Optimizing Selective Protection for CNN Resilience

Abdulrahman Mahmoud*, Siva Kumar Sastry Hari†, Christopher W. Fletcher‡, Sarita V. Adve‡, Charbel Sakr†, Naresh Shanbhag‡, Pavlo Molchanov†, Michael B. Sullivan†, Timothy Tsai†, and Stephen W. Keckler†
*Harvard University, †NVIDIA Corporation, ‡University of Illinois at Urbana-Champaign

***Abstract*—As CNNs are being extensively employed in high performance and safety-critical applications that demand high reliability, it is important to ensure that they are resilient to transient hardware errors. Traditional full redundancy solutions provide high error coverage, but the associated overheads are often prohibitively high for resource-constrained systems. In this work, we propose software-directed selective protection techniques to target the most vulnerable work in a CNN, providing a low-cost solution. We propose and evaluate two domain-specific selective protection techniques for CNNs that target different granularities. First, we develop a feature-map level resilience technique (FLR), which identifies and statically protects the most vulnerable feature maps in a CNN. Second, we develop an inference level resilience technique (ILR), which selectively reruns vulnerable inferences by analyzing their output. Third, we show that the combination of both techniques (FILR) is highly efficient, achieving nearly full error coverage (99.78% on average) for quantized inferences via selective protection. Our tunable approach enables developers to evaluate CNN resilience to hardware errors before deployment using MAC operations as overhead for quicker trade-off analysis. For example, targeting 100% error coverage on ResNet50 with FILR requires 20.8% additional MACs, while measurements on a Jetson Xavier GPU shows 4.6% runtime overhead.**

I. Introduction

Recent advances in deep learning and convolutional neural networks (CNNs) have ushered in an era where machine learning heavily influences both the software and hardware landscapes in the areas of high performance computing (HPC) and safety-critical systems. Diverse application domains including video analytics, climate studies, autonomous vehicle systems, and medical devices have all begun heavily relying on CNNs for improved performance and energy efficiency. Dedicated hardware features such as GPU tensor cores and discrete accelerators such as Google's Tensor Processing Units (TPUs) have further fueled this growth. With the increased utilization of CNNs in many safety-critical domains, understanding the implications of transient hardware errors (also called *soft errors*) on CNN outcomes can help guide and direct resiliency in this domain.

HPC and safety-critical system studies indicate that transient hardware errors caused by particle strikes or voltage droops can have severe unintended consequences on an application's output unless the system is designed to detect these errors [6], [7], [12], [28], [32], [61], [75]. This is particularly important for safety critical systems such as autonomous vehicles, where the sheer scale at which these devices are deployed demand extremely low failure rates for individual components. Modern certification standards, such as ISO-26262 [25] for automotive safety, aim to ensure ultra-low failure rates in hardware units; however, achieving low failure rates without the high overheads of dual- or triple-modular redundancy (DMR or TMR) continues to be a research objective [21], [76].

Prior work has shown that corruptions that manifest as large neuron values are major contributors of DNN vulnerability to soft errors [22], [33], [57], [81]. Typical fault-free neuron values have limited range, falling within a small fraction of the total range offered by the floating-point data representation (e.g., FP32). Thus, range checkers are shown to be effective in mitigating large neuron value corruptions during an inference [9], [46]. However, the applicability of the range checkers is limited in state-of-the-art quantized models that use data types with small ranges (e.g., INT8) because the fault-free neuron values span the full range of the data type by design. For safety-critical applications with the requirement of low SDC rates, it is important to identify and mitigate errors that may silently corrupt output for models that use small data types.

In this work, we study software-directed *selective protection* for CNN models employed at finer-granularities for a low overhead solution compared to indiscriminate redundancy. A few research questions need to be addressed for an effective solution: (Q1) At what granularity should the selective protection be performed? (Q2) Which components at this granularity should be selected for protection? (Q3) How should the selective protection be implemented? Answering these questions requires understanding the resilience characteristics of CNNs. Furthermore, by understanding the effect of soft errors on the outcome of CNNs, we can potentially leverage domain-specific knowledge to develop low-overhead reliability solutions and avoid the heavy hammer of DMR or TMR (which are still commonly used in practice [80]).

We introduce two techniques for selective protection in CNNs – *feature-map level resilience* (FLR) and *inference level resilience* (ILR). FLR selectively protects the vulnerable feature maps (*fmaps* for short) of a CNN before deployment for static, "built-in" resiliency. We find that not all fmaps of a CNN have the same vulnerability to soft errors. By identifying and selectively protecting the highly vulnerable fmaps, FLR helps avoid the uninformed and hefty hammer of full duplication by honing protection efforts on the most important sub-components of the network. However, addressing Q2 (*which fmaps to protect?*) requires a typically expensive resiliency analysis of the CNN that involves simulating many error injections and evaluating the outcomes. Traditional approaches measure a binary outcome for an output corruption: either the error caused an output misclassification, or it did not. To accelerate this analysis, we propose a novel, domain specific metric called ΔLoss which converts the binary metric for output corruptions into a continuous metric. We show that ΔLoss speeds up analysis by 3.2× on average (up to 9.4×) by gathering vulnerability information even when an output misclassification does not occur (§III-B). Addressing Q3, FLR protects the vulnerable fmap computations by only duplicating the corresponding vulnerable filters in the network. We show that FLR exhibits a sublinear error coverage versus runtime overhead tradeoff. For example, for the

networks studied, we show that we can obtain 90% error coverage on average with only 62% overhead (as low as 29% for SqueezeNet).

The second novel technique we present targets the granularity of each *inference* a CNN performs. *ILR* is a dynamic method that identifies inferences deemed vulnerable to soft errors and selectively protects them. The key challenge associated with this technique is identifying *which* inferences need protection (Q2) and ensuring that this determination is done quickly online to employ a verification method. To that end, we analyze the vulnerability of inferences based on the outputs of a model and discover a strong correlation between the correct result's confidence and vulnerability. Specifically, we found that the difference between the top two class confidences (called *Top2Diff*) exhibits a strong inverse relationship (Spearman coefficient of -0.93) with the occurrence of an output misclassification due to a soft error. For Q3, we perform a resiliency analysis on CNNs to identify network-specific thresholds for Top2Diff, and subsequently introduce a lightweight logic check after each inference and rerun (for verification) if the Top2Diff is below the threshold. Our results show that we can get 90% error coverage on average with only 17% overhead (as low as 9% overhead for ResNet50).

As described above, the FLR technique duplicates a fraction of computations in all the inferences; the duplication decision is made before the model is deployed. In contrast, ILR duplicates a full inference, but is invoked dynamically based on inference output. In this paper, we analyze the overhead and protection trade-offs offered by the two techniques. We also consider a novel combination of the two where FLR selectively protects fmaps to complement the selective inference protection by ILR for a target error coverage. Our results show that the combined technique, called FILR, can obtain nearly full error coverage (99.78%) with about 48% overhead on average (as low as 21% for ResNet50).

In summary, the contributions of this paper are as follows:

- We introduce a novel, domain-specific resiliency analysis metric called ΔLoss, which can significantly accelerate error injection campaigns to identify vulnerable feature maps in a CNN by 3.2$\times$ on average. We use ΔLoss for feature-map level resilience (FLR) in a CNN.
- We discover and exploit a strong correlation between the output confidence of an inference and probability of an SDC. Leveraging this relationship, we introduce an inference-level resilience (ILR) technique for CNNs.
- We find that combining our domain specific insights and two techniques of FLR and ILR is better than the sum of its parts. Our combination, FILR, can obtain 99.78% error coverage with just 48% overhead on average (and as low as 21% overhead for ResNet50), using number of additional MACs as a proxy.
- Our implementation on a Jetson Xavier GPU with batch size = 1 shows that FILR incurs 1.6% runtime overhead on average.
- We perform an analysis of error-propagation in CNNs at different levels (network, layer, and fmap) and also explore various error models. Our analysis provides multiple insights into the robustness of CNNs to soft errors.

II. Background and Related Work

CNN Background: CNNs are a class of deep neural networks (DNNs) used to analyze visual imagery such as for image recognition or object detection. A classification CNN takes as input an image which propagates through many computational layers until it arrives at a *softmax* layer. The softmax provides a probability for each class the network aims to predict, indicating the confidence of predicting a specific class. The class with the highest confidence (the *Top-1 confidence*) indicates the CNN's prediction for the image. During training, a cost function (such as the cross entropy loss) is computed from the softmax and backpropogated to update weight values to improve prediction accuracy. At deployment time, a classification CNN operates in feed-forward mode only to perform an *inference*.

A CNN is composed of various layers between the input and the output. The predominant layer type is the convolutional (conv) layer, which typically constitutes 90%–97% of a CNNs total computations [36]. A *neuron* (or activation value) is the fundamental component of a conv layer, computed as a dot product between a filter of weights and an equal-sized portion of the input. Each dot product is composed of many multiply-and-accumulate (MAC) operations. A plane of neurons is known as a feature map (*fmap*). Each conv layer in the network may have a different number of filters, which map one-to-one with the number of output fmaps for that layer.

In this work, we target pretrained CNN models for resiliency hardening, and avoid model retraining. In practice, retraining may not even be an option for proprietary models and datasets.

