Pareto-Optimal Quantized ResNet Is Mostly 4-bit

AmirAli Abdolrashidi*[a], Lisa Wang*[b],
Shivani Agrawal[b], Jonathan Malmaud[b], Oleg Rybakov[b], Chas Leichner[b], Lukasz Lew[b]
[a]University of California, Riverside, CA, USA
[b]Google Research, Mountain View, CA, USA
amirali.abdolrashidi@email.ucr.edu,
{wanglisa, shivaniagrawal, malmaud, rybakov, cleichner, lew}@google.com†

Abstract

Quantization has become a popular technique to compress neural networks and reduce compute cost, but most prior work focuses on studying quantization without changing the network size. Many real-world applications of neural networks have compute cost and memory budgets, which can be traded off with model quality by changing the number of parameters. In this work, we use ResNet as a case study to systematically investigate the effects of quantization on inference compute cost-quality tradeoff curves. Our results suggest that for each bfloat16 ResNet model, there are quantized models with lower cost and higher accuracy; in other words, the bfloat16 compute cost-quality tradeoff curve is Pareto-dominated by the 4-bit and 8-bit curves, with models primarily quantized to 4-bit yielding the best Pareto curve. Furthermore, we achieve state-of-the-art results on ImageNet for 4-bit ResNet-50 with quantization-aware training, obtaining a top-1 eval accuracy of 77.09%. We demonstrate the regularizing effect of quantization by measuring the generalization gap. The quantization method we used is optimized for practicality: It requires little tuning and is designed with hardware capabilities in mind. Our work motivates further research into optimal numeric formats for quantization, as well as the development of machine learning accelerators supporting these formats. As part of this work, we contribute a quantization library written in JAX, which is open-sourced at https://github.com/google-research/google-research/tree/master/aqt.

1. Introduction

While neural networks have brought tremendous progress to the computer vision field over the last decade, they are also often resource-intensive to train and serve. In many real-world computer vision applications, it is therefore of interest to minimize the compute cost (including power consumption/CO_2 output), and the memory footprint, ideally without compromising model quality.

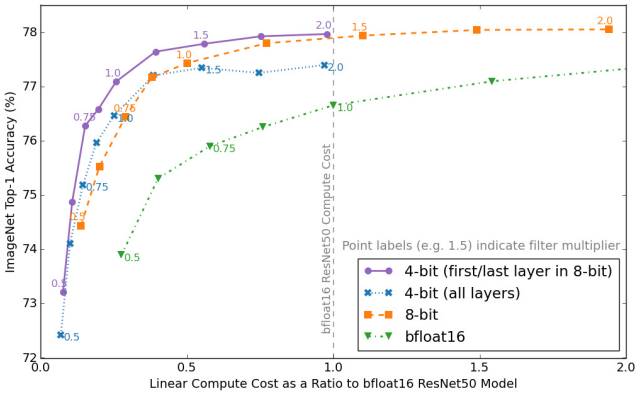

Figure 1. Compute cost-accuracy tradeoff curves, using a *linear* compute cost model.

Quantization [12] has become a popular technique to make neural networks, including computer vision models, more efficient. By reducing the number of bits used, quantization helps to compress the model, thus reducing its memory footprint. With support for quantized ops in hardware, e.g. NVIDIA A100 GPUs [22], quantization can also significantly speed up computation and lower power consumption, which can reduce overall compute costs. Quantization directly reduces per-op power-usage, an important factor in both chip design and the cost of ownership. Table 1 shows NVIDIA A100 peak performance in the context of quantization.

Despite its advantages, there could also be downsides to quantizing a model, e.g., a possible reduction in model quality. Figure 2 displays an example showing quantization steps performed on a series of data, e.g. model weights. During quantization, the tensors would be scaled

*Equal Contribution.
†Correspondence to: wanglisa@google.com, lew@google.com.

Input types	*TOPS*
float16	312
bfloat16	312
int8	624
int4	1248
binary	4992

Table 1. NVIDIA A100 performance on quantized types.

down, clipped, rounded to their closest quantization level and scaled back to their original domain. As it can also be seen in the figure, some of the values after quantization could be considerably far from their original values. Depending on the model, this could result in a quality reduction and, in some cases, prevent the model from converging.

A major hurdle to wider adoption of quantization both in industry and in research is the added engineering complexity introduced by quantization, e.g. some quantization techniques add hyperparameters for clipping bounds, which would require tuning as well. This motivates us to focus on approaches that offer clear benefits while minimizing the amount of added complexity.

In this work, using ResNet [13] as an example, we seek to understand how different quantization precisions affect the compute cost-accuracy tradeoff curves, and find a simple strategy to compress models at different compute cost and quality requirements.

