Towards Hardware Trojan Resilient Convolutional Neural Networks Accelerators

Peiyao Sun[1]. Basel Halak[1]. Tom J. Kazmierski[1]

Abstract

Convolutional neural network accelerators are increasingly used in safety-critical applications, including autonomous vehicles. Therefore, particularly vulnerable to hardware Trojan insertion, a security attack that takes place during the development of integrated circuits. This work presents for the first time, a large-scale study of the impact of hardware Trojan insertion on convolutional neural network accelerators, focusing on those that use approximate commuting techniques, prevalent in embedded applications. We investigate three types of such networks, MobileNet V2, ShuffleNet V2, and GhostNet, trained in datasets of grayscale speed limit sign images and GTSRB. Our results show that certain parts of these architectures are more susceptible to hardware Trojan attacks, specifically a specific set of procession elements, referred to as "important", in the classification, Relu6, and Max pooling layers, respectively. These findings are subsequently used to develop two countermeasures, the first relies on selective hardware redundancy(SHR), and the second uses a combination of hardware and time redundancy(SHTR). The proposed defenses are experimentally validated. Our results show that the SHR provides speedy recovery from an attack while incurring between 6-10% area overheads. Whereas SHTR requires more time to detect the Trojan, and its area overhead is much smaller (~ 0.3%).

Keywords Approximate Computing · CNN accelerator · Hardware Trojan · Lightweight countermeasure · Run-time monitoring

1 Introduction

Convolutional neural networks (CNNs), a subset of Artificial Intelligence algorithms, are essential for image recognition tasks [1], with ubiquitous applications such as facial recognition, and autonomous vehicles. and biometric authentication. However, the integration of CNNs into edge computing devices is challenging due to their constrained energy budget and computation resources[2]. This contrast has highlighted the need for CNN optimization and the creation of specialized CNN accelerators [2-4], spurring research and development in this field. This led to the emergence of lightweight CNN architectures such as MobileNet [5], [6], ShuffleNet [7], [8], and GhostNet [9]. Additionally, employing approximate computing (AC) technology has been suggested to enhance CNNs, by utilizing approximate multipliers, clipping networks, and reducing data lengths [10]–[12]. These strategies facilitate the feasibility of conducting CNN computations on edge devices. Simultaneously, in the area of hardware CNN accelerators, a multitude of designs have been put forward, including the DNNbuilder and Multi-CLP accelerator [13], [14], yet the question of hardware security persists as a pivotal concern to be addressed.

While safety is a theoretical prerequisite in the design and manufacturing of accelerators [15], [16], the current market conditions pose significant challenges. With the IC (integrated circuits) supply chain distributed across global companies, opportunities for hardware-level attacks increase significantly [17], [18]. It is not feasible to assure the trustworthiness of all personnel involved in IC production, exposing every stage of the supply chain to potential hardware-level attacks [17], [18]. Furthermore, successful attacks on CNN accelerators can lead to sensitive data breaches, degraded performance, and hardware piracy [17], [18]. For instance, Hardware Trojans (HTs) can be used to compromise CNN accelerators and such

Peiyao Sun (corresponding author)
ps1a18@soton.ac.uk
Basel Halak
basel.halak@soton.ac.uk
Tom J. Kazmierski
t.j.kazmierski@soton.ac.uk
[1] Electronics and Computer Science School, University of Southampton, United Kingdom

设置了格式: 字体: (中文) DengXian, (中文) 简体中文(中国大陆)

attacks are not rare. Several HTs aimed at CNN accelerators have been proposed [19]–[24], underscoring the need for robust protections against HT attacks in CNN accelerators.

Given the acute threats posed by HTs, a plethora of countermeasures have been introduced. These countermeasures combat not only general HTs but also those specifically targeting CNN hardware accelerators. General countermeasures include techniques like functional filler cells [25], layout filling [26], design obfuscation [27], and encoded circuits [28] aimed at preventing HT insertion. Pre-silicon detection [29], structural testing, functional testing [30], [31], and run-time detection methods [32] have been employed for detecting HTs. Specialized countermeasures against HTs targeting CNN accelerators include FM-ModComp [33], which enhances the likelihood of HT triggering during testing, and CLEANN [34], [35], which detects if the CNN input images carry malicious information as trigger signals. Further, two run-time detection methods from [36] successfully identify abnormal behaviors in the PEs of MaxPooling layers. However, all these countermeasures, barring run-time detection methods, may falter under certain conditions, for instance, when attacks involve a combination of hardware and software Trojans [18] or use special sequences of classification results to activate HTs [35]. To combat these challenges, run-time detection methods are proposed as a final line of defense. Although run-time detection methods usually perform well against HTs, they introduce significant overheads [18]. While some lightweight run-time detection methods have been proposed to mitigate this issue, the robustness of existing lightweight run-time detection methods remains a concern [36].

Current research dominantly explores methods for inserting HTs into CNN accelerators [19]–[24], while countermeasures receive less attention [33], [34], and [36]. Remarkably, none of these protection-oriented works provides a comprehensive vulnerability analysis of the whole CNN accelerator system, a crucial step for designing effective countermeasures, particularly in the context of run-time detection methods. This lack of vulnerability analysis is especially concerning for accelerators based on AC, given that AC-based systems inherently possess more vulnerabilities [37]–[40], leading to a more complex protection design. Consequently, this paper aims to fill this gap by conducting a thorough vulnerability analysis of AC-based CNN accelerators, built on three popular CNN architectures [6], [8], [9]. The goal of this analysis is to determine which layer in each architecture, a hardware trojan causes the most reduction in classification accuracy. This knowledge is subsequently used to selectively protect these vulnerable points in each design, which significantly reduces the potential impact of a hardware Trojan while incurring minimal implementation overheads. The contributions of this work are twofold:

- A comprehensive vulnerability analysis of CNN accelerators of ShuffleNet V2, MobileNet V2, and GhostNet is carried out, wherein a hardware Trojan is inserted in each layer of these architectures and its impact on the classification accuracy was measured. The CNN accelerators studied use TOSAM approximate multiplier [41] and the HT which utilized to evaluate the vulnerable level of every type of layers are based on the function tampering HT proposed on [36]. Experiment results show that PEs for ReLU6 layers, classification layers and MaxPooling layers are the most vulnerable points in these designs.
- Two countermeasures are proposed. The first combines the traditional hardware redundancy [32] and vulnerability analysis result, called selective hardware redundancy. The second technique further reduce the hardware overheads of the first countermeasure by combing selective hardware redundancy and time redundancy mechanism. Both countermeasures have certain ability to correct errors. A comparison of the overheads and performance of these defenses is also included.

The remainder of this paper is structured as follows: In Section 2 outline the research methodology and assumptions, including the threat model and the analysis approach. Section 3 explains the experimental setups and discussed the results of the vulnerability analysis. Section 4 developed the two defense techniques and evaluate their security, detection time, and area overheads. Finally, conclusions are drawn in Section 5.

2 Methodology

This section introduces the threat model adopted in this work and explains the rationale of the analysis methodology.

2.1 Threat Modelling

Hardware Trojans refer to a hardware-level security attack wherein, an adversary makes malicious modifications to the integrated circuits during the design (e.g., IP companies), implementation (e.g., SoC integration), or even the fabrication stage (e.g., malicious IC factory). These changes aim to sabotage the design functionality, and introduce a backdoor, or facilitate information leakage [18], [42], [43]. The attacker is typically assumed to have access to design files design tools or the physical layout, as well as be proficient in IC design [18]. Hardware trojans inserted at the design stage by malicious IP developers are difficult to detect, as they may involve subtle modifications to the circuit designs that are hidden or obfuscated. Techniques to defend against this type of attack include the Unused Circuit identification technique [29], which consists of identifying and removing suspicious circuitry-those circuits not used or otherwise activated by any of the design verification tests. However, such an approach cannot protect against malicious modification of the design specifications, which can feasibly take place at the IP development place, for example, tampering with the training data of a machine learning model can be used to generate a malicious implementation of the model (e.g. machine learning model for an autonomous car that does not recognize certain

traffic signs when a trojan is triggered), therefore the best strategy to defend against a trojan insertion at the IP development stage is to only use trustworthy companies and avoid integrating open-source designs unless thoroughly checked[18], [31]. Attacks at SoC integration and fabrication stages are harder to avoid because of the outsourcing of the IC implementation and fabrication, a trend that is very difficult to reverse due to the significant costs associated with these tasks and the need for affordable electronic systems[18]. Contrary to a hardware trojan attacker at the IP development stage, an adversary at the implementation/fabrication stage can only make limited modifications to the original circuit not to introduce an increase in the area or degradation in the performance, which makes it easy for such Trojan to be detected by typical verification and sign off tools[18], [31], [43].

