PokeBNN: A Binary Pursuit of Lightweight Accuracy

Yichi Zhang*
Cornell University
yz2499@cornell.edu

Zhiru Zhang
Cornell University
zhiruz@cornell.edu

Łukasz Lew*
Google Research
lew@google.com

Abstract

Optimization of Top-1 ImageNet promotes enormous networks that may be impractical in inference settings. Binary neural networks (BNNs) have the potential to significantly lower the compute intensity but existing models suffer from low quality. To overcome this deficiency, we propose PokeConv, a binary convolution block which improves quality of BNNs by techniques such as adding multiple residual paths, and tuning the activation function. We apply it to ResNet-50 and optimize ResNet's initial convolutional layer which is hard to binarize. We name the resulting network family PokeBNN[1]. These techniques are chosen to yield favorable improvements in both top-1 accuracy and the network's cost. In order to enable joint optimization of the cost together with accuracy, we define arithmetic computation effort (ACE), a hardware- and energy-inspired cost metric for quantized and binarized networks. We also identify a need to optimize an under-explored hyper-parameter controlling the binarization gradient approximation.

We establish a new, strong state-of-the-art (SOTA) on top-1 accuracy together with commonly-used CPU64 cost, ACE cost and network size metrics. ReActNet-Adam [33], the previous SOTA in BNNs, achieved a 70.5% top-1 accuracy with 7.9 ACE. A small variant of PokeBNN achieves 70.5% top-1 with 2.6 ACE, more than 3x reduction in cost; a larger PokeBNN achieves 75.6% top-1 with 7.8 ACE, more than 5% improvement in accuracy without increasing the cost. PokeBNN implementation in JAX/Flax [6, 18] and reproduction instructions are open sourced.[2]

1. Introduction

A need for Pareto optimization. Deep learning research is largely driven by benchmarks and metrics. In the past a single metric per benchmark, *e.g.*, top-1 accuracy on ImageNet, was sufficient. Today one needs to account for various model architectures, sizes, and computational costs. This promotes optimizing the Pareto frontier of a quality metric such as top-1 and another cost metric such as FLOPS, latency, or energy consumption.

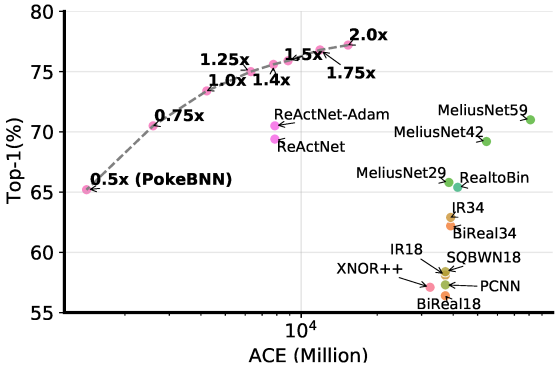


Figure 1. **Comparison of BNNs using top-1 and ACE (Sec. 3).**

The choice of the optimization metric. Since the industry is currently the main user of large-scale inference, the cost metric should be correlated to dollar cost per inference. As the ML hardware gets more mature, what becomes evident is that the energy use is the key metric proportional to the inference cost, especially in the data centers. In Sec. 3 we define a new proxy metric called *arithmetic computation effort* (ACE), which aims to estimate inference cost abstracting of concrete ML hardware.

The impact of quantization and binarization. In numerical formats such as int8 or float16, the less significant bits do not affect a network's output as much as the more significant bits. Yet, processing them consumes the same amount of energy. This might not be optimal. The possible inference cost reduction (or equivalently performance improvement) with each halving of the quantization bits (*e.g.*, 16b to 8b to 4b to 2b to 1b) is at least 2x (*e.g.*, NVIDIA Ampere is 8x faster in int1 than in int8 [38]) and up to 4x as estimated by the ACE metric. For comparison, this is significantly larger than the improvements yielded by upgrading the GPU or TPU by one or two generations.

*Work performed while at Google, equal contribution.

[1] Poke/ˈpɒkɪ/ is pronounced similarly to pocket. PokeConv, PokeBNN, and Pokemon are abbreviations of Pocket Convolution, Pocket Binary Neural Network, and Pocket Monster, respectively.

[2] Source code and reproduction instructions are available in AQT repository: github.com/google/aqt.

Binarization pushes this benefit to the extreme by replacing floating-point dot products with logical XNOR and bit counting operations. If binary neural networks (BNNs) can reach high quality, they are likely to gain a large footprint for inference both in the data center and at the edge.

BNN optimization is hard. Pioneering modern BNNs used to suffer from a more than 20% top-1 accuracy gap compared to their floating-point counterparts [23]. Only recently BNNs have become comparable in quality to the popular ResNet-18 model [33, 34]. One reason is that BNNs tend to have a chaotic, discontinuous loss landscape that renders their optimization challenging [31, 33]. In fact, for the binarization to work one has to change many things compared to standard DNN practices. BNNs require multi-phase training, approximation of gradients, and various architectural adjustments that avoid binarization information bottlenecks.

Our main contributions are as follows:

- We propose *PokeConv*, a binary convolutional block that can substantially improve BNN accuracy. We replace most of the convolutions in ResNet [17] with PokeConv.
- We propose *PokeInit* block to replace ResNet's initial convolutional layer that is hard to binarize. PokeInit significantly reduces the network's cost. PokeInit and PokeConv form the foundation of the PokeBNN family.
- We optimize an under-explored clipping bound hyper-parameter in BNNs that controls the binarization gradient approximation. Ablation in Sec. 6 shows we gain more than 3% in top-1 accuracy through this parameter.
- We motivate and define a novel hardware and energy inspired cost metric called ACE, which is informed by inference costs on hardware yet at the same time it is agnostic to the existing hardware platforms. ACE improves alignment of the research on energy-efficient neural networks and research on ML hardware. We use ACE to quantify the inference cost of PokeBNN.
- We empirically show that on ImageNet [43] PokeBNN establishes the Pareto-SOTA of top-1 together with cost metrics: CPU64, ACE, and network size. We improve over the SOTA ReActNet-Adam by 5.1% top-1 at the same ACE cost (Fig. 1).

