Knowledge Distillation and Gradient Estimation for Active Error Compensation in Approximate Neural Networks

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

Xuyi Wu
Technische Universität München
Munich, Germany
xuyi.wu@tum.de

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 approach for optimizing computational resources of error-resilient applications such as Convolutional Neural Networks (CNNs). However, such approximations introduce an error that needs to be compensated by optimization methods, which typically include a retraining or fine-tuning stage. To efficiently recover from the introduced error, this fine-tuning process needs to be adapted to take CNN approximations into consideration. In this work, we present a novel methodology for fine-tuning approximate CNNs with ultra-low bit-width quantization and large approximation error, which combines knowledge distillation and gradient estimation to recover the lost accuracy due to approximations. With our proposed methodology, we demonstrate energy savings of up to 38% in complex approximate CNNs with weights quantized to 4 bits and 8-bit activations, with less than 3% accuracy loss w.r.t. the full precision model.**

***Index Terms*—Approximate Computing, Neural Networks, Quantization, Approximate multipliers.**

I. Introduction

CNNs play an important role in the field of machine learning, achieving near-human accuracy in tasks such as image classification. However, the implementation of CNN models in edge devices is highly challenging because of large memory requirements, area and power consumption. One paradigm to reduce such requirements is *cross-layer approximate computing*, which comprises approximations at algorithmic and hardware level. In this work, we focus on applying approximate computing to pre-trained CNNs, which facilitates the use of general retraining or fine-tuning schemes that deliver reproducible results, and that can be applied in different application domains. We explore two approximation approaches applied to pre-trained CNNs: Low bit-width quantization at software level, and the use of approximate multipliers at hardware level.

Quantization of CNN parameters to 8 bits has been widely researched. It delivers the same accuracy as the Full Precision (FP) model without additional retraining [1], [2]. However, the use of lower bit-widths can achieve further reduction in memory and power consumption. Therefore, we quantize CNN weights to 4 bits, while maintaining 8-bit activations. We subsequently refer to this configuration as 8A4W quantization.

After applying 8A4W quantization, we incorporate Approximate Multipliers (AMs) in the computation of convolutional and Fully Connected (FC) layers. The combination of ultra-low bitwidth quantization and AMs makes it more challenging to compensate for the approximation error, as we have less degrees of freedom to "move" to regions where the approximation error is smaller. Furthermore, we are interested in using multipliers with large Mean Relative Error (MRE), which have higher energy savings, but which cause large accuracy degradation. To recover from these approximation errors, a fine-tuning stage is required. This stage consists in applying Stochastic Gradient Descent (SGD) optimization to the approximated CNN for a given number of training epochs. During fine-tuning, the SGD method has to be adapted to take the CNN approximation into account, and thus to perform a better compensation of the approximation error. To this end, in this work we propose a fine-tuning methodology which involves the combination of Knowledge Distillation [3] for approximate CNNs, and the gradient estimation of approximate General Matrix Multiplications (GEMMs). We compare our proposals with two state-of-the-art methodologies: passive retraining [4] and alpha-regularization [5]. In summary, our contributions are:

- Novel two-stage Knowledge Distillation for optimized cross-layer CNN Approximation (ApproxKD).
- Novel gradient estimation of approximate GEMMs for error compensation in approximate CNNs by means of SGD algorithms.
- A methodology combining ApproxKD and gradient estimation for effective error recovery in drastically approximated CNNs.

We perform an extensive analysis of our proposed methodology. For this, we use complex CNNs such as ResNet20, ResNet32 [6] and MobileNetV2 [7] for image classification with CIFAR10 [8]. After model compression through 8A4W quantization, we are able to reach further energy savings of up to 38% with an accuracy loss of 2.37% and 2.36% respectively, when applying approximate multipliers to the evaluated ResNets. We also reach energy savings of 28% with an accuracy loss of 1.76% with the more complex MobileNetV2. To the best of our knowledge, we are the first to sucessfully approximate such highly complex CNNs after quantizing weights to less than 8 bits.

978-3-9819263-5-4/DATE21/©2021 EDAA

II. Related Work

Cross-layer approximate computing of CNNs comprises a variety of methods for optimizing the trade-off between quality of application and computational requirements. Regarding quantization of pre-trained CNNs, it has been proved by [1], [2] that 8-bit quantization without accuracy loss is possible, and it does not require fine-tuning. However, when using smaller bitwidths, the accuracy degradation is larger and therefore fine-tuning is required. Knowledge Distillation (KD) was proposed in [3] for distilling the knowledge of an ensemble of CNN models into a single model. Previous works about applying KD for CNN quantization, without considering additional hardware requirements, have been introduced in [10], [11].

