You Already Have It: A Generator-Free Low-Precision DNN Training Framework using Stochastic Rounding

Geng Yuan[1*], Sung-En Chang[1*], Qing Jin[1], Alec Lu[2], Yanyu Li[1], Yushu Wu[1], Zhenglun Kong[1], Yanyue Xie[1], Peiyan Dong[1], Minghai Qin, Xiaolong Ma[3], Xulong Tang[4], Zhenman Fang[2], and Yanzhi Wang[1]

[1] Northeastern University, Boston, MA 02115, USA
[2] Simon Fraser University, Burnaby, B.C. V5A 1S6, Canada
[3] Clemson University, Clemson, SC 29634, USA
[4] University of Pittsburgh, Pittsburgh, PA 15260, USA

Abstract. Stochastic rounding is a critical technique used in low-precision deep neural networks (DNNs) training to ensure good model accuracy. However, it requires a large number of random numbers generated on the fly. This is not a trivial task on the hardware platforms such as FPGA and ASIC. The widely used solution is to introduce random number generators with extra hardware costs. In this paper, we innovatively propose to employ the stochastic property of DNN training process itself and directly extract random numbers from DNNs in a self-sufficient manner. We propose different methods to obtain random numbers from different sources in neural networks and a generator-free framework is proposed for low-precision DNN training on a variety of deep learning tasks. Moreover, we evaluate the quality of the extracted random numbers and find that high-quality random numbers widely exist in DNNs, while their quality can even pass the NIST test suite.

1 Introduction

To fully unleash the full power of the deep neural networks (DNNs) on various resource-constrained edge computing devices, DNN model compression [5,39,40] [34,26,17,29,28,50,31,22,18] has become the fundamental element and core enabler to bridge the gap between algorithm innovation and hardware implementation [30,11,35,25,27,48,47,6,49,21,12,13]. Recently, a surge of research efforts has been devoted to low-precision DNN training to better satisfy the limitation of the computation and storage resources on edge devices [52,53]. However, the commonly used rounding schemes, such as round-up (ceiling), round-down (floor), or rounding-to-nearest, usually lead to severe accuracy degradation in low-precision training scenarios [14]. The reason is that the small gradients below the minimum representation precision are always rounded to zero, hence an information loss for weight updates [16]. Therefore, prior works propose the stochastic rounding

* These authors contributed equally.

scheme to help preserve information and enable low-precision training achieving similar accuracy as the full-precision floating-point training [14,32].

Stochastic rounding is to round up or round down a number (e.g., the gradient) in a probabilistic way. And the probability is proportional to the proximity of the number and its nearby low-precision representation level. In specific, for each time of stochastic rounding and each number to be rounded, a random number needs to be generated to indicate the probabilistic rounding decision.

However, generating a large number of high-quality random numbers in parallel on hardware-based deep learning platforms such as FPGA and ASIC is not a trivial task [2,24]. The commonly used solution is to incorporate a large number of random number generators (RNGs) [19] to generate random numbers on the fly [38,32]. This will inevitably introduce extra hardware costs and complexity. Many prior works just assume the stochastic rounding can be appropriately incorporated in their design by default but barely care about its actual implementation [43,53,52]. For example, considering a representative FPGA-based DNN training design [23], incorporating stochastic rounding will increase 23% hardware costs (i.e., LUTs) [33] to fulfill its computation parallelism.

With the trend of stochastic rounding becoming a "must-do" step in low-precision DNN training [43,46,53,52], we may raise the following question. *Is there a more efficient way to obtain the random numbers without using extra hardware?* Fortunately, the answer is positive. One important thing that is neglected by the prior works is that the neural network training process is based on a certain degree of randomness, for example, the randomness introduced by the mini-batch training with stochastic gradient descent and the randomly shuffled training samples. This indicates that the neural network itself is supposed to be a potential source of random numbers.

In this paper, we innovatively propose to employ the stochastic property of the neural network training process to directly extract random numbers from neural networks. We consider the dynamically changed trainable parameters, training data, and intermediate results during the training process can be regarded as source data with randomness, and random numbers with arbitrary bits can be extracted from them. We propose two methods of random number extraction. One is to extract the corresponding number of bits directly from a source data, but the random numbers obtained in this way are heavily affected by the distribution of the source data. Therefore, we further propose the method of extracting the least significant bit (LSB) from multiple source data and synthesizing a random number with multiple bits (e.g., 8 bits). This method greatly improves the randomness of the obtained random numbers.

Based on that, we argue that when a random number is needed during DNN training, it is not necessary to use an additional random number generator to generate one because you already have it. We can directly extract random numbers from the network by leveraging the stochastic property of the DNN training process. Therefore, we propose a generator-free framework as a more flexible and hardware-economic solution for low-precision DNN training. And we utilize the

extracted random numbers for stochastic rounding in a self-sufficient manner to achieve high model accuracy.

We investigate the quality of random numbers extracted from different types of source data (e.g., trainable parameters, intermediate results). And we find that high-quality random numbers can be widely extracted from certain types of source data (e.g., the gradient of weights) without delicate selections. Most impressively, we find that the extracted random numbers can pass the entire NIST test suite SP800–22 [37], which is one of the most widely used testing suits for random numbers. Note that even the most widely used linear-feedback shift register (LFSR)-based random number generator fails some of the tests.

Moreover, besides obtaining random numbers with uniform distribution needed for stochastic rounding, we further propose a method that can obtain random numbers with arbitrary distributions (e.g., Gaussian and Beta distribution) using the pixels of training data. This further enhances the flexibility of our framework, and it is much more difficult to be achieved when using the conventional hardware-based random number generators.

To validate the effectiveness of our proposed methods, we conduct comprehensive experiments on a variety of deep learning tasks, including image classification task on CIFAR-10/100 dataset using ResNet-20/ResNet-32 and ImageNet dataset using ResNet-18, image super-resolution task on Set-5 and DIV2K using WDSR, and various natural language processing tasks using BERT. Compared to conventional methods that use random number generators, our *generator-free* methods can achieve similar accuracy with a 9% reduction in hardware costs.

The contributions of our paper are summarized as follows:

- Unlike the conventional methods that require many random number generators, we innovatively propose a generator-free framework, which directly extracts random numbers from the neural network by employing the randomness of the training process.
- We explore the validity of different sources for random number extraction. Then, we propose different random number extraction methods and analyze the quality of the extracted random numbers.
- Our methods can widely extract high-quality random numbers that can pass the entire NIST test suite SP800–22, while the widely used LFSR-based random number generator cannot.
- Besides successfully extracting the uniformly distributed random numbers for stochastic rounding, we further propose an image pixel-based method that can obtain random numbers with arbitrary distribution, which is hard to achieve using hardware random number generators.
- Finally, we validate the effectiveness of our generator-free framework on various tasks, including image classification, image super-resolution, and natural language processing. Our framework successfully achieves the same accuracy as the convention methods while eliminating the hardware costs of random number generators.

2 Background

2.1 Rounding Schemes

Rounding technique has been widely used in a range of scientific fields. It usually occurs when compressing the representation precision of a number. The rounding technique can be generally formulated as follows:

$$Round(x) = \begin{cases} \lfloor x \rfloor, & \text{with probability } p(x), \\ \lfloor x \rfloor + \epsilon, & \text{with probability } 1 - p(x), \end{cases} \tag{1}$$

where $Round(x)$ denotes the rounding scheme applied to a given value x. The $\lfloor x \rfloor$ represents to floor the x to the its nearest representation level. And ϵ is the representation precision. The probability $p(x) \in [0, 1]$ and the different rounding schemes can be distinguished by using different constraints on $p(x)$.