Quantization and Resilience: In state-of-the-art systems, once a model is trained, the neuron and weight values are mapped to a smaller value range (e.g., from FP32 to INT8) in a process called *quantization*. The use of smaller data types for storage and computation increases energy efficiency and performance by multiple folds [14], [59], [70], [78]. Research has shown that these benefits can be achieved with limited loss in model accuracy. Thus, many recent inference-targeted devices support INT8/INT4 formats [27], [48].

The *dynamic range* offered by the number system directly impacts resilience. Research has shown that soft errors which manifest as values significantly higher than the expected range can cause egregious inference output corruptions [33]. As the fault-free neuron values use a limited part of the total range offered by the non-quantized data type (e.g., FP32), low-cost range detectors were proposed to detect SDC-causing soft errors [9]. They have been shown to be effective for inferences that use data types with large value ranges.

INT8 quantization provides a natural range limiter [11], [26], [49], [63], [64] because no values (including errors) can escape beyond the quantized realm, which map to a limited real number range by design. Employing additional range detection for models that use INT8 is challenging since most neuron values use the full range offered by INT8. Since errors within the expected dynamic range of neurons (post quantization) can still cause SDCs, we focus on detecting them. Such errors form the baseline for this study.

DNN Resilience and Error Model: Processors deployed in safety-critical systems typically employ ECC/parity to protect large storage structures such as those storing CNN weights and intermediate data [43]. However, the level of protection offered by that alone without logic protection is not sufficient [21], [25], particularly as deep learning continues to become more computationally intensive. In fact, today's systems may employ fully redundant computations in the form of temporal or spatial DMR

for resilience, such as Tesla's Fully Self Driving (FSD) chip [74].

In this work, we focus on understanding the resilience of CNNs in the context of *transient computational errors* at inference time. Specifically, we employ a *single-bit flip* error model in activation values (i.e., *neurons*), a commonly used abstraction for modeling hardware faults [5], [16], [18], [33], [34]. A transient error could either (1) have no effect on a programs output (called *masked*), (2) get detected by the system using low-cost techniques [35], [55], [62], [79], or (3) remain undetected and corrupt the program output (a silent data corruption or SDC). In the context of CNNs, we only consider transient errors which alter the originally correct Top-1 class of an inference as an SDC [10], [76]; we refer to this as a classification mismatch, or simply *mismatch*.

Resiliency analysis techniques used to uncover SDCs can be categorized as *experimental error injection campaigns* or *analytical error propagation models*. 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 [8], [19], [39], [44], [77]. Since a program can consist of trillions of operations and there are a plurality of errors possible for each operation, an error injection campaign can take significant time and resources to completely characterize the resilience of an application [10], [17], [24], [37], [38], [44]. However, most resiliency studies either focus on small networks (using MNIST or CIFAR10 datasets), use a handful of input images, or perform few injection experiments (e.g., 1000 per *network*) which cannot provide data with enough granularity for effective selective protection. Analytical error models attempt to reduce the resource intensity of error injection campaigns by estimating the vulnerability of different operations through higher-level models, taking into account architecture or domain knowledge [13], [30], [34]. However, this approach is often less accurate as it is heuristic-based. This paper primarily uses experimental error injections to analyze the resiliency of CNNs. In §III, we show how to leverage CNN domain knowledge to accelerate error injection significantly without sacrificing analysis accuracy.

Related work on selective duplication: Prior work has explored performing selective duplication in hardware and software at finer granularities. These include kernel-level duplication in GPUs [24], [37], layer duplication [37], fmap duplication [66], and neuron duplication [38]. While this work is not the first to target fmaps, we are the first to evaluate fmaps without requiring retraining. Prior methods with retraining either redistribute vulnerability across a network [66], [67] or introducing additional components that require fine-tuning [38]. One goal of our work is to avoid training altogether due to its high associated costs and sometimes proprietary nature. Additionally, we expand our analysis to the layer and network levels for a comprehensive evaluation (§VIII).

III. FLR Design Overview

Quantifying the vulnerability at finer granularities can help avoid full network duplication by enabling selective protection of the most vulnerable components only, in contrast to traditional DMR techniques. This section introduces a resiliency analysis and hardening technique called *feature-map level resilience* (FLR). Given a pretrained network, FLR targets the computational component of fmaps for fine grained analysis and protection, quantitatively estimates the vulnerability of each fmap using a new, domain-specific metric called ΔLoss, then selectively protects the most vulnerable fmaps via filter duplication. FLR is a software-driven technique, enabling a flexible analysis which can subsequently be deployed on various hardware platform backends. Fig. 1 shows an overview of FLR.

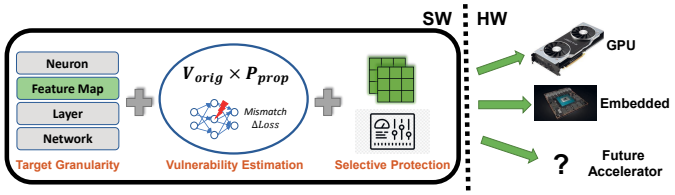


Fig. 1: FLR design overview. Given a pretrained network, FLR (1) targets fmaps for selective analysis and protection, (2) estimates vulnerability of each fmap, and (3) selectively protects the most vulnerable fmaps in software before deployment.

A. FLR Target Granularity

Of the three CNN sub-components (i.e., neuron, fmap, layer), we target feature maps as the sweet-spot for resiliency analysis and hardening for various reasons. First, neuron-level analysis may be *too* fine-grained, with many millions of neurons per CNN (see Table I). Evaluating vulnerability of each neuron via error injection is extremely time consuming. Additionally, neurons are not immune to all translational effects in input images (e.g., rotation, zoom), making this granularity less robust for reliability analysis.

Fmaps and layers, on the other hand, are much more tractable in terms of total components, and are typically trained to exhibit the same behavior across similar images [56], [68]. Performing fmap analysis has the additional benefit that the results can be composed to perform layer- and network-level vulnerability analysis. 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 pretrained network accuracy in the absence of hardware errors.

B. FLR Vulnerability Estimation

FLR quantifies the vulnerability of each fmap in a CNN due to an error by computing the 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 *origination vulnerability* (V_{orig}), which captures the likelihood a transient hardware error corrupts the output of an fmap, and (2) the *propagation probability* (P_{prop}), which is the probability the fmap-level manifestation propagates to and corrupts the CNN output. We compute the vulnerability, V_{fmap}, of each fmap i as:

$$V_{fmap}[i] = V_{orig}[i] \times P_{prop}[i] \quad (1)$$

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 vulnerability can be computed as the sum of vulnerabilities of each of the N fmaps in the CNN as:

$$V_{CNN} = \sum_{i}^{N} V_{fmap}[i] \quad (2)$$

Using Equations 1 and 2, FLR measures the *relative vulnerability*, $Vrel_{fmap}$, of each fmap in the CNN. Intuitively, $Vrel_{fmap}$ is the

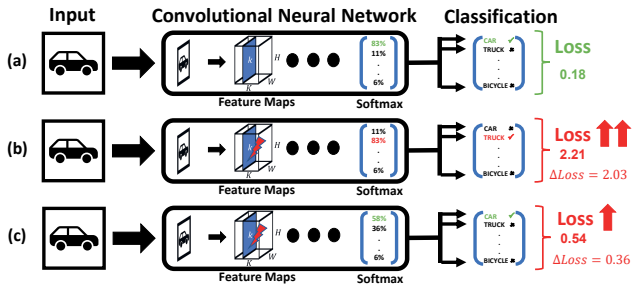


Fig. 2: ΔLoss example where (a) shows an error-free inference classifying the car correctly, (b) shows an example of a mismatch where an error causes the network to select truck instead of car, (c) shows an example where an error causes a drop in confidence for car that does not lead to a mismatch; however,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

contribution of an fmap towards the total CNN vulnerability. This quantity is used to identify the most vulnerable fmaps for protection.

Error Origination Vulnerability: V_{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., convolutions). Assuming that the major storage structures (e.g., DRAM, caches, and register files) are ECC/parity protected in the target hardware platform [43], most of the errors originate from the unprotected computations. V_{orig} 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.

As MAC operations are used for convolutions and generating fmaps, we assume V_{orig} is directly proportional to the number of MACs in a convolution, without loss of generality [72]. In this work, we compute V_{orig} for an fmap as the fraction of the number of MACs used to compute the fmap to the number of MACs in the entire CNN. Our formulation can be extended to compare V_{CNN} *across* different networks, in which case V_{orig} should not be normalized. A network-level vulnerability assessment would be based on the total number of computations performed by different networks (§VIII).

Error Propagation Probability: P_{prop} is the fraction of the fmap-level error manifestations that propagate to the CNN output, producing SDCs. While the true P_{prop} values for fmaps may not be known, we can estimate them for vulnerability ordering using statistical error injections.

Number of Mismatches: As mentioned in §II, counting mismatch from error injections may require many observations for statistical convergence. This metric (although commonly used for its accuracy), 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} measurements. Injection experiments where the softmax changes but not the Top-1 class are not captured by this metric. As a result, estimating an accurate SDC probability 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 resiliency [76]. We address these issues with a new metric to estimate P_{prop} called $\Delta Loss$.

