MODEL ROBUSTNESS WITH IN-PLACE ZERO-SPACE MEMORY PROTECTION FOR CNN AT THE 4-BIT QUANTIZATION LEVEL

Shashwat Singh [*1] **Alexander Behr** [*1] **Hui Guan** [1]

ABSTRACT

Spatial, energy, and hardware costs are especially concerning for safety-critical CNN inferences as they often execute on resource-constrained (mobile) devices, the costs worsen the limit on model size and capacity, and increases the cost of the overall AI solution. It has been observed that 8-bit quantization is a prevalent, robust, and general choice to reduce model size and latency while preserving accuracy for obtaining in-place zero-space cost memory protection for CNNs. In this paper we experimentally evaluate the 4-bit quantized version of in-place zero-space ECC. We evaluate model accuracy obtained from weight-distribution oriented training and the fault-tolerance robustness of the resulting models using fault injection experiments.

1 INTRODUCTION

There has been a surge in the usage of CNNs for safety-critical applications such as autonomous vehicles, medical devices and aerospace. Reliability of CNN inference is becoming an important concern. Memory faults can result from environmental perturbations, temperature variations, voltage scaling, manufacturing defects, wear-out, and radiation-induced soft errors and can cause system level failures and violations of safety specifications. In safety-critical systems, incorrect values represent a serious concern, as these systems must comply with strict safety standards. Existing solutions such as ECC, spatial redundancy and radiation hardening incur large costs.

8-bit quantization is a prevalent, robust, and general choice to reduce model size and latency while preserving accuracy. Existing approach as the one used in (Guan et al., 2019) proposes the first zero space cost memory protection for CNNs using 8-bit quantization. It capitalizes on the opportunities brought by the distinctive properties of CNNs. It amplifies the opportunities by introducing a novel training scheme, Weight Distribution-Oriented Training (WOT), to regularize the weight distributions of CNNs such that they become more amenable for zero-space protection. It then introduces a novel protection method, in-place zero-space ECC, which removes all space cost of ECC protection while preserving protection guarantees.

As an extension to the work done in (Guan et al., 2019) we further investigate the tradeoff between the model robustness and bit width by performing experiments using 4-bit quantization in which we obtain the baseline accuracy using Quantization Aware Training followed by producing a weight-distribution regularized model using Weight Distribution Oriented Training. Fault injection experiments have been performed on these models to evaluate their robustness.

2 RELATED WORK

In (Liu et al., 2017) the authors were able to achieve misclassification after performing fault injections on selected areas whereas in our work we perform fault injections randomly to simulate the real environment setting. In (Mahmoud et al., 2020) a software-directed approach has been used to identify vulnerable computations during a CNN inference and are selectively protected based on their propensity towards corrupting the inference output in the presence of a hardware error. Though this work improves the resilience of the CNN models, it incurs additional computations. In (Zhao et al., 2020), SCA, a secure CNN accelerator that exploits stochastic computing to achieve IP protection at both training and inference phases is presented.

Unlike these above mentioned works, our work focuses on protecting all the data bits instead of selective protection. We feel that protecting all the data bits helps in attaining greater robustness of the CNN models.

In (Li et al., 2017), symptom-based detectors are implemented in software and a second technique, selective latch hardening, is used in hardware for detecting and correcting data-path faults. These techniques focus on only detecting errors whereas in our work we focus on both detection and

[*]Equal contribution [1]University of Massachusetts at Amherst. Correspondence to: Shashwat Singh <shashwatsing@umass.edu>, Alexander Behr <abehr@umass.edu>, Hui Guan <hguan@umass.edu>.

Proceedings of the 3^{rd} MLSys Conference, Austin, TX, USA, 2020.

correction of errors.

3 PREMISES AND SCOPES

This work focuses on the protection of 4-bit quantized CNN models. In the experiments, both activations and weights have been quantized to 4-bit.