After running experiments on ResNet using different precisions and numbers of parameters, we determined that 4-bit and 8-bit models strongly Pareto-dominate bfloat16 models, and mostly-4-bit models outperform 8-bit models. We present our results using two compute cost models (*linear* and *quadratic*), based on different assumptions about the compute speedups when the number of bits are reduced. Both cost models will be defined and justified in Section 4. While no hardware with the *quadratic* cost model is available yet to our knowledge, our results provide strong motivation for the development of such hardware.

2. Background

2.1. Related Work

One of the advantages model quantization provides is enabling more embedded systems to use such models [15]. However, it also provides server-side benefits by reducing data transfers and computation complexities [20]. Therefore, there have been many works in recent literature on improving the quality of various quantized neural networks, leveraging methods such as improved quantization techniques, compact model design, or even a combination of methods [18]. Leveraging embedded systems and supporting hardware for operations such as convolution has also been proposed to enable 4-bit quantization with low quality loss and reduce power consumption [33]. MobileNets [15] reduce convolution complexity by breaking it into two simpler layers, and use two multipliers in the model to reduce size in exchange for an acceptable accuracy loss. The authors in [32] study a multitude of pre-trained neural networks, and observe that they can all be quantized to int8 with their accuracies remaining within 1% of the baseline model. In order to find the best quantization parameters for a given network, neural architecture search (NAS) could be used [38, 9]. EfficientNet [30] uses NAS to explore combinations of changing number of channels, resolution, and depths of different convolutional neural networks, such as ResNet [13], Inception [29] and AmoebaNet [25], in order to find the optimal point with resource constraints in mind, e.g. mobile applications. When compared to a network with similar accuracy, EfficientNets have shown to be up to 8.4x smaller and 6.1x faster on hardware during inference. However, NAS can be very complex and resource-consuming, prompting us to look for a simpler solution.

Post-training quantization (PTQ) enables the user to convert an already trained float model and quantize it without retraining [10, 23, 7, 11]. However, it can also result in drastic reduction in model quality. To address the quality degradation, quantization-aware training (QAT) has been proposed and applied in several papers [6, 8, 32] and was also our method of choice.

Obtaining optimal clipping bounds for quantizing activations has been studied in several works. Choi et al. [6] propose a quantization scheme in which an activation clipping parameter for controlling bounds is introduced and optimized during training. With this technique, they are able to quantize weights and activations to 4-bits with little loss in quality.

2.2. ResNet50

We use the bfloat16 ResNet50 v1.5 model as our baseline and implemented our quantized model on top of JAX [4] MLPerf ResNet50 submission. Figure 3 shows the ResNet50 architecture used in this work. It consists of an initial convolutional layer (*conv_init*), 16 residual blocks and a dense layer at the end. Within each residual block, as shown in Figure 4, there are three convolutional layers in series. In addition, there is also a projection layer at the beginning of each block group only (thus 4 in total in ResNet50), which is responsible for reshaping the inputs so they can be used by the rest of the convolutional layers in the block group.

To change the number of model parameters, we multiply the number of convolutional filters in each layer by a global filter multiplier $c \in (0.5, 2)$.

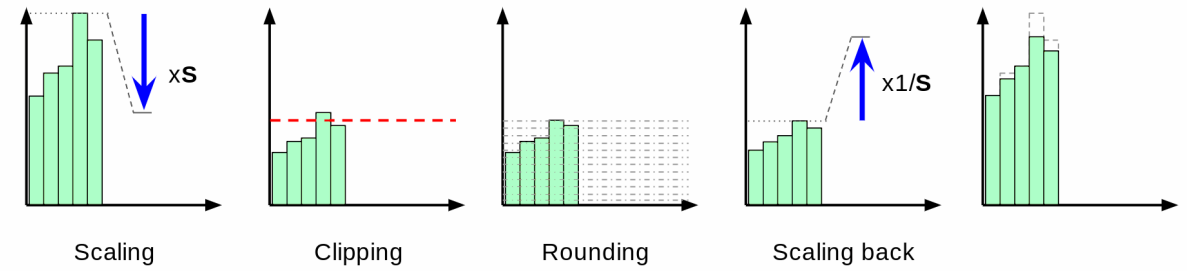

Figure 2. Different steps of quantization for a single data value (left to right), including the difference between the original data and the final quantized output (right).

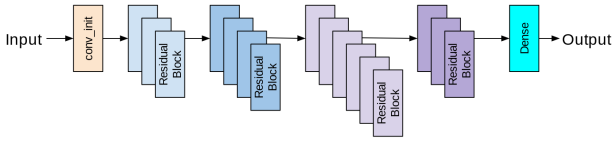

Figure 3. ResNet50 architecture

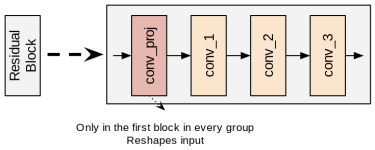

Figure 4. ResNet block

3. Quantization Details

In this work, our goal is to study the impact of quantization on cost-quality tradeoffs at different precisions and find practical, hardware-friendly approaches for model compression, so we chose to build our work on top of popular quantization methods found in literature. We apply quantization to both weights and activations, and introduce quantization during training. We decided to focus on 8-bit and 4-bit quantization, since these formats are already supported in existing hardware.

