



On Reverse Engineering Neural Network Implementation on GPU

Lukasz Chmielewski^{1,3(✉)} and Léo Weissbart^{1,2}

¹ Radboud University, Nijmegen, Nijmegen, The Netherlands
{lukasz, l.weissbart}@cs.ru.nl

² Delft University of Technology, Delft, The Netherlands

³ Riscure BV, Delft, The Netherlands

Abstract. In recent years machine learning has become increasingly mainstream across industries. Additionally, Graphical Processing Unit (GPU) accelerators are widely deployed in various neural network (NN) applications, including image recognition for autonomous vehicles and natural language processing, among others. Since training a powerful network requires expensive data collection and computing power, its design and parameters are often considered a secret intellectual property of their manufacturers. However, hardware accelerators can leak crucial information about the secret neural network designs through side-channels, like Electro-Magnetic (EM) emanations, power consumption, or timing.

We propose and evaluate non-invasive and passive reverse engineering methods to recover NN designs deployed on GPUs through EM side-channel analysis. We employ a well-known technique of simple EM analysis and timing analysis of NN layers execution. We consider commonly used NN architectures, namely Multilayer Perceptron and Convolutional Neural Networks. We show how to recover the number of layers and neurons as well as the types of activation functions. Our experimental results are obtained on a setup that is as close as possible to a real-world device in order to properly assess the applicability and extendability of our methods.

We analyze the NN execution of a PyTorch python framework implementation running on Nvidia Jetson Nano, a module computer embedding a Tegra X1 SoC that combines an ARM Cortex-A57 CPU and a 128-core GPU within a Maxwell architecture. Our results show the importance of side-channel protections for NN accelerators in real-world applications.

Keywords: Deep neural network · Side-channel analysis · Simple power analysis · Reverse engineering

1 Introduction

Deep learning is more and more deployed in many research and industry areas ranging from image processing and recognition [13], image recognition for autonomous vehicles [4], robotics [9], and natural language processing [27], medical applications [15], IoT speech recognition [26] to security [14, 29]. This rapid

deployment is caused by the increased computational capabilities of computers and huge amounts of data available for machine learning. Additionally, it leads to more and more complex machine learning architectures.

In this paper, we focus on the analysis of Multilayer Perceptron (MLP) and Convolutional Neural Network (CNN) implemented using GPU accelerators, as they are the most commonly used feed-forward neural networks architectures.

Designing and finding parameters for neural networks has become an increasingly hard task since the NN architectures become more complex. From the industrial point of view, we can observe an increase in the number of intellectual property (IP) of NNs. Such IPs of commercial interest need to be kept secret. Moreover, in the medical context, the privacy aspects of NNs can also become a threat if revealed.

Additionally, EMVCo, an entity formed by MasterCard and Visa to manage specifications for payment systems, requires deep learning techniques for security evaluations [24]. Due to the above reasons, hackers might want to reverse neural networks to learn secret information.

There exist potentially easier ways to recover a network than using complex side-channels like EM or power consumption. For example, physical access to the device might be sufficient for an attacker to access the NN firmware and to reverse engineer it using binary analysis. As a countermeasure, those devices are equipped with standard protections like blocking binary access, blocking JTAG access, or code obfuscation. Furthermore, the IP vendors usually forbid users to access architectural side-channel information, such as memory and cache due to security and privacy concerns. Additionally, they implement countermeasures in software and hardware against logical attacks that would allow hackers to obtain run-time control on the device.

Therefore, for such protected implementations, side-channel attacks become viable for reverse engineering NNs. Side-channel analysis (SCA) has been widely studied for the last 20 years due to its capability to break otherwise secure algorithms and recover secret information. In 2019, Batina et al. [1] presented the first SCA attack to extract architecture and weights from a multilayer perceptron implemented on a microcontroller; this attack employed both timing and EM side-channels. This attack has shown that SCA is a serious threat to NNs.