Average Delta Cross Entropy Loss ($\Delta Loss$): Cross entropy loss is traditionally used during CNN 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 resiliency, we can calculate the average absolute difference between the cross entropy loss values observed during an error-free inference and an error-injected inference. This can be expressed as:

$$\Delta Loss_{fmap} = \frac{\sum_i^N |(\mathcal{L}_{golden} - \mathcal{L}_i)|}{N} \quad (3)$$

where $\mathcal{L}_{golden}$ is the cross-entropy loss for an error-free inference and $\mathcal{L}_i$ 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 average ΔLoss_{fmap}, the more vulnerable 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 new metric.

C. FLR Selective Protection

Once the fmap vulnerabilities are quantified, FLR selects the most vulnerable fmaps to harden them from SDCs. Individual fmap computations can be protected by duplicating the filters that correspond to them within a convolution operation. 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. Furthermore, as FLR is a highly tunable software-directed selective protection approach, the designer can selectively control the error coverage versus computational overhead trade-off based on the resiliency requirements of the system (§VII).

Besides selective filter duplication and comparison, other error detection methods can also be used by FLR. For example, kernel duplication [2], [3], feature approximations [52], algorithmic-based error detection (ABED) [18], [51], and AN-codes [15] can be used by FLR after fmap vulnerability estimation. On an error detection, recovery mechanisms such as inference rerun or zero-value propagation [53] or triple modular redundancy can be employed to maintain forward progress.

IV. ILR Design Overview

The second granularity we target for CNN resilience is an individual *inference* for an image (§IV-A). In this section, we introduce ILR, a novel, per-image inference confidence-based CNN resiliency technique. ILR selectively reruns images for inferences that are vulnerable to SDCs by using only information provided after an inference is complete, namely the confidences in the softmax. We study two confidence-based criteria, Top1Conf and Top2Diff (§IV-B), and identify a confidence threshold to trigger reruns during deployment (§IV-C).

A. ILR Target Granularity

The target granularity for CNN resiliency chosen for ILR is an individual *inference*. While FLR targets static, structural duplication of select fmaps before network deployment, ILR uses dynamic information to perform selective, full network reruns. The

motivational insight behind ILR is that the classification confidence of a CNN for an inference is related to the probability that a soft error can cause a classification mismatch. Furthermore, despite their importance, SDCs should be an exception and not the norm; thus, to avoid incurring static overheads to have high resilience, a dynamic anomaly detector can significantly reduce overheads while maintaining high error coverage.

B. ILR Inference Vulnerability

To selectively identify which inferences are vulnerable and need protection, we explore two decision functions which operate on the softmax. The first function assesses vulnerability of an inference based on the highest confidence value observed from the softmax (the Top1-Conf). We select this metric to examine if an inference with high confidence in prediction is more robust to soft errors. In this scenario, if the Top1-Conf lies above a certain threshold, the inference is deemed less vulnerable to perturbations, while inferences with Top1-Conf below the threshold should be conservatively rerun to avoid a possible SDC.

The second criterion we explore is the difference between the top two classes in the softmax, called Top2Diff. The intuition behind this choice is that a transient error needs only do enough computational damage to the network to cause the CNN to classify the image as the second highest class, rather than the (originally correct) top class. Thus, a smaller Top2Diff is akin to a smaller catalyst for the soft error to overcome to cause a mismatch, compared to a large Top2Diff which is more robust to mismatches from soft errors. For both decision functions studied, we perform resiliency analysis using error injections to identify the operational threshold for a target error coverage.

C. ILR Selective Protection

ILR protects against SDCs by running the vulnerable inferences (whose output confidence is below a threshold) again and verifying the output is same (or similar) between the two runs. For the second run of the inference, a highly optimized (pruned and quantized) model can be used to reduce the runtime overhead. In this paper, we consider the overhead to determine which inference to verify via a rerun to be negligible and focus on the reexecution overhead incurred as a result of ILR. The ILR method is attractive due to its simplicity in implementation, and it can be performed either in hardware or software.

V. FILR Resiliency: ILR + FLR

ILR and FLR can be employed independently for CNN resiliency, as each targets a different axis for selective resiliency analysis and hardening. In this work, we also explore a combination of the two techniques, to evaluate the benefits of dynamic, selective inference duplication with static, selective fmap duplication.

To combine the two techniques, we first performs ILR analysis to identify the coverage and overhead of different operational thresholds. We then run FLR analysis *only on the subset of inferences not protected by ILR* at a given threshold, and select the optimal combination between ILR threshold and FLR fmaps for protection. Effectively, this enables FLR analysis (which results in a flat, built-in resiliency overhead by duplicating fmaps before deployment) on the SDCs which ILR does not protect against.

TABLE I: CNNs studied with key topological parameters.

Neural Network	Conv Layers	Total Fmaps	Total Neurons	FP32 Accuracy	INT8 Accuracy
AlexNet [29]	5	1,152	484,992	56.52%	56.04%
GoogleNet [73]	57	7,280	3,226,160	69.78%	69.43%
MobileNet [65]	52	17,056	6,678,112	71.87%	62.18%
ShuffleNet [40]	56	8,090	1,950,200	69.35%	67.01%
SqueezeNet [23]	26	3,944	2,589,352	58.18%	57.39%
ResNet50 [20]	53	26,560	11,113,984	76.12%	75.79%
VGG19 [69]	16	5,504	14,852,096	72.36%	72.20%

VI. Evaluation Methodology

We perform our evaluation on 7 CNNs pre-trained on the ImageNet dataset [60], each listed in Table I with a count of topological parameters. We apply INT8 neuron quantization during inference [63], [64], since highly optimized systems typically employ quantization prior to deploying CNNs [11], [26], [49]. Such models run significantly faster with hardware support for reduced-precision operations, which is prevalent in CPUs, GPUs, and accelerators.

We use the PyTorch framework v1.1 [54], and obtain pretrained models for CNNs from the PyTorch TorchVision repository [58]. We use PyTorchFI [41] to perform error injections on the CNNs. All experiments are run on an Amazon EC2 p3.2xlarge instance [45], with an Intel Xeon E5-2686 v4 processor, 64GB of system memory, and an NVIDIA V100 GPU with 16GB of device memory [48]. Our evaluation focuses on a transient, single bit-flip error model (§II). In §VIII, we extend our analysis to a total of three error models, evaluating the impact on fmap and network vulnerability.

A. Analysis Set (AS) and Deployment Set (DS)

ImageNet [60] provides a test set of 50,000 images, which we randomly split into an analysis set (AS) and a deployment set (DS) using an 80/20 ratio for evaluation. Since this work focuses on pretrained networks, we do not use the images from the ImageNet training set as they would already have been used for training. As in a real-life scenario, we assume the developer only has access to the AS for reliability analysis of the CNNs, and we validate our results on the DS. The 40,000 images in the AS are the same across all networks and techniques explored, and similarly for the 10,000 DS images.

During reliability analysis, we are primarily interested in identifying corruptions that change a correct prediction (based on the ground truth) into an incorrect prediction. For an inference that is originally incorrect, the effect of a hardware error can convert the inference to a correct or another incorrect prediction; analyzing both is not relevant for this work as the focus is not on improving model accuracy or analyzing the handling of different incorrect inferences. Thus, for error coverage analysis, we perform error injections only on images which are *originally correct* (i.e., the inference resulted in the same class as the dataset label) during an error-free execution (similar to prior work [10], [33], [66]). When measuring runtime overhead, however, we include *all* images for evaluation because at runtime we do not know apriori which input will give the correct outcome.

While the AS remains the same throughout analysis, the resiliency analysis methodology differs for FLR and ILR since they are different techniques (elaborated in §VI-B and §VI-C). For validation, however, we perform a single, unified error injection campaign on the DS. For the DS, we perform 10 million random error injections per network, where each error injection is performed on a single ran-

dom bit of a random neuron for a random image. In total, we perform 70 million error injection experiments across all networks for the DS.

B. FLR Evaluation Methodology

We partition the evaluation of FLR into two parts: 1) comparing the accuracy and speed of the two metrics, mismatch and ΔLoss; 2) evaluating the coverage versus overhead tradeoff provided by FLR's selective fmap duplication. As mentioned previously in §II, heuristics can also be used for resiliency analysis, and we explored six heuristics from the literature including value-based techniques [72], pruning techniques [4], [47], and gradient-based techniques [63], [64]. An extensive analysis of heuristic evaluation is included in [42]. We found, however, that none had significantly high accuracy relative to our mismatch-based error injection analysis (the golden standard). Thus, for space considerations, we focus only on error injection evaluation in this paper.

To compare mismatch and ΔLoss, we first generate a statistical oracle for fmap vulnerability by performing 12,288 injections per fmap (inj/famp) for each network, which corresponds to at least 99% confidence level with less than 0.23% confidence intervals.[1]. We define our statistical oracle using the number of mismatches obtained at 12,288 inj/fmap (shorthand: Mismatch-12288). For each individual error injection experiment, we flip a random bit of a random neuron in the fmap for a random image. In total, we performed a total of 855 million unique error injections across all CNNs studied for FLR. The large campaigns help statistically validate the effectiveness of our ΔLoss metric for vulnerability analysis and reducing the campaign runtimes.

