Master's Thesis - Preliminary Report

Deep neural network reverse engineering through side channel information leakage

Author :
Thomas BUZER

Supervisors :
Lejla BATINA (Radboud University)
Cédric MARCHAND (Centrale Lyon)

April 19, 2022

Contents

Abstract

Deep neural networks (NNs) have shown tremendous progress in various applications. From self-driving cars to image recognition or natural language processing, they are being integrated in every domain to develop autonomous devices. With that integration comes the effort to implement NNs in efficient platforms dedicated to it. FPGA or GPU have been used to improve time and energy used by these networks but they rose the problem of side-channel information leakage. Several studies have demonstrated hardware vulnerability for both the security of the computation and the intellectual property (IP) of the NNs implementations. For example, ElectroMagnetic (EM) emissions of a data processing unit can leak depth, layer size or layer type of a NN. This work will examine the possibilities of recovering both the architecture and weights of NNs running on hardware accelerators through electromagnetic information leakage. It will be based on both Batina et al. [Bat+19b] and Yu et al. [Yu+20b] works. Divided in two steps, the attack method relies on the acquisition of information about the architecture of the NN through side-channel leakage. The second step consists of retraining the NN with well chosen adversarial training example. The results reveal the ability of reconstructing deep NN and show the importance of the protections against EM side-channel information leakage.

Introduction

Object recognition, natural language processing, autonomous vehicles or art are domains which have benefited from the development of Neural Networks (NN). These algorithms increased performances across many domains but came with a need of computational power. To counter this need, NN have been deployed on dedicated platforms such as FPGAs, ASICs or GPUs which increase efficiency of the computing chips. The collection of the data, the training time and the development of hardware accelerators pushed vendors to keep their models secret.

Yu et al. [Yu+20a] attacked cloud services offered by IBM or Microsoft by developing a method to effectively construct NNs which have the same performances as the secret version kept on the cloud. For less than 10$, the team used adversarial learning to create their networks with few training data. Recent studies also proved that hardware implementation of NN on hardware accelerators were vulnerable to data leakage through Side-Channel Attack (SCA). Even in a black-box methodology, the architecture and weights of the NNs can be reverse engineered. For example, Hua et al. [HZS18] recovered the architecture and weights of NNs running on FPGAs by memory access SCA. Duddu et al. [Vas+19] recovered layer's depth through timing SCA and used reinforced learning to find the best substitute model with performances similar to the attacked NNs. Some protection can be implemented against the access to memory or cache by adversaries. Batina et al. [Bat+19b] used non-intrusive Electromagnetic (EM) SCA to access the weights and characteristics of small NN. Giving the amounts of parameters on recent NN such as ConvNet [Zha+17] or VGGNet [Con+16], this attack is not practical to effectively recover deep NN. To solve this problem, Yu et al. [Yu+20b] presented a black-box methodology using EM SCA to infer possible architectures of the networks. After training the remaining possible networks, the research team used adversarial learning techniques in order to efficiently train the best performing architecture to performances similar to the original networks. This attack focuses on binarized NN, with binary weights and activation, which are commonly used for small devices because of their memory size.

This document presents essential concepts needed to understand articles published in this field. It then focuses on the work of Batina et al. [Bat+19b] and Yu et al. [Yu+20b] and explains the functioning principle of the attack methodology developed by the authors.

The master's thesis aims at measuring the information leakage coming from FPGA implementation of neural networks. Focusing on the Xilinc Ultrascale+

ZCU104 platform, simple neural networks will be implemented in order to measure the electromagnetic emissions of the computing chip.

State of the Art

1 Electromagnetic side-channel information leakage

Side-channel Analysis (SCA) exploits weaknesses of the implementations of secure systems. The computation running on any platform result in information leakage as some physical constant change which could be temperature, Electro-Magnetic (EM) field or execution delay. All these techniques were developed to recover private key from asymmetrical cryptography algorithms [JS01], [GMF01]. SCA can recover parts of the private key which reduces the attack complexity. Modern electronics work by the commutation of bits from 0 to 1 and from 1 to zero. These transitions, through the existence of parasitic capacitors, require energy which must be provided by the power source. Due to the architecture of the logic gates, the energy required to charge the parasitic capacitors is higher than the one to discharge them. As any movement of charge is accompanied by an EM field, both these transitions create EM emission which are slightly different. With small antennas carefully placed on the processing chip, one can corelate the signal from the antenna and the data processed by the chip. These attacks are non-intrusive leaving no traces of the robbery. The Figure 1 shows the experimental setup used in by Batina [Bat+19a] to recover the result of a multiplication inside a microcontroller.

