# PikeLPN: Mitigating Overlooked Inefficiencies of Low-Precision Neural Networks

Marina Neseem[*,†1], Conor McCullough[2], Randy Hsin[2], Chas Leichner[2], Shan Li[2], In Suk Chong[2], Andrew Howard[2], Lukasz Lew[2], Sherief Reda[1], Ville-Mikko Rautio[2], and Daniele Moro[†2]

[1]Brown University, [2]Google

## Abstract

*Low-precision quantization is recognized for its efficacy in neural network optimization. Our analysis reveals that non-quantized elementwise operations which are prevalent in layers such as parameterized activation functions, batch normalization, and quantization scaling dominate the inference cost of low-precision models. These non-quantized elementwise operations are commonly overlooked in SOTA efficiency metrics such as Arithmetic Computation Effort (ACE) [46]. In this paper, we propose $ACE_{v2}$ - an extended version of ACE which offers a better alignment with the inference cost of quantized models and their energy consumption on ML hardware. Moreover, we introduce PikeLPN[1], a model that addresses these efficiency issues by applying quantization to both elementwise operations and multiply-accumulate operations. In particular, we present a novel quantization technique for batch normalization layers named QuantNorm which allows for quantizing the batch normalization parameters without compromising the model performance. Additionally, we propose applying Double Quantization where the quantization scaling parameters are quantized. Furthermore, we recognize and resolve the issue of distribution mismatch in Separable Convolution layers by introducing Distribution-Heterogeneous Quantization which enables quantizing them to low-precision. PikeLPN achieves Pareto-optimality in efficiency-accuracy trade-off with up to 3× efficiency improvement compared to SOTA low-precision models.*

## 1. Introduction

Quantization has long been established as a method to decrease the precision of neural network weights and activations effectively, resulting in smaller models and accelerated processing [11]. Recent studies have shown impressive results in image classification tasks, making the use of low-precision quantization (i.e., 4 bits or fewer) increasingly popular [28, 33, 34, 46]. In these compact models, convolutional and fully connected layers are typically constrained to 4-bit precision or even less, while precision is maintained at higher levels in other layers of the network. For example, the state-of-the-art (SOTA) binary network PokeBNN [46] binarizes the convolutional layers of ResNet-50 [15], and to avoid accuracy loss, they incorporate extra skip connections, extra batch normalization layers, and parameterized activation functions (DPReLU) that are executed in high precision. As illustrated in Figure 2, while this strategy significantly reduces the cost of multiply-accumulate (MAC) operations, it shifts the energy burden to the elementwise operations within these remaining high-precision layers. Although there are fewer of these elementwise operations, they use more energy because they are still in high precision. This indicates a critical area of optimization to improve the overall efficiency of low-precision models.

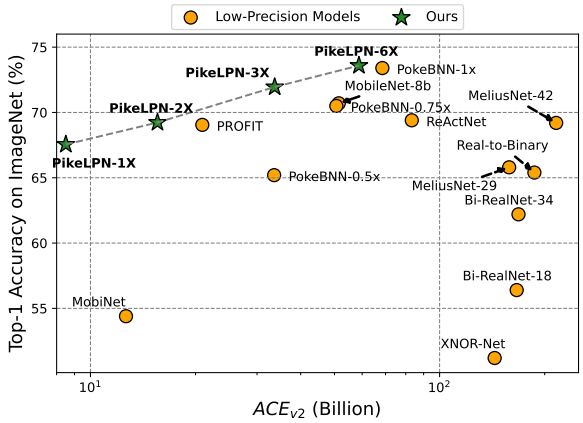



Figure 1. Accuracy vs $ACE_{v2}$ of PikeLPN and SOTA low-precision neural networks. $ACE_{v2}$ is an efficiency metric that estimates the cost of arithmetic operations during inference.

[*]Work done during internship at Google.

[†]Corresponding authors: marina_neseem@brown.edu and daniele-moro@google.com

[1]**Pike** is a slim fast fish, **LPN** stands for Low-Precision Network.

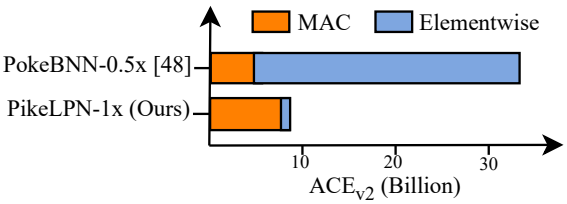



Figure 2. Contribution of multiply-accumulate (MAC) versus elementwise operations to the commonly used efficiency metric $ACE_{v2}$ for PikeLPN-1X and PokeBNN-0.5X [46]. *PikeLPN* selectively increases the precision of MAC operations which allows for effectively quantizing elementwise operations, achieving 3× more efficiency while being 2% more accurate on ImageNet.

We analyze the key efficiency bottlenecks in low-precision models uncovering a fundamental limitation of the efficiency metrics in literature, ACE [46], CPU64 [28, 30], Unit-gate model [47] and FA-count [37]. Those metrics exclude the elementwise operations in arithmetic calculations, a sentiment grounded in the belief that their contribution to the total computation cost is negligible compared to MAC operations. Optimizing for those metrics drives researchers to prioritize the reduction of computational precision in Convolutional and Dense layers, yet they overlook the quantization of elementwise operations. As a result, operations such as batch normalization, activation functions, and quantization scaling multiplications, are often performed at full precision. Moreover, SOTA low-precision models tend to rely extensively on mechanisms like branching [18] and skip connections [15], which significantly increase energy costs associated with memory reads and writes. To overcome this issue, we propose $ACE_{v2}$ which extends the efficiency metric *ACE* to account for all arithmetic operations in quantized neural networks including both elementwise and MAC operations. This would help guide researchers' choices when designing low-precision models.

Guided by our $ACE_{v2}$ metric, we design *PikeLPN* – a novel family of efficient low-precision models. *PikeLPN* quantizes both elementwise and MAC operations. Remarkably, *PikeLPN* not only achieves a 3× cost reduction compared to SOTA binary models [28, 46], it also achieves competitive accuracy levels on ImageNet [10].