3.1 QAT

The quantization algorithm we used is symmetric range-based linear quantization that is well-supported by major CNN frameworks (e.g. Tensorflow, Pytorch). Specifically, let X be a floating-point tensor and X^q be the 4-bit quantized version. X can be either weights or activations from a CNN. The quantization is based on the following formula:

$$X^q = round(X \frac{2^{n-1}-1}{max\{|X|\}}) \quad (1)$$

Where n is the number of bits used for quantization. In our case, $n = 4$. The number of bits used for accumulation is 32. Biases, if exist, are quantized to 32-bit integer.

This process is called Quantization Aware Training (QAT) and is also used in (Guan et al., 2019). The Distiller (Zmora et al., 2018) quantization framework provides an easy to use implementation which integrates with PyTorch.

As highlighted in (Guan et al., 2019) it makes more sense to focus on protection of weights rather than activations. Since weights are usually kept in the memory. The longer they are kept, the higher the number of bit flips they will suffer from. Activations, however, are useful only during an inference process. Given the slight chance of having a bit flip during an inference process, protecting activations is not as pressing as protecting weights.

3.2 WOT

Also proposed in (Guan et al., 2019), after quantization the weight-distribution oriented training process can be used for create a weight-distribution in the model in which there are non-informative bits at regular positions in parameters of the model. These non-infomative bits can be used to store other useful information, in the case of (Guan et al., 2019) and this work, parity bits. Our work uses hamming codes in which m parity bits can be used to protect $2m-m-1$ data bits. Using an extra parity bit enables Single Error Correction / Double Error Detection (SEC-DED). (Guan et al., 2019) shows that this technique in conjunction with 8-bit quantization is highly effective at improving the robustness of models to memory faults.

Our experiments have been performed using two SEC-DED configurations for error protection as shown in figure-1. The first configuration uses the SEC-DED (64,57,1) with 64 total bits, 57 data bits and 7 parity bits. To implement this configuration every fifteenth and even weight is allowed to be large, i.e. the value of weights in every even and fifteenth positions is not regulated further after quantizing it to 4-bits. The second configuration uses the SEC-DED (32,26,1) with 32 total bits, 26 data bits and 6 parity bits. To implement this configuration every 7th and 8th weight is allowed to be large, i.e the value of weights in every seventh and eighth positions is not regulated further after quantizing it to 4-bits.

We use two SEC-DED codes because there is a significant tradeoff between them. For SEC-DED (64,57,1) only 7 out of every 16 weights in the model need to be restricted to 3-bit range and the protection has 64-bit granularity. On the other hand, for SEC-DED (32,26,1) many more weights need to be restricted to the 3-bit range, 6 out of every 8, but it offers protection at 32-bit granularity. In theory SEC-DED (32,26,1) could catch and correct twice the faults as SEC-DED (64,57,1). Twice the protection could very well be worth the extra restriction on the model, but at the 4-bit quantization level every bit of freedom is important to the effectiveness of the model. Our experiments explore this tradeoff.

4 EVALUATIONS

We describe our experimental setting in section 4.1. The experiments described in section 4.2 examine the cost in accuracy of using a 4-bit quantized model compared to 8-bit and full 32-bit models. In section 4.2 we show the cost in accuracy of WOT on 4-bit models. Finally in section 4.3 we show the results of fault injection experiments on 4-bit models with and without inplace SEC-DED protections.

4.1 Experimental Settings

All experiments in this work were done using the ImageNet dataset (Deng et al., 2009) (ILSVRC 2012). In an effort to extend the results of (Guan et al., 2019) we use VGG16_bn (Simonyan & Zisserman, 2015) and SqueezeNet (Iandola et al., 2016) in all experiments and additionally ResNet18 (He et al., 2015) in the QAT and WOT accuracy experiments. These models were downloaded pretrained on Imagenet from TorchVision 1.3.1. Experiments were run using PyTorch 1.3.1, Distille 0.4.0rc0, and CUDA 10.1.

