Efficient Accuracy Recovery in Approximate Neural Networks by Systematic Error Modelling

Cecilia De la Parra
Robert Bosch GmbH
Renningen, Germany
cecilia.delaparra@de.bosch.com

Andre Guntoro
Robert Bosch GmbH
Renningen, Germany
andre.guntoro@de.bosch.com

Akash Kumar
Technische Universität Dresden
Dresden, Germany
akash.kumar@tu-dresden.de

ABSTRACT

Approximate Computing is a promising paradigm for mitigating the computational demands of Deep Neural Networks (DNNs), by leveraging DNN performance and area, throughput or power. The DNN accuracy, affected by such approximations, can be then effectively improved through retraining. In this paper, we present a novel methodology for modelling the approximation error introduced by approximate hardware in DNNs, which accelerates retraining and achieves negligible accuracy loss. To this end, we implement the behavioral simulation of several approximate multipliers and model the error generated by such approximations on pre-trained DNNs for image classification on CIFAR10 and ImageNet. Finally, we optimize the DNN parameters by applying our error model during DNN retraining, to recover the accuracy lost due to approximations. Experimental results demonstrate the efficiency of our proposed method for accelerated retraining (11× faster for CIFAR10 and 8× faster for ImageNet) for full DNN approximation, which allows us to deploy approximate multipliers with energy savings of up to 36% for 8-bit precision DNNs with an accuracy loss lower than 1%.

CCS CONCEPTS

• **Computing methodologies** → *Object recognition*; **Neural networks**; • **Hardware** → *Power estimation and optimization.*

KEYWORDS

approximate computing, approximation error model, deep neural networks, DNN optimization

ACM Reference Format:
Cecilia De la Parra, Andre Guntoro, and Akash Kumar. 2021. Efficient Accuracy Recovery in Approximate Neural Networks by Systematic Error Modelling. In *26th Asia and South Pacific Design Automation Conference (ASPDAC '21), January 18–21, 2021, Tokyo, Japan.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3394885.3431533

1 INTRODUCTION

Deep Learning (DL) architectures have large computational demands, which restrict their implementation in embedded systems. Approximations at software and hardware can reduce such computational demands to allow an embedded implementation [18, 25, 26]. However, the approximation error must be compensated to avoid accuracy losses and mantain the performance of the DL model. Motivated by this, in the present work we simulate, analyze and model the error introduced by approximate multipliers on DNNs, to identify critical elements affected by approximations. Our proposed approximation error model does not only help us to better understand the approximation error on DNNs, but is also useful for recovering the lost accuracy due to approximations, as we target more critical elements identified by our error model. To compensate the approximation error, DNN retraining is typically performed. Using the behavioral simulation of approximate multipliers during retraining is time consuming when compared with accurate DNN training [3, 18]. On the other side, the use of our proposed model has a very small time overhead and can be easily incorporated into the DNN computation using available functions from any open source library for machine learning, without the need of specialized operators for approximate multiplication. We demonstrate through various experiments that our mathematical error model allows fast and efficient approximate retraining, as the model is obtained offline before retraining e.g. from simulation on CPU, using only a small percentage of input data. Furthermore, in various cases, by training with our model we demonstrate better accuracy recovery, compared to using the behavioral simulation and to other state-of-the-art modelling approaches.

In summary, we make the following contributions:

- A thorough error analysis at the finest granularity, that is, at each neuron, in quantized and approximated DNNs.
- We propose a mathematical model of the error introduced by approximate multipliers in the DNN computation, to identify the most critical elements affected by the approximation.
- We validate our proposed error model for approximate DNN retraining to compensate the error introduced by approximate multipliers.

We perform extensive experiments with 10 different approximate multipliers and 4 different DNN architectures, quantized to 8 bits, for CIFAR-10 [11] and ImageNet [21]. Training approximate DNNs with our error model leads to a reduced retraining time of up to 11 times compared to using the behavioral simulation of approximate multipliers on a specialized, GPU-accelerated simulation framework. We also demonstrate negligible accuracy loss using multipliers with a Mean Relative Error (MRE) < 7% and energy savings of up to 36%, compared to the accurate 8-bit implementation.

2 RELATED WORK

In DNNs, approximations can be introduced at software and hardware level. Software approximation approaches in DNNs include filter pruning [10, 15] and precision scaling [4, 13]. Hardware-focused

ASPDAC '21, January 18–21, 2021, Tokyo, Japan

ACM ISBN 978-1-4503-7999-1/21/01…$15.00
https://doi.org/10.1145/3394885.3431533

approximation approaches include, among others, the approximation of adders and multipliers. We specifically target multipliers, as these are the most power-consuming operators. These approximations can be introduced in some neurons (partial approximation), or in all neurons of convolutional and fully-connected layers (full approximation). Authors in [14, 18, 25, 26] focus on partial approximation methods, with energy savings limited by the number of approximated neurons. In this work, we focus on full DNN approximation to increase energy savings, a similar approach to those introduced in [9, 16, 22]. In [22], quantized weights are combined with FP activations. Quantization of DNN weights and inputs to 8 and 12 bits respectively, in small DNN applications, is explored in [16]. A more resource-efficient DNN training methodology for small datasets such as MNIST [12] with approximate hardware is presented in [9]. Within this context, we optimize the incorporation of approximate multipliers with higher energy savings on DNN-based image recognition with more complex datasets such as CIFAR10 and ImageNet.