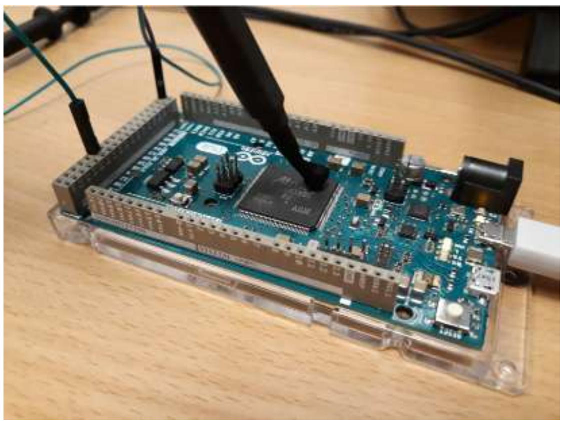


Figure 1: Experimental setup of SCA on ATmega2560 of an Arduino Mega [Bat+19a]

1.1 Simple Electromagnetic Analysis (SEMA)

SEMA is the most basic form of SCA. It extracts information from a single trace sensible to the computation inside the chip. SEMA is easily used on the AES algorithm to distinguish the four different phases of the algorithm and recover the key used.

1.2 Differential Electromagnetic Analysis (DEMA)

DEMA uses statistical techniques to recover data from physical measurements. It compares measurements from the actual physical value to a prediction which has been computed using intermediate data of key hypothesis. This technique allows the adversary to test small parts of the solution (usually a private key) which has been recovered using the Hamming weight for microcontrollers and the Hamming distance for FPGA or GPU. The noisy measurements means that the adversary often needs millions of measurements.

1.3 Correlation Power Analysis (CPA)

CPA [BCF04] consists in the prediction of the variation of a physical value for a range of expected computation executed by the chip. The correlation between all these predictions and the real changes are then evaluated. Usually, CPA is a powerful tool to find the computation made by the chip.

1.4 Data recovered

During operation, the hamming weight, which is the number of '1' in a byte, can be deduced from the loading time of the data bus. As each '0' and '1' require different energy, it is possible to calculate the number of '1' in the data which is going through the bus. If the data going through the bus is a part of the private key of some cryptography algorithm, the attacker can recover sensible information weakening the strength of the overall security.

2 Deep Neural Network recovery

2.1 Principle of neural networks

Neural Networks (NN) are computer systems inspired by biological neural networks. These systems can learn from a great number of training examples.

Using nodes called neurons, the NN connects number of nodes together using various strategies to create complex architectures able to solve many different problems. Each node is connected to a number of other neurons as inputs, the signal carried by this node is a non-linear function of the sum of the inputs. These sums and signals have different weights which are tuned during the learning process.

The simplest architecture of NN is called a Mutli-Layer Perceptron (MLP). This NN consists of an input and an output vector and several hidden layers. Each neuron of the network is linked to the previous layers via a set of weights which are compared to a bias value to determine if the output of the neuron is 1 or -1. To train the MLP, a database of input/output combination is created in order to compare the output of the current MLP to the expected output. The difference between these outputs is used to modify each weight of the networks through a backpropagation algorithm. The architecture of a MLP with two hidden layers is shown on the Figure 2.

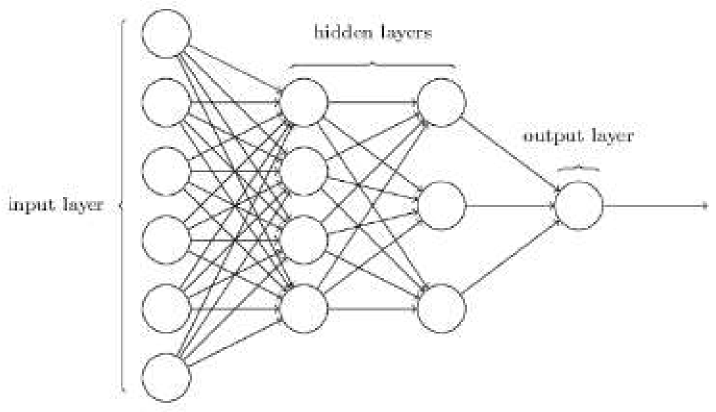


Figure 2: Architecture of a MLP with two hidden layers [Bat+19b]

MLP are only made of fully-connected layers but there are also other kind of layers with different goals. Convolution layers extract sub-features of input data and apply filters and pooling layers globally average to reduce spatial dimension.

