizatino mechanism. Implementing a simulator is complicated and calculating simulation results costs time. Alternatively, we provide a simple yet effective way to improve performance on hardware platforms. We directly restrict the bit rate of each layer to get weights and activations saved on-chip. Our method is easy to implement and does not cause additional overhead to optimization. Another advantage is that, once the rate-distortion curves are generated, our approach is able to find the bit allocation under any network size, by just changing the slope λ. This is better than most prior works which need to re-run the whole methods every time searching the bit allocation for a network size.

3 EXPERIMENTS

We report experimental results in this section. We first show quantization results on four deep neural networks, ResNet-18 (He et al., 2016), ResNet-34, ResNet-50 and MobileNet-v2 (Sandler et al., 2018), on the ImageNet dataset (Deng et al., 2009). We then provide results of inference rate on two hardware platforms, Google TPU (Jouppi et al., 2017) and MIT Eyeriss (Chen et al., 2016).

3.1 PARAMETER SETTINGS

We set hyper-parameters α and β as 0.3 and 0.5, respectively. We enumerate slope λ from -2^{-20} to -2^{20} until network size meets constraint r. Similar as prior works (Wang et al., 2019; Lou et al., 2020), we fine-tune the model after quantization up to 10 epochs with learning rate 0.0001. Straight-through estimator (ETS) (Bengio et al., 2013) is applied to perform back-propagation through non-differentiable quantization functions in fine-tuning.

3.2 QUANTIZATION RESULTS

Table 1 and Table 2 list the results of state-of-the-arts on four deep neural networks, ResNet-18, ResNet-34, ResNet-50, and MobileNet-v2, when weights and activations are quantized to very low bit width (i.e., 2 bits or 4 bits). As our approach allocates unequal bit widths to different weight channels and layer activations, we report the results when networks are quantized to the target size on average for fair comparison, same as prior works (Uhlich et al., 2019; Lou et al., 2020). Our approach, named as Rate-Distortion Optimized Quantization (RDOQ), improves state-of-the-arts on the four neural networks. Specifically, our approach outperforms STOA by 1.0%, 0.5%, and 1.2%, on ResNet-18, ResNet-34, and ResNet-50, respectively.

Note that the focus of our paper is on optimizing the bit allocation of channels, and we so adopt the normal uniform quantizer, while most of prior works in Table 1 and Table 2 focus on improving the quantizers. We notice that bit allocation strategies and quantizer techniques compensate each other. We expect to obtain higher accuracy by combining our bit allocation method with state-of-the-art quantizers, although this is out of the scope of our paper.

3.3 PERFORMANCE ON HARDWARE PLATFORMS

We examine the inference rate on two hardware platforms, Google TPU (Jouppi et al., 2017) and MIT Eyeriss (Chen et al., 2016), both of which are state-of-the-art architectures, inspired by current embedded and high-performance neural-network-targeted accelerator. We adopt the SCALE-Sim

Table 1: Top-1 accuracy of state-of-the-art methods at 2 bits on the ImageNet dataset.

Method	ResNet-18	ResNet-34	ResNet-50
Original (He et al., 2016)	69.3	73.0	75.5
LQ-Nets (Zhang et al., 2018)	64.9	68.8	71.5
PTG (Zhuang et al., 2018)	-	-	70.0
DSQ (Gong et al., 2019)	65.2	70.0	-
QIL (Jung et al., 2019)	65.7	70.6	-
ALQ (Qu et al., 2020)	66.4	71.0	-
APoT (Li et al., 2020)	67.3	70.9	73.4
SAT (Jin et al., 2020)	65.5	-	73.3
LSQ (Esser et al., 2020)	67.6	71.6	73.7
AUXI (Zhuang et al., 2020)	-	-	73.8
DMBQ (Zhao et al., 2021)	67.8	72.1	-
RDOQ (Ours)	**68.8**	**72.6**	**75.0**

Table 2: Results on MobileNet-v2 at 4 bits on the ImageNet dataset.

	Method	Top1 Accu.	Top5 Accu.
MobileNet-v2	Original (Sandler et al., 2018)	71.8	90.2
	HAQ (Wang et al., 2019)	67.0	87.3
	DQ (Uhlich et al., 2019)	69.7	-
	DSQ (Gong et al., 2019)	64.8	-
	AutoQ (Lou et al., 2020)	69.0	89.4
	SAT (Jin et al., 2020)	71.1	89.7
	LLSQ (Zhao et al., 2020)	67.4	88.0
	RDOQ (Ours)	**71.3**	**90.0**

software (Samajdar et al., 2018) to simulate the time cycles of ResNet-50 and MobileNet-v2, when mapped into the two considered hardware platforms.