In the field of approximation error modelling in DNNs, a stochastic error model of fuzzy storage is introduced in [9], and authors in [14] simulate the influence of approximate elements as additive white Gaussian noise. In [6], the influence of approximate multipliers in DNNs is simulated by Gaussian and Uniform distributed noise characterized by the multiplier's MRE. Training with the error model of approximate multipliers was explored in [5], where a layer-wise model based on the multiplier's MRE and standard deviation was used. These proposed error models, however, lack verification by simulation of the modelled elements. We extend this state of the art by validating our proposed error model with the behavioral simulation of the corresponding approximate multipliers implemented for DNN inference.

3 APPROXIMATING DEEP NEURAL NETWORKS

DNN architectures comprise convolutional and fully-connected (FC) layers, which perform weighted multiplication between inputs X and weights W, followed by the addition of a bias b, and a non-linear activation function $\varphi(\cdot)$, such as a Rectified Linear Unit (ReLU).

We explore the incorporation of approximate multipliers in the DNN computation. For this, we first quantize DNN weights and activations from 32-bit floating-point (FP) values to 8 bits, as this delivers acceptable accuracy without retraining. The quantization is performed by (1), where $\widetilde{Z} = \text{round}(Z/\Delta_Z)$, bw is the target bitwidth and Δ_Z is the quantization step size.

$$Z_q = \begin{cases} -2^{bw-1} \times \Delta_Z & \text{if } \widetilde{Z} \leq -2^{bw-1} \\ \widetilde{Z} \times \Delta_Z & \text{if } -2^{bw-1} < \widetilde{Z} < 2^{bw-1} - 1 \\ (2^{bw-1} - 1) \times \Delta_Z & \text{if } \widetilde{Z} \geq 2^{bw-1} - 1 \end{cases} \quad (1)$$

3.1 SMApprox Multipliers

Focused on FPGA applications, we select eight 8x8 multipliers from the SMApprox library [24], which also includes Pareto-optimal designs from other sources such as EvoApprox [17].

To obtain power-efficient multipliers at Lookup Table (LUT) granularity, SMApprox multipliers take advantage of the eight 6-input LUTs and an 8-bit long carry chain available at each configurable logic block of latest versions of Xilinx FPGAs. To this end, four consecutive partial product rows are approximately mapped to 6-input LUTs using three different designs. Each design results in a 4x4 approximate multiplier which can be used as building block for larger multipliers: '0' represents the accurate design, and '1,2,3' indicate the corresponding approximate 4x4 designs. Characteristics of each 4x4 base multiplier design are presented in Table 1 [24].

Table 1: SMApprox 4×4b Base Approximate Multipliers

Metric	4x4 Design			
	0	**1**	**2**	**3**
LUTs	12	12	7	7
Latency [ns]	6.8	6.1	5.2	5.5
Power [mW]	198	180	192	192
MRE [%]	9	7.2	12.6	12.3

Table 2: SMApprox 8×8b Approximate Multipliers

Multiplier	**MRE[%]**	**Power(W)**	Area [LUTs]
1100	2.1	0.221	64
2200	3.8	0.220	62
3300	3.6	0.220	62
1110	4.0	0.220	64
3330	6.4	0.219	59
2220	6.7	0.219	59
3333	13.6	0.216	54
2222	16.6	0.217	52
Xilinx Multiplier IP	0	0.344	64

In this work, we select 8x8 multipliers built with the aforementioned base multipliers. Note that each 8x8 approximate multiplier requires four 4x4 base multipliers, and each can have a different design. Thus, for example, a multiplier denoted by '1100' represents a multiplier with two 4x4 multipliers of type 1 and two 4x4 multipliers of type 0. Note also that the addition of partial products in these 8x8 multipliers is kept accurate. The characteristics of the selected multipliers are shown in Table 2, where the MRE is formally computed by (2), where $n = 2^{bitwidth-1}$. The Xilinx Multiplier IP is the baseline in [24], and the highlighted multipliers belong to the Pareto-front of energy-area. The remaining multipliers were chosen due to their degree of approximation and small MRE.

$$MRE = \frac{1}{n \times n} \sum_{i=0}^{n} \sum_{j=0}^{n} \frac{\left|(i \times j)_{approx} - (i \times j)_{acc}\right|}{\max(1, |(i \times j)_{acc}|)} \quad (2)$$

4 METHODOLOGY DESIGN

In this section, we present our design approach for approximation error modelling. The analysis performed hereby is focused on approximate multipliers, however, this approach can be extended by using other approximate elements, as long as the approximation error introduced by such elements has parameterizable distribution and finite variance and covariance. These requisites are further explained in sub-section 4.2.

