An Efficient Bit-Flip Resilience Optimization Method for Deep Neural Networks

Christoph Schorn*†, Andre Guntoro*, Gerd Ascheid†
*Robert Bosch GmbH, Corporate Research, Renningen, Germany
†Institute for Communication Technologies and Embedded Systems, RWTH Aachen University, Germany
Email: christoph.schorn@de.bosch.com, andre.guntoro@de.bosch.com, gerd.ascheid@ice.rwth-aachen.de

***Abstract*—Deep neural networks usually possess a high overall resilience against errors in their intermediate computations. However, it has been shown that error resilience is generally not homogeneous within a neural network and some neurons might be very sensitive to faults. Even a single bit-flip fault in one of these critical neuron outputs can result in a large degradation of the final network output accuracy, which cannot be tolerated in some safety-critical applications. While critical neuron computations can be protected using error correction techniques, a resilience optimization of the neural network itself is more desirable, since it can reduce the required effort for error correction and fault protection in hardware. In this paper, we develop a novel resilience optimization method for deep neural networks, which builds upon a previously proposed resilience estimation technique. The optimization involves only few steps and can be applied to pre-trained networks. In our experiments, we significantly reduce the worst-case failure rates after a bit-flip fault for deep neural networks trained on the MNIST, CIFAR-10 and ILSVRC classification benchmarks.**

***Index Terms*—Deep neural networks, memory faults, fault tolerance, fault injection, resilience optimization**

I. Introduction

Deep neural networks (DNNs) have become the gold standard approach for various problems that involve inference from high-dimensional data [1]. To enable the widespread use of DNNs, research has recently focused on optimized hardware architectures as well as optimized algorithms for an increased energy efficiency and faster computation of neural networks [2]. Most of the energy in DNN hardware accelerators is spent for storing intermediate results in memory [3]. At the same time, the effort to ensure reliability in modern memory technology is increasing [4]–[7]. Lowering this effort at the cost of an increased memory fault rate offers potential energy savings for applications that are resilient against faults. Recent studies have shown that DNNs offer some inherent fault resilience, but this resilience can vary across different positions inside the neural network [8]–[10]. Reference [9] introduced an analytical method for the neuron resilience prediction and proposed a resilience-based mapping of neuron computations to protected and unprotected hardware elements. However, this mapping introduces extra cost for the routing and hardware protection and can decrease the flexibility to efficiently accelerate different DNN architectures. Thus, instead of protecting critical neurons, a method that increases the resilience of these neurons would be a more desirable solution.

In this paper, we present a novel technique for the resilience optimization of neural networks. In detail, our contributions are the following:

- We extend the method in [9] for predicting feature resiliencies against single bit-flip faults.
- From this analytical analysis we derive optimization steps, which aim at increasing and equalizing the bit-flip resilience across layers and within each layer of a neural network.
- We evaluate our methods using different DNNs on the MNIST [11], CIFAR-10 [12] and ILSVRC-2012 [13] image classification benchmarks.

II. Preliminaries

A. Deep Neural Networks

DNNs consist of multiple layers of neurons, which are interconnected through directed, weighted edges. The weights are optimized in a training phase and usually remain fixed afterwards. We focus on convolutional neural networks (CNNs), which are popular in the computer vision domain [14], although our methods can be applied to other DNN architectures. The basic operation of a CNN is sketched in Fig. 1. A given layer l receives an input with dimension $C^{(l)} \times M^{(l)} \times N^{(l)}$. In the first layer $C^{(0)}$ corresponds to the color channels and $M^{(0)} \times N^{(0)}$ to the spatial resolution of the input image given to the network. The convolution of the lth layer input $\mathbf{F}^{(l)}$ with filter kernels $\mathbf{W}^{(l)}$ along the spatial axes results in a 3D matrix $\mathbf{A}^{(l+1)}$ with elements[1]

$$a_{k,x,y}^{(l+1)} = \sum_{c=0}^{C^{(l)}-1} \sum_{u=0}^{U^{(l)}-1} \sum_{v=0}^{V^{(l)}-1} f_{c,x+u,y+v}^{(l)} \cdot w_{k,c,u,v}^{(l)}, \quad (1)$$
$$0 \le k < C^{(l+1)}, \quad 0 \le x < M^{(l+1)}, \quad 0 \le y < N^{(l+1)}.$$

Here $C^{(l+1)}$ is the number of output feature maps, corresponding to the different filter kernels with dimension $C^{(l)} \times U^{(l)} \times V^{(l)}$. The feature maps have the spatial resolution $M^{(l+1)} \times N^{(l+1)}$. As shown in Fig. 1, some layers additionally perform a spatial sub-sampling operation, e.g. maximum-pooling over non-overlapping 2×2 patches. Finally, to compute the *neuron output* of the layer, a rectified linear unit (ReLU) nonlinearity is applied element-wisely: $\mathbf{F}^{(l+1)} = \max(\mathbf{0}, \mathbf{A}^{(l+1)})$.

