HAWQ-V3: Dyadic Neural Network Quantization

Zhewei Yao [*1] **Zhen Dong** [*1] **Zhangcheng Zheng** [*1] **Amir Gholami** [*1] **Jiali Yu** [2 3] **Eric Tan** [1] **Leyuan Wang** [2]
Qijing Huang [1] **Yida Wang** [2] **Michael W. Mahoney** [1] **Kurt Keutzer** [1]

Abstract

Current low-precision quantization algorithms often have the hidden cost of conversion back and forth from floating point to quantized integer values. This hidden cost limits the latency improvement realized by quantizing Neural Networks. To address this, we present HAWQ-V3, a novel mixed-precision integer-only quantization framework. The contributions of HAWQ-V3 are the following: (i) An integer-only inference where the entire computational graph is performed only with integer multiplication, addition, and bit shifting, without any floating point operations or even integer division; (ii) A novel hardware-aware mixed-precision quantization method where the bit-precision is calculated by solving an integer linear programming problem that balances the trade-off between model perturbation and other constraints, e.g., memory footprint and latency; (iii) Direct hardware deployment and open source contribution for 4-bit uniform/mixed-precision quantization in TVM, achieving an average speed up of $1.45\times$ for uniform 4-bit, as compared to uniform 8-bit for ResNet50 on T4 GPUs; and (iv) extensive evaluation of the proposed methods on ResNet18/50 and InceptionV3, for various model compression levels with/without mixed precision. For ResNet50, our INT8 quantization achieves an accuracy of 77.58%, which is 2.68% higher than prior integer-only work, and our mixed-precision INT4/8 quantization can reduce INT8 latency by 23% and still achieve 76.73% accuracy. Our framework and the TVM implementation have been open sourced (HAWQ, 2020).

[*]Equal contribution [1]University of California, Berkeley [2]Amazon [3]Shanghai Jiao Tong University. Correspondence to: Zhewei Yao <zheweiy@berkeley.edu>, Amir Gholami <amirgh@berkeley.edu>.

Proceedings of the 38^{th} International Conference on Machine Learning, PMLR 139, 2021.

1. Introduction

An important step toward realizing pervasive deep learning is enabling real-time inference, both at the edge and in the cloud, with low energy consumption and state-of-the-art model accuracy. This will have a significant impact on applications such as real-time intelligent healthcare monitoring, autonomous driving, audio analytics, and speech recognition. Over the past decade, we have observed significant improvements in the accuracy of Neural Networks (NNs) for various tasks. However, the state-of-the-art models are often prohibitively large and too compute-heavy to be deployed for real-time use. A promising approach to address this is through quantization (Gray & Neuhoff, 1998; Han et al., 2016), where low-precision quantized integer values are used to express the model parameters and feature maps. That can help reduce the model footprint, and improve inference speed and energy consumption.

However, existing quantization algorithms often use *simulated quantization*, where the parameters are stored with quantization, but are cast to floating point for inference. As a result, all or part of the inference operations (e.g. convolution, matrix operations, batch norm layers, residual connections) are performed using floating point precision. This of course limits the speed up as we cannot utilize low precision logic. To address this, we build upon existing integer-only quantization methods (Jacob et al., 2018), and propose systematic methods to extend them to low and mixed-precision quantization. In particular, we make the following contributions:

- We develop HAWQ-V3, a mixed-precision integer-only quantization framework with integer-only multiplication, addition, and bit shifting with static quantization. Importantly, no floating point and no integer division calculation is performed in the entire inference. This includes the batch norm layers and residual connections, which are typically kept at floating point precision in prior integer-only quantization work (Dong et al., 2019). While keeping these operations in floating point helps accuracy, this is not allowed for integer-only hardware. We show that ignoring this and attempting to deploy a model that uses floating point residual on integer-only hardware can lead to more

than 90% mismatch (Figure G.1). HAWQ-V3 completely avoids this by using a novel approach to perform residual connections in pure integer-only arithmetic. See Section 3.3 and Appendix G for details.

- We propose a novel hardware-aware mixed-precision quantization formulation that uses an Integer Linear Programming (ILP) problem to find the best bit-precision setting. The ILP solver minimizes the model perturbation while observing application-specific constraints on model size, latency, and total bit operations. Compared to the contemporary work of (Hubara et al., 2020), our approach is hardware-aware and uses direct hardware measurement to find a bit precision setting that has the optimal balance between latency and accuracy. See Section 3.4 and Appendix I for details.
- To verify the feasibility of our approach, we deploy the quantized integer-only models using Apache TVM (Chen et al., 2018) for INT8, INT4, and mixed-precision settings. To the best of our knowledge, our framework is the first that adds INT4 support to TVM. By profiling the latency of different layers, we show that we can achieve an average of $1.47\times$ speed up with INT4, as compared to INT8 on a T4 GPU for ResNet50. See Section 3.5 and Table 2 for more details.
- We extensively test HAWQ-V3 on a wide range of workloads, including ResNet18, ResNet50, and InceptionV3, and show that we can achieve a substantial performance improvement, as compared to the prior state-of-the-art. For instance, we achieve an accuracy of 78.50% with INT8 quantization, which is more than 4% higher than prior integer-only work for InceptionV3. Furthermore, we show that mixed-precision INT4/8 quantization can be used to achieve higher speed up as compared to INT8 inference with smaller impact on accuracy as compared to INT4 quantization. For example, for ResNet50 we can speedup latency by 23% as compared to INT8 and still achieve 76.73% accuracy. See Section 4 and Table 1, 2 for more details.

2. Related Work

There have been significant efforts recently to improve the trade-off between accuracy and efficiency of NN models. These can be broadly categorized as follows: (i) Designing new NN architectures (Iandola et al., 2016; Sandler et al., 2018; Tan & Le, 2019); (ii) Co-designing NN architecture and hardware together (Gholami et al., 2018; Han & Dally, 2017; Howard et al., 2019; Wu et al., 2019); (iii) Pruning redundant filters (Han et al., 2015; LeCun et al., 1990; Li et al., 2016; Mao et al., 2017; Molchanov et al., 2016; Yang et al., 2017); (iv) knowledge distillation (Hinton et al., 2014; Mishra & Marr, 2017; Polino et al., 2018; Yin et al., 2020); and (v) using quantization (reduced precision). Here, we provide a more detailed overview of this related work.

Quantization. A common solution is to compress NN models with quantization (Asanovic & Morgan, 1991; Bhalgat et al., 2020; Chin et al., 2020; Dong et al., 2019; Hubara et al., 2016; Jacob et al., 2018; Kim & Kim, 2021; Park et al., 2018a; Rastegari et al., 2016; Sharma et al., 2018; Song et al., 2020; Zhang et al., 2018; Zhou et al., 2017a; 2016), where low-bit precision is used for weights/activations. Quantization reduces model size without changing the original network architecture, and it could potentially permit the use of low-precision matrix multiplication or convolution.