For example, the round-up (ceiling) or round-down (floor) scheme sets $p(x) = 0$ or $p(x) = 1$ consistently. On the other hand, the probability constraint can also relate to x, such as in the round-to-nearest scheme [1] and the stochastic rounding scheme [16,14]. In the round-to-nearest scheme, the probability is set to $p(x) = 1$ for $x \in [0, \frac{\epsilon}{2})$ and $p(x) = 0$ for $x \in [\frac{\epsilon}{2}, \epsilon)$. Instead of the deterministic rounding schemes above, the stochastic rounding scheme lets $p(x) = 1 - \frac{x - \lfloor x \rfloor}{\epsilon}$, making the expected rounding error to be zero, i.e., $\mathbb{E}(Round(x)) = x$. Therefore, stochastic rounding is considered an unbiased rounding scheme [45].

2.2 Stochastic Rounding in Low-Precision DNN Training

Due to the challenges of the intensive computation and storage in DNN training, quantization techniques are commonly used to save hardware resources, which is especially critical for resource-limited devices such as FPGAs and ASICs. In a low-precision DNN training process, data from several sources are mainly to be quantized, including weights, activations, gradients, and errors [44]. The later works [46,8] further propose to quantize the batch normalization (BN) and the optimizer. Among those sources, the model accuracy is largely sensitive to gradient quantization.

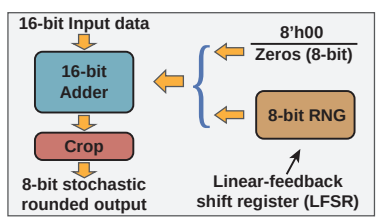

Fig. 1. Conventional stochastic rounding unit design on hardware.

The prior work [14] shows that the DNN training is hard to converge under 16-bit precision gradients when using the conventional rounding-to-nearest scheme. And the 16-bit precision is not even a considerably low-precision compared to 8-bit, 4-bit, even the binary precision. The reason is that when using the rounding-to-nearest scheme on low-precision gradients the small gradients

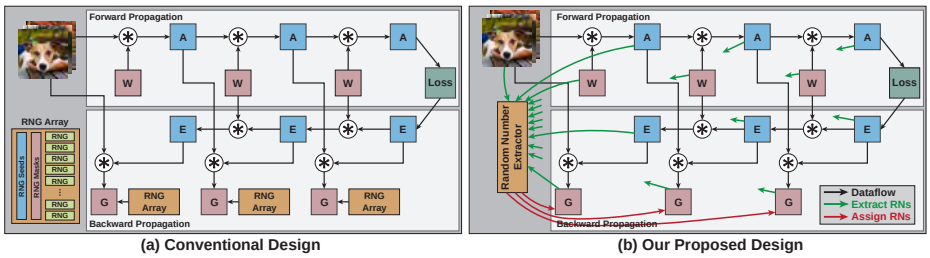

Fig. 2. Overall dataflow in DNN training and the comparison of (a) the conventional design that uses random number generators for stochastic rounding and (b) our proposed generator-free design. The A, W, E, and G stand for Activations, Weights, Errors, and Gradients, respectively. And the ⊛ represents the convolution operation.

below the minimum representation precision are always rounded to zero. This will incur the information loss for weight updates [16]. And by using stochastic rounding scheme, this issue can be mitigated [14]. With the help of stochastic rounding, recent works further quantize the gradients to 8-bit precision while still maintaining a comparable accuracy as the full-precision (i.e., floating-point 32 bits) training.

Many literatures that focus on algorithm optimization for low-precision training assume the stochastic rounding can be appropriately incorporated in their design by default [43,53,52], and they barely care about the actual implementation of stochastic rounding on hardware. However, implementing stochastic rounding for low-precision DNN training on hardware is not a trivial task. The stochastic rounding units (SRUs) are commonly used in prior designs [38,32]. Figure 1 shows the general SRU design on hardware. Assume 8-bit precision is used for DNN training. The 16-bit input data of a SRU is obtained from the convolution result of 8-bit activations and 8-bit errors. A 8-bit random number generator is needed in each SRU to generate random numbers on the fly. The linear-feedback shift register (LFSR) is usually used as the RNG [33]. The generated random number will concatenate with eight zeros as its higher bits and add to the input data. Then the stochastic rounded 8-bit gradient can be obtained by cropping the lower 8 bits away on the 16-bit output data of the adder.

3 A Generator-free Framework for Low-Precision DNN Training

3.1 Framework Overview

As we mentioned in Section 2.2, in order to mitigate the information loss and achieve high accuracy, the stochastic rounding technique is indispensable in low-precision DNN training. Since each gradient requires independent stochastic rounding and a RNG can only generate one random number at each time, in the conventional design, the RNG array modules are used to fulfill the computation

parallelism, as shown in Figure 2 (a). In each RNG array module, there are a large number of RNGs needed. This will introduce considerable hardware costs, especially for high throughput designs. And to make RNGs work independently (i.e., generate independent random numbers), each RNG also needs to have its corresponding seed and mask. This also introduces extra storage overhead.

In this work, we argue that it is unnecessary to use RNGs to generate random numbers during the DNN training process. The reason is that the DNN training process is based on a certain degree of randomness, such as the mini-batch training with stochastic gradient descent and the randomly shuffled training samples. We are supposed to find random numbers directly from the neural network. Therefore, we propose a generator-free framework for low-precision DNN training. As shown in Figure 2 (b), instead of using RNG arrays, we propose to use a random number extractor module to extract random numbers from different sources in the neural network (in Section 3.2). And we propose two methods to extract random numbers from the source data (in Section 3.3). Besides obtaining the uniformly distributed random numbers used for stochastic rounding, we also find a method that can obtain random numbers with arbitrary distribution (in Section 3.4).

3.2 Source of Random Numbers

We consider the accessible data during the neural network training as the source data. The source that can be potentially used for random number extracting should satisfy certain characteristics. The first characteristic is that the source data should be dynamically changing and have stochasticity over time during the training process. The second characteristic is that the source should have a large amount of source data that can fulfill the computation parallelism.

By considering the above characteristics, several candidate sources can be potentially used for random number extraction, including the trainable parameters, intermediate computation results, and input data from training samples, as shown in Figure 2 (b). However, not all the candidate sources are suitable for random number extraction. For example, the intermediate results (activations) after the ReLU layers will contain a large number of zeros, which significantly biased data distribution. Extracting random numbers from such sources cannot obtain high-quality random numbers with uniform distribution, which is desired for stochastic rounding. Therefore, the model accuracy will degrade considerably.