Regarding the use of AMs in CNNs, these can either be used to compute some neurons (partial approximation) or all neurons in convolutional and FC layers (full approximation). Partial, resiliency-based CNN approximation was introduced in [12]. Other examples of partial approximation based on resiliency analysis are [13], [14]. In [15], a framework for layer-wise CNN approximation based on genetic programming was proposed. While partial approximation delivers acceptable trade-offs between accuracy and energy savings, these are bounded by the amount of approximated neurons. Full approximation, on the other hand, can deliver better energy savings, but usually results in larger accuracy degradation. Thus, CNN weights must be optimized to recover from this accuracy loss. Approximate CNN fine-tuning is an efficient optimization method for accuracy recovery in fully approximated CNNs, as demonstrated in [4], [5], [16]. In [4], *passive* and *active* retraining of approximate CNNs were introduced for accuracy recovery and for improving CNN robustness respectively. In [5], a regularization method for approximate CNN retraining, known as *alpha regularization*, was presented. All these works deal with CNNs quantized to a minimum of 8 bits. In this work, we extend the state of the art with our proposed fine-tuning strategies which allow more drastic quantization and approximation of more complex CNNs, such as ResNet [6] and MobileNetV2 [7].

III. Proposed Methodology

We propose an optimization flow to achieve low bitwidth quantization and approximation of multiplications, without accuracy loss. This flow is presented in Algorithm 1 and it consists of two stages. First, the CNN weights and activations are quantized to 4 and 8 bits respectively, and the weights are then updated through fine-tuning and KD to recover the lost accuracy after quantization. Our quantized model has the following characteristics:

- Layer-wise quantization of parameters and activations.
- No zero-points. We use a symmetric linear quantizer, which can be less precise, but which eliminates cross-terms resulting from GEMM involving zero-points [17].
- The quantization step sizes needed for linear quantization are computed using Minimization of the Propagated Quantization Error (MinPropQE) [1].
- Quantization step sizes are rounded to the next power-of-two, to quantize values with a simple shifting operation.

After quantization, we apply the corresponding approximations to our CNN model and perform a second fine-tuning stage, where we apply again KD and combine it with gradient estimation. The details about this two-stage KD for approximate CNNs and its combination with gradient estimation are described in the following sub-sections.

A. ApproxKD: Two-stage Knowledge Distillation

CNNs for image processing tasks, such as classification and semantic segmentation, generally produce class probabilities at the output layer. During CNN fine-tuning, the goal is then to minimize the cross entropy between CNN outputs and labels. This cross-entropy loss function is defined by:

$$C(y) = -\sum_{k=1}^{n} p_k \log \sigma(y)_k \quad , \tag{1}$$

where n is the number of predicted classes, $\sigma(\cdot)$ is the softmax function applied to the CNN output, p is the *hard label* or expected probability value for each class, and y is the CNN prediction or output vector. For computing the gradient of non-differentiable functions in the quantization stage, such as *round*, a Straight-Through-Estimator (STE) [18] is used.

In previous works [10], [11], KD has been applied in a single stage to optimizing quantized CNNs. However, when introducing additional approximation errors, such as those produced by approximate multipliers, a single KD stage is not enough to distill knowledge from a Full-Precision (FP) CNN modeL to an approximated model directly. This because the quantiztion and approximation errors accumulate and thus are more difficult to compensate. Therefore, we propose a two-stage KD methodology for distilling information from a FP into a quantized model and then into an approximated model. This two-stage distillation scheme is further refered to as *ApproxKD*. In ApproxKD, the FP model is used as a *teacher model*, and its information is distilled at different *temperatures* to train the *student models*. Our first student model is the CNN model quantized to 8A4W, and our second student model is the approximated CNN using AMs. These student models are optimized in two sequential stages, subject to our imposed quantization constraints mentioned at the beginning of section III. These two stages are what we refer to as the *Quantization* and *Approximation* stages.

