PikeLPN: Mitigating Overlooked Inefficiencies of Low-Precision Neural Networks

Supplementary Material

1. Elementwise Multiplications

In Section 3.2, we propose ACE_{v2} which extends ACE to account for elementwise multiplications. We derived the number of adders required for multiplying an i-bit number by a j-bit number as $i \cdot j - \max(i, j)$. Here we provide a more detailed justification for the derived formula. For simplicity, we assume both operands have the same number of bits (i.e., $i = j$) in this derivation. Elementwise multiplications requires a multiplier as well as an adder to account for the dot pattern at the completion of the multiplication [2]. We base our derivation on the established implementation of Dadda multiplier [2] and Ripple-Carry Adder (RCA) [1] to estimate the cost of elementwise multiplications. To multiply two i-bit numbers, the Dadda multiplier requires $i^2 - 3i + 2$ adders [2], while the RCA adds another $2i - 2$ adders. This leads to a total number of adders equal to $i^2 - i$. To generalize to operands with different precisions, the cost for elementwise multiplications between an i-bit number and a j-bit number can be derived as $i \cdot j - \max(i, j)$, with $i \cdot j$ reflecting the cost of the multiplier and $\max(i, j)$ representing the final addition. Independently, we performed an empirical verification for $1 \leq i, j \leq 64$ which confirmed the correctness of this formula, showing zero error in predicted adder counts. This refinement in ACE_{v2} cost calculation enhances our understanding of multiplier complexity.

2. Floating Point Elementwise Additions

In Section 3.2, we improve ACE by extending it to include the cost of floating-point elementwise addition. We derive the cost of adding an i-bit and a j-bit floating point numbers, using the formula

$$ACE_{fp-add} = c_a \cdot \max(i, j) \tag{1}$$

For simplicity, we assume both operands have the same number of bits (i.e., $i = j$) in this derivation. c_a reflects the added complexity of floating point operations compared to fixed-point addition. To derive c_a, we look into the components of floating-point adders [6] and analyze the ACE_{v2} cost for each component. Assuming e bits for the exponent and m bits for the mantissa, the main components of the floating-point adder and their corresponding ACE_{v2} costs are as follows:

1. *Exponent Subtraction*: Involves subtracting the exponent bits resulting in an ACE_{v2} cost of $\mathbf{e}$.
2. *Operand Swapping*: Requires a single multiplexer with negligible ACE_{v2} cost.
3. *Limitation of Alignment Shift Amount*: Involves adding the mantissa bits resulting in an ACE_{v2} cost of $\mathbf{m}$.
4. *Alignment Shift*: Involves shifting by the mantissa bits adding an ACE_{v2} cost of $\mathbf{m} \cdot \mathbf{log_2(m)/5}$[1].
5. *Significand Negation*: Involves one bit subtraction resulting in an ACE_{v2} cost of $\mathbf{1}$.
6. *Significand Addition*: Requires mantissa bits addition resulting in an ACE_{v2} cost of $\mathbf{m}$.
7. *Significand Conversion*: Requires two additions adding an ACE_{v2} cost of $\mathbf{2m}$.
8. *Normalization*: Requires shifting e bits resulting in an ACE_{v2} cost of $\mathbf{e} \cdot \mathbf{log_2(e)/5}$.
9. *Rounding and Post-normalization*: Requires adding m bits with an ACE_{v2} cost of $\mathbf{m}$.

Summing the costs for all the components, we get a total cost of $m(5 + \log_2(m)/5) + e + e \cdot \log_2(e)/5 + 1$. Considering the dominant role of mantissa operations, we approximate the total cost to $i(5 + log_2(i)/5)$ where i is the number of bits of the added floating point number. The upper bound for $\log_2(i)/5$ is 1 when m is 32. Therefore, we can derive the cost as $6i$ resulting in $c_a = 6$ in Equation 1. This approximation streamlines ACE_{v2} calculation for floating-point additions. To verify its correctness, we show that it aligns well with the independently measured energy consumption observed in 45nm CMOS technology in Table 1.

3. Model Scaling

To scale PikeLPN-1× to the 2×, 3×, and 6× sizes, we employ a series of scaling techniques including multiplying the output channels of the convolution layers by a scaling factor and increasing the precision of the feature maps at the point-wise convolution layers. These techniques increase both the ACE_{v2} cost and the representational capacity of the model, allowing us to generate a Pareto family of models. The details for each model are described in Table 1. For nomenclature, the scale factor represents the ACE_{v2} cost of the scaled model compared to that of the smallest model. For example, PikeLPN-3× has approximately 3 times the ACE_{v2} cost of PikeLPN-1×.