The machines on which the experiment were hosted on Google Cloud Platform, machine type n1-standard-16 with 16 virtual CPUs, 60GB of RAM, and an NVIDIA Tesla P100 with 16GB memory.

4.2 QAT Baselines

Our first set of experiments establishes the accuracy cost of quantizing to the 4-bit level. Most models can be quantized

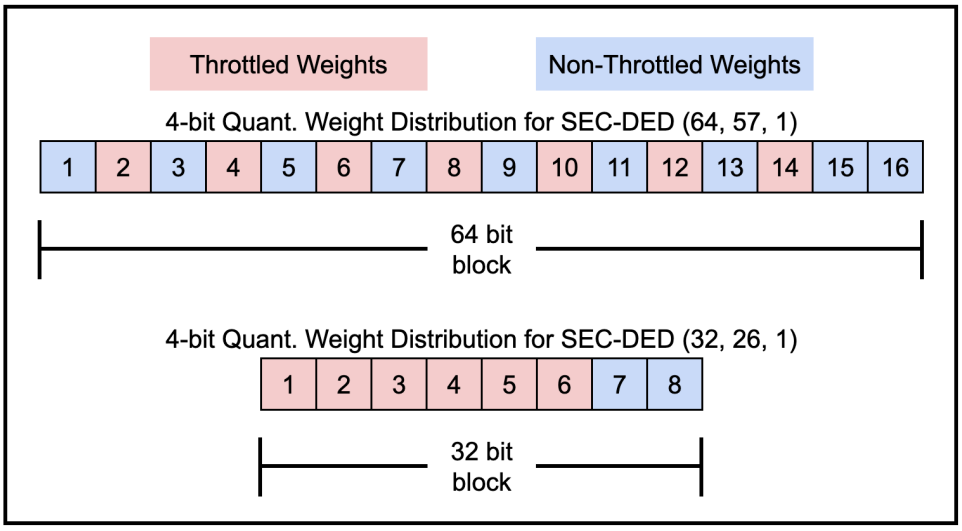

Figure 1. WOT Weight Distributions Corresponding to SEC-DED Codes

Quantization Level	Accuracy (%)		
	ResNet18	VGG16_bn	SqueezeNet
32-bit (original)	69.76	73.36	58.09
8-bit	69.07	72.01	57.01
4-bit	61.00	68.46	42.80

Figure 2. Baseline Accuracies[1]

Drop in Accuracy from 32-bit to 4-bit (%)		
ResNet18	VGG16_bn	SqueezeNet
8.76	4.90	15.29

Figure 3. Accuracy drops from 32-bit model to 4-bit model

SEC-DED Code	Accuracy (%)		
	ResNet18	VGG16_bn	SqueezeNet
(64, 57, 1)	59.92	68.21	42.58
(32, 26, 1)	58.82	67.89	43.24

Figure 4. Accuracies after WOT with 32-bit and 64-bit SEC-DED Code Weight Distributions

SEC-DED Code	Drop in Accuracy from QAT to WOT (%)		
	ResNet18	VGG16_bn	SqueezeNet
(64, 57, 1)	1.08	0.25	0.22
(32, 26, 1)	2.18	0.57	-0.44

Figure 5. Accuracy drops from 4-bit QAT model to 4-bit WOT model

to 8 bits without significant loss in accuracy, but 4-bit quantization is significantly more restrictive to the model. Our experiment started with pre-trained models from TorchVision and ran them through the quantization aware training (QAT) process described in section 3.1. The resulting quantized models had parameter of the desired bit-width, but were not regularized into a standard weight distribution. Each of these trials was run once. Figures 2 and 3 show summaries of our results[1]. While 8-bit quantized models were able to reach accuracies very close to the original models, 4-bit quantization came at a steep price. For ResNet18, VGG16_bn, and SqueezeNet the accuracies dropped, 8.76%, 4.90%, and 15.29% respectively. Interestingly the drop in accuracy is greater for models with fewer parameters. Further study is needed to determine if there is a real relationship there.

