MOZART+: Masking Outputs with Zeros for Improved Architectural Robustness and Testing of DNN Accelerators

Stephane Burel, Adrian Evans, Lorena Anghel

MOZART+: Masking Outputs with Zeros for Improved Architectural Robustness and Testing of DNN Accelerators

Stéphane Burel†, Adrian Evans†, Lorena Anghel‡
†Université Grenoble Alpes, CEA, LIST, Grenoble, France
‡Université Grenoble Alpes, CEA, CNRS, Grenoble INP*, INAC-Spintec
Email: {stephane.burel,adrian.evans}@cea.fr, lorena.anghel@grenoble-inp.fr

***Abstract*—Deep Neural Networks (DNNs) are increasingly used in safety critical autonomous systems. We present MOZART+, a DNN accelerator architecture which provides fault detection and fault tolerance. MOZART+ is a systolic architecture based on the Output Stationary (OS) data-flow, as it is is the data-flow that inherently limits fault propagation. In addition, MOZART achieves fault detection with on-line functional testing of the Processing Elements (PEs). Faulty PEs are swiftly taken off-line with minimal classification impact. We perform a detailed fault-injection study on this systolic architecture, considering different fault-models and comparing different measures of accuracy. We show how to handle the case of layers with a small number of neurons. The implementation of our approach on Squeezenet results in a loss of accuracy of less than 3% in the presence of a single faulty PE, compared to 15-33% without mitigation. The area overhead for the test logic does not exceed 8%. Dropout during training further improves fault tolerance, without a priori knowledge of the faults.**

***Index Terms*—neural network, accelerator, robustness, fault tolerance, convolution**

I. Introduction

THERE are many types of DNN accelerators, spanning GPUs, FPGAs and dedicated ASICs. Researchers are primarily focused on the design of accelerators optimized for performance and energy efficiency [1], [2]. One of the most promising solutions used for the hardware implementation of DNNs are systolic architectures consisting of an array of processing elements (PEs) since they maximize local re-use of activations and weights [3], thus minimizing the costly data transfers with the external memory. These accelerators, if used in safety critical applications, have to show high robustness. Architectural fault-tolerance techniques can achieve robustness with minimal overhead.

Integrated circuits used in automotive applications must comply with ISO-26262. The most critical components need to detect and react to faults in an interval that is shorter than the *fault tolerant time interval*, the maximum time the fault can be present without posing a safety risk. As on-line testing becomes mandatory, new techniques must be developed that have lower overhead than traditional on-line testing.

Standards also impose stringent requirements on overall FIT (Failures in Time) rate (e.g. $\leq$10 FIT for ASIL-D) which are difficult to achieve for large DNN circuits, as highlighted by [4]. State-of-the-art automotive DNN accelerators employ dual-modular redundancy (DMR) [5], but this strategy is increasingly costly and does not exploit the specificity of DNNs.

Fig. 1. MOZART Provides On-Line Testing and Fault Masking with Graceful Degradation in Accuracy

In this paper, we propose an architecture called MOZART+, which combines several new concepts which can be applied to provide fast on-line testing and fault mitigation for the data-path of a DNN accelerator. MOZART relies on two major points : 1) During system operation, fault detection is performed at-speed in order to detect critical faults on the data-path; 2) An existing fault mitigation technique has been adapted to provide graceful degradation in classification accuracy, in the presence of faults, as illustrated in Figure 1. This can be augmented by a training technique which further increases robustness without a priori knowledge of the hardware faults.

The remaining sections are organized as follows. Sec. II presents our architecture. In Sec. III we define the case studies used in our experiments and then in Sec. IV we present an analysis of the impact of faults, in the absence of the Mozart techniques. Then, in Sec. V, we show the effectiveness of our techniques for detecting and masking faults. Sec. VI presents related work on DNN fault tolerance and finally, we conclude with Sec. VII.

II. MOZART Architecture

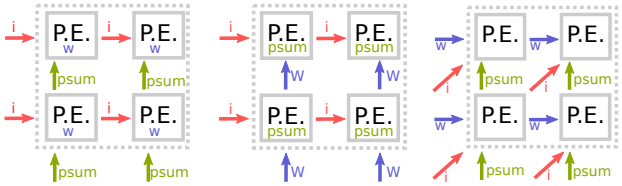
MOZART is a systolic Output Stationary (OS) DNN accelerator with data-path fault detection and mitigation. The data-flow in a DNN accelerator influences how faults propagate through the logic towards the outputs and thus has a significant impact on the accuracy. We show that the OS architecture effectively contains the impact of fault, as a small number of neurons are impacted. A PE in an OS architecture roughly corresponds to a single neuron, thus making it possible to

use an adaptation of the dropout technique to further improve the fault tolerance during the training phase. MOZART also exploits the fact that DNNs can obtain high accuracy even when certain intermediate calculations are masked to zero.

A. Systolic Architectures

Systolic architectures reduce memory transfers by reusing data. They consist of a fixed size array of PEs which perform multiply-accumulate (MAC) operations and the PEs transfer data to their direct neighbours. The calculations of the abstract network are mapped to the PE array.

There are three broad classes of systolic architectures: *weight*, *output* and *row stationary* as shown in Figure 2. They rely on re-use of either weights, activations or both. We briefly present these architectures, however, the reader is referred to [3] for a full explanation.



(a) WS Architecture (b) OS Architecture (c) RS Architecture

Fig. 2. Data-flow in Systolic DNN Accelerator Architectures

1) Weight Stationary Data-flow: With a Weight Stationary (WS) data-flow [6], each column contains weights for a given output channel and performs the calculations for this channel. Weights are pre-loaded into the PEs and remain stationary. Activations flow horizontally and partial sums flow upwards. After several cycles, the sums arrive at the top of the column.

2) Output Stationary Data-flow: Alternatively, in an output stationary (OS) architecture (such as Shidiannao [7]), the partial sums stay fixed in the PE. The columns share weights, and rows share input features. Each PE is dedicated to a single output pixel. Weights and input features flow between the PEs.