2. Related Work

There is a large and active body of research investigating the training and acceleration of BNNs. We only review a subset of the past efforts that have a high influence on the network design presented in this paper. A comprehensive survey can be found in [49].

BNN feasibility. The pioneering works [9, 23, 27] demonstrated the feasibility of BNNs. They established the training framework for neural networks with binarized weights and activations and demonstrated promising results on small datasets such as MINIST and CIFAR-10. However, their preliminary ImageNet results show a large top-1 accuracy drop from 62.5% to 36.1% on AlexNet [29] and from 68.9% to 47.1% on GoogleNet [45].

Multi-phase training. A key effective technique is the multi-phase training [8, 33, 34, 37], where one starts with training an unquantized model and only later enables binarization. Some approaches employ a three-phase training — from the unquantized version, to binarized activations only, to binarized weights and activations [37]. Knowledge distillation is another technique that has commonly been used to improve the accuracy of BNNs [8, 33, 34, 37].

BNN architecture. Another comprehensive line of work explores architectural changes to strive for better model quality. Many of them aim to incur negligible compute and parameter overhead. For example, a channelwise real-valued rescaling of the binarized tensors can effectively mitigate the quantization loss [2,7,42]. Connecting the unquantized input activations of a binarized convolutional layer to its output with a shortcut enhances the gradient flow and the model representation capacity [35]. Squeeze-and-excitation (SE) [22] is another computationally cheap technique that promises quality improvement on small convolutional models including BNNs [37]. FracBNN [50] includes additional BatchNorm Layers [24] in a BNN to speed up convergence. Authors in [8] first show that using a PReLU function [16] after each convolutional layer improves binary model quality. Along this line, it is recently reported that introducing learnable biases into the PReLU function leads to extra improvements in model accuracy [33, 34]. With the evolution, current BNNs have finally exceeded 70% top-1.

3. Arithmetic Computation Effort

In this section we motivate and define ACE, which is designed to reflect neural network inference cost on idealized ML hardware implemented with CMOS methodology.

ACE metric definition. ACE is defined as follows:

$$\text{ACE} = \sum_{i \in I, j \in J} n_{i,j} \cdot i \cdot j \tag{1}$$

where $n_{i,j}$ is the number of multiply-accumulate operations (MACs) between a i-bit number and j-bit number and can be automatically derived from model structure. I and J are sets of all bitwidths used in the inference of a given neural network, typically $I = J = \{1, 2, 4, 8, 16\}$.

The energy use is highly correlated with the total cost of the computation. The inference could be happening in a data center or on edge devices and it can be served from CPUs, GPUs or TPUs. For edge devices, the battery usage is the main concern, which makes the energy use a key bottleneck in many ML applications. In the case of data centers, surprisingly, energy is also the main cost driver. In order to run inferences in a data center, one needs to pay for:

hardware, electricity and power provisioning, and other infrastructure costs. A GPU card may cost 1000 USD and be used for 3-5 years consuming 400W. Electricity bill at 65% utilization and 15 cents per kWh for three years would amount to 0.4kW * 24h * 365 * 3 * 0.65 * 0.15 USD/kWh = ∼1000 USD as well. Interestingly, the cost of the power provisioning in data centers (cooling, transformers, batteries, backup generators) is reported to be more than twice that of the electricity bill (at least in case of Google data centers) [25]. Also, a correlation of ML chip cost is reported to be over 90% with its TDP. Overall, the cost of running inferences is indeed mainly driven by the energy consumption.

The bulk of the computation energy usage is in arithmetic operations energy. Contrary to classic CPUs, ML hardware running inference spends a high fraction of its energy on the actual arithmetic (e.g., multiplications, additions, other functions). For instance, in the case of TPUs, the cost of computation control is amortized over enormous SIMD sizes of 16K to 64K [25,26]. This is usually achieved using systolic arrays [30]. In stark contrast, CPUs have a typical SIMD size of 4 to 32 (e.g., SSE, AVX). We discuss other non-arithmetic energy sinks in the appendix in a full version of paper.

Arithmetic operation energy is proportional to the number of active bit-adders. To multiply two unsigned integers $a < 2^I$, $b < 2^J$, one first computes a value of $I \cdot J$ bits using logical AND operations and sum them in groups:

$$\sum_{0 \le i < I} a_i 2^i \sum_{0 \le j < J} b_j 2^j = \sum_{\substack{0 \le i < I \\ 0 \le j < J}} (a_i \wedge b_j)\, 2^{i+j} \qquad (2)$$

In order to evaluate the sum, one uses bit-adders, carefully taking into account to add bit triplets within one significance group. Bit-adder sums three bits and outputs a two bit result: $p_1 + p_2 + p_3 = 2q_1 + q_2$ where $p_i, q_i \in \{0, 1\}$. Bit-adders are the main building block of all multipliers and adders. Each adder removes one bit from the pool, so taking into account addition into the accumulator (AC in MAC), a multiplication will activate $I \cdot J$ bit-adders.[3] Notably, circuits that are not switching leak negligible amounts of energy, so one only pays for what they use. One may verify that the number of active bit-adders is measured by ACE.

CPU64 metric. Previous BNN research typically use FLOPs + $\frac{1}{64}$BOPs as a cost metric [33–35, 42]. It was motivated by the fact that one 64-bit CPU register can do 64 BOPs in one cycle, compared to one float64 (double precision) operation per cycle. We extend CPU64 to int4 and int8 formats using coefficients 1/16 and 1/8, respectively.

Independent verification of energy use. Remarkably, the actual energy measurements on Google TPUs hardware are reasonably correlated with the ACE metric, grounding it in reality. Tab. 1 reproduces energy measurement reported by Google and Horowitz [19,25] on 45nm and 7nm process node and attaches both ACE and CPU64 metrics. Interestingly, bfloat16 and to a large extent float16 and float32 are also well correlated with ACE both in 45nm and 7nm process nodes. We therefore choose to not special-case the ACE formula for MAC cost on floating-point formats.

[3]While there are many orders in which one can construct adder trees (*e.g.*, Wallace tree [46], Dadda tree [11]), affecting latency and clock speed, the particular order has a limited effect on the energy use.