In this work, we explore the quality of random numbers extracted from different sources and evaluate the performance of low-precision training using the extracted random numbers. We find that the weights and the gradients of weights can generally be used as good sources to extract high-quality random numbers. On the contrary, the sources such as activations and the errors (the gradient of the Conv layers' outputs) are bad sources for random number extraction, and hence a bad low-precision training accuracy. Note that this bad accuracy can be improved using our proposed number-mapping strategy (will be explained in Section 3.3), but it is still lower than the accuracy achieved by using random

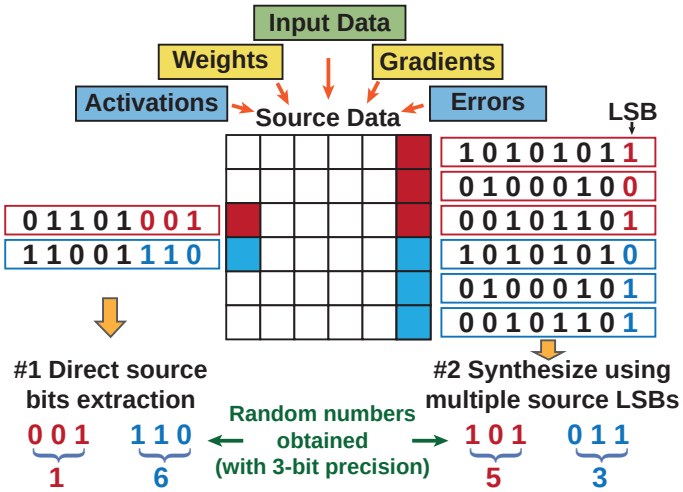

Fig. 3. The two proposed methods of extracting random numbers from source data in DNN training process.

numbers extracted from good sources. More details and comparison results are presented in Section 4.3.

3.3 Methods of Random Number Extraction

After determining the potential sources for random number extraction, we also need to have an appropriate method to extract the actual random numbers.

Method #1: direct source bits extraction. The first method is to directly extract random number with certain bits from the source data. As shown in the left-hand side of Figure 3, assuming the source data matrix/tensor could be one layer's weights, gradients, activations, errors, or input pixels of training samples, from each source data in the matrix/tensor we can extract a random number. The number of bits to extract depends on the required representation precision of the random number. We prefer to use the n lowest bits (e.g., 3 bits) since they are usually the fraction bits that will change frequently. The random numbers obtained in this method do have a certain degree of randomness; however, they are heavily affected by the distribution of the source data. If the distribution of source data is far from uniform, the accuracy of low-precision training will be compromised. This is because non-uniformly distributed random numbers introduce rounding bias during the training process, while the ideal stochastic rounding is unbiased.

Method #2: synthesize using multiple source LSBs. To overcome the non-ideal distribution of the source data, we further propose to synthesize a random number using the least significant bit (LSB) extracted from different source data. As shown in the right-hand side of Figure 3, a 3-bit random number is synthesized by using three LSBs from three different source data. The LSB

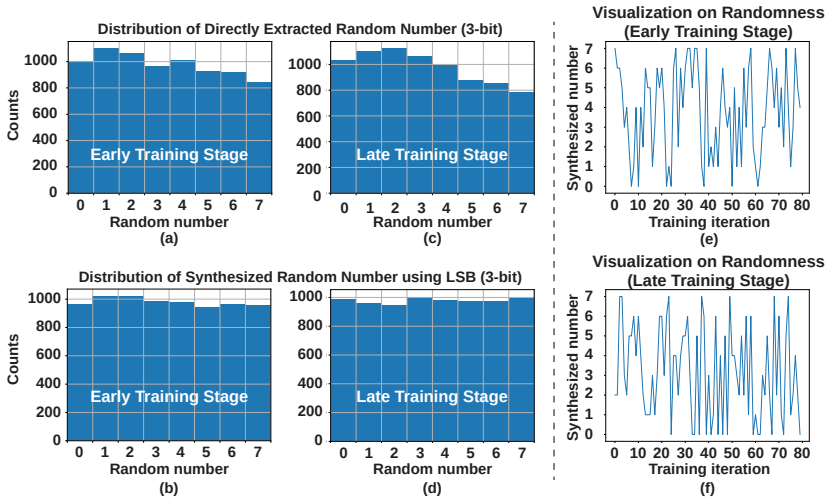

Fig. 4. The performance of the proposed two random number extraction methods in low-precision DNN training using ResNet-20 on CIFAR-10. (a), (b), (c), and (d) show the distribution of the extracted random numbers collected over 20 training epochs at the early training stage and late training stage, respectively. (e) and (f) show the changing trace (randomness) of the extracted (synthesized) random numbers at the early and late training stages, respectively.

of source data is the most frequently changed bit and is independent to other source data. Compared to method #1, the synthesized random numbers have much better quality that is closer to the ideal uniform distribution, making them valid for low-precision DNN training. Though synthesizing random numbers consumes more source data than directly extracting random numbers with certain bits, the synthesized random numbers are still enough to fulfill the computation parallelism due to the adequate sources for the extraction.

During DNN training, the extracted random numbers will be directly assigned to the corresponding stochastic rounding unit to replace the RNGs. In this way, the hardware costs of RNGs can be eliminated. Another advantage of our proposed methods is that the extracted/synthesized random numbers are free from periodic repetition, which is a common issue for the RNGs [36].

Figure 4 shows the quality of extracted random numbers obtained using our methods during a real low-precision DNN training process using ResNet-20 on CIFAR-10 dataset. We pick a location from the second last layer's gradients and observed the random numbers extracted from the same source data location (three locations are picked for the synthesizing method). We collect the extracted random numbers over 20 training epochs. From Figure 4 (a), (b), (c), and (d) we can see that the synthesizing method provides random numbers with higher quality than the directly extracting method in both the early training stage and late training stage. And Figure 4 (e) and (f) shows the synthesized random

numbers obtained in 80 consecutive training iterations. The high randomness can be observed in both the early training stage and the late training stage.

Extraction with Number-Mapping Strategy. As we mentioned in Section 3.2, some source data such as activations (has a large number of zeros) are significantly biased. None of the above two extraction methods can obtain high-quality random numbers. When using such bad random numbers for stochastic rounding in low-precision training, the network even cannot converge. However, if we extract random numbers (e.g., 3-bits) and use a simple mapping strategy which maps the extracted even numbers to the middle numbers within the range (e.g., $\{0, 1, 2, 3, 4, 5, 6, 7\} \rightarrow \{3, 1, 2, 7, 4, 6, 5, 0\}$), then the extracted random numbers will form a bell-like distribution. This can effectively improve the trained model accuracy. Though this method cannot outperform the model accuracy obtained by extracting random numbers from good sources, it still has better accuracy than using nearest rounding. The reason is that the bell-like distribution makes the rounding becomes a hybrid scheme between the ideal stochastic rounding (with uniform random number distribution) and nearest rounding. It mitigates the biases introduced by the bad data distribution while preserving a certain degree of randomness. But we still suggest directly extracting high-quality random numbers from good sources. More details are discussed in Appendix.

3.4 Obtaining Random Numbers with Arbitrary Distribution

In our generator-free framework, besides extracting the uniformly distributed random numbers for stochastic rounding, we further propose a novel image-pixel-based random number extraction method that can extract random numbers with an arbitrary distribution. And this is hard to be achieved in conventional methods using fixed hardware random number generators.

As shown in Figure 5, in DNN training, the training samples in a training dataset will be divided into several mini-batches. If we look at the value of the same pixel location over a training epoch (i.e., n mini-batches), we will find the values of a pixel location scattered within a range following a certain distribution. In each training epoch, every training sample is guaranteed to be used once, but the order of the training samples used is different due to the dataset shuffling for each training epoch. This indicates two facts: ① the value distribution of a pixel location will remain the same over the entire training process; ② the order of values presented in a pixel location is varied between different epochs. These two properties give us a unique opportunity for random number extraction. In our method, we divide the value range of a pixel location into 2^n intervals (e.g., 8 intervals for 3-bit random number, as shown in Figure 5) using $2^n - 1$ threshold levels. Then, depending on the interval in which the pixel value is located, the corresponding random number can be obtained. Since the value distributions of different pixel locations are varied, each pixel requires its own threshold levels. The threshold levels are unique for a certain dataset and can be easily obtained offline. For each training mini-batch, each pixel of input image can create a random number.