3) Row Stationary Data-flow: In a row stationary (RS) architecture (such as Eyeriss [8]), 2-D convolutions are broken into 1-D convolutions, which are processed in a single PE. PEs store multiple weights for the same row and perform simultaneous MAC operations but only need storage of one partial sum. This architecture achieves re-use of both weights and partial sums. We consider a PE which stores 4 weights.

B. Fault Detection and Mitigation

With the MOZART approach, we introduce an on-line functional testing technique which ensures high coverage for impacting faults. During the testing procedure, each PE is taken off-line, one at a time. The PE's outputs are set to zero, exploiting the fact that when a single PE is forced to zero, the impact on classification accuracy is negligible [9], [10]. This enables deterministic scheduling of the testing of every PE.

Rather than relying on logic Built-In Self Test (BIST) to perform on-line testing, we propose an approach based on functional test. The inputs of the PE under test are connected to those of its neighbour, as shown in red in Figure 3. After computation of a partial sum, an external comparator checks the result of the PE under test with that of its neighbour. If the outputs don't match, the PE under test is taken off-line, meaning its external outputs remain set to zero. Once a PE is tested, the next PE starts the test procedure. This fine-grained scheduling minimizes detection time, quickly detects the highly impacting SA1 faults and has no impact on the latency of the calculation.

A transient fault in a PE could result in a mis-match and thus needlessly take the PE off-line. This is prevented by including all PEs in the periodic test procedure, even those that are off-line. If an off-line PE shows no further errors after multiple test iterations, it is then taken back on-line, ensuring that transient faults don't result in the loss of PEs.

We have synthesized a 16x16 PE array, with 8-bit integer multipliers, 32 bit adders, and registers for the partial sums. Synthesis results show that the area overhead of the extra test logic is under 8%.

When a PE is found to be faulty, its external output is set to zero, which in DNN applications produces only a minor loss of accuracy.

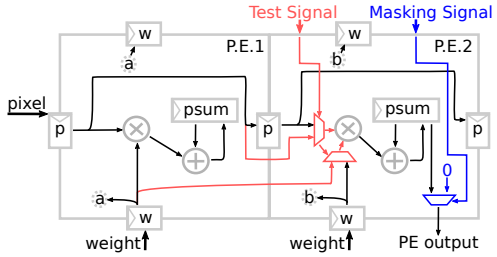
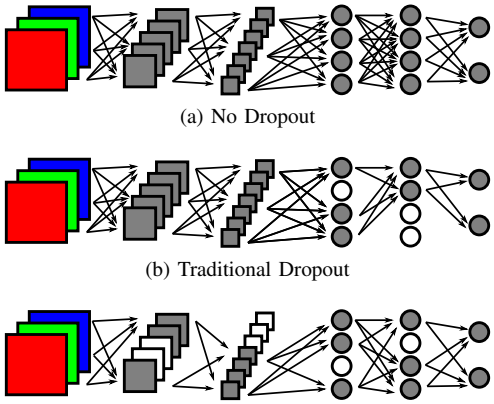


Fig. 3. Schematic of MOZART PE Showing Added Circuitry for Fault Fault Detection (red) and Output Masking (blue)

C. Output Stationary Architecture for Fault-Tolerant Training

Early on, neural network researchers identified the fact that if faults are injected in a network during training, it can improve the fault tolerance during inference [11], [12] *and* it can avoid the problem of over-fitting. Dropout, a technique now commonly used during training on the fully connected layers to prevent over-fitting, consists of setting the value of randomly selected neurons to zero (Figure 4b). This feature is available in all major DNN frameworks.

Some recent authors [13], [14] have shown an interest in using dropout during training to improve fault-tolerance. Solovyev's [13] study was done with random faults in the weights in an abstract network and he showed that dropout in all layers during training can improve the tolerance to these faults. Lee [14] identified that dropout improved the tolerance to stuck-at-zero faults, but had little benefit for random faults, and from this we extrapolate that dropout during training provides tolerance to hardware faults that resemble the removal of a neuron from the network. In the case of RS and WS architectures, PEs in an accelerator do not map directly to neurons, thus limiting the ability of dropout to mitigate PE hardware faults. However, in an OS architecture,



(a) No Dropout

(b) Traditional Dropout

(c) Mozart's Fault Tolerant Training

Fig. 4. Representation of Random Dropout During Training

one abstract neuron maps to a single PE. With MOZART, after any fault is detected, the output of the PE is masked to zero which corresponds to disconnecting neurons from the abstract network, very similar to what occurs during dropout. *By design*, the hardware architecture of MOZART is well suited to exploit dropout during training. Since the same PEs are re-used to compute all layers, not only is dropout applied to the fully-connected layers, but to all the layers (Figure 4c),

The combination of detecting and masking faulty PEs to zero with modified dropout during training ensures that, the classification accuracy is minimally impacted in the presence of faults. It is important to note that the proposed dropout technique does not require a priori knowledge of the location of the faults nor does it result in a degradation of accuracy, in the absence of faults.

III. Case Study

To evaluate the MOZART architecture, we performed a fault-injection study on three different DNNs which have been mapped to multiple systolic architectures.

A. Selected DNNs

We have chosen three networks for our experimental study. Two of the networks classify 50,000 images from the ImageNet test data-set into 1000 categories. Top-1 accuracy indicates whether the top ranked category is correct. Top-5 accuracy indicates that the correct category is among the five top-ranked categories, and this is the metric we have used to evaluate the classification accuracy. One modern and compact network, SqueezeNet [15], has been selected as our primary test case, as it is representative of the networks used in embedded applications. Since many, previous fault tolerance studies [9], [13], [16] have used VGG-16 [17] and LeNet-5, we have also included these two networks. VGG-16 is a large network with a huge number of weights, thus it inherently has more redundancy. LeNet-5 is a small network that uses the MNIST data-set for the recognition of hand-written digits and is no longer representative of modern applications. With LeNet-5, Top-1 accuracy is used as there are only ten categories.

These networks are summarized in Table I. An 8-bit integer format was chosen, as this data representation has proven to be more robust than floating point, while maintaining a minimal loss in accuracy [18].