Table 3 illustrates the inference rate on TPU and Eyeriss. Our approach significantly improves the reference rate, compared with originally uncompressed neural networks. We speed up the inference rate by 3.5x and 3.0x for ResNet-50 on TPU and Eyeriss, and by 1.5x and 2.0x for MobileNet-V2 on TPU and Eyeriss, without hurting the accuracy (loss $\leq 0.5\%$). We also compare our approach with competitive mixed-precision quantization method HAQ (Wang et al., 2019) and competitive equal bit quantization method DoReFa+PACT (Zhou et al., 2016; Choi et al., 2018). Our approach outperforms both HAQ and DoReFa+PACT. We notice that although equal bit quantization method DoReFa+PACT obtains superior quantization results, the performance on hardware platform is not high. This is because DoReFa+PACT is not hardware-aware as they do not optimize the hardware performance in their quantization mechanism.

We should clarify that both TPU and Eyeriss can not support fixed-point computation with mixed-precision, so 8-bit and 16-bit floating-point computation is performed on TPU and Eyeriss, respectively, as default. As a result, our gains come from the reduction of memory access, but not from the reduction of computation. Since our approach enables weights and activations saved on-chip, accessing off-chip memory could be reduced. However, the computation time is the same as that of the original models, as we perform floating-point computation. We expect to see higher inference rate if we evaluate the performance on architectures supporting fixed-point computation with mixed-precision. Fig. 4 illustrates a breakdown of the inference time on TPU and Eyeriss.

3.4 Time Cost

Table 4 lists the time of our approach to find channel-wise bit allocation on four deep neural networks. Our approach takes about a few minutes on a normal CPU (Intel Core i7 6600U CPU with 2.60 GHZ) to find the answer. We also evaluate HAQ (Wang et al., 2019) — the competitive mixed-

precision quantization method built upon reinforcement learning. As we can see, reinforcement-learning-based approach requires several days on multiple GPUs to search the bit allocation for one time, and is orders of magnitude slower, compared with our approach. Our approach provides an alternative way to very quickly explore the hyperparameter space of bit widths for deep neural networks, and is particularly suitable for the case without powerful computation resources.

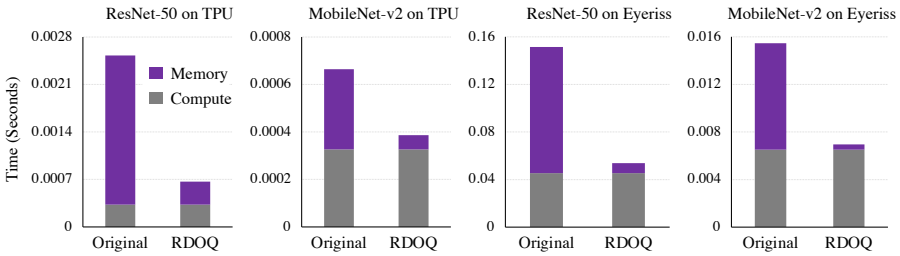

Figure 4: A breakdown for the compute time and memory time on Google TPU and MIT Eyeriss.

Table 3: Inference Rate on Google TPU and MIT Eyeriss. We show the results of our approach with hardware-aware (HA) constrains in this table.

Method	Accuracy	Google TPU	MIT Eyeriss
ResNet-50			
Original (He et al., 2016)	75.5	361	5.6
DoReFa+PACT (Choi et al., 2018)	76.5	646 (1.8×)	12.6 (2.3×)
DoReFa+PACT (Choi et al., 2018)	72.2	920 (2.6×)	13.7 (2.5×)
RDOQ+HA (Ours)	76.5	769 (2.1×)	13.9 (2.5×)
RDOQ+HA (Ours)	76.2	904 (2.5×)	15.0 (2.7×)
RDOQ+HA (Ours)	75.0	1254 (3.5×)	17.0 (3.0×)
MobileNet-V2			
Original (Sandler et al., 2018)	71.1	1504	64
DoReFa+PACT (Choi et al., 2018)	71.2	1698 (1.1×)	104 (1.6×)
DoReFa+PACT (Choi et al., 2018)	70.4	1764 (1.2×)	108 (1.7×)
HAQ (Wang et al., 2019)	71.2	2067 (1.4×)	124 (1.9×)
HAQ (Wang et al., 2019)	68.9	2197 (1.5×)	128 (2.0×)
RDOQ+HA (Ours)	71.3	2197 (1.5×)	127 (2.0×)
RDOQ+HA (Ours)	71.0	2207 (1.5×)	128 (2.0×)
RDOQ+HA (Ours)	70.9	2256 (1.5×)	130 (2.0×)

Table 4: Time cost to find bit allocation on deep neural networks.