[1]Throughout this paper we use capital boldface letters to denote matrices and corresponding lowercase regular font letters to denote matrix elements.

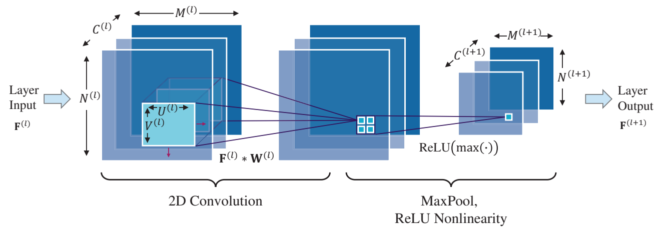
$C^{(l)}$
$M^{(l)}$
Layer Input $\mathbf{F}^{(l)}$
$N^{(l)}$
$U^{(l)}$
$V^{(l)}$
$\mathbf{F}^{(l)} * \mathbf{W}^{(l)}$
$C^{(l+1)}$
$M^{(l+1)}$
$N^{(l+1)}$
ReLU(max(·))
Layer Output $\mathbf{F}^{(l+1)}$
2D Convolution
MaxPool, ReLU Nonlinearity

Fig. 1. Schematic illustration of convolution and pooling operations inside a CNN layer.

B. Memory Faults

Modern nano-scale circuit technology is facing a number of reliability challenges, due to manufacturing variability, aging, increased temperature gradients and radiation-induced soft errors [15]. Memory technology is no exception to this trend, as field studies with high-performance computing systems show [5], [6]. For example, the authors of [16] measured logged dynamic random-access memory (DRAM) error rates of 25 000 to 70 000 errors per billion device operating hours per Mbit in Google's data centers. Field data from mobile and embedded systems has not been published so far, but we expect potentially higher error rates in these systems, due to more adverse environmental conditions, such as higher temperature variations and radiation. Memory errors in CNN accelerators can lead to silent data corruption (SDC) and potential failure of the classification task executed by the neural network [8].

C. Fault Model

In this paper, we consider transient memory faults that result in bit-flip errors in the feature maps of a CNN. Such single event upsets (SEUs) can be for example the result of high-energetic particle strikes that alter the charge stored within a memory cell. Since typical CNN accelerators require only few milliseconds for an image classification [17], the probability for multiple SEUs occurring during one inference is very low. Thus, we measure the effect of a single faulty bit on the network output. In a second step, we additionally evaluate resilience at higher bit error rates (BERs).

III. Neural Network Resilience Prediction

A. Existing Methods

The error resilience analysis of modern DNNs on hardware has recently gained attention. A number of studies have used fault injection simulation to empirically measure this error resilience [8], [10], [18]. However, simulation-based methods have some drawbacks. To get statistically meaningful results, a larger number of fault injections has to be performed and each time the resulting accuracy of the network has to be evaluated. Especially if a resilience measurement for each individual neuron is required, this becomes very time-consuming for large-scale DNNs. Furthermore, it can be difficult to derive optimization steps from these measurements.

In contrast to simulation methods, resilience estimation methods try to derive neuron resilience values analytically. Gradient-based methods have been proposed, which use error backpropagation to assign resilience values to neurons [19], [20]. It was found that neurons which strongly contribute to the network output error are less resilient than neurons with a low contribution. Reference [9] introduced a different approach, which results in more accurate resilience estimates. Instead of error backpropagation, layer-wise relevance propagation (LRP) [21] was used to determine mean neuron relevancies over a set of training data. We take this as the basis for our resilience optimization and therefore briefly introduce the method in the following.