4.3 WOT Results

The next set of experiments prepared WOT models to be used for fault injection and compared the effects of WOT given the two weight distributions detailed in figure 1. These trials were run once. Figures 4 and 5 show summaries of our results. While there is some variation in the change in accuracy from the QAT models to the WOT models it does not seem to have a pattern therefor is likely due to random effects. This result indicates that the greater ratio of throttled parameters in SEC-DED (32, 26, 1) did not have an adverse effect on the accuracy of the model compared to SEC-DED (64, 57, 1).

[1]32-bit and 8-bit model accuracy data is taken from (Guan et al., 2019)

4.4 Fault Injection Results

Our fault injection experiments were carried out on 4-bit WOT pre-trained VGG16_bn and SqeezeNet models. Bits within the parameters of the model were randomly selected to be flipped at fault rates from 1e-6 to 1e-3. Before flipping the bits a fault protection strategy was simulated. Any bits which could not be protected by the strategy were actually flipped and a test set was run through the model to check the accuracy of the results. Each of these trials was run twice.

The "faulty" strategy indicates that no protection was provided and any bit selected was actually flipped. The in-place strategies indicate that a SEC-DED code matching the number of bits indicated was applied. With these strategies a single error could be corrected for every bit block within the parameters (32 bits for (32, 26, 1) and 64 bits for (64, 57, 1)). If any code block had more than one bit selected to be flipped all past the first were actually flipped and a test was run to test the accuracy of the resulting model. Each of these trials was run twice.

Our results are summarized in figure 6. There is some reason for concern. At the lowest rate of fault injection very few faults are actually injected into the in-place models (in fact in most cases none are injected). There should be no drop in accuracy when compared to the model accuracy without fault injection, yet we do see drops in accuracy. In the case of SqueezeNet the drop is very significant. This could be caused by variation in the training and testing set, but it is very unlikely given the magnitude of the accuracy drop. We are not aware of the cause of this strange result at this time.

Still, there is useful information in these results which supports the argument that this method improves the robustness of models in some cases. Figure 7 shows the difference in accuracy drop from trials with a fault rate of 1e-6 to 1e-3 for 4-bit models and 8-bit models[1] In the case of VGG16_bn the drop in accuracy is similar between the 4-bit and 8-bit models, and in the case of SqueezeNet, the 4-bit model outperforms the 8-bit model by 3.48% to 4.07%, a significant improvement.

Another important point to note in the results is that the 32-bit hamming code did not significantly outperform the 64-bit code in this case. This is interesting but the other questions raised by these figures throw doubt on any conclusion which may be drawn from this observation.

5 Limitations

This study was limited significantly by a lack of computational resources (lack of funding to pay for GPU time). For more reliable results the experiments described in section 4 should be run longer and more times.

[1].

Also there is very limited framework support for 4-bit models. If 4-bit quantized in-place zero-space fault protection were to be worth using, framework support for fast operators etc. would be needed.

The strange results summarized in figure 6 look like a bug, which casts doubts on any conclusion which could be drawn from the fault injection experiments.

6 Future Directions

The most important next step for this work would be to debug the fault injection experiments.

Once that is handled it would be interesting to see if there is a real inverse relationship between accuracy drop for 4-bit quantization and number of parameters in the model.

Finally a next step in studying robustness could be taken to examine 2-bit quantization, though it seems likely that the accuracy drop incurred by quantizing to such a small range of values would probably make any model unusable.