3.1. Uniform Quantization

For our experiments, we employ uniform integer quantization, where all quantization buckets are of equal size. This is a necessary choice if one wants to benefit from acceleration in existing hardware. We use scaling and clipping to reduce the impact of outliers. This form of quantization is most popular; for instance, it is found in prior work [12, 32, 33] and open-source machine learning frameworks including TensorFlow [1] and PyTorch [24].

Uniform quantization consists of the following steps, which are also shown in Figure 2:

1. *Scale*: Scale floating point input x by S, usually to ensure efficient use of the target range, e.g. $[-127, 127]$ for 8-bit signed integers. We compute scales per-channel for both weights and activations. The process of obtaining scales will be explained in more detail in the next section on calibration. Note that we do not apply shift, which helps us preserve the 0 value.

2. *Clipping*: An ideal scale would allow us to express all the quantized values in our quantization range, and significantly different values once scaled would be fall into separate quantization buckets. However, that may not always be possible, especially when outliers are present. Scale must be chosen in such a way to balance outlier clipping and bucket resolution. We use unsigned int types for activations coming out of ReLu as they are always positive. This effectively gains 1 bit of precision.

3. *Rounding*: After all the values are within the same scale, each value is rounded to the nearest quantization step, resulting in quantization error. If we need B bits to represent a quantized value, there will be 2^B steps in the range. However, based on our preferences, the range limits could be either *positive* or *symmetric*, giving us a integer range of $[0, 2^B - 1]$ or $[-2^{B-1} + 1, 2^{B-1} - 1]$ respectively.

4. *Scaling back*: After rounding, the values are scaled back to the original range by multiplying with $\frac{1}{S}$. The difference between the original unquantized and the values after quantization is also known as *quantization error*, which is introduced by the clipping and rounding steps.

3.2. Calibration of Clipping Bounds

There are several methods to choose the clipping bounds for the scaling step. One could use hard-coded fixed bounds, e.g. clip values to a range of $[0, 6]$, which is inspired by Relu6 [19]. These fixed bounds can also be picked via hyperparameter tuning, but the number of hyperparameter quickly explodes e.g. if we want to choose different scales for different layers or even different channels within a layer. Prior work [32] as well as our own experiments have shown

that automatically picking bounds during training, a process also known as calibration, yields better accuracy and requires far less hyperparameter tuning compared to fixed bounds. It also allows us to easily customize bounds for each layer and even channel within a layer. For example, if the weight tensor has shape $[3 \times 3 \times 6]$ where 6 is the number of output channels (or convolutional filters), we would have 6 different scales for each of the channels.

To obtain the clipping bounds for activations, we use the following calibration method:

1. For the first N training steps, we do not quantize activations, but compute per-channel statistics of $max(abs(x))$ values and keep track of their exponential moving averages (EMA). We also experimented with other calibration methods, e.g. $c \times stddev(x)$, but found that EMA of $max(abs(x))$ worked best for the ResNet case, and doesn't add sensitive hyperparameters.

2. At training step N, we finalize the calibration of clipping bounds by setting them to their most recent moving averages, turn on activation quantization with the new clipping bounds, and continue training with quantization. Importantly, we only calibrate activation clipping bounds once during training, as we found that more frequent calibrations can lead to feedback loops (e.g. vanishing or exploding bounds) as well as worse results. This choice makes the model insensitive to EMA hyperparameter. The specific step N at which to finalize the calibration and turn on activation quantization is a hyperparameter, but we found that the model is insensitive to the specific choice of the step. We recommend turning on activation quantization and setting the bounds between 10% and 40% of the total number of training steps it takes to reach convergence. We used 20% in all our experiments.

To obtain the clipping bounds for weights, we use the maximum absolute value of the current weight tensor per channel as clipping bounds. As this simple dynamic method has worked well for us, we did not try more complicated calibration methods for weights.

3.3. Quantization Library in JAX and Flax

We implemented a collection of quantization techniques and quantized neural networks on top of the JAX framework [4] and the Flax library [14] to enable fast experimentation, and used it to run the experiments in this paper. All code is at https://github.com/google-research/google-research/tree/master/aqt. To our knowledge, this is the first open source quantization library for JAX and Flax.

Highlighted Features:

1. *Quantized JAX dot and Flax layers:* We wrote a custom version of `jax.lax.dot` (matrix multiply) with optional weight and activation quantization. We also implemented quantized versions of common Flax layers, including *Dense* and *Conv*, which can be used as drop-in replacements.