1) Quantization stage: In this stage, the optimization of the quantized model is performed. The final cost function $C_{s1}(y_q)$ consists of adding a hard loss $C_{hard}(y)$ and a soft loss $C_{soft}(y_q)$. The function $C_{hard}(y_q)$, computed by (1), is the traditional cross-entropy loss between the output y_q of the quantized model and the hard labels p, provided in the training dataset. The cost function $C_{soft}(y_q)$ is computed using the outputs from the teacher model y as soft labels, as in (2), where a *distillation temperature* T_1 is applied to adjust the probability distributions of the teacher and student model. The magnitudes of the gradients back-propagated by the soft loss $C_{soft}(y_q)$ are scaled by T_1^{-2}, and therefore $C_{soft}(y_q)$ is multiplied with T_1^2.

$$C_{soft}(y_q) = -T_1^2 \sum_{k=1}^{n} \sigma(y/T_1)_k \log \sigma(y_q/T_1)_k \tag{2}$$

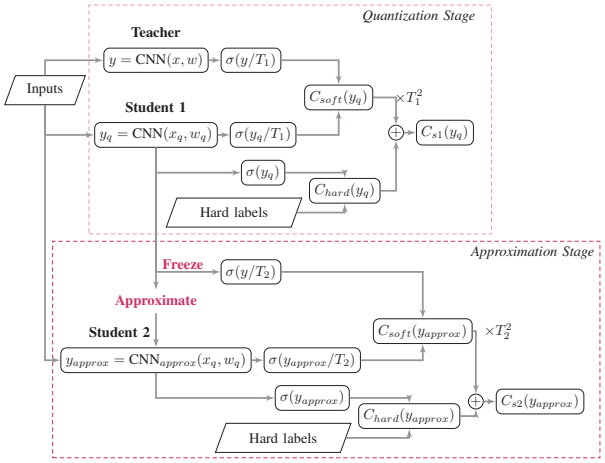

Fig. 1: Definition of cost functions using Knowledge Distillation for optimizing approximate CNNs.

2) Approximation Stage: After the quantization stage, we optimize the approximated version of the quantized CNN. A hard loss $C_{hard}(y_{approx})$ and a soft loss $C_{soft}(y_{approx})$ for the approximated model are defined as in Fig. 1. A second distillation temperature T_2 is applied in this stage. When introducing approximation errors in a CNN and $y_{approx} < y_q$, the output y_{approx} is significantly different from the output y_q of the quantized model. Therefore, the approximated model benefits from a smoother distribution of the soft labels, which is achieved by the condition $T_2 > T_1$. The total loss function is then defined as follows:

$$C_{s2}(y_{approx}) = -T_2^2 \sum_{k=1}^{n} \sigma(y_q/T_2)_k \log \sigma(y_{approx}/T_2) - \sum_{k=1}^{n} p_k \log \sigma(y_{approx})_k \quad (3)$$

B. Gradient estimation of approximate GEMMs

Convolutional and FC layers can be computed using GEMM, as in [5]. After transforming the input feature X and the weights W for GEMM, the output of an approximate layer is computed as follows:

$$\widetilde{y}_{i,j} = \sum_{k}^{n} \widetilde{g}(X_{ik}, W_{kj}) \quad , \quad (4)$$

where $\widetilde{g}(X_{ik}, W_{kj})$ is the approximate multiplication of X_{ik}, W_{kj}. During fine-tuning, the backward pass includes STEs for computing the derivative of functions with undefined gradients, as proposed in [4]. In the case of approximate GEMMs, their gradient is computed using that of the accurate GEMM. If we consider the gradient of the cost function $C(\widetilde{y})$ w.r.t. weights W, the STE of approximate GEMMs is computed as follows:

$$\frac{\partial C(\widetilde{y})}{\partial W} \rightarrow \frac{\partial C(\widetilde{y})}{\partial \widetilde{y}} \frac{\partial y}{\partial W} = \frac{\partial C(\widetilde{y})}{\partial \widetilde{y}} X^T \quad , \quad (5)$$

where $\widetilde{y}$ is the output of approximate GEMMs, y is the output of the accurate GEMM, X^T is the transposed input matrix, and W is the weight matrix.