While the gains on speed/power increase for low-precision quantization, low-precision quantization suffers from accuracy degradation. To address this, recent work uses non-uniform quantizers (Zhang et al., 2018), channel-wise quantization (Krishnamoorthi, 2018), and progressive quantization-aware fine-tuning (Zhou et al., 2017a). Other works try to include periodic regularization to assist quantization (Elthakeb et al., 2020; Naumov et al., 2018), apply post training quantization (Banner et al., 2019; Cai et al., 2020; Hubara et al., 2020), or improve accuracy by changing the channel counts accordingly for different layers (Chin et al., 2020). Despite these advances, performing uniform ultra low-bit quantization still results in a significant accuracy degradation. A promising direction is to use mixed-precision quantization (Dong et al., 2019; Shen et al., 2020; Wang et al., 2019; Zhou et al., 2017b), where some layers are kept at higher precision, while others are kept at a lower precision. However, a challenge with this approach is finding the right the mixed-precision setting for the different layers. A brute force approach is not feasible since the search space is exponentially large in the number of layers.

HAQ (Wang et al., 2019) proposes to search this space by applying a Reinforcement Learning algorithm, while (Wu et al., 2018) uses a Differentiable Neural Architecture Search. However, these searching methods require large computational resources, and their performance is very sensitive to hyper-parameters and even initialization. To address these issues, HAWQ (Dong et al., 2019; 2020) introduces an automatic way to find good mixed-precision settings based on the sensitivity obtained using the Hessian spectrum. However, the Pareto frontier method in (Dong et al., 2020) is not flexible enough to satisfy simultaneously different requirements on hardware. To address this, we propose here an ILP solution that can generate mixed-precision settings with various constraints (such as model size, BOPS, and latency), and which can be solved within seconds on commodity hardware. The contemporary work of (Hubara et al., 2020) also proposes to use an ILP. However, their approach is not hardware aware, and their approach uses FP32 casting.

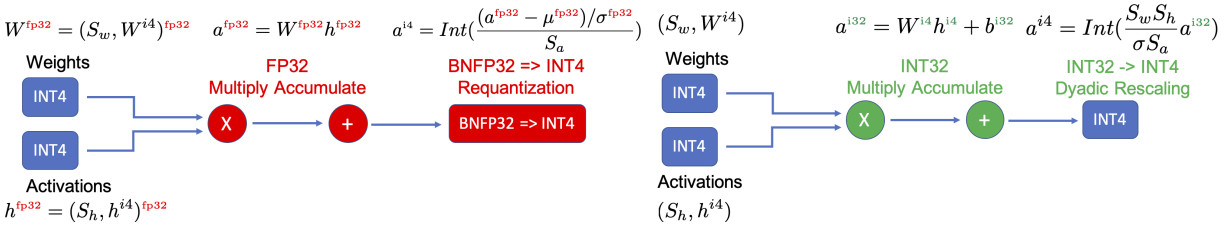

Figure 1. Illustration of fake vs true quantization for convolution and batch normalization folding. For simplicity, we ignore the affine coefficient of BN. (Left) In the simulated quantization (aka fake quantization approach), weights and activations are simulated as integers with floating point representation, and all the multiplication and accumulation happen in FP32 precision. Furthermore, the BN parameters (i.e. μ and σ) are stored and computed using FP32 precision. This is undesirable but can significantly help accuracy since BN parameters are sensitive to quantization. However, with this approach, one cannot benefit from low-precision ALUs. (Right) An illustration of the integer-only pipeline with dyadic arithmetic for convolution and BN folding. The standard deviation (σ) in BN is merged into the quantization scale of the weights, and the mean is quantized to INT32 and merged as a bias into the weights (denoted by b^{i32}) Note that with this approach, all the weights and activations are stored in integer format, and all the multiplications are performed with INT4 and accumulated in INT32 precision. Finally, the accumulated result is requantized to INT4 with dyadic scaling (denoted by $\frac{S_w S_h}{\sigma S_a}$). Importantly, no floating point or even integer division is performed. See Section 3.2 and Appendix D for more details.

Another issue is that the quantized weights and activations need to be converted to floating point precision during inference, as shown in Figure 1. This high-precision casting can have high overhead and limits inference speed, especially for hardware with limited on-chip memory. Using FP32 ALUs also requires a larger die area in the chip, further limiting the peak computational capacity of the hardware. The work of (Jacob et al., 2018) addresses this casting problem by using integer-only quantization in INT8 precision. However, there are several shortcomings associated with their approach (which are addressed in HAWQ-V3). First, (Jacob et al., 2018) does not support low-precision or mixed-precision quantization. We show that this is useful in practice, as it can improve the inference speed by up to 50% with a small impact on accuracy. Second, both (Jacob et al., 2018) and HAWQ are hardware agnostic and do not co-design/adapt the quantization for the target hardware. In contrast, the ILP approach in HAWQ-V3 is hardware aware, and it directly takes this into account when determining mixed-precision bit setting. Third, as we discuss in Section 3.2, the approach used in (Jacob et al., 2018) leads to sub-optimal accuracy for INT8 quantization, while our approach can achieve up to 5% higher accuracy for INT8 inference. Finally, to address the absence of low-precision support in previous works (Dong et al., 2019; Jacob et al., 2018), we extend TVM to support INT4 and mixed-precision quantization, and we validate our results by directly running the quantized model with low bit-width on the hardware. See Appendix A for the discussion of different deployment frameworks.

3. Methodology

Assume that the NN has L layers with learnable parameters, denoted as $\{W_1, W_2, ..., W_L\}$, with θ denoting the combination of all such parameters. For a supervised setting, the goal is to optimize the following empirical risk minimization loss function:

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_{i=1}^{N} l(x_i, y_i; \theta), \tag{1}$$

where (x, y) is the input data and the corresponding label, $l(x, y; \theta)$ is the loss function (e.g., MSE or Cross Entropy loss), and N is the total number of data points. We assume that we have the trained model parameters θ given in floating point precision. Our goal is to quantize the model with the optimal trade-offs among memory footprint, speed, and accuracy. Below, we first define quantization and then present HAWQ-V3.

Uniform Quantization. Quantization restricts NN weights/activations to a finite set of values as follows:

$$Q(r) = \text{Int}(r/S) - Z, \tag{2}$$

where Q is the quantization operator, r is a real valued number (activation or a weight), S is a real valued scaling factor, and Z is the zero point, chosen such that the 0 value would exactly map to quantized values. Furthermore, Int maps a floating point value to an integer value through a rounding operation (e.g., round to nearest and truncation).