**Our contributions** can be summarized as follows:

- We identify and analyze the overlooked cost of non-quantized elementwise operations in SOTA low-precision models. Our analysis shows that the non-quantized elementwise operations used in parameterized activation functions, batch normalization, and quantization scaling dominate the inference cost of low-precision models.
- We propose $ACE_{v2}$ – an extension to the existing hardware-agnostic cost metric *ACE*. $ACE_{v2}$ offers a better alignment with the cost of the low-precision models and their energy consumption on ML hardware by accounting for all arithmetic operations during inference.
- We propose *PikeLPN* – a novel family of low-precision architectures, which improves the efficiency of low-precision models by quantizing both elementwise and multiply-accumulate operations. Specifically, we propose (a) *QuantNorm* for effective batch normalization quantization, (b) *Double Quantization* where quantization parameters are also quantized, and (c) *Distribution-Heterogeneous Quantization* for Separable Convolution layers to tackle their distribution mismatch problem.

The rest of the paper is organized as follows. We review the related work in Section 2. In Section 3, we propose $ACE_{v2}$ providing detailed analysis to the overlooked efficiency bottlenecks by previous cost metrics. Then, guided by the new cost metric, we propose our efficient *PikeLPN* model. Next, we compare PikeLPN to SOTA low-precision models in Section 4. Finally, we conclude in Section 5.

## 2. Related Work

**Low-precision Quantization:** A substantial body of work exists in the realm of low-precision quantization, exemplified by studies that indicate that architectures can be quantized to 4 bits with minimal impact on accuracy [1, 5, 23, 33]. Others perform logarithmic quantization methods known for their hardware efficiency [12, 26, 40]. In addition, there are attempts to push the boundaries by introducing predominantly binary models where some of the convolution layers are quantized to 1 bit while other layers are maintained at a higher precision [28, 35, 46]. Some researchers have also developed automated strategies for mixed-precision modeling to dynamically choose the optimal precision for each layer, contingent upon a predetermined efficiency metric [24]. However, existing approaches primarily focus on the quantization of multiply-accumulate (MAC) operations in convolution and dense layers. They commonly neglect elementwise operations such as those in batch normalization layers and activation functions. Our empirical findings show that this assumption becomes invalid for low-precision models, specifically 4 bits or below.

**Architectural Approaches to Low-precision Models:** Several studies have adopted architectural modifications to enhance the performance of low-precision models. Many such modifications involve the integration of modules consisting solely of elementwise operations, aiming to minimize computational and parameter overhead. For instance, the channelwise real-valued rescaling of binarized tensors has been proposed as an effective means to reduce quantization error [36]. This approach incorporates elementwise floating-point multiplications for each channel. Additional methods, as suggested in [9], advocate for per-vector quantization, which results in multiple elementwise multiplications per channel. Studies like FracBNN [45] and PokeBNN [46] include extra Batch Normalization layers in

their predominantly binary models to expedite the training convergence. Moreover, the use of parameterized activation functions, such as PReLU [14] and DPReLU [46], has become a standard practice for improving the performance of low-precision models [28, 29]. All these modifications necessitate elementwise floating-point multiplications and additions. Moreover, the introduction of skip connections has proven beneficial in enhancing low-precision model quality. Notably, ReActNet [28] and PokeBNN [46] are designed with 4 and 3 parallel branches, respectively. Although skip connections only involve elementwise additions, they contribute to an increased memory access during inference to store multiple activations increasing the inference cost [21].

**Cost Metrics for Efficiency Evaluation:** MAC operations have been recognized in literature as the principal contributors to inference cost of deep learning models. As a result, efficiency metrics have predominantly focused on these specific operations. The *CPU64* metric [27–29] has been used to gauge the efficiency of mixed-precision neural networks when running on CPUs. With the growing utilization of specialized machine learning hardware and accelerators, a newer metric named *ACE* has been introduced [46]. *ACE*, an acronym for *Arithmetic Computation Effort*, is formulated as the product of the number of MAC operations and the bitwidth of the two operands involved, which is directly proportional to the number of active hardware bit-adders required. The Unit-gate model [47] and FA-count [37] correlate very well with ACE and differ only by a small constant factor [2]. All these metrics do not consider elementwise operations. Thus, in this paper, we extend the ACE metric introducing $ACE_{v2}$, and this extension should generalize to other metrics as well. All these metrics, including the extended ACE, are technology node independent.

## 3. Method

In this section, we identify previously overlooked costs in state-of-the-art (SOTA) cost metrics. Additionally, we propose extending the *Arithmetic Computational Effort* (ACE) metric [46] to provide a more accurate representation of the inference cost of low-precision models. Subsequently, we assess the impact of various design alternatives in low-precision models on the cost of inference. Finally, we present *PikeLPN* – a novel family of low-precision models.

### 3.1. Cost Metrics for Low Precision Models

The prevalent notion is that multiply-accumulate operations in the convolution and dense layers are the sole substantial contributors to inference cost in deep learning models [28, 33, 46]. This viewpoint stems from the observation that for full precision models the energy cost of those layers is more than 95% of the total model operations as shown in Figure 3. Consequently, commonly used efficiency metrics for quantized neural networks, such as *CPU64* [27–29] and *ACE* [46], are tailored to exclusively account for multiply-accumulate operations in these specified layers. Optimization in accordance with these metrics drive researchers to prioritize reducing the precision of multiply-accumulate operations in convolution and dense layers while maintaining high precision for all other elementwise operations. Moreover, they re-parameterize the models adding layers that only have elementwise operations to compensate for any accuracy losses by low-precision quantization [28, 46]. However, our analysis reveals that these non-quantized elementwise operations substantially contributes to the arithmetic cost during inference of low-precision models (i.e., 8 bits and lower), thereby challenging the prevailing assumptions.

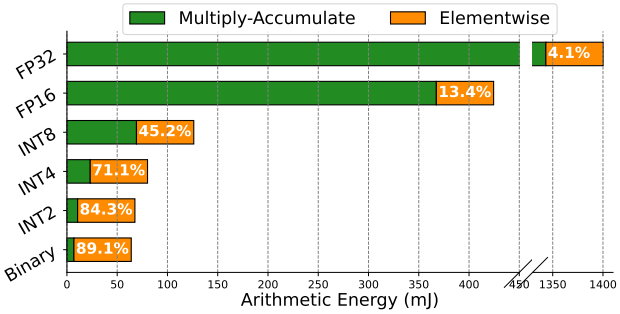



Figure 3. Arithmetic Energy on 45nm CMOS technology by multiply-accumulate operations versus non-quantized elementwise operations for MobileNetV2. Energy costs are calculated using Table 1. The figure reveals that elementwise operations are a substantial contributor to the overall cost in low-precision models.