We generate a cumulative vulnerability distribution based on a greedy selection algorithm for which fmap order to protect. Fmaps are selected for protection by first sorting all fmaps in descending order of vulnerability (based on the metric being considered) and subsequently choosing the first several fmaps whose relative vulnerability adds up to the targeted coverage. The error coverage is always extracted from the oracle mismatches of each fmap. 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 model runtime overhead with the increase in MACs (as MACs are commonly used for performance evaluations [72]), providing a platform-agnostic metric to estimate and compare overheads.

The precise overhead of our technique will be platform-specific, depending on many factors such as: the type of computational resource (CPU, GPU, ASIC), hardware optimizations (availability of tensor cores), device memory bandwidth/compute ratio, version of a backend library used (e.g., cuDNN), CNN model precision, and batch size [50]. We explore one such platform configuration in our evaluation to simulate an embedded device on a safety critical system: an NVIDIA Jetson Xavier with an 8-core ARM v8.2 CPU, 512-core Volta GPU with Tensor Cores, 32 GB of shared memory running with JetPack v4.4, CUDA v10.1, cuDNN v8.0, and batch size of 1. Real-time power-constrained systems typically employ small low-power devices, as well as a small batch sizes since waiting for multiple frames to create a batch may not meet real-time constraints [50]. The additional duplicate fmaps are implemented as additional filters in the layer (as described in §III-C). We performed 1000 runs and measured the average runtime overhead relative to an unhardened model.

To compare the accuracy of the metrics, we measure the average Manhattan distance between each cumulative distribution and the oracle cumulative distribution. We perform a sweep from 64 inj/fmap to 12,288 inj/fmap for each metric, using the Manhattan distance as a measure for how similar the vulnerability estimations are (zero Manhattan distance implies same vulnerability estimations). To compare the speed of each metric in performing the FLR vulnerability analysis, we identify the number of inj/fmap required to attain 99% similarity to the oracle. By ensuring that all infrastructure is held constant during error injection experiments (i.e., the hardware used, the number of images in a batch for parallelizing error injections, and the runtime of an inference), the only differentiating factor for analysis runtime is the total number of error injections performed, which we use to calculate speedup. Finally, we analyze the coverage versus overhead tradeoff for different networks, and show that the expected coverage (as indicated by the AS) is very accurate compared to the actual coverage (as indicated by the DS) (§VII-A).

C. ILR Evaluation Methodology

For ILR evaluation, we perform 1000 error injection experiments per image in the AS for each CNN studied. For each error injection experiment, we flip a random bit of a random neuron in the network at runtime. In total, we perform 184 million unique error injections across all networks for ILR. Our experiments corresponds to a 99% statistical confidence level with less than 0.81% confidence intervals, which we validate on the DS and show very high accuracy (§VII-B).

We evaluate the two logical conditionals for ILR, Top1Conf and Top2Diff, sweeping the threshold values from 0.0 to 1.0 in increments of 0.01, and measuring the provided error coverage and associated overhead. The error coverage indicates how many SDCs are protected against the given Top1Conf/Top2Diff threshold for rerun, while the overhead is the additional number of inferences performed due to ILR reruns.

D. FILR Evaluation Methodology

We evaluate FILR using the same error injection infrastructure as ILR described in §VI-C. For the ILR component of FILR, we select Top2Diff as the decision criterion. Using the AS and given a target coverage, we run ILR analysis to generate different threshold values which provide less coverage than the target. We then run FLR on the subset of inferences not covered by ILR at each threshold value (i.e., SDCs which ILR does not capture at a given threshold), and selectively duplicate the most vulnerable fmaps which bridge the gap to the target coverage. The fmaps selected by FLR will always be duplicated in the model to provide "built-in" redundancy for each inference, and ILR will selectively run individual inferences based on the defined threshold. Thus, FILR overhead is composed of both these components, which we optimize by identifying the right balance between FLR and ILR. We report results at a target coverage of 100% on the AS, and validate the results by measuring the coverage and overhead on the DS.

[1]We use a discrete Bernoulli distribution to compute our confidence intervals for the oracle, using the measured observations of our error rates with the population size of possible errors to compute confidence intervals [31]. Further, we select ∼12,000 inj/fmap as a large number of samples based on our available computational resources for 99% confidence level [71].

VII. Results

A. FLR Results and Analysis

Mismatch versus ΔLoss Convergence: We begin by analyzing the two metrics used to quantify fmap vulnerability. Figure 3 provides empirical evidence for the convergence of Mismatch-based analysis and ΔLoss-based analysis as the number of inj/fmap increases. The X-axis in the figure shows the number of inj/fmap used for each analysis, and the Y-axis shows the Manhattan distance between the vulnerability ranking of fmaps obtained at each point relative to the statistical oracle (Mismatch-12288). Our first observation is the scale on the Y-axis, which indicates that even at 64 inj/fmap, ΔLoss differs from the Oracle by less than 7% on average. Second, the results show that ΔLoss quickly asymptotes to its final ordering of fmaps, and it does so sooner than Mismatch. Table II lists the number of inj/fmap required for Mismatch and ΔLoss to arrive within 1% of the Oracle. For Mismatch, the number of inj/fmap ranges from 640-5632 while for ΔLoss it is lower going from 128-1536 inj/fmap. Additionally, both Mismatch and ΔLoss converge without requiring a full 12288 inj/fmap. To attain a very high accuracy fmap vulnerability ordering, our results show that Mismatch requires 5.2× fewer inj/fmap on average than the Oracle, while ΔLoss requires 16.7× fewer inj/fmap on average, resulting in an average speedup for ΔLoss of 3.2× over Mismatch (up to 9.4×) .

One exception is VGG19, which asymptotically approaches the 97.5% mark rather than 99% mark. While still relatively high, we attribute this to the large average size of fmaps in VGG19. For this network, the statistical error in the large injection campaign of Mismatch-12288 might not be small enough. However, later sections show that this difference is minute when considering the coverage versus overhead trade-off, since the precise ranking of fmaps is less important as long as it is *approximately* well-ordered. Thus, while the Manhattan distance provides us with empirical evidence for the convergence of Mismatch and ΔLoss, FLR does not suffer from small imprecisions in the ordering. Attaining a good ordering quickly is advantageous for faster offline resiliency analysis, and this can be performed with ΔLoss for all networks studied.

Coverage versus Overhead: Fig. 4 shows the performance of FLR as a selective resiliency technique for 6 networks (we exclude AlexNet for space considerations, but the trends are the same), measured by the coverage versus overhead trade-off for selective fmap duplication. The X-axis shows the cumulative coverage by selectively protecting fmaps based on a vulnerability ordering, and the Y-axis shows the corresponding overhead as a percentage of additional MAC operations. We plot the trade-off for 6 vulnerability orderings: the Oracle (Mismatch-12288), Loss-12288, mismatch and loss at the 99% convergence points (Table II), and Mismatch and Loss at 64 inj/fmap. The inclusion of Mismatch-64 and Loss-64 help further illustrate the the faster convergence of ΔLoss relative to Mismatch.

Fig. 4 shows that the computational overhead is always sublinear to coverage, indicating that selective protection is in fact advantageous to full duplication and can even provide large benefits. For example, covering 90% of errors in SqueezeNet incurs only 29% overhead for the network, emphasizing that only a fraction of fmaps possess most of the vulnerability for the network. Similarly, MobileNet attains nearly 98% coverage (for 64% overhead) before a sudden, vertical rise in overhead for the last 2%. In this case,

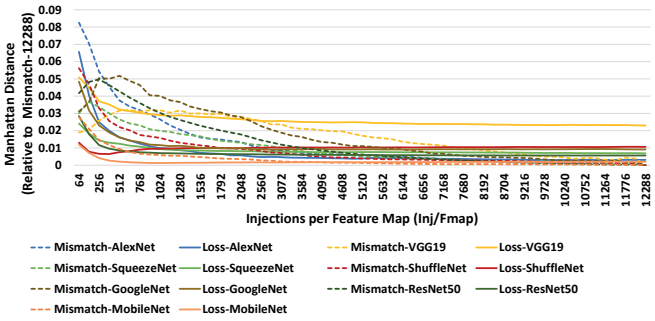


Fig. 3: ΔLoss and Mismatch converge as inj/fmap increase.

TABLE II: Comparison of Mismatch and ΔLoss

Network Neural	Inj/Fmap for 99% Oracle Mismatch	Inj/Fmap for 99% Oracle ΔLoss	Speedup from Oracle Mismatch	Speedup from Oracle ΔLoss	ΔLoss Speedup
AlexNet	2560	896	4.8×	13.7×	**2.9**×
GoogleNet	5632	1536	2.2×	8.0×	**3.7**×
MobileNet	640	128	19.2×	96.0×	**5.0**×
ResNet50	3584	384	3.4×	32.0×	**9.4**×
ShuffleNet	1664	1280	7.4×	9.6×	**1.3**×
SqueezeNet	3072	896	4.0×	13.7×	**3.4**×
VGG19*	2560	1536	4.8×	8.0×	**1.7**×
Geomean	2382	738	5.2×	16.7×	**3.2**×

* For 97.5% similarity to Oracle

we find that MobileNet has a unique feature: an imbalance of fmap sizes (captured by V_{orig}), such that the vulnerability of larger fmaps dominate, while a tail of smaller fmaps can be relegated in protection. For VGG19, despite the small difference in convergence between ΔLoss and Mismatch in fmap ordering (Table II), using selective protection shows that an informed, approximate ordering (as employed by FLR at fewer inj/fmap) still provides an opportunistic coverage versus overhead tradeoff for CNN resiliency.