TABLE I
Characteristics and Accuracy of Selected Networks

Network	Num. Neurons	Num. MACs	Num. Weights	Dataset	Accuracy Top-1	Accuracy Top-5
SqueezeNet	2.6 M	352 M	1.2 M	ImageNet	55%	75%
VGG-16	13.6 M	15 G	138 M	ImageNet	70%	91%
LeNet-5	6518	341 K	60 K	MNIST	99%	-

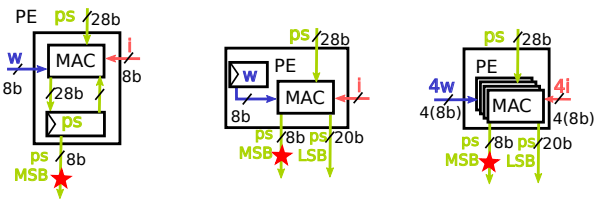
B. Hardware Fault Model

In this study we focus on computational faults occurring at the PE level. Therefore, we define a fault model that covers stuck-at faults affecting the outputs of the PE, as shown Figure 5. Our results thus hold, regardless of the specific implementation of the PE.

The focus of our study is an improved understanding of the propagation of data-path faults, therefore we have adopted a high-level fault-model. We do not claim that this model reflects all possible logic errors occurring in a PE, however, we assert that it is sufficient to compare the propagation of faults between PEs in the three different systolic architectures.

Previous works have studied faults in the DNN weights and PE register files [16], [19], [20] which can be protected with parity or ECC. Others have proposed test methodologies for the interconnect [21]. Protecting the computational logic in the PE is more difficult, as it requires fault tolerance strategies that typically have high area costs, timing penalties and are architecture dependent.

For the WS and RS architectures, we only inject faults in the 8 Most Significant Bits (MSbits) of the partial sum, because the 20 Least Significant Bits (LSBits) are inherently discarded in the data-path of this architecture. A fault in the LSBits could carry into the MSBits, however, to be able to fairly compare results, we wanted to have the same number of faulty bits (8), in all our experiments. Furthermore, as shown later, the LSBits are not very sensitive.



(a) Output Stationary (b) Weight Stationary (c) Row Stationary

Fig. 5. PE Fault Models in OS, WS and RS Architectures

C. Methodology

The objective is to evaluate the impact of faults on different architectures and to evaluate the effectiveness of techniques to mask faulty PEs. For each condition, we evaluate the classification accuracy. The distinction between *average* accuracy and *worst-case* accuracy is important as certain, specific, faults cause the accuracy to drop close to zero.

The evaluation is performed by selecting $N_{batches}$ different, random faults and then, for each fault, testing the accuracy using N_{batch_size} images. Our high-level algorithm is shown in Algorithm 1. For each condition, we report an average accuracy (for all the faults) as well as a *worst-case accuracy* - that is the accuracy of the worst batch.

Algorithm 1 Fault Injection Procedure

```
for all Fault Mitigation in (None, Zero Masking) do
  for all Architecture in (OS, WS, RS) do
    for all PE Array Size in (16x16, 64x64) do
      for all Network in (VGG, SqueezeNet, LeNet) do
        for all Faulty PEs in (1, 2, 4) do
          for all Faulty bits in (1, 2, 4) bits do
            for i from 1 to N_batches do
              Fault ← (random(PE, bits, value))
              Inject Fault
              Batch ← N_batch_size random test images
              Classify Batch
              Record accuracy
              Clear Fault
            end for
            Evaluate Average and Worst Case Accuracy
          end for
        end for
      end for
    end for
  end for
end for
```

1) Batch Size: For each condition, the total number of experiments is $N_{batch\_size} \cdot N_{batches}$, which is large, thus there is no problem evaluating the *average* accuracy. However, for the measurement of the *worst-case* accuracy, N_{batch_size} must be selected with care. Prior to injecting any faults, we performed a series of experiments with VGG-16. In the absence of faults, we set $N_{batches} = 1000$ and swept the value of N_{batch_size}. The results are shown in Figure 6 where the average accuracy is shown with the green line. The blue line show the worst case accuracy observed for each batch size. For example, with a $N_{batch_size} = 25$, we see that at least one batch had an accuracy as low as 64%. In this case, the *worst case* value is not the result of faults, just the random selection of test images.

Based on these results, we selected to use $N_{batch\_size} = 100$ for the remainder of the experiments. With this value, the worst-case accuracy, in the absence of faults, is 10% lower than the average. In the presence of faults, if we observe a *worst case* accuracy that is more than 10% below the average, we can conclude it was indeed the result of the faults, and not a statistical anomaly.

D. Tool Flow

The N2D2 open-source neural network framework [22] was used for performing the experiments. It has support for quantized networks and the ability to output a 'C' model.

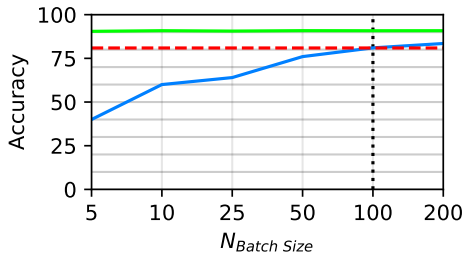


Fig. 6. VGG-16 Worst Accuracy versus Batch Size

By modifying the 'C' model we emulated the effect of faults in the PEs, injecting the types of faults described in Sec III-B. The model was modified so that for each MAC operation, we determined on which PE it would be executed, and depending on whether this PE was faulty, the result of the MAC was corrupted. In this way, the effect of faults in a systolic hardware architecture were emulated using a much faster model. The tool flow is shown in Figure 7.

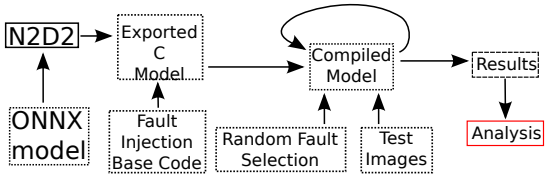


Fig. 7. Fault Injection Methodology

IV. Experimental Results

This section presents the results of our fault injection study, prior to applying the Mozart techniques for detection and mitigation and it is organized as follows. Sec. IV-A presents the impact of faults on the three architectures, without mitigation. In Sec IV-B and IV-C we evaluate the impact of faults in different bit positions and different layers in the network.

