Evaluating Image Reconstruction Attack on FPGA-based CNN Implementation through Power Side Channel Leakage

Ding Dao Xian
School of Electrical and Electronic Engineering

Prof Chang Chip Hong
School of Electrical and Electronic Engineering

***Abstract* –** With the increasing demand for high speed and energy-efficient processing of Deep Neural Networks (DNNs), Field-Programmable Gate Arrays (FPGAs) emerge as a promising solution, offering significant potential for optimizing DNN computations through their parallel processing capabilities, resulting in reduced inference time. Despite its merits, recent studies have shown that conventional implementations of Binarized Neural Networks (BNNs) on FPGAs are vulnerable to Power Side-Channel Attacks. These reports show that once an adversary collects the power trace of an FPGA instance, either remote or on site, they can take advantage of the high correlation between the computation of first convolution layer and the power trace. Due to the line-buffer architecture, adversaries can reconstruct the input image based on the voltage variations. This poses a critical privacy concern when privacy-sensitive applications such as medical image analysis are being inferenced over cloud. In this paper, we evaluate the threat of Power Side-Channel Attacks on Convolutional Neural Networks (CNNs) for image reconstruction and discuss the limitations of existing attack methods. Furthermore, by exploring other designs for CNNs, we show that as research advances towards parallelizing the computations of convolutional layers, this parallelism allows the masking of the power trace despite a line-buffer architecture, thus, rendering existing power side channel attacks ineffective.

Keywords – FPGA-CNN, Power Side-Channel Attack, Image Reconstruction

1. INTRODUCTION

In recent years, the use of Deep Neural Networks (DNNs) has found a wide variety of applications, ranging from autonomous driving to privacy-sensitive applications such as medical image classification. Furthermore, as DNNs become more and more complicated, there is much research being done to accelerate these deep learning models on Field Programmable Gate Arrays (FPGAs) and ASICs, as they consume much lesser power than GPUs while offering higher speed ups than CPUs.

This, combined with the emerging trend of Cloud FPGA (where users pay for access to FPGA resources provided by public cloud providers), has resulted more and more attention on FPGA-based DNN acceleration that allows inference to be completed on the cloud similar to the MLAAS [18]. However, this cloud service model also poses a potential cause for concern when sensitive data is being used over these platforms.

Wei et al [17] has shown that by exploiting the power trace obtained from FPGA implemented Convolutional Neural Networks (CNN), they are able to reconstruct the input image to the network. [17] Furthermore, Moini et al [16] has shown that the adversary needs not physical access to these FPGA instances. Instead, they can use Time to Digital Circuits (TDC) to obtain the power trace remotely and using the same reconstruction technique to recover the input image [16]. Despite the significant impact of this attack to the privacy of users, it raises an important question: How much emphasis should researchers place in developing networks resilient to this kind of attack?

Our paper explores the attack proposed by Wei et al on both Binarized Neural Networks (BNN) and CNNs (Section 2) and explore the assumptions and limitations of their attack (Section 3). Then we explore the current designs of FPGA Implemented CNNs and conclude that as the hardware accelerator designs of CNNs (and BNNs) utilize more computation elements executing in parallel, the parallelism renders these attacks redundant. Thus, the FPGA-based implementation of DNNs become resilient to these image reconstruction attacks.

In summary, we make the following contributions:

(i) We explore in detail the assumptions and limitations of state-of-the-art image reconstruction attacks.

(ii) We explore designs and implementations of FPGA-CNNs and come to the conclusion that research in accelerating DNNs have resulted in by-products that are resilient to the input reconstruction attack. Higher level of computation parallelism not only improves the inference throughputs but also increases the robustness of DNN hardware against power side channel leakage on the input data.

2. BACKGROUND

2.1 Convolutional Neural Networks

CNN is a type of neural network architecture that are widely employed in classifying image, video and other multi-dimensional data [13].

The main feature of CNNs that differs it from previous DNNs is its convolution layer. This convolution layer implements a 2-dimensional convolution process. Since this convolution process was inspired by the visual cortex and is commonly used in image processing [15], the convolutional layer acts extremely well as a feature extractor for multi-dimensional data. This layer takes in 2 inputs: high dimensional data (input feature maps) and the kernel parameter, and outputs a layer (a feature map), filtering noise and transforming a high-dimensional input image to an output feature to be fed to the next layer.

In more detail, we take in the following inputs:

(i) Input feature map (i.e: a 2D-pixel array image): $\mathrm{I}_{x,y}^{j}$ with dimensions: $\mathrm{X} \times \mathrm{Y} \times \mathrm{M}$ where $\mathrm{X} \times \mathrm{Y}$ represents the input image's height and width (or size of feature map) and M represents the number of channels of the input feature maps. (For grayscale, M = 1)

(ii) A kernel: $\mathrm{w}_{x,y}^{i,j}$ with size $\mathrm{K}_x \times \mathrm{K}_y$.