However, there has been little work done on SCA against GPU-based neural networks¹. To the best of our knowledge, there has been no power or EM side-channel attack presented that targets GPU-based neural networks, while GPU is the platform of choice to train and deploy neural networks.

In this work, we aim at evaluating the security of a setup that is as close to a real-world application as possible, and therefore, we target NNs running on GPU. Therefore, we target the Nvidia Jetson Nano, a module computer embedding a Tegra X1 SoC combining an ARM Cortex-A57 CPU and a 128-core GPU with Maxwell architecture. This hardware accelerator is relatively complex in

¹ The only SCA against GPU-based NN that we have found is presented in [28]. However, it works in a different context to ours and is based on software context-switching timing side-channel; see Subsect. 1.1 for details.

comparison to a simple microcontroller. In particular, our setup employs the PyTorch python framework running on the full Linux operating system (on the ARM CPU) to instruct the GPU accelerator to execute NN computations. This complexity poses several technical difficulties for our analysis due to a large amount of noise and misalignment. Because of these challenges, we limit our analysis to so-called simple EM analysis² and we recover numbers of layers and neurons as well as the types of activation function being executed. Our experiments show that all this secret information can be recovered using dozen of EM traces independent of inputs even when significant noise and misalignment are present when sufficient signal processing techniques are used.

Note that we need to analyze a GPU implementation as black-box since the low-level details of the implementation could not be public. Moreover, due to the parallel nature of GPUs, we cannot simply replicate existing attacks for other architectures, but adjust them adequately.

We leave recovering neuron weights and CNN hyperparameters using more complex side-channel attacks to be future work.

1.1 Related Works

Any computation running on a platform might result in physical leakages. Those leakages form a physical signature from the reaction time, power consumption, and EM emanations released while the device is manipulating data. Side-channel analysis (SCA) exploits those physical signatures to reveal secret information about the running program or processed data. In its basic form, SCA was proposed to perform key recovery attacks on cryptographic implementations [10, 11].

The application of SCA is not limited to the type of processing unit and can be applied to microcontrollers as well as other platforms. In [5, 7, 8, 16], EM and power side-channel attacks are performed on GPU-based AES implementations.

In [19] SCA is used to break isolation between different applications concurrently using a GPU. Essentially this work identify different ways to measure leakage using software means from any shared component.

Side-channel attacks can be applied to extract the information of a neural network. Batina et al. [1] presented the first EM side-channel attack to extract the complete architecture and weights from an MLP network implemented on a CPU. Subsequently, Honggang Yu et al. [33] combined simple EM analysis with adversarial active learning to recover a large-scale Binarized Neural Network, which can be seen as a subset of CNNs, implemented on a field-programmable gate array (FPGA). In this attack, the recovery of the weights is not done through EM analysis, but using a margin-based adversarial learning method. This method can be seen as a cryptanalysis against NNs where weights are treated as an attacked secret key. Takato et al. [25] show how to use simple EM analysis to retrieve an activation function from a NN implemented on an Arduino Uno microcontroller. In [32], correlation power analysis is used to reveal neuron

² Simple EM analysis involves visually interpreting EM traces over time in order to recover the secret.

weights from the matrix multiplication implemented with systolic array units on an FPGA. Another relevant attack [30] uses power SCA together with machine learning classifier to reveal internal network architecture, including its detailed parameters, on an ARM Cortex embedded device. Maji et al. [17] demonstrated a timing attack combined with a simple power analysis of microcontroller-based NN to recover hyperparameters and the inputs of the network.

The only previous attack that targets GPU-based NN [28], to the best of our knowledge works in a different setting to ours; a model developer and an adversary share the same GPU when training a network and the adversary aims to break the isolation to learn the trained model. This attack is not applicable to an edge accelerator platform as the training phase is always performed in a controlled environment with more capable resources. This attack also relies on the presence of an adversary sharing GPU resources while our attack does not make such a requirement.

A recent survey of existing SCA methods for architecture extraction of neural networks implementations is presented in [2] and an overview of hardware attacks against NN is given in [31].