A. Impact of Architecture

Before considering any mitigation techniques, we studied the inherent fault tolerance of the three different architectures. In an OS data-flow, each neuron is mapped to a single PE. However, due to folding, which is required to fit the input map into the PE array, and the fact that the PE array is re-used for multiple channels and layers, a given PE is re-used for computing multiple neurons. When there is a fault in a PE, it affects a limited set of specific neurons.

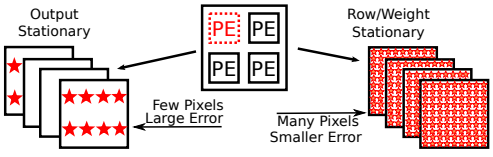


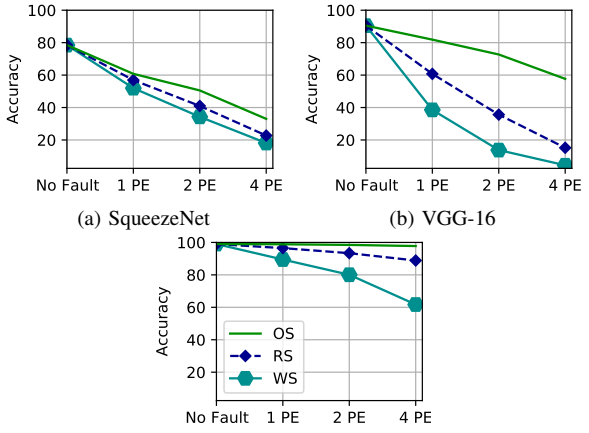
Fig. 8. Propagation of a Fault in a PE in Different Architectures

On the other hand, in a WS architecture, each PE contributes to the computation of every neuron for a channel. In this architecture a fault in a PE propagates to all the neurons for

an output channel. As only one term in the sum is affected, the numerical impact of the fault is lower.

In a RS architecture, a fault in a PE impacts the result of a 1-D convolution, propagates to multiple output pixels and it is thus similar to a WS architecture. This fault propagation behaviour is illustrated in Figure 8.

We performed a series of fault injections to study the fault propagation process and assess the fault tolerance of the three architectures. For this analysis, we limited the fault injection scenarios from one to four PEs, as beyond this number, the drop in accuracy is unacceptable. For each data point on the graphs, 1000 randomly selected PEs and bit positions were selected and then 100 randomly selected images were analyzed in order to evaluate the top-5 classification accuracy (top-1 for LeNet-5). The results are summarized in Figure 9 and we see that in all cases, the OS architecture has a higher accuracy. Note, when not stated otherwise, we present results for an array of 256 PEs.



(a) SqueezeNet (b) VGG-16

(c) LeNet-5

Fig. 9. Average Accuracy of Three Systolic Architectures (OS, RS, WS) in the Presence of a Single Faulty Bit

B. Impact of Faults by Bit Position and Number of Bit Flips

Li [4] performed a study on the impact on accuracy of faults in different bit positions, however, in this paper, we show how this sensitivity varies for two systolic architectures. We performed 40,000 fault injection in each bit position, separately for SA0 and SA1 faults and for both OS and WS architectures and the results are plotted in Figure 10. We observe that with the OS architecture (in blue), faults in the LSBits (6..0) have virtually no impact, and that in the MSBits, only SA1 faults cause the accuracy to drop. This is in contrast to the WS architecture (in green), where SA1 faults in all bit positions are critical and SA0 faults cause a significant drop in accuracy. We note that the impact of SA0 faults in the MSBits of the WS architecture is slightly lower than the LSBits and this is because, due to the distribution of the weights, in many cases, these MSBits are already zero. This bit-by-bit analysis confirms the coarse-grained results presented in the previous section and corroborates the insights about fault propagation illustrated in Figure 8, namely that with an WS architecture, even minor faults in the LSBits propagate and significantly impact the final result.

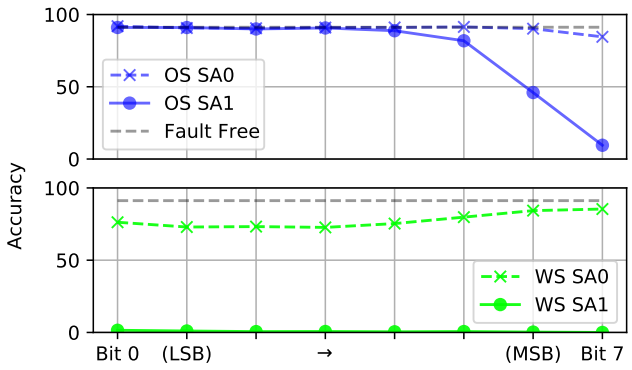
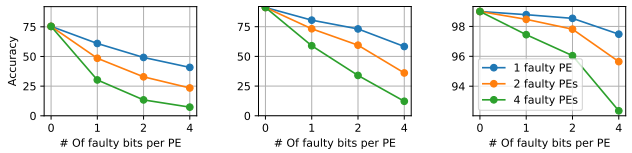


Fig. 10. Average Accuracy Versus Bit-Position for VGG-16 with WS and OS Architectures

We also investigated the impact of the number of stuck-at bits on the final classification accuracy for the OS architecture and the results are shown in Figure 11. Each graph, has curves for the case of 1, 2 or 4 faulty PEs, and the number of faulty bits varies on the horizontal axis. The important thing to note, is that the trend in the drop of accuracy, is similar, regardless of the number of stuck-at bits. With VGG-16, we see that with a single faulty PE, with four stuck-at bit faults, the accuracy is about 60%, with two faulty PEs, each with two stuck-at faults, the accuracy is about 60% and with a four faulty PEs, each with a single stuck-at fault, the accuracy is still around 60%. From these graphs, it appears that with the OS architecture, the drop in accuracy depends on the total number of stuck-at faults, regardless how they are distributed across PEs. For the remainder of the paper, we have limited our fault model to a single SA bit per faulty PE.