(iii) Other parameters include step size: S_x and S_y, bias values: $\beta^{i,j}$ and non-linear activation function: $f()$ which is usually $\tanh$, $\mathrm{sigmoid}$ and ReLU,

We then compute the output by sliding the kernel across the input image via the convolution window with the given steps. The output feature map $\mathrm{O}_{x,y}^{j}$ can be obtained with the following formula [17]:

$$\mathrm{O}_{x,y}^{j} = f\left(\sum_{i=1}^{M}\left(\beta^{i,j} + \sum_{a=0}^{\mathrm{K}_x-1}\sum_{b=0}^{\mathrm{K}_y-1} \mathrm{w}_{x,y}^{i,j} \times \mathrm{I}_{x\mathrm{S}_x+a, y\mathrm{S}_y+b}^{j}\right)\right)$$

(Note that the notation $\mathrm{I}_{x,y}^{j}$ with subscript x and y denotes the pixel in position (x,y) and the postscript j denotes the j-th feature map. The notation $\mathrm{w}_{x,y}^{i,j}$ has same subscript notation but the postscript denotes "between" the i-th feature map and j-th feature map).

A diagram is provided below to provide an intuitive understanding of the convolution operation when M=1:

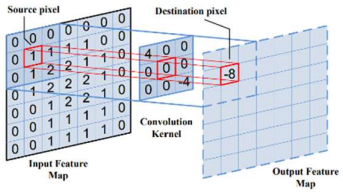

Fig 1: Block Diagram of the convolution operation [17]

In addition to the convolution layer, CNNs consists of other layers, namely: pooling layers and fully-connected layers. The pooling layers are to reduce the dimension of the input feature maps by either limiting the number of neurons or via other normalization methods, thereby reducing the variance in the feature space. Fully connected layers are layers where each neuron is connected with all others in the subsequent layer whilst applying biases due to its point-wise multiplication and activation functions. Due to its complex mapping and the output feature map being a straight vector, fully connected layers are often used as the last layer in classification tasks as most of the feature space has been reduced by the end, allowing a simple classification to be performed [14]. We note that these layers often implement non-linear functions such as $\tanh$ and $\mathrm{sigmoid}$ to create complex mappings between neurons in the current and subsequent layers.

2.2 Binarized Neural Networks

Binarized Neural Networks (BNN) is a subset of CNNs. These networks are aggressive quantized versions of CNNs such that each element of both the convolution kernel and activation functions are constrained to either +1 or -1 [11].

Due to the quantization to binary values, the +1 or -1 are then encoded to 0 for -1 and 1 for +1, allowing multiplication operations to be implemented as XNOR operations on a bitwise level. This provides significant benefit by drastically reducing hardware consumption. This includes reducing the memory storage required [10], reducing memory bandwidth when loading BNN parameters from off-chip DRAM [16] and replacing floating point operations with

binary operations. This aggressive quantization sacrifices accuracy for significant benefits in hardware, allowing applications in IOT and edge inferencing [9].

2.3 Neural Network Accelerator Design

There have been multiple implementations of both CNNs and BNNs targeting FPGA or ASIC. The typical design of the accelerators often follows a similar format. Within the accelerator: the following five units are required: DMA (Direct Memory Access), Data Buffers, Compute Units, Parameter Buffers and Controller. External to the accelerator, often there is a CPU and Off-Chip dynamic random access memory (DRAM) [10, 12]. The DMA is used for communication with the Off-Chip memory and is used in conjunction with the controller for coordination of computation tasks. The data buffers are used for storing input/output feature maps, while the parameter buffers store the weights, with the compute unit is dedicated to performing all the necessary operations like convolution and pooling. The figure below provides a system level block diagram of a BNN implementation:

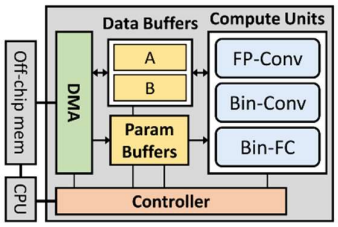

Fig 2: System Level block diagram of FPGA-BNN [10]

For our interest, we explore in more detail the convolution implementation within the compute unit. In order to implement reconfigurable 2D convolution for DSP units, the Line Buffer architecture was proposed [8]. The Line Buffer is essentially a line of shift registers used to temporarily store data to execute 2D convolution and this architecture has been implemented by a number of CNN and BNN accelerators. In these accelerators, at each clock cycle, the line buffer will receive 1 pixel per "block of shift register" (Refer to Fig 3), then a dedicated hardware is used to perform the convolution operation (e.g: for kernel of size 3x3, and M=1, there will be 9 MAC operations per cycle). Once the output feature maps are generated, the data is then stored in the data buffer for preparation for the next round of operation. Then at the next clock cycle, data is then loaded from both the next line of shift registers as well as from the data buffer, and the process repeats. We note that the length of each row of the line buffer is equal to the length of the input feature map. Should there be multiple input feature maps (such as RGB, M=3), then multiple line buffers (3 specifically for RGB) will be parallelly processed at the same time.