2. *Multiple quantization strategies:* The user can choose between several algorithms to calibrate bounds automatically, including running mean of maximum values or statistics-based methods e.g. standard deviation or absolute-deviation of activations.

3. *Flexible configuration system:* Our configuration system enables fine-grained control of quantization settings, e.g., allowing the user to set separate precisions and quantization strategies for different layers which can useful for experimenting with mixed-precision models. Weights and activations within a layer can be set to different precisions as well.

4. *Support for unsigned and signed quantization:* We allow the user to configure whether to use unsigned or signed precision. This is especially useful for 4-bit quantization of positive activations (e.g. after a ReLU activation function) without shifting, since the unsigned integer doubles the resolution in the positive range.

5. *What you train is what you serve:* Optionally, the user can choose to apply our Accurate Quantized Training (AQT) method instead of using fake-quantization [16]. AQT ensures that the forward-pass during training is the same as the forward pass during inference, i.e. the matrix multiplies are in true integer domain. This provides better quality guarantees, enables training-time cost savings and simplifies compiler logic for inference, since no conversion is needed from training to inference graphs.

4. Cost Models for Quantized Neural Networks

4.1. Linear Cost Model for Existing Hardware

Some processors [22, 1] can support operations with reduced-precision operands. These operations are faster and consume less power than full-precision operations. In particular, integer operations can provide a higher throughput and lower cost than floating point operations [32]. The NVIDIA A100 [22] processor architecture supports 16-bit, 8-bit, and 4-bit multiplications. As Table 1 shows, the hardware can process convolution and matrix multiplication operations twice as quickly when their inputs are quantized to 8 bits as when they are quantized to 16 bits. In other words, 8-bit computations have half the *compute cost* of 16-bit computations. Moving from 8 bits to 4 bits provides a

similar benefit. This means that there is a linear relationship between the (power of 2) total number of bits in the operation and the cost of that operation for these processors.

4.2. Energy-Motivated Quadratic Cost Model

In the process of quantizing a model, each halving of number of bits used leads to power usage being at least halved for all hardware aspects of running that model (SRAM and DRAM memory reads and writes, data movement around the chip, arithmetic operations and nonlinear functions). For the multiplications, however, power usage scales better than linear as the precision is reduced. Given the extreme importance of multiplier power to ML accelerators (e.g. in matrix multiplications, einsums, and convolutions), this motivates a closer look at the fundamental cost model for multiplications.

Hardware multiplication of two n-bit numbers requires reducing n^2 bits to $2n$ bits i.e.,

$$a \cdot b = \sum_{0 \le i,j < n} 2^{i+j} \text{ AND } (a_i, b_i)$$

This is done using appropriately wired adder circuits, e.g. Wallace trees. Each of them takes 3 bits of input and produces two bits of output. This means that an n-bit multiplier requires roughly n^2 adders and AND gates. Multiplying two $2n$-bit numbers therefore requires $4n^2$ adders. Further, one $2n$-bit multiplier is roughly equivalent to *four* n-bit multipliers. This relationship motivates a quadratic cost model. The quadratic relationship holds for both the area and the power of integer multiplier circuits.

4.3. Cost Modeling of Layers and Full Network

The compute cost of the whole neural network is modeled as the sum of the costs of each layer. Since the cost of multiplication operations usually dominates the cost of arithmetic operations [2], layer cost is approximated by the sum of the costs of all the multiplication operations. While we use this approach to assign a cost to each layer, a ResNet model mostly consists of convolution layers and dense layers. The computational costs for these layers is a follows:

$$\text{Cost}_{\text{Conv2D}} = B K_{\text{h}} K_{\text{w}} A_{\text{w}} A_{\text{h}} C_{\text{in}} C_{\text{in}} M$$

$$\text{Cost}_{\text{Dense}} = B C_{\text{in}} C_{\text{out}} M$$

B is batch size, K_{h} and K_{w} are kernel width and height, A_{w} and A_{h} are layer's output image width and height, C_{in} and C_{out} are number of input and output channels, M is either a linear or quadratic cost coefficient defined above. We report only relative costs between quantization levels so for bfloat16, 8-bit integer, and 4-bit integer layers, M is respectively 16, 8, 4 in the linear cost model and 16, 4, 1 in the quadratic cost model.

Similarly, as a metric of memory usage, *memory cost* can also be approximated to the total number of weight bits (ignoring biases and other small contributions) in the entire model. The number of bits for each convolution and dense layer will be as below:

$$\text{Mem}_{\text{Conv2D}} = K_{\text{h}} K_{\text{w}} C_{\text{in}} C_{\text{in}} M'$$

$$\text{Mem}_{\text{Dense}} = C_{\text{in}} C_{\text{out}} M'$$

where M' for bfloat16, 8-bit integer, and 4-bit integer layers is 16, 8, 4 respectively in both the linear and quadratic cost models. This is an appropriate approximation of memory consumption because within a given layer, we quantize weights and activations to the same precision.

