# HarDNN: Fine-Grained Vulnerability Evaluation and Protection for Convolutional Neural Networks

Abdulrahman Mahmoud[1], Siva Kumar Sastry Hari[2], Christopher W. Fletcher[1], Sarita V. Adve[1], Charbel Sakr[1], Naresh Shanbhag[1], Pavlo Molchanov[2], Michael B. Sullivan[2], Timothy Tsai[2], and Stephen W. Keckler[2]

[1]University of Illinois at Urbana-Champaign, [2]NVIDIA Corporation

***Abstract*—As CNNs are increasingly being employed in high performance computing and safety-critical applications, ensuring they are reliable to transient hardware errors is important. Full duplication provides high reliability, but the overheads are prohibitively high for resource constrained systems. Fine-grained resilience evaluation and protection can provide a low-cost solution, but traditional methods for evaluation can be too slow. Traditional approaches use error injections and essentially discard information from experiments that do not corrupt outcomes. In this work, we replace the binary view of errors with a new continuous domain-specific metric based on cross-entropy loss to quantify corruptions, allowing for faster convergence of error analysis. This enables us to scale up to large networks. We study the effectiveness of this method using different error models and also compare with heuristics that aim to predict vulnerability quickly. We show that selective, fine-grained protection of the most vulnerable components of a CNN provides a significantly lower overhead solution than full duplication. Lastly, we present a framework called HarDNN that packages all these solutions for easy application.**

## I. Introduction

Deep learning and convolutional neural networks (CNNs) have seen a recent surge in usage across many application domains such as climate, image, and video analysis in High Performance Computing (HPC) systems and autonomous vehicles and medical devices in safety-critical systems. The design of efficient platforms which accelerate CNN executions are also on the rise, such as dedicated tensor units on GPUs as well as domain-specific accelerators. As CNNs continue to permeate the fabric of everyday life with increasing utilization in HPC and safety-critical applications, it is important that they are resilient to transient hardware errors, also called soft errors.

Studies have shown that hardware errors could have severe unintended consequences unless the system is designed to detect these errors [48], [64]. Many HPC field studies [9], [22], [59] and exascale reports and challenges [3], [4], [19], assert the importance of designing error tolerant systems. The significance of addressing hardware errors extends across diverse applications, such as climate analytics [20], image and video analytics, recommendation systems [11], natural language processing [27], and automotive safety [29]. For example, following a series of unintended acceleration events by Toyota vehicles, a taskforce following up on a NASA investigation showed that, "as little as a single bit flip ... could make a car run out of control." To mitigate such scenarios, hardware in safety-critical systems must fulfill high integrity requirements, such as those outlined in the ISO-26262 standard for functional safety of road vehicles [17].

Processors deployed in exascale and safety-critical systems will employ ECC/parity to protect large storage structures (storing weights and intermediate data). The level of protection offered by that alone without logic protection will likely not be sufficient, particularly as deep learning continues to grow and delivers high performance computational power to many critical application domains. Conventional reliability solutions such as full duplication through hardware or software are still used in practice to ensure high resilience [1], [18], [57]. For example, Tesla's recent Full Self Driving (FSD) system deploys two fully redundant DNN accelerator chips along with accompanying redundant control logic, power, and peripheral packaging on the board for reliability [54], despite the limited power and area constraints of real-time systems. The real-time computational demands of the perception and planning tasks performed by such systems is increasing [38] and paying the high overheads in cost, area, power, and/or performance is highly undesirable. Moreover, the all-or-nothing protection offered by full duplication may result in over-protection and inefficient use of resources.

Lower overhead solutions require understanding resilience of DNNs at a finer granularity, which is challenging and faces many open problems. Challenges as posed in a recent survey of CNN reliability techniques [60] include: (1) defining what an error comprises in CNNs, (2) scaling up reliability analysis to large networks, and (3) leveraging domain specific insights to inform reliability techniques. With the goal of developing a reliability solution that is much lower cost than full duplication, we address these challenges in this paper.

Instead of approaching a CNN as a single, large computational block, we define and explore a CNN's vulnerability characteristics at finer granularities (e.g., feature maps and layers). We hypothesize that not all sub-components of a CNN contribute equally to the overall network vulnerability, and develop methodologies to quantify vulnerability at the finer granularity. Traditional methods for resilience evaluation measure a network's vulnerability in a binary manner (either an output corruption occurred or did not) via error injection experiments and are prohibitively slow. This paper introduces a

new, domain-specific error metric based on cross-entropy loss (called ΔLoss) to quantify corruption, replacing the binary view of silent output corruptions with a continuous metric. Our new domain-specific metric enables scaling resilience evaluation up to larger CNNs – to the best of our knowledge, this paper presents the first large-scale reliability analysis for CNNs. The following are the main contributions of this paper.

- We provide vulnerability formulation for CNN resilience at a fine granularity (at feature map level), which composes error origination probabilities with the probabilities of error propagation to the CNN output.
- We introduce a new method called ΔLoss for fast and accurate CNN component-level vulnerability evaluation. Our results indicate that ΔLoss can provide more accurate results (i.e., with smaller error bars) compared to prior metrics while requiring 6× fewer error injection experiments. This enables us to scale the CNN reliability analysis to large networks.
- We study the effect of using different error injection models based on common quantization methods used for optimizing CNNs for vulnerability [7]. While the total vulnerability of a CNN may vary, the relative contribution of individual CNN components (feature map) remains unchanged.
- We explore multiple heuristics to predict relative vulnerabilities of different components (feature maps) in CNNs, based on techniques from quantization, pruning, and saliency research. While these are fast, the relative vulnerability estimates from error injection-based methods remain significantly more accurate.
- We study the benefits of employing the above methods for selective fine-grain component protection in CNNs for a low-overhead resilience solution. For example, we show how selecting an inherently reliable CNN and selectively protecting just the most vulnerable components can potentially eliminate the overheads that full duplication may incur without the analysis.
- We perform an analysis to understand why errors may propagate through CNNs, and find a very high correlation between an input image with a large margin between the top two class predictions and the probability of a hardware error resulting in an output misclassification.
- We package all these solutions into a framework called HarDNN for easy application on pretrained CNNs. HarDNN enables fine-grained vulnerability evaluation and selective protection to obtain desired levels of resilience improvement with low computational overheads.

## II. Background

A convolutional neural network (CNN) is a class of deep neural networks (DNNs) used to analyze visual imagery such as image recognition or object detection and has recently gained traction for use in many HPC and safety-critical applications [58], [60]. A fundamental computational unit in a CNN is the *neuron* (also known as the activation value), computed as the result of a dot product between a *filter* of weights (typically stored as a 3-D tensor) and an equal-sized portion of the input (also a 3-D tensor). An *output feature map* (or *fmap* for short) is a plane of neurons, obtained by convolving a filter over an input. A single output fmap is a 2-D tensor. Mathematically, a convolution is comprised of many dot products, where each dot product is composed of many multiply-and-accumulate (MAC) operations. Several filters are often convolved over the same input to produce the same number of output fmaps (one filter produces one output fmap), which are organized in a 3-D tensor. This combined operation is referred to as a convolutional *layer*. A non-linear activation function is typically applied element-wise to the output fmaps and is considered part of the layer A CNN consists of a series of convolutional layers followed by some fully-connected layers.

During training, a CNN uses a backpropogation algorithm to modify the filters (consisting of weights) of the network to improve the prediction accuracy. Once trained, a CNN operates in feed-forward mode only to predict the class of an input image, called an *inference*. This paper focuses on CNN inferences, which is used by applications for image analysis.

For classification CNNs, the focus of this paper, the final layer in the network is a *softmax* layer, which provides a probability distribution for each possible class the network is trained to predict. The class with the highest probability (the *Top-1* class) is the chosen prediction of the network during an inference.

**Resilience Analysis:** A hardware error can affect an application's execution in one of three ways: (1) it can have no effect on the outcome (called *masked*), (2) it can cause the application to crash, which can typically be detected by the system [13], [25], [42], [49], [62], or (3) it can corrupt the output without leaving detectable traces (called *silent data corruptions or SDCs)*. SDCs are potentially unacceptable to the end-users. [6], [14], [30], [33], [61].

Reliability analysis techniques used in practice to help uncover SDCs can be categorized as *experimental error injection campaigns* and *analytical error propagation modeling*. An *error injection* emulates a hardware error by perturbing internal program state, and then executing the program to completion to evaluate the effect of the error [6], [14], [30], [33], [61]. Since a program can consist of billions of operations and there are a plurality of errors possible for each operation, error injection is fundamentally statistical in nature. An error injection campaign can take a large amount of time and resources to completely characterize the resilience of an application [12], [34], [61]. Analytical error models attempt to reduce the resource intensity of error injection by estimating the vulnerability of different operations through higher-level models that take into account architecture or domain knowledge [10], [21], [24]. This paper focuses on leveraging domain insights to improve error injection campaigns for analysis of CNN vulnerability, and also compares against multiple analytical models from the literature (Section IV-B).

