Towards a Better 16-Bit Number Representation for Training Neural Networks

Himeshi De Silva[1], Hongshi Tan[2], Nhut-Minh Ho[2], John L. Gustafson[3], and Weng-Fai Wong[2]

[1] Institute for Infocomm Research, A*STAR, Singapore
himeshi_de_silva@i2r.a-star.edu.sg
[2] National University of Singapore
{tanhs, minhhn, wongwf}@comp.nus.edu.sg
[3] Arizona State University
jlgusta6@asu.edu

Abstract. Error resilience in neural networks has allowed for the adoption of low-precision floating-point representations for mixed-precision training to improve efficiency. Although the IEEE 754 standard had long defined a 16-bit float representation, several other alternatives targeting mixed-precision training have also emerged. However, their varying numerical properties and differing hardware characteristics, among other things, make them more or less suitable for the task. Therefore, there is no clear choice of a 16-bit floating-point representation for neural network training that is commonly accepted. In this work, we evaluate all 16-bit float variants and upcoming *posit*™ number representations proposed for neural network training on a set of Convolutional Neural Networks (CNNs) and other benchmarks to compare their suitability. Posits generally achieve better results, indicating that their non-uniform accuracy distribution is more conducive for the training task. Our analysis suggests that instead of having the same accuracy for all weight values, as is the case with floats, having greater accuracy for the more commonly occurring weights with larger magnitude improves the training results, thereby challenging previously held assumptions while bringing new insight into the dynamic range and precision requirements. We also evaluate the efficiency on hardware for mixed-precision training based on FPGA implementations. Finally, we propose the use of statistics based on the distribution of network weight values as a heuristic for selecting the number representation to be used.

Keywords: Neural Networks · 16-bit Floating-point · Half-precision, Posit

1 Introduction

Owing to the size of datasets and the complexity of models used for learning at present, training a large neural network can easily require days or weeks even on modern GPUs. Because 16-bit data types halve the required memory

and bandwidth demands of 32-bit floating-point and therefore can significantly improve performance and energy efficiency, IEEE 754 standard "half-precision" floats (IEEE16) and several other 16-bit floating-point variations have been proposed for deep learning. However, each of these formats have significantly different properties giving them varying degrees of suitability for training neural networks. Figure 1 shows the *relative decimal accuracy* (RDA) (defined in Section 7) between consecutive number pairs for 16-bit float formats (all published variations) and posit formats studied in this work, which we collectively refer to as FP16. The width of a graph shows the dynamic range while the height shows the RDA distribution in that range, for each format. The different accuracy distributions of the number representations lead to varying success in neural network training, as demonstrated by the test loss for a representative benchmark in Figure 2. To overcome accuracy degradation, researchers have had to rely on custom accuracy saving techniques. However, the exact accuracy degradation behavior of each FP16 format and the generality of each technique is not known. While mixed-precision training with IEEE16 has reported 2–4.5× speedup, mechanisms used to improve training accuracy also increases training overhead [19]. Therefore, there is no clear consensus as to what the optimal 16-bit float representation is for training or what characteristics of a number format lead to better training results.

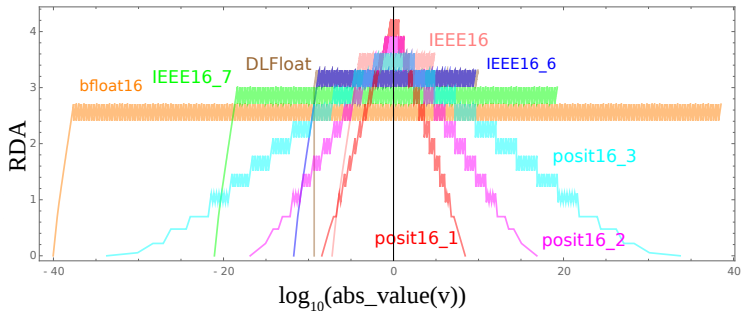


Fig. 1: RDA - FP16

CNNs have been studied and adopted widely for tasks such as image classification. Here, we test all FP16 number formats previously proposed for CNNs (as well as new ones), under *identical conditions*, to assess their relative capabilities for CNN training benchmarks of varying sizes. We also do the same for newer language translation models to test the versatility of the formats. In this paper we will:

- Demonstrate empirically that for a 16-bit IEEE-type float format, the optimal number of exponent bits producing the best results is 6.
- Show that posits produce better accuracy results, which suggests that their non-uniform accuracy distribution is more suitable for training.

– Analyze the accuracy for a selected benchmark to show that having more accuracy for commonly occurring weights with larger magnitude is key. Because all previously studied 16-bit float formats have uniform accuracy distributions (Figure 1), this idea has not been presented previously.
– Propose a new accuracy saving technique for neural network training that adds no overhead, which is that of shifting the peak RDA of posits based on the distribution of weight values.
– Evaluate the FP16 formats in terms of hardware costs for conversion and accuracy under different rounding modes to gain a more holistic and practical understanding of their capabilities in mixed-precision training.

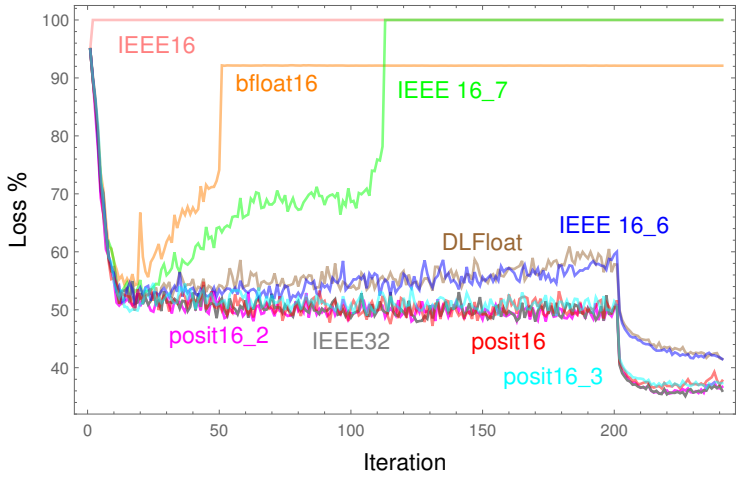


Fig. 2: Test loss - FP16

2 Related Work

The IEEE 754 standard [3] provides all details related to floating-point data types that are implemented on most systems. Its "half-precision" format, IEEE16, specifies 5 exponent bits. To achieve training results comparable to IEEE32 using IEEE16 representations for weights, activations and gradients, several accuracy saving mechanisms need to be employed. They include maintaining copies of master weights in IEEE32 for weight update, loss scaling (not required for CNNs) and accumulating IEEE16 vector dot products into IEEE32 values [17]. Mixed-precision training with bfloat16 requires the same techniques except for loss scaling [2]. Another FP16 format, DLFloat, has a 6-bit exponent and aims to simplify the hardware for floating-point computations [1]. When it is used in training, weights, biases, activations, errors and weight updates are represented

in DLfloat, with fused-multiply-add instructions used to perform the matrix multiplications and convolution operations at 16-bit precision.