4. QuantNorm Layer

As shown in Subsection 3.3, *QuantNorm* reduces quantization error during training improving the performance of our

[1] ACE_{v2} cost for shift operation is derived as $i \cdot \log_2(j)/5$ in Subsection 3.2

Table 1. Comparison of PikeLPN variants' training parameters, with dropout rates calibrated to mitigate overfitting. Each model's training duration and learning rate strategy are customized according to its complexity. They are initialized with weights from an floating point PikeLPN model, employing a consistent learning rate of 10^{-12} during the tail period to enhance stability and validation accuracy, crucial for smaller models.

PikeLPN Size	**1×**	**2×**	**3×**	**6×**
$\boldsymbol{ACE_{v2}}$ ($\times 10^9$)	8.68	15.74	33.97	59.10
Channel Multiplier	1.0	1.0	1.5	2.0
Activation precision (int bits, frac bits, sign bit)	(6, 1, 1)	(8, 7, 1)	(8, 7, 1)	(8, 7, 1)
Removal of BN layers between depthwise and pointwise convolutions	Yes	No	No	No
Constant learning rate tail period (epochs)	300	300	20	50
Training epochs	500	1500	1000	1000
Dropout rate	1e-3	1e-3	0.5	0.7

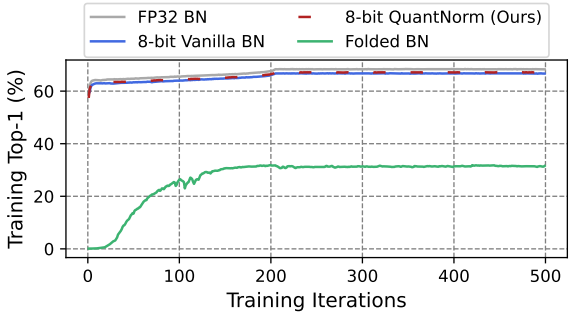
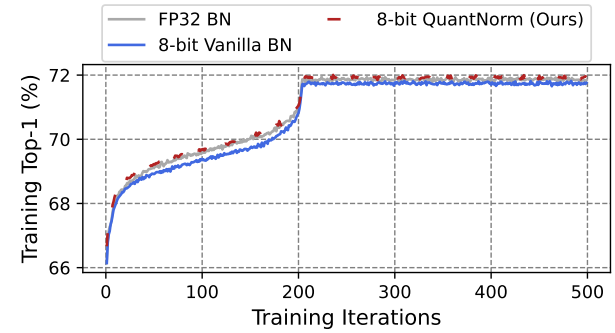


Figure 1. Training Top-1 Accuracy during QAT on ImageNet at different Batch Norm Quantization techniques for PikeLPN-1×.

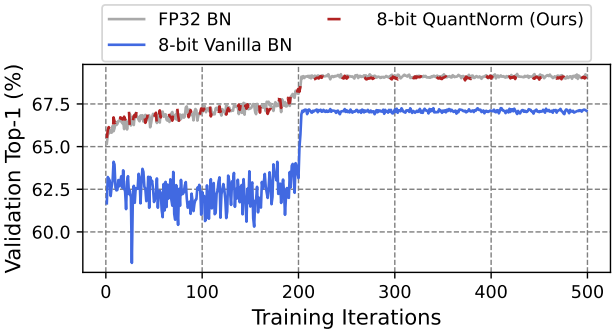
PikeLPN models on ImageNet image classification dataset [3]. As shown in Figure 6, *QuantNorm* maintains close-to-FP validation accuracy when using it during PikeLPN-1× training. Figure 1 shows the top-1 training accuracy while training the same model using different batch normalization quantization techniques. Moreover, to ensure that the same behaviour persists at different *PikeLPN* scales, Figures 2a and 2b shows the top-1 training and validation accuracies respectively of PikeLPN-2× when using our proposed *QuantNorm* layer versus the vanilla batch norm quantization shown in Equation 5.

5. Early LR Decay

As mentioned in Subsection 4.1, our *PikeLPN* models are trained using an *AdamW* optimizer and a Cosine Decay Schedule. The initial learning rate is $1e-4$ and annealed using a cosine schedule to $1e-12$. Figure 3 summarizes the training behaviour with various learning rate schedules. The x-axis represents the training iterations which we limit



(a) Training Accuracy (%)



(b) Validation Accuracy (%)

Figure 2. Top-1 Accuracy during QAT on ImageNet at different Batch Norm Quantization techniques for PikeLPN-2×.