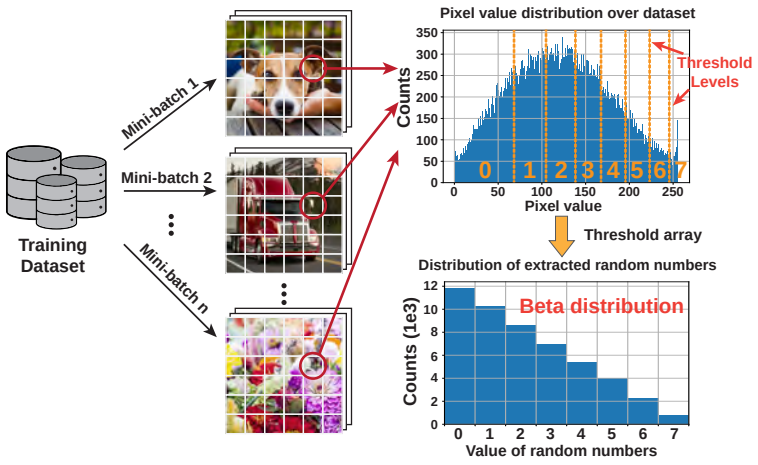

Fig. 5. Extract random numbers with arbitrary distribution using the image-pixel-based extraction method.

It is worth noting that the content in the edge pixels of images are often the background. As a result, the values of those edge pixels are more likely to be concentrated in very large or small, making it possible that some intervals are indistinguishable and fail to obtain the well-distributed random numbers. Therefore, we exclude the edge pixels for random number extraction.

In Figure 5, we show a real example that uses our image-pixel-based method obtaining random numbers with the beta distribution ($\alpha = 1, \beta = 2$). It can be observed that a high-quality beta distribution is obtained. Besides, we can also obtain other desired distributions by simply modifying the threshold levels. And the threshold arrays can be easily generated offline by going over the training dataset once. More details are shown in Appendix.

4 Results

4.1 Experiments Setup

In this Section, we evaluate our proposed framework and methods on various practical deep learning tasks, including image classification task, image super-resolution task , and natural language processing tasks. All of our models including the full-precision (FP) models and the low-precision (LP) models are trained on a GPU server with 4× NVIDIA 2080Ti GPUs. The PyTorch framework is used for model training. For the low-precision training, we quantize all weights, activations, gradients, and errors using fewer bits (detailed numbers are given in corresponding result tables). The "Ours" method in the result tables indicates the results obtained using our generator-free framework, which extracts random numbers from the network as we proposed. To make fair comparisons, in the

Table 1. Accuracy comparison of ResNet-20 and ResNet-32 on CIFAR-10 and CIFAR-100 dataset. W: weight, A: activation, G: gradient.

Model	Method	Precision (W/A/G)	Generator Free	Accuracy	Model	Method	Precision (W/A/G)	Generator Free	Accuracy
CIFAR10					CIFAR100				
ResNet-20	FP	32-bit	-	92.17	ResNet-32	FP	32-bit	-	74.53
	Zhu et al. [53]	8-bit	×	92.12		$LP_{stochastic}$	8-bit	×	74.47
	Ours	8-bit	✓	92.15		**Ours**	8-bit	✓	74.41
	$LP_{stochastic}$	6-bit	×	91.83		$LP_{stochastic}$	6-bit	×	74.03
	Ours	6-bit	✓	91.88		**Ours**	6-bit	✓	74.06

"LP" method, we adopt the same training hyperparameters as our generator-free method, but with the random numbers generated from the simulated RNGs. We simulate the behavior of LFSR-based RNGs to generate random numbers for the "LP" method. For all our results, if not specified, they are extracted from the gradients of the model's second layer by synthesizing using multiple source LSBs method (i.e., Method#2) and without using number-mapping strategy. We mainly evaluate four different extraction sources including the activations (i.e., Conv outputs, before BN and ReLU), errors (the gradients of activations), weights, and gradient (of weights).

4.2 Accuracy of Low-precision DNN Training

Accuracy for Image Classification. We evaluate our framework on CIFAR-10/100 dataset [20] using ResNet-20/32 [15]. The results are shown in Table 1. The 8-bit precision and 6-bit precision are used, respectively. For the 8-bit precision results on CIFAR-10, we compare our method with Zhu et al. [53]. Both of methods achieve similar accuracy as the FP result, thanks to the superiority of stochastic rounding. However, 16-bit RNGs are required in Zhu et al. [53], while we are generator-free. With the 6-bit precision, though both the LP method and our method have around 0.3% accuracy degradation, our method still achieves similar accuracy as the LP method. For the results on CIFAR-100 using ResNet-32, since the CIFAR-100 task is harder than CIFAR-10, all results with 8-bit precision start to have a minor accuracy degradation compared to the FP result. And for both 8-bit precision and 6-bit precision, our generator-free method can achieve similar accuracy as the LP method that requires RNGs.

Table 2 shows the results on ImageNet dataset [7] using ResNet-18. Our generator-free method achieve the same accuracy as the floating point training result. We have a clear advantage compared to nearest rounding method. More importantly, our method also outperforms the hardware random number generator-based method ($LP_{stochastic}$) with a 0.4% margin. The reason is that our extracted random numbers have higher quality, which has a more critical impact on the harder dataset (ImageNet v.s. CIFAR).

Accuracy for Other Tasks. Table 3 shows the comparison results for image super-resolution task using WDSR-B network [10]. The model is trained on

Table 2. Comparison with existing works using ResNet-18 on ImageNet dataset.

Model	Method	Precision (W/A/G)	Generator Free	Accuracy
ResNet-18	WAGEUBN [46]	8-bit	×	67.40
	FP8 [43]	8-bit	×	67.34
	Uint8 [53]	8-bit	×	69.67
	ADint8 [52]	8-bit	×	70.21
	FP	32-bit	-	71.12
	$LP_{nearest}$	8-bit	-	68.13
	$LP_{stochastic}$	8-bit	×	70.72
	Ours	8-bit	✓	71.10

DIV2K [41] dataset and tested on Set-5 [3] and DIV2K dataset. We evaluate our framework using input size of 640×360 and 320×180 for 2× and 4× resolution up scaling, respectively. And Table 4 shows the evaluation results for natural language processing (NLP) tasks using BERT [9] on a variety of datasets from the General Language Understanding Evaluation (GLUE) [42] benchmark. From the results, we can observe that the image super-resolution task is more sensitive to low-precision training compared to classification and NLP tasks. Our generator-free method consistently achieves similar accuracy as the conventional method, which validates the effectiveness of our proposed methods and framework.

4.3 Comparison of Different Extraction Sources

We first investigate the low-precision training model accuracy using extracted random numbers from different sources including activations, errors, weights, and gradients. And for each type of source, we also cover different layers from the first, middle, and last part of the model. From Table 5 we can find consistent phenomenons on both datasets. The errors are the worst source for random number extraction, and its later layers are relatively better than the front layers. This is because the errors generally contain a dominant number of zeros in low-precision training, which will significantly bias the distribution of random numbers. The weights and gradients are the two of the good sources which can achieve high training accuracy because they generally tend to have a normal

Table 3. Accuracy comparison for image super-resolution using WDSR-B network on SET-5 and DIV2K dataset.