Since the STE of approximate GEMMs introduces noise in the backpropagation during fine-tuning, we propose to estimate the gradient of approximate GEMMs more precisely, by taking the approximation error into account. For an element W_{dc} in matrix W, the gradient of $C(y)$ is defined as in (6), where $y_{ij} = \sum_k W_{ik} X_{kj}$, and all terms with the index $i \neq d$ are cancelled, according to (7).

$$\frac{\partial C}{\partial W_{dc}} = \sum_i \sum_j \frac{\partial C}{\partial y_{ij}} \frac{\partial y_{ij}}{\partial W_{dc}} \quad (6)$$

$$\frac{\partial y_{ij}}{\partial W_{dc}} = \begin{cases} \frac{\partial y_{dj}}{\partial W_{dc}} & \text{if } i = d \\ 0 & \text{otherwise} \end{cases} \quad (7)$$

The element-wise gradient of an approximate GEMM is then defined as follows:

$$\frac{\partial C}{\partial W_{dc}} = \sum_i \sum_j \frac{\partial C}{\partial \widetilde{y}_{ij}} \frac{\partial \widetilde{y}_{ij}}{\partial W_{dc}} \quad , \quad (8)$$

where $\widetilde{y}_{ij} = y_{ij} + \epsilon_{ij}$. The approximation error ϵ_{ij} is the difference between y_{ij} and $\widetilde{y}_{ij}$, given the same inputs X and weights W. We propose to estimate ϵ_{ij} as a function f dependent on y. In this way, the gradient of $C(\widetilde{y})$ can be estimated more accurately:

$$\frac{\partial C}{\partial W_{dc}} = \sum_j \frac{\partial C}{\partial y_{dj}} \left(\frac{\partial y_{dj}}{\partial W_{dc}} + \frac{\partial \epsilon}{\partial W_{dc}} \right) \quad (9)$$

$$= \sum_j \frac{\partial C}{\partial y_{dj}} \frac{\partial y_{dj}}{\partial W_{dc}} \left(1 + \frac{\partial}{\partial y_{dj}} f(y_{dj}) \right) \quad (10)$$

To avoid large computational overhead required to compute $\frac{\partial}{\partial y_{dj}} f(y_{dj})$, the approximation error ϵ is estimated in this work as a piecewise linear function. For example, in the case of the approximation error depited in Fig. 2, the error is estimated as follows:

$$\epsilon_{i,j} \approx f(y_{i,j}) = \min(a, \max(\widetilde{k} y_{i,j} + c, b)) \quad (11)$$

Thus, $\frac{\partial C}{\partial W}$ can be defined by:

$$\frac{\partial C}{\partial W} = (1 + K) \frac{\partial C}{\partial \widetilde{y}} X^T \quad , \quad (12)$$

where K is a matrix defined by the derivative of the piecewise function (11):

$$k_{i,j} = \begin{cases} \widetilde{k} & \text{if } a < \epsilon_{i,j} < b \\ 0 & \text{otherwise} \end{cases} \quad (13)$$

C. ApproxKD + Gradient Estimation

We now combine KD and the Gradient Estimation (GE) of approximate GEMMs to minimize the approximation error in a fully quantized and approximated CNN, as presented in Algorithm 1. Note that if the derivative of the error funtion $f(y_q)$ is non-zero, GE is applied each fine-tuning iteration of the approximation stage. Otherwise, if $\frac{\partial f(y_q)}{\partial y} = 0$, this is equivalent to using STE during back-propagation.

Algorithm 1: ApproxKD + Gradient Estimation

Data: FP CNN with weights w and inputs X, fine-tuning epochs e_1, e_2, distillation temp. T_1, T_2, approximation error function $f(y_q)$
Result: Optimized Approximate CNN
Quantization stage
for e_1 *epochs* **do**
  for *each training minibatch* **do**
    Quantize $x, w \rightarrow x_q, w_q$;
    Compute $y_q = CNN(x_q, w_q)$;
    Minimize $C_{s1}(y_q)$ with temperature T_1 ;
    Update w by computing $\frac{\partial C(y_q)}{\partial w}$;
  end
end
Approximation stage
Approximate $CNN \rightarrow CNN_{approx}$;
for e_2 *epochs* **do**
  for *each training minibatch* **do**
    Quantize $x, w \rightarrow x_q, w_q$;
    Compute $y_{approx} = CNN_{approx}(x_q, w_q)$;
    Minimize $C_{s2}(y_{approx})$ with temperature T_2;
    if $\frac{\partial f(y_q)}{\partial y} \neq 0$ **then**
      Compute $\frac{\partial C(y_{approx})}{\partial w}$ using (10) with $f(y_q)$;
    else
      Compute $\frac{\partial C(y_{approx})}{\partial w}$ with STE ;
    end
    Update w with $\frac{\partial C(y_{approx})}{\partial w}$;
  end
end