Validation: Figure 5 validates the use of ΔLoss as a metric for vulnerability analysis, where we show the estimated coverage predicted by FLR using ΔLoss on the AS (X-axis), and comparing it to the actual coverage as measured by the number of SDCs protected against on the DS (Y-axis). The results show that ΔLoss is representative of the actual vulnerability as measured by mismatches in the DS. Thus, the prediction provided by ΔLoss is an excellent alternative for system developers for error analysis compared to Mismatch. Furthermore, FLR has no false positives (i.e., no detection without an underlying hardware error) by design because it uses duplication and an equality check.

B. ILR Results and Analysis

Correlation Between Inference Confidence and SDCs: We discovered a strong correlation between inference output and the vulnerability of the inference. Figure 6 illustrates this correlation. We plot the number of SDCs for 1000 randomly selected images run on AlexNet on the primary Y-axis. The secondary Y-axis shows the image's error-free Top1Conf and Top2Diff values. We measure the Spearman correlation between the per-images SDC rate and the two confidence metrics we extract. For AlexNet, the Spearman correlations are -0.87 for Top1Conf and -0.93 for Top2Diff, where -1.0 indicates a perfect inverse relationship. Both metrics exhibit a very high correlation relationship between the number of SDCs observed for an image and the image's confidence, which we can leverage for resiliency analysis and hardening.

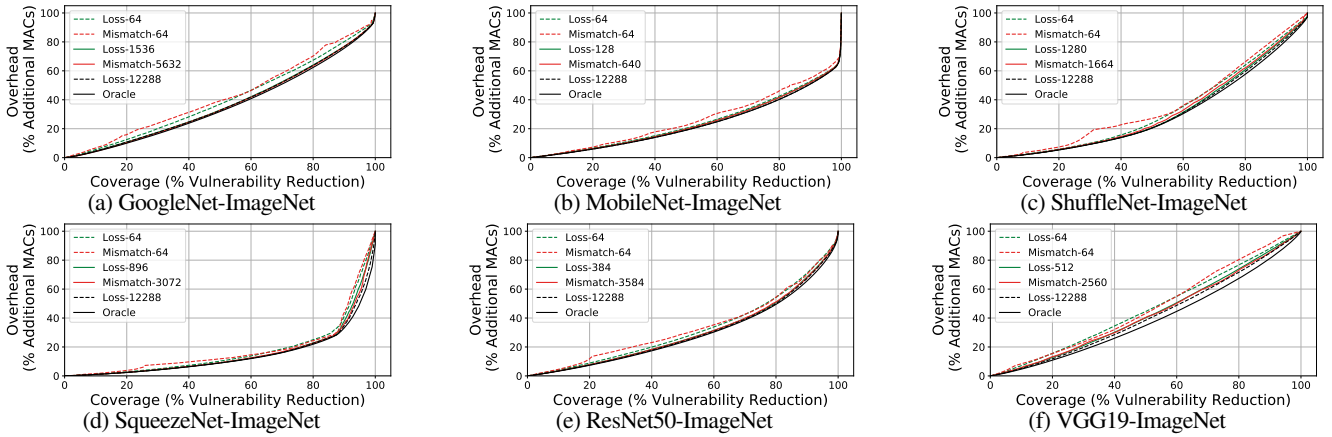


Fig. 4: FLR vulnerability reduction versus computational overhead.

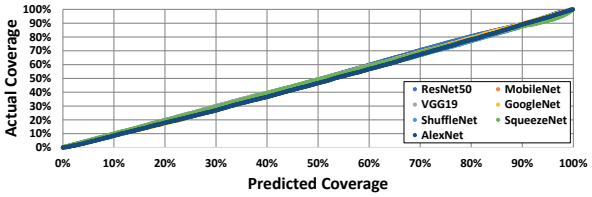


Fig. 5: FLR validation of predicted versus actual coverage.

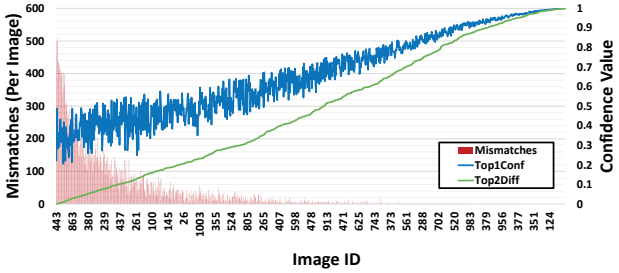
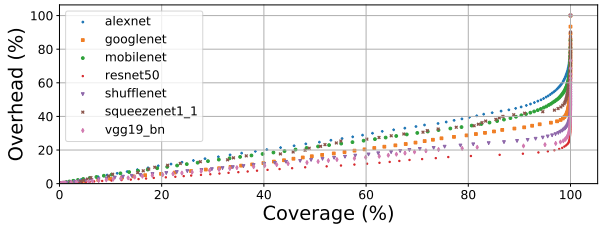
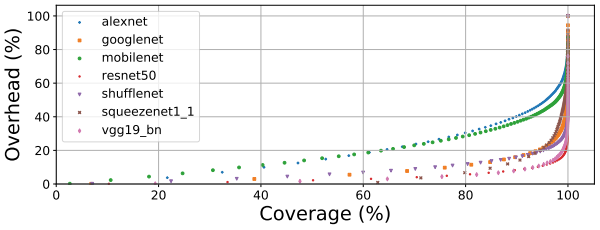


Fig. 6: Correlation between an image's confidence and SDCs (AlexNet). Images are sorted in ascending order based on their Top2Diff.



(a) Metric: Top1Conf



(b) Metric: Top2Diff

Fig. 7: ILR coverage versus overhead tradeoff at different thresholds.

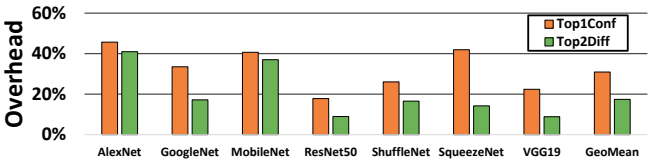


Fig. 8: ILR overhead at 90% coverage.

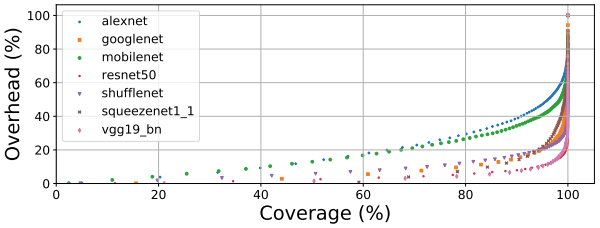


Fig. 9: ILR Validation for Top2Diff on DS.

Coverage versus Overhead: While both ILR metrics show high correlation for SDC detection, we find that Top2Diff performs better overall. Figure 7 illustrates this difference. Each point shows the coverage and overhead obtained for a confidence threshold set by ILR (as described in §VI-C). For both metrics, we find that ILR provides a favorable trade-off in terms of obtaining high coverage and incurring low overhead, signified by all points being below the x=y line. More importantly, the pareto-optimal threshold values are strictly better for Top2Diff, illustrated by a lower "knee" for each network. ResNet50, for example, can achieve 90% coverage at only 9% overhead using ILR with Top2Diff, while incurring 18% overhead with Top1Conf. Figure 8 summarizes this result for all networks, showing that on average, we can obtain 90% coverage with only 17% overhead using Top2Diff, compared to 31% overhead on average with Top1Conf. We focus on Top2Diff as the decision criterion for ILR moving forward as it performs better.

Validation: We validate ILR by measuring the coverage versus overhead trade-off on the DS. Figure 9 shows the tradeoff plot for ILR using Top2Diff, showing similar trends as analyzed on the AS (Figure 7b). Results for ILR with Top1Conf on the DS (not shown here for space constraints) are also very similar to the results on AS. That the AS and DS contain exclusively different images reinforces the use of Top2Diff as a confidence-based metric for CNN resiliency by quantitatively showing a strong correlation between the confidence of an inference and SDCs.

Analysis: Our results show that confidence-based metrics for SDC detection are highly effective. Top2Diff in particular helps explain the phenomenon of a mismatch, showing that a soft error has a higher probability of causing a mismatch if the margin

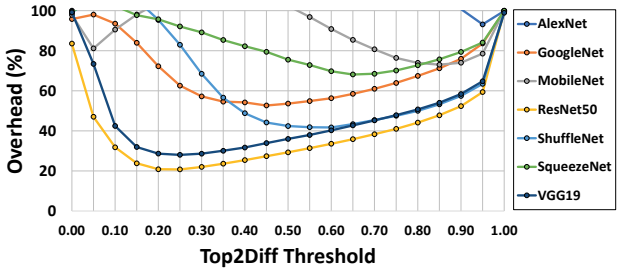


Fig. 10: FILR overhead for 100% coverage on AS.

between the top two classes is small, which translates to a higher probability of an SDC appearing. For example, we found that (correctly classified) images with a Top2Diff less than 0.01 had an SDC rate as high as 18% for ResNet50 (average of 11.9% for all networks), which is significantly higher than the overall SDC rate (less than 1% on average) across all networks and images.