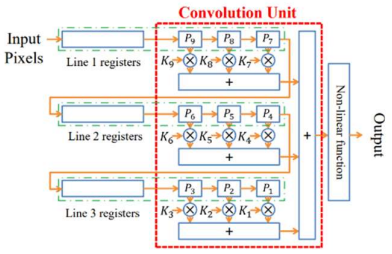

Fig 3: Line Buffer architecture in Convolution Unit of FPGA-BNN [17]

2.4 Power Side Channel Attacks

In the field of Side-Channel Attacks, Power Side-Channel exploits the information provided by observing the power trace left behind by a device while algorithms are being performed on it. This is often successful in cases where insecure hardware implementations result in a strong correlation between the circuit's input and its power consumption.

Traditionally, this exploitation have been used successfully against cryptographic devices. Well known examples include the Differential Power Analysis (DPA) proposed by Paul Kocher [7], as well as the Correlation Power Analysis (CPA) model by Eric Brier [6]. The main intuition of these attacks is that statistical behaviour of the devices are collected beforehand, forming a "Power Template", and from the template, statistical differences (in DPA) or correlation peaks (in CPA) can be observed and extracted for predictions. As a visualization, an output of the CPA attack on an FPGA implemented AES encryption algorithm has been provided in Fig. 4:

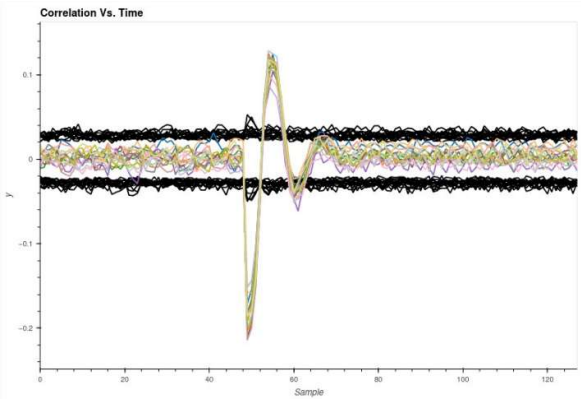

Fig 4: Correlation Peaks as seen from CPA attack on a Chipwhisperer CW305 board.

For the image reconstruction attack, Wei et al showed that the attack is possible with (Active

Adversarial attack) and without a power template (Passive Adversarial attack). [17]

2.5 Image Reconstruction Attacks

In 2018, Wei et al proposed an image reconstruction attack on Neural Network Accelerators. [17]. In essence, their attack is based on the assumption that there is a high correlation between the power consumption and the convolution layer.

In their attack, they stated 2 necessary assumptions for the adversary:

(i) Adversary knows the size of input image: $X \times Y$ and the size of parameters for the first convolution layer (specifically, kernel size: K_x, K_y, number of input feature maps: M, Step size: S_x, S_y and number of output feature maps)

(ii) Adversary can obtain the power trace of a DNN inference. For this assumption, oscilloscope measurements or trojans can be used.

The base power model used is one where: the "power consumption of the device depends on the internal activities". This implies that should the data inside the device (more specifically the convolution unit) stays the same between clock cycles (during loading), then there should not be too much power consumption. However, if the active data comes in, then the power consumption should increase dramatically. Applying this idea to the Neural Network Accelerator, this means that within the convolution unit, if at the current clock cycle, the data inside the line buffer is background, and at the next clock cycle, the data loaded into the unit is also "background", then the magnitude of power consumption should be low. However, if it is from background to foreground or vice versa, then we would expect high power consumption. By monitoring the power consumption in each cycle, we will be able to differentiate between foreground and background and generate a "silhouette" of the image.

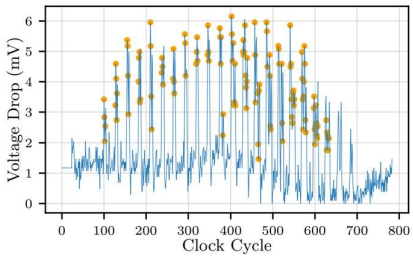

Fig 5: Power Trace of the first convolution unit sampled per unit clock cycle [16]

Combining this concept with the earlier assumption, this means that, weights of the network need not be known, instead only basic parameters relating to the first convolution layer such as image and kernel size are needed to be able to reconstruct the input image. As an example, suppose the MNIST dataset is used as input (28 x 28 x 1) alongside a CNN with 64 kernels $j \in [1,64]$ of size 3 x 3, and step size of 1 with no activation function nor bias. Hence, their output feature map of the j-th convolution layer is a simple:

$$O^{j}_{x,y} = \sum_{a=0}^{2}\sum_{b=0}^{2} w^{j}_{x,y} \times I^{j}_{x+a,y+b}$$

From this output feature map, the user can then reconstruct the image using two of the following angles of approach: Passive and Active.