TABLE I: Evaluated CNNs

CNN	#Params($\times 10^6$)	#MAC Ops($\times 10^9$)	FP Acc.[%]
ResNet20	0.3	0.041	91.04
ResNet32	0.5	0.069	91.88
MobileNetV2	2.2	0.296	94.89

IV. Experimental Results

The CNNs used for evaluation are presented in Table I. All experiments were performed using a GPU Nvidia GTX 1080 Ti and Tensorflow [19]. We evaluate the performance of 8A4W quantized CNNs computed with approximate multipliers. We perform our experiments with ResNet20, ResNet32 and MobileNetV2 trained with CIFAR10. For improved computational efficiency, we fold all Batch Normalization (BN) layers in the evaluated ResNets [9]. On the other hand, BN layers are kept in MobileNetV2 to avoid a large accuracy drop. To limit the number of experiments, we approximate using only one multiplier type to compute all neurons (uniform approximation). All experiments with approximate CNNs are performed in the simulation framework ProxSim [5]. The accuracies of the 8A4W quantized CNNs before and after Fine-Tuning (FT) using KD (quantization stage) with $T_1 = 1$ are reported in Table II.

For the experiments with approximate CNNs, we use multipliers from the EvoApprox library [20], adapted for 8×4 bit multiplication, as well as 8×4 truncated multipliers [21], without bias correction. All relevant multipliers used in our experiments, together with their MRE and estimated energy savings from [20], [21] are reported in Table V. Note that the number assigned to the truncated multipliers refers to the number of Least Significant Bits (LSB) truncated from the multiplication product. The multipliers from the EvoApprox library are selected from the Pareto front of CNN accuracy and energy savings [20]. The provided MRE is formally computed for all possible values by (14), where N_X and N_W are the bitwidths of activations X and weights W, $g(\cdot)$ represents the accurate multiplication function and $\widetilde{g}(\cdot)$ denotes the approximate multiplication. In this work, the reported energy savings in multiplications are estimated using those of a single multiplier.

TABLE II: 8A4W Quantization - Results

CNN	Acc. before FT[%]	Acc. after normal FT[%]	Acc. after FT w/KD[%]
ResNet20	82.88	90.51	**90.60**
ResNet32	83.66	91.23	**91.29**
MobileNetV2	10.01	93.70	**93.81**

$$MRE = \frac{1}{2^{N_X} * 2^{N_W}} \sum_{j=0}^{2^{N_X}-1} \sum_{k=0}^{2^{N_W}-1} \frac{|g(j,k) - \widetilde{g}(j,k)|}{\max(g(j,k), 1)} \quad (14)$$

A. Ablation study - ApproxKD

We evaluate the influence of the distillation temperature T_2 in ApproxKD for approximate CNNs, using the hereby proposed approximate multipliers, and temperature parameters $T = \{1, 2, 5, 10\}$ over 60 epochs. For this evaluation, we use ResNet20. The results obtained with the proposed temperature settings are reported in Table III. It is evident that the temperature value T_2 plays an important role when retraining approximate CNNs using ApproxKD. The accuracy difference between the best and the worst final accuracy is more than 4% when the multiplier's MRE is larger than 18%. A high temperature T_2 is required for multipliers with large MRE, while a lower T_2 is suitable for multipliers with small MRE. This correlation, as explained in sub-section III-A2, can be explained by the flattening effect of high temperatures on the probability distribution of the teacher model's output. When using an approximate multiplier with large MRE, y_{approx} has a different distribution compared to y_q. Therefore, the optimization worsens when using lower values of T_2, which maintains the original probability distribution of y_q.