LRP first propagates a given input image $\mathbf{F}^{(0)}$ forwards through the network and then propagates neuron relevancies for the classification of this image backwards using the following propagation rule

$$r^{(l)}_{k,x,y} = \sum_{(c,u,v)\in E} \frac{f^{(l)}_{k,x,y} \cdot \max\left(0, w^{(l+1)}_{k,c,u,v}\right) \cdot r^{(l+1)}_{c,u,v}}{\sum_{(\tilde{k},\tilde{u},\tilde{v})\in G} f^{(l)}_{\tilde{k},\tilde{u},\tilde{v}} \cdot \max\left(0, w^{(l+1)}_{\tilde{k},c,u,v}\right)}, \quad (2)$$

where E is the set of indices (c,u,v) corresponding to outbound weight connections of neuron (k,x,y) in layer l and G is the set of indices for inbound weight connections of neuron (c,u,v) in layer $l+1$. The initial relevance $\hat{\mathbf{R}}^{(L)}$ at the output of the last layer is set to 1 for the output neuron belonging to the target class of the image and 0 for all other neurons. It was suggested in [9] to compute mean relevance values for all neurons of the network based on a set D of training images:

$$\hat{\mathbf{R}}^{(l)} = \frac{1}{|D|} \sum_{\mathbf{F}^{(0)}\in D} \mathbf{R}^{(l)}\left(\mathbf{F}^{(0)}\right). \quad (3)$$

B. Bit-Flip Fault Resilience Prediction

In the following, we introduce two modifications of the previously described method, which proved to be useful for the optimization of resilience against bit-flip faults.

1) Feature-Aggregated Relevance: Convolutional layers extract features at multiple positions of their input by shifting different weight kernels over this input, which results in a feature map for each kernel. Though different positions in this feature map can have different average relevancies, we are only interested in a global relevance score per extracted feature. This can be obtained by summing up the result from (3) over the spatial dimensions:

$$\hat{\rho}^{(l)}_{k} = \sum_{m=0}^{M^{(l)}-1} \sum_{n=0}^{N^{(l)}-1} \hat{r}^{(l)}_{k,m,n}, \quad 0 \le k < C^{(l)}. \quad (4)$$

2) Initial Error Propagation: The effect of a single bit-flip fault in the output of a layer only concerns a single value in a feature map of that layer. However, when propagated through the next following layer, it can spread across all feature maps. We suggest to take this initial error propagation into account and consider its effect on the features with different relevance values in the subsequent layer. Error propagation certainly depends on the current operating point of the layer, due to the nonlinearity function at the layer output. For example, if the input to a ReLU is much less than zero, small perturbations of this input are masked. However, as an approximation, the error amplification in the first layer behind the fault linearly depends on the magnitude of the weights connected to the faulty neuron. Based on this, we define a feature bit-flip criticality score $\hat{\eta}_c^{(l-1)}$ as

$$\hat{\eta}_c^{(l-1)} = \sum_{k=0}^{C^{(l)}-1} \sum_{u=0}^{U^{(l)}-1} \sum_{v=0}^{V^{(l)}-1} \frac{\text{abs}\left(w_{k,c,u,v}^{(l)}\right)}{C^{(l)} \cdot U^{(l)} \cdot V^{(l)}} \hat{\rho}_k^{(l)}, \quad 0 \leq c < C^{(l-1)}. \tag{5}$$

This criticality score attains a high value, if a fault in feature c of layer $l-1$ strongly affects those features k in layer l, for which the average relevancy $\hat{\rho}_k^{(l)}$ is high.

IV. Resilience Optimization of Neural Networks

Using the analytical insights from the previous section, we derive a methodology for the resilience optimization of DNNs. Our goal is to obtain a more homogeneous resilience distribution inside the DNN, which obviates the need for special protection of critical parts in the network. We take two aspects into account: *Architectural optimization*, which focuses on the avoidance of critical bottleneck layers, and *feature optimization*, which focuses on balancing the feature criticality inside each layer.

A. Architectural Optimization

It was pointed out in [9], that the average resilience of a layer is proportional to the number of neurons in that layer. Consequently, layers with only few neuron outputs are less resilient against faults and should be avoided from a reliability standpoint. On the other hand, additional neurons increase the energy consumption of the hardware, since additional memory transfers and multiply-accumulate operations have to be performed. Moreover, a larger network that requires more memory will also experience a higher amount of random bit-flips. Thus, our architectural optimization step focuses only on critical bottleneck layers, which have very few neurons compared to the other layers of the network and therefore drastically increase the worst case failure rate. In our experiments we show that in some cases such bottlenecks can be avoided with only a minor modification of the network architecture.