NN used for the attacks in the paper from Yu et al. [Yu+20b] are Binary Neural Networks. These are NN with weights and parameters which have been reduced to either -1 or 1. These networks are used because of their memory optimization and ease of computation. -1 values are coded with a 0 bit and 1 are coded with 1. This allows the computation to use a bitwise XNOR operation instead of the decimal addition.

2.2 Goal of the recovery

The training of NNs is long and computationally intensive, it also often requires human supervision. This leads to a high cost of the training of complex NNs. The aim of the recovery is to construct a new NN which have comparable performances as the original one. The cost of the recovery is often much cheaper than the original training. Yu et al. [Yu+20a] showed their ability to produce a NN with similar performances up to 99% of the original networks with less than 10$. Their attack targets NN based on the cloud via Microsoft or IBM services.

2.3 Adversarial attack method

Adversarial Learning (AL) aims at selecting special examples for training in order to optimize the learning process. It relies on finding examples closer to the decision boundaries. New attacks like Feature Fool generate their own image pool, the idea is that examples carefully chosen extract more information from the attacked networks. The new network does not necessarily have the exact same architecture depending on the information acquired through other means. Yu et al. [Yu+20a] did not have access to any information about the architecture of the original networks so they used a general pre-trained image recognition NN.

3 Methodology of the attack

3.1 Threat model

The threat model is the description of the victim NN. F_v is the victim NN with domain input $X \in \mathbb{R}^n$ and output $Y \in \mathbb{R}^m$. The attacker aims at creating a new NN F_s which has similar performances as F_v which means $\forall x \in X, F_v(x) \approx F_s(x)$. In this attack scenario, the attacker only has access to EM information leakage and can control the input of the NN. It is assumed that the attacker has no prior knowledge of the NN and is working on a black-box methodology. The accelerator implementation of the NN is supposed to be vulnerable to EM SCA. Since defence mechanism against information leakage is not wide-spread, this assumption is usually true.

3.2 Test setup

In their articles, Batina et al. [Bat+19b] and Yu et al. [Yu+20b] recorded the EM field thanks to a near-field antenna connected to an oscilloscope which was synchronised with the processing unit in order to easily identify the different phases of the computation. Attacks have been performed on the microcontrollers ATmega328P and ARM Cortex-M3 and the FPGA ZYNQ XC7000.

3.3 Activation function recovery

Study has been conducted on the processing time delay introduced by the activation function of a neuron by Batina et al. [Bat+19b]. The Figure 3 shows the delay (in ns) induced by the activation function for four different common ones. The test has been made when the input was ranging in the $[-2; 2]$ interval. It clearly shows a pattern which allows an attacker to recognise the function used if he has control over the input of a neuron.