Upscale rate	Method	bits	PSNR	PSNR_Y	SSIM	PSNR	PSNR_Y	SSIM
			SET-5			DIV2K		
2x	FP	32	34.9591	37.0072	0.9564	33.3994	34.9098	0.9356
	$LP_{stochastic}$	8	34.1163	36.4235	0.9516	32.8305	34.5763	0.9317
	Ours	8	34.1971	36.5799	0.9535	32.8124	34.5243	0.9308
4x	FP	32	28.8708	30.6974	0.8691	27.9794	29.4714	0.8133
	$LP_{stochastic}$	8	28.3235	30.2768	0.8647	27.6667	29.2359	0.8045
	Ours	8	28.3351	30.2786	0.8662	27.6182	29.2295	0.8039

Table 4. Accuracy comparison for natural language processing tasks using BERT.

Method	bits	MRPC	STS-B	RTE	COLA	MNLI	QQP	SST2	QNLI	Avg.
FP	32	89.66	89.19	66.43	57.27	84.37	91.18	92.66	91.40	82.77
$LP_{stochastic}$	8	89.58	88.86	66.43	56.92	84.25	91.13	92.61	91.22	82.61
Ours	8	89.62	88.82	66.37	57.04	84.35	90.92	92.65	91.14	82.62

Table 5. Comparison of low-precision training model accuracy using extracted random numbers from different sources. We use 6-bit quantization on weights, activations, errors, and gradeints. 6-bit random numbers are extracted for stochastic rounding.

Model	Source	layer-2	layer-10	layer-19	Model	Source	layer-2	layer-18	layer-31
CIFAR10					CIFAR100				
ResNet-20	Activation	90.36	89.97	87.40	ResNet-32	Activation	70.19	70.41	68.66
	Error	71.11	72.87	74.91		Error	8.34	11.59	54.22
	Weight	91.76	91.69	91.62		Weight	73.32	73.30	73.39
	Gradient	91.88	91.47	91.68		Gradient	74.06	73.91	73.87

distribution which will make the LSB has relatively balanced zeros and ones. However, based on our observations, we found the weights have a much lower flipping rate on their LSB compared to the gradients. We conjecture that this is caused by the low-precision representation in training, which limits the small perturbation on weights to a certain degree compared to the floating-point 32-bit training. This makes the gradient a better source for random number extraction. The evaluation of information entropy will be shown in Section 4.4 and more discussions can be found in Appendix.

4.4 Randomness Tests and Hardware Saving

To quantitatively evaluate the quality of the extracted random numbers, we test the information entropy of the random numbers throughout the training process. The results are shown in Table 6. We can find that the random numbers extracted from weight and gradient have highest information entropy and they are close to 6-bit. Combining the accuracy results from the Table 5, we can find a positive relationship between the information entropy and the final low-precision training accuracy. We also test the quality of extracted random numbers using the widely used NIST test suite SP800–22 [37]. And we surprisingly find that the random numbers extracted from gradients can generally pass the entire tests, which indicates the random numbers has very high quality. Note that even the widely used LFSR-based random number generators can fail the NIST test. More details are discussed in Appendix.

From the hardware perspective, compared to hardware-based random number generators, our generator-free approach does not require any arithmetic or logical expressions (do no use LUT), and thus is more hardware-friendly and enables resource-efficient design implementations. We estimate the LUT saving using Xilinx Vitis high-level synthesis (HLS) tool and based on the representative

Table 6. The information entropy of extracted random numbers. The results are tested on CIFAR-100 dataset ResNet-32 with 6-bit quantization and 6-bit random numbers.

Source	Ideal	Activation	Error	Weight	Gradient
Entropy (bits)	6	4.9738	2.8017	5.8429	5.8774

FPGA-based DNN training design methodology [23] with LFSR-based stochastic rounding unit (SRU) design [33]. We can successfully save 38.5% and 9% LUTs costs of the SRU and the overall design, respectively. Note that the LUTs are considered a tight resource in FPGA-based DNN training design. Generally, a design prefers not to reach a high LUTs utilization rate since it will lead to routing problems on the FPGA that can significantly slow down the working frequency [51]. Even in the FPGA vendor's (Xilinx) design [4], the LUT utilization does not pass 70% on edge FPGA (PYNQ-Z1) or 50% on cloud FPGAs (AWS F1). Therefore, the LUTs reduction achieved by our method is considerable.

4.5 Discussion and Future Works

In this paper, we pave a new way to obtain random numbers for low-precision training. We explore the performance of different types of sources heuristically, and a systematic exploration can be done for future works. By finding a large number of high-quality random numbers that can be easily extracted from DNNs, our work may also inspire more research in the security field. Moreover, besides the random numbers used by stochastic rounding, our image-pixel-based method can also extract random numbers with arbitrary distribution, which can be potentially used for a broader range of tasks and is worth being further investigated.

5 Conclusion

In this paper, we argue that when random numbers are needed during the DNN training, it is unnecessary to pay extra hardware costs for random number generators because we already have them. Therefore, we explore the validity of different sources and methods for high-quality random number extraction. We propose a generator-free framework to extract and use the random numbers during low-precision DNN training. Moreover, we propose an image-pixel-based method that can extract random numbers with arbitrary distribution, which is hard to achieve using hardware random number generators. Our framework successfully achieves the same accuracy as the convention methods while eliminating the hardware costs of random number generators.

Acknowledgement

This work was partly supported by NSF CCF-1919117 and CCF-1937500; NSERC Discovery Grant RGPIN-2019-04613, DGECR-2019-00120, Alliance Grant ALLRP-552042-2020; CFI John R. Evans Leaders Fund.

References

1. Ieee standard for floating-point arithmetic. IEEE Std 754-2019 (Revision of IEEE 754-2008) pp. 1–84 (2019). https://doi.org/10.1109/IEEESTD.2019.8766229
2. Best, S., Xu, X.: An all-digital true random number generator based on chaotic cellular automata topology. In: 2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD). pp. 1–8. IEEE (2019)
3. Bevilacqua, M., Roumy, A., Guillemot, C., Alberi-Morel, M.L.: Low-complexity single-image super-resolution based on nonnegative neighbor embedding. In: Proceedings of the British Machine Vision Conf. pp. 135.1–135.10. BMVA Press (2012)
4. Blott, M., Preußer, T.B., Fraser, N.J., Gambardella, G., O'brien, K., Umuroglu, Y., Leeser, M., Vissers, K.: Finn-r: An end-to-end deep-learning framework for fast exploration of quantized neural networks. ACM Transactions on Reconfigurable Technology and Systems (TRETS) **11**(3), 1–23 (2018)
5. Chang, S.E., Li, Y., Sun, M., Shi, R., So, H.K.H., Qian, X., Wang, Y., Lin, X.: Mix and match: A novel fpga-centric deep neural network quantization framework. In: 2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA). pp. 208–220. IEEE (2021)
6. Chu, C., Wang, Y., Zhao, Y., Ma, X., Ye, S., Hong, Y., Liang, X., Han, Y., Jiang, L.: Pim-prune: Fine-grain dcnn pruning for crossbar-based process-in-memory architecture. In: 2020 57th ACM/IEEE Design Automation Conference (DAC). pp. 1–6. IEEE (2020)
7. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: Imagenet: A large-scale hierarchical image database. In: 2009 IEEE conference on computer vision and pattern recognition. pp. 248–255. Ieee (2009)
8. Dettmers, T., Lewis, M., Shleifer, S., Zettlemoyer, L.: 8-bit optimizers via block-wise quantization (2021)
9. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805 (2018)
10. Fan, Y., Yu, J., Huang, T.S.: Wide-activated deep residual networks based restoration for bpg-compressed images. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops. pp. 2621–2624 (2018)
11. Fang, H., Mei, Z., Shrestha, A., Zhao, Z., Li, Y., Qiu, Q.: Encoding, model, and architecture: Systematic optimization for spiking neural network in fpgas. In: 2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD). pp. 1–9. IEEE (2020)
12. Fang, H., Shrestha, A., Zhao, Z., Qiu, Q.: Exploiting neuron and synapse filter dynamics in spatial temporal learning of deep spiking neural network. In: Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence. IJCAI'20 (2021)
13. Fang, H., Taylor, B., Li, Z., Mei, Z., Li, H.H., Qiu, Q.: Neuromorphic algorithm-hardware codesign for temporal pattern learning. In: 2021 58th ACM/IEEE Design Automation Conference (DAC). pp. 361–366. IEEE (2021)
14. Gupta, S., Agrawal, A., Gopalakrishnan, K., Narayanan, P.: Deep learning with limited numerical precision. In: International conference on machine learning. pp. 1737–1746. PMLR (2015)
15. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR). pp. 770–778 (2016)