B. Evaluation of ApproxKD + GE

We now evaluate GE and ApproxKD separately, as well as their combination (ApproxKD + GE), using the best temperatures T_2 obtained in the previous sub-section. Regarding GE, the function $f(y_q)$ was estimated using 50 MonteCarlo simulations of a single convolution with values drawn from normal distributions, within the corresponding quantization ranges, which takes less than 1 second with our experimental settings. Examples of the estimated functions are depicted in

TABLE III: Ablation results - Fine-tuning approximate ResNet20 with ApproxKD

Multiplier		MRE[%]	Savings[%]	worst Temp.	best Temp.	Initial Acc.[%]	worst Final Acc.[%]	best Final Acc.[%]
Truncated	3	5.5	16	10	2	84.61	89.95	90.41
	4	9.6	28	1	5	37.57	89.54	89.65
	5	18.1	38	1	5	10.70	87.02	87.99
EvoApprox8b	470	2.3	1	10	2	89.16	89.57	90.55
	29	7.9	9	10	5	59.06	89.72	89.99
	111	11.6	12	1	5	41.18	88.52	89.25
	104	19.2	18	1	10	51.53	83.60	86.77
	469	20.5	18	1	10	47.14	81.25	85.51
	228	20.4	19	1	10	47.65	81.33	85.65
	145	20.5	21	1	10	46.70	81.10	85.37
	249	48.8	61	–	–	10.00	10.02	10.02

Figs. 2 and 3. The truncated multipliers have a biased error, and therefore we observe a negative slope of the approximated function. On the other hand, the error of EvoApprox multipliers (see Fig. 3) can only be estimated as a constant, due to the unbiased nature of its approximation error. Thus, $\frac{\partial f(y_q)}{\partial y} = 0$ for AMs from the EvoApprox library, and therefore f ne-tuning with ApproxKD and ApproxKD+GE delivers the same results.

We perform f ne-tuning according to the approximation stage from Algorithm 1. Note that we only f ne-tune with approximate multipliers that cause an accuracy degradation larger than 1% w.r.t. the FP accuracy. We compare the obtained results with other two methods from the literature: normal or passive retraining [4] and alpha-regularization [5], consequently refered to as normal and alpha respectively. The number of epochs is 30, the mini-batch size is 128, and we apply learning rates of ¶$1e-4$ $1e-5$◇ with a decay of 0.1 every 15 epochs. For alpha reg., we use $\alpha = 1e-11$, which generally delivers the best results (from a set of values $\alpha = 1e-6$ ▷▷▷ $1e-12$). The obtained results with ResNet20 are reported in Table V. While KD and GE outperform the other approaches separately, the combination of both, (ApproxKD + GE), always delivers the best results. The largest difference in accuracy between normal and ApproxKD + GE f ne-tuning is of 3.16%, with the truncated multiplier 5. The f ne-tuning accuracy over all epochs using this multiplier is depicted in Fig. 4. We observe that from the f rst epoch, ApproxKD + GE and ApproxKD consistently have the best accuracy improvements, followed by GE. While alpha achieves slightly better accuracy than normal f ne-tuning during the f rst 5 epochs, afterwards both have a similar behavior, which indicates that alpha underfperforms when more drastic approximations are applied. Another observation is that the f nal accuracy of the approximated CNN strongly depends on the multiplier's MRE, and not on the accuracy before retraining. Most notable is the case of truncated multiplier 5 and EvoApprox 249. Both have the same initial accuracy, but while the f rst can reach acceptable accuracy after f ne-tuning, the second multiplier, having an MRE of 48.8%, can only perform "random guessing", even after optimization. Regarding retraining time, ApproxKD + GE has a computational overhead of only 17% w.r.t. normal f ne-tuning, which takes 2027 sec. for 30 epochs in ProxSim (see Table IV).

The results obtained with ResNet32 using the same hyperparameters as ResNet20 are reported in Table VI. Overall, we observe the same tendency of ApproxKD + GE outperforming the other f ne-tuning approaches. Based on these results, we evaluate only normal f ne-tuning and ApproxKD + GE with MobileNetV2, with similar hyperparameters as before. As this CNN has larger accuracy degradation, we increase T_2 by 1, for all AMs. The obtained accuracies are presented in Table VII. In general, ApproxKD + GE achieves improved accuracy recovery even in highly complex CNNs such as MobileNetV2.

TABLE IV: Computational overhead of ApproxKD and GE

f ne-tuning method	time [s]
Normal	2027
Gradient Estimation	2169
Alpha-Regularization	2312
ApproxKD	2365
ApproxKD + GE	2386

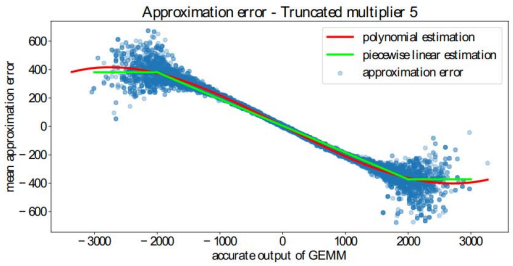

Fig. 2: Estimation of approximation error of truncated multiplier 5.