A reduced-precision dot product algorithm which initially accumulates smaller chunks of a long dot product and then accumulates their partial sums hierarchically coupled with stochastic floating-point rounding has been used to reduce the representation of weights, activations and gradients to an 8-bit float format. The accumulations for forward and backward passes occur in a 16-bit float with a 6-bit exponent and so does the regularization, momentum computation and weight update — which uses a master copy of the weights at 16-bit precision [24]. Another attempt at reducing the representation to 8-bit float format for weights, activations, errors and gradients uses techniques such as quantization, stochastic rounding, loss scaling, IEEE32 computations and accumulations and maintaining master weight copies in IEEE16 [16].

A hybrid exponent size scheme has been proposed to enable 8-bit floats to work across a broader range of networks and models. This scheme stores weights and activations with a 4-bit exponent while tensors in the backward pass are stored with a 5-bit exponent. The very low number of bits of significance requires loss scaling with multiple scaling factors, and conversion across different formats along with other optimizations [22]. Recently, ultra-low (4-bit) precision combined with other techniques have been proposed for training [23]. These training schemes require either enhanced versions of 16-bit accuracy-saving techniques or even more complex mechanisms that add overhead. Because all of them still make use of 16-bit or 32-bit floats, assessing the various 16-bit formats and their capabilities is still useful even when using such 8-bit representations.

Taking a cue from neural network inference, floats with precision as low as a single bit have been tested for training [25]. Similarly, 16-bit (and smaller) fixed-point representations have also been studied for training DNNs [4,6,12]. However, these either work only on small training examples or require mechanisms such as dynamic scaling, shared exponents, specialized instructions for high precision steps, or other complex accuracy management schemes and improvements to match IEEE32 accuracy. For example, the Flexpoint format actually has a 16-bit significand with separate exponent bits. Our own experiments with fixed-point did not produce useful results even for the smallest benchmarks, thus we are excluding it from discussion.

Although posits have been explored briefly for training and inference [7, 8, 15], it has not been evaluated or compared against other formats extensively. Moreover, as we will demonstrate, layer-wise scaling is not necessary for training with 16-bit posits.

3 16-Bit Float and Posit Formats

3.1 IEEE 754 Standard type float formats

An IEEE float contains three fields: a 1-bit sign S, a w-bit biased exponent E, and a $(t = p - 1)$-bit trailing significand field digit string $T = d_1 d_2 ... d_{p-1}$; the

leading significand bit, d_0, is implicitly encoded in E. The value v inferred from this representation is given by Eq. 1. The default rounding mode as defined by the Standard is round-to-nearest-even (RNE).

$$v = \begin{cases} \text{NaN (Not a Number)} & \text{if } E = 2^w - 1, T \neq 0 \\ (-1)^S \times (+\infty) & \text{if } E = 2^w - 1, T = 0 \\ (-1)^S \times 2^{1-emax} \times & \\ \quad (0 + 2^{1-p} \times T) & \text{if } E = 0, T \neq 0 \\ 0 & \text{if } E = 0, T = 0 \\ (-1)^S \times 2^{E-bias} \times & \\ \quad (1 + 2^{1-p} \times T) & \text{all other cases} \end{cases} \tag{1}$$

Since 2008, the Standard defines a 16-bit *binary16* format, commonly referred to as "half-precision" and here referred to as IEEE16, for which $w = 5, t = 10$. The value v of an IEEE16 float can be inferred by substituting these values along with $p = 11$, $emax = 15$, $bias = 15$, in Eq. 1.

The *bfloat16* format is supported in the deep learning framework Tensorflow. According to information available, the bfloat16 format is used for storing activations and gradients in memory for mixed-precision training. While there is no official standard for this format, a bfloat16 number is similar to an IEEE 754 floating-point data with a 1-bit sign, $w = 8$ and $t = 7$ [11]. Therefore the value v of a bfloat16 representation can be derived by substituting these values along with $p = 8$, $emax = 127$, $bias = 127$, in Eq. 1.

We also designed an IEEE-style 16-bit format that has a 6-bit exponent and 9-bit fraction and a format with a 7-bit exponent and 8-bit fraction. In our work, we will refer to these formats as IEEE16_6 and IEEE16_7 respectively. The IEEE16_6 format is similar to DLFloat in Section 3.2 except that IEEE16_6 supports the default rounding and all exceptions of the IEEE 754 Standard, including subnormals. The value, v of a IEEE16_6 representation can be derived by substituting values $w = 6$, $p = 10$, $emax = 31$, $bias = 31$, in Eq. 1. The value v of a IEEE16_7 representation can be derived by substituting values $w = 7$, $p = 9$, $emax = 63$, $bias = 63$, in Eq. 1.

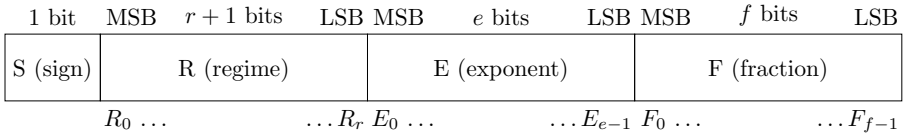
3.2 DLFloat

DLFloat is a representation similar to IEEE16_6 [1]. However, to simplify the hardware design for DLFloat, it does not support subnormals (i.e. when $E = 0$). Instead, when $E = 0$, v is treated like a normal float with a bias of 31. Other differences are that there is a single representation for infinities and NaN, the sign of zero is ignored, and the only rounding mode for DLFloat is round to nearest, up (RNU).