Passive Attack: This attack assumes that the adversary cannot freely inference the DNN and instead can only obtain the power trace. For this attack, the adversary will need to find a "threshold" (usually via a histogram (Fig. 6) of power consumed per cycle), and this threshold would determine if the current feature map is background or foreground. Once the power trace is obtained, the attacker can reconstruct the image by monitoring each clock cycle and determining whether the pixel is background or foreground based on the power consumption.

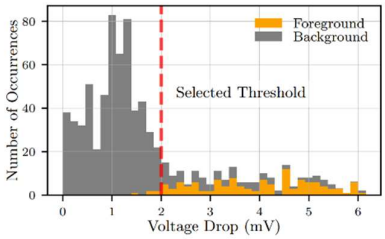

Fig 6: Histogram of the voltage drops [16]

Active Attack: This attack assumes that the adversary can freely inference the DNN. In this attack, instead of separating pixels into "background" and "foreground", the adversary would try to obtain possible pixel values for each image. To this, the attacker would first generate a "Power Template" by storing a map of pixel values to power consumption. Then following the same procedure as the Passive Attack, the adversary would reconstruct the image by monitoring each clock cycle, but instead of choosing "background" and "foreground", they can use the Power Template stated earlier and reconstruct the image from that table.

We note that although all previous experiments were conducted on BNNs, there is no "structural" difference between BNN and CNNs in terms of the architecture (with the exception of some minor differences such as having XNOR-MAC instead of MAC procedures). CNNs also implement the line buffer architecture, thus, this attack is applicable to CNNs as well.

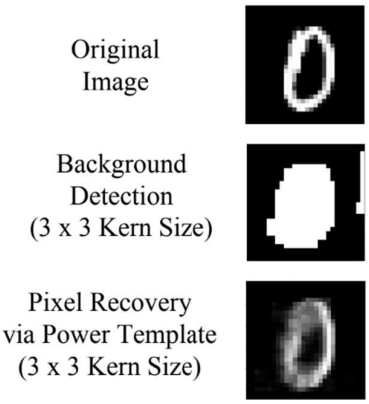

Fig 7: Outputs of the recovered image [17]

3. DISCUSSION ON THE ATTACK

In this section, we discuss some of the assumptions and limitations of the attack listed by the authors, then lead into the next section on the common design countermeasures that are based on these assumptions.

Assumptions: There are multiple assumptions proposed by the author for this attack to be successfully executed.

One important assumption is that the adversary needs to be able to obtain a "high-quality" power trace. This assumption is important not only because a power trace is used as the basis of the attack, but also because without a "high-quality" power trace, noise sources during power extraction can result in imperfect data. Hence, precise measurement is needed for the per-clock cycle measurements. In terms of precision, Wei et al proposed some noise filtering techniques based on power traces collected by an oscilloscope, as well as the "Cycle Power Extraction" algorithm to align each cycle [17]. In terms of applicability, Moini et al has shown that in the realm of Cloud FPGAs, adversaries can create their own circuits using TDCs and collocate the victim's DNN circuits to obtain a power trace, allowing them to bypass the requirement for physical access to the circuit. [16] Furthermore, due to the nature of TDCs, the power trace collected are in terms of "per-clock" (1 measurement per clock cycle), so the "Cycle Power Extraction" algorithm need not apply.

The second assumption is that most of the power consumption arises from the convolution unit. Wei et al showed that after experimenting with different configurations of Image Channel, Line Size and Kernel Size, the convolution unit occupies more than 80% of total power consumption, thus this assumption is extremely valid.

The last fundamental assumption requires the adversary to know the input image and kernel size as well as basic parameters of the first convolution layer. For the input image size, the nature of the input is trivial to know, and as for the other parameters, depending on the task, it is possible to brute force search and check each of the possible values due to the small search space of $(\mathrm{K}_x, \mathrm{K}_y, \mathrm{M}, \mathrm{S}_x, \mathrm{S}_y)$.

Limitations: For passive attacks, the attack can only separate between foreground and background, resulting in a limited use-case. For inputs where the background is noisy or messy, this can result in failure to recover the image, hence only images with uniform background are useful. Furthermore, if the number of background pixels is lesser than foreground, a threshold may be difficult to find when plotting the histogram, resulting in failure as well. For active attacks, the power template solves the issue due to the more detailed output of the mapping, tolerating noisier backgrounds, and further allowing other datasets than MNIST. However, the pixel-level accuracy is still limited. Specifically, Wei et al was able to recover images from DDSM (Digital Database for Screening Mammography), but only the silhouette was able to be recovered, whilst important features of the breasts were noisy.