The threat model of this work is based on the threat model proposed in the [44]. Comparing with the threat model in the [44], we narrow down the range of the steps where adversaries may be appeared. We only consider that HT attacks at the SoC or fabrication stage, wherein the adversary can make limited modifications to the digital design. The threat model is shown in Fig.1.

The adversary's target is to manipulate the processing elements (PEs) of the CNN accelerator to degrade its performance. Specifically, the attacker aims to ensure that the CNN accelerator operates normally when the hardware Trojan (HT) is inactive, but misclassifies images when the HT is triggered.

The primary challenges for the adversary in these two phases are: first, gaining a deep understanding of the integrated circuit (IC) design to insert the Trojan without disrupting the normal functionality of the system; second, remaining covert enough to bypass the verification and testing stages [45].

Moreover, in both the SoC and manufacturing stages, any modification may affect critical parameters such as system timing and power consumption, which are commonly used to detect hardware Trojans. As a result, attackers face significant limitations, as tampering with large-scale processing elements (PEs) can easily be detected through such deviations [45].

Attacks at the IP development stage such as those involving malicious modification of specifications or training data are beyond the scope of this work.

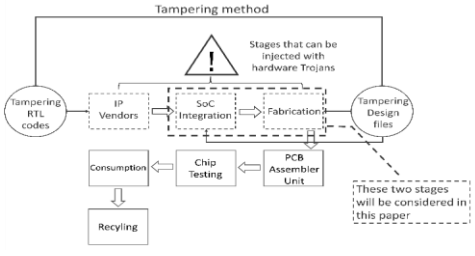


Fig. 1. Threat Modeling of Hardware Trojans

2.2 Principles of the Proposed Vulnerability Analysis

The study aims to identify locations for a hardware trojan insertion in the architecture or a CNN that leads to the most degradation in the accuracy classification. To achieve this, the first step is to modularize the CNN system and locate the attackable area, this is done by dividing the network layers into separate modules that are universally present in all architectures, namely CONVLs, FCLs, BNLs, pooling layers, activation layers, and CLLs. Three lightweight CNN architectures are considered here, MobileNet V2 [6], ShuffleNet V2 [8], and GhostNet [9], and a hardware implementation was developed for each design. The second phase is to devise a bespoke hardware trojan tailored for each layer. Thirdly, each design is modified, and evaluated experimentally, wherein only one processing element (PE) is attacked each time. The subsequent analysis included an estimation classification accuracy of the architecture before and after each hardware trojan insertion. The outcome of this analysis is subsequently used to develop an enhanced design for the CNN accelerators that is more resilient.

3 Experimental Setups and Implementation

This section outlines the hardware architecture of the CNN accelerator, the structure of the hardware trojans used in this work, and the experimental setups. It also provides a summary of the evaluation results.

3.1 Hardware Architecture of CNN Accelerators

The architecture of the three CNN structures used here, MobileNet V2, ShuffleNet V2, and GhostNet, can be divided into the following layers convolution (CONVL), Batch Normalization Layer(BNL), activation layers, Max Pooling layer (MPL)s global average pooling layers, concat layers, add layers, fully connected layer (FCL), classification layers(CLL), channel shuffle layers and channel split layers.

We did not adopt existing CNN accelerator architectures because most do not employ approximate computing components and are not open source. Instead, we designed a custom accelerator incorporating approximate multipliers to enable our security-focused analysis[46,47,48].

The core contribution of this work is a general strategy for defending against hardware Trojans: when attacker capabilities are limited, we identify and selectively protect the most vulnerable components. This idea applies broadly to RTL-level CNN accelerators. Furthermore, our mitigation leverages the structural uniformity of processing elements

(PEs) in CNN accelerators[46,47,48] to reduce hardware overheads of the runtime detection units, making the approach architecture-independent and widely applicable. For RTL-level CNN hardware accelerators with a large number of identical processing elements (PEs), our general strategy for defending against hardware Trojans is applicable, as are the countermeasures we will describe later.

Each of these layers is implemented using its unique procession elements (PE) to adhere to the modular design approach explained in Section 2. The elements associated with each layer have been implemented using either precision or approximate computing multipliers. The work in [38] has shown that the input data provided for a CNN image have varied levels of importance with respect to the accuracy of the resulting classification. Therefore, all the feature maps have been divided into the critical region and the insignificant region. The important features are processed with high-precision PEs or important PEs and the unimportant features can be processed with low-precision PEs or unimportant PEs.

The channel shuffle layer, channel split layer, and concat layer are typically implemented as lockup tables in the system memory, an approach that was also adopted in this work. The structures of the CONVL and BNL are shown in Fig. 2 and Fig. 3. PEs for the CONVL mainly consist of one approximate multiplier used to handle the multiplication operation in the CONV operation, one register, one adder used to handle the sum operation, and one counter used to match the dimensions of the CONV kernels.

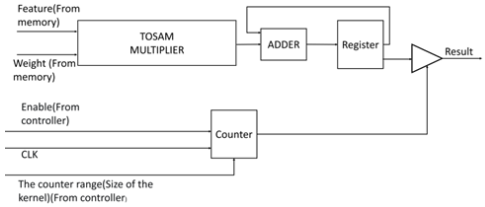


Fig. 2. Structure of processing element that for processing convolutional layers and fully connected layers

The PEs for the BNL consists of one approximate multiplier and two adders for processing BN calculation. In addition, based on the content in the last paragraph, these two kinds of PE have two different working accuracy levels with different approximate multiplier accuracy levels. For activation layers, the ReLU6 activation function which can limit the value of feature between 0 to 6, is utilized instead of the ReLU function in these three CNNs. This design decision was made to align the implementation with the use case of CNN for edge computing devices, most of which have constrained data length. The structure of the ReLU6 units is shown in Fig. 4. The structure of the PE of the FCL is the same as the PE in Fig. 3.

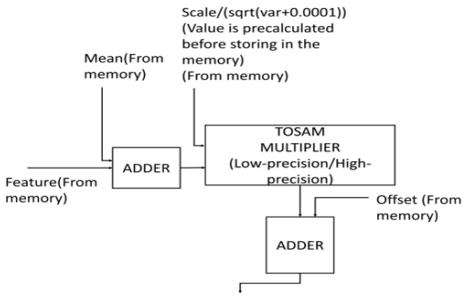


Fig. 3. Structure of processing element that for processing batch normalization layers

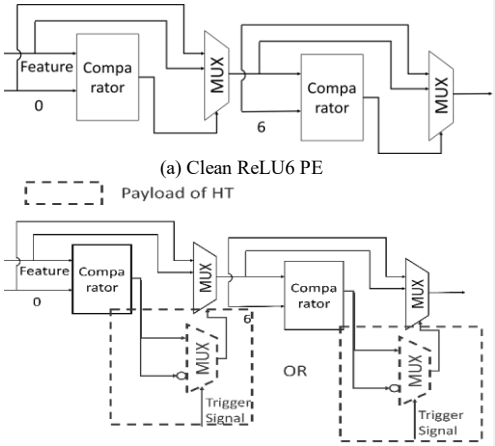


Fig. 4. Structure of hardware units for processing activation layers (ReLU6). (a) shows the structure of the clean ReLU6 PE and (b) shows the structure of the Trojan inserted ReLU6 PE. The section enclosed by the dashed line represents the payload of the hardware Trojan.