5. Experiments and Results

To evaluate the model quality, in all the experiments we computed and saved the top-1 accuracy on the ImageNet eval dataset [26]. We first compare our 4-bit results on ResNet50 without changing the number of parameters to prior work. We then share and discuss the results from compute cost-accuracy tradeoff experiments, where we change the number of parameters and quantization bits.

5.1. Comparison to Prior Work

While the primary focus of our work is to evaluate the impact of quantization on Pareto curves, we want to provide evidence for the competitiveness of our quantization method compared to prior work. Table 2 shows the top-1 accuracy of a ResNet50 model quantized using various methods, differing by goals and constraints, which we did our best to take into account as much as possible. As can be seen, our quantization method achieves or beats the results found in prior work on ResNet-50 quantization. E.g. our 4-bit model (with first and last layers in 8-bit) achieves a higher accuracy compared to results in the PACT paper [6], as well as the XILINX paper [33].

One may notice that even though all the results in Table 2 are obtained on ResNet50, they differ significantly in Top-1 accuracy, potentially due to differences in ResNet versions and hyperparameter choices. Therefore, we focus on quantization loss, i.e. the difference between Top-1 of unquantized and quantized model, as the main metric to evaluate quality of the quantization algorithms. In our case, both our 4-bit model (with first and last layers in 8-bit) and our fully 8-bit model outperform the bfloat16 baseline model, highlighting a regularizing effect of quantization. This regularizing effect can be clearly seen in the last three rows of Table 2. Differences in the generalization gap show that the unquantized model is overfitting significantly more to the training data than the quantized models do. We would like

to point out that other works may differ slightly due to quantization of ops other than Conv2D or MatMul, BatchNorm folding, etc.. We did not attempt to catch all the differences.

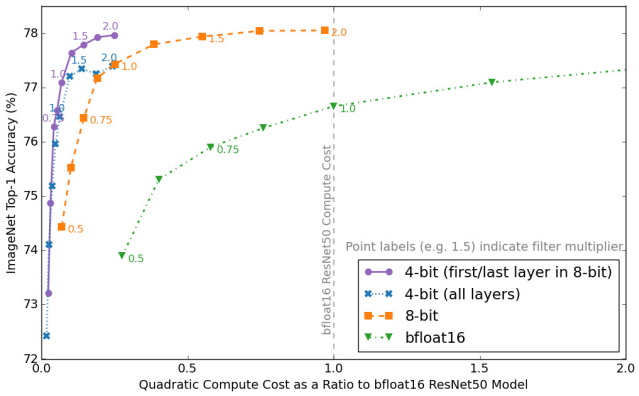

Figure 5. Quadratic compute Cost-Accuracy Tradeoff Curves

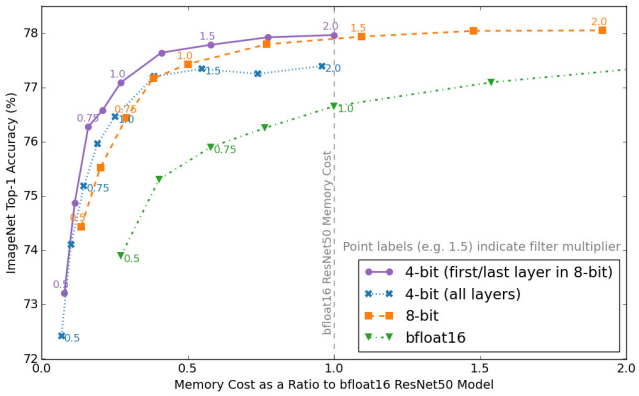

Figure 6. Memory Cost-Accuracy Tradeoff Curves

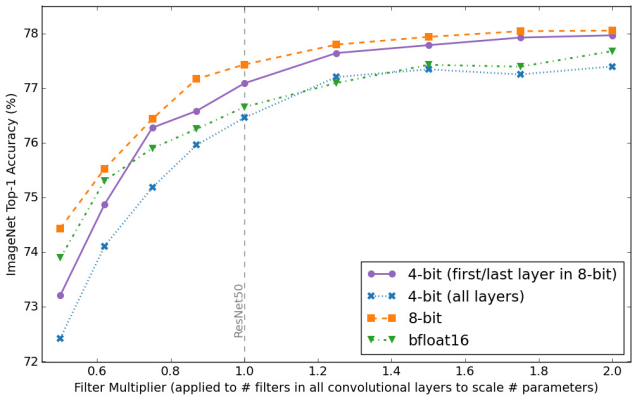

Figure 7. Accuracy with respect to filter multiplier. Models with the same multiplier have the same number of parameters.