ILR's overhead is attributed to reruns caused by inferences that have small Top2Diff, i.e., the inputs for which the model is confused about the correct class. Networks that exhibit a higher classification accuracy, such as ResNet50, suffer less from this phenomenon compared to AlexNet. Analogously, ILR's overhead can be high (a maximum of 100%) for a sequence of images that have small Top2diff, which can be caused by operating on images that are considered out-of-distribution (OOD). Improving model accuracy for OOD images should improve the Top2diff for many images and hence ILR's overhead. An OOD detector [1] (which remains an active research topic) can also be employed to temporarily switch to FLR to avoid near-100% overhead, and is an interesting future direction to pursue.

C. FILR Results and Analysis

Combination of Techniques Results: Figure 10 shows the results for FILR, which combines the two resiliency techniques of FLR and ILR as an optimized resiliency solution. All points in the figure represent 100% SDC coverage on the AS. The X-axis shows the Top2Diff threshold used by ILR, which also influences the FLR analysis as described in §VI-D. The Y-axis shows the overhead for FILR, which is a result of both the static overhead from fmap duplication and dynamic overhead from inference reruns.

A Top2Diff threshold of 0 corresponds to no coverage or overhead contribution from ILR and only FLR protection. The 1.0 Top2Diff threshold corresponds to only using ILR (with FLR not needing to protect any fmaps). The sweep of Top2Diff thresholds in between show that there exists an optimal point for each network below the 1.0 Top2Diff threshold, where ILR and FLR collaborate to achieve high, 100% coverage with an overhead that is below 100% (for design points with >100% overhead, full duplication is preferable). On average, the overhead via FILR is 48.07% across networks, and as low as 20.78% for ResNet50.

Validation: Figure 11 shows the validation (on the DS) of the optimal points from FILR (based on the AS), as described in §VI-D. We find very high validation accuracy, showing 99.78% coverage at an average of 47.66% overhead (as low as 20.47% for ResNet50). These results show that the FILR technique is better than the sum of its parts, where each technique individually required near 100% overhead to obtain near 100% coverage.

Implementation Measurements: We implemented FLR at the optimal point indicated by FILR (Figure 10) and measured the static

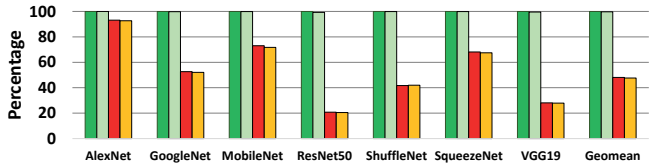


Fig. 11: FILR validation at optimal Top2Diff thresholds.

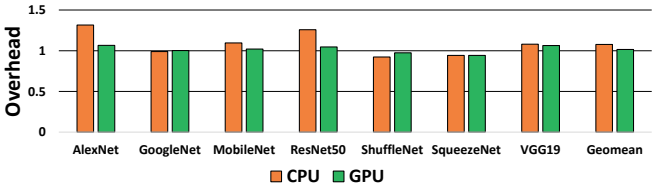


Fig. 12: Runtime overhead at optimal FILR design points (Jetson Xavier).

model runtime overhead on a Jetson AGX Xavier, on the CPU and GPU separately. Figure 12 shows the results. We find that the measured runtime overheads are indeed platform-specific. CPU runtime overheads are higher on average, increasing the model runtime by approximately 7.7% on average, while GPU runtime overhead is much less at 1.6% on average. While many factors can influence the exact runtime of the hardened models (as explained in §VI), our results show that depending on the platform and runtime environment, many computations can effectively be "hidden" by the availability of spare hardware resources, effectively providing lower overheads than predicted by our MAC-based model. While exact runtime overheads would require a more thorough analysis and is part of our future work, this study shows the portability of implementing our techniques on different hardware platforms, as well as show the differing results which may be accompanied by the platform of choice.

Analysis: FILR is effective at flattening the sharp rise observed by ILR alone (in Figure 7b) at higher coverage points. More generally, FILR combines the benefits of each technique, by providing a low overhead starting point via ILR, followed by a shallow growth for higher error coverage via FLR's selective feature duplication. The key behind this symbiotic relationship is using FLR to focus selective protection of fmaps for the subset of SDCs missed by ILR. Furthermore, using ΔLoss as the resiliency metric for FLR has a subtle yet important contribution in the combined analysis, since it can help distinguish fmap vulnerabilities faster at fine granularity.

D. Errors Not Captured by INT8 Range Detectors

As discussed in §II, the error coverage of a CNN model is closely coupled with the dynamic range of the underlying data format. In our case, all results in §VII are for CNN models that use INT8 quantization. The 0% error coverage points in Fig. 4, 6, and 7 refer to the baseline scenarios when the INT8 quantized models are not equipped with FLR/ILR. In these baseline scenarios, the range checking capabilities implicitly provided by quantizing *are* present. Thus, our results demonstrate the detection coverage FLR/ILR provide *beyond* the range checking detection of INT8 quantization. The corresponding network-level vulnerabilities are shown in Table III (discussed further in §VIII). Since FLR/ILR can be tuned for a desired coverage or overhead budget, the results in Fig. 4 and 7 show the coverage vs. overhead graphs for FLR/ILR, respectively. For FILR, we set the target coverage at 100% and find the lowest overhead solution by finding an appropriate Top2Diff threshold (Fig. 10).

TABLE III: Network level vulnerability (smaller is better). $E := 10^n$.

Network	V_{CNN}	Network	V_{CNN}	Network	V_{CNN}
AlexNet	$6.21E-3$	ShuffleNet	$6.38E-5$	MobileNet	$4.33E-5$
GoogleNet	$1.54E-4$	SqueezeNet	$9.54E-5$	VGG19	$1.75E-4$
ResNet50	$3.79E-5$				

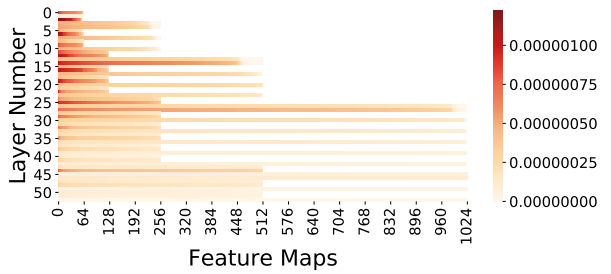


Fig. 13: Layer level analysis. ΔLoss on ResNet50 (cutoff at 1024 fmaps).

TABLE IV: Error Models

Name	Format	Quantized?	Description
FP-Rand	32-Bit Floating-point	No	Random value from [-max, max]
FxP-Rand	8-Bit Fixed-point	Yes	Random, multi-bit flips
FxP-Flip	8-Bit Fixed-point	Yes	Random, single bit flip

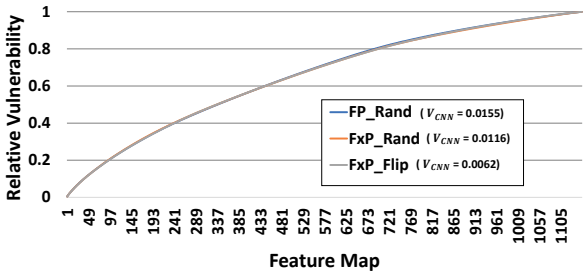


Fig. 14: $Vrel_{fmap}$ is similar across error models, even with different V_{CNN} (AlexNet-ImageNet).

VIII. CNN Model Resilience Analysis

Network Level Analysis: Our evaluation methods allow us to perform a network level resilience analysis and compare total vulnerability values, V_{CNN}, of different CNNs, allowing a developer to make an informed decision about selecting a CNN that meets the resilience, performance, and accuracy targets. We show network-level vulnerability results in Table III. Since we use ΔLoss to predict P_{prop}, the listed values are in an arbitrary unit, but allow for relative comparison and selection (as validated by Figure 5). A separate small experiment can be performed to calibrate the scale to real probabilities. These results show that some models can be orders of magnitude more resilient than others. For example, ResNet50 can be about 164× more resilient than AlexNet, solving the same problem and trained on the same dataset. For a given vulnerability target, a user may select a model that meets the desired target directly or employ selective protection (using FILR, ILR, or FLR) to meet the target. For example, given a vulnerability target of 7.0×10^{-5}, selecting ResNet50, MobileNet, or ShuffleNet would satisfy the reliability need of the system without additional protection via any technique. Selecting SqueezeNet or GoogleNet would require duplicating a fraction of inferences or fmaps in ILR or FLR, respectively, to cover the difference.