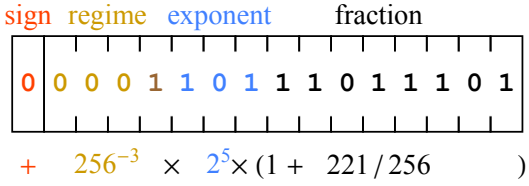
3.3 Posit

A *posit* number, as described in its Standard [20], is defined by four fields as shown in Figure 3(a) and has recently shown promise in CNN training [15, 18].

To decode a posit value, one must know the the total width of the posit, *nbits* (16, in this case), and the width *es* of the exponent field E. We test *es* values of 1, 2 and 3 referred to as posit16_1, posit16_2 and posit16_3. Other posit environment variables of interest are *maxpos*, the largest real value expressible as a posit and its reciprocal *minpos*, the smallest nonzero value expressible as a posit.



(a) posit Binary Format



(b) Posit Example

Fig. 3: General Posit Format and 16-bit Posit Example

$$v = \begin{cases} 0 & \text{if } p = 00\cdots0 \\ \text{Not-a-Real (NaR)} & \text{if } p = 10\cdots0 \\ (1 - 3S + F)\times & \\ \quad 2^{(-1)^S(R\times 2^{es}+E+S)} & \text{otherwise} \end{cases} \tag{2}$$

The four fields of the posit are a 1-bit sign S, regime R consisting of r bits identical to R_0 terminated by $1 - R_0$ ($r + 1$ bits total length) or by reaching the end of the posit (r bits total length), exponent E represented by e exponent bits, terminated by a maximum of *es* or the end of the posit, and fraction F represented by f fraction bits terminated by the end of the posit. The value of R is $-r$ if R_0 is 0, and $r-1$ if R_0 is 1. E is *es* bits wide, with `0` bit padding in the least significant bits if the exponent field has fewer than *es* bits because of the regime length. F represents an unsigned integer divided by 2^f. The representation (S, R, E, F) of the posit p represents the value v which is inferred from the fields by Eq. 2 (The equation gives the binary representation of p in the first two cases). The equation for v looks quite different from that for standard floats because posits are based on 2's complement representation whereas floats are based on sign-magnitude representation. Not-a-Real (NaR) condenses all exception values into a single case.

Rounding for a real value x to a posit is given by the following rules; If x is exactly expressible as a posit, it is unchanged. If $|x| > maxpos$, x is rounded to $sign(x) \times maxpos$. If $0 < |x| < minpos$, x is rounded to $sign(x) \times minpos$. For all other values RNE is used. To further understand the decoding of a posit number consider the example in Figure 3(b). Here $nbits = 16$, $es = 3$. The value of the posit can be derived from Eq. 2. A more intuitive decoding that works for positive values is to raise $2^{2^{es}}$ to the power of R and multiply it by 2^E and $1 + F$. Posits representing negative values can be decoded by negating them (2's complement) and applying the same technique.

4 Evaluation

Given that bandwidth improvements have a significant impact on training performance, we evaluated the FP16 formats for storing parameters during learning. We modified the training framework, such that it can take any new number format (of any bitwidth) and train, provided the conversion functions to and from IEEE32 [10]. Specifically, the data type of the structure that stores and communicates data was changed to FP16 and so that it applied to all weights, biases, activations and gradients. A conversion to IEEE32 from FP16 occurs just before the computation and a conversion to FP16 from IEEE32 happens soon after. This configuration was used for forward/backward passes and weight updates and allows for maximum bandwidth savings for any 16-bit format. The same setup was used both for training and validation.

Our goal with these experiments was to observe how each format will perform in training on its own to identify format characteristics that are most suitable for training. Therefore, we did not use precision enhancement techniques such as maintaining a master copy of weights. *The original IEEE32 hyperparameters (i.e. batch size, training epochs, learning rates, momentum, weight decay) were unchanged for the FP16 experiments.* This approach is consistent with current literature. For reproducibility, we fixed the random seed and did not use cuDNN. Tests were conducted using publicly available models. To minimize conversion overhead between the formats and FP32, we also implemented some BLAS routines. CNN datasets used are MNIST, FASHION-MNIST, CIFAR10, CIFAR100 [13], and ILSVRC 2012 [21]; the networks used are LeNet, cuda-convnet, NIN [14], SqueezeNet [9], and AlexNet. All are publicly available, or provided with the deep learning framework.

5 Results

Table 1 shows a summary of the CNN benchmark configurations that were used for our experiments and accuracy results in the fourth block of rows for each FP16 format. The results indicate that although all of the formats perform well for the smallest models and datasets, some formats begin to struggle as the benchmark complexity increases. The IEEE-type format that delivers the best

Model	LeNet	LeNet	convnet	NIN	Squeeze -Net	Alex -Net	Res -Net18	Trans. Base	Trans. Base
Dataset	MNIST	FMNIST	CIFAR 10	CIFAR 100	Image -Net	Image -Net	Image -Net	30K	IWSLT 14
Batch Size	64	64	100	128	32	256	32	128	128
Iterations	10K	10K	60K	120	170	450	800	2270	30780
IEEE32	98.70%	89.10%	78.70%	56.06%	56.40%	57.04%	67.88%	35.42	23.54
bfloat16	98.24%	89.08%	76.02%	0.96%	0.32%	52.40%	61.88%	35.18	21.68
DLFloat	98.66%	89.38%	77.96%	45.48%	54.24%	46.56%	69.12%	35.49	9.43
IEEE16	98.70%	89.22%	73.02%	NaN	0.00%	53.08%	NaN	0	Error
IEEE16_6	98.72%	89.60%	78.56%	46.28%	54.72%	46.84%	68.00%	35.59	12.6
IEEE16_7	98.46%	89.54%	78.74%	NaN	0.24%	9.96%	67.52%	35.16	9.46
posit16_1	98.72%	89.38%	9.76%	54.92%	50.80%	50.68%	0.00%	34.31	9.45
posit16_2	98.78%	89.36%	77.74%	53.92%	56.80%	53.60%	67.64%	35.18	24.97
posit16_3	98.66%	89.30%	79.72%	53.74%	56.48%	53.16%	67.60%	35.06	24.32

Table 1: Benchmark configuration, accuracy, BLEU score

accuracy across networks is IEEE16_6. DLFloat shows similar performance although slightly inferior to IEEE16_6 possibly due to the absence of subnormals. This clearly shows that the best configuration of bits for an IEEE 754 type format for CNN training is possibly IEEE16_6 with 6 bits of exponent and 9 bits of fraction. The posit representations, especially posit16_2, perform considerably better than any other format across all networks. It is closely followed by posit16_3 in many cases and posit16_1 in some cases (for example NIN/CIFAR100), despite posit16_1's limited dynamic range.