The high-precision mode of the TOSAM approximate multiplier shown in Fig. 5 used in the high-precision PE is backward-compatible with the low-precision modes.

The additional layer adds the two features together and stores them in memory. The functionality of this layer has been implemented with only one adder.

Then, the three network structures used in this article mainly use two types of pooling, the MPL, and the global average pooling layer. The PEs of these two pooling layers are shown in Fig.7 and Fig.6. The MPL selects the maximum value through digital comparators. The PE of global average pooling calculates the average value of every channel. When the total number of features (assuming equal to N) in the one

channel is a power of two, shifting accumulated results is used instead of the division, which can further reduce the overheads of the whole system. Finally, for the CLL, the calculation of the CLL is the same as that of the MPL, which is extracting the maximum feature. The specific structure of PEs for processing CLL is the same as the structure of Pes for processing MPL which is shown in Fig.7.

In addition, the structure of the CNN accelerator adopted in this paper is shown in Fig. 8. This accelerator is based on the work in [49]. Different modules are formed by different PEs that deal with different layers. Each module starts to work after receiving the enabled signal from the controller. When the corresponding work is done, a complete signal is given to the controller so that the controller knows when to give the start signal to the next sequential module.

Each module receives inputs (features, weights, parameters) from memory and sends outputs (features) to memory.

The memory mapping unit in the controller determines which memory address each PEs reads data from and sends data to. The PEs for the CONVLs, BNLs, and FCLs are also divided into low-precision PEs (unimportant PEs) and high-precision PEs (important PEs).

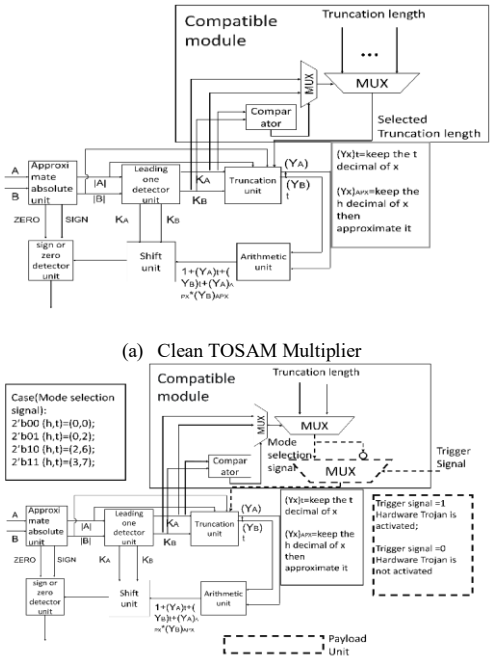
In addition, although all ReLU6 layers' PEs, MPLs' Pes, and addition layers' PEs are at the same precision level, they will also be divided into important PEs and unimportant PEs for processing important features and unimportant PEs respectively. Finally, the ratio of the number of PEs in the low-precision mode to the number of PEs in the high-precision mode is approximately equal to the ratio of the size of the important region to the non-important region in the feature map corresponding to this layer.

3.2 Hardware Trojan Insertion

In this section, several types of HTs which used to evaluate the vulnerable level of every type of layers in CNN are introduced. These HTs based on the HT designed in [36], which was used to attack the MaxPooling layer. In this section, this HT will be used to attack other layers including CONVLs, BNLs, FCLs, MPLs, ReLU6 layers, and CLLs. However, it is important to note that the design of hardware Trojans is not the focus of this paper. The following description of the hardware Trojan design is provided to offer a comprehensive experimental background, enabling readers to reproduce our work.

3.2.1 Hardware Trojan design

The HT in this paper is composed of two parts, the trigger recognition unit, and the payload unit, the same as the HT in work [36]. The accelerator architecture with HTs is shown in Fig. 8. The modules circled by dashed boxes are all places where HTs will be injected in this paper and the dashed unit is the trigger recognition unit. The methods of injecting the HTs are explained in detail in the following paragraphs.



(a) Clean TOSAM Multiplier

(b) Trojan inserted TOSAM Multiplier

Fig. 5. Structure of Clean and Trojan inserted TOSAM multiplier which supports multiple accuracy working modes. (a) shows the structure of the clean TOSAM Multiplier and (b) shows the structure of the Trojan inserted TOSAM Muliplier. The section enclosed by the dashed line represents the payload of the hardware Trojan.

First, consider the trigger recognition unit. Because the HT is not main content in this paper, the trigger mechanism of the HT is same the most of the existing HTs [21]–[23]. The trigger signal is hidden in the feature value, it reads the feature value from the target memory address and then processes this feature value to identify if the HT is activated. The result will be sent to the payload section. When the trigger condition is met, the result of the trigger recognition unit will activate the payload unit.

This attack can be implemented on the convolutional layers, batch normalization layers, and fully connected layers. The payload units of HTs proposed in this paper can modify the function of the compatible module in the high-precision approximate multipliers in the PEs of the CONVLs, BNLs, or FCLs. They also can modify the functions of the PEs in the ReLU6 layers and MPLs. The specific attack purposes of HT in different modules are introduced below.

- HTs in the high-precision approximate multiplier change their mode to low-precision. The specific

structure of the attacked high-precision multiplier is shown in Fig.5 (b). These HTs can be utilized to attack CONVLs, BNLs, and FCLs.

- HTs in the ReLU6 cause the output of the ReLU6 to be outside the expected range (0 to 6). The specific structure of the attacked PEs of ReLU6 layers is shown in Fig.4 (b).
- HTs in the MaxPooling PEs force modify its output such that the minimum value is obtained as opposed to the expected maximum. The specific structure of the attacked PEs of MPLs is shown in Fig.7 (b).

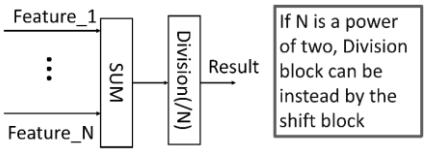
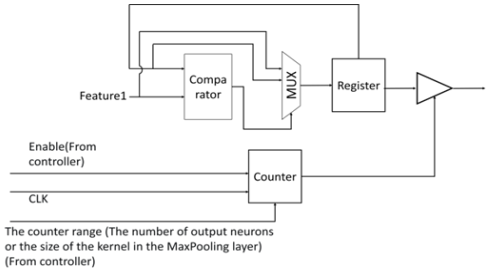
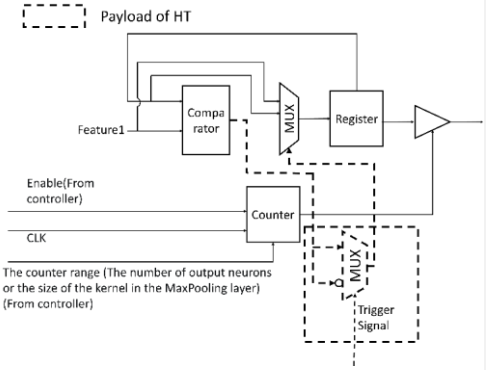


Fig. 6. Structure of global average layer's processing element



(a) Clean MaxPooling PE



(b) Trojan inserted MaxPooling PE

Fig. 7. Structure of processing elements that for processing classification layers and MaxPooling layers. (a) shows the structure of the clean MaxPooling PE and (b) shows the structure of the Trojan inserted MaxPooling PE. The section enclosed by the dashed line represents the payload of the hardware Trojan.

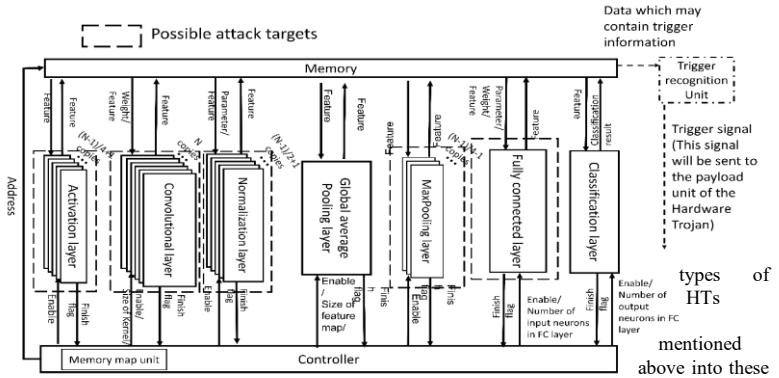


Fig. 8. CNN accelerator architecture based on MobileNet V2, ShuffleNet V2, and GhostNet and injected with HT.