(a) SqueezeNet (b) VGG-16 (c) LeNet-5

Fig. 11. Impact of Number of Faulty Bit (x-axis) on Three Networks for an OS Architecture

C. Impact of Faults by Layer

Several authors [4], [23]–[25] have studied the impact of faults in different layers of DNNs, focusing on specific accelerators. We performed a series of experiments, for two architectures, where single bit faults were inserted in the each layer of VGG-16 and the results are shown in Figure 12. In this figure, the dots show the average accuracy and error bars show worst-case accuracy. The first observation, is that the OS architecture (in blue) is more robust for every layer, and, even for worst-case accuracy, only a few layers show a drop in accuracy. On the other hand, with the WS architecture

(in green), the worst-case accuracy drops to zero for all convolutional layers, and in the middle layers, even the drop in average accuracy is significant.

It is interesting to contrast these results to those from [25], who also studied the sensitivity of the layers of VGG-16 but executing on a GPU. In their study, they report that the first layers are most critical (which is similar to the findings of Li [4]. We see a similar trend, where the worst-case OS accuracy drops significantly for the first two layers. However, with the WS, the middle convolutional layers are most sensitive. We believe this is because the features being treated in the middle convolutional layers are coarse grained, and due to the broad fault propagation with the WS architecture, these faults are likely to impact the final classification.

Considering the results in Figure 12 and previous works, it becomes clear that it is difficult to draw general conclusions about the sensitivity of specific layers, as this depends significantly on the fault model and the data-flow.

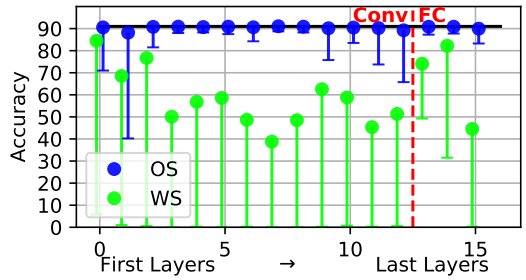


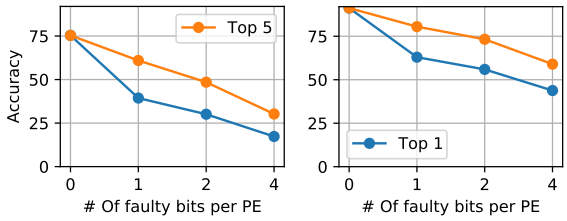
Fig. 12. Accuracy of VGG-16 with Faults in Different Layers - Dots show average and error bars show worst case

D. Accuracy Measured by Top-1 versus Top-5

Many authors [26] have used the ImageNet dataset when evaluating the fault tolerance of DNN hardware. Most have chosen Top-5 accuracy as their measure of the impact of faults, however, this metric alone, does not tell us whether the ranking was changed. We simply measure whether the image being analyzed dropped out of the Top-5 ranking. To ensure that our conclusions would hold if the stricter Top-1 metric were used, we evaluated both metrics in the presence of faults. In Figure 13, for the OS architecture, we plot the accuracy in the presence of single-bit SA faults with 1, 2 or 4 faulty PEs. We note that the relative drop in accuracy is similar with both metrics. Thus, for the remainder of this work, we only report results for Top-5 accuracy, so that our measurements can be compared with those from other studies.

V. Mozart Detection and Masking Techniques

In this section, we show how the Mozart techniques provide effective fault detection and masking. Sec. V-A shows the effectiveness of the proposed functional test strategy. Sec. V-B shows the behaviour of the networks when PEs are masked to zero. Sec. V-D analyzes the efficiency of the dropout technique. Sec V-C shows how a shortcoming in the original



(a) SqueezeNet (b) VGG-16

Fig. 13. Top-1 and Top-5 Average Accuracy versus Number of Faulty PEs for OS Architecture

proposal can be applied, when dealing with small networks. Finally, Sec. V-E demonstrates the scalability of the approach.

A. Effectiveness of On-Line Test

A key element of the MOZART approach is on-line testing, which continuously checks the sanity of the PEs, as stuck-at-1 (SA1) faults on a single erroneous PE can severely impact the accuracy. The main idea is to sequentially take individual PEs out of the computation and compare the result with their neighbour's. The PE under test is advanced each round of calculation, that is, the computation of the neurons in the array completes and the array is reloaded. For VGG-16, to compute a single image, the array is loaded 55 679 times.

To evaluate the detection capability of this technique, we injected SA0, SA1 fault injections on every bit of the outputs of each of the 256 PEs (4096 faults). For each fault, 100 randomly selected images were evaluated to see if the fault could be detected by comparing the partial sum of the PE with that of its neighbour. Table II shows the percentage of all SA1 faults detected after a given number of images.

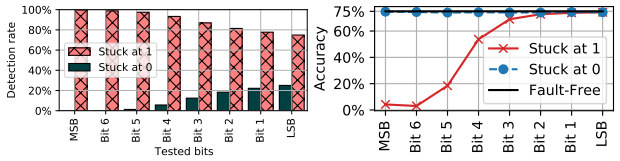
TABLE II
Percentage of SA1 Faults Detected After N Images

Percentage of SA1 Faults Detected After N Images	Number of Images			
	1	2	4	8
SqueezeNet	100%	100%	100%	100%
VGG-16	100%	100%	100%	100%
LeNet-5	94%	98%	99%	100%

For the larger networks, a *single* image was sufficient to detect all the SA1 faults. This was true for *any* of the 100 random images we used as stimulus for VGG16 and SqueezeNet. For LeNet-5 with a single image, 94% of the SA1 faults could be detected, which is already a good level of coverage. In fact, for LeNet-5, more images are required because, with a small network, the number of operations performed per image is is small (see Table I). These experiments show that the low-overhead approach to on-line test can quickly detect the highly impacting SA1 faults.