Method	Device	ResNet-18	ResNet-34	ResNet-50	Mobile-V2
HAQ (Wang et al., 2019)	GPUx4	-	-	115 Hours	79 Hours
RDOQ (Ours)	CPUx1	2 Mins	3 Mins	7 Mins	6 Mins

4 Related Work and Discussion

We discuss prior mixed-precision quantization works related to our work. ReLeQ (Elthakeb et al., 2018) proposed an end-to-end deep reinforcement learning (RL) framework to automate the process of discovering quantization bit width. Alternatively, HAQ (Wang et al., 2019) leveraged reinforcement learning to determine quantization bit width, and employed a hardware simulator to generate direct feedback signals to the RL agent. DNAS (Wu et al., 2019) proposed a differentiable neural architecture search framework to explore the hyperparameter space of quantization bit width. Differentiable Quantization (DQ) (Uhlich et al., 2019) learned quantizer parameters, including step size

Table 5: A comparison with prior mixed-precision quantization methods.

Approach	**Bit Allocation Scheme**	**Hardware -Aware**	**Optimization (Complexity)**
ReLeQ (Elthakeb et al., 2018)	Layer-Wise	No	Reinforcement Learning (High)
HAQ (Wang et al., 2019)	Layer-Wise	*Yes*	Reinforcement Learning (High)
DNAS (Wu et al., 2019)	Layer-Wise	No	Architecture Search (High)
DQ (Uhlich et al., 2019)	Layer-Wise	No	Training from Scratch (High)
HAWQ (Dong et al., 2019)	Layer-Wise	No	Retraining (High)
ALQ (Qu et al., 2020)	Layer-Wise	No	Retraining (High)
PTQ (Banner et al., 2018)	*Channel-Wise*	No	Analytic Solution (*Low*)
FracBits (Yang & Jin, 2020)	*Channel-Wise*	No	Training from Scratch (High)
AutoQ (Lou et al., 2020)	*Channel-Wise*	*Yes*	Reinforcement Learning (High)
DMBQ (Zhao et al., 2021)	*Channel-Wise*	No	Retraining (High)
RDOQ (Ours)	*Channel-Wise*	*Yes*	Lagrangian Formulation (*Low*)

and range, by training with straight-through gradients, and then inferred quantization bit width based on the learned step size and range. Hessian AWare Quantization (HAWQ) (Dong et al., 2019) introduced a second-order quantizatino method to select the quantization bit width of each layer, based on the layer's Hessian spectrum. Adaptive Loss-aware Quantization (ALQ) (Qu et al., 2020) directly minimized network loss w.r.t. quantized weights, and used network loss to decide quantization bit width. These methods all employed layer-wise bit allocation scheme.

Post Training Quantization (PTQ) (Banner et al., 2018) adopted channel-wise bit allocation to improve quantization precision, and provided an analytic solution for quantization bit width, assuming that parameters obey certain distributions. FracBits (Yang & Jin, 2020) generalized quantization bit width to an arbitrary real number to make it differentiable, and learned channel-wise bit allocation during training. Distribution-aware Multi-Bit Quantization (DMBQ) (Zhao et al., 2021) proposed loss-guided bit-width allocation strategy to adjust the bit-width of weights and activations channel-wisely. PTQ, FracBits, and DMBQ all did not take the impact of performance on hardware platforms into account. AutoQ (Lou et al., 2020) proposed a hierarchical deep reinforcement learning approach to find quantization bit width of channels and optimize hardware metrics (e.g., latency and energy) simultaneously. Different with AutoQ, our approach provides an alternative way to quickly explore the hyperparameter space of bit width with linear time complexity, and is able to find global optimal solution in a rate-distortion optimized manner. Table 5 illustrates the differences between the mixed-precision quantization approaches.

One prior work (Gao et al., 2019) interpreted neural network compression from a rate-distortion's perspective. The main focus of (Gao et al., 2019) was giving an upper bound analysis of compression and discussing the limitations. (Gao et al., 2019) did not give a way to search bit allocation, and there was no practical results provided.

5 Conclusion

Channel-wise bit allocation brings higher quantization precision and superior accuracy. Our approach provides an ultra-fast way to explore the hyperparameter space of channel bit widths with linear time complexity, using Lagrangian Formulation. The quantization of deep neural networks is formulated as a rate-distortion optimization problem, and the fast optimization method is proposed, by utilizing the additivity of output distortion. Moreover, we consider the impact on hardware platforms in the design of our quantization mechanism, and present a simple yet effective method to improve hardware performance. We restrict the bit rate of each layer to allow as many weights and activations as possible saved on-chip, and add hardware-aware constraints in our objective function to improve inference rate on target hardware platforms. The hardware-aware constraints can be incorporated into our objective function seamlessly, without incurring additional overhead for optimization. Extensive experiments show that our approach improves state-of-the-arts on four deep neural networks. Hardware simulation results demonstrate that our approach is able to accelerate deep learning inference considerably on two hardware platforms.