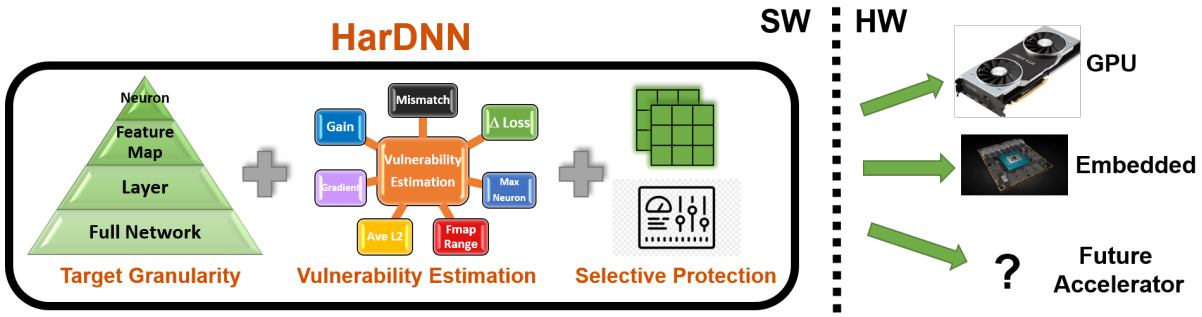



Fig. 1: HarDNN overview. Given a pre-trained model, HarDNN (1) targets feature maps (fmaps) for resilience analysis, (2) estimates the relative vulnerability of each fmap in the CNN, and (3) enables selective protection of the most vulnerable components before model deployment for optimized (real-time) inference.

## III. HarDNN Overview

HarDNN is a software-directed resiliency analysis and hardening framework. It takes as input a pre-trained model, performs fine-grained resilience evaluation, and outputs a fine-grained vulnerability profile which can be used to harden the network before deployment by protecting the most vulnerable components using a selective protection method (e.g., low-cost duplication). HarDNN transforms the original CNN model into a transient-error-resilient model without retraining and without any loss of network accuracy, in the absence of hardware errors. Figure 1 depicts the high-level overview of HarDNN.

In order to effectively analyze and protect a CNN from transient errors, the following three fundamental questions need to be addressed. (1) What granularity (i.e., neuron, feature map, layer, or entire model) should the resilience protection target? (2) Which subset of the components at the target granularity are most vulnerable and need protection? (3) How should the selective protection be implemented?

### A. Target Granularity: Feature Maps

As described in Section II, a CNN consists of many neurons organized in a hierarchical manner to form fmaps and layers. Neuron-level resiliency analysis would provide the most fine-grained control for selective hardening. However, neuron level granularity has a few issues: first, neurons may be *too* fine-grained, with many millions of neurons per CNN (e.g., ResNet50 and VGG19 trained for ImageNet have over 11 and 14 million neurons, respectively). These numbers increase with input size. Evaluating vulnerability of each of the neurons via error injections will be extremely time consuming and inefficient. Additionally, neurons are not immune to all translational effects in input images (e.g., rotation, zoom), making this granularity not very robust for reliability analysis.

Fmaps and layers, on the other hand, are much more tractable in terms of total components (ResNet50 and VGG19 have 26,560 and 5,504 fmaps across 53 and 16 convolutional layers, respectively), and are typically trained to have the same behavior across similar images [43], [55]. In this paper, we focus on quantifying vulnerability and hardening at the fmap level. An additional benefit of targeting this level is that the results can be composed to perform layer- and network-level vulnerability analysis (explored in Section VII. To the best of our knowledge, this is the first work to target fmaps for vulnerability analysis and selective protection with no retraining and no loss in original pre-trained network accuracy in the absence of hardware errors.

### B. Vulnerability Estimation

HarDNN quantifies the vulnerability of each fmap in a CNN due to an error by computing likelihood that an error manifests and propagates to the output and produces an SDC. We compute the likelihood as the product of two components: (1) the probability with which a transient hardware error (caused by a high-energy particle strike, for example) corrupts the output of an fmap (this quantity is called $P_{orig}$) and (2) the probability the fmap-level manifestation propagates to and corrupts the CNN output (called $P_{prop}$). $P_{orig}$ depends on the computations involved in generating the fmap and the low-level hardware implementation of the platform being used. The second component, $P_{prop}$, depends on the input to the CNN, the CNN topology, and the output of a CNN. As described in Section II, the output of a classification CNN is a probability distribution for each class the model is trained to predict. An error that alters the Top-1 class of an inference is typically considered an SDC [8], [60]. For simplicity, we refer to it as a *mismatch*.

$P_{prop}$ can be estimated in one of two ways: using statistical error injections or profiling the network without requiring error injections. Error injection based methods can provide an accurate measure for fmap vulnerability; however, they are traditionally slow due to the large number error injections necessary for statistical guarantees [12], [33], [37]. One of the primary reasons why error injection campaigns are long is due to the fact that SDCs are binary in nature, requiring many observations for statistical convergence. We can exploit the domain-specific knowledge that the outcome of an inference is a probability distribution per class, and leverage this information while performing error injections. To that end, we propose a new method to estimate $P_{prop}$ called $\Delta$Loss, with the primary insight of changing the concept of an error from a

binary view (i.e., mismatch) to a continuous spectrum. ΔLoss is described in Section IV-B.

We also explore six profile based heuristics that predict per fmap $P_{prop}$ using no error injection runs. These heuristics effectively trade-off the accuracy of the vulnerability estimate for even more speed, enabling a quick first-order estimate of fmap vulnerability ranking.

### C. Selective Protection

Once the fmap vulnerabilities are quantified, HarDNN employs selective duplication to harden the computations of the most vulnerable fmaps. Individual fmaps can be duplicated by duplicating the filters that correspond to them. Filter duplication results in two copies of the same logical fmap, where any mismatches between the two copies are used to detect errors during inference and trigger a higher-level system response. The duplicated fmaps need to be dropped before execution of the subsequent layer. The comparison of the two duplicate feature maps can be performed lazily to remove it from the critical path. HarDNN's highly tunable software-directed selective protection approach allows the designer to control the error coverage versus computational overhead trade-off based on the resiliency requirements of the system. We study this trade-off for CNNs in Section V-B.

## IV. Vulnerability Quantification in CNNs

We define the vulnerability of the CNN, $V_{CNN}$, as the probability that the CNN produces an SDC due to a transient hardware error that occurs during inference. This probability can be computed as the sum of vulnerabilities of each of the $N$ fmaps in the CNN.

$$V_{CNN} = \sum_{i}^{N} V_{fmap}[i] \tag{1}$$

We compute $V_{fmap}$ for each fmap $i$ as:

$$V_{fmap}[i] = P_{orig}[i] \times P_{prop}[i] \tag{2}$$

where $P_{orig}$ is the probability with which an error originates in fmap $i$ (corrupting the fmap's output), and $P_{prop}$ is the probability that the error in fmap $i$ propagates to the CNN output and results in an SDC.

HarDNN also estimates *relative vulnerability* of each of the fmaps in the CNN, i.e., the contribution of an fmap towards the total CNN vulnerability. We scale $V_{fmap}[i]$ with $V_{CNN}$ to obtain the relative vulnerability of fmap $i$.

$$Vrel_{fmap}[i] = V_{fmap}[i]/V_{CNN}. \tag{3}$$

### A. Error Origination Probability ($P_{orig}$)

$P_{orig}$ depends on the implementation of the architecture on which the CNN is being run and the computation that generates a feature map (e.g., convolution). Assuming that the major storage structures (e.g., DRAM, caches, and register files) are ECC/parity protected in the target hardware platform [32], most of the errors originate from the unprotected computations. The probability can be computed using the hardware details, the numerical precision of the computation, raw failure rates of the logic and storage structures, and the computation structure.

Given that MAC operations are used to perform a convolution and produce an fmap, we assume that the origination probability is directly proportional to the number of MACs in a convolution, without loss of generality. In this work, we compute $P_{orig}$ as the fraction of the number of MACs used to compute the fmap to the number of MACs in the entire CNN.

To compare vulnerability ($V_{CNN}$) across different networks, the formulation can be extended as the *rate* of origination ($R_{orig}$), focusing on the total number of MAC computations in an fmap and the FIT rate (describing the vulnerability) of the MAC units. We revisit this distinction in more detail in Section V-C, describing how additional information about MAC implementation and precision can be seamlessly incorporated in our analysis.