Table 2 gives the dynamic ranges of all formats. Note that posit16_1's dynamic range is smaller than all of the other formats except for IEEE16. Even in that case, it is only marginally greater for values of smaller magnitude. However, formats with greater dynamic range such as bfloat16 do not perform as well. This suggests that limited dynamic range is not the sole reason for the failure of a format as it is often attributed to in the case of IEEE16. However, IEEE16 shows better performance than other float types in the case of AlexNet/Imagenet, which highlights the challenges of developing a format that can perform well across a wide range of networks and datasets.

Because neural networks are increasingly being used in application areas such as language translation, we also tested these formats on the Transformer Base Model for language translation from German to English. The last two columns of Table 1 shows the BLEU score results from these experiments. All formats perform well for the 30K dataset. However, for the IWSLT-14 dataset posit16_2 manages to perform as well as IEEE32, demonstrating again the versatility of the posit formats.

Aside from accuracy, the FP16 formats also differ in their rounding modes. Posits use saturation rounding (see Section 3.3) for values beyond their dynamic range while floats underflow to 0 or overflow to infinity. We experimented with changing the rounding mode to saturation rounding for float formats, as well as

to underflow to 0 and overflow to infinity for posits, and performed the training. Saturation rounding severely affected the accuracy of all the float formats. Due to its larger dynamic range, underflows in posit16_3 are rare and therefore its results remained unchanged. posit16_2's results remained the same or improved except in the case of Squeezenet. posit16_1 showed the most dramatic changes in some cases due to its smaller dynamic range. This suggests that having the posit distribution's accuracy in the appropriate range of the learnt parameters is more important than the choice of whether they saturate or underflow.

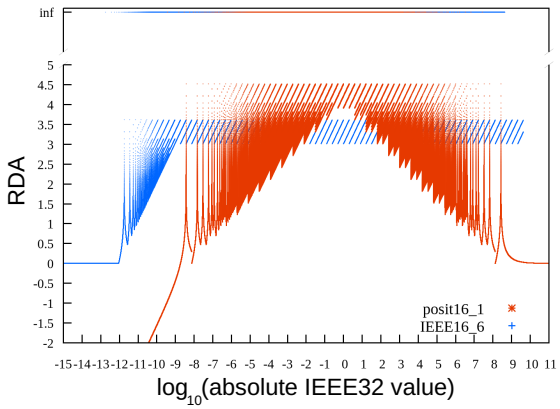


Fig. 4: RDA w.r.t IEEE32 of IEEE16_6 vs posit16_1

Maintaining a copy the weights at a higher precision (referred to as "master" weights) for weight updates is a technique used to improve the accuracy for training with float formats [2, 17]. It works by updating 32-bit weights with 16-bit gradients and then converting them to 16-bit weights for use in the forward and backward phases of the training. We implemented this technique for the NIN/CIFAR100 benchmark and IEEE16 and bfloat16 formats. The NaN result for IEEE16 did not change as the cause for it was activations overflowing to infinity that were then being multiplied by zero. For bfloat16 the accuracy improved to 47.86%. Given that this technique requires 3× more memory for weights and that there are other formats that perform better without it, the technique only adds additional work and is redundant.

6 Hardware Implications

All 16-bit floating point formats provide the same savings in memory consumption. However, any performance gain they bring about is dampened by costs associated with converting between the format and IEEE32 as well as their implementation on hardware when employed in mixed-precision training. To understand these hardware associated costs we implemented the conversion routines

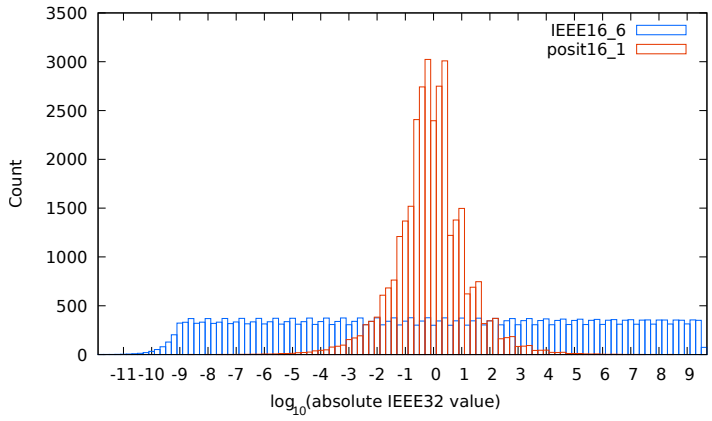


Fig. 5: Distribution of Infinitely Accurate Numbers of IEEE16_6 vs posit16_1

on an FPGA platform. Table 3 presents these results on a Xilinx U250 Alveo Data Center Accelerator Card with synthesis done in Vivado 2018.2. Looking at the depth, i.e., the number of cycles required for one conversion, bfloat16 is the most efficient because it only involves rounding and right shifting to convert an IEEE32 value to a bfloat16 value, and a left shift of a bfloat16 by 16 bits to get to the IEEE32 value. DLFloats, which eliminate subnormals as well as simplify rounding, follows bfloat16. Owing to the variable length regime R, the posit formats have a slightly higher depth than these two formats. Among the posit formats there is only a slight difference in depths. However, the IEEE 754 style formats are the most expensive for conversion owing to the more complicated conversion logic (subnormals, rounding modes, etc.).

7 Accuracy Analysis

Given the numerous differences between the floating-point formats, we set out to identify which of their characteristics are responsible for better accuracy. Due to the many complexities of the training process, we selected the NIN/CIFAR100 benchmark for the analysis as it is the simplest with discerning accuracy results. To further make our analysis concise, we picked the best performing posit and float formats, which are posit16_1 and IEEE16_6 respectively. For both formats, we sampled activations, weights, biases and gradients during training and analyzed each value's RDA as compared to the RDA of its corresponding value when training with IEEE32. In effect, we treat the IEEE32 value as the precise value. The RDA between an exact value x and its approximated value $\hat{x}$ is obtained from the following equation [5];

$$RDA(x, \hat{x}) = \log_{10}(|x|/|x - \hat{x}|)$$

We measure the RDA when $x \neq 0$. When $x = \hat{x}$, the RDA is ∞. Because we treat the IEEE32 values as the reference, we measure how accurate each of these formats are when representing IEEE32 values.