Figure 3 illustrates the relative contributions of low-precision multiply-accumulate operations and non-quantized elementwise operations to the total energy consumption by arithmetic computations at various precisions. The data reveals a notable trend: the proportion of energy consumed by elementwise operations becomes more significant as the precision decreases. For example, in binary-quantized models, those non-quantized elementwise operations account for up to 89% of the total cost. This observation highlights the limitations of existing metrics in accurately gauging the efficiency of quantized models. Consequently, we propose $ACE_{v2}$ which extends the *ACE* metric [46] to account for both multiply-accumulate operations as well as elementwise operations. We anticipate that our comprehensive $ACE_{v2}$ metric will enable more informed optimization choices within the research community.

### 3.2. Introducing $ACE_{v2}$

*ACE* has been used to estimate the cost of inference on idealized ML hardware implemented with CMOS methodology [46]. *ACE* is defined by its authors as the number

[2] They do not account for carry-save format for local accumulator representations typically used in systollic arrays.

Table 1. Cost under 45nm CMOS technology [16, 44] [3]. $f(i,j)$ refers to the formula used to calculate the $ACE_{v2}$ cost where $i$ and $j$ are the precisions of the two operands. $c_a = 6$ and $c_s = 5$. The correlation coefficient between $ACE_{v2}$ and the independently measured arithmetic energy consumption is 0.991.

| | MULTIPLY | | ADD | | SHIFT | |
|---|---|---|---|---|---|---|
| | **Energy** $(pJ)$ | $\boldsymbol{ACE_{v2}}$ | **Energy** $(pJ)$ | $\boldsymbol{ACE_{v2}}$ | **Energy** $(pJ)$ | $\boldsymbol{ACE_{v2}}$ |
| FP32 | 3.7 | 992 | 0.9 | 192 | - | - |
| FP16 | 1.1 | 240 | 0.4 | 96 | - | - |
| $f(i,j)$ | $i \cdot j$ - $max(i,j)$ | | $c_a \cdot max(i,j)$ | | - | |
| INT32 | 3.1 | 992 | 0.1 | 32 | 0.13 | 32 |
| INT16 | - | 240 | - | 16 | 0.057 | 12.8 |
| INT8 | 0.2 | 56 | 0.03 | 8 | 0.024 | 4.8 |
| INT4 | - | 12 | - | 4 | - | 1.6 |
| INT2 | - | 2 | - | 2 | - | 0.4 |
| Binary | - | - | - | 1 | - | - |
| $f(i,j)$ | $i \cdot j$ - $max(i,j)$ | | $max(i,j)$ | | $i \cdot log_2(j)/c_s$ | |

of bitadders (i.e., digital circuit adding 3 bits to form a 2 bit number – carry and sum) required to perform every multiply-accumulate operation. The authors justify that definition by showing a high correlation coefficient (i.e., 0.946) between the number of bitadders and the independently measured energy consumption on 45nm CMOS technology. While *ACE* provides a hardware-agnostic method to evaluate the efficiency of quantized neural networks, it fails to include the elementwise operations which can be the dominating cost factor in low precision models as shown in Figure 3. Moreover, *ACE* does not provide a way to estimate the cost of shift operations which are required to implement non-linear base-2 logarithmic quantization [43, 44]. We propose $ACE_{v2}$ which improves *ACE* by extending it to include elementwise multiplication, elementwise addition, and shift operations. We establish the $ACE_{v2}$ formulas for the previously discussed operations as shown in Table 1.

**Elementwise Multiplications:** Using established methods for constructing multipliers, such as adder trees proposed by Wallace and Dadda [8, 42], we calculated the number of adders needed to multiply an $i$-bit number by a $j$-bit number as $i \cdot j - max(i,j)$. This formula exactly matches the optimal number of adders for $1 <= i, j <= 64$. See Section 6 in the Appendix for a detailed explanation.

**Elementwise Additions:** Fixed-point numbers added using established adders [4] activate an upper bound of $max(i,j)$ bit adders to add i-bit and j-bit numbers. Floating-point adders additionally require exponent alignment, significand addition, and normalization steps [38], resulting in a much higher energy consumption compared to fixed-point adders as shown in Table 1. We analyze the operations needed in floating point adders [38] and come to an $ACE_{v2}$ cost of $6\times$ the cost of a fixed-point adder. Therefore, we derive $ACE_{v2}$ for floating point adders using $c_a \cdot max(i,j)$ with $c_a = 6$. See Appendix Section 7 for a detailed explanation.

**Shift Operations:** A Barrel Shifter is an established method to shift and rotate $i$-bit numbers by $j$ locations in modern processors [13]. The barrel shifter is implemented as a cascade of $i \log_2(j)$ *2:1* multiplexers. Therefore, we derive $ACE_{v2}$ for a shift operation as $i \log_2(j)/c_s$ where $c_s$ is the ratio of the cost of a *2:1* multiplexer compared to a full adder. Since a full adder can be efficiently implemented using five *2:1* multiplexers based on [22], we assign $c_s = 5$.

To verify the correctness of our $ACE_{v2}$ metric, Table 1 shows a 0.991 correlation coefficient between the *independently* measured energy consumption of various arithmetic units on the 45nm CMOS technology and its $ACE_{v2}$ cost, a notable improvement compared to the 0.946 correlation coefficient in $ACE$ [46]. Using those definitions, we estimate a more accurate arithmetic cost for any quantized model.

[3] Energy costs for low-precision operations can be extrapolated linearly for addition and quadratically for multiplication [6].

[4] While there are many methods for constructing adders, such as Carry Lookahead Adder [32] and Ripple Carry Adder [3], the particular implementation has a limited effect on the energy use.

Table 2. The contribution of non-quantized Batch Normalization Layers to the overall $ACE_{v2}$ cost.

| **Model** | **BN Adds** $(Million)$ | **BN Mults** $(Million)$ | **BN** $\boldsymbol{ACE_{v2}}$ (%) |
|---|---|---|---|
| MobileNetV2 $(4W, 4A)$ | 6.67 | 6.67 | **41.87** |
| ResNet50 $(1W, 1A)$ | 10.58 | 10.58 | **41.38** |

### 3.3. Overlooked Efficiency Bottlenecks

**Batch Normalization:** Batch normalization layers, which necessitate elementwise multiplications and additions, typically retain parameters in floating-point format during deep neural network quantization to maintain training stability and prevent accuracy loss [28, 35, 46]. Consequently, these operations are performed using floating-point (FP32) arithmetic, with a single FP32 operation consuming approximately $18\times$ more energy than an INT8 multiplication, as detailed in Table 1. Assessing the impact of these non-quantized batch normalization layers in Table 2 reveals that they can account for as much as 42% of the total $ACE_{v2}$ cost in various low-precision models. This substantial contribution shows the importance of considering the cost of these operations and potentially quantizing its parameters.