This formulation for Q corresponds to uniform quantization. However, some work in the literature has also explored non-uniform quantization (Park et al., 2018b; Wang et al., 2019; Zhang et al., 2018). Although non-uniform quantization may achieve higher accuracy for a fixed bit-width, such approaches are typically difficult to deploy on hardware to reduce latency.[1] As such, for HAWQ-V3, we only focus on uniform quantization. Meanwhile, for HAWQ-V3, we use (i) symmetric quantization for weights and asymmetric

[1] However, they can reduce total model footprint.

quantization for activations; and (ii) static quantization for all the scaling factors S. Meanwhile, we apply channel-wise quantization for different convolutional output channels. Please see Appendix B for more details.

3.1. Quantized Matrix Multiplication and Convolution

Consider a layer with hidden activation denoted as h and weight tensor denoted as W, followed by ReLU activation. First, h and W are quantized to $S_h q_h$ and $S_w q_w$, where S_h and S_w are the real valued quantization scales, q_h and q_W are the corresponding quantized integer values. The output result, denoted with a, can be computed as follows:

$$a = S_w S_h (q_w * q_h), \tag{3}$$

where $q_w * q_h$ is the matrix multiplication (or convolution) calculated with integer in low precision (e.g., INT4) and accumulated in INT32 precision. This result is then requantized and sent to the next layer as follows:

$$q_a = \text{Int}\left(\frac{a}{S_a}\right) = \text{Int}\left(\frac{S_w S_h}{S_a}(q_w * q_h)\right), \tag{4}$$

where S_a is the pre-calculated scale factor for the output activation.

In HAWQ-V3, the $q_w * q_h$ operation is performed with low-precision integer-only multiplication and INT32 accumulation, and the final INT32 result is quantized by scaling it with $S_w S_h / S_a$. The latter is a floating point scaling that needs to be multiplied with the accumulated result (in INT32 precision). A naive implementation requires floating point multiplication for this stage. However, this can be avoided by enforcing the scaling to be a dyadic number. Dyadic numbers are rational numbers with the format of $b/2^c$, where $b,\ c$ are two integer numbers. As such, a dyadic scaling in Eq. 4 can be efficiently performed using INT32 integer multiplication and bit shifting. Given a specific $S_w S_h / S_a$, we use DN (representing Dyadic Number) to denote the function that can calculate the corresponding b and c:

$$b/2^c = \text{DN}\left(S_w S_h / S_a\right). \tag{5}$$

An advantage of using dyadic numbers besides avoiding floating point arithmetic, is that it removes the need to support division (which typically has an order of magnitude higher latency than multiplication) in the hardware. This approach is used for INT8 quantization in (Jacob et al., 2018), and we enforce all the rescaling to be dyadic for low-precision and mixed-precision quantization as well.

3.2. Batch Normalization

Batch normalization (BN) is an important component of most NN architectures, especially for computer vision applications. BN performs the following operation to an input activation a:

$$\text{BN}(a) = \beta \frac{a - \mu_B}{\sigma_B} + \gamma \tag{6}$$

where μ_B and σ_B are the mean and standard deviation of a, and $\beta,\ \gamma$ are trainable parameters. During inference, these parameters (both statistics and trainable parameters) are fixed, and therefore the BN operations could be fused with the convolution (see Appendix D). However, an important problem is that quantizing the BN parameters often leads to significant accuracy degradation. As such, many prior quantization methods keep BN parameters in FP32 precision (e.g., (Cai et al., 2020; Chin et al., 2020; Choi et al., 2018; Dong et al., 2020; Park et al., 2018b; Zhang et al., 2018), just to name a few). This makes such approaches not suitable for integer-only hardware. While using such techniques help accuracy, HAWQ-V3 completely avoids that. We fuse the BN parameters with the convolution and quantized them with integer-only approach (Please see Figure 1 where we compare simulated qauntization and HAWQ-V3 for BN and convolution.).

Another important point to discuss here is that we found the BN folding used in (Jacob et al., 2018) to be sub-optimal. In their approach BN and CONV layers are fused together while BN running statistics are still kept updating. This actually requires computing each convolution layer twice, once without BN and then with BN (as illustrated in (Jacob et al., 2018, Figure C8)). However, we found that this is unnecessary and degrades the accuracy. Instead, in HAWQ-V3, we follow a simpler approach where we first keep the Conv and BN layer unfolded, and allow the BN statistics to update. After several epochs, we then freeze the running statistics in the BN layer and fold the CONV and BN layers (please see Appendix D for details). As we will show in Section 4, this approach results in better accuracy as compared to (Jacob et al., 2018).

3.3. Residual Connection

Residual connection (He et al., 2016) is another important component in many NN architectures. Similar to BN, quantizing the residual connections can lead to accuracy degradation, and as such, some prior quantization works perform the operation in FP32 precision (Choi et al., 2018; Wang et al., 2019; Zhang et al., 2018). There is a common misunderstanding that this may not be a big problem. However, this actually leads to complete loss of signal, especially for low precision quantization. The main reason for this is that quantization is not a linear operation, that is $Q(a + b) \neq Q(a) + Q(b)$ (a, b are floating point numbers). As such, performing the accumulation in FP32 and then quantizing is not the same as accumulating quantized values. Therefore, it is not possible to deploy quantization methods that keep residual connection in FP32 in integer-only hardware (we provide more detailed discussion of this

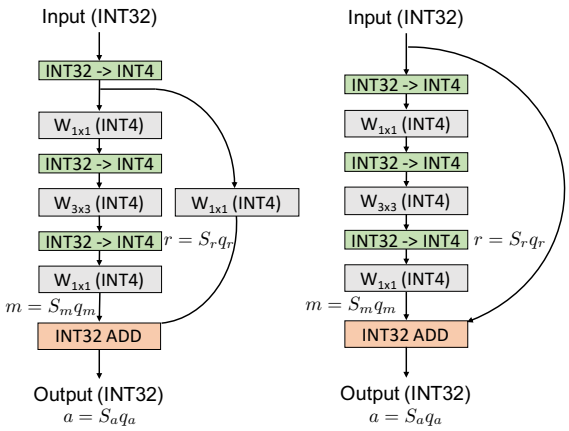

Figure 2. Illustration of HAWQ-V3 for a residual block with and without transition layer. Input feature map is given in INT32 precision, which is requantized to INT4 precision (green boxes) before any convolution layer (gray boxes). The BN layer is folded into the convolution. The residual addition is performed in INT32 precision, and the final accumulated result is re-scaled and sent to the next layer. For blocks with a transition layer, we only quantize the input once to INT4 and we use the same result for both 1×1 convolutions.

in Appendix F and also quantify the resulting error which can be more than 90%).