### B. Error Propagation Probability ($P_{prop}$)

$P_{prop}$ is the fraction of the fmap-level error manifestations that propagate to the CNN output, producing SDCs. This probability depends on values of the fmap outputs, error model, and how the error may propagate through subsequent layers to the output, which depends heavily on the input image. While the true $P_{prop}$ values for fmaps may not be known, we can estimate them using two types of methods: statistical error injection based methods and profiling based methods that do not require error injections.

As mentioned in Section III-B, counting mismatch from error injections may require many observations for statistical convergence. This metric suffers from two issues: (1) It is a binary metric, which means that only error injections that change the Top-1 class can affect the $P_{prop}$ value. Injection experiments where the output values changed but not the Top-1 class, no information about the corruption is recorded. As a result, estimating an accurate SDC percentage requires many injection experiments. (2) The Top-1 mismatch-based SDC metric does not extend naturally to other, non-classification CNNs. This is one of the open problems expressed in the recent survey of DNN reliability [60]. We address these issues with a new method to estimate $P_{prop}$ called ΔLoss.

**Average Delta Cross Entropy Loss (ΔLoss):** With the goal of replacing the binary metric for error propagation (represented by mismatch) with a continuous metric, we propose using the average delta cross entropy loss (ΔLoss) for $P_{prop}$ estimation. Cross entropy loss is traditionally used during DNN training to measure how different the predicted result is from the expected (known) result to improve the prediction accuracy of the network. More generally, it is used in information theory to measure the entropy between two distributions – the true distribution and the estimated distribution. Adapting this metric to reliability, we calculate the absolute difference between the cross entropy loss values observed during an error-free inference and an error-injected inference. This can be expressed as:

TABLE I: Summary of estimation techniques for $P_{prop}$.

**Estimation based on error injections**

| Method Name | Description | Analytical Runtime Estimate |
|---|---|---|
| Mismatch | Top-1 Misclassification rate in fmap due to error | inj/fmap * fmaps * forward |
| ΔLoss | Average delta cross entropy loss of fmap due to error (Equation 4) | inj/fmap * fmaps * forward |

**Estimation based on heuristics, with no error injections**

| Method Name | Description | Analytical Runtime Estimate |
|---|---|---|
| MaxNeuron | Max neuron value observed for fmap | samples * forward |
| FmapRange | Range of neuron values observed for fmap | samples * forward |
| L2 | Average L2-norm value of fmap | samples * forward |
| Gradient | Average magnitude of gradients for fmap | samples * (forward + backward) |
| Gain | Analytical model of Top-1 class change for variation in fmap [50] | samples * (forward + (backward * (classes - 1))) |
| ModGain | Alternative formulation of Gain analytical model (Equation 5) | samples * (forward + (backward * (classes - 1))) |

**Terminology: samples**: # of images. **forward**: Average runtime of a single inference for a CNN. **backward**: Average runtime of a single back-propogation for a CNN. **fmaps**: # of fmaps in CNN. **inj/fmap**: Number of error injection experiments per fmap. **classes**: # of output classes in CNN.

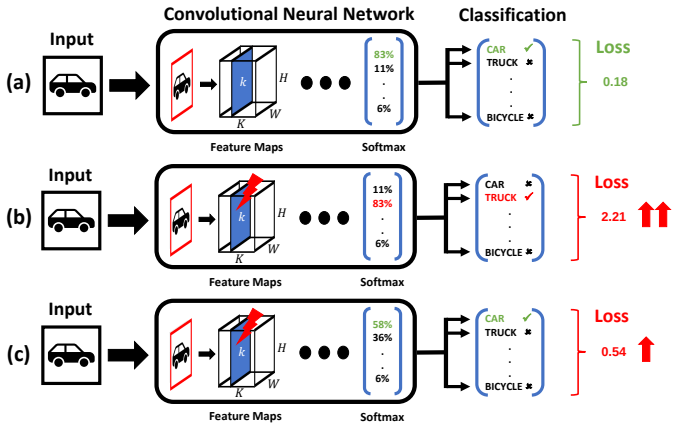



Fig. 2: ΔLoss example where (a) shows an error-free inference classifying the car correctly, (b) shows an example of a mismatch where an error causes the network to select truck instead of car, (c) shows an example where an error causes a drop in confidence for car that does not lead to a mismatch; however, the drop can be captured by the loss value increasing compared to (a).

$$\Delta Loss_{fmap} = \frac{\sum_i^N \mid (\mathcal{L}_{golden} - \mathcal{L}_i) \mid}{N} \tag{4}$$

where $\mathcal{L}_{golden}$ is the cross-entropy loss for an error-free inference, $\mathcal{L}_i$ is the cross-entropy loss for the $i^{th}$ error-injected inference across $N$ total error injections. We use the absolute difference to capture the magnitude of the change in cross entropy loss observed due to an error injection. The larger the $\Delta\text{Loss}_{fmap}$, the higher the $P_{prop}$ of the fmap. Since this method does not predict the SDC percentage, it can be used only to estimate the relative $P_{prop}$. Figure 2 illustrates the advantage of using this method.

**Non-Injection Based Heuristics:** These methods rely on information from a set of error-free inferences to predict relative $P_{prop}$ values. Our non-injection based heuristics fall under two general categories.

*Forward-only:* These heuristics obtain fmap-level information using observations only from the forward pass (i.e., an inference). We explore three metrics and they are briefly described in Table I: MaxNeuron, FmapRange, and average L2 of an fmap. These heuristics are commonly used as first-order approximations of fmap sensitivity in many machine learning studies, such as for choosing filters to prune or to identify ranges for quantization [36], [56]. These techniques are extremely fast, as they depend only on the number of samples explored (i.e., the number of images studied) and the runtime of an inference. Many current deep learning frameworks optimize these operations on a GPU, and we can be leveraged for fast prediction.

*Backprop-based:* These heuristics obtain fmap-level information using observations from both a forward and backward (back-propagation) pass. Backprop provides additional information via differentiation for vulnerability estimation. We study three techniques in this spectrum as well: average gradient values for an fmap, gain (a technique for analytically modeling the expectation of a class change [50]), and a modified formulation of gain which we develop to adapt it for reliability analysis.

Gain utilizes a model of independent noise neuron corruption [50]. We propose to study the effects of *replacing* a neuron by a random scalar belonging to the fmap's dynamic range as discussed later in Section V-C. Under such setup, it can be shown that the Gain approach may be re-purposed to obtain *ModGain*, where the gain of fmap $F$ is:

$$E_F = \mathbf{E}\left[\sum_{i=1, i\neq\hat{y}}^{M} \frac{\sum_{a\in F} a^2 \left|\frac{\partial(z_i - z_{\hat{y}})}{\partial(a)}\right|^2}{|z_i - z_{\hat{y}}|^2}\right] \tag{5}$$

where $F$'s neurons $\{a\}_{a\in F}$ are combined with the $M$ soft outputs $\{z_i\}_{i=1}^{M}$ given a predicted label of $\hat{y}$. Compared to the forward-pass only techniques, the three techniques which leverage backprop will be slower in estimating $P_{prop}$ due to the extra operations (summarized in Column 3 of Table I). However, we can again leverage framework optimizations and expect this to run faster than error injections campaings due to the limited number of total inferences compared to error injection experiments.

TABLE II: CNNs studied with key topological parameters and training accuracy.

| Neural Network | Dataset Name | Convolutional Layers | Total Feature Maps | Total Neurons | Average Neurons/Fmap | Floating Point Top-1 Network Accuracy | INT8 Quantized Top-1 Network Accuracy |
|---|---|---|---|---|---|---|---|
| ResNet50 | ImageNet | 53 | 26,560 | 11,113,984 | 418 | 76.12% | 75.79% |
| MobileNet | ImageNet | 52 | 17,056 | 6,678,112 | 391 | 71.87% | 62.18% |
| VGG19 | ImageNet | 16 | 5,504 | 14,852,096 | 2,698 | 72.36% | 72.20% |
| GoogleNet | ImageNet | 57 | 7,280 | 3,226,160 | 443 | 69.78% | 69.43% |
| ShuffleNet | ImageNet | 56 | 8,090 | 1,950,200 | 241 | 69.35% | 67.01% |
| SqueezeNet | ImageNet | 26 | 3,944 | 2,589,352 | 656 | 58.18% | 57.39% |
| AlexNet | ImageNet | 5 | 1,152 | 484,992 | 421 | 56.52% | 56.04% |
| VGG19 | CIFAR100 | 16 | 5,504 | 303,104 | 55 | 71.94% | 71.89% |
| AlexNet | CIFAR100 | 5 | 1,152 | 10,752 | 9 | 43.87% | 43.82% |
| VGG19 | CIFAR10 | 16 | 5,504 | 303,104 | 55 | 93.34% | 93.38% |
| AlexNet | CIFAR10 | 5 | 1,152 | 10,752 | 9 | 77.24% | 77.21% |

Table I summarizes all methods we explore for estimating relative $P_{prop}$ values.