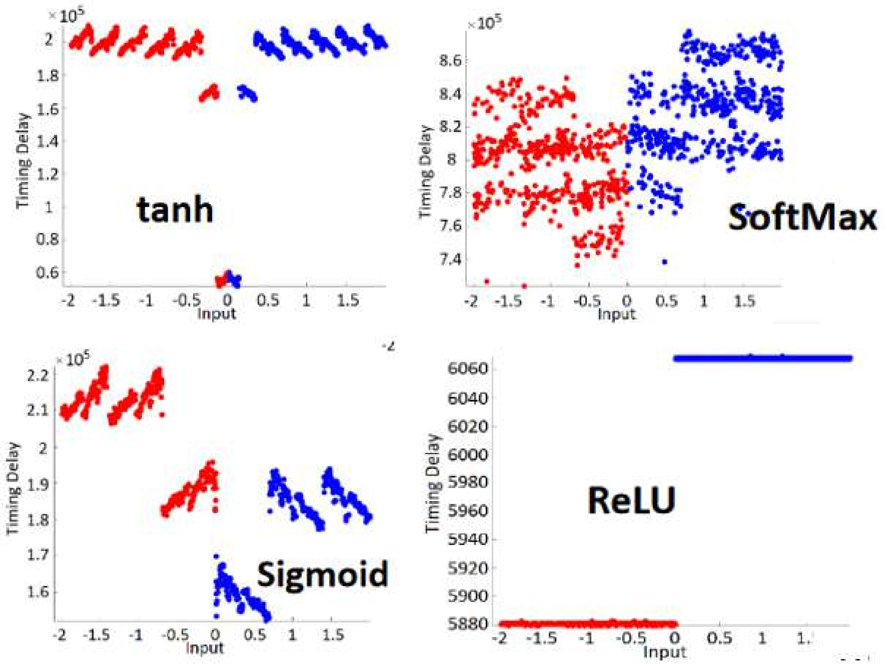


Figure 3: Influence of activation function on processing delay [Bat+19b]

3.4 Weights

The real numbers used in NN are usually 32 bits floating point integers which consists of a sign bit, 8 exponent bits and 23 mantissa bits shown on the Figure 4. Noting $b_{31} \ldots b_0$ the bits of the stored number a, the value of this number can be recovered with this formula:

$$a = (-1)^{b_{31}} \times 2^{(b_{30} \ldots b_{23})_2 - 127} \times (1.b_{22} \ldots b_0)_2 \tag{1}$$

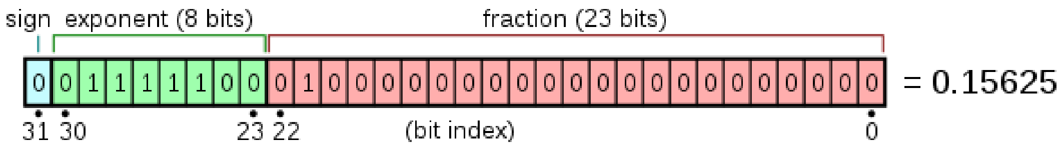


Figure 4: Example of 32 bits float : `en.wikipedia.org/wiki/IEEE_754`

In order to recover the weights, the attacker can focus on the multiplication between the known input x and the hidden weight w. Correlation between the SCA and the guessed weight are performed with each weight possibility. To reduce the search space, the attacker assumes that the weights are in the $[-N; N]$ interval, N being estimated by the attacker as the maximum weight value of any given NN. To further reduce the search space, the attacker defines a precision p with which the weight will be estimated. The number of weights to test is then limited to $s = 2 \times N/p$. When working with mantissa only recovery, the sign is not taken into account which means that all negative weights are not tested separately. The Figure 5 from Batina et al. [Bat+19b] shows for each byte of the mantissa the correlation of the different w with the SCA measured traces. It shows in black the weight guessed against all other weights tested in red. In this article, the correlation has been determined with 1000 different random x inputs. The corelation spikes up to 0.4 with the right value of the weight when it stays below 0.25 (values for the first byte of the mantissa). The overall low correlation is due to the noisy measurements. The distribution of the correlation spikes in the time domain can be attributed to the difference of execution time across the different inputs. After recovering the mantissa of the weight, the same attack can be performed on the sign and exponent part of the float. These figures clearly show the ability of the attack to distinguish the correct weight from all the other possibilities. SCA attacks on weights of a NN are easier than attacks on cryptographic implementation because the cryptographic keys must be recovered exactly when the real number w can be approximated.

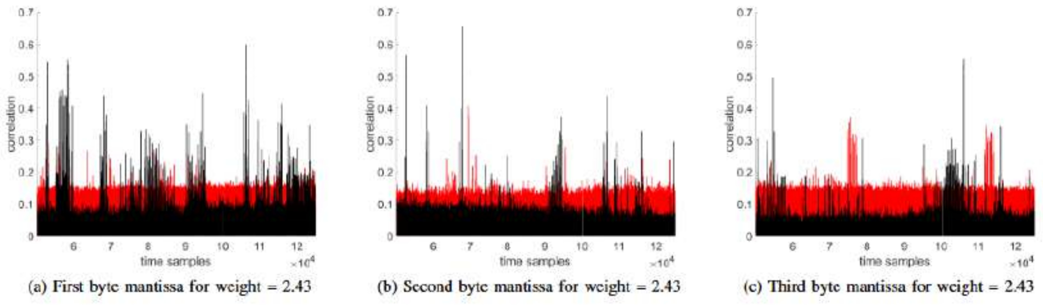


Figure 5: Correlation of different weights candidate on multiplication operation [Bat+19b]