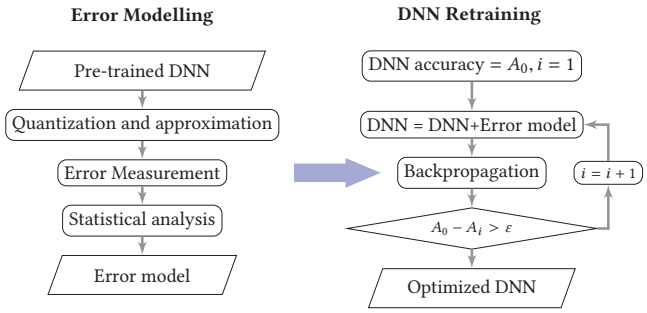


Figure 1: Modelling the Approximation Error in DNNs

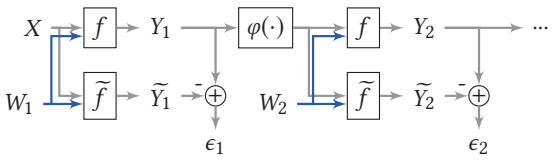


Figure 2: Approximation error measurement on a DNN

4.1 Design Approach

The error ϵ introduced by an approximate multiplier in a single multiplication is of deterministic nature. However, if we now consider the approximation error introduced in the computation of a convolutional (3) or FC layer (4), the dependencies of ϵ increase proportionally to the dimension and possible values of tensors X and W, as illustrated in equations (3) and (4) respectively.

$$\widetilde{Y}(i,j) = \sum_m \sum_n X_{m,n} W_{i-m,j-n} + \epsilon_{X_{m,n},W_{i-m,j-n}} + b_{i,j} \quad (3)$$

$$\widetilde{Y}(i,j) = \sum_n X_{i,n} W_{n,j} + \epsilon_{X_{i,n},W_{n,j}} + b_{i,j} \quad (4)$$

Such complexity induces a non-deterministic behavior in ϵ. We present in Fig. 1 our methodology to statistically model this error on pre-trained and approximated DNNs. As input, we receive a pre-trained DNN model with 32-bit FP parameters. This model is then quantized to 8 bits, and a given approximate multiplier is incorporated in the DNN computation. Then, we perform a statistical analysis of the error introduced by such approximation. As output we obtain an error model for each DNN neuron.

This methodology can either be applied to heterogeneous approximation (more than one type of approximate multiplier) or to homogeneous approximation, which is the focus of this work. Each step of the error modelling methodology is detailed in the following sub-sections.

For behavioral simulation of approximate multipliers in DNNs, we use ProxSim [3]. The behavioral simulation of different approximate multipliers is hereby implemented with CUDA [19] for GPU acceleration. As case study for modelling the approximation error, we implement ALL-CNN [23].

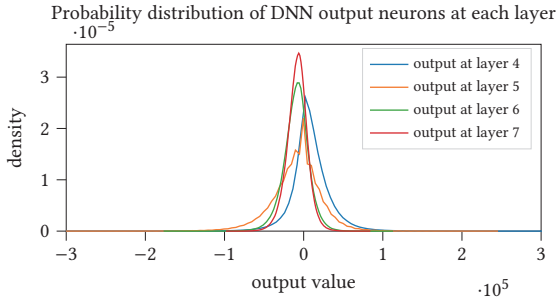


Figure 3: Output distributions on ALL-CNN layers.

4.2 Statistical Analysis of the Approximation Error

We measure the approximation error at the output feature map of each layer. As quantization operations have negligible computational overhead, these do not require to be modelled and can be instead accurately computed at each training step. Therefore, to obtain a highly accurate model, the error is measured after quantization of weights and activations, to avoid including the quantization noise in our approximation error model. The error measurement is performed as depicted in Fig. 2, where:

- f is the accurate operation between inputs and kernel, and $\widetilde{f}$ the approximate counterpart.
- Y_l and $\widetilde{Y}_l$ are the accurate and approximate output tensors of the l-th layer.
- ϵ_l is the tensor containing the approximation error at the l-th layer.

4.2.1 Correlation Analysis. For obtaining the correlation between outputs and appproximation error, we calculate the Pearson correlation coefficient $\rho_{y,\epsilon}$ between output y and approximation error ϵ **at each neuron**. For this, we assume that:

- The output of each neuron before any nonlinearity function $\varphi(\cdot)$ can be approximated by a Normal distribution, characterized by its mean μ and variance σ^2. This statement is corroborated by the work in [7, 20], and holds for our case study, as shown in Fig. 3.
- The approximation error has finite variance and covariance.

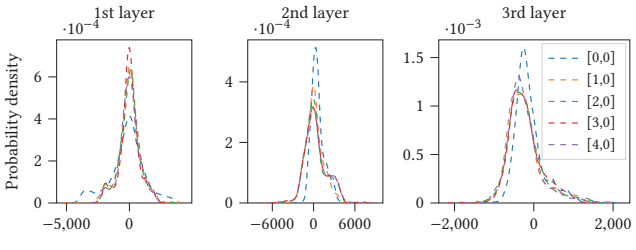
In a given DNN layer with a 3D output of size [H,W,Ch], the correlation $[\rho_{y,\epsilon}]_{i,j,k}$ between an output neuron $y_{i,j,k}$ and its corresponding approximation error $\epsilon_{i,j,k}$ is computed as follows:

$$[\rho_{y,\epsilon}]_{i,j,k} = \frac{\mathbf{Cov}(y_{i,j,k}, \epsilon_{i,j,k})}{\sigma_{y_{i,j,k}}, \sigma_{\epsilon_{i,j,k}}} \quad , \quad (5)$$