## V. Methodology

We evaluate HarDNN on 11 CNNs across three datasets. Table II lists each of the CNNs, along with the number of layers, fmaps, neurons, and accuracy on the respective dataset. We use the PyTorch v1.1 framework [41] for evaluation, and obtained pretrained models for CNNs trained on ImageNet [46] from the PyTorch TorchVision repository [44], and CNNs trained on CIFAR10/100 from github [63]. All experiments were run on an Amazon EC2 p3.2xlarge instance [35], which has an Intel Xeon E5-2686 v4 server processor, 64GB of memory, and an NVIDIA V100 GPU with 16GB of memory [39].

For our evaluation, we use the provided test set of each dataset (10,000 images in CIFAR10 and CIFAR100; 50,000 in ImageNet), while removing images which are incorrectly classified by the error-free network since our focus is on analyzing the resilience of the network during correct execution. Unlike prior work, which limits the number of images explored for reliability analysis (just 10 images on larger data sets [8]), we do not impose such limitations. This is not only important for statistical guarantees, but also shows the scalability of our solution.

### A. Evaluating HarDNN's Vulnerability Estimation Methods

We quantitatively evaluate a method's accuracy by sorting the fmaps in descending order of relative vulnerability (Equation 3) and comparing the obtained rankings. Intuitively, this tells us which fmaps are deemed most critical as ranked by a specific method. For mismatch, this results in an ordering of fmaps with the most SDCs descending to the fmaps with least SDCs. Similarly, for all the other techniques (including $\Delta$Loss), we sort the fmaps based on the relative vulnerability computed using their respective relative $P_{prop}$ estimates. We measure the expected error coverage of each method by looking up the number of mismatches associated with the each fmap, and generating a cumulative vulnerability plot based on the fmap ordering of the vulnerability estimation method. This indicates how many errors we protect against when selectively protecting the top N fmaps as ordered by the different methods.

We compare our error injection-based $\Delta$Loss method to the mismatch-based method. To study how the two methods behave as a function of the number of experimental injections performed, we sweep the number of injections per feature map (inj/fmap) from 64 to 12288[1]. We compare each fmap ranking (at every inj/fmap point) to the ranking obtained by mismatch at 12288 (shorthand: mismatch-12288) by measuring the average Manhattan distance between the two rankings. A distance of zero indicates perfect accuracy relative to mismatch-12288. Since different networks have different number of feature maps (see Table II), this resulted in a total of 572 billion error injection experiments across all networks. We parallelized these experiments by batching up to the memory capacity of our evaluation platform, yet these operations took days to complete.

For heuristic evaluation, we split the original test set into two partitions: evaluation set (ES) and test set (TS) using an 80-20 split (as is common in ML studies [47]). We generate a relative fmap ranking for each heuristic by running the heuristic on the ES using the respective $P_{prop}$ estimation, and assess prediction accuracy using the TS to quantify how well the methods predict the relative $P_{prop}$ compared to error-injection based methods.

We measure the runtime of each HarDNN vulnerability estimation method in generating a ranking, and report the measurements on our infrastructure. We also derive the analytical runtime estimates as summarized in Table I.

### B. Error Coverage and Computational Overhead

HarDNN identifies the most vulnerable fmaps (based on a given method) and estimates with high fidelity the expected error coverage and computational overhead by protecting the top N fmaps (where N is a user-selected parameter). For a given set of fmaps, $F$, that are duplicated, we define coverage as the cumulative relative vulnerability of those fmaps. From a developer's point of view, depending on the method used, this coverage is the cumulative vulnerability estimate given by that method on the ES. We refer to this as the *predicted coverage*. The *actual coverage* by protecting $F$, however,

[1]We select 12288 inj/fmap as a "large" number of samples based on our available computational resources, and statistically evaluate this sampling choice in Table III.

could be different. We experimentally measure this coverage by determining the coverage of $F$ on the TS using the error injection-based results. We validate the predicted coverage against the actual coverage to evaluate their similarity.

To target a specific coverage value, we use a greedy algorithm. We sort all fmaps in descending order of vulnerability (based on the metric being considered) and choose the first several fmaps whose relative vulnerability adds up to the targeted coverage. We model the expected computational overhead as the total number of MAC operations in those selected fmaps as a fraction of the total MAC operations in all fmaps. We use MACs as a reasonable proxy to the actual overhead, while providing some abstraction for the actual hardware used. This is a conservative calculation, as some hardware platforms (such as GPUs) can efficiently hide the computational overhead by utilizing spare hardware resources [16], [32]. Our methodology is designed to make HarDNN platform agnostic, providing a portable analysis for CNN reliability on any hardware backend.

We study the coverage vs. expected overhead results to compare the trade-offs offered by all the HarDNN methods, both error injection based methods and the heuristics. We also use it to analyze different CNNs' resilience characteristics.

### C. Error Models

As described in Section II, we assume that a transient error on a flip-flop during a MAC operation of a convolution will corrupt a single neuron's value. Prior work also found that such low-level errors can manifest as single or multiple bit flips [5]. Additionally, highly optimized inference systems typically employ quantization prior to deploying CNNs. Such models run significantly faster with hardware support for reduced-precision operations, which is prevalent in GPUs and CPUs. These benefits come with a small but acceptable loss in classification accuracy (reflected in Table II). Based on these consideration, we evaluate $P_{prop}$ using the following three error models.

In each of these models, an error is injected in a neuron that is randomly chosen from an fmap and substituting the original value with the erroneous value. (1) ***FP-Rand*** represents a random, multi-bit error in a neuron storing a floating-point value. A random value between [-$max$, $max$] is selected as the injected error, where $max$ is the maximum observed neuron value in the fmap across the training set. FP-Rand limits the the error by bounding it between a range. Previous work found that inference is highly sensitive to errors in the sign and exponent bits and a simple output fmap-level range detector can mitigate many of the most severe corruptions [23], which we incorporate in this error model. We use 16-bit floating point (FP16) precision for our evaluation. (2) ***FxP-Rand*** considers 8-bit integer (INT8) *fixed-point quantization* (a commonly used scheme in many commercial products [31], [40]), which quantizes the CNN based on the range of neuron values observed during training. The erroneous value is a randomly selected INT8 value. (3) ***FxP-Flip*** represents a random single bit-flip on a fixed-point quantized 8-bit integer neuron.

TABLE III: Comparing error in ΔLoss and mismatch measurements from error injection experiments with 99% confidence.

| Neural Network | Error at 2,048 inj/fmap | | Error at 12,288 inj/fmap | |
|---|---|---|---|---|
| | Mismatch | ΔLoss | Mismatch | ΔLoss |
| ResNet50 | 380% | 21% | – | – |
| MobileNet | 180% | 15% | – | – |
| VGG19 | 430% | 28% | 184% | 11% |
| GoogleNet | 352% | 21% | 185% | 9% |
| ShuffleNet | 232% | 20% | 160% | 9% |
| SqueezeNet | 344% | 27% | 203% | 11% |
| AlexNet | 197% | 31% | 131% | 12% |

The different computational precisions used for FP-* and FxP-* impact the FIT rate of the MAC operation and the error origination rate, $R_{orig}$, as described in Section IV-A. Two factors affect the FIT rate of a MAC – the number of unprotected bits in the MAC and the logical error propagation probabilities through the multiplier and accumulator implementations. In this paper, we assume $R_{orig}$ for FxP-Rand to be 0.75× the $R_{orig}$ for FP-Rand, primarily based on the reduction in the number of unprotected bits[2]. We assume $R_{orig}$ for FxP-Rand and FxP-Flip are identical.

Unless otherwise noted, our reported results focus on the FxP-Flip error model, which is a commonly used model [8], [23]. Section VII-C evaluates the effect of using different error models for fmap vulnerability analysis.

## VI. Evaluating HarDNN Methods

### A. Mismatch vs. ΔLoss

Mismatch's binary view of an SDC may require many samples (i.e., inj/fmap) for a statistically significant estimate of $P_{prop}$ per fmap (as described in Section IV-B). Table III quantifies this behavior based on our experiments. We report the calculated error bars for the observed (relative) $P_{prop}$ values with 2048 inj/fmap for different CNNs using ImageNet. Results show high error in the relative $P_{prop}$ estimates using the mismatch metric for all the studied CNNs. For example, for GoogleNet the expected error in the measurement is 352% on average at 99% confidence. The error in measurement for the ΔLoss metric is much lower, around 21% on average for GoogleNet. The error reduces as we increase the number of injections per fmap. We performed a larger experiment with 12,288 inj/fmap[3] and found that the error bars decreased by nearly half for each of the two metrics. Overall, Table III shows that ΔLoss can provide more accurate results (based on error bars) with >6× fewer injections compared to mismatch on the studied CNNs.