Detecting SA0 faults is more difficult because the neuron output values are often zero, especially the MSBits of the integer representation. In Figure 14a, for Squeezenet, we show the fraction of SA faults that were detected after a single round of computation. SA1 faults are easily detected, especially in the MSBits. Conversely, SA0 faults in the MSBits are hard to detect. This is not an important issue, as a single SA0 fault

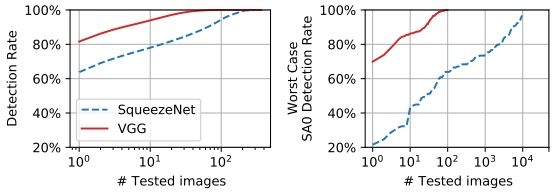
has a negligible impact on classification accuracy, as shown in Figure 14b. These results are consistent with [10], [19].



(a) Detection Rate after One Round (b) Accuracy Impact of Single Bit Fault

Fig. 14. Detection Rate and Accuracy with Single Bit SA Faults on Squeezenet

To understand if SA0 faults can be detected with functional test, we performed two further experiments, focusing on Squeezenet and VGG. First, we considered the 2048 possible SA0 faults, and for each of these faults, we applied 100 randomly selected images. In Figure 15a, we report the percentage of these faults that are detected after a given number of images. We see that after 100 images, all SA0 faults were detected for VGG-16 and 94% are detected for Squeezenet. This difference can be explained since in VGG-16 a typical PE is tested 215 times for every input image, versus only 42 for Squeezenet. With LeNet-5, as the 16x16 PE array is under-utilized, the SA0 detection is poor and not presented.



(a) Average (b) Worst case

Fig. 15. Number of Images Required to Detect Latent SA0 Faults

To obtain a bound on the worst case number of images required to detect any SA0 faults in Squeezenet, we performed a second experiment. For every possible SA0 fault, 10,000 images were evaluated. We then ranked the images from the one which detected the fewest faults to the one that detected the most. In Figure 15b we present the cumulative detection rate, starting on the left with the image with the lowest coverage. Thus, from this graph, we can observe a worst case detection time for SA0 faults. After 10,000 images, we succeed in detecting 97% of the SA0 faults, which is reasonable, given that these faults are benign.

One might argue that with a more accurate fault model (eg. faults inside the arithmetic logic of the MAC), by relying on the incoming images as stimuli for the on-line testing, there is a risk of latent faults not being detected. If such faults exist, and if the current, real-world stimuli do not activate them, they are not immediately of concern. If the stimulus changes, and the previously latent fault is activated, then the fault is quickly detected by our continuous on-line testing.

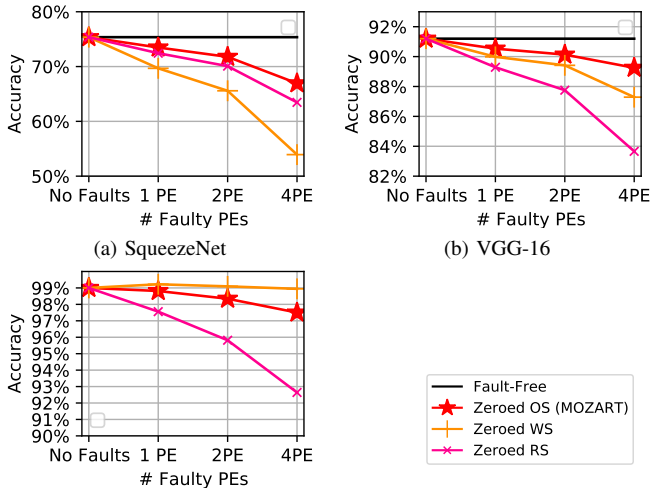
B. Fault Masking

In the previous section, we have discussed the procedure for on-line testing. To ensure a fault tolerant design, after detecting a faulty PE, we suppress the potentially serious impact of the fault by forcing the outputs of the faulty PE to zero, removing it from subsequent computations.

1) Average Accuracy: In Figures 16a to 16c we show the *average* classification accuracy when the output of one, two or four PEs is masked to zero, for each of the three architectures (OS,WS,RS).

A key point is that for these networks, the output masking technique is most effective for the OS architecture and for Squeezenet and VGG-16, the drop in accuracy when a single PE's output is masked is very small. As more PEs are masked, with the OS architecture, the loss in accuracy is gradual.

LeNet5 has only 10 neurons in the output layer so the impact of faults in this layer has a high probability of corrupting the final result. In this case the OS architecture is more sensitive but the loss in accuracy with 4 masked PEs is only 1.5%.



(a) SqueezeNet (b) VGG-16

(c) LeNet-5

Fig. 16. Average Accuracy with PEs Masked to Zero

2) Worst Case Accuracy: Figure 16 only shows the *average* accuracy. To study the worst-case, we performed a second series of experiments. We generated ≈100,000 groups of 100 randomly selected images and analyzed the accuracy with each architecture. For the unprotected architectures, we injected random faults (1,2 or 4 PEs, with 1 SA faults). For the architectures protected with zero masking, we set to zero 1,2 or 4 PEs. In Table III, we report the minimal observed accuracy across all the groups. First, note that with the unprotected architectures, in the worst case, the accuracy drops to zero for the large networks, even with faults on a single PE. This is a key observation for safety applications.

The second observation is that, with zero masking, the worst case accuracy of the MOZART architecture is *significantly* better than RS and WS, especially for SqueezeNet. With WS and RS architectures, there exist certain faults that cause a drastic drop in accuracy, unlike the OS architecture which

limits the extent of propagation, as illustrated in Figure 8. This is an important take away when designing fault tolerant systolic accelerators.

TABLE III
WORST CASE ACCURACY FOR GROUPS OF 100 IMAGES (HIGHER IS BETTER)

	Network	Unprotected				Zero Masking		
Fault Type		Fault Free	OS	WS	RS	OS (Mozart)	WS	RS
1 PE	SqueezeNet	64	0	0	0	**58**	2	52
	VGG	82	0	0	0	**82**	**82**	79
	Lenet	95	5	5	4	95	**96**	77
2 PEs	SqueezeNet	64	0	0	0	**53**	1	45
	VGG	82	0	0	0	**80**	77	74
	Lenet	95	2	3	2	92	**95**	72
4 PEs	SqueezeNet	64	0	0	0	**53**	0	16
	VGG	82	0	0	0	**79**	73	63
	Lenet	95	4	2	2	90	**95**	58

C. Mozart+

In the previous section, the MOZART approach provided improved fault tolerance for the larger Squeezenet and VGG-16 networks, but for LeNet-5 (figure 16c) we see that the WS architecture actually performs better. The problem is that in LeNet-5 there are only ten output neurons. If any one of these is faulty, and thus masked to zero, an entire output class is lost, resulting in a major loss in accuracy.