1.2 Contributions

In this paper, for the first time, to the best of our knowledge, we investigate using simple EM analysis to break side-channel security of NN (namely Multilayer Perceptron and Convolutional Neural Networks) running on a GPU. We present how to successfully recover the number of layers and neurons as well as the types of activation functions. Most importantly, our results show the importance of side-channel protections for NN accelerators in real-world applications.

We leave recovering neuron weights and CNN hyperparameters using more complex side-channel attacks, like DPA or template attack to be future work.

Our experimental results are obtained on the setup that is as close as possible to a real-world device setup in order to properly assess the applicability and extendability of our methods.

1.3 Organization of the Paper

Section 2 presents the necessary background on simple EM analysis, NNs, and the employed GPU architecture. Subsequently, our threat model is described in Sect. 3 and the target and NN implementation in Sect. 4. We present our reverse engineering methods and experimental results in Sect. 5. Finally, conclusions and future work are presented in Sect. 6.

2 Background

In this section we introduce the concepts of simple EM analysis (SEMA), Artificial Neural Network, and we describe a GPU Architecture that we analyze.

2.1 Simple EM Analysis

In the second half of the nineties, Paul Kocher et al. presented the first research publications about practical side-channel attacks [10, 11]. They realized that the security of cryptographic algorithms does not only depend on the mathematical properties but also on implementation details, regardless of whether the implementation is hardware or software.

By monitoring side-channels of cryptographic implementations of DES and RSA using a low-cost oscilloscope, more specifically the power consumption [10] and the execution time [11], they were able to discover the side-channel leakages corresponding to the private key usage. As a result, they were able to recover the private key with little effort and low cost. The methods presented by Kocher et al. in [10] are called Simple Power Analysis (SPA) and Differential Power Analysis (DPA). SPA involves visually interpreting power consumption traces over time to recover the secret key³. Variations in power consumption occur particularly strongly as the device performs different operations. If the sequence of such operations depends on the key, then using a standard digital oscilloscope the attacker would learn information about the key. Additionally, the attacker might further use frequency filters and averaging to improve the signal quality.

DPA improves on SPA by employing a statistical analysis between intermediate values of cryptographic computations and the corresponding side-channel traces in order to recover the key.

In this paper we employ technique called Simple EM analysis (SEMA) that is equivalent to SPA but in the EM domain. For example, how to recognize different amount of layers in NN is shown in Fig. 7.

Template Attack [3] (TA) is another notable SCA technique that can be seen as an improved version of SPA. It combines statistical modeling with SPA and DPA and can achieve better accuracy than SPA. It consists of two phases, called *profiling* and *matching*. An attacker first creates a “*profile*” of a sensitive device, which is under their full control, and then *matches* this profile to measurements from the victim’s device to quickly find the secret key.

We leave using aforementioned DPA and TA to recover neuron weights and CNN hyperparameters as future work.

2.2 Artificial Neural Network

Artificial Neural Network (ANN) is a category of computing systems that can exhibit generalization for a given task beyond the training data. A Neural Network (NN) is a network of many simple computing units (also called nodes) connected by communication channels to transmit a signal. The units transform numerical inputs together with local data (i.e., weights) to produce an output. During the so-called ‘training process’, weights and biases are adjusted to minimize a given loss function and stores the experimental knowledge about the training task.

A simple type of neural network is a perceptron (also called neuron). The perceptrons perform an inner product of the inputs in_i and weights w_i , plus

³ In our case the secret key is the network architecture.