Empirical measurements obtained from our experiments also validate the above trends. Figure 3 illustrates this using AlexNet across different datasets. As the number of inj/fmap increase, the vulnerability estimates obtained by both mismatch and ΔLoss converge relative to Mismatch-12288, as

[2]Since each MAC unit typically has two input registers (8-bits for INT8 and 16-bits for FP16) and one accumulator (of 32 bits), we use $(8+8+32)/(16+16+32) = 0.75$.

[3]12,288 injections per fmap for ResNet50 and MobileNet with >15,000 fmaps each would require more than one week of computation time per CNN.

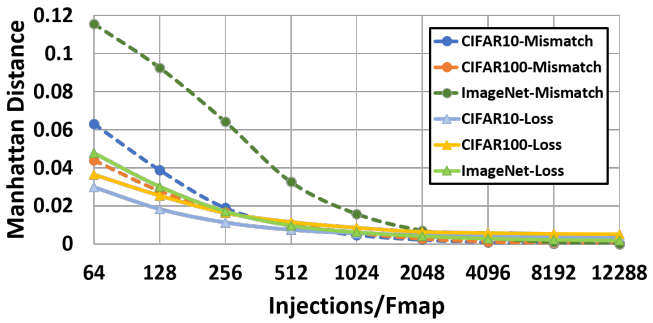



Fig. 3: At large inj/fmap, mismatch and loss converge to provide similar fmap vulnerability estimates (AlexNet). Loss converges faster.

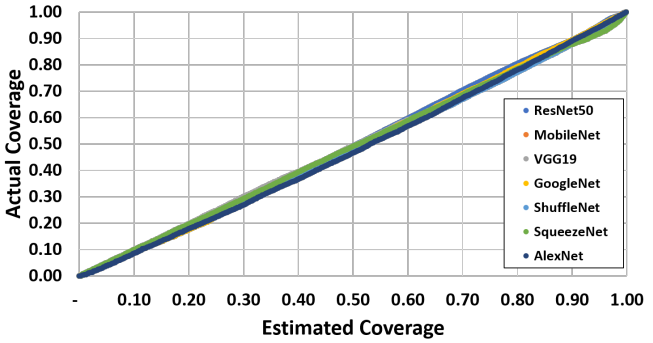



Fig. 4: Validation of predicted coverage vs actual coverage.

described in Section V. These results show that ΔLoss requires *fewer* injections to asymptotically arrive at its relative fmap vulnerability estimates, which indicates that ΔLoss can reach a more accurate relative fmap estimate *faster* than mismatch. The key insight behind this behavior is that ΔLoss can extract information from each error injection (even if it is masked), which the mismatch ignores.

Since the developer may not always have the true relative vulnerability of each fmap, s/he requires an accurate tool to make an informed decision regarding the coverage vs. overhead tradeoff. We compare the predicted coverage by ΔLoss to the actual coverage in Figure 4 (as described in Section V), and we find that, not only is ΔLoss reasonably quick at vulnerability estimation, it is also representative of the actual vulnerability as measured by mismatches in the TS. Thus, the prediction provided to the developer is very accurate, with ΔLoss providing an excellent alternative for error coverage.

### *B. Heuristics*

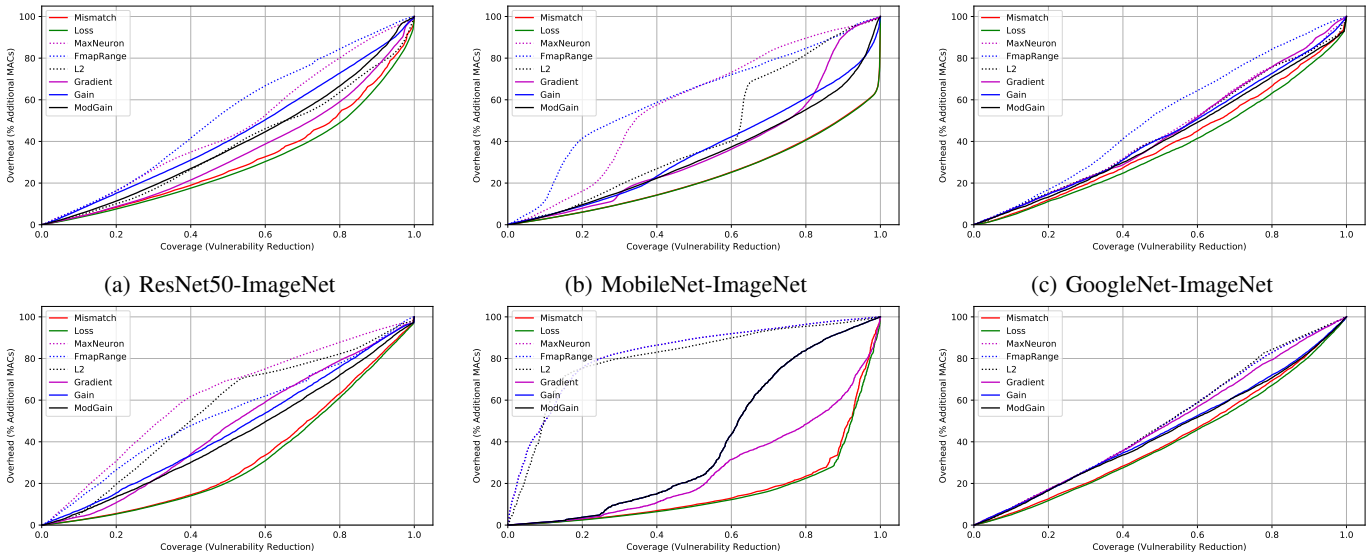
HarDNN provides an optimization platform for selectively protecting feature maps while understanding the associated overhead. We study the trade-offs offered by different HarDNN techniques and use that to evaluate the methods. Figure 5 shows the tradeoffs offered by the eight vulnerability estimation techniques presented in Section IV-B for fmap protection (measured by cumulative error coverage) and the computational overhead (measured by additional MAC operations due to duplication). We show mismatch and ΔLoss cumulative vulnerabilities using 12288 inj/fmap, and compare to ΔLoss ranking as we have shown it has the highest accuracy (among the two) in the previous section. Figure 6 shows the average distance between each vulnerability estimation method, relative to the lowest curve, ΔLoss.

ΔLoss provides the best observed tradeoff for all networks studied, indicated by the lower overhead at higher coverage points in Figure 5, with mismatch also performing nearly as well (1% average overhead difference at any coverage point). The results clearly show that not all feature maps are equally important, which can be leveraged for a sub-linear overhead protection scheme. For example, reducing the vulnerability of the SqueezeNet network by 10× requires only 38% overhead by duplicating just a fraction of the fmaps.

An accurate estimation of fmaps' relative vulnerabilities based on this knapsack formulation is important; otherwise the fmap selection process could yield low coverage with high computational overheads as illustrated by the forward-only heuristics for SqueezeNet (and also for the other networks). The backprop-based heuristics, (Gradient, Gain, and ModGain) perform better overall, ranging between 15% and 17% overhead on average based on Figure 5, compared to the forward pass heuristics (21%-26% average additional overhead as shown in Figure 5). This indicates that the information gathered from the gradients is beneficial for estimating relative vulnerability, but is not sufficient for accurate estimates (as compared to error injection based methods).

Table IV lists the measured runtimes for three networks: a small, medium, and large network (AlexNet, GoogleNet, and ResNet50, respectively). We find that the runtime trends between the analytical models (Table I) and the measured results are similar. In other words, the forward-pass heuristics are the fastest (running in less than a minute on average), followed by the backward-pass (a few minutes to a few hours), followed by error injections (multiple hours to days). Gain and ModGain under-performed due to an implementation limitation, where the evaluation platform did not have support for batching with different differentiation values as required by the gain formulation. If batching were possible, we would have expected Gain and ModGain to perform only slightly slower than Gradient, since its performance depends on the number of classes being predicted by the network model in addition to the backprop operations (Table I).

Overall, we report an interesting tradeoff between the multiple vulnerability estimation methods evaluated. Heuristics can provide a very quick but less accurate ranking of fmaps for cursory evaluation. Of the studied heuristics, ModGain exhibits the lowest variance across network. For the highest accuracy guarantees, the user should perform error injections with ΔLoss, and can select the number of inj/fmap for their desired bounds on error. Although runtime results presented in Table III are for 2048 and 12288 inj/fmap, we evaluated and found that as few as 64 inj/fmap can suffice. For example,




(a) ResNet50-ImageNet (b) MobileNet-ImageNet (c) GoogleNet-ImageNet

(d) ShuffleNet-ImageNet (e) SqueezeNet-ImageNet (f) AlexNet-ImageNet

Fig. 5: Vulnerability reduction versus computational overhead.