5.2. Compute Cost-Accuracy Tradeoff Experiments

We ran experiments on ResNet50 with different number of parameters and quantization bits. To change the

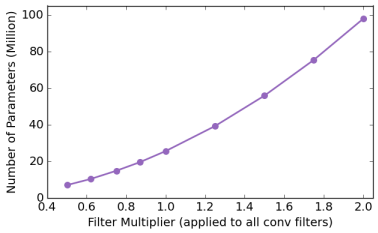

Figure 8. Number of parameters with respect to filter multiplier.

number of parameters, we multiply the number of filters in each convolutional layer by a global scalar, which we refer to as *filter multiplier*. We swept over nine filter multipliers $\{0.5, 0.62, 0.75, 0.87, 1.0, 1.25, 1.5, 1.75, 2.0\}$. For instance, standard ResNet50 (multiplier 1.0) has 25.5M parameters and a filter multiplier of 2.0 results in 97.8M parameters. See Figure 8 for a plot showing how filter multipliers affect the number of parameters. We explored four quantization settings which are represented as four curves on the figures. All settings apply both to weights and activations.

1. *4-bit*: All convolutional and dense layers including first and last layers (*conv_init* and *dense*) in 4-bit.

2. *4-bit, with first/last in 8-bit*: All layers in 4-bit, except first and last layers are in 8-bit. It is common practice to quantize these two layers to a lesser extent [6, 33, 6], since they tend to be most sensitive to quantization. We also found through our own layer sensitivity analysis that the first layer in ResNet is the most sensitive to quantization, followed by the last layer. Some solutions exist to address the challenge of quantizing first and last layers in neural networks, e.g. by remapping the input features [35]. We chose not to explore these special case methods, as it is not the focus of this work.

3. *8-bit*: All layers quantized to 8-bit.

4. *bfloat16*: Baseline, all layers in bfloat16.

The cross-product of the filter multipliers and quantization settings results in a total of 36 experiments.

Figure 1 showcases compute cost and Top-1 eval accuracy for ResNet50 with different parameters and quantization settings. The x-axis shows the linear compute cost (see Section 4.1) normalized to that of the baseline bfloat16 model. In this figure, we are comparing the four quantization settings described above. Each curve corresponds to a quantization setting, the different points on the curve correspond to different filter multipliers. The Pareto curves show the tradeoff between the compute cost and the accuracy in our experiments (towards upper-left is better).

As the curves in Figure 1 show, the 4-bit models with the first and last layers quantized to 8 bits achieve the best compute cost-accuracy Pareto curve, outperforming bfloat16,

Prior work	*float (baseline)*	*int8*	*int8 quantization loss*	*int4*	*int4 quantization loss*
IAO [16]	76.40%	74.90%	-1.50%	-	-
OCS [36]	76.10%	75.70%	-0.40%	66.20%[a]	-9.90%
XILINX [33] v2[d]	77.60%	77.47%	-0.13%	74.12%	-3.48%
KURE [27]	76.30%	-	-	74.30%	-2.00%
XILINX [33] v1[d]	76.15%	76.02%	-0.13%	74.59%	-1.56%
LQ-Net [34]	76.40%	-	-	75.10%	-1.30%
SSPS [28]	77.15%	-	-	76.22%[c]	-0.93%
PACT [6]	76.90%	-	-	76.50%[b]	-0.40%
HAQ [31]	76.15%	-	-	76.14%[c]	-0.01%
ZeroQ [5]	77.72%	77.67%	-0.05%	-	-
TQT [17]	75.20%	75.40%	+0.20%	-	-
FAQ [21]	76.15%	76.52%	+0.37%	76.25%	+0.10%
PQ+TS+Guided** [37]	75.60%	-	-	75.90%	+0.30%
NICE [3]	76.15%	-	-	76.50%	+0.35%
NVIDIA QAT [32]	76.16%	76.85%	+0.69%	-	-
This work	76.65%	77.43%	**+0.78%**	77.09%[b]	**+0.44%**
Train log-loss (this work)	0.633	0.723	-	0.762	-
Eval log-loss (this work)	0.974	0.881	-	0.889	-
Generalization gap (this work)	0.341	0.158	-	**0.125**	-

[a] int8 activations [b] int4 w/int8 first/last layer [c] Mixed precision [d] ResNet v1 and v2; quantizes element-wise operations

Table 2. ImageNet Top-1 accuracy comparison of baseline and quantized ResNet-50 models sorted by int8 and int4 quantization loss which is defined as difference in Top-1 between baseline and quantized model. Last tree rows list models' train log-loss (the value optimized by the training process) and eval log-loss. Generalization gap is the difference between them.

8-bit and all-4-bit models. The differences in the Pareto curves become more pronounced if we use the theoretical quadratic cost model (see 4.2 Quadratic Model), as shown in Figure 5. As expected, the Pareto curves are now separated more, showing an even clearer advantage of quantized models and 4-bit in particular. This motivates the development of hardware where quadratic cost savings are supported.