**Activation Layers:** In recent literature, low-precision models have increasingly replaced ReLU [2] activation functions with parameterized activation functions such as PReLU [14] and DPReLU [46] to improve performance and training stability of quantized models [28, 34]. The dynamic parameterized rectified linear unit (DPReLU), for instance, is defined by the following piecewise function:

$$DPReLU(x) = \begin{cases} \eta(x-\alpha)-\beta & \text{if } x-\alpha > 0 \\ \gamma(x-\alpha)-\beta & \text{otherwise} \end{cases} \quad (1)$$

Here, the parameters $\eta$, $\alpha$, $\beta$, and $\gamma$ are represented in floating-point format. Consequently, the computation of

Table 3. The contribution of non-quantized parameterized activation functions to the overall $ACE_{v2}$ cost. Analysis performed by applying different activation functions to a 4-bit MobileNetv2.

| **Activation** | **Adds** $(Million)$ | **Mults** $(Million)$ | $\boldsymbol{ACE_{v2}}$ $(\times 10^9)$ | **Overhead** (%) |
|---|---|---|---|---|
| ReLU [2] | 0 | 0 | 20.44 | - |
| PReLU [14] | 0 | 6.1 | 26.5 | **+29.6%** |
| DPReLU [46] | 6.1 | 6.1 | 27.67 | **+35.3%** |

DPReLU necessitates both elementwise floating-point multiplications and additions. Our study, detailed in Table 3, assesses the impact of these elementwise operations on the $ACE_{v2}$ cost. We find that in a 4-bit MobileNetV2 model, the incorporation of different activation functions — namely ReLU, PReLU, and DPReLU — significantly influences the cost. Specifically, the use of PReLU and DPReLU, despite their benefits on accuracy, introduces up to 35% increase in the overall inference cost. This finding highlights the need to balance the benefits of parameterized activation functions with their computational demands.

**Skip Connections:** Skip connections are regarded as zero-cost operations in terms of arithmetic computation. Consequently, previous work overused them to improve the model performance without having any measurable effect on the cost [28, 35, 46]. For instance, ReActNet [28] incorporated four parallel branches, quadrupling its memory footprint compared to a single-path model. PokeBNN [46] followed a similar design, incorporating three parallel branches. However, such branching necessitates the concatenation of feature maps from previous layers, leading to an increase in the amount of data concurrently stored in memory. That increase the required memory reads and writes which have significant costs. As an example, in a processor with a 32KB cache designed using 45nm CMOS technology, moving an 8-bit element from the cache consumes approximately $2.5pJ$ of energy. This is about $12\times$ the energy needed for an INT8 multiplication operation, which requires only around $0.2pJ$ as shown in Table 1. This disparity becomes even more profound when data must be transfered from DRAM, where the energy requirement balloon to $162.5pJ$ – $810\times$ higher than the INT8 multiplication [16]. Quantifying this overhead in a hardware-agnostic manner is challenging since it is influenced by a multitude of factors including the underlying hardware architecture, memory location, and model size. Yet, understanding its impact remains crucial to design efficient models. We advocate for the adoption of *Arithmetic Intensity* as a practical metric to measure memory reads and writes during inference [21]. Arithmetic Intensity ($AI_c$) is defined as the ratio of the arithmetic operations ($M_c$) to the amount of data, including both Weights ($W$) and Activations ($A$), required to execute these operations as shown in Equation 2.

$$AI_c = \frac{M_c}{W + A} \tag{2}$$

Table 4. Arithmetic Intensity computed according to Equation (3) for a ResNet-50 model with various number of branches.

| **Arithmetic Intensity** (Ops/Element ↑) | | |
|---|---|---|
| 2 Branches | 3 Branches | 4 Branches |
| **73.5** | 49.66 | 36.75 |

Table 5. $ACE_{v2}$ of a 4-bit MobileNetV2 and a binary ResNet50 model with various quantization granularities. The *Overhead* represents the percentage of cost required by the extra FP operations due to quantization (i.e. quantization scaling).

| **Quantization Granularity** | **Mults** $(Million)$ | $\boldsymbol{ACE_{v2}}$ $(\times 10^9)\downarrow$ Total | Overhead (%) |
|---|---|---|---|
| **MobileNetV2 -** $< 4W, 4A >$ | | | |
| Layerwise [11] | 6.67 | 20.44 | **32.52%** |
| Channelwise [11] | 6.67 | 20.44 | **32.52%** |
| Sub-Channelwise [9] | 13.35 | 27.06 | **48.97%** |
| **ResNet50 -** $< 1W, 1A >$ | | | |
| Layerwise [11] | 10.63 | 28.13 | **32.03%** |
| Channelwise [11] | 10.63 | 28.13 | **32.03%** |
| Sub-Channelwise [9] | 32.75 | 50.08 | **63.55%** |

Consequently, Arithmetic Intensity serves as an indicator of the amount of memory reads and writes to perform computational operations. Adding branches lead to a substantial increase in the amount of data that must be loaded to execute a relatively small number of operations; hence decreasing the arithmetic intensity as shown in Table 4.

**Quantization Granularity Overhead:** Uniform quantization, a widely adopted technique in SOTA low-precision models [33, 35, 46], transforms discrete integer values, $q$, into continuous real values, $r$ through the affine relation

$$r = S(q - Z) \tag{3}$$

where $S$ is a scale factor. $S$ is a critical component of quantization which is typically learned as an arbitrary floating-point value during training. In the inference phase, this necessitates an elementwise multiplication by $S$, contributing to computational overhead [20]. Proper scaling is crucial in quantization to mitigate quantization error enabling quantized models to maintain high accuracy. Quantization granularity dictates the level at which scaling factors are applied in a model [11]. For example, Layerwise quantization assigns a single scale factor based on all weights within a layer. Channelwise quantization, widely adopted in state-of-the-art low-precision models, allocates a unique scaling factor to each channel, catering to the varying distributions of weights and potentially enhancing model accuracy. Sub-Channelwise quantization takes this further by assigning several scaling factors within each channel, allowing for even finer adjustments at the expense of increased computational cost [9]. All quantization granularities add one or more elementwise multiplications per channel. Table 5 compares the $ACE_{v2}$ cost of such quantization granularities. In the popular Channelwise quantization, the overhead from elementwise multiplications is 32% of the total cost.