Table 1. **ADD/MUL energy use in femto-Joules (fJ) [19, 25], and the corresponding CPU64 and ACE metrics. The correlation coefficient between ACE and the sum of ADD and MUL energy is 0.992 for 7nm and 0.946 for 45nm, whereas the CPU64-energy correlation is much smaller: 0.703 for 7nm and 0.724 for 45nm.**

	ADD Energy (fJ)		MUL Energy (fJ)		MAC	
	45nm	7nm	45nm	7nm	CPU64	ACE
float32	900	380	3700	1310	1	1024
float16	400	160	1100	340	1	256
bfloat16	-	110	-	210	-	256
int32	100	30	3100	1480	-	1024
int8	30	7	200	70	1/8	64
int4	-	-	-	-	1/16	16
int2	-	-	-	-	1/32	4
binary	-	-	-	-	1/64	1

Implementation of high precision with binary arithmetic. If we interpret a_i, b_i as binary matrices and $a_i \wedge b_i$ as binary matrix multiplication, then Eq. (2) can be used to implement higher precision matrix multiplication on binary hardware. The cost of that emulation is $I \cdot J$, which is consistent with ACE metric. The result holds for all linear operations including convolution.

Comparison to other metrics. Informed by the arithmetic energy use, ACE for MACs of N-bit and N-bit is quadratic in N as opposed to our CPU64 extension which is linear in N. ACE generalizes FLOPS and CPU64 allowing for evaluation of mixed quantization models. ACE allows for evaluation of MACs with different bitwidths for weights and activations. This is useful as one of them is often much easier to quantize or binarize. ACE is informed by CMOS hardware design and manufacture constraints yet at the same time is hardware target agnostic. With that we aim to better predict the performance of energy-efficient neural networks on the future ML hardware. This is an advantage over popular methods of tuning the model for latency on GPUs or mobile hardware such as smartphones [20, 44].

4. PokeBNN

In this section, we introduce the design methodology of PokeBNN family. As a preliminary, we first define the quantization and binarization math used throughout the design. We then introduce PokeConv — a binarization friendly convolution replacement, and PokeInit — a quantized and cost-optimized initial layer replacement. Finally, we combine the proposed techniques and use ResNet as a

template to present the entire PokeBNN architecture. We quantize the final layer to 8 bits, in effect, all the linear and convolutional layers are quantized to 8, 4 bits, or binarized.

4.1. Quantization and Binarization Equations

While both quantization and binarization methods are well studied [1, 23], we summarize them for completeness. In case of binarization, the clipping bound B is usually hardcoded to 1 [23] or 1.3 [4]. In Sec. 6 we show the importance of optimizing B.

Quantization. In order to use energy-efficient integer and binary convolutions and matrix multiplications, one needs to convert floating-point numbers into integers. We define the casting operation as follows:

$$\begin{aligned} \text{clip}(x, x_{min}, x_{max}) &= \min(x_{max}, \max(x_{min}, x)) \\ \text{int}_b(x) &= \text{round}(\text{clip}(x, -C_b + \epsilon, C_b - \epsilon)) \end{aligned}$$

where b denotes the bitwidth, $C_b = 2^{b-1} - 0.5$ is the end point of the quantization grid, and ϵ is a small floating-point number making sure that the rounding avoids overflow. For unsigned values, one uses $\text{uint}_b(x) = \text{floor}(\text{clip}(x, 0, 2^b - \epsilon))$. For simplicity, we will focus on the signed case in the subsequent discussions.

During the backpropagation, the round function is ignored, *i.e.*, $\frac{d\,\text{round}}{dx}(x) = 1$. This is known as the straight-through estimator (STE) [23]. The derivative of clip operation is the usual 1 inside of the clip interval and 0 outside.

Applying casting directly to the arguments of convolution is inappropriate as their dynamic range can be different than the clipping bounds. Instead, one assumes (or estimates) bound B based on the distribution of argument and then appropriately rescale it:

$$Q_b(x) = \text{int}_b(x \cdot \frac{C_b}{B}) \cdot \frac{B}{C_b} \tag{3}$$

One may note that the gradient is: $\frac{dQ_b}{dx}(x) = \mathbf{1}_{x \in (-B, B)}$.

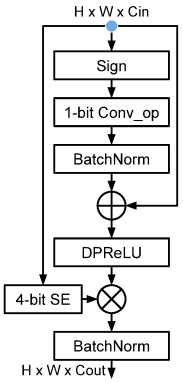
For non-binary activations, we obtain B by calculating the maximum absolute value in a batch, and using exponentially moving average ($\alpha = 0.9$). Importantly, we freeze B when we enable activation quantization. Without freezing, we observe a feedback loop leading to inferior results or divergence. For binary activations, we show in Sec. 6 that the value of B makes a remarkable impact on the model quality. We use a fixed $B = 3$ in the experiments.

For all weights we use output-channel-wise bounds: $B_o = max_i|w_{i,o}|$ (where i, o are indexing input and output channels). They are never frozen. As equations indicate, we do not center the distribution. It is sufficient to just scale it.

Binarization. When $b = 2$, Eq. (3) yields ternarization, but for binarization ($b = 1$) it would round every number to 0. Instead we use $Q_1(x) = \text{sign}(x)$. Its gradient is the same as that of Eq. (3). Importantly, while the forward pass does not depend on bound B, the gradient does. There is a line of work that studies a continous gradient estrimators to the discrete functions above [13, 35, 41, 51].

4.2. PokeConv

We now propose the core convolutional (Conv) building block in our BNN, *PokeConv*. Fig. 2 shows its diagram and the corresponding pseudocode. The design of PokeConv strictly follows the goal of optimizing the accuracy and ACE trade-off. The additional operations around the binarized Conv layer are designed to be computationally lightweight and to help BNN training converge faster.




```
def PokeConv(
    x, r1, conv_op, ch, st=1):
  r = x
  x = Conv_op(x, ch, st)
  x = BatchNorm(x)
  x = ReshapeAdd(x, r, pad_ch)
  # The next line has an effect
  # only in the 3rd PokeConv call
  x = ReshapeAdd(x, r1, tile_ch)
  x = DPReLU(x)
  x = x * SE_4b(r, ch)
  x = BatchNorm(x)
  return x
```

Figure 2. **PokeConv building block** — "r1" is the original ResNet shortcut; "ch" is the number of channels or features; and "st" is stride.