Layer Level Analysis: We perform a layer level study to understand whether vulnerable fmaps are clustered in certain layers. Figure 13 shows a heatmap of ResNet50's fmap vulnerabilities (V_{fmap}) 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 (<33%) account for a large percentage of a CNN's vulnerability (>76%), and that the vulnerable fmaps are distributed across different layers. This reinforces our selective and granular protection strategy for CNNs, particularly since fmap granularity enables an efficient approach to target the most vulnerable components without overprotecting. Additionally, we find that earlier layers in general are more vulnerable overall, which we can attribute to the cascade effect of an earlier computation on later layers. Intuitively, this also informs why adversarial inputs are a big concern, and additional research identifying the precise intersection between transient computational errors and adversarial input errors is a promising future direction.

Error Model and Relative Vulnerability: We evaluate the effect of three error models on vulnerability estimation (Table IV), to study the extensibility to other number formats. FP-Rand and FxP-Rand represent multiple bit perturbations for floating and fixed point representations, respectively, and FxP-Flip represents a single bit flip for INT8 format (used throughout our evaluation up to this point). For the FP-rand model, the injected neuron value is selected to lie within a range. The *max* value is set by profiling the DNN and recording the maximum observed inference value. This range restriction is applied to model a range detector [9], [33]. FxP-Rand injects a random 8-bit quantized value, modeling a multi-bit perturbation. Fig. 14 shows the cumulative relative vulnerability ($Vrel_{fmap}$) of the fmaps in AlexNet, where the X-axis is sorted in descending order of $Vrel_{fmap}$ using Mismatch-12288. For the comparison, we use the same fmap order on the X-axis, based on FxP-Flip's $Vrel_{fmap}$.

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. The absolute total vulnerability (V_{CNN}), however, changes with the different errors models. FP-Rand and FxP-Rand exhibit closer network vulnerabilities, as both models have the same dynamic range and multi-bit perturbation error model. The V_{CNN} is slightly lower for FxP-Rand due to the influence of the numerical precision of the MACs (INT8) on V_{orig}. FxP-Flip shows even lower V_{CNN}, which we attribute to the less egregious error model, i.e., a single-bit perturbation, compared to a multi-bit perturbation with a max-value cutoff. While a limited study, this promising result shows that our technique can be used with various error models, as well as provide a new insight into the granularity of protection for domain-specific resilience of CNNs.

Range Detector vs. selective replication-based protection: The advantage of using a range detector is its very low cost. For CNN inferences that use data formats with a large value range (e.g., FP32 or FP16), range detectors are highly effective in reducing SDCs. A disadvantage of range checking is that it is ineffective for highly quantized, state-of-the-art models that use data types whose dynamic range is small and the neuron values span the entire range in fault-free operations by design. The advantage of selective protection-based techniques (e.g., FLR and ILR) is that they can be used to mitigate errors regardless of data format and dynamic range. A disadvantage of these techniques is that they can incur higher overheads but can be tuned based on desired coverage or performance overhead target as described in §VII.

IX. Conclusion

We introduce and evaluate three software-driven, tunable, selective protection techniques for CNN resilience: feature map level resiliency (FLR), inference level resilience (ILR), and a combined optimization technique (FILR). We leverage domain-specific insights to speed up resiliency analysis with the introduction of ΔLoss as a metric, and apply our insights to mitigate hardware error propagation in CNNs. Our results show that our FILR can achieve very high error coverage of 99.78% for quantized inferences. We use MAC operations for fast and portable overhead trade-off analysis (e.g., ResNet50 requires 20.8% additional MACs), while showing low implementation overhead on a Jetson Xavier GPU (4.6% for ResNet50).

Acknowledgements

This work was supported in part by the ADA and C-BRIC research centers, JUMP centers co-sponsored by SRC and DARPA. This work is also supported by the National Science Foundation under grants CCF 19-56374 and CCF 17-04834 and by the DAPRA Domain-Specific System-on-Chip (DSSOC) program. This work was largely performed while Abdulrahman Mahmoud was a graduate student at the University of Illinois and an intern at NVIDIA.

References

[1] V. Abdelzad, K. Czarnecki, R. Salay, T. Denouden, S. Vernekar, and B. Phan, "Detecting out-of-distribution inputs in deep neural networks using an early-layer output," *ArXiv*, vol. abs/1910.10307, 2019.

[2] K. Adam, I. I. Mohamed, and Y. Ibrahim, "Analyzing the resilience of convolutional neural networks implemented on gpus: Alexnet as a case study," *International Journal of Electrical and Computer Engineering Systems (IJECES)*, vol. 12, no. 2, 2021.

[3] ——, "A selective mitigation technique of soft errors for dnn models used in healthcare applications: Densenet201 case study," *IEEE Access*, vol. 9, pp. 65 803–65 823, 2021.

[4] S. Anwar, K. Hwang, and W. Sung, "Structured pruning of deep convolutional neural networks," *J. Emerg. Technol. Comput. Syst.*, vol. 13, no. 3, Feb. 2017.

[5] R. A. Ashraf, R. Gioiosa, G. Kestor, R. F. DeMara, C. Cher, and P. Bose, "Understanding the propagation of transient errors in hpc applications," in *SC '15: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2015, pp. 1–12.

[6] F. Cappello, G. Al, W. Gropp, S. Kale, B. Kramer, and M. Snir, "Toward Exascale Resilience: 2014 Update," *Supercomput. Front. Innov.: Int. J.*, 2014.

[7] F. Cappello, A. Geist, B. Gropp, L. Kale, B. Kramer, and M. Snir, "Toward exascale resilience," *Int. J. High Perform. Comput. Appl.*, vol. 23, no. 4, p. 374–388, Nov. 2009.

[8] C.-K. Chang, S. Lym, N. Kelly, M. B. Sullivan, and M. Erez, "Hamartia: A fast and accurate error injection framework," in *Proceedings of the International Conference on Dependable Systems and Networks Workshops (DSN-W)*. IEEE, 2018, pp. 101–108.

[9] Z. Chen, G. Li, and K. Pattabiraman, "A low-cost fault corrector for deep neural networks through range restriction," in *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2021, pp. 1–13.

[10] Z. Chen, G. Li, K. Pattabiraman, and N. DeBardelenben, "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*, ser. SC '19, 2019.

[11] F. Conti, "Technical report: Nemo dnn quantization for deployment model," *ArXiv*, vol. abs/2004.05930, 2020.

[12] 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*, vol. 30, no. 2, 2019.

[13] S. Feng, S. Gupta, A. Ansari, and S. 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)*, 2010, pp. 385–396.

[14] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. Mahoney, and K. Keutzer, "A survey of quantization methods for efficient neural network inference," *ArXiv*, vol. abs/2103.13630, 2021.

[15] B. F. Goldstein, V. C. Ferreira, S. Srinivasan, D. Das, A. S. Nery, S. Kundu, and F. M. G. França, "A lightweight error-resiliency mechanism for deep neural networks," in *2021 22nd International Symposium on Quality Electronic Design (ISQED)*, 2021, pp. 311–316.

[16] H. Guan, L. Ning, Z. Lin, X. Shen, H. Zhou, and S.-H. Lim, "In-place zero-space memory protection for cnn," *ArXiv*, vol. abs/1910.14479, 2019.

[17] S. K. S. Hari, S. V. Adve, H. Naeimi, and P. 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)*, 2012, pp. 123–134.

[18] S. K. S. Hari, M. Sullivan, T. Tsai, and S. W. Keckler, "Making convolutions resilient via algorithm-based error detection techniques," *IEEE Transactions on Dependable and Secure Computing*, pp. 1–1, 2021.

[19] S. K. S. Hari, T. Tsai, M. Stephenson, S. W. Keckler, and J. 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)*. IEEE, 2017, pp. 249–258.

[20] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015. [Online]. Available: http://arxiv.org/abs/1512.03385

[21] Y. He, P. Balaprakash, and Y. Li, "Fidelity: Efficient resilience analysis framework for deep learning accelerators," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2020, pp. 270–281.

[22] S. Hong, P. Frigo, Y. Kaya, C. Giuffrida, and T. Dumitras, "Terminal brain damage: Exposing the graceless degradation in deep neural networks under hardware fault attacks," in *28th USENIX Security Symposium (USENIX Security 19)*, Santa Clara, CA, Aug. 2019, pp. 497–514.

[23] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and<1mb model size," *CoRR*, vol. abs/1602.07360, 2016. [Online]. Available: http://arxiv.org/abs/1602.07360

[24] 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.

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

[26] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2704–2713, 2018.

[27] N. P. Jouppi, D. Hyun Yoon, M. Ashcraft, M. Gottscho, T. B. Jablin, G. Kurian, J. Laudon, S. Li, P. Ma, X. Ma, T. Norrie, N. Patil, S. Prasad, C. Young, Z. Zhou, and D. Patterson, "Ten lessons from three generations shaped google's tpuv4i : Industrial product," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, 2021, pp. 1–14.

[28] P. Kogge, S. Borkar, D. Campbell, W. Carlson, W. Dally, M. Denneau, P. Franzon, W. Harrod, J. Hiller, S. Keckler, D. Klein, and R. Lucas, "Exascale computing study: Technology challenges in achieving exascale systems," *Defense Advanced Research Projects Agency Information Processing Techniques Office (DARPA IPTO), Techinal Representative*, vol. 15, 01 2008.