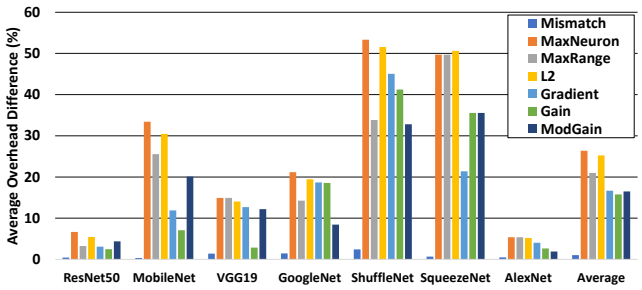
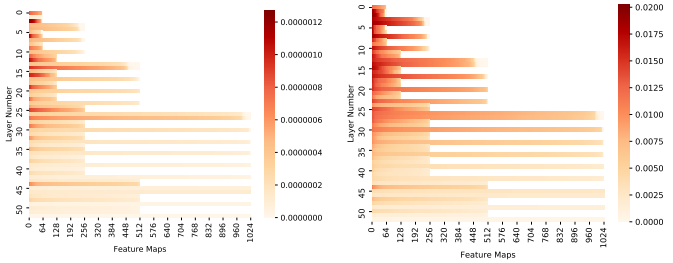



Fig. 6: Additional overhead for HarDNN methods compared to ΔLoss, averaged across different coverage targets.




(a) $V_{fmap}$ ranking (b) $P_{prop}$ ranking

Fig. 7: Layer level vulnerability analysis with ΔLoss on ResNet50-ImageNet (cutoff at 1024 fmaps).

TABLE IV: Measured runtimes in hours for $P_{prop}$ estimation. *Implementation required batch size = 1 for Gain/ModGain.

| Method | AlexNet | GoogleNet | ResNet50 |
|---|---|---|---|
| Mismatch, ΔLoss @ 2048 inj/fmap | 1.25 | 8.20 | 35.50 |
| Mismatch, ΔLoss @ 12288 inj/fmap | 7.45 | 48.82 | - |
| MaxNeuron, FmapRange, Average L2 | 0.01 | 0.01 | 0.02 |
| Gradient | 0.01 | 0.08 | 0.25 |
| Gain* | 0.35 | 3.50 | 15.38 |
| ModGain* | 0.38 | 4.50 | 16.98 |

using ΔLoss can bring the runtime of ResNet50 down by 32× to just over 1 hour, while still observing smaller error bars than mismatch-2048 with 99% confidence (75% for ΔLoss-64 vs. 381% for mismatch-2048).

## VII. CNN Reliability Analysis

### A. Layer Level Analysis

As layers in a CNN consist of many fmaps, we study whether the vulnerable fmaps are clustered in certain layers or not. Figure 7a shows a heatmap of ResNet50's fmap vulnerabilities ($V_{fmap}$), which are computed using ΔLoss. Fmaps per layer are sorted based on $V_{fmap}$ values. The darker the color, the more vulnerable the fmap. We find that on average, a small fraction of fmaps (<0.33) account for a large percentage of a CNN's vulnerability. The figure also show that the highly vulnerable fmaps are distributed across layers. A layer-level analysis and protection would provide an inefficient solution as it will likely protect more fmaps than required to meet the resilience requirements. A fine-grained analysis at the fmap level provided by HarDNN informs which fmaps to target duplication for an optimal solutions.

We study the importance of incorporating $P_{orig}$ in the vulnerability formulation by comparing the $P_{prop}$ and $V_{fmap}$ values. Figure 7b shows the profile of the $P_{prop}$ values. Prior work used similar quantities to determine what and how much to protect [8], [28], [45]. This figure when compared to Figure 7a shows that the fmap and layer vulnerabilities differ, and could lead to protecting significantly different set of fmaps or layers.

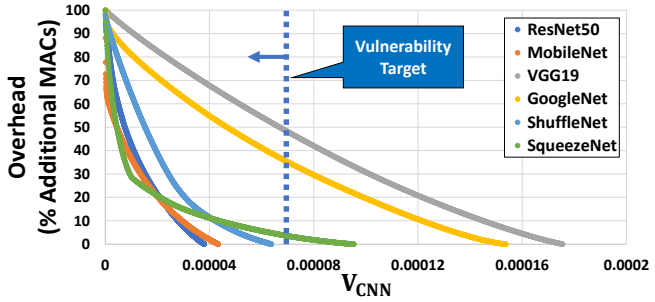



Fig. 8: Network level analysis for ImageNet networks.

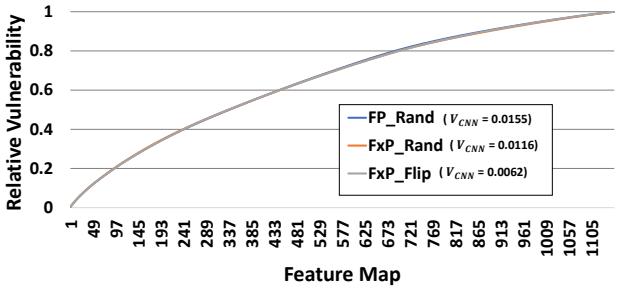



Fig. 9: $Vrel_{fmap}$ is similar across error models, even with different $V_{CNN}$ (AlexNet-ImageNet).

### B. Network Level Analysis

HarDNN further allows a developer to compare total vulnerability values, $V_{CNN}$, of different CNNs, allowing them to make an informed decision about selecting a CNN that meets the resilience, performance, and accuracy targets. Results for six different CNNs trained on the ImageNet dataset to solve the same classification problem are shown in Figure 8. For a given vulnerability target, this analysis informs the developer how many fmaps need protection for a selected CNN and estimates the associated overheads. Since we use $\Delta$Loss to predict $P_{prop}$, the x-axis values are in an arbitrary units, but allow for relative comparison and selection (as validated by Figure 4). A separate small experiment can be performed to calibrate the scale to real probabilities. For example, based on a vulnerability target of 0.00007 (as exemplified in Figure 8), selecting ResNet50 would satisfy the reliability need without additional protection. Selecting SqueezeNet and GoogleNet would require duplicating a fraction of fmaps which would amount to approximately 5% and 35% overheads, respectively.

### C. Error Model Choice and Relative Vulnerability

As discussed in Section V-C, we evaluate the effect of different error models on vulnerability estimation. Figure 9 shows the cumulative relative vulnerability ($Vrel_{fmap}$) of the fmaps in AlexNet-ImageNet, where the x-axis is sorted in descending order of $Vrel_{fmap}$, which are measured using 12,288 inj/fmap and mismatch as the SDC determination criterion. For the comparison, we use the same fmap order on the x-axis, which is obtained based on FxP-Flip $Vrel_{fmap}$.

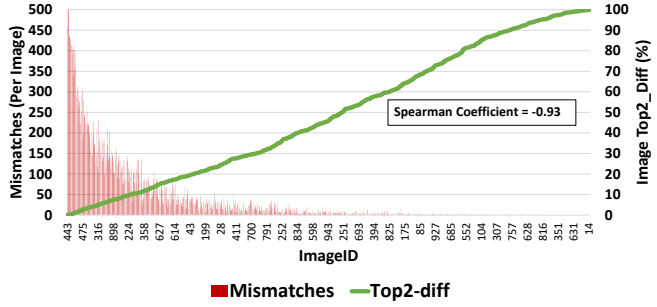



Fig. 10: Top2Diff analysis for AlexNet-CIFAR10. The probability that an input is affected by an error to produce an SDC is inversely correlated with the margin between the top two classes of the error-free inference.

Results show that an fmap's contribution towards the total network vulnerability is practically the same for the different error models, with less than .0001% difference. We, however, found that the absolute total vulnerability, $V_{CNN}$, changes with the models. FP-Rand and FxP-Rand exhibit similar propagation probabilities ($P_{prop}$), as both models have the same dynamic range and multi-bit perturbation error model. However, $V_{CNN}$ is lower for FxP-Rand due to the influence of the numerical precision of the MACs (INT8) on $R_{orig}$ (as explained in Section IV-A). FxP-Flip shows lower $V_{CNN}$, which we attribute to the less egregious error model, i.e., a single-bit perturbation, compared to a random value from the entire range.

### D. Why Do Errors Result in Mismatches

With the aim of understanding why some errors result in a mismatches but others do not, we conducted a separate error injection campaign for AlexNet. We chose 500 random images from the CIFAR10 dataset, and performed 2048 inj/fmap for *each* image (total of 1.2 billion injections). We recorded the probability distribution of each of the 10 classes.

We find a strong correlation between the number of mismatches and an image's error-free classification confidence. We measured the Spearman correlation coefficient between an image's error-free top-1 confidence and the number of observed mismatches, and found it to be -0.87, where -1.0 indicates a perfect inverse relationship. We also find that the smaller the difference between the top two class probabilities (which we call the *top2diff*), the higher the probability of a misclassification happening for the image. Figure 10 shows our results. We also measured the Spearman correlation coefficient between an image's error-free top2diff and the number of mismatches, and found it to be -0.93. If the gap between the probabilities of the top 2 classes is large, the error has to be significant enough to change the class, which explains the strong inverse correlation with the top2diff. Further, this is another reason why Gain/ModGain performed well from the heuristics explored, as they aim to model the probability of an error changing the output to *each* other class.