Additionally, the attack is sensitive to the choice of parameters for the first convolution layer. The choice of kernel sizes: $\mathrm{K}_x, \mathrm{K}_y$ and step sizes: $\mathrm{S}_x, \mathrm{S}_y$ can also impact the recovery with larger kernel and step sizes resulting in a much lower pixel level accuracy, albeit usually at the cost of classification accuracy. The number of input feature maps: M also affect the results. For RGB images, much more data is needed during the construction of the power template due to the high number of possibilities for pixel value at each value of M. This results in a more complex image reconstruction process and requires more effort.

Hence from the assumptions and limitations, we see that despite the limitations on the use cases, with some effort, the assumptions required can be easily fulfilled.

4. DESIGN COUNTERMEASURES

In this section, we assume the scenario where assumptions are fulfilled and limitations are not an issue, and consider countermeasures based on the designs of the FPGA-CNN that are independent on the parameters of the convolution layer chosen by the user. We initially explore the common design countermeasures to side-channel attacks (4.1), then we discuss some of the finer details of the attack and what other relationship can be exploited (4.2). Then in the following sections, we deliberate on how some implementations of the convolution layer can create designs resilient to the reconstruction attack. We also evaluate the following designs with the PPA (Power, Performance, Area) metric and highlight the specific trade-offs needed for each design.

4.1 Common Side-Channel Resilient Design

For power-side channel attacks, the effectiveness of the attack depends heavily on the power trace obtained by the adversary. Hence, there is high priority in developing designs that reduce this dependency. The most intuitive method of doing this is by adding noise to the channel, and two most commonly used methods of implementation are either: Implementing it in the algorithm (Masking) [5] or by generating noise independent to the system (random noise generation) [4]

Masking: In Masking (also called secret-sharing), the idea is to reduce the dependency between power trace and input data by randomly splitting the intermediate variable during computation such that this intermediate result cannot reveal sensitive information, and further encoding (or called masking) these intermediate result such that the result can only be obtained by combining all these shares. [5]. A possible design of this is by adding a random value to each input pixel (only the ones that are being convoluted with) to act as the encoder. Note that the random values are generated such that they are complement to one another, allowing their sum to be zero. Thus, once the convolution operation is performed, at the end when the result of the point-wise multiplication is summed, the random values will sum to zero, thus masking the input image without changing the output data. This however requires a known source of randomness and requires invasive design changes to the convolution modules and incurs additional cost. Specifically, there is significant area-time-memory costs. Furthermore, if the amount of bits/information that is being masked is little (such as for a 3x3 kernel, only 9 values need to be masked), it may be vulnerable to probing attacks and can be brute forced [5].

Random Noise Generation: In random noise generation, the idea is to generate random noise independent to the circuit implementing the algorithm. This random noise generated will incur additional power consumption, thus superposing the power trace with the noise's power, resulting in a noisy power trace. [4]. However, a critical flaw is that the circuits required to generate such noise to sufficiently hide the power consumed by internal activities is power hungry. For instance, to protect an AES circuit from CPA, the noise generator incurred 4 times more energy than the AES circuit [3], which defeat the purpose of energy-efficient hardware acceleration. For our case, since only the first convolutional layer is important, the random noise generator could be activated only when the first layer is activated and be switched off in subsequent layers, thus saving power. Nevertheless, this still incurs extra hardware resources.

4.2 Other Dependencies

Instead of using "general" power side-channel resilient designs, we can exploit relationships between the image reconstruction technique and the design of the convolution layer.

The first relationship/dependency that the image reconstruction technique relies on is the line buffer implementation (i.e: a line buffer architecture must be implemented in the convolution unit). This implies that the convolution unit should be implemented with the design where the kernel slides across for each clock cycle. This is often not an issue, because most convolution units have hardware (area) constraints due to the input feature space having relatively large dimensions (even for the 28 x 28 MNIST Dataset).

The second dependency is that the convolution unit needs to be implemented in a simple and predictable way where only 1 kernel is used to slide throughout the entire input feature map in a known order of sliding (such as left to right, up to down, etc).

By taking advantage of these 2 dependencies, additional flaws in the attack can be found.

4.3 Counter 1: Random Scheduling

A proposal that Wei et al proposed was to perform random scheduling. Essentially this takes advantage of the second dependency where the direction of the kernel slides needs to be known. By randomizing this schedule, adversaries do not know which position: (x, y) the convolution operation is current at, so passive adversaries are not able to reconstruct the image as they do not know which

pixel to start with, and active adversaries cannot create a power feature vector in the first place as they do not know which pixels values they sample from. [17]

This method breaks the normal data flow and impedes the data reuse for accelerating the processing and saving power from off-chip memory access. It also requires additional hardware resources as additional data buffers, randomizers and accumulators are needed to be integrated into the circuit. Furthermore, the design of such a circuit may be difficult as considerations to the logic is needed, making this countermeasure difficult and an unattractive option.

4.4 Counter 2: Parallel Convolution

A simple countermeasure, and an already implemented feature across multiple current FPGA-CNN design is the parallelization of the computing for convolution layer. There are multiple ways of doing parallel convolution, but in essence, this often requires creating multiple copies of the kernel to performing parallel convolution operations across the input image.