We avoid this in HAWQ-V3, and use INT32 for the residual branch. We perform the following steps to ensure that the addition operation can happen with dyadic arithmetic. Let us denote the activation passing through the residual connection as $r = S_r q_r$.[2] Furthermore, let us denote the activation of the main branch before residual addition as $m = S_m q_m$, and the final output after residual accumulation by $a = S_a q_a$. Then we will have:

$$q_a = \mathrm{DN}\left(S_m / S_a\right) q_m + \mathrm{DN}\left(S_r / S_a\right) q_r. \tag{7}$$

Note that with this approach, we only need to perform a dyadic scaling of q_m and add the result with the dyadically scaled q_r. All of these operations can happen with integer-only arithmetic. Also we should note that in our approach all the scales are statically known. These steps are schematically illustrated in Figure 2 for a residual connection with/without downsampling. Similar approach is performed for concatenation layer as well (see Appendix E).

3.4. Mixed Precision and Integer Linear Programming

Uniformly quantizing all the layers to low bit-width (e.g. INT4) could lead to significant accuracy degradation. However, it is possible to benefit from low-precision quantization by keeping a subset of sensitive layers at high precision (Dong et al., 2019). The basic idea is to keep sensitive layers at higher precision and insensitive layers at lower precision. An important component of HAWQ-V3 is that we directly consider hardware-specific metrics such as latency, to select the bit-precision configuration. This is important since a layer's latency does not necessarily halve when quantized from INT8 to INT4 precision. In fact, as we discuss in Section 4, there are specific layer configurations that do not gain any speed up when quantized to low precision, and some that superlinearly benefit from quantization.[3] As such, quantizing the former will not lead to any latency improvement, and will only hurt accuracy. Therefore, it is better to keep such layers at high precision, even if they have low sensitivity. These trade-offs between accuracy and latency should be taken into consideration when quantizing them to low precision. Importantly, these trade-offs are hardware-specific as latency in general does not correlate with the model size and/or FLOPS. However, we can consider this by directly measuring the latency of executing a layer in quantized precision on the target hardware platform. This trade-off is schematically shown in Figure 3 (and later quantified in Figure I.1). We can use an Integer Linear Programming (ILP) problem to formalize the problem definition of finding the bit-precision setting that has optimal trade-off as described next.

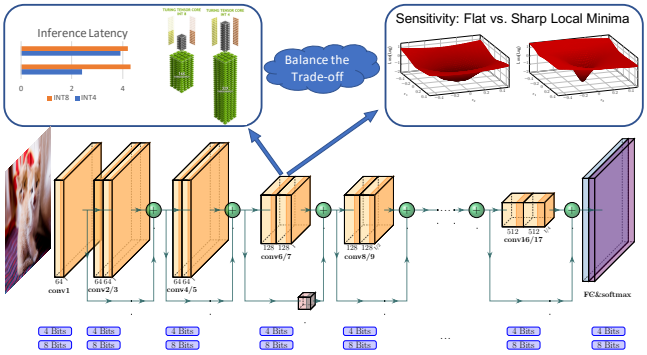

Figure 3. Illustration of inference speed and generalization performance trade-off of ResNet18. For each layer, we need to consider the speedup of INT4 vs INT8 and the sensitivity based on the second order (Hessian) sharpness (Dong et al., 2020) of this layer.

Assume that we have B choices for quantizing each layer (i.e., 2 for INT4 or INT8). For a model with L layers, the search space of the ILP will be B^L. The goal of solving the ILP problem is to find the best bit configuration among these B^L possibilities that results in optimal trade-offs between model perturbation Ω, and user-specified constraints such as model size, BOPS, and latency. Each of these

[2] This is either the input or the output activation after the downsampling layer.

[3] The speedup of each layer is calculated by the latency of INT8 divided by that of INT4. For uniform 4-bit and mixed-precision models, the speedup is calculated related to uniform 8-bit model.

bit-precision settings could result in a different model perturbation. To make the problem tractable, we assume that the perturbations for each layer are independent of each other (i.e., $\Omega = \sum_{i=1}^{L} \Omega_i^{(b_i)}$, where $\Omega_i^{(b_i)}$ is the i-th layer's perturbation with b_i bit)[4]. This allows us to precompute the sensitivity of each layer separately, and it only requires BL computations. For the sensitivity metric, we use the Hessian based perturbation proposed in (Dong et al., 2020, Eq. 2.11). The ILP problem tries to find the right bit precision that minimizes this sensitivity, as follows:

$$\text{Objective: } \min_{\{b_i\}_{i=1}^{L}} \sum_{i=1}^{L} \Omega_i^{(b_i)}, \tag{8}$$

$$\text{Subject to: } \sum_{i=1}^{L} M_i^{(b_i)} \leq \text{ Model Size Limit}, \tag{9}$$

$$\sum_{i=1}^{L} G_i^{(b_i)} \leq \text{ BOPS Limit}, \tag{10}$$

$$\sum_{i=1}^{L} Q_i^{(b_i)} \leq \text{ Latency Limit}. \tag{11}$$

Here, $M_i^{(b_i)}$ denotes the size of i-th layer with b_i bit quantization, $Q_i^{(b_i)}$ is the associated latency, and $G_i^{(b_i)}$ is the corresponding BOPS required for computing that layer. The latter measures the total Bit Operations for calculating a layer (van Baalen et al., 2020):

$$G_i^{(b_i)} = b_{w_i} b_{a_i} \text{MAC}_i,$$

where MAC_i is the total Multiply-Accumulate operations for computing the i-th layer, and b_{w_i}, b_{a_i} are the bit precision used for weight and activation.[5] Note that it is not necessary to set all these constraints at the same time. Typically, which constraint to use depends on the end-user application.

We solve the ILP using open source PULP library (Roy & Mitchell, 2020) in Python, where we found that for all the configurations tested in the paper, the ILP solver can find the solution in less than 1 second given the sensitivity metric. For comparison, the RL based method of (Wang et al., 2019) could take tens of hours to find the right bit-precision setting. Meanwhile, as can be seen, our ILP solver can be easily used for multiple constraints. However, the complexity of Pareto frontier proposed by (Dong et al., 2020) is exponentially increasing for multiple constraints. In Section 4.2, we show the results with different constraints.

We should also mention that the contemporary work of (Hubara et al., 2020), also proposed an ILP formulation. However, our approach is hardware-aware and we directly deploy and measure the latency of each layer in hardware.

[4] Similar assumption can be found in (Dong et al., 2019; 2020).