7 Conclusion

In this work we study an extension of the in-place zero-space fault protection proposed by (Guan et al., 2019) to 4-bit quantized models. We propose a set of weight-distributions for weight- distribution oriented training which conform to the needs of SEC-DED (64, 57, 1) and SEC-DED (32, 26, 1) error correction codes. Our evaluations show that there is a significant cost to quantizing a model to the 4-bit level. We also saw that models trained to support SEC-DED (32, 26, 1) were not significantly less accurate than those trained to support SEC-DED (64, 57, 1). The higher granularity of SEC-DED (32, 26, 1) indicates it could offer better protection at the in-place cost. Our experiments on fault injection do not support or contradict this idea. Due to possible bugs those results are questionable. However, one observation from those results is useful; accuracy of all of our 4-bit quantized models dropped less than the same model quantized to the 8-bit level. This supports the notion that quantization to lower bit-widths can provide higher robustness to faults.

References

Deng, J., Dong, W., Socher, R., Li, L., Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255, 2009. doi: 10.1109/CVPR.2009.5206848.

Guan, H., Ning, L., Lin, Z., Shen, X., Zhou, H., and Lim, S. In-place zero-space memory protection for CNN. *CoRR*, abs/1910.14479, 2019. URL http://arxiv.org/abs/1910.14479.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL http://arxiv.org/abs/1512.03385.

Iandola, F. N., Moskewicz, M. W., Ashraf, K., Han, S., Dally,

Model	Strategy	Accuracy drop (%) under different fault rates			
		1.00E-06	1.00E-05	1.00E-04	1.00E-03
VGG16_bn	faulty	1.270	1.138	3.506	66.823
	in-place 64-bit code	0.804	0.828	0.852	3.606
	in-place 32-bit code	1.072	1.072	1.104	2.272
SqueezeNet	faulty	6.725	7.417	11.099	39.158
	in-place 64-bit code	4.766	4.766	4.888	5.888
	in-place 32-bit code	6.676	6.676	6.762	8.390

Figure 6. Accuracies of 4-bit Quantized VGG16_bn and SqueezeNet without Protection and With Both SEC-DED Codes

Model	Strategy	4-bit	8-bit
VGG16_bn	faulty	65.553	21.620
	in-place 64-bit code	2.802	0.930
	in-place 32-bit code	1.200	n/a
SqueezeNet	faulty	32.433	64.710
	in-place 64-bit code	1.122	5.190
	in-place 32-bit code	1.714	n/a

Figure 7. Differences in drop in accuracy (%) from fault rate 1e-6 to 1e-3 of 4-bit and 8-bit models[1]

W. J., and Keutzer, K. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size. *CoRR*, abs/1602.07360, 2016. URL http://arxiv.org/abs/1602.07360.

Li, G., Hari, S. K. S., Sullivan, M., Tsai, T., Pattabiraman, K., Emer, J., and Keckler, S. W. Understanding error propagation in deep learning neural network (dnn) accelerators and applications. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '17, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450351140. doi: 10.1145/3126908.3126964. URL https://doi.org/10.1145/3126908.3126964.

Liu, Y., Wei, L., Luo, B., and Xu, Q. Fault injection attack on deep neural network. pp. 131–138, 11 2017. doi: 10.1109/ICCAD.2017.8203770.

Mahmoud, A., Hari, S. K. S., Fletcher, C. W., Adve, S. V., Sakr, C., Shanbhag, N. R., Molchanov, P., Sullivan, M. B., Tsai, T., and Keckler, S. W. Hardnn: Feature map vulnerability evaluation in cnns. *CoRR*, abs/2002.09786, 2020. URL https://arxiv.org/abs/2002.09786.

Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2015.

Zhao, L., Zhang, Y., and Yang, J. Sca: A secure cnn accelerator for both training and inference. In *Proceedings of the 57th ACM/EDAC/IEEE Design Automation Conference*, DAC '20. IEEE Press, 2020. ISBN 9781450367257.

Zmora, N., Jacob, G., and Novik, G. Neural network distiller, June 2018. URL https://doi.org/10.5281/zenodo.1297430.