V. Conclusion and Outlook

In this work, we present an optimization f ow which combines eff cient retraining methodologies for cross-layer approximation of pre-trained CNNs. This f ow allows to apply drastic approximations in neural networks quantized to ultra low bit-widths. More specif cally, after applying 8A4W linear quantization, we explore the use of approximate multipliers with large MRE. By using two-stage knowledge distillation combined with gradient estimation for cross-layer approximate CNN retraining, we demonstrate energy savings estimated to 38%, using truncated multipliers (with an MRE close to 20%), with an accuracy loss of up to 2.37% and 2.36% (ResNet32 and ResNet20 respectively), w.r.t. the 8A4W accurate CNN. We are also able to reach energy savings of 28% with an accurcy loss of 1.76% using the complex MobileNetV2. The proposed methodologies will be further extended for lower

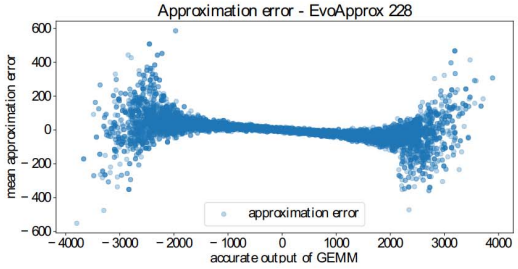

Fig. 3: Error of EvoApprox 228.

TABLE V: Comparison of retraining methods for Approximate CNNs with 8A4W linear quantization using ResNet20

Multiplier		MRE [%]	Savings[%]	Initial Acc[%]	Final Normal	Final GE	Final alpha	Final ApproxKD	Final ApproxKD+GE
Trunc.	1	0.5	2	**90.54**					
	2	2.1	8	89.67	90.31	90.35	90.29	90.39	**90.44**
	3	5.5	16	84.61	90.17	90.23	90.16	90.39	**90.41**
	4	11.0	28	40.22	89.33	89.45	89.32	89.44	**89.51**
	5	19.8	38	10.00	84.63	86.25	84.96	87.56	**87.79**
EvoA.	470	2.1	1	89.16	90.50	–	90.47	**90.55**	**90.55**
	29	7.9	9	59.06	89.90	–	89.93	**89.99**	**89.99**
	228	18.9	19	47.65	84.09	–	83.93	**85.65**	**85.65**
	249	48.8	61	10.02	10.00	–	10.04	10.02	10.02

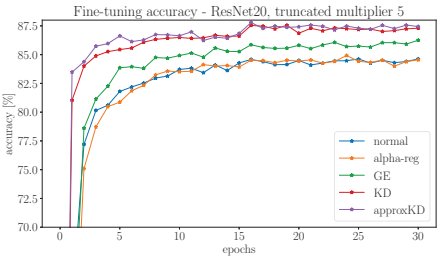

Fig. 4: Fine-tuning accuracy vs. epoch of ResNet20 approximated with truncated multiplier 5.

TABLE VI: Results of training approximate ResNet32

Multiplier		Initial Acc.[%]	Final Normal	Final GE	Final alpha	Final ApproxKD	Final ApproxKD+GE
Trunc.	1	91.11	–	–	–	–	–
	2	90.79	91.19	91.21	91.18	91.28	**91.29**
	3	87.40	90.56	90.72	90.61	90.84	**90.96**
	4	45.37	89.54	90.08	89.75	90.10	**90.19**
	5	10.01	86.77	87.95	86.78	88.12	**88.93**
EvoA.	29	54.92	89.73	–	89.72	**90.32**	**90.32**
	111	63.43	88.13	–	88.16	**89.05**	**89.05**
	104	58.70	82.29	–	83.33	**86.11**	**86.11**
	469	48.73	81.67	–	82.95	**84.57**	**84.57**
	228	48.70	81.61	–	82.70	**84.29**	**84.29**
	145	48.81	80.75	–	81.45	**84.19**	**84.19**

bitwidth quantization, as well as for the incorporation of more than one approximation technique into the CNN computation.

REFERENCES

[1] S. Vogel, J. Springer, A. Guntoro, and G. Ascheid, "Self-supervised quantization of pre-trained neural networks for multiplierless acceleration," in *DATE'19*.