B. Feature Optimization

Error resilience typically also varies between the features of a layer. Feature optimization focuses on equalizing this *intra-layer* resilience. We achieve this without changing the architecture of the network and thus avoiding any computational overhead at run time. Based on (5), the criticality scores of the features in a given layer can be equalized by adjusting the weights of the subsequent layer accordingly:

$$\tilde{w}_{k,c,u,v}^{(l)} = \frac{w_{k,c,u,v}^{(l)}}{\hat{\eta}_c^{(l-1)} \cdot C^{(l-1)}} \sum_{\tilde{c}=0}^{C^{(l-1)}-1} \hat{\eta}_{\tilde{c}}^{(l-1)} \quad 0 \leq k < C^{(l)}, \quad 0 \leq c < C^{(l-1)}, \quad 0 \leq u < U^{(l)}, \quad 0 \leq v < V^{(l)}. \tag{6}$$

In (6) a normalization factor for $\hat{\eta}_{\tilde{c}}^{(l-1)}$ is used, to obtain an average weight scaling factor of one. Thus, weights connected to a feature with above average criticality are scaled down, while weights connected to a feature with below average criticality are scaled up. The adjustment of the network weights initially leads to a degradation of the DNN classification accuracy, which is why a re-training of the network has to be performed afterwards. However, this fine-tuning requires only a small fraction of the steps compared to the the original training of the DNN. Since network training affects feature relevancies and weights again, we suggest an iterative procedure of weight adjustment and fine-tuning, as shown in Algorithm 1.

Algorithm 1 Feature Resilience Optimization

```
Input: Pre-trained and quantized model $DNN_0$, training dataset $D$, number of iterations $T$.
Output: Resilience-optimized model $DNN_T$.
 1: for $t = 0$ to $T-1$ do
 2:     for each $layer \in DNN_t.layers$ do
 3:         $\rho \leftarrow$ GetFeatureRelevance $(layer, D)$
 4:         $w \leftarrow$ GetLayerWeights $(layer)$
 5:         $w \leftarrow$ AdjustWeights $(w, \rho)$ {using equation (6)}
 6:         SetLayerWeights $(layer, w)$
 7:     end for
 8:     if $t < T-1$ then
 9:         $DNN_{t+1} \leftarrow$ FineTune $(DNN_t, D)$
10:     end if
11: end for
```

In the last iteration, no fine-tuning is performed to keep the weights constant after the final adjustment. Our experimental results show that the network adapts to the feature resilience equalization after few iterations and classification accuracy does not drop below the original value anymore.

V. Experiments

A. Networks and Datasets

To evaluate our methods, we use three common image classification benchmarks, namely MNIST [11], CIFAR-10 [12] and ILSVRC-2012 [13]. For MNIST and CIFAR-10 we have designed and trained two rather small-scale CNNs, as summarized in Table I, following common design strategies. MNIST consists of 28×28 pixel gray-scale images of handwritten digits and CIFAR-10 of 32×32 pixel RGB images of different objects. Both have 10 classes. The ILSVRC-2012 benchmark is considerably more challenging, since it has 1000 different categories and images with 224×224 pixel RGB-colored resolution. For this benchmark we use the SqueezeNet [22] CNN architecture.

B. Quantization

DNNs are often trained on graphics processing units (GPUs) using 32-bit floating-point arithmetic. However, when these networks are deployed on a dedicated accelerator, bit-width can be reduced to save energy [2]. Furthermore, limiting the full-scale range of the data representation improves the error resilience of the networks [10]. We emulate a linear quantization scheme with dynamic fixed-point format in our

TABLE I

ARCHITECTURE OF THE CNNS, USED FOR THE MNIST AND CIFAR-10 BENCHMARKS RESPECTIVELY, BEFORE ARCHITECTURAL OPTIMIZATION.