## VIII. Related Work

Due to the rise of CNN utilization in HPC and safety-critical applications, there is a recent surge in research on CNN resilience. Many proposed methods for CNN reliability require network retraining. This has been performed for redistributing vulnerability across a network [51], [53] and introducing additional components that require finetuning [28]. One of the primary goals of our work is to avoid training altogether due to the high associated costs. Retraining is often not an option for proprietary models as the training recipe and datasets may not be available.

Many works have explored error injection analysis for CNNs [8], [15], [16], [26], [28], [51], [52]. Most of these works, however, have limited studies by either focusing on only a few small networks (using MNIST and CIFAR10 datasets), performing relatively few injection experiments (a few 1000 injections per *network*), or cap the number of images studied. To the best of our knowledge, we perform the first large scale CNN reliability study across many networks and datasets. Further, we introduce a new method, $\Delta$Loss, which opens up several new research directions including allowing other techniques to validate their results for accuracy without limitations of very large error injection campaigns.

Prior work has explored performing selective duplication at finer granularity than full DMR, such as kernel-level duplication in GPUs [16], layer level duplication [26], feature map level duplication [51], and neuron level duplication [28]. Further, recent work has shown that software level TMR hardening can provide similar error protection compared to hardware duplication, while costing less area [26]. In this work, we target fmap-level duplication in software, and evaluate its composition at the layer and network levels. While this work is not the first to target fmaps level granularity, we are the first to evaluate protection of fmaps without retraining and while performing a large scale study and analysis.

Aside from recent reliability-centric analysis, HarDNN is similar to pruning-based ML research [2], [36], in that pruning techniques aim to identify important filters and remove them from the model, while we seek to identify the most vulnerable fmaps in order to protect them from errors. Pruning techniques, however, operate under a different error model, assuming an entire filter is zeroed out, while with HarDNN we focus on single neuron perturbations.

## IX. Conclusion

This paper presents HarDNN, a software-directed technique to identify vulnerable computations in CNNs for selective protection. HarDNN operates at the feature map level granularity, and introduces $\Delta$Loss as an accurate error-injection based method for relative vulnerability estimation, and explores different heuristics for fast vulnerability assessment. Additionally, we analyze the tradeoff between error coverage and computation overhead for selective protection. Results show that the relative vulnerability of an fmap is similar across three error models studied, and that the improvement in resilience for the added computation is super linear with HarDNN. For future work we plan to extend HarDNN to include other applications of neural networks.

## Acknowledgements

This material is based upon work supported in part by the Applications Driving Architectures (ADA) Research Center, a JUMP Center co-sponsored by the Semiconductor Research Corporation (SRC) and Defense Advanced Research Projects Agency (DARPA). A portion of this work was performed while Abdulrahman Mahmoud interned at NVIDIA.

## References

[1] Wendy Bartlett and Lisa Spainhower. Commercial Fault Tolerance: A Tale of Two Systems. *IEEE Transactions on Dependable and Secure Computing (TDSC)*, pages 87–96, January 2004.

[2] Davis Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John Guttag. What is the state of neural network pruning?, arxiv 2020.

[3] Franck Cappello, Geist Al, William Gropp, Sanjay Kale, Bill Kramer, and Marc Snir. Toward Exascale Resilience: 2014 Update. *Supercomput. Front. Innov.: Int. J.*, 2014.

[4] Franck Cappello, Al Geist, Bill Gropp, Laxmikant Kale, Bill Kramer, and Marc Snir. Toward exascale resilience. *Int. J. High Perform. Comput. Appl.*, 23(4):374–388, November 2009.

[5] N. Chandramoorthy, K. Swaminathan, M. Cochet, A. Paidimarri, S. Eldridge, R. V. Joshi, M. M. Ziegler, A. Buyuktosunoglu, and P. Bose. Resilient low voltage accelerators for high energy efficiency. In *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2019.

[6] Chun-Kai Chang, Sangkug Lym, Nicholas Kelly, Michael B Sullivan, and Mattan Erez. Hamartia: A fast and accurate error injection framework. In *Proceedings of the International Conference on Dependable Systems and Networks Workshops (DSN-W)*, pages 101–108. IEEE, 2018.

[7] A. Chatterjee and L. R. Varshney. Towards optimal quantization of neural networks. In *2017 IEEE International Symposium on Information Theory (ISIT)*, pages 1162–1166, June 2017.

[8] Zitao Chen, Guanpeng Li, Karthik Pattabiraman, and Nathan DeBardeleben. Binfi: An efficient fault injector for safety-critical machine learning systems. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '19, 2019.

[9] S. Di, H. Guo, R. Gupta, E. R. Pershey, M. Snir, and F. Cappello. Exploring properties and correlations of fatal events in a large-scale hpc system. *IEEE Transactions on Parallel and Distributed Systems*, 30(2):361–374, 2019.

[10] Shuguang Feng, Shantanu Gupta, Amin Ansari, and Scott Mahlke. Shoestring: Probabilistic soft error reliability on the cheap. In *Proceedings of the the International Symposium on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 385–396, 2010.

[11] Udit Gupta, Samuel Hsia, Vikram Saraph, Xiaodong Wang, Brandon Reagen, Gu-Yeon Wei, Hsien-Hsin S. Lee, David Brooks, and Carole-Jean Wu. Deeprecsys: A system for optimizing end-to-end at-scale neural recommendation inference. *CoRR*, abs/2001.02772, 2020.

[12] Siva Kumar Sastry Hari, Sarita V. Adve, Helia Naeimi, and Pradeep Ramachandran. Relyzer: Exploiting application-level fault equivalence to analyze application resiliency to transient faults. In *Proceedings of the International Symposium on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 123–134, 2012.

[13] Siva Kumar Sastry Hari, Man-Lap Li, Pradeep Ramachandran, Byn Choi, and Sarita V. Adve. mSWAT: Low-cost Hardware Fault Detection and Diagnosis for Multicore Systems. In *Proc. of International Symposium on Microarchitecture (MICRO)*, 2009.

[14] Siva Kumar Sastry Hari, Timothy Tsai, Mark Stephenson, Stephen W Keckler, and Joel Emer. Sassifi: An architecture-level fault injection tool for gpu application resilience evaluation. In *2017 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 249–258. IEEE, 2017.

[15] Sanghyun Hong, Pietro Frigo, Yigitcan Kaya, Cristiano Giuffrida, and Tudor Dumitras. Terminal brain damage: Exposing the graceless degradation in deep neural networks under hardware fault attacks. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 497–514, Santa Clara, CA, August 2019. USENIX Association.

[16] Y. Ibrahim, H. Wang, M. Bai, Z. Liu, J. Wang, Z. Yang, and Z. Chen. Soft error resilience of deep residual networks for object recognition. *IEEE Access*, 2020.

[17] International Organization for Standardization. Road vehicles – Functional safety. https://www.iso.org/standard/43464.html, 2011.

[18] X. Iturbe, B. Venu, E. Ozer, and S. Das. A Triple Core Lock-Step (TCLS) ARM® Cortex®-R5 Processor for Safety-Critical and Ultra-Reliable Applications. In *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshop (DSN-W)*, pages 246–249, June 2016.

[19] Peter Kogge, S. Borkar, Dan Campbell, William Carlson, William Dally, Monty Denneau, Paul Franzon, William Harrod, Jon Hiller, Stephen Keckler, Dean Klein, and Robert Lucas. Exascale computing study: Technology challenges in achieving exascale systems. *Defense Advanced Research Projects Agency Information Processing Techniques Office (DARPA IPTO), Techinal Representative*, 15, 01 2008.

[20] Thorsten Kurth, Sean Treichler, Joshua Romero, Mayur Mudigonda, Nathan Luehr, Everett Phillips, Ankur Mahesh, Michael Matheson, Jack Deslippe, Massimiliano Fatica, Prabhat, and Michael Houston. Exascale deep learning for climate analytics. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*, SC '18. IEEE Press, 2018.

[21] Ignacio Laguna, Martin Schulz, David F Richards, Jon Calhoun, and Luke Olson. Ipas: Intelligent protection against silent output corruption in scientific applications. In *Proceedings of the International Symposium on Code Generation and Optimization (CGO)*, pages 227–238. IEEE, 2016.

[22] S. Levy, K. B. Ferreira, N. DeBardeleben, T. Siddiqua, V. Sridharan, and E. Baseman. Lessons learned from memory errors observed over the lifetime of cielo. In *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 554–565, 2018.