7.1 Accuracy differences between posit16_1 and IEEE16_6

Figure 4 shows the RDA of both formats w.r.t. IEEE32. Note that although posit16_1 has a smaller dynamic range than IEEE16_6, in the range of approximately $[-1.2, 1.2]$ (i.e. $1/16 \leq |x| \leq 16$) posits have superior accuracy. Figure 5 shows the distribution of values that exactly represent the IEEE32 value, i.e. are infinitely accurate; posit16_1 shows a normal distribution around 0 while IEEE16_6 has a mostly uniform distribution. These figures show that posit16_1 has accuracy superior to IEEE16_6 near zero, but the probability of superior accuracy decreases as the magnitude of the value moves away from 0.

7.2 Loss behavior

The NIN training network contains layers of type convolution, pooling, relu, dropout and softmax (with loss) at one end. It trains for a total of 120K iterations. Figure 6 gives the training loss from the softmax with loss function E for the two formats (and IEEE32). Posit16_1's loss closely follows that of IEEE32 while IEEE16_6's loss actually increases despite still converging. Therefore, we start the analysis by looking at the end of the network which is the loss activation and gradient values at the last layer. Figures 7 and 8 show these activations and gradients respectively near the end of training. The median loss activation accuracy in the case of posit16_1 is greater than that of IEEE16_6 despite most of the values falling outside of posit16_1's optimal accuracy range. Even in the case of gradients, the median value of posit16_1 is still slightly better even though the gradients have more values of even smaller magnitude. Note that in both these graphs, posit16_1's accuracy for smaller values is substantially lower as compared to that of IEEE16_6, due to its limited dynamic range, suggesting that the accuracy of the values with smaller magnitude has a lesser impact on the training accuracy.

7.3 Effect on Weights

The accuracy of loss gradients is affected by the weights and vice-versa. Therefore, we shift our attention to the weight layer before the loss (convolution layer *cccp6*) in the network. Figure 9 plots the accuracy of weight gradients in this layer at the end of the training. These weight gradient values are computed from the loss gradients and similarly although most values lie outside the optimal posit accuracy range, posits still retain accuracy similar to IEEE16_6 for weight gradients too. However, looking at the accuracy of weights at the end of the training in Figure 10, posits clearly have better median accuracy. Figure 11 shows the distribution of the weight values in this layer at the end of training.

In combination with Figures 10 and 4 it is clear that a greater proportion of weight values which are also larger in magnitude fall inside posit16_1's optimal accuracy range. The accuracy of weights in all other layers in the network also showed similar behavior resulting in all of them having higher median accuracy for posits when compared to IEEE16_6.

7.4 Posit Accuracy in Training

The accuracy of learned weights has a significant impact on the training process as all other values (i.e. activations, gradients, loss) are computed from them. Therefore, the higher posit accuracy for weights most likely transcends to other values such as gradients which often may not fall into the posit optimal accuracy range, as seen from Figures 7, 8 and 9. This in turn results in improved overall training accuracy for posits. It is also worthwhile to note that the range of the weight values stabilizes early on in the training, in this case within the first 500 iterations. This also helps to lessen the potential impact of deficiencies in accuracy at the beginning of the training.

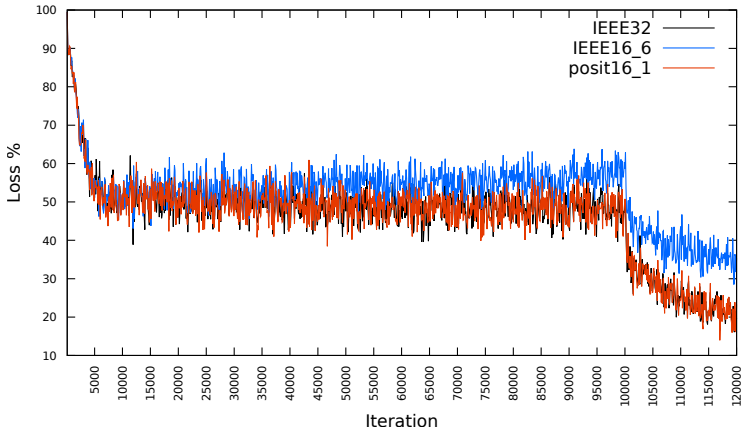


Fig. 6: NIN/CIFAR100 Training Loss

In this benchmark, the weight values are less than 1 in magnitude. Therefore, the posit accuracy distribution gives more accuracy to the weight values with larger magnitude and less accuracy to the smaller ones. This in turn gives larger activations more accuracy, which has a cascading effect through the network. The larger weights are also less effected by the errors in the small gradient values, as is evident in the plots. Moreover, the larger the gradient value, the more accuracy it will have in the case of posits. We believe that the same principle, which is to improve the accuracy of weight values by giving them more accuracy in the representation, is behind the idea of maintaining a master copy of weights at higher precision in CNN training. However, our analysis suggests