REFERENCES

Ron Banner, Yury Nahshan, Elad Hoffer, and Daniel Soudry. Post-training 4-bit quantization of convolution networks for rapid-deployment. *arXiv preprint arXiv:1810.05723*, 2018.

Y. Bengio, N. Leonard, and A. Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. In *arXiv:1308.3432*, 2013.

Yu-Hsin Chen, Tushar Krishna, Joel S Emer, and Vivienne Sze. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE Journal of Solid-State Circuits*, 52(1):127–138, 2016.

Jungwook Choi, Zhuo Wang, Swagath Venkataramani, and Kailash Gopalakrishnan Pierce I-Jen Chuang, Vijayalakshmi Srinivasan. Pact: Parameterized clipping activation for quantized neural networks. In *arXiv*, 2018.

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

Zhen Dong, Zhewei Yao, Amir Gholami, Michael W. Mahoney, and Kurt Keutzer. Hawq: Hessian aware quantization of neural networks with mixed-precision. In *arXiv*, 2019.

Ahmed T. Elthakeb, Prannoy Pilligundla, Fatemehsadat Mireshghallah, Amir Yazdanbakhsh, and Hadi Esmaeilzadeh. Releq: A reinforcement learning approach for deep quantization of neural networks. In *NeurIPS Workshop on ML for Systems*, 2018.

Steven K. Esser, J. L. McKinstry, D. Bablani, R. Appuswamy, and D. Modha. Learned step size quantization. In *ICLR*, 2020.

Weihao Gao, Yu-Han Liu, Chong Wang, and Sewoong Oh. Rate distortion for model compression: From theory to practice. In *International Conference on Machine Learning*, pp. 2102–2111. PMLR, 2019.

A. Gersho and R.M. Gray. Vector quantization and signal compression. In *Kluwer Academic Publishers*, 1991.

Ruihao Gong, Xianglong Liu, Shenghu Jiang, Tianxiang Li, P. Hu, J. Lin, Fengwei Yu, and J. Yan. Differentiable soft quantization: Bridging full-precision and low-bit neu- ral networks. In *ICCV*, 2019.

Song Han, Jeff Pool, John Tran, and William J Dally. Learning both weights and connections for efficient neural networks. *arXiv preprint arXiv:1506.02626*, 2015.

Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *ICLR*, 2016.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

Qing Jin, Linjie Yang, Zhenyu Liao, and Xiaoning Qian. Neural network quantization with scale-adjusted training. *BMVC*, 2020.

N. Jouppi et al. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ISCA '17, pp. 1–12, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-4892-8. doi: 10.1145/3079856.3080246. URL `http://doi.acm.org/10.1145/3079856.3080246`.

Sangil Jung, Changyong Son, Seohyung Lee, JinWoo Son, J. Han, Youngjun Kwak, Sung Ju Hwang, and Changkyu Choi. Learning to quantize deep networks by optimizing quantiza- tion intervals with task loss. In *CVPR*, 2019.

A. Krizhevsky, I. Sutskever, and G. Hinton. ImageNet classification with deep convolutional neural networks. In *NIPS*, 2012.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. In *Nature*, 2015.

Yuhang Li, Xin Dong, and Wei Wang. Additive powers-of-two quantization: An efficient non-uniform discretization for neural networks. In *ICLR*, 2020.

Qian Lou, Feng Guo, Minje Kim, Lantao Liu, and Lei Jiang. Autoq: Automated kernel-wise neural network quantizations. In *ICLR*, 2020.

Zhongnan Qu, Zimu Zhou, Yun Cheng, and Lothar Thiele. Adaptive loss-aware quantization for multi-bit networks. In *arXiv*, 2020.

Ananda Samajdar, Yuhao Zhu, Paul N. Whatmough, Matthew Mattina, and Tushar Krishna. Scale-sim: Systolic CNN accelerator. *CoRR*, abs/1811.02883, 2018. URL `http://arxiv.org/abs/1811.02883`.

Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *arXiv*, 2018.

Y. Shoham and A. Gersho. Efficient bit allocation for an arbitrary set of quantizers (speech coding. In *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 1988.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

David S Taubman and Michael W Marcellin. Jpeg2000 image compression fundamentals, standards and practice. 2002.

Stefan Uhlich, Lukas Mauch, Fabien Cardinaux, Kazuki Yoshiyama, Javier Alonso Garcia, Stephen Tiedemann, Thomas Kemp, and Akira Nakamura. Mixed precision dnns: All you need is a good parametrization. *arXiv preprint arXiv:1905.11452*, 2019.

Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. HAQ: Hardware-aware automated quantization with mixed precision. In *CVPR*, 2019.

Bichen Wu, Yanghan Wang, Peizhao Zhang, Yuandong Tian, Peter Vajda, and Kurt Keutzer. Mixed precision quantization of convnets via differentiable neural architecture search. In *ICLR*, 2019.