3.3 Implementation and Analysis

The experiment setups have included building approximate accelerators of MobileNetV2, ShuffleNetV2, and Ghost-Net based on the TOSAM approximate multiplier, inserting all types of HTs mentioned above into these three accelerators, and analyzing the impact of each of these HTs on the classification accuracy of each design.

In this study, the data used are GTSRB dataset in grayscale.

For this dataset, we first use a subset of them, traffic signs for speed limits, with a total of 10 kinds of images, of which there are 18207 images as the training set, 1440 images as the validation set, and 180 images as the test set. After this experiment, the overall dataset of GTSRB is used, with a total of 43 kinds of images, of which 38000 images are used as the training set, 1209 images are used as the validation set, and 12630 images are used as the test set.

3.3.1 Structure of Nets

The three CNN architectures selected in this paper are MobileNetV2, ShuffleNetV2, and GhostNet. The data length used was 32 bits given the target application in edge computing. The structures of the three networks are shown in Fig.9, Fig.10, and Fig.11. In addition, for detecting if the impact of the HT will be varied with the different number of kernels, for MobileNetV2 CNN, we design an additional CNN with the different number of kernels. For both groups of networks, ShuffleNetV2 and GhostNet, the numbers of kernels are 32 for the first CONVLs in the speed limit sign network (the CONV layer used to increase the number of channels), and for the first CONVLs in the GTSRB network, The numbers of kernels are 128. For the MobileNetV2 network, the number of kernels of the first CONVL in the speed limit sign network is 32. However, for the first CONVL in the GTSRB network, the numbers of kernels are 32 and 64, respectively.

In addition, the function of TOSAM approximation multiplier is $A \times B \approx 2^{K_A} + 2^{K_B} \times (1 + (Y_A)_t + (Y_B)_t + (Y_A)_{APX} \times (Y_B)_{APX})$. In this function, K_A and K_B are the positions of the first '1' bit of A and B, respectively. Truncate data A and B to (h+1) bits and the approximate value of the truncated date are $(Y_A)_{APX}$ and $(Y_B)_{APX}$. Truncate data A and B to t bits and truncated data are $(Y_A)_t$ and $(Y_B)_t$. The high-precision mode of TOSAM approximate multipliers for processing important parts of the feature map is the mode (h, t) = (2,6). In addition, for high-precision approximate multiplier also supports the low-precision mode, (h, t)=(1,4).

Furthermore, for the low-precision approximate multiplier, the working mode is the mode (h,t)=(1,4) and if the multiplier or multiplicand cannot provide enough valid bits, the result is 0.

Each layer consists of several processing elements, some of which will be responsible for processing “important” data that are essential for accurate classification, while other processing elements will be processing data that do not significantly impact the classification's accuracy. This depends on the nature of the data being processed.

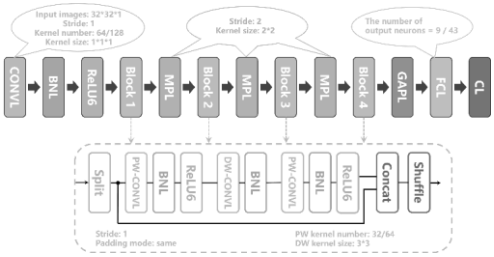


Fig. 9. The structure of the net based on the ShuffleNet V2.

Then, we have experimentally found that for a feature map of size 32×32 elements (32 bits), the 8 outermost layers of features are insignificant regions. For the feature map of size 16×16, the features of the 4 outermost layers are insignificant regions. For a feature map of size 8×8, the features of the 2 outermost layers are insignificant regions. For a feature map of size 4×4, the features of the 1 outermost layer are insignificant regions.

These findings are justified by the fact that the traffic signs in the images analyzed are typically placed at the center of the picture, or the outer part of the diagram does not have important information.

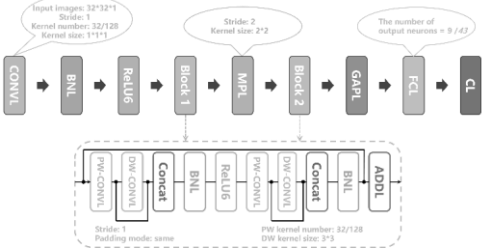


Fig. 10. The structure of the net based on the GhostNet.

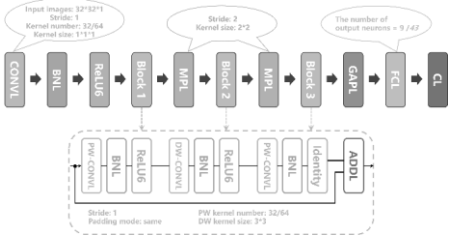


Fig. 11. The structure of the net based on the MobileNet V2.

As shown in the Fig.12 the channels of the feature map are equally divided into 4 groups (because the number of the channel of the output feature map are 16, 32, 64 for speed limit sign and the number of the channel of the output feature maps are 32, 64, 128 for GTSRB dataset all of them are divisible by the factor of 4), which are assigned to the 4 important PEs (It means that there are 4 important PEs in

every layer). The important data is the data located at the center of the feature map(dark regions). Each important PEs only processes the calculation related to the important data in the channels that are assigned to it. The rest data (unimportant data) will be processed by the unimportant PEs. To allow all PEs to complete all calculations of one feature map at the same time, the ratio of unimportant PE to important PE is equal to the ratio of the number of important data to the number of non-important data. So, there are 12 unimportant PEs in every layer.

The four important PEs assigned to each layer in all architectures, each tasked with processing a group of the output channels' important data. The twelve unimportant PEs are dedicated to processing less significant data and every 4 unimportant PEs are dedicated to process a group of the output channel's unimportant data. Additionally, within the FCLs , 9 PEs or 43 PEs are allocated, defined as important and every PE is dedicated to representing the possibility of the input image to be classified into one category. Finally, within the CLLs, 1 PE is allocated, defined as an important PE.

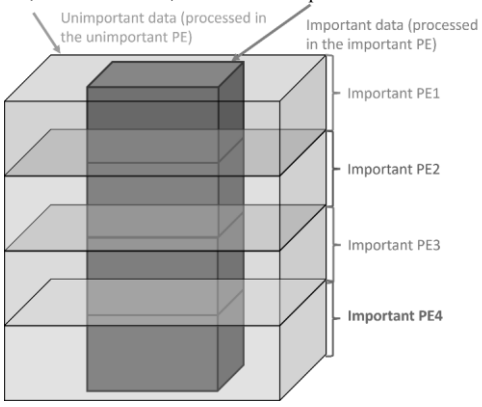


Fig. 12. The assignment of the important PEs in each layer.

3.3.2 *Impact of Hardware Trojans*

For detecting the impact of these HTs, the different probabilities of the picture being misclassified are detected, when HTs are not activated or HTs are activated and attacking on different kinds of layers. To prevent chance events, three networks were trained for every different CNN with different architectures or different number of kernels. The results shown in this section are the average of three sets of network simulation results. The simulations of this investigation were established in both System Verilog and MATLAB. Initially, the network training was conducted using MATLAB to ascertain the critical parameters, predominantly the network weights. Subsequently, Three CNN accelerators, focused solely on forward propagation, were constructed at the Register-Transfer Level (RTL) using the System Verilog language, facilitated by the Quartus and Questasim platforms. Then, HTs were built with System Verilog and injected into these accelerators. To expedite results, an emulation of these RTL-level accelerators, with and without the HT, was conducted using MATLAB. After making sure that the key data, such as the feature maps corresponding to each layer, were critically examined to ensure consistency between the MATLAB emulation and the RTL-level simulation results, MATLAB served as the principal platform for conducting subsequent experiments, including those aimed at assessing the impact of HTs.