[29] A. Krizhevsky, "One weird trick for parallelizing convolutional neural networks," *CoRR*, vol. abs/1404.5997, 2014. [Online]. Available: http://arxiv.org/abs/1404.5997

[30] I. Laguna, M. Schulz, D. F. Richards, J. Calhoun, and L. Olson, "Ipas: Intelligent protection against silent output corruption in scientific applications," in *Proceedings of the International Symposium on Code Generation and Optimization (CGO)*. IEEE, 2016, pp. 227–238.

[31] R. Leveugle, A. Calvez, P. Maistri, and P. Vanhauwaert, "Statistical fault injection: Quantified error and confidence," in *Design, Automation Test in Europe Conference Exhibition (DATE)*, 2009, pp. 502–506.

[32] 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*, 2018, pp. 554–565.

[33] G. Li, S. K. S. Hari, M. Sullivan, T. Tsai, K. Pattabiraman, J. Emer, and S. 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*, ser. SC '17. New York, NY, USA: ACM, 2017, pp. 8:1–8:12.

[34] G. Li, K. Pattabiraman, S. K. S. Hari, M. Sullivan, and T. Tsai, "Modeling soft-error propagation in programs," in *Proceedings of the International Conference on Dependable Systems and Networks (DSN)*, 2018.

[35] M.-L. Li, P. Ramachandran, S. K. Sahoo, S. V. Adve, V. S. Adve, and Y. 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.

[36] S. R. Li, J. Park, and P. T. P. Tang, "Enabling sparse winograd convolution by native pruning," *ArXiv*, vol. abs/1702.08597, 2017.

[37] 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*, vol. 66, no. 1, pp. 216–222, 2019.

[38] L. Liu and J. Deng, "Dynamic deep neural networks: Optimizing accuracy-efficiency trade-offs by selective execution," *CoRR*, vol. abs/1701.00299, 2017. [Online]. Available: http://arxiv.org/abs/1701.00299

[39] Q. Lu, M. Farahani, J. Wei, A. Thomas, and K. Pattabiraman, "Llfi: An intermediate code-level fault injection tool for hardware faults," in *2015 IEEE International Conference on Software Quality, Reliability and Security*. IEEE, 2015, pp. 11–16.

[40] N. Ma, X. Zhang, H. Zheng, and J. Sun, "Shufflenet V2: practical guidelines for efficient CNN architecture design," *CoRR*, vol. abs/1807.11164, 2018. [Online]. Available: http://arxiv.org/abs/1807.11164

[41] A. Mahmoud, N. Aggarwal, A. Nobbe, J. R. S. Vicarte, S. V. Adve, C. W. Fletcher, I. Frosio, and S. K. S. Hari, "Pytorchfi: A runtime perturbation tool for dnns," in *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, 2020, pp. 25–31.

[42] A. Mahmoud, S. Hari, C. W. Fletcher, S. Adve, C. Sakr, N. R. Shanbhag, P. Molchanov, M. B. Sullivan, T. Tsai, and S. Keckler, "Hardnn: Feature map vulnerability evaluation in cnns," *ArXiv*, vol. abs/2002.09786, 2020.

[43] A. Mahmoud, S. K. S. Hari, M. B. Sullivan, T. Tsai, and S. 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*, ser. SC '18, 2018.

[44] A. Mahmoud, R. Venkatagiri, K. Ahmed, S. Misailovic, D. Marinov, C. W. Fletcher, and S. 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*, ser. ASPLOS '19, New York, NY, USA, 2019, p. 1087–1103.

[45] F. P. Miller, A. F. Vandome, and J. McBrewster, *Amazon Web Services*. Alpha Press, 2010.

[46] S. Mittal, "A survey on modeling and improving reliability of dnn algorithms and accelerators," *Journal of Systems Architecture*, vol. 104, p. 101689, 2020.

[47] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient transfer learning," *CoRR*, vol. abs/1611.06440, 2016. [Online]. Available: http://arxiv.org/abs/1611.06440

[48] NVIDIA, "NVIDIA Tesla V100 GPU Accelerator," https://images.nvidia.com/content/technologies/volta/pdf/tesla-volta-v100-datasheet-letter-fnl-web.pdf, 2018.

[49] NVIDIA, ""Improving INT8 Accuracy Using Quantization Aware Training and the NVIDIA Transfer Learning Toolkit"," https://developer.nvidia.com/blog/improving-int8-accuracy-using-quantization-aware-training-and-the-transfer-learning-toolkit/, Aug 2020.

[50] ——, ""NVIDIA Data Center Deep Learning Product Performance"," https://developer.nvidia.com/deep-learning-performance-training-inference, Nov 2020.

[51] E. Ozen and A. Orailoglu, "Low-cost error detection in deep neural network accelerators with linear algorithmic checksums," *Journal of Electronic Testing*, pp. 1–16, 2020.

[52] ——, "Boosting bit-error resilience of dnn accelerators through median feature selection," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 11, pp. 3250–3262, 2020.

[53] ——, "Just say zero: Containing critical bit-error propagation in deep neural networks with anomalous feature suppression," in *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2020, pp. 1–9.

[54] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," in *NIPS-W*, 2017.

[55] K. 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.

[56] L. Perez and J. Wang, "The effectiveness of data augmentation in image classification using deep learning," *CoRR*, vol. abs/1712.04621, 2017. [Online]. Available: http://arxiv.org/abs/1712.04621

[57] L. Ping, J. Tan, and K. Yan, *SERN: Modeling and Analyzing the Soft Error Reliability of Convolutional Neural Networks*, New York, NY, USA, 2020, p. 445–450.

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

[59] V. Rajagopal, C. K. Ramasamy, A. Vishnoi, R. N. Gadde, N. R. Miniskar, and S. K. Pasupuleti, "Accurate and efficient fixed point inference for deep neural networks," in *2018 25th IEEE International Conference on Image Processing (ICIP)*, 2018, pp. 1847–1851.

[60] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[61] 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.

[62] S. Sahoo, M.-L. Li, P. Ramchandran, S. V. Adve, V. Adve, and Y. Zhou, "Using Likely Program Invariants to Detect Hardware Errors," in *Proc. of International Conference on Dependable Systems and Networks (DSN)*, 2008.

[63] C. Sakr, Y. Kim, and N. R. Shanbhag, "Analytical guarantees on numerical precision of deep neural networks," in *ICML*, 2017.

[64] C. Sakr and N. 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)*, pp. 1090–1094, 2018.

[65] M. Sandler, A. G. Howard, M. Zhu, A. Zhmoginov, and L. Chen, "Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation," *CoRR*, vol. abs/1801.04381, 2018.

[66] 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)*, March 2018, pp. 979–984.

[67] C. Schorn, A. Guntoro, and G. 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*, 2019, pp. 1507–1512.

[68] C. Shorten and T. M. Khoshgoftaar, "A survey on image data augmentation for deep learning," *Journal of Big Data*, vol. 6, pp. 1–48, 2019.

[69] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014.

[70] Z. Song, B. Fu, F. Wu, Z. Jiang, L. Jiang, N. Jing, and X. Liang, "Drq: Dynamic region-based quantization for deep neural network acceleration," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, 2020, pp. 1010–1021.

[71] C. R. Systems. (2012) Sample size calculator. Website. [Online]. Available: https://www.surveysystem.com/sscalc.htm

[72] V. Sze, Y. H. Chen, T. J. Yang, and J. S. Emer, *Efficient Processing of Deep Neural Networks*, 2020.

[73] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," *CoRR*, vol. abs/1409.4842, 2014.

[74] E. Talpes, D. D. Sarma, G. Venkataramanan, P. Bannon, B. McGee, B. Floering, A. Jalote, C. Hsiong, S. Arora, A. Gorti, and G. S. Sachdev, "Compute solution for tesla's full self-driving computer," *IEEE Micro*, vol. 40, no. 2, pp. 25–35, 2020.

[75] L. Tan and N. DeBardeleben, "Failure analysis and quantification for contemporary and future supercomputers," *ArXiv*, vol. abs/1911.02118, 2019.

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

[77] R. Venkatagiri, A. Mahmoud, S. K. S. Hari, and S. 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)*, 2016, pp. 1–14.

[78] R. Venkatesan, Y. S. Shao, M. Wang, J. Clemons, S. Dai, M. Fojtik, B. Keller, A. Klinefelter, N. Pinckney, P. Raina, Y. Zhang, B. Zimmer, W. J. Dally, J. Emer, S. W. Keckler, and B. Khailany, "Magnet: A modular accelerator generator for neural networks," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2019, pp. 1–8.

[79] N. J. Wang and S. J. Patel, "ReStore: Symptom-Based Soft Error Detection in Microprocessors," *IEEE Transactions on Dependable and Secure Computing*, vol. 3, no. 3, July-Sept 2006.

[80] WikiChip, "FSD Chip - Tesla - WikiChip," https://en.wikichip.org/wiki/tesla_*car_company*/fsd_chip, 2019.

[81] J. . Zhang, "Towards energy-efficient and reliable deep learning inference," Ph.D. dissertation, 2020.