Figure 6 displays the memory cost-accuracy tradeoff points for the quantized models versus the baseline, which shows a very similar pattern to the compute cost-accuracy figures.

Figure 7 summarizes the same experiments, however the x-axis is now showing the filter multiplier applied to all convolution layers. Models with the same multiplier have the same number of parameters/model architecture. We observe that the 4-bit models w/8-bit first/last achieve a better accuracy than the baseline bfloat16 starting at $multiplier = 0.75$ and gets very close to the 8-bit quantized model towards $multiplier = 2.0$. The 8-bit models have better accuracies than bfloat16 models for all multipliers we tried. This means that even without changing the number of parameters, quantization offers an accuracy improvement for most multipliers, most likely due to its regularization effects.

5.3. Tradeoff-Aware Quantization Recipe

Based on our analysis, we propose a simple recipe for model compression on ResNet with minimal hyperparameter tuning.

1. Quantize all layers to 4 bits, and first and last layers (*conv_init* and *dense*) to 8 bits.

2. Change the number of parameters with a global filter multiplier to achieve the desired tradeoff based on the compute cost/memory cost and quality requirements.

Future Work

We intend to expand our analysis to additional models, especially those with architectures already optimized for efficiency, such as MobileNet, EfficientNet and Transformers. Furthermore, we would like to extend our research to binary quantization, with the goal to finding even more efficient networks. Another area of future work is to improve training costs with quantization, as this work focuses on optimizing inference costs. In addition, we hope to evaluate our quantized models on real hardware supporting these formats, e.g. NVIDIA A100.

Conclusion

In this work, we analyzed how quantization at different precisions influences the compute cost-quality Pareto curves on ResNet models. We found that quantization consistently improves the tradeoff regardless of where we are on the compute cost-quality tradeoff graph. In other words, for each bfloat16 model, we found quantized models with lower compute cost and higher accuracy. Additionally, models in 4-bit (with first and last layers in 8-bit) offer a consistent advantage over 8-bit, using both linear and quadratic cost models. This observation suggests that 4-bit may be a preferred numeric format for quantizing neural networks. Based on our results, we proposed a simple practical approach to compress models given compute cost and quality constraints, consisting of two steps: quantize the model to 4-bit, then multiply the number of parameters in each layer by a global factor to achieve the desired tradeoff. We invite further research into comparing different numeric formats for quantization, and hope that this line of work will inform the development of future hardware. We also encourage more works in model compression to compare methods on cost-quality tradeoff graphs, as this provides a more nuanced and thorough analysis. Lastly, we open-sourced our quantization library, in the hopes that it will accelerate quantization research and deployment with JAX.

Acknowledgements We thank Reiner Pope and Mike Gunter for fruitful discussions and feedback on this work. We would also like to thank James Bradbury and Anselm Levskaya for providing invaluable support on JAX/Flax as we implemented our AQT quantization library.

References

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, pages 265–283, 2016. 3, 4

[2] Ramesh C Agarwal, Susanne M Balle, Fred G Gustavson, Mahesh Joshi, and Prasad Palkar. A three-dimensional approach to parallel matrix multiplication. *IBM Journal of Research and Development*, 39(5):575–582, 1995. 5

[3] Chaim Baskin, Natan Liss, Yoav Chai, Evgenii Zheltonozhskii, Eli Schwartz, Raja Giryes, Avi Mendelson, and Alexander M Bronstein. Nice: Noise injection and clamping estimation for neural network quantization. *arXiv preprint arXiv:1810.00162*, 2018. 7

[4] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs. http://github.com/google/jax, 2018. 2, 4

[5] Yaohui Cai, Zhewei Yao, Zhen Dong, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. Zeroq: A novel zero shot quantization framework. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13169–13178, 2020. 7

[6] Jungwook Choi, Zhuo Wang, Swagath Venkataramani, Pierce I-Jen Chuang, Vijayalakshmi Srinivasan, and Kailash Gopalakrishnan. Pact: Parameterized clipping activation for quantized neural networks. *arXiv preprint arXiv:1805.06085*, 2018. 2, 5, 6, 7

[7] Steve Dai, Rangharajan Venkatesan, Haoxing Ren, Brian Zimmer, William J Dally, and Brucek Khailany. Vs-quant: Per-vector scaled quantization for accurate low-precision neural network inference. *arXiv preprint arXiv:2102.04503*, 2021. 2

[8] Zhen Dong, Zhewei Yao, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. Hawq: Hessian aware quantization of neural networks with mixed-precision. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 293–302, 2019. 2

[9] Thomas Elsken, Jan Hendrik Metzen, Frank Hutter, et al. Neural architecture search: A survey. *J. Mach. Learn. Res.*, 20(55):1–21, 2019. 2