As described in Section 3, the attacker's ability is limited to making one modification per architecture. For example, they are only able to attack one MUX in one PE, but they can select which PE to target.

The possibility of images being misclassified of the accelerators studied have been evaluated for all HT insertion scenarios. Firstly, the impacts of HTs when HTs are utilized to attack different kinds of important PEs are introduced.

The results, listed in Table 1, Table 2 and Table 3, show that the impact caused by the attacks on CONVLs, BNLs, and FCLs is not significant. When HTs attack important PEs in these layers, the possibility of the image being misclassified is nearly same as when no HT is injected.

On the other hand, an attack on classification, ReLU6, or Max Pooling layers leads to that images will have a high probability of being misclassified. More specifically, the probabilities of images being misclassified are 100% when a HT injected in classification layer for all studied architectures. The probabilities of images being misclassified are in the range of 68% to 91% for all studied architectures, when important PEs of ReLU6 layers are under attacking.

Table 1 The possibility of image being misclassified of MobileNet V2 before or after HTs being activated and when being attacked on different kinds of important PEs of all layers and unimportant PEs of MPLs and ReLU6 Layers.

Attacked Layer	GTSRB (More Kernels)	GTSRB (Less Kernels)	Speed Limit Sign
CONVL	6%	8%	6%
BNL	6%	8%	6%
ReLU6 (Important)	73%	75%	78%
ReLU6 (Unimportant)	71%	73%	71%
MPL (Important)	16%	35%	28%
MPL	11%	12%	9%

(Unimportant)			
FCL	6%	8%	5%
CLL	100%	100%	100%
Without Attack	4%	6%	6%

Table 2 The possibility of image being misclassified of ShuffleNet V2 before or after HTs being activated and when being attacked on different kinds of important PEs of all layers and unimportant PEs of MPLs and ReLU6 Layers.

Attacked Layer	GTSRB	Speed Limit Sign
CONVL	8%	14%
BNL	7%	10%
ReLU6 (Important)	88%	89%
ReLU6 (Unimportant)	68%	72%
MPL (Important)	24%	31%
MPL (Unimportant)	11%	12%
FCL	6%	9%
CLL	100%	100%
Without Attack	5%	9%

Table 3 The possibility of image being misclassified of GhostNet before or after HTs being activated and when being attacked on different kinds of important PEs of all layers and unimportant PEs of MPLs and ReLU6 Layers.

Attacked Layer	GTSRB	Speed Limit Sign
CONVL	4%	11%
BNL	5%	11%
ReLU6 (Important)	91%	88%
ReLU6 (Unimportant)	70%	68%
MPL (Important)	25%	33%
MPL (Unimportant)	10%	8%
FCL	5%	9%
CLL	100%	100%
Without Attack	5%	9%

Then, the probabilities of images being misclassified are in range 16-35%, when important PEs of MPL are under attacking.

Then the impact of HTs when HTs are utilized to attacked different kinds of unimportant PEs are introduced. Because the impacts of HTs when they are utilized to attack CONVLs', BNLs' and FCLs' important PEs are not serious, the impact of attacking on these layers' will be not introduced. In addition, because for CLLs, there are only important PEs, so PEs in CLLs will be also not considered in this part. The simulation results are shown in Table 1, Table 2, Table 3. Based on the data shown in these tables, the ReLU's unimportant PEs are also vulnerable, but the unimportant PEs of MPL are robustness.

The above analysis demonstrates that different part of the CNN architectures exhibits various level of vulnerability to a HT attack. The classification layer is the weakest followed by the ReLU6 layer. Next is the MPLs, CONVLs and BNLs, and FCLs are robustness to this kind of function tampering HT attack. It is also worth noting that all PEs in the ReLU6 layers are vulnerable.

4 Countermeasure

In this section, we will introduce the design of the two countermeasures and related evaluation.

4.1 Selective Hardware Redundancy (SHR)

The essence of this approach is to use the outcome of the vulnerability analysis from the previous section to introduce hardware redundancy selectively, which suggests that only the selected vulnerable PEs will be protected. The analysis has shown that the CLLs, MPLs and Relu6 layers are the most vulnerable, and For MPLs, only the important PEs are vulnerable. Therefore, only the important PEs MPLs and all PEs in ReLU6 layers and CLLs need to be protected. This approach uses the simple majority voting mechanism described in [32].

The structure of this approach is elaborated in Fig.13, wherein triple modular hardware redundancy is introduced for each processing element to be protected, and a majority voter is then used to determine the final output. This means that even if an adversary attack one of those elements, redundant hardware is still able to perform correct computation.

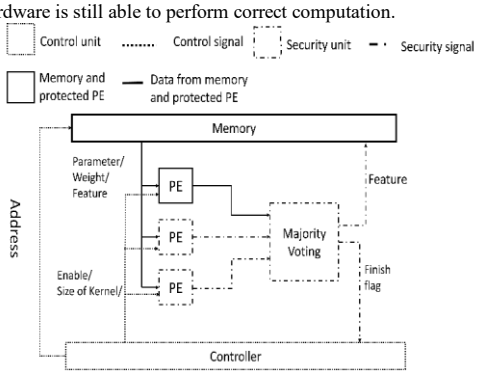


Fig. 13. The structure of Selective hardware redundancy.

4.2 Selective Hardware and Time Redundancy (SHTR)

This second approach relies on the intrinsic similarity of the hardware structure of the processing elements in CNN accelerators to further reduce the implementation overheads. For example, the PEs used for the convolution module, no matter in MobileNetV2, ShuffleNetV2 or GhostNet, have the same structure. The only difference is that the input data and the number in the calculation loop are not the same. This allows the countermeasure, RIA in a recent paper in [36], has the potential ability to protect the accelerators of CNNs. The RIA is a lightweight real-time monitoring method. This also fits well with the premise of lightweight protection. Here, the second countermeasure proposed in this paper is combining the RIA with the SHR mechanism. In the following content in this paper, we will call the countermeasure in this section the SHTR.

The working principles of this method are as follows. The procession elements to be protected are identified based on the vulnerability analysis outcome from Section 3. Additional two security processing elements are also added to each layer and used to verify the correctness of the output of each "important" processing element. This verification is performed by applying the same input data to these two modules and using a majority voter to compare the output of the two security elements and the processing element being checked. Every clock cycle, the control circuitry chooses one "important" processing element to check once all elements have been checked the process repeats. The hardware architecture of this approach is shown in Fig.14.

The control unit is responsible for coordinating the checking process by fetching the input and corresponding output data of the PE being checked from the memory.
The output from the processing element being checked (O) and from the two security elements (S1, S2) are stored in register A.

The checker compares (O) with the output of the majority voter. The outcome of this comparison is monitored by the control unit. If a mismatch is found, this means the processing element being checked has been compromised and will subsequently be replaced by the backup PE.

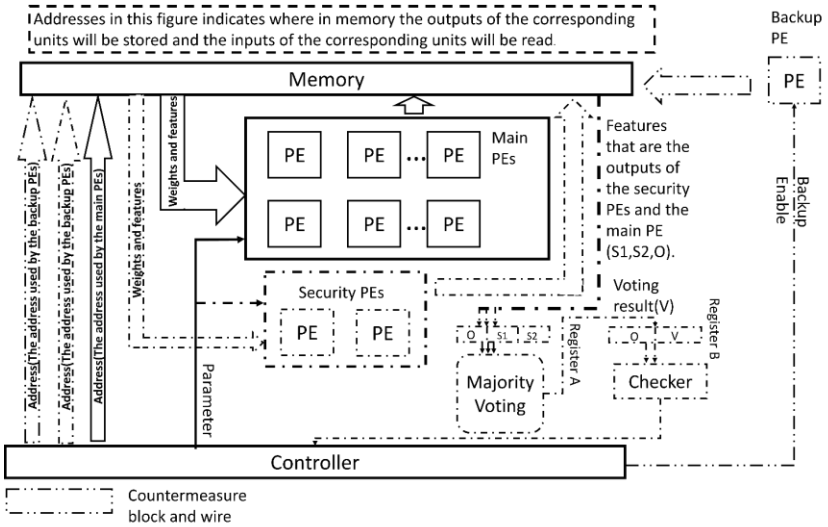


Fig. 14. The Structure of the Selective Hardware and Time Redundancy Approach

4.3 Evaluation of Proposed Countermeasures

This section compares the two proposed defenses in terms of security, and overheads.