Linjie Yang and Qing Jin. Fracbits: Mixed precision quantization via fractional bit-widths. *arXiv preprint arXiv:2007.02017*, 2020.

D. Zhang, J. Yang, D. Ye, and G. Hua. Lq-nets: learned quantization for highly accurate and compact deep neural networks. In *ECCV*, 2018.

Sijie Zhao, Tao Yue, and Xuemei Hu. Distribution-aware adaptive multi-bit quantization. In *CVPR*, 2021.

Xiandong Zhao, Ying Wang, Xuyi Cai, Cheng Liu, and Lei Zhang. Linear symmetric quantization of neural networks for low-precision integer hardware. *ICLR*, 2020.

Wang Zhe, Jie Lin, Vijay Chandrasekhar, and Bernd Girod. Optimizing the bit allocation for compression of weights and activations of deep neural networks. In *ICIP*, 2019.

S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. In *arXiv preprint arXiv:1606.06160*, 2016.

Y. Zhou, Seyed-Mohsen Moosavi-Dezfooli, Ngai-Man Cheung, and P. Frossard. Adaptive quantization for deep neural network. In *AAAI*, 2018.

Bohan Zhuang, Chunhua Shen, Mingkui Tan, Lingqiao Liu, and Ian Reid. Towards effective low-bitwidth convolutional neural networks. In *cvpr*, 2018.

Bohan Zhuang, Lingqiao Liu, Mingkui Tan, Chunhua Shen, and Ian Reid. Training quantized neural networks with a full-precision auxiliary module. In *CVPR*, 2020.

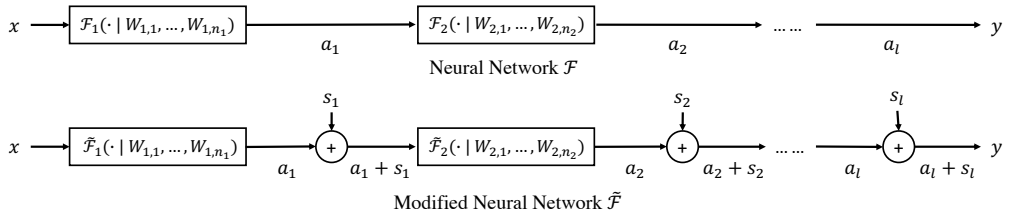

Figure 5: Examples of a neural network $\mathcal{F}$ and a modified neural network $\widetilde{\mathcal{F}}$.

A Appendix

In this appendix, we provide the analysis for the additivity property of output distortion. We show a mathematical derivation for the additivity property by linearizing the output distortion using Taylor series expansion. We then illustrate the implementation details of the optimization method and the pseudo codes, and discuss the intermediary results of our approach.

A.1 Additivity of Output Distortion

The output distortion δ, caused by quantizing all weight channels and activations, equals the sum of all output distortions due to the quantization of each individual weight channel and activation layer

$$\delta = \sum_{i=1}^{l} \sum_{j=1}^{n_i} \delta_{i,j}^{w} + \sum_{i=1}^{l} \delta_i^a \tag{9}$$

if the neural network is continuously differentiable in every layer, the quantization errors can be considered as small deviations distributed with zero mean.

We first define the main notations. Let

$$\mathcal{F}(W_{1,1}, ..., W_{1,n_1}, ..., W_{l,1}, ..., W_{l,n_l})$$

denote a neural network and

$$\widetilde{\mathcal{F}}(W_{1,1}, ..., W_{1,n_1}, ..., W_{l,1}, ..., W_{l,n_l}, s_1, ..., s_l)$$

denote a modified neural networks of $\mathcal{F}$ where an element-wise add layer with parameter s_i is followed for each activation a_i (see Fig. 5). Based on this definition, we have

$$\mathcal{F}(W_{1,1}, ..., W_{l,n_l}) = \widetilde{\mathcal{F}}(W_{1,1}, ..., W_{l,n_l}, 0, ..., 0) \tag{10}$$

Define two variables X_0 and ΔX, where

$$X_0 = \big(W_{1,1}, ..., W_{l,n_l}, 0, ..., 0\big)$$
$$\Delta X = \big(\Delta W_{1,1}, ..., \Delta W_{l,n_l}, \Delta s_1, ..., \Delta s_l\big)$$

Assume that the quantization error can be considered as small deviation. We apply the Taylor series expansion up to first order term on $\widetilde{\mathcal{F}}$ at X_0,

$$\begin{aligned} \widetilde{\mathcal{F}}(X_0 + \Delta X) - \widetilde{\mathcal{F}}(X_0) = &\sum_{i,j} \frac{\partial \widetilde{\mathcal{F}}}{\partial W_{i,j}} \cdot \Delta W_{i,j} \\ &+ \sum_{i} \frac{\partial \widetilde{\mathcal{F}}}{\partial s_i} \cdot \Delta s_i. \end{aligned} \tag{11}$$