[10] Jun Fang, Ali Shafiee, Hamzah Abdel-Aziz, David Thorsley, Georgios Georgiadis, and Joseph H Hassoun. Post-training piecewise linear quantization for deep neural networks. In *European Conference on Computer Vision*, pages 69–86. Springer, 2020. 2

[11] Sahaj Garg, Anirudh Jain, Joe Lou, and Mitchell Nahmias. Confounding tradeoffs for neural network quantization. *arXiv preprint arXiv:2102.06366*, 2021. 2

[12] Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding, 2016. 1, 3

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 2

[14] Jonathan Heek, Anselm Levskaya, Avital Oliver, Marvin Ritter, Bertrand Rondepierre, Andreas Steiner, and Marc van Zee. Flax: A neural network library and ecosystem for JAX. http://github.com/google/flax, 2020. 4

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

[16] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2704–2713, 2018. 4, 7

[17] Sambhav R Jain, Albert Gural, Michael Wu, and Chris H Dick. Trained quantization thresholds for accurate and effi-

cient fixed-point inference of deep neural networks. *arXiv preprint arXiv:1903.08066*, 2019. 7

[18] Eunhui Kim and Kyong-Ha Lee. Spatial shift point-wise quantization. *IEEE Access*, 8:207683–207690, 2020. 2

[19] Alex Krizhevsky. Convolutional deep belief networks on cifar-10, 2010. 3

[20] Guangli Li, Lei Liu, Xueying Wang, Xiao Dong, Peng Zhao, and Xiaobing Feng. Auto-tuning neural network quantization framework for collaborative inference between the cloud and edge. In *International Conference on Artificial Neural Networks*, pages 402–411. Springer, 2018. 2

[21] Jeffrey L McKinstry, Steven K Esser, Rathinakumar Appuswamy, Deepika Bablani, John V Arthur, Izzet B Yildiz, and Dharmendra S Modha. Discovering low-precision networks close to full-precision networks for efficient embedded inference. *arXiv preprint arXiv:1809.04191*, 2018. 7

[22] NVIDIA. Nvidia a100 tensor core gpu architecture. https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/nvidia-ampere-architecture-whitepaper.pdf, 2020. Accessed 24 February 2021. 1, 4

[23] Jihun Oh, SangJeong Lee, Meejeong Park, Pooni Walagaurav, and Kiseok Kwon. Weight equalizing shift scaler-coupled post-training quantization. *arXiv preprint arXiv:2008.05767*, 2020. 2

[24] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2017. 3

[25] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, volume 33, pages 4780–4789, 2019. 2

[26] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015. 5

[27] Moran Shkolnik, Brian Chmiel, Ron Banner, Gil Shomron, Yuri Nahshan, Alex Bronstein, and Uri Weiser. Robust quantization: One model to rule them all. *arXiv preprint arXiv:2002.07686*, 2020. 7

[28] Qigong Sun, Licheng Jiao, Yan Ren, Xiufang Li, Fanhua Shang, and Fang Liu. Effective and fast: A novel sequential single path search for mixed-precision quantization. *arXiv preprint arXiv:2103.02904*, 2021. 7

[29] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017. 2

[30] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, pages 6105–6114. PMLR, 2019. 2

[31] Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. Haq: Hardware-aware automated quantization with mixed precision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8612–8620, 2019. 7

[32] Hao Wu, Patrick Judd, Xiaojie Zhang, Mikhail Isaev, and Paulius Micikevicius. Integer quantization for deep learning inference: Principles and empirical evaluation. *arXiv preprint arXiv:2004.09602*, 2020. 2, 3, 4, 7

[33] Xilinx. Convolutional neural network with int4 optimization on xilinx devices. https://www.xilinx.com/support/documentation/white_papers/wp521-4bit-optimization.pdf, 2020. Accessed 1 March 2021. 2, 3, 5, 6, 7

[34] Dongqing Zhang, Jiaolong Yang, Dongqiangzi Ye, and Gang Hua. Lq-nets: Learned quantization for highly accurate and compact deep neural networks. In *Proceedings of the European conference on computer vision (ECCV)*, pages 365–382, 2018. 7

[35] Yichi Zhang, Junhao Pan, Xinheng Liu, Hongzheng Chen, Deming Chen, and Zhiru Zhang. Fracbnn: Accurate and fpga-efficient binary neural networks with fractional activations. In *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 171–182, 2021. 6

[36] Ritchie Zhao, Yuwei Hu, Jordan Dotzel, Chris De Sa, and Zhiru Zhang. Improving neural network quantization without retraining using outlier channel splitting. In *International conference on machine learning*, pages 7543–7552. PMLR, 2019. 7

[37] Bohan Zhuang, Chunhua Shen, Mingkui Tan, Lingqiao Liu, and Ian Reid. Towards effective low-bitwidth convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7920–7928, 2018. 7

[38] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016. 2