From our research [20 – 26], we see that the most common approach to doing this is via data partitioning of the input data. In this scenario, the input data is partitioned into different groups, and each group has a kernel where the convolution operation is performed across the group:

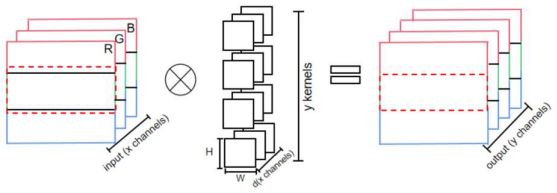

Fig 8: Data Partitioning using y number of kernels [2]

Then to preserve the logic, the convolution of the data in between groups is performed afterwards, and then summed to recover some of the “missing pixels” in the output feature space. We note that sometimes, this step is skipped. The drawback to such methods of parallelism is that since the kernel is duplicated and not partitioned, there is impacts to the memory footprint [2]. Another method of implementation is via group convolution. From the repositories, we have not seen anyone implementing this methodology due to the difficulty as well as the loss in accuracy [12, 20-26]. (Figure 10 is provided in the appendix for group convolution).

This works as a countermeasure, in a similar way to noise generation. As two or more kernels (and by extension, convolution operations) are operating at the same time, the power generated are summed and “superposes” onto each other, effectively masking the power consumption. Furthermore, since the power generated are superposed, this makes it difficult to separate the power traces into their individual parts as well.

Often, as a layer becomes more and more parallelized, there is a trade-off between area and time (measured in number of clock cycles) required. However, for the purpose of creating a resilient design to the image reconstruction attack, only two kernels are theoretically needed for parallel processing to masking the power trace.

Due to the simplicity of parallel convolution, it has seen widespread use across engineers designing FPGA-CNNs and is implemented in all of the git repositories that we have surveyed [20 – 26]. Despite it is originally used for acceleration, this design has yielded resilience to the image reconstruction attack, making most current implementations of FPGA-CNNs resilient to such attacks.

4.5 Counter 3: Combinational Convolution

Another possible countermeasure is to not implement the line buffer for the convolution unit. The line buffer is the most common implementation for the 2D convolution unit. This is because following the system level architecture design in Section 2.3, we wish for the compute unit (especially for convolution) to be a separate, modular, reusable unit.

As seen in Bosi et al [8], a 3x3 convolution unit can be implemented with the following design where a Control Unit is implemented as a finite state machine with a counter and a comparator to keep track of the convolution operations, and the line buffer is used to temporarily store the input pixel values. Furthermore, by incorporating design procedures by Landeta and Malinowski [1], it is possible to use this unit to create arbitrary size convolutions. For instance, by using 4 of these 3x3 convolution units, a 5x5 convolution unit can be implemented (Figures 11, 12, 13, 14 are provided in the Appendix for a more detailed visualization of this procedure).

Fundamentally, such an architecture is chosen in the first place not only due to its elegance, but also to hardware (area) constraints. Supposing there was no such constraint, and if all the user cares about is having the convolution unit perform the operation as quickly as possible, then a purely

combinational design will perform much better, removing the need for a line buffer architecture within the convolution compute unit.

Depending on the CNN design and physical limitations, several system designs can be explored. For instance, if all the convolution layers utilize the same parameters, then a single convolution compute unit can be used, but we note that this is often not the case.

In a recent FYP (Final Year Report) paper named "AcceLeNetor" by Wang [19], a LeNet-5 architecture was implemented with purely convolutional circuits in the convolution layers. Since the LeNet-5 architecture was chosen, there were only 3 convolutional layers. Furthermore, in his design, all the convolution units were of kernel size 5x5 with step sizes of 1, with the same input and output feature space size, and no biases. Additionally, all data was stored with a bit width of just 8 bits.

The design was implemented as a 3-stage pipeline structure where an average of 1 clock cycle is taken for each convolution procedure (not that this is not precisely the case for the design, so the term average is used), allowing the accelerator to complete inference in just 3 clock cycles.

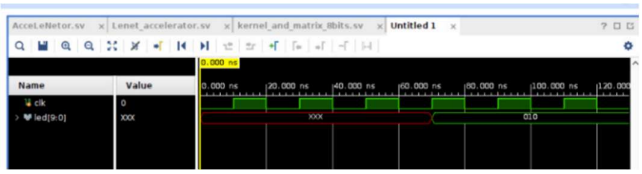

Fig 9: Testbench simulation of AcceLeNetor:

In terms of hardware cost, on an Intel Cyclone V SOC (5CSXFC6D6F31C6), the resource utilization was at 75%, whereas for a xc7a100tftg256-2 Xilinx FPGA, resource utilization was around 90%. Furthermore, for a bit width of 16 bits, the design failed to compile for both FPGAs due to the large size of the design. However, we also note that this design may not be a good comparison for how much hardware resource a non-line-buffer convolution implementation uses. In this design, the author stored the input images and weights as a register file instead of having it off-chip, so the significant hardware overhead could be from this rather than the non-line-buffer convolution implementation use. In terms of recognition accuracy, on the MNIST dataset, it was able to achieve an accuracy of 96.99%.