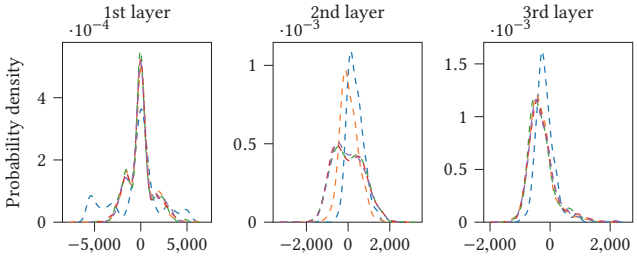
where $i \in [1, H]$, $j \in [1, W]$ and $k \in [1, Ch]$. In Table 3, the mean value of ρ at some layers of ALL-CNN is given for different multipliers. We observe a linear correlation between output and error at diferent levels of significance. Following this, we propose to build our neuron-wise error model based on correlated, normally distributed stochastic signals, characterized by $\rho_{y,\epsilon}$, the expected approximation error value μ_ϵ, and the standard deviation of the approximation error σ_ϵ.

Table 3: $\rho_{y,\epsilon}$ in ALL-CNN

Multiplier	CNN layer			
	1st	3rd	5th	7th
2222	0.35	0.69	0.78	0.72
3333	0.41	0.70	0.79	0.71
2220	0.44	0.68	0.83	0.74
3330	0.45	0.70	0.82	0.73



(a) SMApprox 3330



(b) SMApprox 2222

Figure 4: Error distributions on first neurons on 1st-3rd layers of ALL-CNN. Labels indicate the neuron position in the 1st channel of the output tensor.

4.2.2 Generating the Approximation Error Signal. For a given DNN layer, let us denote the output signal at neuron i, j, k as in (6), and the target correlated error signal $\epsilon_{i,j,k}$ as in (7), where:

- a_1, a_2 are coefficients to be solved.
- $s_{1|i,j,k}(t), s_{2|i,j,k}(t)$ are normally-distributed signals with variance $\sigma_1 = \sigma_2 = 1$ and mean $\mu_1 = \mu_2 = 0$, and uncorrelated ($\rho_{s_1,s_2} = 0$).

The signal $s_{1|i,j,k}(t)$ is derived from the output of the corresponding neuron i, j, k using p training samples, and the signal $s_{2|i,j,k}(t)$ is randomly drawn from its correspondent distribution.

$$y_{i,j,k}(t) = \mu_{y_{i,j}} + \sigma_{y_{i,j,k}} s_{1|i,j,k}(t) \tag{6}$$

$$\epsilon(t) = a_{1|i,j,k} s_{1|i,j,k}(t) + a_{2|i,j,k} s_{2|i,j,k}(t) + \mu_{\epsilon_{i,j,k}} \tag{7}$$

We denote the variance of $\epsilon_{i,j,k}$ as:

$$\sigma^2_{\epsilon_{i,j,k}} = E[\epsilon^2_{i,j,k}] - \mu^2_{\epsilon_{i,j,k}} \tag{8}$$

With this, we solve $a_{1|i,j,k}$ and $a_{2|i,j,k}$:

$$a_{1|i,j,k} = [\rho_{y,\epsilon}]_{i,j,k} \sigma_{\epsilon_{i,j,k}} \tag{9}$$

$$a_{2|i,j,k} = \sigma_{\epsilon_{i,j,k}} \sqrt{1 - \left|[\rho_{y,\epsilon}]_{i,j,k}\right|^2} \tag{10}$$

We then rewrite (7) as follows:

$$\epsilon_{\theta_{i,j,k}}(t) = \sigma_{\epsilon_{i,j,k}}([\rho_{y,\epsilon}]_{i,j,k} s_{1|i,j,k}(t) + s_{2|i,j,k}(t)\sqrt{1 - \left|[\rho_{y,\epsilon}]_{i,j,k}\right|^2}) + \mu_{\epsilon_{i,j,k}} \tag{11}$$

Our resulting model of the approximation error at a single neuron is obtained from (11). The error tensor $\Xi_\theta(t)$ at a given convolutional layer with H×W×Ch neurons is then computed as in (12), where:

- $\boldsymbol{\rho}_{y,\epsilon}$ is a tensor of shape [H,W,Ch] containing the correlation coefficient between output and error of each layer's neuron.
- $\boldsymbol{\sigma}_\epsilon$ and $\boldsymbol{\mu}_\epsilon$ are tensors of shape [H,W,Ch] containing the statistical moments of the approximation error at each neuron.
- $S_1(t)$ is a tensor derived from the output neurons of the corresponding layer.
- $S_2(t)$ is a tensor formed by H×W×Ch random signals drawn from the same distribution as $s_{2|i,j,k}$.

$$\Xi_\theta(t) = \boldsymbol{\sigma}_\epsilon(\boldsymbol{\rho}_{y,\epsilon} S_1(t) + S_2(t)\sqrt{1 - \left|\boldsymbol{\rho}_{y,\epsilon}\right|^2}) + \boldsymbol{\mu}_\epsilon \tag{12}$$