Then $\|\widetilde{\mathcal{F}}(X_0 + \Delta X) - \widetilde{\mathcal{F}}(X_0)\|^2$ can be written as

$$\begin{aligned} &\Big(\sum_{i,j} \Delta W_{i,j}{}^{\top} \cdot \frac{\partial \widetilde{\mathcal{F}}}{\partial W_{i,j}}^{\top} + \sum_{i} \Delta s_i{}^{\top} \cdot \frac{\partial \widetilde{\mathcal{F}}}{\partial s_i}^{\top} \Big) \\ &\cdot \Big(\sum_{i,j} \frac{\partial \widetilde{\mathcal{F}}}{\partial W_{i,j}} \cdot \Delta W_{i,j} + \sum_{i} \frac{\partial \widetilde{\mathcal{F}}}{\partial s_i} \cdot \Delta s_i \Big) \end{aligned} \tag{12}$$

Because quantization errors in different layers are independently distributed with zero mean, the cross terms of (4) disappear when taking the expectation. That is:

$$\begin{aligned} &E(\Delta W_{i,j}{}^{\top} \cdot \frac{\partial \widetilde{\mathcal{F}}}{\partial W_{i,j}}^{\top} \cdot \frac{\partial \widetilde{\mathcal{F}}}{\partial W_{k,t}} \cdot \Delta W_{k,t}) \\ =&E(\Delta W_{i,j}{}^{\top}) \cdot \frac{\partial \widetilde{\mathcal{F}}}{\partial W_{i,j}}^{\top} \cdot \frac{\partial \widetilde{\mathcal{F}}}{\partial W_{k,t}} \cdot E(\Delta W_{k,t}) = 0 \end{aligned} \tag{13}$$

as is the case also for the cross products between $W_{i,j}$ and n_i (all i, j), and s_i and s_j ($i \neq j$). Then, we can obtain

$$\begin{aligned} &E(\|\widetilde{\mathcal{F}}(X_0 + \Delta X) - \widetilde{\mathcal{F}}(X_0)\|^2) \\ =&\sum_{i,j} E\Big(\|\frac{\partial \widetilde{\mathcal{F}}}{\partial W_{i,j}} \cdot \Delta W_{i,j}\|^2\Big) + \sum_{i} E\Big(\|\frac{\partial \widetilde{\mathcal{F}}}{\partial s_i} \cdot \Delta s_i\|^2\Big) \end{aligned} \tag{14}$$

Eq. (14) is the result we want because, again, according to the Taylor series expansion up to first order terms,

$$\begin{aligned} \frac{\partial \widetilde{\mathcal{F}}}{\partial W_{i,j}} \cdot \Delta W_{i,j} = &\widetilde{\mathcal{F}}(..., W_{i,j} + \Delta W_{i,j}, ..., W_{l,n_l}, 0, ...) \\ &- \widetilde{\mathcal{F}}(..., W_{i,j}, ..., W_{l,n_l}, 0, ...) \end{aligned} \tag{15}$$

$$\begin{aligned} \frac{\partial \widetilde{\mathcal{F}}}{\partial s_i} \cdot \Delta s_i = &\widetilde{\mathcal{F}}(W_{1,1}, ..., W_{l,n_l}, 0, ..., \Delta s_i, ...) \\ &- \widetilde{\mathcal{F}}(W_{1,1}, ..., W_{l,n_l}, 0, ..., 0, ...) \end{aligned} \tag{16}$$

After dividing both sides of (14) by the dimensionality of the output vector of the neural network, the left side becomes δ and the right side becomes the sum of all output distortion due to the quantization of each individual weight channel and activation layer.

A.2 Implementation Details of Optimization

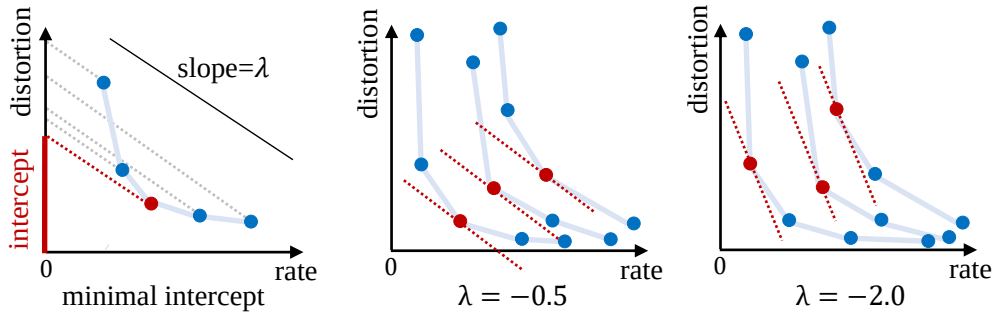

Figure 6: Examples of the optimization method. In the left figure, the red point has minimal intercept on Y-Axis and it is selected. The middle and right figures show the selected points on three curves when $\lambda = -0.5$ and $\lambda = 2.0$, respectively.