Adding residuals around each binary convolution. We connect the unquantized input activations to the output of the Conv and BN combination with a shortcut. Since the Conv layer is binarized, this shortcut is important as it removes the information bottleneck to the succeeding layer [35].

A practical question emerges naturally — how do we design the shortcut if there is a mismatch between the input and output channel or spatial dimensions? Unfortunately, there is a lack of study on the general solution to it. A commonly-used method is adding a 1×1 Conv layer with proper strides [17]. However, this should be avoided since these layers are needed around most of the PokeConv blocks and they will increase ACE tremendously. ReActNet [34] proposes to duplicate activations when the number of channels doubles, although this is only feasible when the channel number is exactly doubling.

We propose to use a simple zero padding when the number of channels expands. Namely, given a channel expansion factor of K, an input tensor x with n channels is padded as follows: $\text{pad}(x)_i = x_i \cdot \mathbf{1}_{i<n}$. We find that zero padding works the best for local shortcuts, and tiling proposed by ReActNet [34] works the best for the original ResNet shortcuts. Using a single method in both cases

yields inferior results.

When the channel number decreases by a factor of K, we use an average pooling of the neighboring K channels: $\text{avg\_ch}(x)_i = \frac{1}{K}\sum_{0\le k<K} x_{i\cdot K+k}$.

On the spatial dimension, we use an average pooling on the shortcut for downsampling. The pseudocode of aggregating the spatial and channel reshaping on residuals (the argument "r") is shown below:

```
def ReshapeAdd(x, r, expand_ch_op):
  if r is None: return x
  if r.ch < x.ch: r = expand_ch_op(r, x.ch)
  if r.ch > x.ch: r = avg_ch(r, x.ch)
  if r.shape != x.shape:
    r = avg_pool_3x3(r, x.ch, st=2)
  return x + r
```

Using binarization-friendly nonlinearity. While the binarization function is nonlinear itself, inserting additional nonlinearity can further improve the BNN model quality as reported by prior studies [8, 34], especially if the function learns to shift and reshape the activation distribution [34].

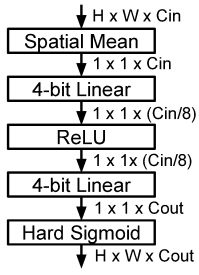
We propose to add a nolinear function — Dynamic PReLU (DPReLU) [36] after the residual addition, defined as follows:

$$\text{DPReLU}(x) := \begin{cases} \eta(x-\alpha)-\beta & x-\alpha>0 \\ \gamma(x-\alpha)-\beta & \text{otherwise} \end{cases} \quad (4)$$

Here α, β, γ, and η are all channelwise learnable parameters. They are initialized to 0, 0, 0.25, 1.0, respectively. Aside from the activation shifting, DPReLU has learnable slopes on both linear pieces. It introduces an additional reshaping flexibility compared to RPReLU proposed by ReActNet [34].

Squeeze-and-excitation (SE) [22] helps scaling. SE is a computationally cheap technique that improves model quality. It allows the network to incorporate global knowledge on given inputs. BNN such as real-to-binary [37] uses a different variant of SE as well.

In our design we apply an SE block as used in MobileNetV3 [20]. The diagram and pseudocode are shown in Fig. 3.




```
def SE_4b(x, ch)
  s = SpatialMean(x)
  s = Linear_4b(s, x.ch/8)
  s = ReLU(s)
  s = Linear_4b(s, ch)
  s = ReLU6(s+3)/6
  return s
```

Figure 3. **Squeeze-and-Excitation: quantized to 4-bit.**

Importantly, the ACE metric reports that SE dense layers incur a non-negligible cost. We therefore propose to quantize the SE blocks to 4 bits, and set the hidden projection length to 1/8 of the input. Experiments indicate that this modification incurs no accuracy loss.

Additional normalization layers. We place an additional BatchNorm layer [24] on the output (*i.e.*, before the residual split). FracBNN [50] suggests this modification in order to speed up the convergence. This extra layer is even more important for PokeBNN. Firstly, its bias term learns to shift the distribution properly so that it balances the sign activations to the next PokeConv layer. Secondly, its adjustment on the distribution allows the aggressive 4-bit quantization of the first SE dense layer. Moreover, it normalizes the shortcut around binary convolutions and facilitates the gradient flow.

Limitations. We use the default zero padding for Conv layers, which introduces a third value for a small fraction of pixels. We believe that most of the prior works suffer from the same limitation. Padding with alternating 1 and -1 proposed and evaluated in FBNA [15] would resolve this limitation.

4.3. PokeInit and Projection Layer Optimization

After replacing regular Convs with PokeConvs, we find two other components in ResNet-50 that are ACE-costly: (1) the input 7×7 Conv layer; (2) the 1×1 projection Conv layers for shortcuts in downsampling blocks. These layers are conventionally excluded from binarization [32].

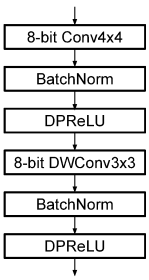
Replacing 1×1 projection Conv layers with ReshapeAdd. The 1x1 projection Conv layers would incur 360 million MACs.

We propose to completely remove these downsampling projection layers and replace them with the ReshapeAdd function defined for shortcuts. We use tiling instead of zero-padding for channel expansion, *i.e.*, for an input tensor x that has n channels, $\text{tile}(x)_i = x_{(i \bmod n)}$.

PokeInit. The unquantized input layer alone requires 118 million MACs. The main sources of the large number of MACs are (1) the 7×7 kernel size and (2) the large output spatial resolution 112×112.

To optimize the ACE cost, we fuse the stride-2 max-pooling with the first stride-2 convolution, yielding stride=4. This reduces the output spatial resolution by 4×. We then reduce the kernel size from 7×7 to 4×4. Further downsizing the kernel will lead to an information loss as there will be pixels not convolving with the kernels. To increase the receptive field of the input block, we follow it with a 3×3 depthwise Conv layer. We denote such an input layer combination as *PokeInit*. Its pseudocode is shown in Fig. 4.