In case of computing the neuron-wise error model of FC layers, the error is obtained with (12) as well. However, $\boldsymbol{\sigma}_\epsilon, \boldsymbol{\rho}_{y,\epsilon}, \boldsymbol{\mu}_\epsilon$, and $\Xi_\theta(t)$ are matrices instead of 3D tensors. Values of $\boldsymbol{\rho}_{y,\epsilon}, \boldsymbol{\sigma}_\epsilon$ and $\boldsymbol{\mu}_\epsilon$ are computed offline using p training data samples, while signals $S_1(t)$ and $S_2(t)$ are drawn at runtime. The resulting error $\Xi_\theta(t)$ at a given layer is then added to the layer's output as depicted in Fig. 5. This is performed for all approximated layers.

4.3 DNN Retraining with approximations

The main goal of modelling the approximation error Ξ_θ, is to improve approximate DNN retraining, performed as in Fig. 1 Approximate DNNs are retrained using Stochastic Gradient Descent (SGD) methods. In this work, Straight-Through-Gradients [1] are implemented as in (13) to estimate the gradient of layers approximated with the behavioral simulation.

$$\Delta w = -\eta \frac{\partial C}{\partial w} \simeq -\eta \frac{\partial C}{\partial \widetilde{\mathbf{Y}}} \frac{\partial \mathbf{Y}}{\partial w} \tag{13}$$

For approximate DNN retraining applying our error model, we use the already existing gradient definitions for all operations that comprise Ξ_θ. The gradient is decomposed as follows:

$$\frac{\partial C}{\partial w} = \frac{\partial C}{\partial \widetilde{\mathbf{Y}}} \left[\frac{\partial \mathbf{Y}}{\partial w} + \frac{\partial \Xi_\theta}{\partial w} \right] \tag{14}$$

where:

$$\frac{\partial \Xi_\theta}{\partial w} = \boldsymbol{\sigma}_\epsilon \boldsymbol{\rho}_{y,\epsilon} \frac{\partial S_1(t)}{\partial w} \tag{15}$$

Thus, the influence of our error model in Δw is proportional to $\boldsymbol{\sigma}_\epsilon \boldsymbol{\rho}_{y,\epsilon}$. Typically, in a single neuron, $[\rho_{y,\epsilon}]_{i,j,k} \ll \sigma_{\epsilon_{i,j,k}}$ (see Fig. 4). Therefore, $\sigma_{\epsilon_{i,j,k}}$ indicates the contribution of the noise model to the gradient computation.

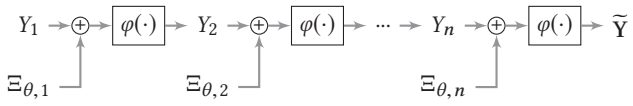


Figure 5: The obtained error model is added to each NN layer before the activation function.

Table 4: Evaluated DNNs

DNN	Dataset	FP acc.	8b acc.	#MAC ops.
ResNet8	CIFAR10	85.77	84.61	12M
ResNet14		89.53	89.20	27M
ALL-CNN		89.13	89.03	281M
ResNet18		65.88	65.15	1,826M

A small variance $\sigma_{\epsilon_{i,j,k}}$ can lead to an improvement in the DNN generalization because it provides some regularization [2], and therefore this delivers better results than training with the behavioral simulation. On the other hand, a large variance, correlated to a large MRE, can result in a sub-optimal convergence, which decreases the overall accuracy of the approximated DNN. We further analyze the effects of the approximation error variance in sub-section 5.2.

5 EVALUATION

We now present the evaluation results of our approximation error model applied for approximate DNN retraining. We compare the obtained results with:

- Behavioral simulation.
- The proposed error model from [5]: in each DNN layer, an auxiliary tensor *Err* is drawn from a Gaussian distribution characterized by the multiplier's MRE and standard deviation. The layer's output is computed by (16), where $\odot$ denotes element-wise multiplication. We subsequently refer to this model as *baseline model*.

$$Y = (X * (W \odot Err)) + b \tag{16}$$

- Additive white Gaussian noise added to the output of each layer. This noise is characterized by the Signal to Noise Ratio (SNR in dB), of the approximation error w.r.t. the accurate output, as proposed in [14].

All experiments were performed using an Nvidia GTX 1080 Ti and Tensorflow. We evaluate our approximation error model using two Residual Networks (ResNet) [8] for image classification with CIFAR-10, one simple convolutional arquitecture, the ALL-CNN [23], also for image classification with CIFAR-10, and ResNet18 for large-scale image classification with ImageNet. The characteristics of all evaluated DNNs are presented in Table 4.

With this variety of CNNs, we aim to demonstrate that our model works with simple sequential architectures (ALL-CNN) as well as with more complex ones (ResNets) for classification tasks of different complexity.