[5] b_{w_i} and b_{a_i} are always the same in HAWQ-V3. As such, HAWQ-V3 does not need to cast lower-precision integer numbers, e.g., INT4, to higher-precision integer numbers, e.g., INT8, which is more efficient than (Cai et al., 2020; Dong et al., 2020; Wang et al., 2019).

3.5. Hardware Deployment

Model size alone is not a good metric to measure the efficiency (speed and energy consumption) of NNs. In fact, it is quite possible that a small model would have higher latency and consume a larger amount of energy for inference. The same is also true for FLOPs. The reason is that neither model size nor FLOPs can account for cache misses, data locality, memory bandwidth, underutilization of hardware, etc. To address this, we need to deploy and directly measure the latency.

We target Nvidia Turing Tensor Cores of T4 GPU for deployment, as it supports both INT8 and INT4 precision and has been enhanced for deep learning network inference. The only API available is the WMMA kernel call which is a micro-kernel for performing matrix-matrix operations in INT4 precision on Tensor Cores. However, there is also no existing compiler that would map a NN quantized to INT4 to Tensor Cores using WMMA instructions. To address this challenge, another contribution of our work is extending TVM (Chen et al., 2018) to support INT4 inference with/without mixed precision with INT8. This is important so we can verify the speed benefits of mixed-precision inference. To accomplish this, we had to add new features in both graph-level IR and operator schedules to make INT4 inference efficient. For instance, when we perform optimizations such as memory planning, constant folding, and operator fusion, at the graph-level IR, 4-bit data are involved. However, on byte-addressable machines, manipulating 4-bit data individually leads to inefficiency in storage and communication. Instead, we pack eight 4-bit elements into an INT32 data type and perform the memory movement as a chunk. In the final code generation stage, the data type and all memory access will be adjusted for INT32. By adopting similar scheduling strategies to Cutlass (NVIDIA, 2020), we implement a new direct convolution schedule for Tensor Cores for both 8-bit and 4-bit data in TVM. We set the knobs for the configurations such as thread size, block size, and loop ordering so that the auto-tuner in TVM could search for the best latency settings.

Another important point is that we have completed the pipeline to test directly the trained weights and to avoid using random weights for speed measurements. This is important, since small discrepancies between the hardware implementation may go unnoticed from the quantization algorithm in the NN training framework (PyTorch in our case) which does not use TVM for the forward and backward propagation. To avoid any such issue, we made sure that the results between TVM and PyTorch match for every single layer and stage to machine-precision accuracy, and we verified the final Top-1 accuracy when executed in the hardware with integer-only arithmetic. In Appendix G, we present the error accumulation of feature maps for ResNet50 with INT4

quantization, which uses fake quantization in PyTorch and is deployed in TVM.

4. Results

In this section, we first discuss ImageNet results on various models (ResNet18/50 and InceptionV3) for INT8, INT4, and mixed-precision INT4/8 with/without distillation. Afterward, we study the different use cases of the ILP formulation, and the corresponding trade-offs between model size, latency, and accuracy. Detailed discussion on the implementation and set up is provided in Appendix H. For all the experiments we made sure to report and compare with the highest accuracy known for the baseline NN model in FP32 (i.e., we use a strong baseline for comparison). This is important since using a weak baseline accuracy could lead to misleading quantization accuracy.

4.1. Low Precision Integer-Only Quantization Results

We first start with ResNet18/50 and InceptionV3 quantization on ImageNet, and compare the performance of HAWQ-V3 with other approaches, as shown in Table 1.

Uniform 8-bit Quantization. Our 8-bit quantization achieves similar accuracy compared to the baseline. Importantly, for all the models HAWQ-V3 achieves higher accuracy than the integer-only approach of (Jacob et al., 2018). For instance, on ResNet50, we achieve 2.68% higher accuracy as compared to (Jacob et al., 2018). This is in part due to our BN folding strategy that was described in Section 3.2.

Uniform 4-bit Quantization. To the best of our knowledge, 4-bit results of HAWQ-V3 are the first integer-only quantization results reported in the literature. The accuracy results for ResNet18/50, and InceptionV3 are quite high, despite the fact that all of the inference computations are restricted to be integer multiplication, addition, and bit shifting. While there is some accuracy drop, this should not be incorrectly interpreted that uniform INT4 is not useful. On the contrary, one has to keep in mind that certain use cases have strict latency and memory footprint limit for which this may be the best solution. However, higher accuracy can be achieved through mixed-precision quantization.

Mixed 4/8-bit Quantization. The mixed-precision results improve the accuracy by several percentages for all the models, while slightly increasing the memory footprint of the model. For instance, the mixed-precision result for ResNet18 is 1.88% higher than its INT4 counterpart with just a 1.9MB increase in model size. Further improvements are also possible with distillation (denoted as HAWQV3+DIST in the table). For ResNet50, the distillation can boost the mixed-precision by 1.34%. We found that distillation helps most for mixed-precision quantization, and we found little to no improvement for uniform INT8, or uniform INT4 quantization cases.[6]

Table 1. Quantization results for ResNet18/50 and InceptionV3. Here, we abbreviate Integer-Only Quantization as "Int", Uniform Quantization as "Uni", the Baseline Accuracy as "BL", Weight Precision and Activation Precision as "Precision", Model Size as "Size" (in MB), Bit Operations as "BOPS" (in G), and Top-1 Accuracy as "Top-1". Also, "WxAy" means weight with x-bit and activation with y-bit, and 4/8 means mixed precision with 4 and 8 bits. "MP" means mixed precision with bitwidth ranging from 1-bit to 8-bit, and "W1*" means the bitwidth is 1-bit but the network architecture is changed (by using more channels). Our result with/without distillation is represented as HAWQV3+DIST/HAWQ-V3.

(a) *ResNet18*

Method	Int	Uni	BL	Precision	Size	BOPS	Top-1
Baseline	✗	–	71.47	W32A32	44.6	1858	71.47
RVQuant (Park et al., 2018b)	✗	✗	69.91	W8A8	11.1	116	70.01
HAWQ-V3	✓	✓	71.47	W8A8	11.1	116	**71.56**
PACT (Choi et al., 2018)	✗	✓	70.20	W5A5	7.2	50	69.80
LQ-Nets (Zhang et al., 2018)	✗	✗	70.30	W4A32	5.8	225	70.00
HAWQ-V3	✓	✓	71.47	W4/8A4/8	6.7	72	70.22
HAWQV3+DIST	✓	✓	71.47	W4/8A4/8	6.7	72	**70.38**
CalibTIB(Hubara et al., 2020)	✗	✓	71.97	W4A4	5.8	34	67.50
HAWQ-V3	✓	✓	71.47	W4A4	5.8	34	**68.45**