4.3.1 Security Analysis

From a security perspective, both defenses assume that a hardware trojan is likely to be inserted in parts of the design such that it has a significant impact on the accuracy of the classification. Consequently, both defenses only protect vulnerable PEs as defined in the architecture of each accelerator. The vulnerable PEs in this experiment including the important PEs of MPLs, CLLs and all PEs of the ReLU6 PE. The SHR defense approach allows the system to function

correctly even if all protected elements have a trojan inserted. However, if the adversary is to maliciously modifies the processing element and one of its replicas, the self-correct ability will be failed. Increasing the number of the replicas is able to solve this problem. For example, if the adversaries are able to attack two PEs, the number of the replicas of every PEs need to be increased to 4.

The SHTR approach can invalidate a single HT attack on any of the protected procession elements by using the backup module. This method also works if one of the security processing elements or backup processing elements is attacked. Because the Majority Voting unit make sure only when most of voters are attacked, the voting result is wrong. But if the ability of adversaries is increased, the Majority Voting unit also face the invalidation risk. In this condition, the solving method is same as the SHR, increasing the number of the security PEs and backup PEs. For example, if the adversaries are able to attack 3 PEs, the number of security PE need to be increased to 6 and the number of backup PEs need to be increased to 3.

To compare the two approaches, we implemented protection on the three most sensitive layers (CLL, Relu6 layers, and MPLs). In this experiment, there are some differences in the protected objects between the two protection modes of SHR and SHTR. For SHR, the important PEs of MPLs and all PEs of ReLU6 layers and CLLs are protected. For SHTR, because the structure of PEs of MPLs' important PEs and unimportant PEs are the same, so without any additional overheads, the unimportant PEs of MPLs also can be protected, just required more time to finish one loop checking. In addition, this extra time is tolerable compared to the time it takes to classify an image. So, for SHTR, all PEs of ReLU6 layers MPLs and CLLs are protected.

We consider a scenario, wherein an adversary at the system integration stage can insert a single trojan in the accelerator. We assume that the adversary has sufficient knowledge of the implementation to choose a location for this trojan to cause the maximum degradation in the classification accuracy. In this case, the attacker would insert a HT in the PE of the CLL, which will cause the most serious impact, in the design that is not protected. For the two protected designs, the attackers are able to pay attention the protect circuit, but because that they are only able to insert one HT in one PE, so they would target the unimportant PE in the MPL.

The performance of each implementation was then measured ad shown in Table 4. The results show the defense techniques proposed in this work can significantly reduce the impact of such an attack. In addition, for SHTR mechanism, the impact of unimportant PEs of MPLs being attacked also can be recovered.

Table 4 The possibility of image being misclassified of GhostNet, MobileNetV2, ShuffleNetV2 accelerator before or after HT being activated and without protection or with SHR or SHTR.

Network	Dataset	Original	Attacked	Attacked with SHR	Attacked with SHTR
MobileNetV2	GTSRB (More Kernels)	4%	100%	11%	6%
	GTSRB (Less Kernels)	6%	100%	12%	8%
	Speed Limit Sign	6%	100%	9%	6%
ShuffleNetV2	GTSRB	5%	100%	11%	8%
	Speed Limit Sign	9%	100%	14%	14%
GhostNet	GTSRB	4%	100%	10%	5%
	Speed Limit Sign	8%	100%	11%	11%

4.3.2 Hardware Trojan Impact Recovery Time

The SHR approach allows immediate recovery from a Trojan insertion attack, as the output compromised processing element will be overridden by the majority voting circuitry. On the other hand, the SHTR scheme requires more time to detect and recover from a trojan insertion because important processing elements are checked sequentially, this means the accelerator with a trojan inserted will produce the wrong results for a period required to check all processing elements, and subsequently substitute a compromised module with one of the backup PE.

Let us take the example, analyzed previously where the SHTR approach is used to protect the three most sensitive layers (all PEs of CLLs, Relu6 layers, and MPLs). Furthermore, This implementation uses one protection module for both the MPLs and CLLs, the structure of the processing element in these two layers is the same, so we only need one backup PE.

In this case, the checking process runs in two parallel threads. The first checks the ReLU6 layers' PEs serially, and the second serially examines MPLs' and CLLs' PEs.

In Thread 1, the ReLU6 layers' PEs requires one clock cycle to produce the result, during which the output of one PE can be examined. Furthermore, the checking process includes five phases: (1) security PE result generation, (2) Register A data reading, (3) majority voting, (4) checker unit result comparison, and (5) controller's result generation. Every phase

need one clock cycle to complete operation. Therefore, in total, we need 6 clock cycles to complete check one PE and the implementations are pipelined. The specific timing diagram of SHTR units of Thread 1 is shown in Fig.15.

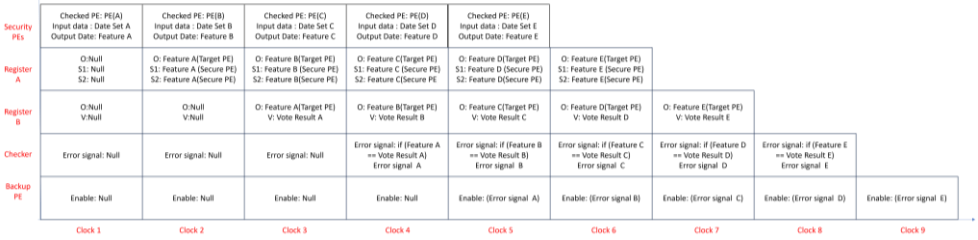


Fig. 15. The timing diagram of the all units of SHTR

Therefore, for Thread 1, ShuffleNetV2, MobileNetV2, and GhostNet need 148 (9*16+4), 116 (7*16+4), and 52 (12+4) clock cycles respectively to check one round. Here, the first part of the calculation represents the time to scan all PEs, and the latter part represents the remaining clock cycle required to complete the checking for the final PE.

Similarly, for Thread 2, the MPLs' PEs requires 4 clocks cycles to produce the final result, but Security PEs can utilize the intermediate results to check the performance of the monitored PEs, so still requiring 1 clock cycle for checking one PE. Hence, 52, 36, and 20 clock cycles are required for ShuffleNetV2, MobileNetV2, and GhostNet respectively. Thread 1 requires more time, hence should be considered as the worst-case scenario detection time for the SHTR approach.

4.3.3 Implementation Overheads

To estimate the area overheads of proposed countermeasures, the six architectures studied here have been implemented using System Verilog. Three versions of each design were constructed, without countermeasures, with selective hardware redundancy (SHR), and with Selective Hardware and Time Redundancy (SHTR). In the last two cases, the protection is applied to the three most sensitive layers (classification, Relu6, and Max Pooling).

All designs have been synthesized using Quartus on the FPGA device 5CGXFC9A6U19A7. The respective areas are obtained from the synthesis report in terms of the Adaptive Logic Module (ALM) used in each case, these are listed in Table 5.

The Results show the area overheads associated with the SHR approach rages between 10-6% of the original design areas. Comparing with the traditional Majority Voting mechanism which will protect all PEs, the additional overhead of the SHR is significantly reduced. While those associated with the SHTR one only a fraction of additional overheads of SHR, estimated to be around 0.4-0.2 %. This was expected given, comparing with overheads of PEs which contain multipliers, the overheads of the PEs of the MPL, CLL and ReLU6 is tiny. So, the additional overheads of the SHR is less than the Majority Voting. Furthermore, the additional hardware requirements of the SHR approach that triplicate each important procession element, compared to only using three additional PE for each type of PEs by the SHTR techniques, the overheads of SHTR is able to be further reduced.