To further reduce the cost, we quantize PokeInit to 8 bits. This optimization reduces the cost of the input layer from 118 million float MACs to 6.6 million int8 MACs.




```
def PokeInit(x):
  x = Conv4x4_8b(x, ch=32, st=4)
  x = BatchNorm(x)
  x = DPReLU(x)
  x = Conv3x3_depthwise_8b(x, ch=64)
  x = BatchNorm(x)
  x = DPReLU(x)
  return x
```

Figure 4. **PokeInit: quantized to 8-bit.**

4.4. Model Assembly

We assemble PokeConv and PokeInit with ResNet-50 template with an 8-bit linear classifier head as shown in the pseudocode.

As discussed in the previous sections, we now apply PokeConv and PokeInit to the base ResNet-50, and remove the 1×1 projection Conv layers therein. Apart from those we also quantize the classifier to 8 bits. A pseudocode of the final network is as follows:

```
CH = 64 * M

def PokeBNN50(x):
  x = PokeInit(x)
  for i in range(16):
    st = 2 if i in (3, 7, 13) else 1
    if i < 3: ch = CH
    elif: i < 7: ch = CH * 2
    elif: i < 13: ch = CH * 4
    r = x
    x = PokeConv(x, None, Conv1x1_1b, ch, st=1)
    x = PokeConv(x, None, Conv3x3_1b, ch, st=st)
    x = PokeConv(x, r, Conv1x1_1b, 4*ch, st=1)
  x = SpatialMean(x)
  x = Linear(x)
  return x
```

Note that all Conv and linear layers do not use biases except for those that are not followed by BatchNorms. We follow ResNet [17] to configure BatchNorm initializations.

5. Experiments

In this section, we empirically evaluate PokeBNN on the ILSVRC12 ImageNet [43] classification dataset with a resolution of 224×224. We only apply random crop and flip as data augmentation.

5.1. Training Setup

We conduct experiments on 64 TPU-v3 chips with a batch size of 8192. We use Adam optimizer [28] ($\beta_1 = 0.9, \beta_2 = 0.99$) with a linear learning rate decay. The initial learning rate is 6.4e-4. The weight decay is set to 5e-5 throughout the training. BatchNorm momentum is set to 0.9. To estimate the clipping bound B for activation of non-binary quantized layers, we follow the method in [1] and use exponentially moving average ($\alpha = 0.9$) of maximum value in a batch.

We train PokeBNN for a total of 750 epochs and employ the two-phase training. We find that the first semi-unquantized phase is needed for only as little as 50 epochs. 4-bit and 8-bit activations are quantized at epoch 50. All weights (8-bit, 4-bit, and binary) are quantized at epoch 50. 1-bit activations are always 1-bit during the training.

We use the knowledge distillation setting to train PokeBNN, which requires computing KL-divergence loss. The modification is as simple as replacing the one-hot ground truth label with the teacher prediction. We use an 8-bit ResNet-50 as the teacher model. We also tried distilling from a vision transformer [12], but surprisingly the result is similar.

In order to measure the accuracy, after decaying the learning rate to zero, we continue training for a few epochs. We observe both top-1 oscillating due to the batch normalization statistics being updated further. We find that training and evaluation top-1 are completely uncorrelated. Hence it would be unfair to follow a practice of taking the maximum top-1. All top-1 numbers in this paper are averaged top-1 over several epochs where learning rate is already zero. The difference between the mean and maximum accuracy is about 0.5% to 1%.

5.2. Evaluation Results

To have a fair comparison, we scale the number of output channels in a PokeConv block to change the model size (*e.g.*, PokeBNN-2x means doubling the number of output channels). All results are collected in Tab. 2. The standard deviation of the top-1 across 5 runs of PokeBNN-1x with different random weights is 0.034%.

Importantly, for the prior work, we assume all FP32 operations could be replaced by BF16 without accuracy loss. PokeBNN does not use FP32. Based on the data we analyze Pareto curve of accuracy vs. cost metrics, and compare the trade-off of PokeBNN to the baselines in the literature. We have several key observations thereof:

PokeBNN establishes the SOTA Pareto frontier for BNNs under the ACE metric as visualized in Fig. 1. The accuracy of PokeBNN scales notably well with the model size. Though binarized from ResNet-50, scaling the number of channels by 2× in PokeBNN leads to a 77.2% top-1 accuracy, slightly higher as the 4-bit ResNet-50 (Tab. 2), yet with a more than 4.6× higher efficiency.

Most BNN models in the literature produce below 65% top-1 accuracy on ImageNet. ReActNet [34] and ReActNet-Adam [33] for the first time reach ResNet-18 level accuracy near 70% by leveraging the MobileNet architecture [21]. With the same ACE budget as the current SOTA ReActNet-Adam, our PokeBNN-1.4x achieves 75.6% top-1 accuracy, more than 5% higher. A small variant PokeBNN-0.75x has

the same top-1 as ReActNet-Adam but reduces the ACE by more than 3×.

Compared to the MeliusNet-59 [4] variant that has the highest accuracy in the BNN literature (Tab. 2), a large variant PokeBNN-2x is 6% more accurate and meanwhile still 5.3× more efficient in ACE.

In addition, we test PokeConv on ResNet-18 architecture and observe that PokeBNN-0.5x Pareto-dominates it.

PokeBNN also establishes the SOTA Pareto frontier for BNNs under the commonly-used CPU64 metric. We plot the Pareto curve using the widely adopted CPU64 metric in the literature. As shown in Fig. 5, the trade-off trend is roughly the same as compared to the proposed ACE metric. Notably, some BNNs (*e.g.*, MeliusNet-29) show a less favourable trade-off under the CPU64 metric than the one in Fig. 1. This is because ACE captures the fact that a binary operation is more than 64× cheaper than a floating-point operation in terms of energy use.