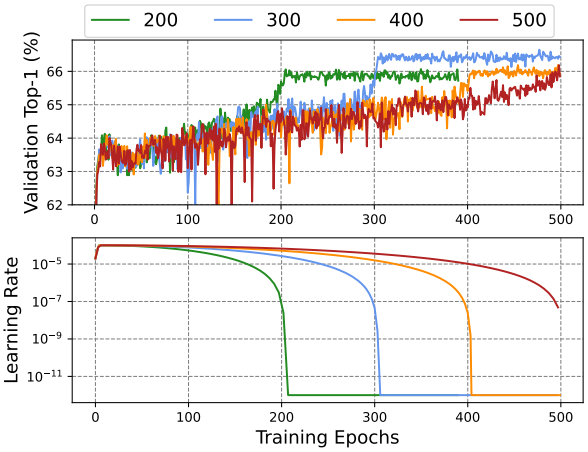


Figure 3. Validation Top-1 Accuracy for PikeLPN-1× model with various learning rate decay schedules. All the training sessions match exactly except for the number of decay steps which ranges between 200 and 500 epochs.

to 500 epochs. The y-axis in the top graph represents the validation top-1 accuracy on ImageNet, while the y-axis in the bottom graph represents the learning rate. All the training sessions match exactly except for the number of decay steps which ranges between 200 and 500 epochs. The fig-

Table 2. *PikeLPN* versus baselines - detailed analysis for contribution of elementwise operations to ACE_{v2}. *Total* represents the total percentage of elementwise operations from ACE_{v2}. *BN*, *ACT* and *QP* represents the detailed contribution of batch norm layers, activation layers and quantization overhead respectively.

Model	**MAC ACE_{v2}**	**Elementwise ACE_{v2}**			
		Total	BN	Act	QP
PokeBNN-0.5x	4.2	95.8%	43.4%	29.2%	21.5%
PikeLPN-1×	96.4	3.9%	3.7%	0%	0.2%
PROFIT	48%	52%	19%	17%	16%
PikeLPN-2×	97.9	2.1%	2%	0%	0.1%
PokeBNN-1x	6.2	93.8%	42.2%	28.4%	20.9%
PikeLPN-6×	98.9	1.13%	0.98%	0%	0.2%

ure highlights two main observations. First, training for the final few epochs at a constant low learning-rate (i.e., $1e-12$) help the weights of the low-precision models stabilize and significantly boost the accuracy (i.e., $1-2\%$). Second, the number of decay steps is an important hyperparameter when training low-precision models. For example, we noticed that for PikeLPN-1×, setting the number of decay steps to 300 gives an extra $0.5-1\%$ improvement in validation accuracy.

6. Detailed Analysis for Contribution of Elementwise operations to ACE_{v2}

Table 2 shows the detailed contributions of different elementwise computation sources to the overall ACE_{v2} cost.

7. Comparing ACE to similar metrics

In Section 3.2, we propose an extension to ACE, but our proposed extension could in principle generalize to other metrics similar to ACE. All previous metrics similar to ACE known by the authors only account for accumulate/dot-product operations and not elementwise operations, making our proposed extension generally valuable. Even so, we chose to specifically extend the ACE metric as opposed to other metrics due to ACE's simplicity and efficacy in predicting energy costs. We extend ACE because it was built with ML researchers in mind, creating balance between complexity and abstraction of hardware energy which -- from a physics perspective in CMOS - is likely to limit future ML hardware. We would like to compare ACE to four other metrics that are often brought up when trying to predict hardware costs: (1) Fanout-of-4 inverter delay (FO4) [7] is a constraint in hardware design, but not necessarily a target for ML researchers. (2) Ristretto [4] measures power cost through a labor-intensive synthesis, inaccessible to ML researchers. (3, 4) The Unit-gate model [8] and full-adder count [5] correlate very well with ACE and differ only by a small constant factor [2]. Therefore our ACE_{v2} extension would generalize to these metrics. Moreover, all those metrics, including ACE_{v2}, are technology independent.

[2]Unlike ACE_{v2}, their application to DNNs does not take the cost elementwise operations into account nor does it account for carry-save format for local accumulator representations typically used in systollic arrays.

References

[1] S. Archana and G. Durga. Design of low power and high speed ripple carry adder. In *2014 International Conference on Communication and Signal Processing*, pages 939–943, 2014. 1

[2] Wesley Donald Chu. Wallace and dadda multipliers implemented using carry lookahead adders. 2013. 1

[3] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009. 2

[4] Philipp Gysel, Jon Pimentel, Mohammad Motamedi, and Soheil Ghiasi. Ristretto: A framework for empirical study of resource-efficient inference in convolutional neural networks. *IEEE transactions on neural networks and learning systems*, 29(11):5784–5789, 2018. 3

[5] Charbel Sakr, Yongjune Kim, and Naresh Shanbhag. Analytical guarantees on numerical precision of deep neural networks. In *International Conference on Machine Learning*, pages 3007–3016. PMLR, 2017. 3

[6] P.-M. Seidel and G. Even. On the design of fast ieee floating-point adders. In *Proceedings 15th IEEE Symposium on Computer Arithmetic. ARITH-15 2001*, pages 184–194, 2001. 1

[7] Ivan Sutherland, Robert F Sproull, and David Harris. *Logical effort: designing fast CMOS circuits*. Morgan Kaufmann, 1999. 3

[8] Reto Zimmermann. Computer arithmetic: Principles, architectures, and vlsi design. *Personal publication (Available at http://www. iis. ee. ethz. ch/ zimmi/-publications/comp arith notes. ps. gz)*, 1999. 3