4.4 Discussion

The countermeasures proposed in this paper demonstrate broad applicability to existing CNN hardware accelerator architectures. These countermeasures require only that the CNN hardware accelerator include buffers capable of temporarily storing the input and output data of PEs (features and weights). As many modern CNN hardware accelerators are equipped with on-chip memory buffers for storing data and filter weights [13,50,51,52], the proposed countermeasures can be readily applied to a wide range of such accelerators.

The primary focus of the proposed solutions is the mitigation of function-tampering hardware Trojans—those designed to produce erroneous outputs. However, these countermeasures exhibit limited efficacy against hardware Trojans that do not affect functionality, such as those designed to leak sensitive information. This represents a key limitation of the approach.

Nevertheless, there are additional limitations to consider regarding the proposed countermeasures. If an adversary successfully compromises the Majority Voting unit, the SHR mechanism would become ineffective. However, the SHTR mechanism remains effective under the threat model assumed in this work. This is because SHTR is capable of detecting discrepancies between the output of the Majority Voting unit and that of the monitored PE. Upon detection, SHTR disables the monitored PE and activates a backup PE. The system continues to function until all backup PEs are exhausted. In this context, since the attacker is assumed to only target the Majority Voting unit and cannot compromise other processing units, the data generated by the backup PE remains correct, thereby attacking on Majority Voting unit cannot cause serious impact.

To make SHTR ineffective, an adversary would need to compromise multiple components of the CNN hardware accelerator, including both the PEs and the Majority Voting unit. This significantly increases the complexity of executing a

successful attack, which can be seen as enhancing the overall robustness of the hardware accelerator.

Table 5 Comparison of overheads of accelerators, with or without protection

Network Type	Area overheads (ALM) (Speed Limit Signs CNN / GTSRB CNN)						
	Original	Majority Voting	With SHR	With SHTR	Majority Voting Increase	With SHR Increase	With SHTR Increase
ShuffleNet V2	78188/ 84478	156309/ 168889	85677/ 91967	78316/ 84606	99%/ 99%	10%/ 9%	0.2%/ 0.2%
MobileNet V2	60404/ 66694	120741/ 133321	66198/ 72488	60532/ 66822	99%/ 99%	10%/ 9%	0.2%/ 0.2%
GhostNet	36404/ 42694	72741/ 85321	39085/ 45375	36532/ 42822	99%/ 99%	7%/ 6%	0.4%/ 0.3%

5 Conclusion

This work presents the first large-scale study of the impact of function tampering HTs insertion on CNN accelerators, with a specific focus on those that use approximate commuting techniques, which are prevalent in embedded applications. The work investigates three main types of such networks, MobileNet V2, ShuffleNet V2, and GhostNet, which have been trained in the grayscale version of whole dataset of GTSRB and speed limit sign images subset of GTSRB. Next, hardware accelerators of these designs were developed using System Verilog. The work then proceeded to develop a unique hardware Trojan for each layer of the network, such as it can feasibly be inserted by an attacker. Next, a comprehensive experimental analysis was carried out to determine the insertion locations in each hardware architecture that causes the largest drop in classification accuracy. For the network under consideration, there is found to be a specific set of procession elements, which we have referred to as "important", in the classification, Relu6, and Max pooling layers. These findings have subsequently been used to develop two countermeasures, the first relies on hardware redundancy (SHR), and the second on a combination of hardware and time redundancy(SHTR). Such techniques are only applied to the most vulnerable points in each architecture to reduce overheads. The two proposed defenses were evaluated in terms of security, attack detection/recovery time, and area overheads. The results show that the SHR provides speedy recovery from an attack while incurring between 10-6% area overheads. On the other hand, SHTR requires more time to recover the impact caused by HT, but its area overhead is much smaller (~ 0.3%). In addition, when the abilities of attackers are limited, the performance of SHTR is better than SHR, in terms of the possibility of images being misclassified when HTs are injected into the accelerators. Future research will focus on other types of hardware-level attacks, such as fault injections.

Declarations

Ethical Approval There was no involvement of humans or animals in this study. We give consent to Springer to publish this paper.

Competing interests The authors declare no competing interests.

Authors' contributions P.S. and B.H. wrote the main manuscript. B.H. and T.J.K. reviewed the manuscript and proposed the critical suggestion.

Funding Not applicable

Availability of data and materials Not applicable

References

1. K. O'Shea and R. Nash, "An introduction to convolutional neural networks," arXiv preprint arXiv:1511.08458, 2015.
2. N. Abderrahmane, E. Lemaire, and B. Miramond, "Design space exploration of hardware spiking neurons for embedded artificial intelligence," Neural Networks, vol. 121, pp. 366–386, 2020.

3. R. Struharik, B. Vukobratovic´, A. Erdeljan, and D. Rakanovic´, “Conna– compressed cnn hardware accelerator,” in 2018 21st Euromicro Conference on Digital System Design (DSD). IEEE, 2018, pp. 365–372.
4. C.-Y. Chen, J. Choi, K. Gopalakrishnan, V. Srinivasan, and S. Venkatara- mani, “Exploiting approximate computing for deep learning acceleration,” in 2018 Design, Automation & Test in Europe Conference & Exhibition (DATE). IEEE, 2018, pp. 821–826.
5. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, “Mobilenets: Efficient convolutional neural networks for mobile vision applications,” arXiv preprint arXiv:1704.04861, 2017.
6. M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, “Mobilenetv2: Inverted residuals and linear bottlenecks,” in Proceedings of the IEEE Conference on computer vision and pattern recognition, 2018, pp. 4510–4520.
7. X. Zhang, X. Zhou, M. Lin, and J. Sun, “Shufflenet: An extremely efficient convolutional neural network for mobile devices,” in Proceedings of the IEEE Conference on computer vision and pattern recognition, 2018, pp. 6848–6856.
8. N. Ma, X. Zhang, H.-T. Zheng, and J. Sun, “Shufflenet v2: Practical guidelines for efficient CNN architecture design,” in Proceedings of the European Conference on computer vision (ECCV), 2018, pp. 116–131.
9. K. Han, Y. Wang, Q. Tian, J. Guo, C. Xu, and C. Xu, “Ghostnet: More features from cheap operations,” in Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, 2020, pp. 1580– 1589.
10. S. Venkataramani, X. Sun, N. Wang, C.-Y. Chen, J. Choi, M. Kang, A. Agarwal, J. Oh, S. Jain, T. Babinsky, et al., “Efficient ai system design with cross-layer approximate computing,” Proceedings of the IEEE, 2020.
11. Y. Wang, H. Li, and X. Li, “Real-time meets approximate computing: An elastic cnn inference accelerator with an adaptive trade-off between QoS and qor,” in 2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC). IEEE, 2017, pp. 1–6.
12. Z. Liu, A. Yazdanbakhsh, T. Park, H. Esmaeilzadeh, and N. S. Kim, “Simul: An algorithm-driven approximate multiplier design for machine learning,” IEEE Micro, vol. 38, no. 4, pp. 50–59, 2018.
13. X. Zhang, J. Wang, C. Zhu, Y. Lin, J. Xiong, W.-m. Hwu, and D. Chen, “Dnnbuilder: An automated tool for building high-performance dnn hardware accelerators for FPGAs,” in 2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD). ACM, 2018, pp. 1–8.
14. Y. Shen, M. Ferdman, and P. Milder, “Maximizing cnn accelerator efficiency through resource partitioning,” ACM SIGARCH Computer Architecture News, vol. 45, no. 2, pp. 535–547, 2017.
15. P. Marwedel, Embedded system design. Springer, 2006, vol. 1.
16. P. Kocher, R. Lee, G. McGraw, and A. Raghunathan, “Security as a new dimension in embedded system design,” in Proceedings of the 41st annual Design Automation Conference, 2004, pp. 753–760.
17. Y. Jin, “Introduction to hardware security,” Electronics, vol. 4, no. 4, pp. 763–784, 2015.
18. Halak, “Cist: A threat modeling approach for hardware supply chain security,” in Hardware Supply Chain Security. Springer, 2021, pp. 3–65.
19. Y. Nozaki, S. Takemoto, Y. Ikezaki, and M. Yoshikawa, “Lut oriented hardware trojan for FPGA based ai module,” in 2020 6th International Conference on Applied System Innovation (ICASI). IEEE, 2020, pp. 46–49.
20. Z. Liu, J. Ye, X. Hu, H. Li, X. Li, and Y. Hu, “Sequence triggered hardware trojan in neural