5.1 Model Runtime

We evaluate the acceleration of approximate DNN retraining with our error model, w.r.t. retraining using the behavioral simulation. This means, less training time and better accuracy in the same number of epochs, where epoch is understood as a complete forward and backward pass of the training dataset on a DNN. We retrain the DNN parameters by injecting our error model as in Fig. 5. By doing so, when compared to the behavioral simulation we achieve:

Table 5: DNN retraining - Execution time per epoch

DNN	ResNet8	ResNet14	ALL-CNN	ResNet18
Behavioral sim.	118s	198s	627s	17,101s
Error model	**51s**	**57s**	**59s**	**2085s**

Table 6: Overhead of modelling the DNN approximation error for a given approximate multiplier

DNN	ResNet8	ResNet14	ALL-CNN	ResNet18
Duration	8s	13s	15s	48s
W.r.t training time	16%	23%	25%	3%

Table 7: Cross-evaluation of approximation error model

retrained	Accuracies ALL-CNN[%]				Accuracies ResNet8[%]			
model	8b-quant	1110	3330	2222	FP	1110	3330	2222
8b-quant	**89.03**	88.64	88.41	85.71	**84.61**	82.68	80.42	54.41
1110	89.00	**88.88**	88.63	85.91	84.60	**85.48**	83.11	59.74
3330	88.66	88.76	**88.87**	85.99	83.96	84.48	**84.23**	65.31
2222	88.23	88.49	88.39	**86.58**	80.21	80.83	79.20	**67.96**

- A training up to 11× faster when retraining the evaluated CNNs with CIFAR-10 for accuracy recovery.
- Training time 8.2× faster when retraining ResNet18 for large-scale classification with ImageNet.

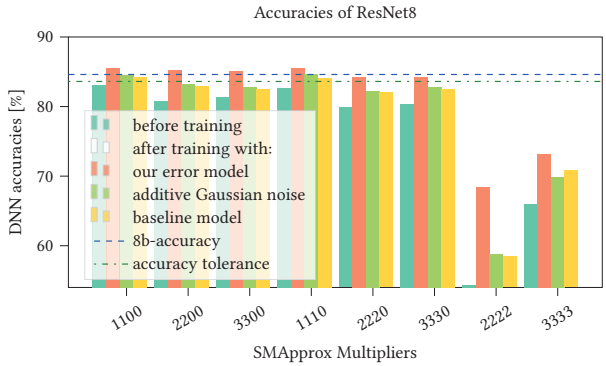
The obtained results are reported in Table 5, given a batch size of 64 images for all experiments. These results show that the improvement in execution time is proportional to the number of MAC ops. and DNN layers. Furthermore, in Table 6, we present the time required for modelling the approximation error of one given multiplier, which represents a small overhead compared to the total time required for DNN retraining.

5.2 Model Quality for DNN Retraining

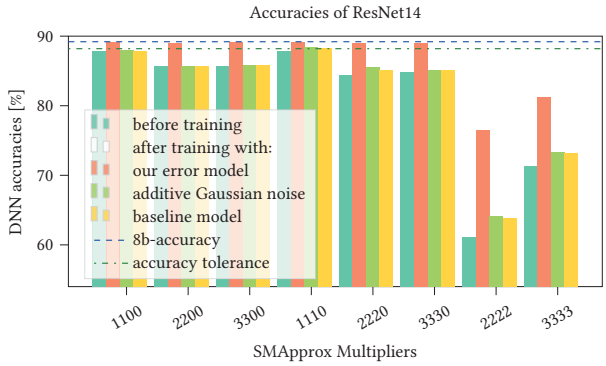
5.2.1 Optimization of DNNs for CIFAR10. Following recent works [18, 22], we propose an accuracy degradation limit ε of 1% with respect to the quantized 8-bit accuracy after retraining. We retrain the original DNNs quantized to 8 bits and approximated with each multiplier from Table 2. In this retraining stage, we apply the methods described at the beginning of this section.

We use SGD with momentum of 0.9, a learning rate (lr) of $1e-4$ and a batch size of 64 samples over 5 epochs. For a fair and accurate comparison, in all experiments the final DNN accuracy is computed with the behavioral simulation only. In Figs. 6a and 6b, we present the results of retraining with our proposed model, as well as with the baseline model and with additive Gaussian noise. We perform this comparison with ResNet8 and ResNet14. Our approximation error model delivers the best results, and we are able to reach our proposed tolerance and even surpass the original 8-bit accuracy in 6 out of 8 cases.

In Figs. 7a, 7b and 7c, we present the comparison of training with our proposed approximation error model and training with the behavioral simulation, using ResNet8, ResNet14 and ALL-CNN respectively. We also use $\varepsilon = 1\%$. For multipliers with MRE $< 7\%$, we reach slightly better accuracy by using our error model, compared



(a) ResNet8-CIFAR10



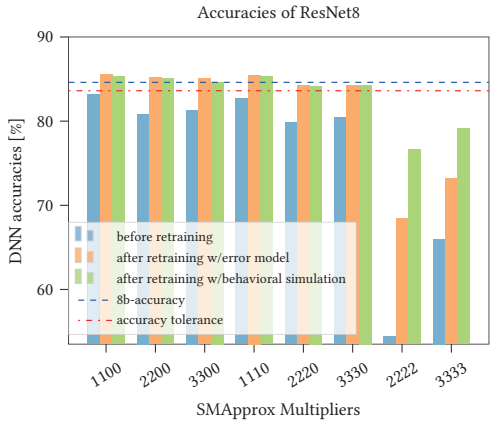
(b) ResNet14-CIFAR10

Figure 6: Comparison of different approximation error models for DNN retraining.