16. Höhfeld, M., Fahlman, S.E.: Probabilistic rounding in neural network learning with limited precision. Neurocomputing **4**(6), 291–299 (1992)
17. Hou, Z., Qin, M., Sun, F., Ma, X., Yuan, K., Xu, Y., Chen, Y.K., Jin, R., Xie, Y., Kung, S.Y.: Chex: Channel exploration for cnn model compression. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 12287–12298 (2022)
18. Kong, Z., Dong, P., Ma, X., Meng, X., Niu, W., Sun, M., Yuan, G., Ren, B., Qin, M., Tang, H., Wang, Y.: Spvit: Enabling faster vision transformers via soft token pruning. arXiv preprint arXiv:2112.13890 (2021)
19. Krawczyk, H.: Lfsr-based hashing and authentication. In: Annual International Cryptology Conference. pp. 129–139. Springer (1994)
20. Krizhevsky, A., Hinton, G., et al.: Learning multiple layers of features from tiny images (2009)
21. Li, Y., Fang, H., Li, M., Ma, Y., Qiu, Q.: Neural network pruning and fast training for drl-based uav trajectory planning. In: 2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC). pp. 574–579. IEEE (2022)
22. Liu, N., Yuan, G., Che, Z., Shen, X., Ma, X., Jin, Q., Ren, J., Tang, J., Liu, S., Wang, Y.: Lottery ticket preserves weight correlation: Is it desirable or not? In: International Conference on Machine Learning (ICML). pp. 7011–7020. PMLR (2021)
23. Luo, C., Sit, M.K., Fan, H., Liu, S., Luk, W., Guo, C.: Towards efficient deep neural network training by fpga-based batch-level parallelism. Journal of Semiconductors **41**(2), 022403 (2020)
24. Luo, Y., Wang, W., Best, S., Wang, Y., Xu, X.: A high-performance and secure trng based on chaotic cellular automata topology. IEEE Transactions on Circuits and Systems I: Regular Papers **67**(12), 4970–4983 (2020)
25. Ma, X., Guo, F.M., Niu, W., Lin, X., Tang, J., Ma, K., Ren, B., Wang, Y.: PCONV: The missing but desirable sparsity in DNN weight pruning for real-time execution on mobile devices. In: Proceedings of the AAAI Conference on Artificial Intelligence (AAAI). vol. 34, pp. 5117–5124 (2020)
26. Ma, X., Lin, S., Ye, S., He, Z., Zhang, L., Yuan, G., Tan, S.H., Li, Z., Fan, D., Qian, X., et al.: Non-structured dnn weight pruning–is it beneficial in any platform? IEEE Transactions on Neural Networks and Learning Systems (TNNLS) (2021)
27. Ma, X., Niu, W., Zhang, T., Liu, S., Lin, S., Li, H., Wen, W., Chen, X., Tang, J., Ma, K., et al.: An image enhancing pattern-based sparsity for real-time inference on mobile devices. In: Proceedings of the European conference on computer vision (ECCV). pp. 629–645. Springer (2020)
28. Ma, X., Qin, M., Sun, F., Hou, Z., Yuan, K., Xu, Y., Wang, Y., Chen, Y.K., Jin, R., Xie, Y.: Effective model sparsification by scheduled grow-and-prune methods. In: Proceedings of the International Conference on Learning Representations (ICLR) (2021)
29. Ma, X., Yuan, G., Li, Z., Gong, Y., Zhang, T., Niu, W., Zhan, Z., Zhao, P., Liu, N., Tang, J., et al.: Blcr: Towards real-time dnn execution with block-based reweighted pruning. In: International Symposium on Quality Electronic Design (ISQED). pp. 1–8. IEEE (2022)
30. Ma, X., Yuan, G., Lin, S., Ding, C., Yu, F., Liu, T., Wen, W., Chen, X., Wang, Y.: Tiny but accurate: A pruned, quantized and optimized memristor crossbar framework for ultra efficient dnn implementation. In: 2020 25th Asia and South Pacific design automation conference (ASP-DAC). pp. 301–306. IEEE (2020)

31. Ma, X., Yuan, G., Shen, X., Chen, T., Chen, X., Chen, X., Liu, N., Qin, M., Liu, S., Wang, Z., et al.: Sanity checks for lottery tickets: Does your winning ticket really win the jackpot? Advances in Neural Information Processing Systems (NeurIPS) **34** (2021)
32. Mikaitis, M.: Stochastic rounding: Algorithms and hardware accelerator. In: 2021 International Joint Conference on Neural Networks (IJCNN). pp. 1–6. IEEE (2021)
33. Na, T., Ko, J.H., Kung, J., Mukhopadhyay, S.: On-chip training of recurrent neural networks with limited numerical precision. In: 2017 International Joint Conference on Neural Networks (IJCNN). pp. 3716–3723. IEEE (2017)
34. Niu, W., Li, Z., Ma, X., Dong, P., Zhou, G., Qian, X., Lin, X., Wang, Y., Ren, B.: Grim: A general, real-time deep learning inference framework for mobile devices based on fine-grained structured weight sparsity. IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI) (2021)
35. Niu, W., Ma, X., Lin, S., Wang, S., Qian, X., Lin, X., Wang, Y., Ren, B.: Patdnn: Achieving real-time dnn execution on mobile devices with pattern-based weight pruning. In: Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS). pp. 907–922 (2020)
36. Roth Jr, C.H., John, L.K.: Digital systems design using VHDL. Cengage Learning (2016)
37. Rukhin, A., Soto, J., Nechvatal, J., Smid, M., Barker, E.: A statistical test suite for random and pseudorandom number generators for cryptographic applications. Tech. rep., Booz-allen and hamilton inc mclean va (2001)
38. Su, C., Zhou, S., Feng, L., Zhang, W.: Towards high performance low bitwidth training for deep neural networks. Journal of Semiconductors **41**(2), 022404 (2020)
39. Sun, M., Li, Z., Lu, A., Li, Y., Chang, S.E., Ma, X., Lin, X., Fang, Z.: Film-qnn: Efficient fpga acceleration of deep neural networks with intra-layer, mixed-precision quantization. In: Proceedings of the 2022 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. pp. 134–145 (2022)
40. Sun, M., Ma, H., Kang, G., Jiang, Y., Chen, T., Ma, X., Wang, Z., Wang, Y.: Vaqf: Fully automatic software-hardware co-design framework for low-bit vision transformer. arXiv preprint arXiv:2201.06618 (2022)
41. Timofte, R., Gu, S., Wu, J., Van Gool, L.: Ntire 2018 challenge on single image super-resolution: Methods and results. In: Proceedings of the IEEE conference on computer vision and pattern recognition workshops. pp. 852–863 (2018)
42. Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., Bowman, S.R.: Glue: A multi-task benchmark and analysis platform for natural language understanding. In: Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP. pp. 353–355 (2018)
43. Wang, N., Choi, J., Brand, D., Chen, C.Y., Gopalakrishnan, K.: Training deep neural networks with 8-bit floating point numbers. Advances in neural information processing systems **31** (2018)
44. Wu, S., Li, G., Chen, F., Shi, L.: Training and inference with integers in deep neural networks. arXiv preprint arXiv:1802.04680 (2018)
45. Xia, L., Anthonissen, M., Hochstenbach, M., Koren, B.: A simple and efficient stochastic rounding method for training neural networks in low precision. arXiv preprint arXiv:2103.13445 (2021)
46. Yang, Y., Deng, L., Wu, S., Yan, T., Xie, Y., Li, G.: Training high-performance and large-scale deep neural networks with full 8-bit integers. Neural Networks **125**, 70–82 (2020)