[2] S. Ullah, S. Gupta, K. Ahuja, A. Tiwari, and A. Kumar, "L2l: A highly accurate logg_2_lead quantization of pre-trained neural networks," in *DATE'20*, March 2020.

TABLE VII: Results of training approximate MobileNetV2

Multiplier		Initial Acc.[%]	Final Normal	Final ApproxKD+GE
Truncated	1	93.64	93.91	**94.07**
	2	92.94	93.87	**94.02**
	3	76.62	93.24	**93.58**
	4	10.00	92.82	**93.13**
	5	10.00	85.79	**87.01**
EvoApprox8b	470	91.76	93.43	**93.78**
	228	24.19	86.79	**87.26**

[3] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," in *NIPS Deep Learning and Representation Learning Workshop*, 2015. [Online]. Available: http://arxiv.org/abs/1503.02531

[4] X. He, L. Ke, W. Lu, G. Yan, and X. Zhang, "Axtrain: Hardware-oriented neural network training for approximate inference," *Proceedings of the International Symposium on Low Power Electronics and Design*, Jul 2018.

[5] C. D. L. Parra, A. Guntoro, and A. Kumar, "Proxsim: Simulation framework for cross-layer approximate dnn optimization," in *DATE'20*.

[6] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR'16*, 2016.

[7] M. Sandler, A. G. Howard, M. Zhu, A. Zhmoginov, and L. Chen, "Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation," *IEEE Conference on Computer Vision and Pattern Recognition*, 2018. [Online]. Available: http://arxiv.org/abs/1801.04381

[8] A. Krizhevsky, "Learning multiple layers of features from tiny images," *University of Toronto*, 2009.

[9] M. Nagel, M. V. Baalen, T. Blankevoort, and M. Welling, "Data-free quantization through weight equalization and bias correction," *ICCV'19*, Oct 2019. [Online]. Available: http://dx.doi.org/10.1109/ICCV.2019.00141

[10] A. Polino, R. Pascanu, and D. Alistarh, "Model compression via distillation and quantization," *ArXiv*, vol. abs/1802.05668, 2018.

[11] M. Haroush, I. Hubara, E. Hoffer, and D. Soudry, "The knowledge within: Methods for data-free model compression," *arXiv preprint, arXiv:1912.01274*, 2019.

[12] S. Venkataramani, A. Ranjan, K. Roy, and A. Raghunathan, "Axnn: Energy-efficient neuromorphic systems using approximate computing," in *2014 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, Aug 2014, pp. 27–32.

[13] Q. Zhang, T. Wang, Y. Tian, F. Yuan, and Q. Xu, "Approxann: An approximate computing framework for artificial neural network," in *DATE'15*, 2015.

[14] M. A. Hanif, A. Marchisio, T. Arif, R. Hafiz, S. Rehman, and M. Shafique, "X-dnns: Systematic cross-layer approximations for energy-efficient deep neural networks," *J. Low Power Electron.*, vol. 14, no. 4, pp. 520–534, 2018.

[15] V. Mrazek, Z. Vasicek, L. Sekanina, M. A. Hanif, and M. Shafique, "Alwann: Automatic layer-wise approximation of deep neural network accelerators without retraining," *ICCAD'19*, Nov.

[16] S. S. Sarwar, S. Venkataramani, A. Ankit, A. Raghunathan, and K. Roy, "Energy-efficient neural computing with approximate multipliers," *J. Emerg. Technol. Comput. Syst.*, 2018.

[17] S. R. Jain, A. Gural, M. Wu, and C. Dick, "Trained uniform quantization for accurate and efficient neural network inference on fixed-point hardware," *arXiv preprint, arXiv:1903.08066*, 2019.

[18] Y. Bengio, N. Léonard, and A. C. Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation," vol. abs/1308.3432, 2013. [Online]. Available: http://arxiv.org/abs/1308.3432

[19] M. Abadi *et al.*, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. [Online]. Available: https://www.tensorflow.org/

[20] V. Mrázek, R. Hrbáček, Z. Vašíček, and L. Sekanina, "Evoapprox8b: Library of approximate adders and multipliers for circuit design and benchmarking of approximation methods," in *DATE'17*.

[21] S. S. Kidambi, F. El-Guibaly, and A. Antoniou, "Area-efficient multipliers for digital signal processing applications," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 1996.