Layer	Filter Shape $U^{(l)} \times V^{(l)}$		Output Shape $C^{(l+1)} \times M^{(l+1)} \times N^{(l+1)}$	
	MNIST	**CIFAR-10**	**MNIST**	**CIFAR-10**
0	3×3	3×3	$16 \times 28 \times 28$	$32 \times 32 \times 32$
1	3×3 maxp. 2×2	3×3	$64 \times 14 \times 14$	$32 \times 16 \times 16$
2	3×3	3×3	$64 \times 14 \times 14$	$64 \times 16 \times 16$
3	global avg. pool	3×3	$64 \times 1 \times 1$	$64 \times 8 \times 8$
4	1×1	3×3	$10 \times 1 \times 1$	$128 \times 8 \times 8$
5		3×3		$128 \times 8 \times 8$
6		global avg. pool		$128 \times 1 \times 1$
7		1×1		$10 \times 1 \times 1$

experiments. Unless otherwise noted, a bit-width of 8 for both the activations and weights of the DNNs is used. We determine the optimal full-scale range for each layer using the optimization procedure described in [23]. This allows us to apply quantization to pre-trained networks with little to no degradation of the classification accuracy.

C. Fault Injection Framework

We use fault injection simulations to evaluate the results of our optimization procedure. For this purpose we have developed a simulation framework on top of Keras [24] with TensorFlow back-end [25] that allows us to perform fast bit-level fault injections in the feature maps of a CNN. Most of the computation workload required for the simulation can be efficiently computed on a GPU. The framework automatically adds some operations behind each neuron output stage of a given CNN, which emulate a fixed-point format and allow for a bit-wise fault injection in the neuron output memory by applying a definable Boolean fault mask (see Fig. 2).

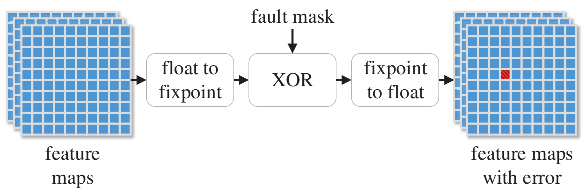

Fig. 2. Steps performed by our fault injection framework between the computation of two neural network layers.

D. Failure Rate

Our goal is to determine the probability that a bit-flip leads to a *failure* of the classifier. As failure we define the inability to correctly classify an input image that would have been correctly classified without the fault. We determine the failure rate by injecting a fault in a neuron activation and then measuring the fraction of incorrect classifications over a set of input images. This set consists of the originally correctly classified samples from a test set of 10 000 images for each CNN. Unless otherwise noted, we report the failure rate for faults on the most significant bit (MSB) of neuron outputs, which have the largest effect compared to other bit positions.

E. Results

1) Effect of Network Architecture: Intermediate layers with very few neuron outputs in comparison to the remaining layers are significantly less resilient than the rest of the network. Taking a look at the architectures in Table I, one can see that the second last layers of the two networks have only 64 and 128 output neurons respectively. The architectures of these networks can be optimized by moving the global average pooling layers at the end and placing the 1×1 convolutions before them. This simple change neither affects the number of weights, nor has a significant effect on the achievable classification accuracy after training. However, it increases the output size of the second last layer to $64 \times 14 \times 14 = 12\,544$ for the MNIST CNN and $128 \times 8 \times 8 = 8192$ for the CIFAR-10 CNN respectively. It can be seen in Fig. 3a and Fig. 4a that worst-case failure rates after a bit-flip fault are substantially reduced by this optimization. The histograms count the number of neurons which, when hit by a bit-flip fault, lead to a certain failure rate of the network. Failure rates are evaluated for each individual neuron of the networks.

SqueezeNet also has bottleneck layers. However, since the smallest layer outputs still consist of 8112 neurons, these layers are less critical compared to the MNIST and CIFAR-10 case. We found that increasing the number of neurons in the bottleneck layers can further increase resilience, but comes at the cost of a much higher number of weights and operations. Thus, we regard architectural optimization as inefficient and only perform feature optimization in this case.

2) Feature Optimization Results: We further optimize bit-flip resilience using the method described in Algorithm 1. The number of iterations is chosen for each CNN such that the classification accuracy after the final weight adjustment lies above the baseline of the original float32 model. It can be seen in Fig. 5a that the method quickly converges to this point for all three benchmarks, while smaller networks require fewer iterations. Furthermore, Fig. 5b shows that the mean over all layers of the standard deviation (SD) of the normalized feature criticality scores is reduced, which indicates that resilience becomes more homogeneously distributed across features.

The effect of our feature optimization method on the failure rate after a single MSB-flip fault can be seen in Fig. 3b, Fig. 4b and Fig. 6 for the for the MNIST CNN, CIFAR-10 CNN and ILSVRC SqueezeNet respectively. In all three benchmarks feature optimization can decrease the worst case failure rate by more than 40%. While we tested each individual neuron for the MNIST and CIFAR-10 networks, for ILSVRC we limited our analysis to 48 positions in each of the four most critical feature maps according to (5), due to the much longer simulation time for this benchmark.