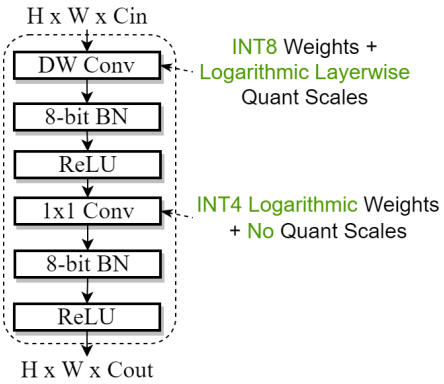



Figure 4. *PikeLPN* building block architecture.

Table 6. Top-1 Accuracy on ImageNet vs $ACE_{v2}$ cost of PikeLPN using various quantizers for the Depthwise and Pointwise Layers. PW-Convolution layers contribute to 95% of the number of multiply-accumulate operations in the model, that is why we lower the precision of the PW Conv layers to 4 bits while we keep the DW Conv layers at 8-bits.

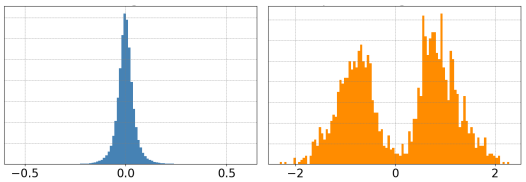
| **Pointwise Conv.** | | **Depthwise Convs** | | **Top-1** | $\boldsymbol{ACE_{v2}}$ |
|---|---|---|---|---|---|
| Weights | Q-Params | Weights | Q-Params | (%) | ($\times 10^9$) |
| Linear-4 | Arbitrary | Linear-8 | Arbitrary | 68.50 | 20.91 |
| Linear-4 | PoT | Linear-8 | PoT | 68.41 | 15.93 |
| PoT-4 | - | PoT-8 | - | 64.50 | 10.05 |
| PoT-4 | - | Linear-8 | Arbitrary | 67.60 | 12.86 |
| PoT-4 | - | Linear-8 | PoT | **67.55** | **10.95** |

### 3.4. PikeLPN Architecture

Based on our comprehensive analysis, we introduce *PikeLPN*, a novel architecture engineered to mitigate the inefficiencies of SOTA low-precision models. This section introduces the basic block of our proposed *PikeLPN* model, explores quantization strategies for the different layers, and proposes a novel method for quantizing batch normalization layers without compromising the model's accuracy.

**PikeLPN Basic Block:** To engineer an effective low-precision model, we first design the baseline architecture with building blocks that are inherently efficient. With this principle in mind, our architecture adopts separable convolutional layers, subdivided into depthwise and pointwise convolutions, in line with the framework established by MobileNetV1 [17]. Those layers are widely recognized for their computational efficiency and have been integrated into SOTA efficient ConvNets [39, 41]. Figure 4 illustrates the building block for *PikeLPN*. To maximize computational efficiency, the used architecture deliberately avoids parameterized activation functions and skip connections that are likely to increase computational cost as explained in Subsection 3.3. Finally, our model uses the first and last blocks from the MobileNetV1 architecture due to their proven effectiveness and reliability.

**Quantizing Separable Convolution Layers:** Linear quantizers results in a set of equally spaced values since they use affine mapping as shown in Equation 3. Non-uniform quantizers have different constraints. For example, Power-of-two (PoT) [31] restrict quantization levels to be powers-of-two values. They can be used to increase the representational density of small values, furthermore, they have the benefit of replacing the multiplication operations during inference with shifts which are significantly cheaper as shown in Table 1. However, using PoT quantizers for both pointwise (PW) and depthwise (DW) convolution operations in the separable convolution block leads to significant accuracy degradation as shown in the third row of Table 6. To get some insights, we analyze the distribution of the full-precision weights of PikeLPN when pre-trained on ImageNet. Figures 5(a) and 5(b) visualize the distributions of a sample PW and DW weights respectively. Interestingly, the majority of the weights of the PW layer lie around $\pm 0.1$, while the weights in the DW layer are distributed around $\pm 2$. This mismatch in weights distribution across different layers makes low-precision quantization for the separable convolution blocks challenging because the used values fail to capture both distributions. To address this problem, we propose using *Distribution Heterogeneous Quantization* where the pointwise weights use the more efficient PoT quantizer while the depthwise weights use a linear quantizer. It is important to note that pointwise convolutions contribute to 95% of the number of multiply-accumulate operations in *PikeLPN*; hence using the PoT quantizer in pointwise layers only improves the model's efficiency by 50% as shown in Table 6.




(a) (b)

Figure 5. Weights distribution of pre-trained PW and DW Convolution layers in PikeLPN where (a) Sample Pointwise layer weights (b) Sample Depthwise layer weights.

**Double Quantization:** Quantization requires extra elementwise multiplications by a floating-point scaling factor which add significant overhead as shown in Table 5. While we can not completely remove the scale factor, we can reduce the overhead from quantization scale multiplications by quantizing those quantization parameters. We refer to quantizing the quantization parameters as *Double Quantization*. We consider using a PoT scale for the linear depthwise quantizer in *PikeLPN* which can potentially reduce the elementwise operation from $3.7mJ$ to $0.13mJ$ based on Table 1. Our experiments indicates negligible effect on accuracy when applying *Double Quantization* as shown in Table 6.

**Quantizing Batch Norm Layers:** Batch normalization layers are used in most modern deep learning models to

stabilize the training and improve their performance [19]. Batch normalization is computed as follows:

$$batchnorm(x) = \frac{(x - \mu) * \gamma}{\sqrt{\sigma^2 + \epsilon}} + \beta \quad (4)$$

Where $x$ is the input feature map and the batch norm parameters $\mu$, $\gamma$, $\sigma$, $\beta$ are represented as floating-point values. To avoid performing floating point multiplications and additions, those parameters need to be quantized as follows:

$$Qbatchnorm(x) = \frac{(x - Q(\mu)) * Q(\gamma)}{\sqrt{Q(\sigma)^2 + \epsilon}} + Q(\beta) \quad (5)$$

Computation folding is a commonly used approach to reduce the overhead of batch normalization operations in quantized models (i.e., mainly in 8 bit models) [20]. However, the batch normalization parameters (i.e., $\mu$, $\gamma$, $\sigma$, and $\beta$) have to be quantized to the same precision of the preceding convolution layers to enable folding. Doing that in low-precision models (i.e., 4 bits or lower) leads to a significant loss in accuracy as shown in Figure 6. That is why previous low-precision model research [33, 35, 46] excluded batch normalization layers from the quantization process, where they keep the batch norm parameters as floating point numbers. However, as we showed earlier in Table 2, the non-quantized batch normalization operations can add up to 40% overhead to the model's $ACE_{v2}$ cost.