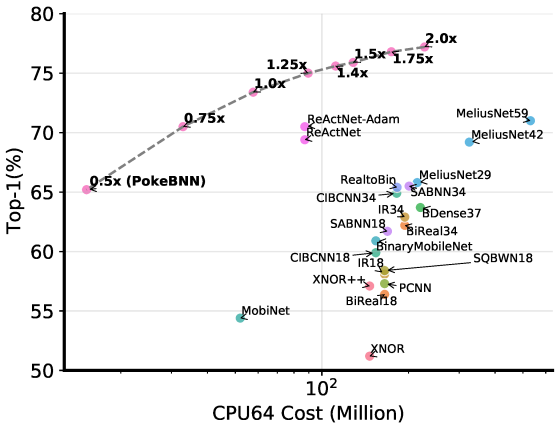


Figure 5. **Comparison of BNNs using top-1 and CPU64 cost.**

PokeBNN outperforms prior BNNs on the size-accuracy trade-off. Model size is another important dimension that indicates the memory requirement. We therefore plot the model size vs. top-1 accuracy in Fig. 6, which shows that PokeBNN is also on the SOTA Pareto frontier when compared to the prior arts.

6. Ablation Study

In this section we provide a detailed ablation on our proposed techniques. We measure the impact of each individual technique on PokeBNN-1.0x.

Clipping bound ablation. The clipping bound B, as a hyperparameter, plays a major role in low-bitwidth quantization [1, 10], but has rarely been explored in the past BNN research. In BNNs, although few works manually set the bound for binary activations $B = 1.3$ [4, 5], there is a lack of study on it and under most circumstances $B = 1$ by default [23, 34, 35].

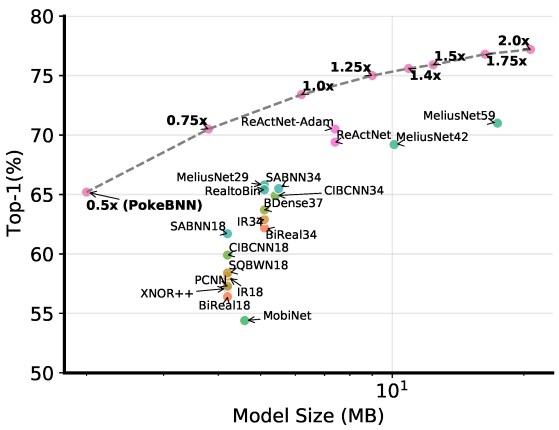


Figure 6. **Comparison of BNNs using top-1 and model size.**

In our experiments, we find that B makes a remarkable impact on the BNN accuracy. We sweep B for the binarized activations over a set of values ranging from 1.0 to 6.0. Each PokeConv has the same bound. Results are in Tab. 3.

There is a 3.3% accuracy gap between between $B = 3$ and most commonly used value $B = 1$. We hypothesize that a larger clipping threshold improves Lipschitz constant of the loss surface and also reduces number of dead neurons (i.e., neurons with gradient zero). This is consistent with the prior observation [1, 10] that the clipping bound is important for ultra low-bitwidth quantization.

PokeConv ablation. We remove each component in PokeConv one at a time and study the impact. The results are in Tab. 4. Removing DPReLU from PokeConv causes the largest accuracy drop, even larger than replacing PokeConv with the original binarized ResNet block. We hypothesize that this is because the change eliminates nonlinearity on both shortcuts, which impedes the model learning. We also replace DPReLU with RPReLU [34] and ReLU. This leads to a 0.2% and 0.6% top-1 degradation, respectively. Since 0.2% is above 3 standard deviations (3 × 0.034%), DPReLU indeed improves the model quality over the other two candidates.

The other components (*i.e.*, 4-bit SE, additional shortcuts and BatchNorms) all have at least 3% top-1 impact on the model. Given they incur negligible overhead in ACE, they are favourable design choices.

We also experiment with adding back the 1×1 projection Conv layers. The top-1 result is 73.5%, only 0.1% higher. Completely removing these layers is therefore sensible.

PokeInit ablation. In PokeBNN-1.0x, we replace PokeInit with ResNet's original 7×7 input Conv layer followed by a maxpooling. The top-1 is 73.5%, only 0.1% higher. Given that PokeInit reduces the number of operations in the input layer by 18×, it is a favourable trade-off.

We also experiment with removing the 3×3 depthwise Conv layer in PokeInit. This results in a 73.1% top-1. The

Table 2. **Final results and comparison to prior arts** — When calculating ACE for FP32 operations, we assume they can be cast to BF16 without accuracy loss. "–" indicates unavailable data. The standard deviation of top-1 across 5 different seeds for PokeBNN-1.0x is 0.034%. BF16 PokeBNN is a variant where all convolutions and dense layers are in BF16. The bottom four rows show the base models for context, all other models are binary.