Algorithm 1 Generate rate-distortion curves of channels.

Input: Neural Network $\mathcal{F}$; Images $\mathbf{I}$.
Output: Rate-distortion curves of channels: G_1, G_2, ..., G_K.
  Compute original output vector $\mathbf{Y} = \mathcal{F}(\mathbf{I})$.
  for each channel $\mathbf{C}_i$ of $\mathcal{F}$ **do**
    for bit-width b ranging from 0 bit to 16 bits **do**
      Calculate quantization step size Δ.
      Quantize $\mathbf{C}_i$ with Δ and b: $q(\mathbf{C}_i)$.
      Compute the bit rate R of $q(\mathbf{C}_i)$, and the modified output vector $\widehat{\mathbf{Y}}$.
      Compute output distortion $d = \|\mathbf{Y} - \widehat{\mathbf{Y}}\|^2.mean()$.
      Collect point $P = (R, d)$: $G_i = G_i \cup P$.
    end for
  end for

Algorithm 2 Optimize the bit allocation of channels according to Equation 18.

Input: Rate-distortion curves of the channels: G_1, G_2, ..., G_K; Slope value λ.
Output: Solution of optimal bit allocation S; Bit rate R.
  Initialization: $S = \emptyset$, $R = 0$.
  for each rate-distortion curve G_i **do**
    Set minimal intercept to infinity, $Y_{min\_intercept} = \infty$; Set $id = -1$.
    for each point P_j on G_i **do**
      $x_0 = P_j \rightarrow x$; $y_0 = P_j \rightarrow y$; $Y_{intercept} = y_0 - \lambda \cdot x_0$.
      if $Y_{intercept} < Y_{min\_intercept}$ **then**
        Update $Y_{min\_intercept} = Y_{intercept}$; Update $id = j$.
      end if
    end for
    Collect the selected point: $S = S \cup \{P_{id}\}$, $R = R + P_{id} \rightarrow x$.
  end for

We formulate the quantization of weight channels and layer activations as a rate-distortion optimization problem, as illustrated in Section 2

$$\min \delta = \|\boldsymbol{y} - \widehat{\boldsymbol{y}}\|_2^2 \;\; s.t. \sum_{i=1}^{l} \sum_{j=1}^{n_i} R_{i,j}^w + \sum_{i=1}^{l} R_i^a \leq r, \tag{17}$$

We utilize the addtivity of the output distortion and apply the classical Lagrangian formulation to solve 17. The optimal condition is presented as

$$\frac{\partial \delta_{1,1}^w}{\partial R_{1,1}^w} = ... = \frac{\partial \delta_{1,n_1}^w}{\partial R_{1,n_1}^w} = \frac{\partial \delta_1^a}{\partial R_1^a} = ... = \frac{\partial \delta_{l,1}^w}{\partial R_{l,1}^w} = ... = \frac{\partial \delta_{l,n_l}^w}{\partial R_{l,n_l}^w} = \frac{\partial \delta_l^a}{\partial R_l^a} = \lambda \tag{18}$$

The above equation expresses the optimal condition that the slopes of all output distortion versus bit rate functions should be equal. Based on this equation, the optimziation problem can be solved by enumerating λ and selecting the point with slope equal to λ on each rate-distortion curve.

Specifically, we first generate the rate-distortion curves for each channel and activation layer. In our case, the rate-distortion curves are discrete points. For example, if the range of bit width is from 1-bit to 8-bits, then the rate-distortion curve is a discrete curve with 8 points. We enumerate λ and select the point on each curve with slope equal to λ. We may enumerate many values of λ to find the best solution with minimal output distortion under the size constraint. Assume that we have N curves and M points in each curve. The time complexity to find optimum bit allocation is $O(K \cdot M \cdot N)$, where K is the total number of slope λ to be evaluated.

Figure 6 illustrates an example of the optimization method. The blue curves in Figure 6 denote the rate-distortion curves. We totally show 3 rate-distortion curves in the example. The optimization starts from enumerating the value λ. Given a λ, we then find the point on each curve. Specifically, we compute the intercepts on Y-axis for the lines, with slope λ, passing the points on the curve. The point with minimal intercept on Y-axis will be selected. Note that we only choose the point with

minimal Y-axis intercept for each curve. The X-axis value of the selected point corresponds to the bit rate of the related channel or activation layer.