[23] Guanpeng Li, Siva Kumar Sastry Hari, Michael Sullivan, Timothy Tsai, Karthik Pattabiraman, Joel Emer, and Stephen W. Keckler. 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, pages 8:1–8:12, New York, NY, USA, 2017. ACM.

[24] Guanpeng Li, Karthik Pattabiraman, Siva Kumar Sastry Hari, Michael Sullivan, and Timothy Tsai. Modeling soft-error propagation in programs. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN)*, 2018.

[25] Man-Lap Li, Pradeep Ramachandran, Swarup Kumar Sahoo, Sarita V. Adve, Vikram S. Adve, and Yuanyuan Zhou. Understanding the Propagation of Hard Errors to Software and Implications for Resilient Systems Design. In *Proc. of International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2008.

[26] F. Libano, B. Wilson, J. Anderson, M. J. Wirthlin, C. Cazzaniga, C. Frost, and P. Rech. Selective hardening for neural networks in fpgas. *IEEE Transactions on Nuclear Science*, 66(1):216–222, 2019.

[27] J. Liu, M. C. Kurt, and G. Agrawal. A practical approach for handling soft errors in iterative applications. In *2015 IEEE International Conference on Cluster Computing*, pages 158–161, Sep. 2015.

[28] Lanlan Liu and Jia Deng. Dynamic deep neural networks: Optimizing accuracy-efficiency trade-offs by selective execution. *CoRR*, abs/1701.00299, 2017.

[29] A. Lotfi, S. Hukerikar, K. Balasubramanian, P. Racunas, N. Saxena, R. Bramley, and Y. Huang. Resiliency of automotive object detection networks on gpu architectures. In *2019 IEEE International Test Conference (ITC)*, pages 1–9, 2019.

[30] Qining Lu, Mostafa Farahani, Jiesheng Wei, Anna Thomas, and Karthik Pattabiraman. Llfi: An intermediate code-level fault injection tool for hardware faults. In *2015 IEEE International Conference on Software Quality, Reliability and Security*, pages 11–16. IEEE, 2015.

[31] Zhenlin Luo, Andres Felipe Rodriguez Perez, Pallavi G, Gomathi Ramamurthy, Sneha Kola, Evarist M Fomenko, Rajesh Poornachandran, Ling Yan Guo, Karan Puttannaiah, Niveditha Sundaram, and Denis Samoilov. "Accelerate INT8 Inference Performance for Recommender Systems with Intel Deep Learning Boost (Intel DL Boost)". https://software.intel.com/content/www/us/en/develop/articles/accelerate-int8-inference-performance-for-recommender-systems-with-intel-deep-learning.html, Oct 2019.

[32] Abdulrahman Mahmoud, Siva Kumar Sastry Hari, Michael B. Sullivan, Timothy Tsai, and Stephen W. Keckler. Optimizing software-directed instruction replication for gpu error detection. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*, SC '18, 2018.

[33] Abdulrahman Mahmoud, Radha Venkatagiri, Khalique Ahmed, Sasa Misailovic, Darko Marinov, Christopher W. Fletcher, and Sarita V. Adve. Minotaur: Adapting software testing techniques for hardware errors. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '19, pages 1087–1103, New York, NY, USA, 2019. ACM.

[34] Abdulrahman Mahmoud, Radha Venkatagiri, Khalique Ahmed, Sasa Misailovic, Darko Marinov, Christopher W. Fletcher, and Sarita V. Adve. Minotaur: Adapting software testing techniques for hardware errors. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '19, page 1087–1103, New York, NY, USA, 2019. Association for Computing Machinery.

[35] Frederic P. Miller, Agnes F. Vandome, and John McBrewster. *Amazon Web Services*. Alpha Press, 2010.

[36] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient transfer learning. *CoRR*, abs/1611.06440, 2016.

[37] Bin Nie, Lishan Yang, Adwait Jog, and Evgenia Smirni. Fault site pruning for practical reliability analysis of gpgpu applications. In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 749–761, Oct 2018.

[38] NVIDIA. Self-Driving Car Hardware — NVIDIA DRIVE. https://www.nvidia.com/en-us/self-driving-cars/drive-platform/hardware/.

[39] NVIDIA. Nvidia tesla v100 gpu accelerator. Website, 2018.

[40] NVIDIA. "NVIDIA T4 Tensor Core GPU Datasheet". https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/tesla-t4/t4-tensor-core-datasheet-951643.pdf, Mar 2019.

[41] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017.

[42] Karthik Pattabiraman, G. P. Saggese, D. Chen, Z. Kalbarczyk, and R. K. Iyer. Dynamic Derivation of Application-Specific Error Detectors and their Implementation in Hardware. In *Proc. of European Dependable Computing Conference (EDCC)*, 2006.

[43] Luis Perez and Jason Wang. The effectiveness of data augmentation in image classification using deep learning. *CoRR*, abs/1712.04621, 2017.

[44] PyTorch. Pytorch classification models. "https://pytorch.org/docs/stable/torchvision/models.html", 2019.

[45] Brandon Reagen, Udit Gupta, Lillian Pentecost, Paul Whatmough, Sae Kyu Lee, Niamh Mulholland, David Brooks, and Gu-Yeon Wei. Ares: A framework for quantifying the resilience of deep neural networks. ACM, 2018.

[46] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[47] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, USA, 3rd edition, 2009.

[48] Safety Research and Strategies, Inc. Toyota unintended acceleration and the big bowl of 'spaghetti' code. "http://www.safetyresearch.net/blog/articles/toyota-unintended-acceleration-and-big-bowl-%E2%80%9Cspaghetti%E2%80%9D-code", 2013.

[49] Swarup Sahoo, Man-Lap Li, Pradeep Ramchandran, Sarita V. Adve, Vikram Adve, and Yuanyuan Zhou. Using Likely Program Invariants to Detect Hardware Errors. In *Proc. of International Conference on Dependable Systems and Networks (DSN)*, 2008.

[50] Charbel Sakr and Naresh R. Shanbhag. An analytical method to determine minimum per-layer precision of deep neural networks. *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1090–1094, 2018.

[51] C. Schorn, A. Guntoro, and G. Ascheid. Accurate neuron resilience prediction for a flexible reliability management in neural network accelerators. In *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 979–984, March 2018.

[52] Christoph Schorn, Thomas Elsken, Sebastian Vogel, Armin Runge, Andre Guntoro, and Gerd Ascheid. Automated design of error-resilient and hardware-efficient deep neural networks. *ArXiv*, abs/1909.13844, 2019.

[53] Christoph Schorn, Andre Guntoro, and Gerd Ascheid. An efficient bit-flip resilience optimization method for deep neural networks. In *Design, Automation & Test in Europe Conference & Exhibition, DATE 2019, Florence, Italy, March 25-29, 2019*, pages 1507–1512, 2019.

[54] Sean Hollister. Tesla's new self-driving chip is here, and this is your best look yet. https://www.theverge.com/2019/4/22/18511594/tesla-new-self-driving-chip-is-here-and-this-is-your-best-look-yet, 2019.

[55] Connor Shorten and Taghi M. Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6:1–48, 2019.

[56] Hai Shu and Hongtu Zhu. Sensitivity analysis of deep neural networks. Website, 2019.

[57] Alex Shye, Joseph Blomstedt, Tipp Moseley, Vijay Janapa Reddi, and Daniel A. Connors. PLR: A Software Approach to Transient Fault Tolerance for Multicore Architectures. *IEEE Transactions on Dependable and Secure Computing (TDSC)*, 6(2):135–148, April 2009.

[58] V. Sze, Y. Chen, T. Yang, and J. S. Emer. Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE*, 105(12):2295–2329, Dec 2017.

[59] Li Tan and Nathan DeBardeleben. Failure analysis and quantification for contemporary and future supercomputers. *ArXiv*, abs/1911.02118, 2019.

[60] C. Torres-Huitzil and B. Girau. Fault and error tolerance in neural networks: A review. *IEEE Access*, 2017.

[61] Radha Venkatagiri, Abdulrahman Mahmoud, Siva Kumar Sastry Hari, and Sarita V. Adve. Approxilyzer: Towards a Systematic Framework for Instruction-level Approximate Computing and its Application to Hardware Resiliency. In *Proc. of International Symposium on Microarchitecture (MICRO)*, pages 1–14, 2016.

[62] Nicholas J Wang and Sanjay J Patel. ReStore: Symptom-Based Soft Error Detection in Microprocessors. *IEEE Transactions on Dependable and Secure Computing*, 3(3), July-Sept 2006.

[63] Wei Yang. Pytorch-classification. "https://github.com/bearpaw/pytorch-classification", 2017.

[64] Junko Yoshida. Toyota case: Single bit flip that killed. https://www.eetimes.com/document.asp?doc$_i d = 1319903, 2013$.