(b) *ResNet50*

Method	Int	Uni	BL	Precision	Size	BOPS	Top-1
Baseline	✓	✓	77.72	W32A32	97.8	3951	77.72
Integer Only (Jacob et al., 2018)	✓	✓	76.40	W8A8	24.5	247	74.90
RVQuant (Park et al., 2018b)	✗	✗	75.92	W8A8	24.5	247	75.67
HAWQ-V3	✓	✓	77.72	W8A8	24.5	247	**77.58**
PACT (Choi et al., 2018)	✗	✓	76.90	W5A5	16.0	101	76.70
LQ-Nets (Zhang et al., 2018)	✗	✗	76.50	W4A32	13.1	486	76.40
RVQuant (Park et al., 2018b)	✗	✗	75.92	W5A5	16.0	101	75.60
HAQ (Wang et al., 2019)	✗	✗	76.15	WMPA32	9.62	520	75.48
OneBitwidth (Chin et al., 2020)	✗	✓	76.70	W1*A8	12.3	494	76.70
HAWQ-V3	✓	✓	77.72	W4/8A4/8	18.7	154	75.39
HAWQV3+DIST	✓	✓	77.72	W4/8A4/8	18.7	154	**76.73**
CalibTIB(Hubara et al., 2020)	✗	✓	77.20	W4A4	13.1	67	73.70
HAWQ-V3	✓	✓	77.72	W4A4	13.1	67	**74.24**

(c) *InceptionV3*

Method	Int	Uni	BL	Precision	Size	BOPS	Top-1
Baseline	✗	✓	78.88	W32A32	90.9	5850	78.88
Integer Only (Jacob et al., 2018)	✓	✓	78.30	W8A8	22.7	366	74.20
RVQuant (Park et al., 2018b)	✗	✗	74.19	W8A8	22.7	366	74.22
HAWQ-V3	✓	✓	78.88	W8A8	22.7	366	**78.76**
Integer Only (Jacob et al., 2018)	✓	✓	78.30	W7A7	20.1	280	73.70
HAWQ-V3	✓	✓	78.88	W4/8A4/8	19.6	265	74.65
HAWQV3+DIST	✓	✓	78.88	W4/8A4/8	19.6	265	**74.72**
HAWQ-V3	✓	✓	78.88	W4A4	12.3	92	70.39

[6] We used simple distillation without extensive tuning. One might be able to improve the results further with more sophisticated distillation algorithms.

Overall, the results show that HAWQ-V3 achieves comparable accuracy to prior quantization methods including both uniform and mixed-precision quantization (e.g., PACT, RVQuant, OneBitwidth, HAQ which use FP32 arithmetic, and/or non-standard bit precision such as 5 bits, or different bit-width for weights and activations). Similar observations hold for InceptionV3, as reported in Table 1c.

Table 2. Mixed-precision quantization results for ResNet18 and ResNet50 with different constraints. Here, we abbreviate constraint level as "Level". Model Size as "Size", Bit Operations as "BOPS", the speedup as compared to INT8 results as "Speed", and Top-1 Accuracy as "Top-1", The last column of Top-1 represents results of HAWQ-V3 and HAWQV3+DIST. Note that for uniform INT8 ResNet50 (ResNet18), the latency is 1.06ms (0.40ms) per images.

(a) *ResNet18*

	Level	Size (MB)	BOPS (G)	Speed	Top-1
INT8	–	11.2	114	1x	71.56
Size	High	**9.9**	103	1.03x	71.20/71.59
	Medium	**7.9**	98	1.06x	70.50/71.09
	Low	**7.3**	95	1.08x	70.01/70.66
BOPS	High	8.7	**92**	1.12x	70.40/71.05
	Medium	6.7	**72**	1.21x	70.22/70.38
	Low	6.1	**54**	1.35x	68.72/69.72
Latency	High	8.7	92	**1.12x**	70.40/71.05
	Medium	7.2	76	**1.19x**	70.34/70.55
	Low	6.1	54	**1.35x**	68.56/69.72
INT4	–	5.6	28	1.48x	68.45

(b) *ResNet50*

	Level	Size (MB)	BOPS (G)	Speed	Top-1
INT8	–	24.5	247	1x	77.58
Size	High	**21.3**	226	1.09x	77.38/ 77.58
	Medium	**19.0**	197	1.13x	75.95/76.96
	Low	**16.0**	168	1.18x	74.89/76.51
BOPS	High	22.0	**197**	1.16x	76.10/76.76
	Medium	18.7	**154**	1.23x	75.39/76.73
	Low	16.7	**110**	1.30x	74.45/76.03
Latency	High	22.3	199	**1.13x**	76.63/76.97
	Medium	18.5	155	**1.21x**	74.95/76.39
	Low	16.5	114	**1.28x**	74.26/76.19
INT4	–	13.1	67	1.45x	74.24

4.2. Mixed-precision Results with Different Constraints

Here, we discuss various scenarios where different constraints could be imposed for quantization, and the interesting trade-offs associated with each scenario. The ILP problem in Eq. 8 has three constraints of model size, BOPS, and latency. We consider three different thresholds for each of the constraints and study how the ILP balances the trade-offs to obtain an optimal quantized model. We also focus on the case, where the practitioner is not satisfied with the performance of the INT4 quantization and wants to improve the performance (accuracy, speed, and model size) through mixed-precision quantization (INT4 and INT8). The ILP formulation enables the practitioner to set each or all of these constraints. Here, we present results when only one of these constraints is set at a time. The results are shown in Table 2, which is split into three sections of Size (model size), BOPS, and Latency. Each section represents the corresponding constraint as specified by the practitioner. The ILP solver then finds the optimal mixed-precision setting to satisfy that constraint, while maximizing accuracy. See Appendix I for the example of the latency constraint for ResNet18.

We start with the model size and BOPS constraints for ResNet18. The model size of pure INT4 quantization is 5.6MB, and INT8 is 11.2MB. However, the accuracy of INT4 quantization is 68.45% which maybe low for a particular application. The practitioner then has the option to set the model size constraint to be slightly higher than pure INT4. One option is to choose 7.9MB which is almost in between INT4 and INT8. For this case, the ILP solver finds a bit-precision setting that results in 71.09% accuracy which is almost the same as INT8. This model is also 6% faster than INT8 quantization.

Another possibility is to set the speed/latency as a constraint. The results for this setting are represented under the "Latency" row in Table 2. For example, the practitioner could request the ILP to find a bit-precision setting that would result in 19% faster latency as compared to the INT8 model (see "Medium" row). This results in a model with an accuracy of 70.55% with a model size of only 7.2MB. A similar constraint could also be made for BOPS.