47. Yuan, G., Behnam, P., Cai, Y., Shafiee, A., Fu, J., Liao, Z., Li, Z., Ma, X., Deng, J., Wang, J., et al.: Tinyadc: Peripheral circuit-aware weight pruning framework for mixed-signal dnn accelerators. In: 2021 Design, Automation & Test in Europe Conference & Exhibition (DATE). pp. 926–931. IEEE (2021)
48. Yuan, G., Liao, Z., Ma, X., Cai, Y., Kong, Z., Shen, X., Fu, J., Li, Z., Zhang, C., Peng, H., et al.: Improving dnn fault tolerance using weight pruning and differential crossbar mapping for reram-based edge ai. In: 2021 22nd International Symposium on Quality Electronic Design (ISQED). pp. 135–141. IEEE (2021)
49. Yuan, G., Ma, X., Ding, C., Lin, S., Zhang, T., Jalali, Z.S., Zhao, Y., Jiang, L., Soundarajan, S., Wang, Y.: An ultra-efficient memristor-based dnn framework with structured weight pruning and quantization using admm. In: 2019 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED). pp. 1–6. IEEE (2019)
50. Yuan, G., Ma, X., Niu, W., Li, Z., Kong, Z., Liu, N., Gong, Y., Zhan, Z., He, C., Jin, Q., et al.: Mest: Accurate and fast memory-economic sparse training framework on the edge. Advances in Neural Information Processing Systems (NeurIPS) **34** (2021)
51. Zhang, C., Sun, G., Fang, Z., Zhou, P., Pan, P., Cong, J.: Caffeine: Toward uniformed representation and acceleration for deep convolutional neural networks. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **38**(11), 2072–2085 (2018)
52. Zhao, K., Huang, S., Pan, P., Li, Y., Zhang, Y., Gu, Z., Xu, Y.: Distribution adaptive int8 quantization for training cnns. In: Proceedings of the Thirty-Fifth AAAI Conference on Artificial Intelligence (2021)
53. Zhu, F., Gong, R., Yu, F., Liu, X., Wang, Y., Li, Z., Yang, X., Yan, J.: Towards unified int8 training for convolutional neural network. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 1969–1979 (2020)

Appendix

A Discussion of Extraction with Number-Mapping Strategy

As we mentioned in the main paper (Section 3.3), extracting random numbers from sources with biased distribution will lead to low-precision training hard to converge to good accuracy. But the accuracy can be significantly improved by adopting a number-mapping strategy. For example, we map the original random number to a new representation, e.g., $\{0, 1, 2, 3, 4, 5, 6, 7\} \rightarrow \{3, 1, 2, 7, 4, 6, 5, 0\}$. In other words, the extracted '0' will be converted to '3' and so on during the stochastic rounding process. In this way, if '0' takes the majority of all extracted random numbers, by converting it to '3' (a middle number), it will not make the rounding decision significantly biased to rounding down.

Table A1. Comparison of low-precision training model accuracy using extracted random numbers from different sources. The "NM" stands for the results with number-mapping strategy. We use 6-bit quantization on weights, activations, errors, and gradeints. 6-bit random numbers are extracted for stochastic rounding.

Source	layer-2	layer-10	layer-19	layer-2	layer-18	layer-31
	CIFAR10 on ResNet20			CIFAR100 on ResNet32		
Activation	90.36	89.97	87.40	70.19	70.41	68.66
Activation w/ NM	90.47	90.12	87.73	71.95	72.36	72.74
Error	71.11	72.87	74.91	8.34	11.59	54.22
Error w/ NM	86.97	86.53	90.67	73.10	72.64	72.47
Weight	91.76	91.69	91.62	73.32	73.30	73.39
Weight w/ NM	91.78	91.67	91.64	73.37	73.57	73.63
Gradient	91.88	91.47	91.68	74.06	73.91	73.87
Gradient w/ NM	91.81	91.53	91.69	74.04	73.95	73.83

Table A1 shows the comparison between the accuracy with and without using number-mapping strategy. We can observed that using number-mapping strategy can significantly improve the model accuracy when using extracting random numbers from bad sources (i.e., activations and errors), while it cannot further improve the results from sources with good distribution (i.e., gradients).

B Visualization of Random Number Extraction on Different Sources

Figure A1 shows the visualization of LSB's changing trend of a specific source location along the entire training process on CIFAR-100 using ResNet32. In specific, we randomly select a location from a type of source (e.g., the 3rd weight of a layer) and we keep tracking the value of its LSB along the entire training process. We reshape the 1-D (row) view to the 2-D format for better visualization. Each pixel represents the LSB value at a certain training iteration. And the beige and black color represent value of '1' and '0', respectively. And we can see that the LSB in the error is '0' for the most of time, and the '1' and '0' in the gradient have randomly and evenly distributed. And the model accuracy results also show that using gradients as the source of random number extraction is the best choice.

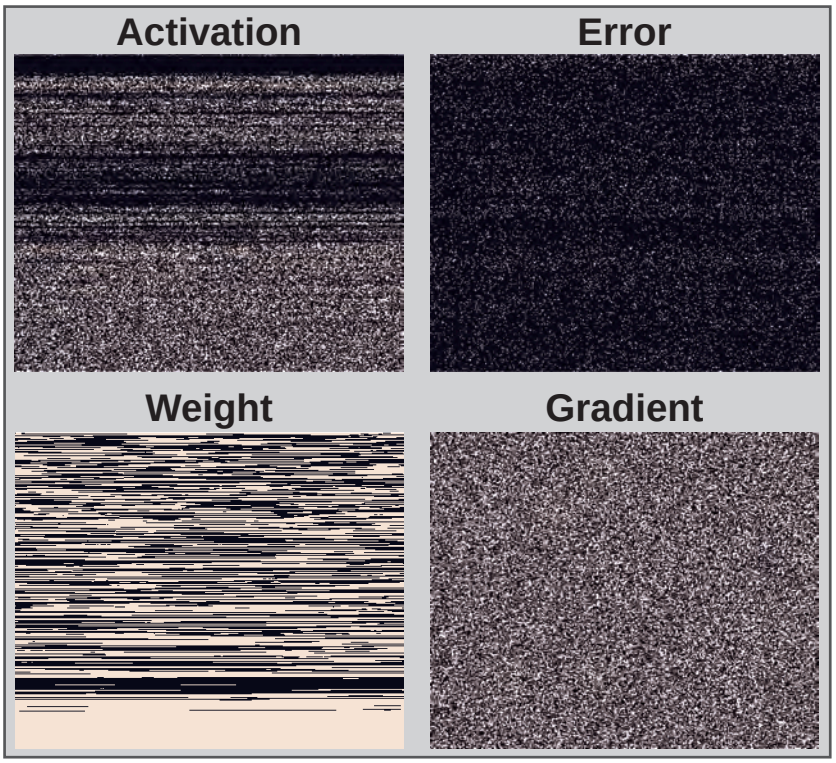

Fig. A1. Visualization of LSB's changing trend of a specific source location along the entire training process.