In general, combinational convolution circuits can be used to resist the image reconstruction attack, and there are areas of exploration for combinational convolution, especially for hardware acceleration. However, this would come at a cost of reconfigurability and high amounts of hardware resources.

5. CONCLUSION

In conclusion, we can see that the image reconstruction attack, as novel as it may seems, has some limitations. Not only are there limitations based on the user's choice of CNN parameters, but also there are additional dependencies that result in basic CNN design choices, that can allow engineers to deter this attack. Such designs can range from complicated designs to protect from general power side channel attacks, to the basic yet extremely effective (and efficient) parallel convolution method. Thus, resulting in most CNN designs unintentionally, yet in a lucky coincidence, being resistant to these attacks.

ACKNOWLEDGEMENT

I would like to express my appreciation to my supervisor, Prof Chang Chip Hong, for giving me the opportunity to work on this project and explore this area of research. His vision, and support has provided me with the valuable opportunity to explore the research methodology.

Additionally, I would like to express very much gratitude and my heartfelt thanks to my mentor, Dr Liu Wenye. His constant guidance has helped me surmount difficulties, navigate through challenging situations, and steer away from potential roadblocks. His continual mentorship, and support have not only to bring this project into fruition, but also ensure an enjoyable research experience.

I would also like to acknowledge [20, 21, 22, 23, 24, 25. 26] for providing their source codes when exploring FPGA-CNN designs. Furthermore, I would also like to accredit Wang Di from the School of Electrical and Electronics Engineering at NTU for implementing one of the accelerators used for exploration as part of his Final Year Report.

I would like to acknowledge the funding support from Nanyang Technological University – URECA Undergraduate Research Programme for this research project.

APPENDIX

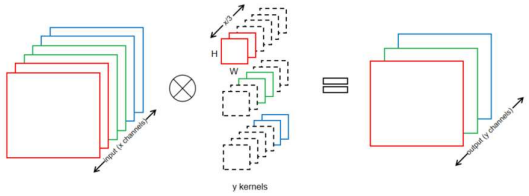

Fig 10: Data Partitioning using y number of kernels [2]

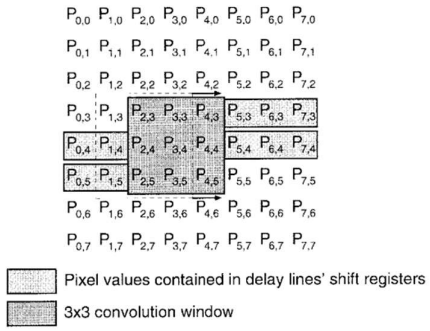

Fig 11: Example of a 3x3 convolution window where $P_{x,y}$ denotes the pixel value of the input data [2]

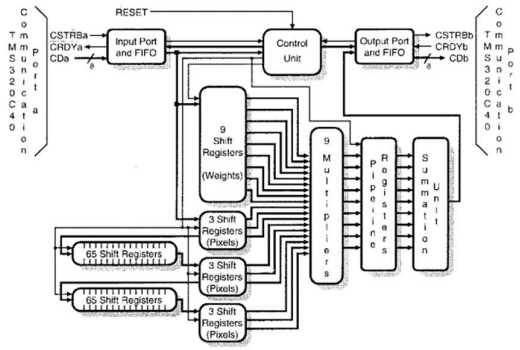

Fig 12: Block Diagram of a 3x3 Convolution Unit [2]

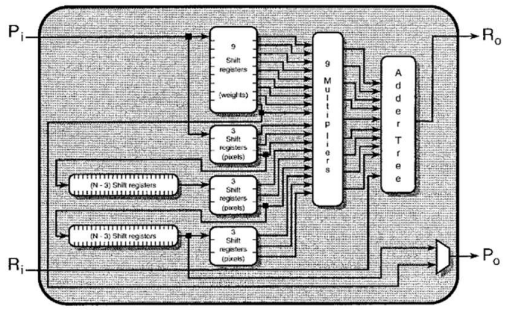

Fig 13: A Simplified Block Diagram of the 3x3 Convolution Unit above [2]

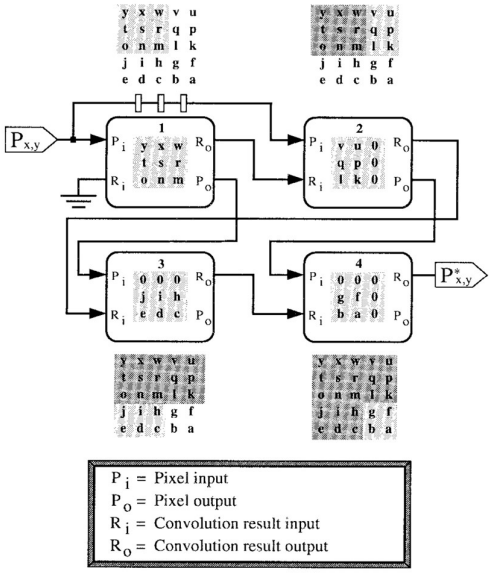

Fig 14: A 5x5 Convolution Unit implemented using the four 3x3 Convolution Units show in Fig 13 [2]

REFERENCES

[1] D. Landeta and C. W. Malinowski, "A 2-D convolver architecture for real-time image processing," *SPIE Proceedings*, 1989.