Model	MAC Operations (10^6) FP32	BF16	INT8	INT4	Binary	ACE (10^9)	CPU64 (10^6)	Size (MB)	Top-1 (%)
AlexNet-BNN [23]	-	-	-	-	-	-	-	-	36.1
GoogleNet-BNN [23]	-	-	-	-	-	-	-	-	47.1
XNOR-Net [42]	120	-	-	-	1700	32.4	146.6	4.2	51.2
XNOR-Net++ [7]	120	-	-	-	1700	32.4	146.6	4.2	57.1
Bi-RealNet-18 [35]	139	-	-	-	1680	37.3	165.2	4.2	56.4
Bi-RealNet-34 [35]	139	-	-	-	3530	39.1	194.2	5.1	62.2
IR-Net-18 [41]	-	-	-	-	-	37.3	165.2	4.2	58.1
IR-Net-34 [41]	-	-	-	-	-	39.1	194.2	5.1	62.9
SQ-BWN-18 [48]	-	-	-	-	-	37.3	165.2	4.2	58.4
PCNN [14]	-	-	-	-	-	37.3	165.2	4.2	57.3
BDenseNet37-Dilated [5]	-	-	-	-	-	-	220.0	5.1	63.7
CI-BCNN-18 [47]	-	-	-	-	-	-	154.0	4.2	59.9
CI-BCNN-34 [47]	-	-	-	-	-	-	182.0	5.4	64.9
MobiNet [39]	-	-	-	-	-	-	**52.0**	4.6	54.4
BinaryMobileNet [40]	-	-	-	-	-	-	154.0	-	60.9
MeliusNet-29 [4]	129	-	-	-	5470	38.5	214.5	5.1	65.8
MeliusNet-42 [4]	174	-	-	-	9690	54.2	325.4	10.1	69.2
MeliusNet-59 [4]	245	-	-	-	18300	81.0	530.9	17.4	**71.0**
Real-to-Binary Net [37]	156.4	-	-	-	1676	41.7	182.6	5.1	65.4
SA-BNN-18 [32]	-	-	-	-	-	-	169.0	4.2	61.7
SA-BNN-34 [32]	-	-	-	-	-	-	201.0	5.5	65.5
SA-BNN-50 [32]	-	-	-	-	-	-	-	-	68.7
QuickNetSmall [3]	-	-	-	-	-	-	-	**4.0**	59.4
QuickNet [3]	-	-	-	-	-	-	-	4.2	63.3
QuickNetLarge [3]	-	-	-	-	-	-	-	5.4	66.9
ReActNet-A [34]	11.9	0	0	0	4816.9	**7.9**	87.2	7.4	69.4
ReActNet-Adam [33]	11.9	0	0	0	4816.9	**7.9**	87.2	7.4	70.5
PokeBNN-2.0x	0	0	10.7	14.5	14412.2	**15.3**	227.4	20.7	**77.2**
PokeBNN-1.75x	0	0	10.2	11.1	11037.1	**11.9**	174.4	16.3	**76.8**
PokeBNN-1.5x	0	0	9.7	8.2	8111.7	**8.9**	128.5	12.4	**75.9**
PokeBNN-1.4x	0	0	9.5	7.1	7037.2	**7.8**	111.6	10.9	**75.6**
PokeBNN-1.25x	0	0	9.2	5.7	5635.8	**6.3**	89.6	9.0	**75.0**
PokeBNN-1.0x	0	0	8.7	3.6	3609.5	**4.2**	57.7	6.2	**73.4**
PokeBNN-0.75x	0	0	8.2	2.0	2032.7	**2.6**	32.9	3.8	**70.5**
PokeBNN-0.5x	0	0	7.6	0.9	905.6	**1.4**	15.2	2.0	**65.2**
FP32 ResNet-50 [17]	4089.2	0	0	0	0	1046.8	4089.2	97.3	76.7
BF16 ResNet-50 [1]	0	4089.2	0	0	0	1046.8	4089.2	48.6	76.7
INT4 ResNet-50 [1]	0	0	120.1	3969.1	0	71.2	263.1	13.1	77.1
BF16 PokeBNN	0	3621.8	0	0	0	927.2	3621.8	50.3	79.2

Table 3. **Impact of the activation clipping bound B in the binarization function.**

Clipping Bound B	1.0	1.3	2.0	3.0	4.0	5.0	6.0
Top-1 (%)	70.1	71.4	72.9	**73.4**	73.3	72.8	72.4

Table 4. **Ablate each component in PokeConv.** "All" indicates replacing PokeConv with the original 1-bit ResNet Conv block.

Remove Module	SE	DPReLU	Shortcuts	BN	All
Top-1 (%)	70.6	60.4	68.1	70.2	61.9

depthwise layer trades 2.7% of the total ACE cost for 0.3% accuracy, which is also a fair trade-off.

Precision ablation. Increasing the weight or activation precision in PokeConv from 1-bit to 4-bit results in a 75.2% and 76.8% top-1, respectively. Both of these variants have an ACE cost of 15, and both are significantly better than INT4 ResNet [1] but worse than PokeBNN-1.75x. This result indicates that binarization indeed allocates energy better than int4 formats.

7. Conclusion

The main ingredients of PokeBNN: PokeConv, PokeInit, and the clipping bound ($B = 3$), together establish a strong SOTA in the domain of cost-efficient networks. ACE metric improves alignment of research on cost-efficient neural networks with future ML hardware. Our results indicate that binarization may indeed be a good choice in cost-accuracy trade-off. The main price of these benefits is a 750-epoch long training.

There are several unanswered questions. How to take energy of memory access into account in a synthetic metric? How could the Poke architecture be further simplified or improved? Could architecture templates different than ResNet-50 or perhaps neural architecture search yield significantly better networks?

Acknowledgements. The authors would like to thank Catalyst, JAX, and Flax teams for valuable implementation, discussions, and suggestions on AQT library. This work is supported in part by NSF Award #2007832.

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. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, 2021. 4, 6, 7, 8

[2] Yash Akhauri. HadaNets: Flexible quantization strategies for neural networks. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, 2019. 2

[3] Tom Bannink, Adam Hillier, Lukas Geiger, Tim de Bruin, Leon Overweel, Jelmer Neeven, and Koen Helwegen. Larq compute engine: Design, benchmark and deploy state-of-the-art binarized neural networks. *Machine Learning and Systems*, 2021. 8

[4] Joseph Bethge, Christian Bartz, Haojin Yang, Ying Chen, and Christoph Meinel. MeliusNet: An improved network architecture for binary neural networks. *IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2021. 4, 7, 8

[5] Joseph Bethge, Haojin Yang, Marvin Bornstein, and Christoph Meinel. BinaryDenseNet: Developing an architecture for binary neural networks. *International Conference on Computer Vision (ICCV) Workshops*, 2019. 7, 8

[6] 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. *Version 0.1*, 2018. 1

[7] Adrian Bulat and Georgios Tzimiropoulos. Xnor-net++: Improved binary neural networks. *arXiv preprint arXiv:1909.13863*, 2019. 2, 8

[8] Adrian Bulat, Georgios Tzimiropoulos, Jean Kossaifi, and Maja Pantic. Improved training of binary networks for human pose estimation and image recognition. *arXiv preprint arXiv:1904.05868*, 2019. 2, 5

[9] Zhiyong Cheng, Daniel Soudry, Zexi Mao, and Zhenzhong Lan. Training binary multilayer neural networks for image classification using expectation backpropagation. *arXiv preprint arXiv:1503.03562*, 2015. 2

[10] 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. 7

[11] L. DADDA. Some schemes for parallel multipliers. *Alta Frequenza*, 1965. 3

[12] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. *International Conference on Learning Representations (ICLR)*, 2021. 6

[13] Ruihao Gong, Xianglong Liu, Shenghu Jiang, Tianxiang Li, Peng Hu, Jiazhen Lin, Fengwei Yu, and Junjie Yan. Differentiable soft quantization: Bridging full-precision and low-bit neural networks. *International Conference on Computer Vision (ICCV)*, 2019. 4