C Checking the Quality of Random Number using the NIST Test Suite

We use the NIST test suite SP800-22 to evaluate the quality of the random numbers extracted from different sources. Figure A2 shows the report of the NIST test when we are using the 2nd layer's gradients as the source for random number extraction throughout an entire training process. Our extracted random number can pass all the tests, indicating a very high random number quality.

Test Name	P-Value	Result
monobit_test	0.8512318716652107	PASS
frequency_within_block_test	0.6550591135121175	PASS
runs_test	0.6663240502419868	PASS
longest_run_ones_in_a_block_test	0.5667348815869525	PASS
binary_matrix_rank_test	0.6842085773521752	PASS
dft_test	0.7360320699994816	PASS
non_overlapping_template_matching_test	0.9999988863312715	PASS
overlapping_template_matching_test	0.6467261043438572	PASS
maurers_universal_test	0.1044507167463431	PASS
linear_complexity_test	0.98989840879283	PASS
serial_test	0.37045666806568434	PASS
approximate_entropy_test	0.7872286033166492	PASS
cumulative_sums_test	0.9058052629683142	PASS
random_excursion_test	0.19019732893198868	PASS
random_excursion_variant_test	0.20123449731469215	PASS

Fig. A2. The report of the test on the NIST test suite SP800–22. We use the 2nd layer's gradients as the source for random number extraction.

Figure A3 shows the testing results on the NIST test suite SP800–22 using extracted random numbers from different types and layers of sources. In the figure, we denote the source used for random number extraction in a $source_type-layer_num$ format(e.g., g-10 stands for the gradients in the 10th layer). We show the testing results of the first, middle, and last layers of different types of sources. From the figure we can see that even though extracting random numbers from weights can achieve comparable model accuracy as using gradient as the source, they still fail the most of the randomness tests. Both the relatively lower information entropy and the quite low bit-flipping frequency lead to this failure.

And we can also observe that, generally, the gradients have better randomness than the other sources. However, the last layer fails the majority of the tests, where the first layer fails some of them but is better than the last layer. The gradient of the middle layer can pass all the tests and is the best source for random number extraction. It is worth noting that besides the middle layer shown in the figure, we also find that the gradients from other layers (except the first and last layer) can generally pass all or the majority of the tests. This

indicates that there are a lot of good sources in the DNN available for high-quality random number extraction.

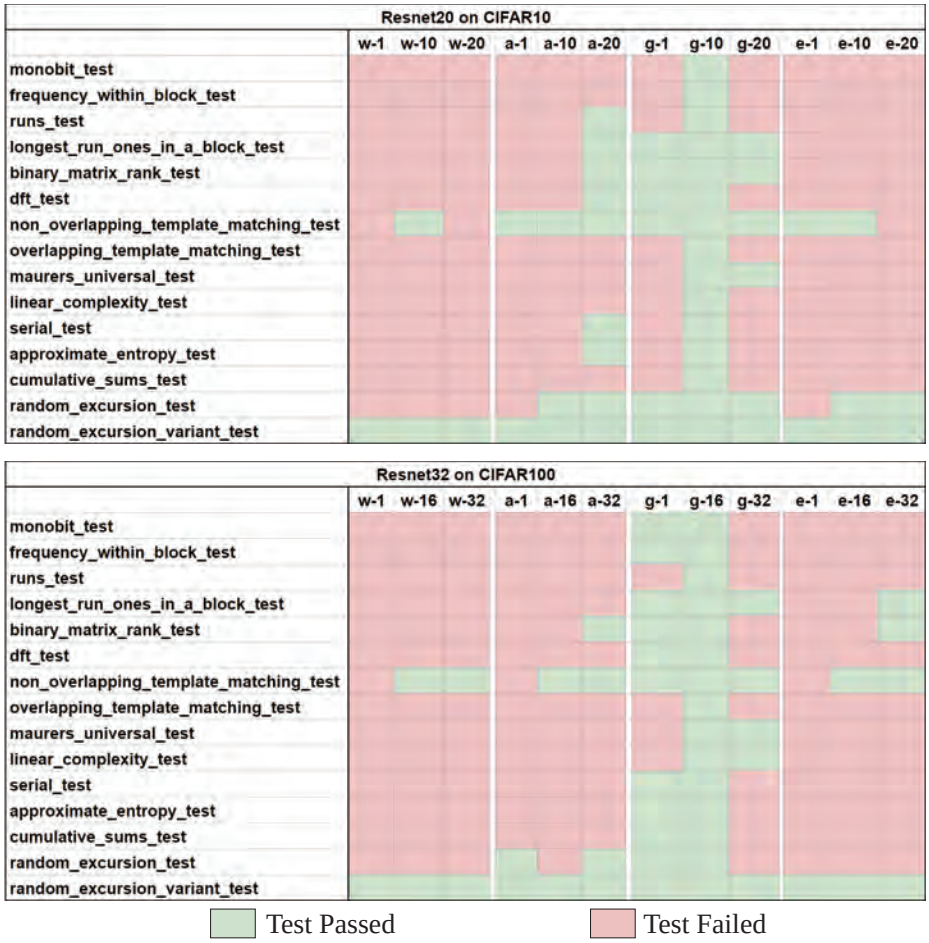

Fig. A3. Testing results on the NIST test suite SP800–22 using extracted random numbers from different types and layers of sources. We denote the source in a $source_type - layer_num$ format (e.g., g-10 stands for the gradients in the 10th layer). w: weight, a: activation, g: gradient, e: error.

D Random Numbers with Specified Distributions

We can also precisely control our image-pixel-based extraction method to obtain other specified distributions in high-quality. Figure A4 shows three different distributions obtained using the same input pixel location, which are (a) the uniform

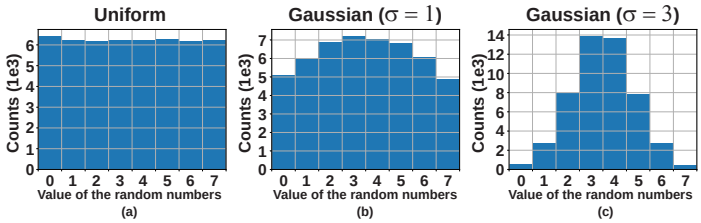

Fig. A4. Obtain high-quality specified distributions using our image-pixel-based random number extraction method.

distribution, (b) the Gaussian distribution with $\sigma = 1$, and (c) the Gaussian distribution with $\sigma = 3$. The distributions can be obtained by simply changing the corresponding threshold arrays. For a sanity check, we also randomly sampled several pixel locations to evaluate the quality of the distributions. High-quality results with desired distribution can always be obtained, which verifies the effectiveness of our proposed method.