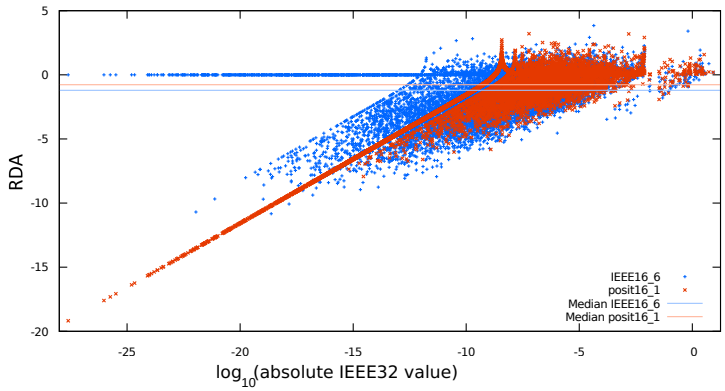


Fig. 7: Loss Layer Activations Accuracy

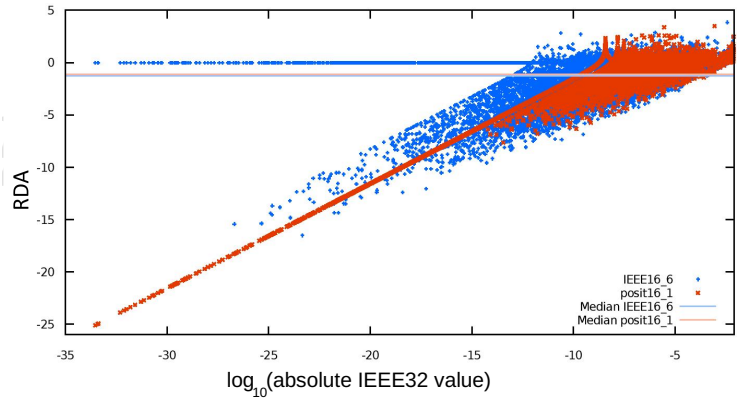


Fig. 8: Loss Layer Activation Gradients Accuracy

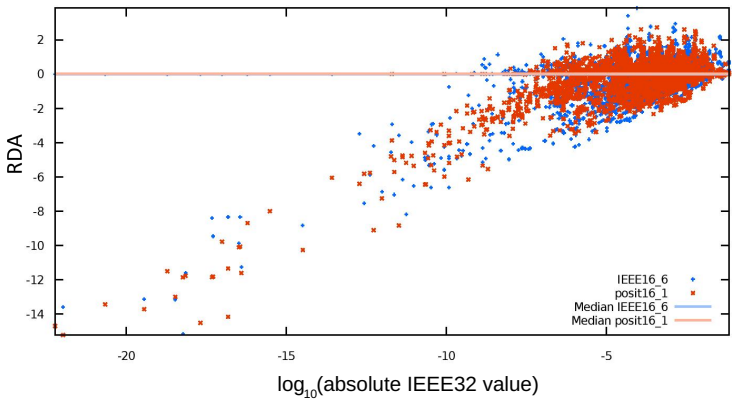


Fig. 9: Accuracy - Weight Gradients

Format	Min. Exp. (Normal)	Max. Exp.	Min. Value (Normal)	Max. Val
IEEE32	-149 (-126)	127	1.4×10^{-45} (1.8×10^{-38})	3.4×10^{38}
bfloat16	-133 (-126)	127	9.2×10^{-41} (1.8×10^{-38})	3.4×10^{38}
DLFloat	-31	32	2.3×10^{-10}	8.6×10^{9}
IEEE16	-24 (-14)	15	6.0×10^{-8} (6.1×10^{-5})	6.6×10^{4}
IEEE16_6	-39 (-30)	31	1.8×10^{-12} (9.3×10^{-10})	4.3×10^{9}
IEEE16_7	-70 (-62)	63	8.5×10^{-22} (2.2×10^{-19})	1.8×10^{19}
posit16_1	-28	28	3.7×10^{-9}	2.7×10^{8}
posit16_2	-56	56	1.4×10^{-17}	7.2×10^{16}
posit16_3	-112	112	1.9×10^{-34}	5.2×10^{33}

Table 2: Exponent and dynamic ranges of the formats

that greater accuracy is needed mostly for the larger magnitude weights that are more prevalent during training.

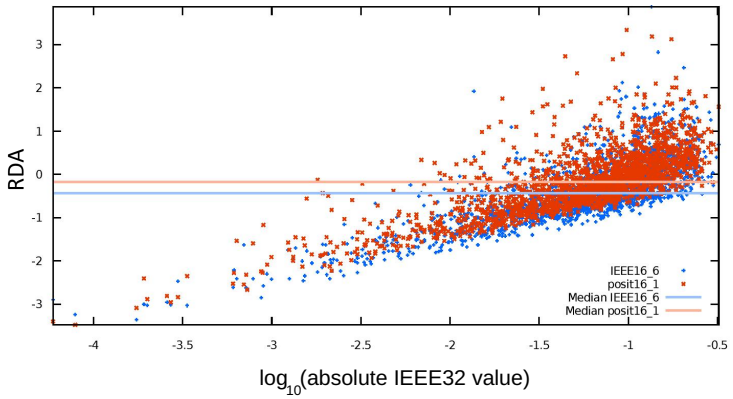


Fig. 10: Accuracy - Weights

This hypothesis also rings true for other benchmarks that were tested. Figures 13 and 12 show the corresponding weight distributions and accuracy at the third quartile for the best performing float and posit formats at the end of training for the SqueezeNet/ImageNet and AlexNet/ImageNet benchmarks. We selected the third quartile in this case to show the accuracy close to the peak of the weight distribution. Note that in these cases too, posits retain superior accuracy

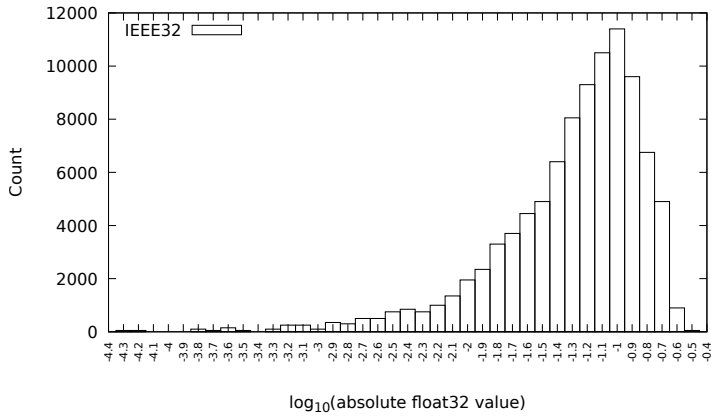


Fig. 11: Weight Histogram in layer *cccp6*

for weights, although in the case with AlexNet/ImageNet when the peak of the weight distribution (i.e. the highest frequency of weights) is more centered, the difference is less pronounced. In such cases, the weight values tend to fall outside of the optimal accuracy range of posits, and float formats with uniform accuracy distributions such as IEEE16 can achieve comparable training results.

With these observations, we deduce that the larger weight values which also occur more frequently inside the optimal accuracy range of posits, most likely contributes to posits' superior accuracy result of this benchmark.