To be more precise, the attacker can focus on the mantissa multiplication between x and w. The operation dealing with 32 bits float, the search space is $[0; 2^23 - 1]$. In order to reduce the search space and the computational power required, the precision of the mantissa recovered by the attacker can be limited to 7 bits instead of 23 which shrinks the search space to $[0; 2^7 - 1]$. The Figure 6 shows the application of this technique in the article from Batina et al. [Bat+19b]. The first figure shows the targeted value being distinguished from the incorrect values. The second one points an error from the recovery of the mantissa weight. The recovered value is 1100100000 when the target was 1100011110. Applying the sign and exponent, the recovered value is 0.890625 when the target was 0.89. The attack successfully recovered the weight to the 4th place digit.

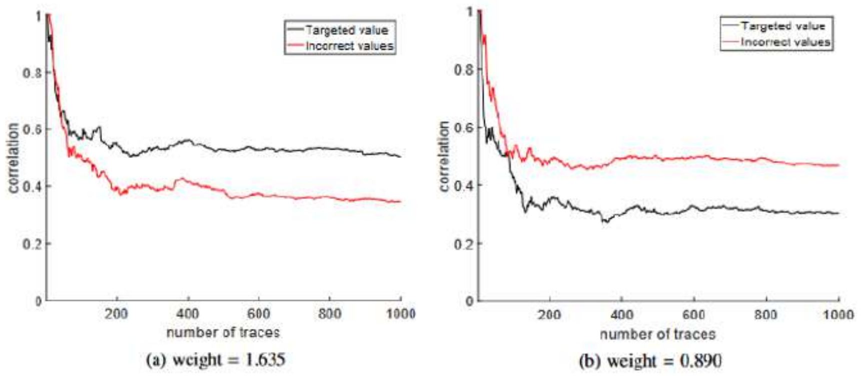
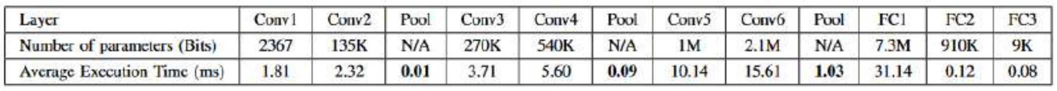


Figure 6: Correlation comparison between correct and incorrect mantissa of the weight [Bat+19b]

3.5 *Architecture of the Networks*

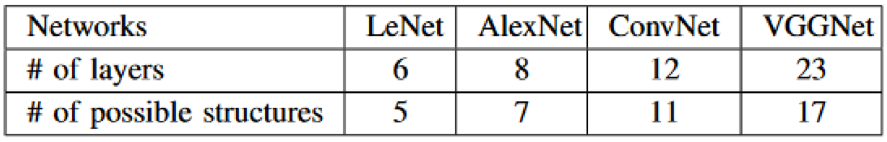
Architecture and filter parameters of the network attacked is mainly recovered through processing time analysis. The article from Yu et al. [Yu+20b] implemented a classical NN called ConvNet on a FPGA and recorded 10k EM traces from the chip. The Figure 7 shows the mean execution time of each layer of the ConvNet network. They found that different layer types have different execution time due to the computation complexity. They also linked execution time to the number of parameters of the layer.



Layer	Conv1	Conv2	Pool	Conv3	Conv4	Pool	Conv5	Conv6	Pool	FC1	FC2	FC3
Number of parameters (Bits)	2367	135K	N/A	270K	540K	N/A	1M	2.1M	N/A	7.3M	910K	9K
Average Execution Time (ms)	1.81	2.32	**0.01**	3.71	5.60	**0.09**	10.14	15.61	**1.03**	31.14	0.12	0.08

Figure 7: Layers parameters configurations and average execution time for ConvNet - N/A indicates that there is no parameter [Yu+20b]

Through EM analysis, the attacker can reverse engineer the architecture of the network. Due to the number of layers and layer types, this analysis leaves some possible combination of layers possible which lead to different architecture possible. Moreover, convolution and pooling filter size cannot be recovered from the noisy EM measurements. To reduce the number of NN possibilities, the attacker can use the relation between consecutive layers or common filter size used in wide spread NN. Using these techniques to eliminate outliers' architectures, the number of possibilities can be reduced to the results shown in the Figure 8.