[14] Jiaxin Gu, Ce Li, Baochang Zhang, Jungong Han, Xianbin Cao, Jianzhuang Liu, and David Doermann. Projection convolutional neural networks for 1-bit cnns via discrete back propagation. *AAAI Conference on Artificial Intelligence*, 2019. 8

[15] Peng Guo, Hong Ma, Ruizhi Chen, Pin Li, Shaolin Xie, and Donglin Wang. FBNA: A fully binarized neural network accelerator. *International Conference on Field Programmable Logic and Applications (FPL)*, 2018. 5

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *International Conference on Computer Vision (ICCV)*, 2015. 2

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 2, 4, 6, 8

[18] 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. *Version 0.3*, 2020. 1

[19] Mark Horowitz. 1.1 computing's energy problem (and what we can do about it). *International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, 2014. 3

[20] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, and Hartwig Adam. Searching for mobilenetv3. *International Conference on Computer Vision (ICCV)*, 2019. 3, 5

[21] 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

[22] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. 2, 5

[23] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. *Advances in Neural Information Processing Systems (NeurIPS)*, 2016. 2, 4, 7, 8

[24] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *International Conference on Machine Learning (ICML)*, 2015. 2, 5

[25] Norman P. Jouppi, Doe Hyun Yoon, Matthew Ashcraft, Mark Gottscho, Thomas B. Jablin, George Kurian, James Laudon, Sheng Li, Peter Ma, Xiaoyu Ma, Thomas Norrie, Nishant Patil, Sushma Prasad, Cliff Young, Zongwei Zhou, and David Patterson. Ten lessons from three generations shaped google's tpuv4i : Industrial product. *International Symposium on Computer Architecture (ISCA)*, 2021. 3

[26] Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre-luc

Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C. Richard Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Daniel Killebrew, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Matt Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon. In-datacenter performance analysis of a tensor processing unit. *SIGARCH Comput. Archit. News*, 2017. 3

[27] Minje Kim and Paris Smaragdis. Bitwise neural networks. *arXiv preprint arXiv:1601.06071*, 2016. 2

[28] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 6

[29] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems (NeurIPS)*, 2012. 2

[30] HT Kung and Charles E Leiserson. Systolic arrays (for vlsi). *Sparse Matrix Proceedings 1978*, 1979. 3

[31] Hao Li, Soham De, Zheng Xu, Christoph Studer, Hanan Samet, and Tom Goldstein. Training quantized nets: A deeper understanding. *Advances in Neural Information Processing Systems (NeurIPS)*, 2017. 2

[32] Chunlei Liu, Peng Chen, Bohan Zhuang, Chunhua Shen, Baochang Zhang, and Wenrui Ding. SA-BNN: State-aware binary neural network. *Proceedings of the AAAI Conference on Artificial Intelligence*, 2021. 5, 8

[33] Zechun Liu, Zhiqiang Shen, Shichao Li, Koen Helwegen, Dong Huang, and Kwang-Ting Cheng. How do adam and training strategies help bnns optimization? *International Conference on Machine Learning (ICML)*, 2021. 1, 2, 3, 6, 8

[34] Zechun Liu, Zhiqiang Shen, Marios Savvides, and Kwang-Ting Cheng. ReActNet: Towards precise binary neural network with generalized activation functions. *European Conference on Computer Vision (ECCV)*, 2020. 2, 3, 4, 5, 6, 7, 8

[35] Zechun Liu, Baoyuan Wu, Wenhan Luo, Xin Yang, Wei Liu, and Kwang-Ting Cheng. Bi-real net: Enhancing the performance of 1-bit cnns with improved representational capability and advanced training algorithm. *European Conference on Computer Vision (ECCV)*, 2018. 2, 3, 4, 7, 8

[36] Kien Mai Ngoc, Donghun Yang, Iksoo Shin, Hoyong Kim, and Myunggwon Hwang. Dprelu: Dynamic parametric rectified linear unit. *The 9th International Conference on Smart Media and Applications*, 2020. 5

[37] Brais Martinez, Jing Yang, Adrian Bulat, and Georgios Tzimiropoulos. Training binary neural networks with real-to-binary convolutions. *International Conference on Learning Representations*, 2020. 2, 5, 8

[38] NVIDIA. NVIDIA A100 Tensor Core GPU Architecture, 2020. 1

[39] Hai Phan, Dang The Huynh, Yihui He, Marios Savvides, and Zhiqiang Shen. MoBiNet: A mobile binary network for image classification. *IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2020. 8

[40] Hai Phan, Zechun Liu, Dang Huynh, Marios Savvides, Kwang-Ting Cheng, and Zhiqiang Shen. Binarizing mobilenet via evolution-based searching. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020. 8

[41] Haotong Qin, Ruihao Gong, Xianglong Liu, Mingzhu Shen, Ziran Wei, Fengwei Yu, and Jingkuan Song. Forward and backward information retention for accurate binary neural networks. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020. 4, 8

[42] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. XNOR-Net: ImageNet classification using binary convolutional neural networks. *European Conference on Computer Vision (ECCV)*, 2016. 2, 3, 8

[43] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision (IJCV)*, 2015. 2, 6

[44] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. 3

[45] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015. 2

[46] C. S. Wallace. A suggestion for a fast multiplier. *IEEE Transactions on Electronic Computers*, 1964. 3

[47] Ziwei Wang, Jiwen Lu, Chenxin Tao, Jie Zhou, and Qi Tian. Learning channel-wise interactions for binary convolutional neural networks. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019. 8

[48] Jianguo Li Yinpeng Dong and Renkun Ni. Learning accurate low-bit deep neural networks with stochastic quantization. *British Machine Vision Conference (BMVC)*, 2017. 8

[49] Chunyu Yuan and Sos S Agaian. A comprehensive review of binary neural network. *arXiv preprint arXiv:2110.06804*, 2021. 2

[50] Yichi Zhang, Junhao Pan, Xinheng Liu, Hongzheng Chen, Deming Chen, and Zhiru Zhang. FracBNN: Accurate and FPGA-efficient binary neural networks with fractional activations. *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2021. 2, 5

[51] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. DoReFa-Net: Training low bitwidth

convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160*, 2016. 4