network accelerator," in 2020 IEEE 38th VLSI Test Symposium (VTS). IEEE, 2020, pp. 1–6.
21. J. Ye, Y. Hu, and X. Li, "Hardware trojan in FPGA CNN accelerator," in 2018 IEEE 27th Asian Test Symposium (ATS). IEEE, 2018, pp. 68–73.
22. T. A. Odetola, H. R. Mohammed, and S. R. Hasan, "A stealthy hardware trojan exploiting the architectural vulnerability of deep learning architectures: Input interception attack (ii)," arXiv preprint arXiv:1911.00783, 2019.
23. Yang, J. Hou, M. Wu, K. Mei, and L. Geng, "Hardware trojan attacks on the reconfigurable interconnections of convolutional neural networks accelerators," in 2020 IEEE 15th International Conference on Solid-State & Integrated Circuit Technology (ICSICT). IEEE, 2020, pp. 1–3.
24. J. Clements and Y. Lao, "Hardware trojan attacks on neural networks," arXiv preprint arXiv:1806.05768, 2018.
25. K. Xiao, D. Forte, and M. Tehranipoor, "A novel built-in self-authentication technique to prevent inserting hardware trojans," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 33, no. 12, pp. 1778–1791, 2014.
26. S. C. Konigsmark, D. Chen, and M. D. Wong, "Information dispersion for trojan defense through high-level synthesis," in Proceedings of the 53rd Annual Design Automation Conference, 2016, pp. 1–6.
27. X. T. Ngo, S. Guilley, S. Bhasin, J.-L. Danger, and Z. Najm, "Encoding the state of integrated circuits: a proactive and reactive protection against hardware trojans horses," in Proceedings of the 9th Workshop on Embedded Systems Security, 2014, pp. 1–10.
28. X. T. Ngo, S. Bhasin, J.-L. Danger, S. Guilley, and Z. Najm, "Linear complementary dual code improvement to strengthen encoded circuit against hardware trojan horses," in 2015 IEEE International Symposium on Hardware Oriented Security and Trust (HOST). IEEE, 2015, pp. 82–87.
29. M. Hicks, M. Finnicum, S. T. King, M. M. Martin, and J. M. Smith, "Overcoming an untrusted computing base: Detecting and removing malicious hardware automatically," in 2010 IEEE Symposium on security and privacy. IEEE, 2010, pp. 159–172.
30. G. Voyiatzis, K. G. Stefanidis, and P. Kitsos, "Efficient triggering of trojan hardware logic," in 2016 IEEE 19th International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS). IEEE, 2016, pp. 1–6.
31. S. Dupuis, M.-L. Flottes, G. Di Natale, and B. Rouzeyre, "Protection against hardware trojans with logic testing: Proposed solutions and challenges ahead," IEEE Design & Test, vol. 35, no. 2, pp. 73–90, 2017.
32. H. A. Amin, Y. Alkabani, and G. M. Selim, "System-level protection and hardware trojan detection using weighted voting," Journal of Advanced Research, vol. 5, no. 4, pp. 499–505, 2014.
33. T. A. Odetola, A. Adeyemo, F. Khalid, and S. R. Hasan, "Fm-mod comp: Feature map modification and hardware-software co-comparison for secure hardware accelerator-based cnn inference," Microprocessors and Microsystems, p. 104827, 2023.
34. M. Javaheripi, M. Samragh, G. Fields, T. Javidi, and F. Koushanfar, "Cleann: Accelerated trojan shield for embedded neural networks," in Proceedings of the 39th International Conference on Computer-Aided Design, 2020, pp. 1–9.
35. Q. Xu, M. T. Arafin, and G. Qu, "Security of neural networks from a hardware perspective: A survey and beyond," in Proceedings of the 26th Asia and South Pacific Design Automation Conference, 2021, pp. 449– 454.
36. P. Sun, B. Halak, and T. Kazmierski, "Towards hardware trojan resilient design of convolutional neural networks," in 2022 IEEE 35th International System-on-Chip Conference (SOCC). IEEE, 2022, pp. 1–6.

37. P. Yellu, M. R. Monjur, T. Kammerer, D. Xu, and Q. Yu, “Security threats and countermeasures for approximate arithmetic computing,” in 2020 25th Asia and South Pacific Design Automation Conference (ASP- DAC). IEEE, 2020, pp. 259–264.
38. P. Yellu, L. Buell, D. Xu, and Q. Yu, “Blurring boundaries: A new way to secure approximate computing systems,” in Proceedings of 2020 on Great Lakes Symposium on VLSI, 2020, pp. 327–332.
39. F. Regazzoni, C. Alippi, and I. Polian, “Security: the dark side of approximate computing?” in 2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD). IEEE, 2018, pp. 1–6.
40. S. Keshavarz and D. Holcomb, “Privacy leakages in approximate adders,” in 2017 IEEE International Symposium on Circuits and Systems (ISCAS). IEEE, 2017, pp. 1–4.
41. S. Vahdat, M. Kamal, A. Afzali-Kusha, and M. Pedram, “Tosam: An energy-efficient truncation-and rounding-based scalable approximate multiplier,” IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 27, no. 5, pp. 1161–1173, 2019.
42. V. Venugopalan and C. D. Patterson, “Surveying the hardware trojan threat landscape for the internet-of-things,” Journal of Hardware and Systems Security, vol. 2, no. 2, pp. 131–141, 2018.
43. S. Bhunia, M. S. Hsiao, M. Banga, and S. Narasimhan, “Hardware trojan attacks: Threat analysis and countermeasures,” Proceedings of the IEEE, vol. 102, no. 8, pp. 1229–1247, 2014.
44. Clements J, Lao Y. Hardware trojan attacks on neural networks[J]. arXiv preprint arXiv:1806.05768, 2018.
45. Xue M, Gu C, Liu W, et al. Ten years of hardware Trojans: a survey from the attacker's perspective[J]. IET Computers & Digital Techniques, 2020, 14(6): 231-246.
46. Yanamala R M R, Pullakandam M. A high-speed reusable quantized hardware accelerator design for CNN on constrained edge device[J]. Design Automation for Embedded Systems, 2023, 27(3): 165-189.
47. He J, Zhang M, Xu J, et al. Optimizing CNN Hardware Acceleration with Configurable Vector Units and Feature Layout Strategies[J]. Electronics, 2024, 13(6): 1050.
48. Rosero-Montalvo P D, Tözün P, Hernandez W. Optimized CNN architectures benchmarking in hardware-constrained edge devices in IoT environments[J]. IEEE Internet of Things Journal, 2024.
49. Zhang C, Li P, Sun G, et al. Optimizing FPGA-based accelerator design for deep convolutional neural networks[C]//Proceedings of the 2015 ACM/SIGDA international symposium on field-programmable gate arrays. 2015: 161-170.
50. Lian X, Liu Z, Song Z, et al. High-performance FPGA-based CNN accelerator with block-floating-point arithmetic[J]. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2019, 27(8): 1874-1885.
51. Wang Z, Xu K, Wu S, et al. Sparse-YOLO: Hardware/software co-design of an FPGA accelerator for YOLOv2[J]. IEEE Access, 2020, 8: 116569-116585.
52. Huang W, Wu H, Chen Q, et al. FPGA-based high-throughput CNN hardware accelerator with high computing resource utilization ratio[J]. IEEE Transactions on Neural Networks and Learning Systems, 2021, 33(8): 4069-4083.