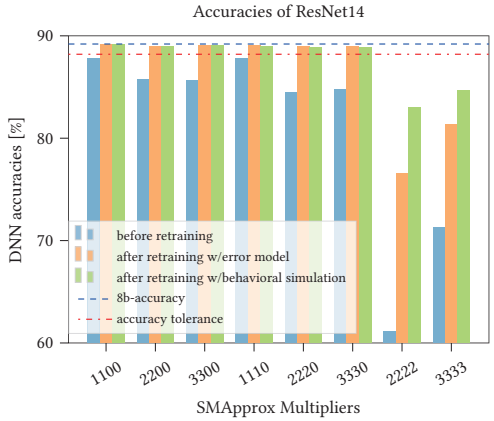
to retraining with the behavioral simulation (from 0.06% to 0.41% improvement in ResNet8 and from 0.02% to 0.14% in ResNet14). This thanks to the regularization effect of the model (see sub-section 4.3), as the error variance is proportional to the MRE. We also surpass ε in all evaluated cases. For multipliers with MRE > 7%, we also achieve accuracy improvement, but ε is never reached. The behavioral simulation delivers better accuracy than our error model because of the larger noise variance introduced by it during retraining.

5.2.2 Optimization of ResNet18 for ImageNet. We retrain the quantized and approximated ResNet18 using each multiplier from Table 2 with an MRE <7%, as multipliers with larger MRE introduce very large accuracy degradations in this network. We use SGD with a lr of $1e-4$, and a batch size of 64. ImageNet is substantially larger than CIFAR10, therefore we retrain only with 20% of the training data for 2 epochs. As reported in Fig. 8, we are able to surpass the accuracy degradation limit $\varepsilon = 1\%$ in 4 out of 6 evaluated cases. Note that we use the full validation dataset for our evaluations.

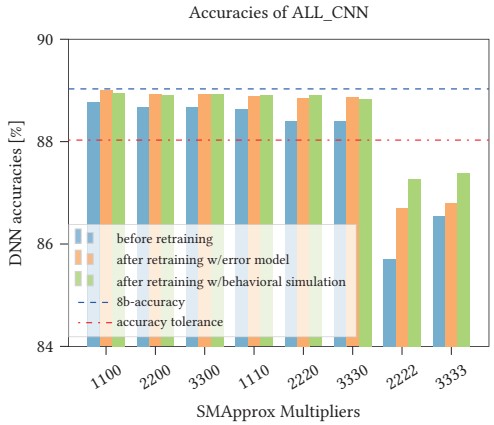
The presented results show that our error model allows faster DNN retraining in case of implementing approximate multipliers with small and moderate MRE. Furthermore, our model achieves better accuracy in these cases because the controlled and targeted additive noise from our model acts as an additional regularizer to



(a) ResNet8-CIFAR10



(b) ResNet14-CIFAR10



(c) ALL-CNN-CIFAR10

Figure 7: DNN accuracies after retraining with behavioral simulation vs. retraining with error model - CIFAR10.

the original cost function [2]. Given the obtained accuracies after retraining, by using our methodology we can deploy multipliers such as SMApprox 3330, with energy savings of up to 36% with a strict accuracy tolerance of 1%.

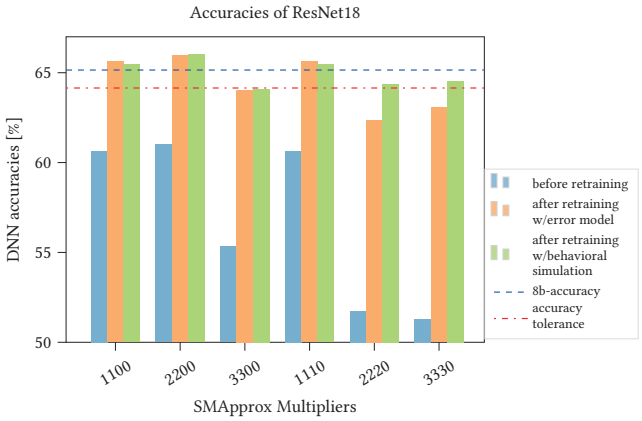


Figure 8: ResNet18-ImageNet

Figure 9: Accuracies of retrained DNNs with behavioral simulation vs. error model - ImageNet.

5.2.3 Characteristic approximation error noise. To corroborate that our error model differs from uncorrelated additive Gaussian noise, we use our retrained ALL-CNN and ResNet8 with the error models of SMApprox 2222, 1110 and 3330 to perform cross-evaluation. For this, we use the corresponding retrained DNN with the first multiplier to perform inference using the second and third multiplier and vice versa. Our results, presented in Table 7, show that our error model trains the DNN specifically for the error introduced by the corresponding multiplier and therefore results in significant improvement only for that approximate element, and causes accuracy degradation otherwise.