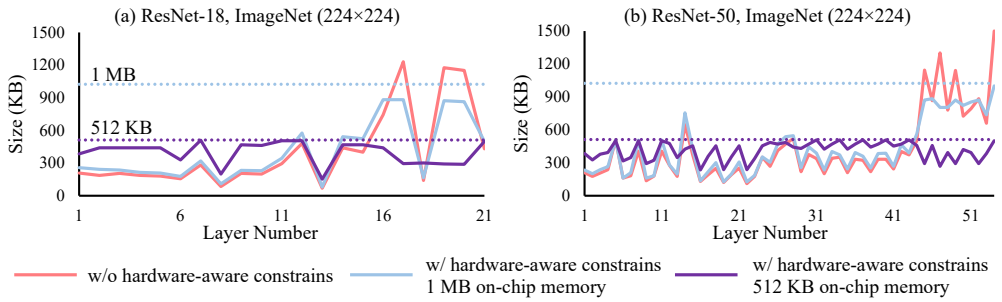

Figure 7: Distributions of bit rate across layers given different on-chip memory constraints.

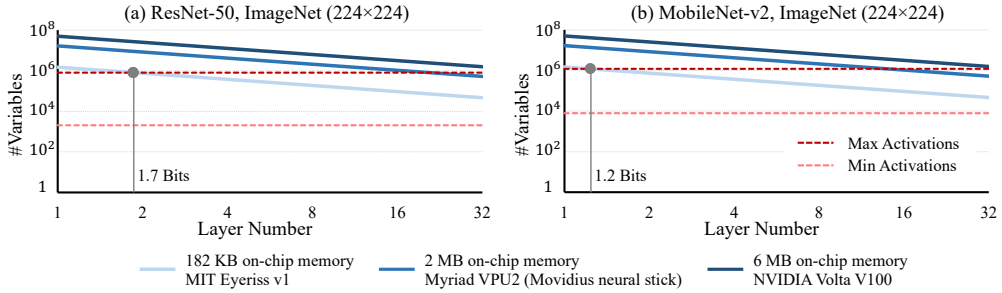

Figure 8: Number of variables that the on-chip memory of specific hardware platforms can accommodate, with different bit width per variable. The minimum and maximum numbers of activations that a single layer can have are highlighted.

A.3 Distributions of bit rate across layers

Fig. 7 illustrates an example of the distribution for the bit rate under different hardware-aware constraints. Intuitively, our approach assigns lower bit width to layers with lots of variables and higher bit width to layers with just a few variables, to meet constraints by balancing the size across layers. Fig. 8 illustrates the number of variables that hardware platforms can accommodate, with different bit width. We can see that on both ResNet-50 and MobileNet-v2, some layers can have more than 1 million activations, and the bit width assigned to these layers have to be less than two bits if on-chip memory capacity is 182 KB, which is the case of MIT Eyeriss.

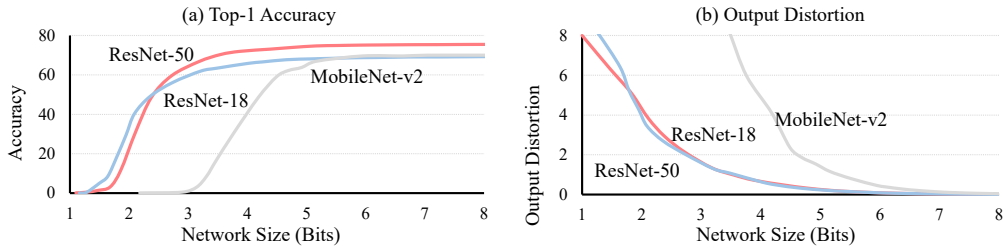

Figure 9: Top-1 accuracy and output distortion under different network size on ResNet-18, ResNet-50, and MobileNet-v2.

A.4 RELATIONSHIPS BETWEEN OUTPUT DISTORTION AND ACCURACY

Fig. 9 illustrate the accuracy and output distortion under different network size on three deep neural networks. As one can see, accuracy is highly related to output distortion. Large output distortion also indicates huge accuracy loss. By minimizing the output distortion, our approach is thus able to well maintain the accuracy at very low bit width.

Table 6: Architecture parameters of considered deep learning hardware platforms.

Paramter	MIT Eyeriss	Google TPU
On-chip memory	181.5 KBytes	28 MBytes
Off-chip memory-access bandwidth (BW)	1 GByte/sec	13 GBytes/sec
Computing performance (Perf)	34 GOPs	96 TOPs

A.5 CONFIGURATIONS ON HARDWARE PLATFORMS

We have explored the impact our approach has on the inference rate for different hardware architectures inspired by current embedded and high-performance DNN-targeted hardware accelerators (i.e., MIT Eyeriss Chen et al. (2016) and Google TPU Jouppi et al. (2017), summarized in Table 6)—such platforms have shown superior inference rate compared to GPUs Jouppi et al. (2017). We assume that accessing off-chip memory can be done in parallel while computing is performed (i.e., weights can be prefetched from off-chip memory).