Networks	LeNet	AlexNet	ConvNet	VGGNet
# of layers	6	8	12	23
# of possible structures	5	7	11	17

Figure 8: Possible architecture configurations for different networks [Yu+20b]

All the remaining possible architectures are then trained and compared to each other and to the original black-box network. The results on the ConvNet [Zha+17] (12 layers) and VGGNet [Con+16] (23 layers) are shown in the Figure 9. The importance of the choice of the architecture of a NN is really clear on the performances of these networks. The performances of the best architecture are lower but similar to the black-box original networks.

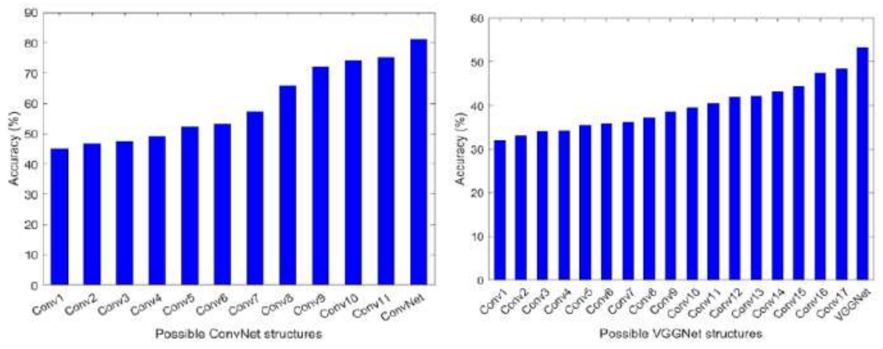
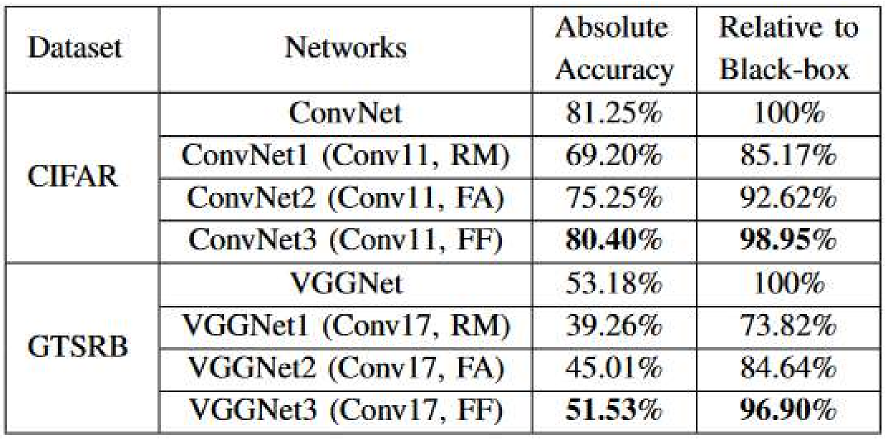


Figure 9: Performances of ConvNet and VGGNet possible architectures [Yu+20b]

Once the architecture of the NN is determined, it is trained through different algorithms. The researchers proved their technique to be more efficient than previous ones. Their "FeatureFool" (FF) method shows its performance in the Figure 10 against random image selection and Feature Adversary. This method trained a network with performances reaching over 96% efficiency over the black-box NN.