Several very interesting observations can be made from these results. (i) The correlation between model size and BOPS is weak which is expected. That is a larger model size does not mean higher BOPS and vice versa. For example, compare Medium-Size and High-BOPS for ResNet18. The latter has lower BOPS despite being larger (and is actually faster as well). (ii) The model size does not directly correlate with accuracy. For example, for ResNet50, High-BOPS has a model size of 22MB and accuracy of 76.76%, while High-Size has a smaller model size of 21.3MB but higher accuracy of 77.58%.

In summary, although directly using INT4 quantization may result in large accuracy degradation, we can achieve significantly improved accuracy with much faster inference as compared to INT8 results. This gives the practitioner a wider range of choices beyond just INT8 quantization. Finally, we should mention that the accuracy and speed for all of the results shown for ResNet18/50 and InceptionV3 have been verified by directly measuring them when executed

in quantized precision in hardware through TVM. As such, these results are actually what the practitioner will observe, and these are not simulated results.

5. Conclusions

In this work, we presented HAWQ-V3, a new low-precision integer-only quantization framework, where the entire inference is executed with only integer multiplication, addition, and bit shifts. In particular, no FP32 arithmetic or even integer division is used in the entire inference. We presented results for uniform and mixed-precision INT4/8. For the latter, we proposed a hardware-aware ILP based method that finds the optimal trade-off between model perturbation and application specific constraints such as model size, inference speed, and total BOPS. The ILP problem can be solved very efficiently, under a second for all the models considered here. We showed that our approach can achieve up to 5% higher accuracy as compared to the prior integer-only approach of (Jacob et al., 2018). Finally, we directly implemented the low-precision quantized models in hardware by extending TVM to support INT4 and INT4/8 inference. We verified all the results, by matching the activation of each layer with our PyTorch framework (up to machine precision), including the verification of the final accuracy of the model. The framework, the TVM implementation, and the quantized models have been open sourced (HAWQ, 2020).

Acknowledgments

The UC Berkeley team acknowledges gracious support from Intel corporation, Intel VLAB team, Google Cloud, Google TPU Research Cloud team, Amazon, and Nvidia. Amir Gholami was supported through a gracious fund from Samsung SAIT. Michael W. Mahoney would also like to acknowledge the UC Berkeley CLTC, ARO, NSF, and ONR. Our conclusions do not necessarily reflect the position or the policy of our sponsors, and no official endorsement should be inferred.

References

PyTorchCV Library, 2020. URL https://pypi.org/project/pytorchcv/.

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al. Tensorflow: A system for large-scale machine learning. In *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, pp. 265–283, 2016.

Asanovic, K. and Morgan, N. *Experimental determination of precision requirements for back-propagation training of artificial neural networks*. International Computer Science Institute, 1991.

Banner, R., Nahshan, Y., and Soudry, D. Post training 4-bit quantization of convolutional networks for rapid-deployment. In *Advances in Neural Information Processing Systems*, pp. 7950–7958, 2019.

Bhalgat, Y., Lee, J., Nagel, M., Blankevoort, T., and Kwak, N. Lsq+: Improving low-bit quantization through learnable offsets and better initialization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pp. 696–697, 2020.

Cai, Y., Yao, Z., Dong, Z., Gholami, A., Mahoney, M. W., and Keutzer, K. ZeroQ: A novel zero shot quantization framework. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 13169–13178, 2020.

Chen, T., Li, M., Li, Y., Lin, M., Wang, N., Wang, M., Xiao, T., Xu, B., Zhang, C., and Zhang, Z. MXNet: A flexible and efficient machine learning library for heterogeneous distributed systems. *arXiv preprint arXiv:1512.01274*, 2015.

Chen, T., Moreau, T., Jiang, Z., Zheng, L., Yan, E., Shen, H., Cowan, M., Wang, L., Hu, Y., Ceze, L., et al. TVM: An automated end-to-end optimizing compiler for deep learning. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pp. 578–594, 2018.

Chetlur, S., Woolley, C., Vandermersch, P., Cohen, J., Tran, J., Catanzaro, B., and Shelhamer, E. cuDNN: Efficient primitives for deep learning. *arXiv preprint arXiv:1410.0759*, 2014.

Chin, T.-W., Chuang, P. I.-J., Chandra, V., and Marculescu, D. One weight bitwidth to rule them all. *arXiv preprint arXiv:2008.09916*, 2020.

Choi, J., Wang, Z., Venkataramani, S., Chuang, P. I.-J., Srinivasan, V., and Gopalakrishnan, K. PACT: Parameterized clipping activation for quantized neural networks. *arXiv preprint arXiv:1805.06085*, 2018.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. ImageNet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pp. 248–255. Ieee, 2009.

Dong, Z., Yao, Z., Gholami, A., Mahoney, M. W., and Keutzer, K. HAWQ: Hessian AWare Quantization of neural networks with mixed-precision. In *The IEEE International Conference on Computer Vision (ICCV)*, October 2019.

Dong, Z., Yao, Z., Arfeen, D., Gholami, A., Mahoney, M. W., and Keutzer, K. HAWQ-V2: Hessian aware trace-weighted quantization of neural networks. *Advances in neural information processing systems*, 2020.

Dukhan, M. NNPACK, 2016.

Elthakeb, A. T., Pilligundla, P., Mireshghallah, F., Elgindi, T., Deledalle, C.-A., and Esmaeilzadeh, H. Gradient-based deep quantization of neural networks through sinusoidal adaptive regularization. *arXiv preprint arXiv:2003.00146*, 2020.

Gholami, A., Kwon, K., Wu, B., Tai, Z., Yue, X., Jin, P., Zhao, S., and Keutzer, K. SqueezeNext: Hardware-aware neural network design. *Workshop paper in CVPR*, 2018.

Gray, R. M. and Neuhoff, D. L. Quantization. *IEEE transactions on information theory*, 44(6):2325–2383, 1998.

Gulli, A. and Pal, S. *Deep learning with Keras*. Packt Publishing Ltd, 2017.

Han, S. and Dally, B. Efficient methods and hardware for deep learning. *University Lecture*, 2017.

Han, S., Pool, J., Tran, J., and Dally, W. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pp. 1135–1143, 2015.

Han, S., Mao, H., and Dally, W. J. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *International Conference on Learning Representations*, 2016.

HAWQ. https://github.com/zhen-dong/hawq.git, October 2020.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Hinton, G., Vinyals, O., and Dean, J. Distilling the knowledge in a neural network. *Workshop paper in NIPS*, 2014.

Howard, A., Sandler, M., Chu, G., Chen, L.-C., Chen, B., Tan, M., Wang, W., Zhu, Y., Pang, R., Vasudevan, V., et al. Searching for MobileNetV3. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1314–1324, 2019.

Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., and Bengio, Y. Binarized neural networks. In *Advances in neural information processing systems*, pp. 4107–4115, 2016.

Hubara, I., Nahshan, Y., Hanani, Y., Banner, R., and Soudry, D. Improving post training neural quantization: Layer-wise calibration and integer programming. *arXiv preprint arXiv:2006.10518*, 2020.

Iandola, F. N., Han, S., Moskewicz, M. W., Ashraf, K., Dally, W. J., and Keutzer, K. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., Adam, H., and Kalenichenko, D. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2704–2713, 2018.

Jacob, B. et al. gemmlowp: a small self-contained low-precision gemm library.(2017), 2017.

Jain, A., Bhattacharya, S., Masuda, M., Sharma, V., and Wang, Y. Efficient execution of quantized deep learning models: A compiler approach. *arXiv preprint arXiv:2006.10226*, 2020.

Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., and Darrell, T. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*, pp. 675–678, 2014.

Kim, S. and Kim, H. Zero-centered fixed-point quantization with iterative retraining for deep convolutional neural network-based object detectors. *IEEE Access*, 9:20828–20839, 2021.

Krishnamoorthi, R. Quantizing deep convolutional networks for efficient inference: A whitepaper. *arXiv preprint arXiv:1806.08342*, 2018.

LeCun, Y., Denker, J. S., and Solla, S. A. Optimal brain damage. In *Advances in neural information processing systems*, pp. 598–605, 1990.

Li, H., Kadav, A., Durdanovic, I., Samet, H., and Graf, H. P. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

Mao, H., Han, S., Pool, J., Li, W., Liu, X., Wang, Y., and Dally, W. J. Exploring the regularity of sparse structure in convolutional neural networks. *Workshop paper in CVPR*, 2017.

Mishra, A. and Marr, D. Apprentice: Using knowledge distillation techniques to improve low-precision network accuracy. *arXiv preprint arXiv:1711.05852*, 2017.

Molchanov, P., Tyree, S., Karras, T., Aila, T., and Kautz, J. Pruning convolutional neural networks for resource efficient inference. *arXiv preprint arXiv:1611.06440*, 2016.

Naumov, M., Diril, U., Park, J., Ray, B., Jablonski, J., and Tulloch, A. On periodic functions as regularizers for quantization of neural networks. *arXiv preprint arXiv:1811.09862*, 2018.

NVIDIA. Cutlass library, 2020. URL https://github.com/NVIDIA/cutlass.

Park, E., Kim, D., and Yoo, S. Energy-efficient neural network accelerator based on outlier-aware low-precision computation. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pp. 688–698. IEEE, 2018a.

Park, E., Yoo, S., and Vajda, P. Value-aware quantization for training and inference of neural networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 580–595, 2018b.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. Automatic differentiation in pytorch. 2017.

Polino, A., Pascanu, R., and Alistarh, D. Model compression via distillation and quantization. *arXiv preprint arXiv:1802.05668*, 2018.

Rastegari, M., Ordonez, V., Redmon, J., and Farhadi, A. Xnor-Net: ImageNet classification using binary convolutional neural networks. In *European Conference on Computer Vision*, pp. 525–542. Springer, 2016.

Roy, J. and Mitchell, S. PuLP is an LP modeler written in Python. 2020. URL https://github.com/coin-or/pulp.

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. MobileNetV2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4510–4520, 2018.

Seide, F. and Agarwal, A. CNTK: Microsoft's open-source deep-learning toolkit. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 2135–2135, 2016.

Sharma, H., Park, J., Suda, N., Lai, L., Chau, B., Chandra, V., and Esmaeilzadeh, H. Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural networks. In *Proceedings of the 45th Annual International Symposium on Computer Architecture*, pp. 764–775. IEEE Press, 2018.

Shen, S., Dong, Z., Ye, J., Ma, L., Yao, Z., Gholami, A., Mahoney, M. W., and Keutzer, K. Q-BERT: Hessian based ultra low precision quantization of bert. In *AAAI*, pp. 8815–8821, 2020.

Song, Z., Fu, B., Wu, F., Jiang, Z., Jiang, L., Jing, N., and Liang, X. Drq: dynamic region-based quantization for deep neural network acceleration. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pp. 1010–1021. IEEE, 2020.

Tan, M. and Le, Q. V. Efficientnet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946*, 2019.

van Baalen, M., Louizos, C., Nagel, M., Amjad, R. A., Wang, Y., Blankevoort, T., and Welling, M. Bayesian bits: Unifying quantization and pruning. *arXiv preprint arXiv:2005.07093*, 2020.

Vasilache, N., Zinenko, O., Theodoridis, T., Goyal, P., DeVito, Z., Moses, W. S., Verdoolaege, S., Adams, A., and Cohen, A. Tensor comprehensions: Framework-agnostic high-performance machine learning abstractions. *arXiv preprint arXiv:1802.04730*, 2018.

Wang, K., Liu, Z., Lin, Y., Lin, J., and Han, S. HAQ: Hardware-aware automated quantization. *In Proceedings of the IEEE conference on computer vision and pattern recognition*, 2019.

Wu, B., Wang, Y., Zhang, P., Tian, Y., Vajda, P., and Keutzer, K. Mixed precision quantization of convnets via differentiable neural architecture search. *arXiv preprint arXiv:1812.00090*, 2018.

Wu, B., Dai, X., Zhang, P., Wang, Y., Sun, F., Wu, Y., Tian, Y., Vajda, P., Jia, Y., and Keutzer, K. FBNet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 10734–10742, 2019.

Yang, T.-J., Chen, Y.-H., and Sze, V. Designing energy-efficient convolutional neural networks using energy-aware pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5687–5695, 2017.

Yao, Z., Gholami, A., Keutzer, K., and Mahoney, M. W. PyHessian: Neural networks through the lens of the Hessian. *arXiv preprint arXiv:1912.07145*, 2019.

Yin, H., Molchanov, P., Alvarez, J. M., Li, Z., Mallya, A., Hoiem, D., Jha, N. K., and Kautz, J. Dreaming to distill: Data-free knowledge transfer via deepinversion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8715–8724, 2020.

Zhang, D., Yang, J., Ye, D., and Hua, G. LQ-Nets: Learned quantization for highly accurate and compact deep neural networks. In *The European Conference on Computer Vision (ECCV)*, September 2018.

Zhou, A., Yao, A., Guo, Y., Xu, L., and Chen, Y. Incremental network quantization: Towards lossless CNNs with low-precision weights. *International Conference on Learning Representations*, 2017a.

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

Zhou, Y., Moosavi-Dezfooli, S.-M., Cheung, N.-M., and Frossard, P. Adaptive quantization for deep neural network. *arXiv preprint arXiv:1712.01048*, 2017b.