3) Evaluation of Multi-Bit-Flip Resilience: Our methods and previous analysis focused on the single bit-flip fault case. Now, we evaluate, if our optimization also has a positive effect

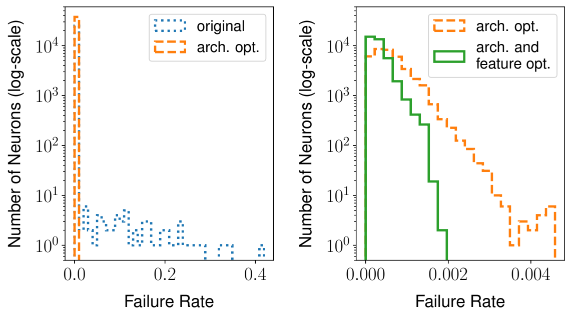

Fig. 3. Histogram plots showing the number of neurons of the MNIST CNN that result in a certain failure rate, when an MSB-flip occurs in their output. (a) Comparison between original and architectural optimized model. (b) Comparison of architectural optimized model before and after feature optimization.

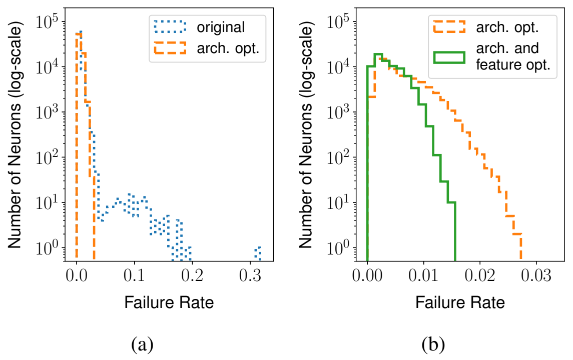

Fig. 4. Histogram plots showing the number of neurons of the CIFAR-10 CNN that result in a certain failure rate, when an MSB-flip occurs in their output. (a) Comparison between original and architectural optimized model. (b) Comparison of architectural optimized model before and after feature optimization.

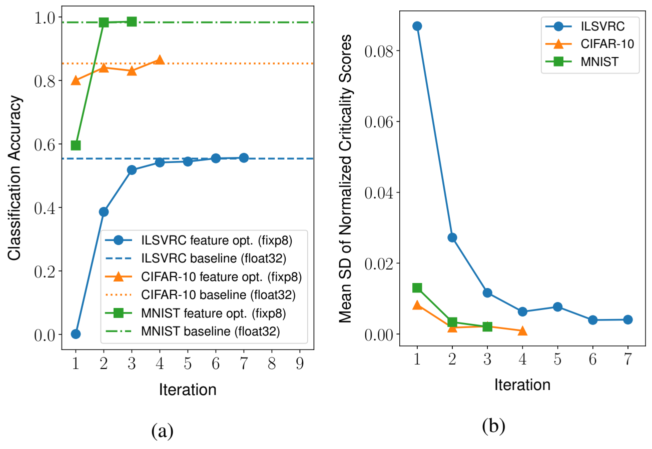

Fig. 5. Convergence of the feature optimization algorithm for the architectural optimized and quantized benchmark networks, measured after setting the adjusted weights in each iteration of Algorithm 1 (a) Classification accuracies in comparison to the original float32 models. (b) Mean SD of normalized feature criticality scores in each iteration: $\frac{1}{L}\sum_{l=0}^{L-1} \mathrm{SD}\left(\frac{\hat{\boldsymbol{\eta}}^{(l)}}{\mathrm{mean}(\hat{\boldsymbol{\eta}}^{(l)})}\right)$.