When processing fully connected layers, in a systolic architecture, it is necessary to use batching to fully utilize the PE matrix. Multiple images are processed simultaneously, with each line processing a different image. To avoid the problem of losing an output class, we propose an alternate and simple technique for the last layer (called Mozart+). Namely, after a one (or several) faulty PE(s) are detected, when processing the last layer, we propose to reduce the batching factor by two. The images in the reduced size batch are mapped to rows that contain non-faulty PEs. With this approach, we can handle up to N/2 faulty PEs, where N is the size of the array. The improved fault tolerance for LeNet is shown in figure 17. In table IV shows the compute overhead due to the reduced batch size. This is shown as the number of additional times the array needs to be loaded.

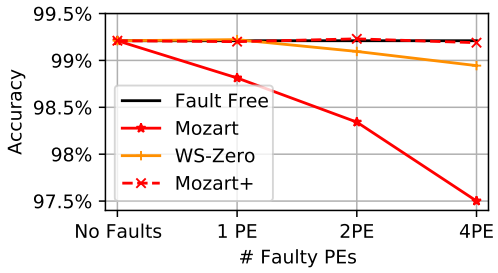


Fig. 17. Fault Tolerance of LeNet with Mozart+ Technique - Accuracy versus Number of Faulty PEs

TABLE IV
NUMBER OF LOADS OF A 16X16 OS PE ARRAY FOR EACH NETWORK

Network	All but last FC Layer	Last FC Layer	Mozart+ Overhead
Lenet	83	1	1.2%
VGG	55 616	63	0.1%
SqueeezeNet	11 554	-	-

D. Dropout During Training

Dropout during training can be used to improve the robustness during inference. We trained Squeezenet with 5% dropout on *all* layers, whereas typically dropout is only applied to the last layers. The training time increased ($\approx 3x$), but the resulting configuration provided additional fault tolerance, as shown in Figure 18. Using the newly trained network and a MOZART architecture, we injected single bit faults in a varying number of PEs. With faults in 4 PEs, due to the new training, the accuracy increased from 67% to 72%. This modified training was performed once, with no knowledge of the faults. This is different from [9], where the re-training was done based on knowing the position of the faults.

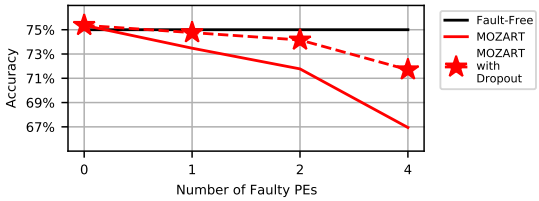


Fig. 18. Squeezenet Accuracy with Faulty PEs when Trained with Dropout

E. Scalability

Up to now, data has been presented for a 256 PE (16x16) array. We tested the MOZART architecture for different PE array sizes and the results are shown in Figure 19. These results are for Squeezenet with a single SA fault on a single PE. As expected, when the array size increases, the impact of a single faulty PE is reduced. In all cases, MOZART provides a significant improvement.

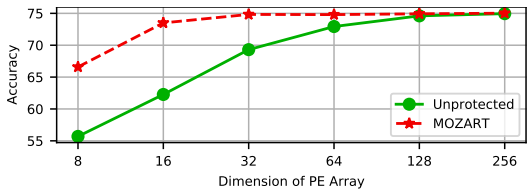


Fig. 19. Squeezenet Accuracy for Varying Array Sizes (1 SA fault on 1 PE)

VI. RELATED WORK

Historically, the fault tolerance of systolic arrays has been studied with the objective that the final computed result be identical in the presence of a fault [27]. For DNNs, this constraint can be relaxed, since classification accuracy is never perfect and the requirement is only that it remain acceptable.

Until recently, most studies of DNN fault tolerance worked with an abstract model of the network, independent of the hardware. In [16], the authors perform a sensitivity study of three networks applying bit flips in the weights. In the *Ares* framework [19], faults are injected in the weights and activations of the abstract model. In [18], the authors study GPU-based DNNs and conclude that small, integer numeric formats result in increased robustness compared to floating point. An in-depth study of fault injections on the RTL model of an accelerator mapped to a FPGA is presented in [28]. It is shown that SA0 faults have minimal impact on accuracy but their study is limited to a small data-set.

In [29], the authors propose an opportunistic approach for on-line testing of DNN accelerators, exploiting the fact that, due to folding, some PEs are temporarily idle. Since not all PEs are tested, they can can not *guarantee* coverage, which is a requirement for safety standards.

In [10], the authors propose a fault tolerant DNN accelerator using Razor techniques to detect timing faults. The main focus is low voltage operation, not safety applications.

The work of Zhang [9] is the closest to that presented in this paper. They propose fault mitigation for a WS accelerator, using a multiplexer to mask the output of the erroneous multiplier of a PE. They also propose Fault-Aware Pruning(FAP). Their results are interesting, but have certain limitations. First, they have chosen a WS Architecture, which does not limit fault propagation. Also, the FAP techniques requires knowledge of the hardware faults prior to training. Finally, they propose no test technique to detect faults and only study LeNet-5, which is not representative of modern DNNs.

VII. Conclusions

Autonomous systems that rely on DNNs must meet safety requirements. To address this need, we have presented the MOZART approach for a fault tolerant DNN accelerator. We analyzed the robustness of the three most common systolic architectures used in DNN accelerators (OS,WS and RS), a study which has not been performed previously and show that the OS architecture inherently limits the propagation of faults.

The second aspect of MOZART, is an on-line fault detection scheme, based on temporarily taking PEs out of service to compare their results with their neighbours. Using this technique, within the time required to process a single image, all the SA1 faults can be detected for the two large networks. Fast fault detection is a requirement for safety and we achieve this with low hardware overhead.