Another solution is to quantize the batch normalization parameters at a higher precision. Figure 6 shows the validation accuracy curve during training when batch normalization parameters are represented as INT8 values (denoted as *8-bit Vanilla BN*). Although the accuracy is better than the folded batch norm, we can still notice some degradation in accuracy compared to non-quantized batch norm layers. To minimize the accuracy loss, we propose a novel *QuantNorm* layer. In our *QuantNorm* layer, we re-write the batch norm quantization operation as shown in Equation 6 where we first multiply by a quantized scale $s$, then add a quantized bias $b$. $s$ is represented as the quantized division between the $\gamma$ and $\sigma$ parameters as shown in Equation 7. Using *QuantNorm* helps reduce quantization error by allowing high precision division in the scale $s$ computation during training. As shown in Figure 6, our *QuantNorm* layer maintains close-to-FP accuracy without any extra costs compared to vanilla quantization for batch norm layer. After training, we pre-compute $s$ to avoid high precision division during inference.

$$Qbatchnorm(x)_{improved} = x * s - b \quad (6)$$

$$s = Q(\frac{\gamma}{\sqrt{\sigma^2 + \epsilon}}) \quad (7)$$

$$b = Q(\beta) - Q(\mu) * s \quad (8)$$

**Model Scaling:** To generate a Pareto family of models, we scale the number of output channels as practiced in the MobileNetV1 model [17]. We also scale the precision of the input activation to the pointwise convolution layers in the PikeLPN block. We show more details about scaling *PikeLPN* in Appendix Section 8.

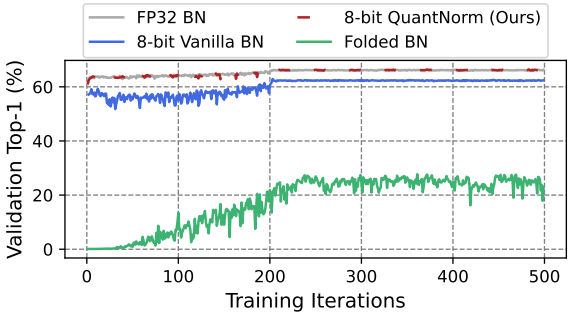



Figure 6. Validation Top-1 Accuracy during QAT on ImageNet for different Batch Norm Quantization techniques.

## 4. Experiments

### 4.1. Implementation and Training

All models are implemented using QKeras [7], then we performed Quantization-aware training (QAT) [20]. We train and evaluate the *PikeLPN* family of models on the ILSVRC12 ImageNet classification dataset [10]. To train our low-precision models, we follow a multi-phase training approach. We first train the full-precision model, then we quantize the model as explained previously in Subsection 3.4, and train for another 500 epochs. All Models are trained with an effective batch size of 256 using an *AdamW* optimizer and a Cosine Decay schedule. We use label smoothing regularization with cross-entropy loss and a smoothing factor of 0.1 for all models. The initial learning rate is $1e-4$ and annealed using a cosine schedule to $1e-12$. An interesting observation was that training for the final 100 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. More details and visualization about this behaviour in added in the Appendix. We use standard augmentation techniques like resizing, cropping, and flipping. At test time, all *PikeLPN* models are evaluated on images of resolution $224 \times 224$.

### 4.2. Results

To evaluate the accuracy-efficiency trade-off by *PikeLPN*, we compare its performance to state-of-the-art low-precision models. Figures 7 and 1 show that *PikeLPN* establishes the SOTA Pareto frontier for low-precision and binary models in terms of arithmetic energy consumption and $ACE_{v2}$ cost respectively. Table 7 compares *PikeLPN* to SOTA low-precision models in terms of Top-1 Accuracy on ImageNet, Energy consumption in $millijoules$, $ACE_{v2}$, and Arithmetic Intensity. We clearly see how the elementwise operations dominate (i.e., 31-93%) the

Table 7. Results – *PikeLPN* versus SOTA low-precision models in terms of Accuracy and Efficiency Metrics. $ACE_{v2}$ is measured according to the definition in Section 3.2. The fourth and fifth columns show the contribution to the overall $ACE_{v2}$ cost by multiply-accumulate and elementwise operations respectively. *Energy* represents the arithmetic energy according to $45nm$ CMOS technology according to table 1. *Arithmetic Intensity* is an indication for the memory reads and writes required by the model as explained in Section 3.3. *Used Precisions* represent the the precision of the various operations in the mixed-precision models.