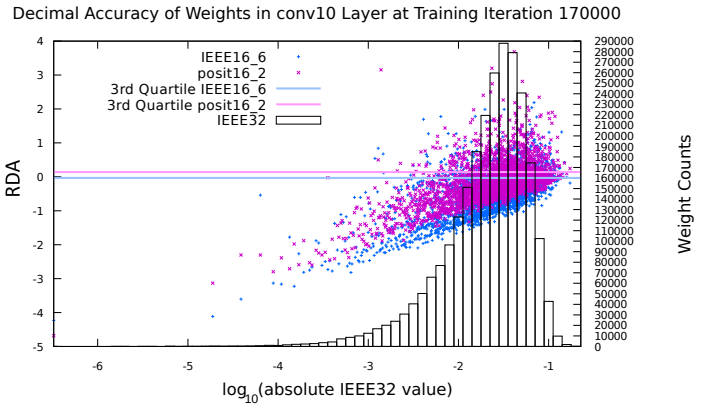


Fig. 12: Accuracy - Weights (SqueezeNet)

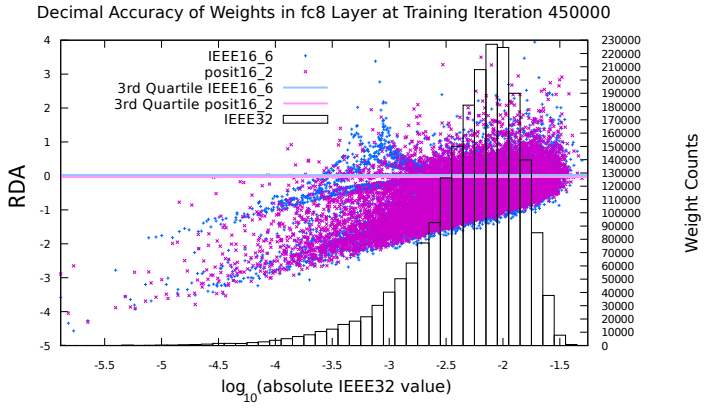


Fig. 13: Accuracy - Weights (AlexNet)

8 Shifting the Accuracy Peak

Based on our analysis, the accuracy peak of posits can be shifted into the desirable range of the large magnitude and high frequency weights to improve training accuracy. This technique can only be applied with posits, since float formats have uniform accuracy distributions (Figure 1) and no peak. Therefore, posits have the versatility to combat both dynamic range and precision issues in training while float formats can only overcome dynamic range problems. Because the bias for shifting can be determined offline after training a few iterations, this technique adds no additional overhead except for shifting the weights.

The peak of posit16_1's accuracy lies in the range $[1/4, 4]$ ($[-0.6, 0.6]$ in Figure 4). Figure 11 shows the value distribution of the layer *cccp6* where the larger weights begin around -0.6 on the x-axis. Based on this, we scaled posit16_1 by $1/4$ to shift its peak accuracy by $\log_{10}(1/4) \approx -0.6$. This improves the CIFAR/NIN accuracy to 55.58%, and increased the median value of weights of the layer from -0.1790 to -0.0747. Using the same technique we were able to drastically improve cuda-convnet/CIFAR10's accuracy for posit16_1 to 79.28% by using a bias of $1/64$. Figure 14 shows the IEEE32 weights (which we consider to be the actual weight value) and the difference in the accuracy of the posit weights in the last layer at the end of the training before and after the accuracy peak is shifted for this benchmark.

In our experience, because not all weight layers can be analyzed to figure out a bias value, the last weight layer of the network, which is also closest to the loss, can be used as a heuristic for this purpose. The distribution of the significant weights, once identified, can explain the differences of the posit results with different *es* values and also help pick which posit format and bias to use for training. For example, the larger the range of the distribution of values, the higher the *es* value that shows better performance for it. The bias can be

FP16 Format	Conversion	LUT	LUTMem	Reg	DSP	Depth
bfloat16	F	0	0	0	0	1
	B	3	0	0	0	1
DLFloat	F	27	0	18	0	2
	B	63	0	43	0	3
IEEE16	F	223	1	469	3	18
	B	325	0	278	3	12
IEEE16_6	F	217	1	462	3	18
	B	331	0	277	3	12
IEEE16_7	F	216	1	457	3	18
	B	341	0	278	3	12
posit16_1	F	115	0	84	0	3
	B	703	0	318	0	5
posit16_2	F	112	0	83	0	3
	B	723	0	316	0	5
posit16_3	F	118	0	84	0	3
	B	596	0	314	0	5

Table 3: Conversion in Hardware. 'F' is the conversion from the format to IEEE32, and 'B' is conversion from IEEE32 to the format. BRAM and DRAM are 0, and Initial Interval is 1, for all formats.

identified based on the values of the weights close to the peaks and calculating the distance from it to the accuracy peak. Other posit studies for training have suggested user layer-wise scaling, a technique also used with floats, to improve accuracy for low-precision training [15]. This can be costly because the scale needs to be calculated throughout the training for each type of parameter in each layer. Instead, our results indicate that for 16-bit posit training, looking at the weight distribution to calculate a bias and other posit parameters can suffice thus creating minimum overhead.

9 Discussion and Conclusion

Traditional FP16 formats studied so far for CNN training all have uniform accuracy distributions and differ mostly on their bit configuration. There is no consensus in the community as to which bit configuration is optimal. This has led to various silo works studying the effectiveness of each format and developing techniques to mitigate their individual shortcomings. However, there is no clear evidence for why those formats are the best for training versus other formats. While it is difficult to generalize numerical behavior for all neural networks, we believe our work provides useful insights through uniform (controlled) experiments. In summary:

– The IEEE 754 standard 16-bit format is inferior for out-of-the-box training of neural networks compared to the other float types.

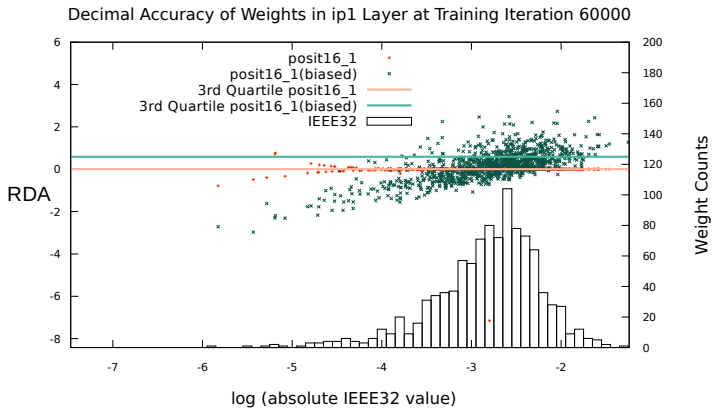


Fig. 14: Accuracy before and after shifting the accuracy peak - cuda-convnet/CIFAR10

- Non-uniform accuracy formats such as posits provide broader versatility for neural network training.
- We showed that analyzing the dynamic range and precision as they relate to the distribution of the weights is a useful indicator for selecting the FP16 format to use.
- We showed that the shifting the accuracy peak of posits leads to better results.