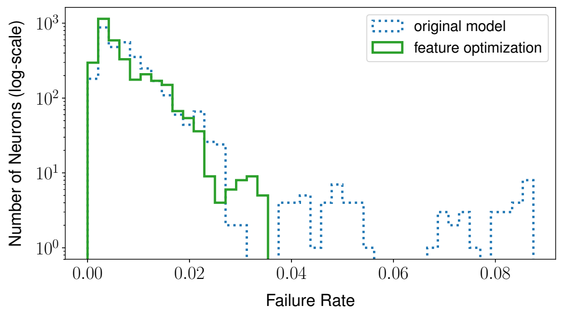

Fig. 6. Histogram plot showing the number of neurons of the ILSVRC SqueezeNet that result in a certain failure rate, when an MSB-flip occurs in their output. Comparison between original and feature-optimized model. A subset of the most critical neurons of the networks was evaluated.

on the resilience against multiple bit-flip faults that occur during an image classification. Therefore we inject bit-flips randomly, with a uniform probability for each bit position, in all DNN activations and measure resulting failure rates on the test sets. We do this for different BERs and report mean values for 1000 runs for each BER on MNIST and CIFAR-10 and 100 runs for each BER on ILSVRC. The results are shown in Fig. 7. Resilience-optimization significantly lowers the failure rates in all cases, except for ILSVRC and MNIST with a BER of 10^{-2}. The mean relative failure rate reductions over all BERs are about 15% for ILSVRC, 27% for CIFAR-10 and 29% for MNIST.

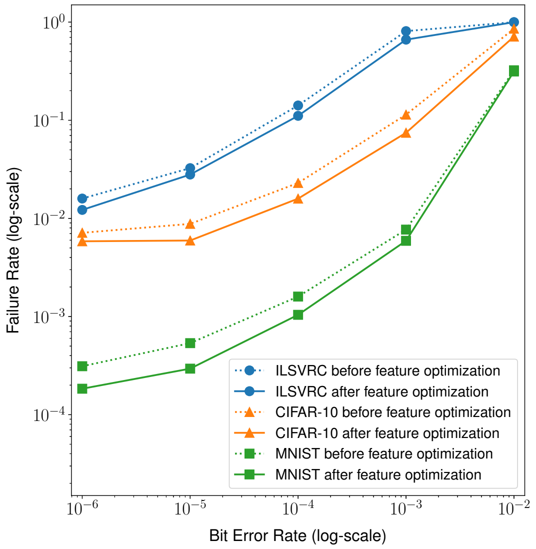

Fig. 7. Failure rates at different BERs before and after feature optimization. Each bit of all network activations is flipped with the corresponding BER.

VI. Related Work

Hardening neural networks against hardware faults is an active field of research. A number of publications follow the approach of introducing hardware faults during neural network training [26]–[30]. Reference [26] considers timing variations,

[27], [28] SRAM supply voltage scaling and [29], [30] hard defects in memristors and resistive RAM respectively. The drawback of these approaches is that they complicate the training process, since fault injections have to be performed by placing hardware in the training loop or through realistic fault simulations. In contrast, our feature optimization method uses an efficient and easy to implement fine-tuning procedure and can be applied to pre-trained networks.

A significance driven mapping of network weight bits to memory cells with different fault resilience is suggested by [31]. This approach is less flexible, since the optimal fraction of resilient and less resilient memory cells can differ for different network architectures. Furthermore, the authors did not follow a mathematical approach to determine weight resiliencies, but relied on their intuition.

Reference [32] proposes a resilience optimization by replication of critical neurons and weights. This is related to our architectural optimization approach. However, they use exhaustive simulation to determine criticality values, which is infeasible for large-scale DNNs. Their approach could be improved by combining it with our resilience estimation.

VII. Conclusion

We have introduced a novel fault resilience optimization method for DNNs. This method has a theoretical foundation in the relevance analysis of neurons. It is easy to implement and can be efficiently applied to pre-trained networks, requiring only few fine-tuning steps. We proved the effectiveness of the method with several benchmark networks on large-scale image recognition benchmarks. Our results show that failure rates after single and multiple bit-flip faults in neuron activations can be significantly reduced. This enables the use of memory elements with lower fault protection, which are easier to realize and more energy efficient.

References

[1] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[2] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, “Efficient processing of deep neural networks: A tutorial and survey,” *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[3] Y.-H. Chen, J. Emer, and V. Sze, “Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks,” in *43rd International Symposium on Computer Architecture (ISCA)*, 2016, pp. 367–379.

[4] O. Mutlu, “The RowHammer problem and other issues we may face as memory becomes denser,” in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2017.

[5] J. Meza, Q. Wu, S. Kumar, and O. Mutlu, “Revisiting memory errors in large-scale production data centers: Analysis and modeling of new trends from the field,” in *45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2015, pp. 415–426.

[6] V. Sridharan *et al.*, “Memory errors in modern systems: The good, the bad, and the ugly,” in *Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2015, pp. 297–310.