| **Model** | **Accuracy** (%) | **Arithmetic Computational Effort ($ACE_{v2}$)** Total ($\times 10^9 \downarrow$) | MAC (%) | Elementwise (%) | **Energy** ($mJ \downarrow$) | **Arithmetic Intensity** (Ops/Element $\uparrow$) | **Used Precisions** |
|---|---|---|---|---|---|---|---|
| XNOR-Net [36] | 51.2 | 143.78 | - | - | 587.69 | - | 32, 1 |
| MobiNet [34] | 54.4 | 12.64 | 13.17 | 86.83 | 50.66 | 28 | - |
| Bi-RealNet-18 [27] | 56.4 | 166.26 | - | - | 678.75 | - | 32, 1 |
| Bi-RealNet-34 [27] | 62.2 | 168.11 | - | - | 691.47 | - | 32, 1 |
| MobileNet (8W, 4A) [25] | 64.0 | 33.8 | 68.96 | 31.04 | 118.54 | 39.57 | 32, 8, 4 |
| MobileNet (4W, 8A) [25] | 65.0 | 33.8 | 68.96 | 31.04 | 118.54 | 39.57 | 32, 8, 4 |
| Real-to-Binary Net [30] | 65.4 | 186.85 | - | - | 762.24 | - | 32, 1 |
| MeliusNet-29 [4] | 65.8 | 158.21 | - | - | 656.81 | - | 32, 1 |
| PokeBNN-0.5x [46] | 65.2 | 33.58 | 4.18 | 95.81 | 143.78 | 24.5 | 32, 8, 4, 1 |
| **PikeLPN-1**$\times$ (Ours) | **67.55** | **8.50** | 96.38 | 3.62 | **34.98** | 39.57 | 8, 4 |
| PROFIT [33] | 69.05 | 20.91 | 47.51 | 52.49 | 82.70 | 39.57 | 32, 4 |
| MeliusNet-42 [4] | 69.20 | 215.71 | - | - | 901.82 | - | 32, 1 |
| **PikeLPN-2**$\times$ (Ours) | **69.23** | **15.56** | 97.87 | 2.13 | **64.20** | 39.57 | 16, 8, 4 |
| ReActNet [28] | 69.4 | 83.24 | 26.78 | 73.22 | 361.63 | 36.75 | 32, 1 |
| PokeBNN-0.75x [46] | 70.5 | 50.61 | 5.11 | 94.88 | 218.51 | 40.48 | 32, 8, 4, 1 |
| MobileNet (8bit) [25] | 70.7 | 51.44 | 79.61 | 20.39 | 173.68 | 39.57 | 32, 8 |
| **PikeLPN-3**$\times$ (Ours) | **71.95** | **33.70** | 98.52 | 1.48 | **139.59** | **52.66** | 16, 8, 4 |
| PokeBNN-1x [46] | 73.4 | 68.56 | 6.16 | 93.83 | 298.44 | 40.48 | 32, 8, 4, 1 |
| **PikeLPN-6**$\times$ (Ours) | **73.59** | **58.74** | 98.87 | 1.13 | **243.85** | **63.38** | 16, 8, 4 |

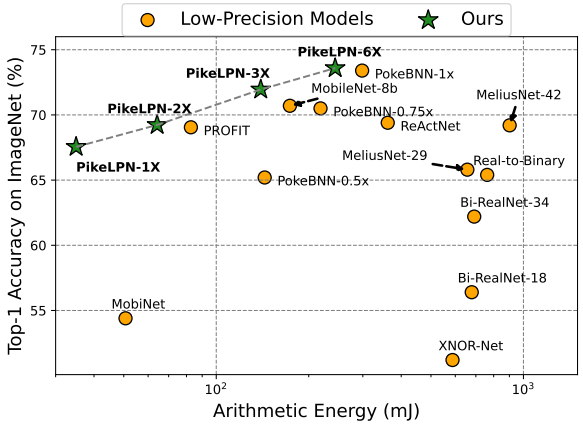



Figure 7. Accuracy and Energy Consumption by the arithmetic operations of our *PikeLPN* vs SOTA low-precision neural networks.

$ACE_{v2}$ cost for other low-precision models. On the other hand, *PikeLPN* carefully quantizes the elementwise operations reducing their contribution to the total energy consumption to less than 5%. Additionally, *PikeLPN*-1$\times$ is 1.5$\times$ more efficient in terms of both $ACE_{v2}$ and arithmetic energy consumption compared to *MobiNet* [35] (i.e., A binary version of MobileNetV1 with added skip connections) while achieving **13.2%** higher Top-1 Accuracy on ImageNet. Moreover, *PikeLPN*-3$\times$ achieves 1.5% higher Top-1 accuracy than PokeBNN-0.75$\times$ [46] (i.e., A binary ResNet-50 with parameterized activation functions) while being 35% more efficient. In terms of arithmetic intensity, *PikeLPN* shows a much higher arithmetic intensity when compared to other low-precision models, this is mainly due to the absence of any skip connections. As mentioned earlier in Section 3.3, high arithmetic intensity is advantageous as it suggests a greater proportion of computational operations per data element, which can lead to reducing the memory reads and writes by the model; hence reducing the overall energy consumption during inference.

## 5. Conclusion

Our investigation into SOTA low-precision models uncovered overlooked efficiency bottlenecks, particularly noting that operations traditionally considered negligible—such as elementwise operations in activation functions, batch normalization, and quantization scaling can contribute up to 90% of the inference cost. Addressing these challenges, we proposed $ACE_{v2}$ which extends the efficiency metric *ACE* to better reflect the inference cost of low-precision models. Moreover, we introduced *PikeLPN*, a novel family of models that quantizes both elementwise and multiply-accumulate operations. Specifically, we propose (a) a novel *QuantNorm* layer for effective batch normalization quantization, (b) *Double Quantization* where quantization parameters are also quantized, and (c) *Distribution-Heterogeneous Quantization* for Separable Convolution layers to tackle their distribution mismatch problem. *PikeLPN* achieves up to a threefold reduction in inference cost over existing low-precision models while improving the Top-1 accuracy in ImageNet dataset.

## References


[1] AmirAli Abdolrashidi, Lisa Wang, Shivani Agrawal, Jonathan Malmaud, Oleg Rybakov, Chas Leichner, and Lukasz Lew. Pareto-optimal quantized resnet is mostly 4-bit. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3091–3099, 2021. 2

[2] Abien Fred Agarap. Deep learning using rectified linear units (relu). 2018. 4, 5

[3] 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. 4

[4] Joseph Bethge, Christian Bartz, Haojin Yang, Ying Chen, and Christoph Meinel. Meliusnet: Can binary neural networks achieve mobilenet-level accuracy? *arXiv preprint arXiv:2001.05936*, 2020. 8

[5] Yoni Choukroun, Eli Kravchik, Fan Yang, and Pavel Kisilev. Low-bit quantization of neural networks for efficient inference. In *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, pages 3009–3018. IEEE, 2019. 2

[6] Wesley Donald Chu. Wallace and dadda multipliers implemented using carry lookahead adders. 2013. 4

[7] Claudionor N Coelho Jr, Aki Kuusela, Shan Li, Hao Zhuang, Jennifer Ngadiuba, Thea Klaeboe Aarrestad, Vladimir Loncar, Maurizio Pierini, Adrian Alan Pol, and Sioni Summers. Automatic heterogeneous quantization of deep neural networks for low-latency inference on the edge for particle detectors. *Nature Machine Intelligence*, 3(8):675–686, 2021. 7

[8] Luigi Dadda. Some schemes for fast serial input multipliers. In *1983 IEEE 6th Symposium on Computer Arithmetic (ARITH)*, pages 52–59, 1983. 4

[9] Steve Dai, Rangha Venkatesan, Mark Ren, Brian Zimmer, William Dally, and Brucek Khailany. Vs-quant: Per-vector scaled quantization for accurate low-precision neural network inference. *Proceedings of Machine Learning and Systems*, 3:873–884, 2021. 2, 5