[2] J. Wang, W. Tong, and X. Zhi, "Model parallelism optimization for CNN FPGA accelerator," *Algorithms*, vol. 16, no. 2, p. 110, 2023.

[3] D. Das *et al.*, "High efficiency power side-channel attack immunity using noise injection in attenuated signature domain," *2017 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, 2017.

[4] A. Shamir, "Protecting smart cards from passive power analysis with detached power supplies," *Cryptographic Hardware and Embedded Systems — CHES 2000*, pp. 71–77, 2000.

[5] E. Prouff and M. Rivain, "Masking against side-channel attacks: A formal security proof," *Advances in Cryptology – EUROCRYPT 2013*, pp. 142–159, 2013.

[6] E. Brier, C. Clavier, and F. Olivier, "Correlation Power Analysis with a leakage model," *Lecture Notes in Computer Science*, pp. 16–29, 2004. doi:10.1007/978-3-540-28632-5_2

[7] P. Kocher, J. Jaffe, and B. Jun, "Differential Power Analysis," *Advances in Cryptology — CRYPTO' 99*, pp. 388–397, 1999.

[8] B. Bosi, G. Bois, and Y. Savaria, “Reconfigurable pipelined 2-D convolvers for Fast Digital Signal Processing,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 7, no. 3, pp. 299–308, 1999

[9] H. Yu, H. Ma, K. Yang, Y. Zhao, and Y. Jin, “DeepEM: Deep Neural Networks model recovery through EM side-channel information leakage,” *2020 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, 2020.

[10] R. Zhao *et al.*, “Accelerating binarized convolutional neural networks with software-programmable fpgas,” *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2017.

[11] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, “Binarized Neural Networks: Training deep neural networks with weights and activations constrained to +1 or -1,”

[12] S. Liang, S. Yin, L. Liu, W. Luk, and S. Wei, “FP-BNN: Binarized Neural Network on FPGA,” *Neurocomputing*, vol. 275, pp. 1072–1086, 2018

[13] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Frechen: MITP, 2018.

[14] L. Batina, S. Bhasin, D. Jap, and S. Picek, “{CSI} {NN}: Reverse engineering of neural network architectures through electromagnetic side channel,”

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

[16] S. Moini, S. Tian, D. Holcomb, J. Szefer, and R. Tessier, “Remote Power side-channel attacks on BNN accelerators in fpgas,” *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2021.

[17] L. Wei, B. Luo, Y. Li, Y. Liu, and Q. Xu, “I know what you see,” *Proceedings of the 34th Annual Computer Security Applications Conference*, 2018.

[18] M. Ribeiro, K. Grolinger, and M. A. M. Capretz, “MLaaS: Machine learning as a Service,” *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*, 2015.

[19] D. Wang, “Accelenetor: FPGA-accelerated neural network implementation for side-channel analysis,” Home, https://dr.ntu.edu.sg/handle/10356/166976

[20] Cornell-Zhang, “Cornell-Zhang/BNN-FPGA: Binarized convolutional neural networks on software-programmable fpgas,” GitHub, https://github.com/cornell-zhang/bnn-fpga

[21] AniketBadhan, “Aniketbadhan/convolutional-neural-network: Implementation of CNN using Verilog,” GitHub, https://github.com/AniketBadhan/Convolutional-Neural-Network

[22] cedard234, “CEDARD234/accelenetor: FYP project. A VERILOGHDL based hardware accelerator.,” GitHub, https://github.com/cedard234/AcceLeNetor

[23] sandy2008, “SANDY2008/CNN-FPGA,” GitHub, https://github.com/sandy2008/CNN-FPGA

[24] Omarelhedaby, “Omarelhedaby/CNN-FPGA: Implementation of CNN on ZYNQ FPGA to classify handwritten numbers using MNIST database,” GitHub, https://github.com/omarelhedaby/CNN-FPGA

[25] MasLiang, “Masliang/CNN-on-FPGA: FPGA,” GitHub, https://github.com/MasLiang/CNN-On-FPGA

[26] padhi499, “PADHI499/image-classification-using-CNN-on-FPGA: Project is about designing a trained neural network on FPGA to classify an image input using CNN.,” GitHub, https://github.com/padhi499/Image-Classification-using-CNN-on-FPGA