Dataset	Networks	Absolute Accuracy	Relative to Black-box
CIFAR	ConvNet	81.25%	100%
	ConvNet1 (Conv11, RM)	69.20%	85.17%
	ConvNet2 (Conv11, FA)	75.25%	92.62%
	ConvNet3 (Conv11, FF)	**80.40%**	**98.95%**
GTSRB	VGGNet	53.18%	100%
	VGGNet1 (Conv17, RM)	39.26%	73.82%
	VGGNet2 (Conv17, FA)	45.01%	84.64%
	VGGNet3 (Conv17, FF)	**51.53%**	**96.90%**

Figure 10: Accuracy on test sets. RM = Random, FeatureAdversary = FA, FeatureFool = FF [Yu+20b]

Conclusion

The implementations of Neural Networks have proven to be vulnerable to theft attack even in a black-box methodology. Batina et al. [Bat+19b] proposed a methodology to determine the activation function of a neural network layer through electromagnetic side-channel information leakage. Using correlation between their measurements on the microcontroller and possible weights, they extracted weights from the computation with great accuracy. The attack proposed by Yu et al. [Yu+20b] recovered the architecture of the NN and the new networks had similar performances as the stolen ones. They used side-channel information to recover the architecture and main parameters of the neural networks. They used common networks including ConvNet [Zha+17], VGGNet [Con+16] and others to show that their approach is more effective at stealing deep networks than previous works. Further research must be conducted to develop defence mechanisms against theft for NN on hardware accelerators.

Implementations of deep neural networks on FPGA have been proven to leak information about the architecture and shape of the networks. This internship aims at measuring the leakage of the weights of neural networks. The main target will be the Xilinx Ultrascale+ ZCU104 which includes both FPGA and processor architecture. After implementing simple neural networks on the board, experiments will be conducted to measure the electromagnetic information leakage of the system.

References

[GMF01] K. Gandolfi, C. Mourtel, and O. Francis. “Electromagnetic Analysis: Concrete Results”. In: *Cryptographic Hardware and Embedded Systems* 2162:251–61 (2001). Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001. `https://doi.org/10.1007/3-540-44709-1_21`.

[JS01] Quisquater Jean-Jacques and David Samyde. “ElectroMagnetic Analysis (EMA): Measures and Counter-Measures for Smart Cards.” In: *Smart Card Programming and Security* (2001). Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2001. `https://doi.org/10.1007/3-540-45418-7_17`, pp. 200–210.

[BCF04] E. Brier, C. Clavier, and O. Francis. “Correlation Power Analysis with a Leakage Model”. In: *Cryptographic Hardware and Embedded Systems* 3156:16–29 (2004). Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004. `https://doi.org/10.1007/978-3-540-28632-5_2`.

[Con+16] A. Conneau et al. “Very deep convolutional networks for natural language processing”. In: *CoRR* abs/1606.01781 (2016). Available: `https://arxiv.org/pdf/1606.01781.pdf`.

[Zha+17] R. Zhao et al. “Accelerating Binarized Convolutional Neural Networks with Software-Programmable FPGAs”. In: *Int'l Symp. on Field Programmable Gate Arrays (FPGA)* (2017).

[HZS18] W. Hua, Z. Zhang, and G. E. Suh. “Reverse engineering convolutional neural networks through side-channel information leaks”. In: *Proceedings of the 55th Annual Design Automation Conference*. 2018, pp. 4:1–4:6.

[Bat+19a] L. Batina et al. “CSI NN: Reverse Engineering of Neural Network Architectures Through Electromagnetic Side Channel”. In: *USENIX*. `https://www.usenix.org/sites/default/files/conference/protected-files/sec19_slides_batina.pdf`. 2019.

[Bat+19b] L. Batina et al. “CSI NN: Reverse engineering of neural network architectures through electromagnetic side channel”. In: *28th USENIX Security Symposium*. 2019, pp. 515–532.

[Vas+19] Duddu Vasisht et al. “Stealing Neural Networks via Timing Side Channels.” In: *ArXiv:1812.11720 [Cs]* (2019). `https://arxiv.org/pdf/1812.11720.pdf`.

[Yu+20a] H. Yu et al. “CloudLeak: Large-Scale Deep Learning Models Stealing Through Adversarial Examples”. In: *Proceedings 2020 Network and Distributed System Security Symposium*. `https://doi.org/10.14722/ndss.2020.24178`. 2020.

[Yu+20b] H. Yu et al. “DeepEM: Deep Neural Networks Model Recovery through EM Side-Channel Information Leakage”. In: *IEEE International Symposium on Hardware Oriented Security and Trust (HOST)* (2020). `https://doi.org/10.1109/HOST45689.2020.9300274`.