Third, we have shown that connecting the output of a PE to zero to mask known faults is particularly effective with the OS architecture. We not only analyzed the average classification accuracy for a large number of faults, as is common in other studies, we also measured a *worst case* accuracy when groups of images are analyzed. A key take away, is that there exist pathological faults which can cause the worst case accuracy to drop close to zero. This is an important message, and future studies of fault tolerance in DNNs should consider this metric.

Other authors have shown that DNNs can be trained to perform well when a set of known faults are present but this is of little practical value, as it would require massive compute resources to perform a customized training for each device. We have shown that dropout applied to all layers during training, increases fault tolerance, regardless of which PE is faulty.

A full safety analysis of an integrated circuit is beyond the scope of a scientific paper. It would require considering technology specific faults and analyzing the full design including control logic. Nonetheless, the concepts we presented constitute innovative safety mechanisms, applicable to the architecture and design of the data-path of systolic DNN accelerators.

References

[1] A. Reuther *et al.*, "Survey and Benchmarking of Machine Learning Accelerators." IEEE, Sep. 2019, pp. 1–9.

[2] Y. Wang *et al.*, "Benchmarking the performance and energy efficiency of ai accelerators for ai training," in *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*, May 2020, pp. 744–751.

[3] V. Sze *et al.*, "Efficient Processing of Deep Neural Networks: A Tutorial and Survey," *Proceedings of the IEEE*, vol. 105, no. 12, Dec. 2017.

[4] G. Li *et al.*, "Understanding error propagation in deep learning neural network (DNN) accelerators and applications." ACM Press, 2017.

[5] K. Matsubara *et al.*, "A 12nm autonomous-driving processor with 60.4tops, 13.8tops/w cnn executed by task-separated asil d control," in *2021 IEEE International Solid- State Circuits Conference*, vol. 64, 2021.

[6] S. Park *et al.*, "4.6 A1.93tops/W scalable deep learning/inference processor with tetra-parallel MIMD architecture for big-data applications." IEEE, Feb. 2015, pp. 1–3.

[7] Z. Du *et al.*, "ShiDianNao: shifting vision processing closer to the sensor." ACM Press, 2015, pp. 92–104.

[8] Y.-H. Chen *et al.*, "Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks." IEEE, Jun. 2016, pp. 367–379.

[9] J. J. Zhang *et al.*, "Fault-Tolerant Systolic Array Based Accelerators for Deep Neural Network Execution," *IEEE Design & Test*, vol. 36, no. 5, pp. 44–53, Oct. 2019.

[10] B. Reagen *et al.*, "Minerva: Enabling Low-Power, Highly-Accurate Deep Neural Network Accelerators." IEEE, Jun. 2016, pp. 267–278.

[11] R. Clay and C. Sequin, "Fault tolerance training improves generalization and robustness," in *[Proceedings 1992] IJCNN International Joint Conference on Neural Networks*, vol. 1, 1992, pp. 769–774 vol.1.

[12] J. Nijhuis *et al.*, "Limits to the fault-tolerance of a feedforward neural network with learning." IEEE Comput. Soc. Press, 1990, pp. 228–235.

[13] R. A. Solovyev *et al.*, "Study of Fault Tolerance Methods for Hardware Implementations of Convolutional Neural Networks," *Optical Memory and Neural Networks*, vol. 28, no. 2, pp. 82–88, Apr. 2019.

[14] M. Lee *et al.*, "Fault tolerance analysis of digital feed-forward deep neural networks," in *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2014, pp. 5031–5035.

[15] F. Iandola and K. Keutzer, "Keynote: small neural nets are beautiful: enabling embedded systems with small deep-neural- network architectures," in *2017 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, Oct 2017, pp. 1–10.

[16] Arechiga and Michaels, *The Robustness of Modern Deep Learning Architectures against Single Event Upset Errors*, 2019, 67.

[17] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *3rd International Conference on Learning Representations, ICLR 2015*, 2015.

[18] F. F. dos Santos *et al.*, "Impact of Reduced Precision in the Reliability of Deep Neural Networks for Object Detection." IEEE, May 2019.

[19] B. Reagen *et al.*, "Ares: A framework for quantifying the resilience of deep neural networks." IEEE, Jun. 2018, pp. 1–6.

[20] S. Kim *et al.*, "Matic: Learning around errors for efficient low-voltage neural network accelerators," in *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2018, pp. 1–6.

[21] S. Motaman *et al.*, "A Perspective on Test Methodologies for Supervised Machine Learning Accelerators," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 3, pp. 562–569, Sep. 2019.
[22] O. Bichler, "N2d2," https://github.com/CEA-LIST/N2D2.
[23] F. F. d. Santos *et al.*, "Analyzing and Increasing the Reliability of Convolutional Neural Networks on GPUs," *IEEE Transactions on Reliability*, vol. 68, no. 2, pp. 663–677, Jun. 2019.
[24] Y. Ibrahim *et al.*, "Soft error resilience of deep residual networks for object recognition," *IEEE Access*, vol. 8, pp. 19 490–19 503, 2020.
[25] J. Wei *et al.*, "Analyzing the impact of soft errors in vgg networks implemented on gpus," *Microelectronics Reliability*, vol. 110, p. 113648, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S002627141930914X
[26] S. Mittal, "A survey on modeling and improving reliability of DNN algorithms and accelerators," *Journal of Systems Architecture*, vol. 104, p. 101689, Mar. 2020.
[27] J. H. Kim and S. M. Reddy, "On the design of fault-tolerant two-dimensional systolic arrays for yield enhancement," *IEEE Transactions on Computers*, vol. 38, no. 4, pp. 515–525, 1989.
[28] B. Salami *et al.*, "On the resilience of RTL NN accelerators: Fault characterization and mitigation," in *IEEE SBAC-PAD*.
[29] W. Li *et al.*, "Soft Error Mitigation for Deep Convolution Neural Network on FPGA Accelerators." IEEE, Aug. 2020, pp. 1–5.