[10] 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, 7

[11] Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W Mahoney, and Kurt Keutzer. A survey of quantization methods for efficient neural network inference. In *Low-Power Computer Vision*, pages 291–326. Chapman and Hall/CRC, 2022. 1, 5

[12] Soheil Hashemi, Nicholas Anthony, Hokchhay Tann, R Iris Bahar, and Sherief Reda. Understanding the impact of precision quantization on the accuracy and energy of neural networks. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, pages 1474–1479. IEEE, 2017. 2

[13] Irina Hashmi and Hafiz Md. Hasan Babu. An efficient design of a reversible barrel shifter. In *2010 23rd International Conference on VLSI Design*, pages 93–98, 2010. 4

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015. 3, 4, 5

[15] 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. 1, 2

[16] Mark Horowitz. 1.1 computing's energy problem (and what we can do about it). In *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, pages 10–14, 2014. 4, 5

[17] 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. 6, 7

[18] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017. 2

[19] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. pages 448–456, 2015. 7

[20] 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. 5, 7

[21] Nandan Kumar Jha and Sparsh Mittal. Modeling data reuse in deep neural networks by taking data-types into cognizance. *IEEE Transactions on Computers*, 70(9):1526–1538, 2020. 3, 5

[22] Iosr Journals, B. Ananda Babu, Jamshid M. Basheer, and Abdelmoty .M. Abdeen. Power optimized multiplexer based 1 bit full adder cell using .18 µm cmos technology. 2015. 4

[23] Sangil Jung, Changyong Son, Seohyung Lee, Jinwoo Son, Jae-Joon Han, Youngjun Kwak, Sung Ju Hwang, and Changkyu Choi. Learning to quantize deep networks by optimizing quantization intervals with task loss. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4350–4359, 2019. 2

[24] I. Koryakovskiy, A. Yakovleva, V. Buchnev, T. Isaev, and G. Odinokikh. One-shot model for mixed-precision quantization. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7939–7949, Los Alamitos, CA, USA, 2023. IEEE Computer Society. 2

[25] Raghuraman Krishnamoorthi. Quantizing deep convolutional networks for efficient inference: A whitepaper. *arXiv preprint arXiv:1806.08342*, 2018. 8

[26] Yuhang Li, Xin Dong, and Wei Wang. Additive powers-of-two quantization: An efficient non-uniform discretization for neural networks. *arXiv preprint arXiv:1909.13144*, 2019. 2

[27] Zechun Liu, Baoyuan Wu, Wenhan Luo, Xin Yang, Wei Liu, and Kwang-Ting Cheng. Bi-real net: Enhancing the per-

formance of 1-bit cnns with improved representational capability and advanced training algorithm. In *Proceedings of the European conference on computer vision (ECCV)*, pages 722–737, 2018. 3, 8

[28] Zechun Liu, Zhiqiang Shen, Marios Savvides, and Kwang-Ting Cheng. Reactnet: Towards precise binary neural network with generalized activation functions. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XIV 16*, pages 143–159. Springer, 2020. 1, 2, 3, 4, 5, 8

[29] Zechun Liu, Zhiqiang Shen, Shichao Li, Koen Helwegen, Dong Huang, and Kwang-Ting Cheng. How do adam and training strategies help bnns optimization. In *International conference on machine learning*, pages 6936–6946. PMLR, 2021. 3

[30] Brais Martinez, Jing Yang, Adrian Bulat, and Georgios Tzimiropoulos. Training binary neural networks with real-to-binary convolutions. *arXiv preprint arXiv:2003.11535*, 2020. 2, 8

[31] Daisuke Miyashita, Edward H Lee, and Boris Murmann. Convolutional neural networks using logarithmic data representation. *arXiv preprint arXiv:1603.01025*, 2016. 6

[32] Yu-Ting Pai and Yu-Kumg Chen. The fastest carry lookahead adder. In *Proceedings. DELTA 2004. Second IEEE International Workshop on Electronic Design, Test and Applications*, pages 434–436, 2004. 4

[33] Eunhyeok Park and Sungjoo Yoo. Profit: A novel training method for sub-4-bit mobilenet models. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part VI 16*, pages 430–446. Springer, 2020. 1, 2, 3, 5, 7, 8

[34] Hai Phan, Yihui He, Marios Savvides, Zhiqiang Shen, et al. Mobinet: A mobile binary network for image classification. In *Proceedings of the IEEE/CVF winter conference on applications of computer vision*, pages 3453–3462, 2020. 1, 4, 8

[35] Hai Phan, Zechun Liu, Dang Huynh, Marios Savvides, Kwang-Ting Cheng, and Zhiqiang Shen. Binarizing mobilenet via evolution-based searching. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 13417–13426, 2020. 2, 4, 5, 7, 8

[36] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European conference on computer vision*, pages 525–542. Springer, 2016. 2, 8

[37] 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. 2, 3

[38] 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. 4

[39] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pages 6105–6114. PMLR, 2019. 6

[40] Hokchhay Tann, Soheil Hashemi, R. Iris Bahar, and Sherief Reda. Hardware-software codesign of accurate, multiplier-free deep neural networks. In *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, 2017. 2

[41] Pavan Kumar Anasosalu Vasu, James Gabriel, Jeff Zhu, Oncel Tuzel, and Anurag Ranjan. Mobileone: An improved one millisecond mobile backbone. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7907–7917, 2023. 6

[42] C. S. Wallace. A suggestion for a fast multiplier. *IEEE Transactions on Electronic Computers*, EC-13(1):14–17, 1964. 4

[43] Haoran You, Xiaohan Chen, Yongan Zhang, Chaojian Li, Sicheng Li, Zihao Liu, Zhangyang Wang, and Yingyan Lin. Shiftaddnet: A hardware-inspired deep network. *Advances in Neural Information Processing Systems*, 33:2771–2783, 2020. 4

[44] Haoran You, Huihong Shi, Yipin Guo, et al. Shiftaddvit: Mixture of multiplication primitives towards efficient vision transformer. *arXiv preprint arXiv:2306.06446*, 2023. 4

[45] 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. 2

[46] Yichi Zhang, Zhiru Zhang, and Lukasz Lew. Pokebnn: A binary pursuit of lightweight accuracy. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12475–12485, 2022. 1, 2, 3, 4, 5, 7, 8

[47] 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. 2, 3