Although newer FP16 formats should still work in tandem with other improvements such as efficient training techniques and hyper parameter optimization, our goal in this work was to isolate the contribution of each format. Thus, we provided an unbiased evaluation of all existing 16-bit formats for training CNNs on a set of benchmarks of varying size in terms of accuracy and hardware performance. With our analysis we deduced that the superior accuracy of posits was due to the non-uniform error distribution which allows larger weights to have more accuracy. With this insight, we proposed an accuracy saving technique that shifts the peak posit accuracy into the desirable range. Best of all, it has no additional overhead. In this work we advocate for the design of 16-bit formats based on understanding the accuracy requirements of neural network training. We hope that it will guide the exploration of innovative non-traditional 16-bit, or even shorter, formats.

References

1. Agrawal, A., Fleischer, B., Mueller, S., Sun, X., Wang, N., Choi, J., Gopalakrishnan, K.: Dlfloat: a 16-bit floating point format designed for deep learning training and inference. In: 2019 IEEE 26th Symposium on Computer Arithmetic (ARITH). pp. 92–95. IEEE (2019)

2. Burgess, N., Milanovic, J., Stephens, N., Monachopoulos, K., Mansell, D.: Bfloat16 processing for neural networks. In: 2019 IEEE 26th Symposium on Computer Arithmetic (ARITH). pp. 88–91. IEEE (2019)
3. Committee, I.M.S.: IEEE Standard for Floating-Point Arithmetic. IEEE Std. 754-2019 (2019)
4. Das, D., Mellempudi, N., Mudigere, D., Kalamkar, D., Avancha, S., Banerjee, K., Sridharan, S., Vaidyanathan, K., Kaul, B., Georganas, E., et al.: Mixed precision training of convolutional neural networks using integer operations. arXiv preprint arXiv:1802.00930 (2018)
5. De Silva, H., Gustafson, J.L., Wong, W.F.: Making strassen matrix multiplication safe. In: 2018 IEEE 25th International Conference on High Performance Computing (HiPC). pp. 173–182. IEEE (2018)
6. Gupta, S., Agrawal, A., Gopalakrishnan, K., Narayanan, P.: Deep learning with limited numerical precision. In: International Conference on Machine Learning. pp. 1737–1746 (2015)
7. Ho, N.M., De Silva, H., Gustafson, J.L., Wong, W.F.: Qtorch+: Next generation arithmetic for pytorch machine learning. In: Conference on Next Generation Arithmetic. pp. 31–49. Springer (2022)
8. Ho, N.M., Nguyen, D.T., De Silva, H., Gustafson, J.L., Wong, W.F., Chang, I.J.: Posit arithmetic for the training and deployment of generative adversarial networks. In: 2021 Design, Automation & Test in Europe Conference & Exhibition (DATE). pp. 1350–1355. IEEE (2021)
9. Iandola, F.N., Han, S., Moskewicz, M.W., Ashraf, K., Dally, W.J., Keutzer, K.: Squeezenet: Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size. arXiv preprint arXiv:1602.07360 (2016)
10. Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., Darrell, T.: Caffe: Convolutional architecture for fast feature embedding. In: Proceedings of the 22nd ACM international conference on Multimedia. pp. 675–678. ACM (2014)
11. Kalamkar, D., Mudigere, D., Mellempudi, N., Das, D., Banerjee, K., Avancha, S., Vooturi, D.T., Jammalamadaka, N., Huang, J., Yuen, H., et al.: A study of bfloat16 for deep learning training. arXiv preprint arXiv:1905.12322 (2019)
12. Köster, U., Webb, T., Wang, X., Nassar, M., Bansal, A.K., Constable, W., Elibol, O., Gray, S., Hall, S., Hornof, L., et al.: Flexpoint: An adaptive numerical format for efficient training of deep neural networks. In: Advances in neural information processing systems. pp. 1742–1752 (2017)
13. Krizhevsky, A., Hinton, G., et al.: Learning multiple layers of features from tiny images. Citeseer (2009)
14. Lin, M., Chen, Q., Yan, S.: Network in network. arXiv preprint arXiv:1312.4400 (2013)
15. Lu, J., Lu, S., Wang, Z., Fang, C., Lin, J., Wang, Z., Du, L.: Training deep neural networks using posit number system. arXiv preprint arXiv:1909.03831 (2019)
16. Mellempudi, N., Srinivasan, S., Das, D., Kaul, B.: Mixed precision training with 8-bit floating point. arXiv preprint arXiv:1905.12334 (2019)
17. Micikevicius, P., Narang, S., Alben, J., Diamos, G., Elsen, E., Garcia, D., Ginsburg, B., Houston, M., Kuchaiev, O., Venkatesh, G., et al.: Mixed precision training. arXiv preprint arXiv:1710.03740 (2017)
18. Murillo, R., Del Barrio, A.A., Botella, G.: Deep pensieve: A deep learning framework based on the posit number system. Digital Signal Processing p. 102762 (2020)

19. Nvidia: Training mixed precision user guide. https://docs.nvidia.com/deeplearning/sdk/pdf/Training-Mixed-Precision-User-Guide.pdf (2020), accessed: 2020-01-07
20. Posit standard documentation (2022), https://posithub.org/docs/posit_standard-2.pdf, accessed: 2023-01-07
21. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al.: Imagenet large scale visual recognition challenge. International journal of computer vision **115**(3), 211–252 (2015)
22. Sun, X., Choi, J., Chen, C.Y., Wang, N., Venkataramani, S., Srinivasan, V.V., Cui, X., Zhang, W., Gopalakrishnan, K.: Hybrid 8-bit floating point (hfp8) training and inference for deep neural networks. In: Advances in Neural Information Processing Systems. pp. 4901–4910 (2019)
23. Sun, X., Wang, N., Chen, C.Y., Ni, J., Agrawal, A., Cui, X., Venkataramani, S., El Maghraoui, K., Srinivasan, V.V., Gopalakrishnan, K.: Ultra-low precision 4-bit training of deep neural networks. Advances in Neural Information Processing Systems **33** (2020)
24. Wang, N., Choi, J., Brand, D., Chen, C.Y., Gopalakrishnan, K.: Training deep neural networks with 8-bit floating point numbers. In: Advances in neural information processing systems. pp. 7675–7684 (2018)
25. Zhou, S., Wu, Y., Ni, Z., Zhou, X., Wen, H., Zou, Y.: Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. arXiv preprint arXiv:1606.06160 (2016)