6 SUMMARY AND CONCLUSIONS

In this work, we present a novel methodology to model the effects of approximate multipliers in DNNs. Our proposed approximation error model is verified by simulation on several DNNs for medium and large-scale image classification. This error model allows for approximate DNN computation and retraining using already available operators from any machine learning framework, as it can be obtained offline from any edge device or other simulation source. We implement our error model to accelerate approximate DNN retraining, achieving up to to 11× faster retraining for DNNs for image classification with CIFAR10 and 8.2× faster for image classification with ImageNet, compared to retraining with the behavioral simulation. Moreover, we are able to reach an accuracy loss of less than 1%, outperforming other state-of-the-art error models. Through this retraining step, we attain full DNN approximation for medium and large-scale image recognition. With the obtained results, not only an extended insight in the error introduced by approximate multipliers was gained, but this work shall further allow significant energy savings by deploying such approximate arithmetic circuits on dedicated hardware accelerators for more complex DNNs.

ACKNOWLEDGMENTS

This work was funded by the ECSEL Joint Undertaking project TEMPO, in collaboration with the European Union's Horizon 2020 Research and Innovation Program and National Authorities, under grant agreement No. 826655.

REFERENCES

[1] Martín Abadi, Ashish Agarwal, Paul Barham, et al. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. https://www.tensorflow.org/
[2] Chris M. Bishop. 1995. Training with Noise is Equivalent to Tikhonov Regularization. *Neural Comput.* (1995).
[3] C. De la Parra, A. Guntoro, and A. Kumar. 2020. ProxSim: GPU-based Simulation Framework for Cross-Layer Approximate DNN Optimization. In *DATE '20*.
[4] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. 2015. Deep Learning with Limited Numerical Precision. *ICML '15* (2015).
[5] I. Hammad et al. 2019. Deep Learning Training with Simulated Approximate Multipliers. In *ROBIO*.
[6] Issam Hammad and Kamal El-Sankary. 2018. Impact of Approximate Multipliers on VGG Deep Learning Network. *IEEE Access* (2018).
[7] Song Han et al. 2016. Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding. In *ICLR '16*.
[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep Residual Learning for Image Recognition. *CVPR '15* (2015).
[9] Xin He et al. 2018. AxTrain: Hardware-Oriented Neural Network Training for Approximate Inference. *ISLPED '18* (2018).
[10] Yihui He, Xiangyu Zhang, and Jian Sun. 2017. Channel Pruning for Accelerating Very Deep Neural Networks. *ICCV '17* (2017).
[11] Alex Krizhevsky. 2009. Learning Multiple Layers of Features from Tiny Images. *University of Toronto* (2009).
[12] Yann LeCun and Corinna Cortes. 2010. MNIST handwritten digit database. (2010). http://yann.lecun.com/exdb/mnist/
[13] Darryl Dexu Lin, Sachin S. Talathi, and V. Sreekanth Annapureddy. 2016. Fixed Point Quantization of Deep Convolutional Networks. *ICML '16* (2016).
[14] M.A.Hanif et al. 2018. Error resilience analysis for systematically employing approximate computing in convolutional neural networks. *DATE '18* (2018).
[15] Alberto Marchisio, Muhammad Hanif, Maurizio Martina, and Muhammad Shafique. [n.d.].
[16] Vojtech Mrazek et al. 2016. Design of Power-efficient Approximate Multipliers for Approximate Artificial Neural Networks. In *ICCAD '16*.
[17] Vojtěch Mrázek, Radek Hrbáček, Zdeněk Vašíček, and Lukáš Sekanina. 2017. EvoApprox8B: Library of Approximate Adders and Multipliers for Circuit Design and Benchmarking of Approximation Methods. In *DATE '17*.
[18] Vojtech Mrazek, Zdenek Vasícek, Lukás Sekanina, Muhammad Abdullah Hanif, and Muhammad Shafique. 2019. ALWANN: Automatic Layer-Wise Approximation of Deep Neural Network Accelerators without Retraining. *ICCAD '19* (2019).
[19] John Nickolls, Ian Buck, Michael Garland, and Kevin Skadron. 2008. Scalable Parallel Programming with CUDA. *Queue* (2008).
[20] E. Park, J. Ahn, and S. Yoo. 2017. Weighted-Entropy-Based Quantization for Deep Neural Networks. In *CVPR '17*.
[21] Olga Russakovsky et al. 2015. ImageNet Large Scale Visual Recognition Challenge. *IJCV '15* (2015).
[22] Syed Shakib Sarwar et al. 2018. Energy-Efficient Neural Computing with Approximate Multipliers. *J. Emerg. Technol. Comput. Syst.* (2018).
[23] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin A. Riedmiller. 2014. Striving for Simplicity: The All Convolutional Net. (2014).
[24] Salim Ullah, Sanjeev Sripadraj Murthy, and Akash Kumar. 2018. SMApproxLib: Library of FPGA-based Approximate Multipliers. In *DAC '18*.
[25] Swagath Venkataramani et al. 2014. AxNN: Energy-efficient neuromorphic systems using approximate computing. *ISLPED '14* (2014).
[26] Qian Zhang et al. 2015. ApproxANN: An Approximate Computing Framework for Artificial Neural Network. In *DATE '15*.