[7] Y. Chen, H. H. Li, I. Bayram, and E. Eken, “Recent technology advances of emerging memories,” *IEEE Design & Test*, vol. 34, no. 3, pp. 8–22, 2017.

[8] G. Li *et al.*, “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*, 2017.

[9] C. Schorn, A. Guntoro, and G. Ascheid, “Accurate neuron resilience prediction for a flexible reliability management in neural network accelerators,” in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2018.

[10] B. Reagen *et al.*, “Ares: A framework for quantifying the resilience of deep neural networks,” in *55th Annual Design Automation Conference (DAC)*, 2018.

[11] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[12] A. Krizhevsky, “Learning multiple layers of features from tiny images,” Master Thesis, University of Toronto, 2009.

[13] O. Russakovsky *et al.*, “Imagenet large scale visual recognition challenge,” *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015.

[14] J. Gu *et al.*, “Recent advances in convolutional neural networks,” *Pattern Recognition*, vol. 77, pp. 354–377, 2018.

[15] J. Henkel *et al.*, “Reliable on-chip systems in the nano-era,” in *50th Annual Design Automation Conference (DAC)*, 2013, pp. 695–704.

[16] B. Schroeder, E. Pinheiro, and W.-D. Weber, “DRAM errors in the wild: A large-scale field study,” in *SIGMETRICS/Performance’09*, 2009.

[17] A. Canziani, A. Paszke, and E. Culurciello, “An analysis of deep neural network models for practical applications,” *CoRR*, vol. abs/1605.07678, 2016.

[18] J. Marques, J. Andrade, and G. Falcao, “Unreliable memory operation on a convolutional neural network processor,” in *IEEE International Workshop on Signal Processing Systems (SiPS)*, 2017.

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

[20] Q. Zhang, T. Wang, Y. Tian, F. Yuan, and Q. Xu, “ApproxANN: An approximate computing framework for artificial neural network,” in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2015, pp. 701–706.

[21] S. Bach, A. Binder, G. Montavon, F. Klauschen, K.-R. Müller, and W. Samek, “On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation,” *PLOS ONE*, vol. 10, no. 7, p. e0130140, 2015.

[22] F. N. Iandola, S. Han, M. W. Moskewicz, A. K, W. J. Dally, and K. Keutzer, “SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and $<$ 0.5MB model size,” *CoRR*, vol. abs/1602.07360, 2016.

[23] S. Vogel, M. Liang, A. Guntoro, W. Stechele, and G. Ascheid, “Efficient hardware acceleration of CNNs using logarithmic data representation with arbitrary log-base,” in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2018.

[24] F. Chollet *et al.* (2015) Keras. [Online]. Available: https://keras.io

[25] M. Abadi *et al.* (2015) Tensorflow: Large-scale machine learning on heterogeneous distributed systems. [Online]. Available: https://www.tensorflow.org/

[26] J. Deng *et al.*, “Retraining-based timing error mitigation for hardware neural networks,” in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2015, pp. 593–596.

[27] L. Yang and B. Murmann, “SRAM voltage scaling for energy-efficient convolutional neural networks,” in *18th International Symposium on Quality Electronic Design (ISQED)*, 2017, pp. 7–12.

[28] S. Kim, P. Howe, T. Moreau, A. Alaghi, L. Ceze, and S. Visvesh, “MATIC: Learning around errors for efficient low-voltage neural network accelerators,” in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2018.

[29] L. Xia, M. Liu, X. Ning, K. Chakrabarty, and Y. Wang, “Fault-tolerant training with on-line fault detection for RRAM-based neural computing systems,” in *54th Annual Design Automation Conference (DAC)*, 2017.

[30] C. Liu, M. Hu, J. P. Strachan, and H. Li, “Rescuing memristor-based neuromorphic design with high defects,” in *54th Annual Design Automation Conference (DAC)*, 2017, pp. 1–6.

[31] G. Srinivasan, P. Wijesinghe, S. S. Sarwar, A. Jaiswal, and K. Roy, “Significance driven hybrid 8T-6T SRAM for energy-efficient synaptic storage in artificial neural networks,” in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2016.

[32] F. M. Dias, R. Borralho, P. Fontes, and A. Antunes, “FTSET: A software tool for fault tolerance evaluation and improvement,” *Neural Computing and Applications*, vol. 19, no. 5, pp. 701–712, 2010.