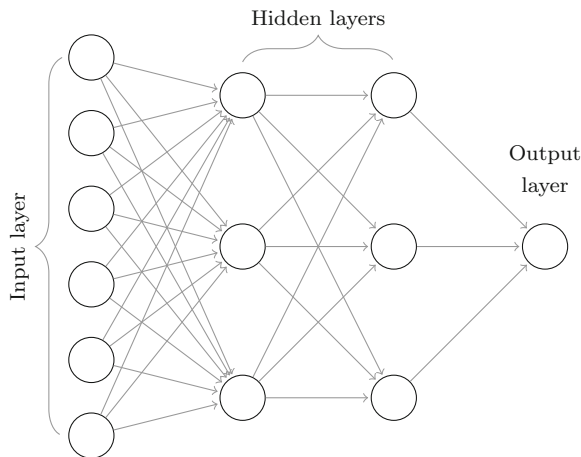


Fig. 1. Schematic of a multi-layer perceptron

a bias b through an activation function to compute its output. The activation function is usually a logistic or *tanh* function. Hence, the formula of an artificial neuron output is typically computed according to Eq. (1).

$$o = \tanh(b + \text{sum}[w_i * x_i]) \quad (1)$$

A more advanced type of neural network is the Multi-layer Perceptron (MLP). An MLP is a neural network constituted of multiple perceptrons grouped in layers to form a directed graph. Each perceptron in a layer is fully connected with a given weight w for every node of the following layer. An MLP consists of at least three layers: one input layer, one output layer, and one hidden layer. An MLP is represented on Fig. 1 An MLP with more than one hidden layer is considered a deep learning architecture. To train the network, the backpropagation algorithm is used, which is a generalization of the least mean squares algorithm in the linear perceptron. Backpropagation is used by the gradient descent optimization algorithm to adjust the weight of neurons by calculating the gradient of the loss function [18]. After the training phase, only the forward loop is computed without backward propagation. This mode is called ‘inference’ and is the mode used when deploying a neural network.

More advanced deep neural network units involve convolution operation in place of the inner product of perceptrons. Neural networks with at least one layer of convolution units are called Convolutional Neural Networks (CNN). CNNs are commonly applied when analyzing visual imagery to take advantage of the hierarchical pattern in data and assemble patterns of increasing complexity using smaller and simpler patterns called filters. The convolutional layer consists of several filters (i.e., two-dimensional array of real values) used to detect a specific type of feature in the input. The operation consists in successive dot products of the filters with patches of the inputs of corresponding size. Because the filter

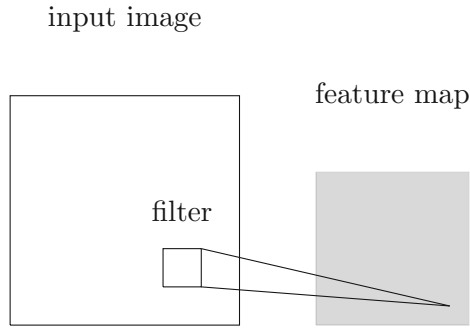


Fig. 2. Convolutional layer operation

size is smaller than the input size, the filters is shifted across the whole input area with overlapping to produce a feature map. This feature map is the layer's output. The output of one convolution operation with one filter is depicted in Fig. 2 Commonly, the filters are also called the weights of the convolutional layer. CNNs usually consist of successive Convolutional layers followed by an MLP structure to perform a classification task.

2.3 GPU Architecture

GPU is a specialized computer hardware designed to accelerate parallel computing for image processing. Deep learning algorithms can benefit from GPU high parallelization to boost their performances, especially when dealing with visual data. A GPU groups several GPU cores into a Streaming Multiprocessor (SM). The specific SM of Maxwell GPU architecture is shown in Fig. 3. All GPU cores within a SM can handle floating-point operations in a Single Instruction Multiple Data (SIMD) paradigm. This way, the exact same processing can be applied to a large volume of data to reach a higher throughput than for a CPU.

CUDA is the Software Development Kit (SDK) introduced by NVIDIA that gives direct access to the GPU's instruction set and facilitates general-purpose programming. From the programming perspective, a program that runs on a GPU is divided into parallel threads groups into wraps of 32 threads partitioned into blocks within grids executed on the SM [21]. When the number of blocks in a SM is less than the number of blocks assigned for the operation, the blocks are queued and scheduled to be executed at a later time. This method allows programs to be scalable for the hardware it is executed on and offers speed up for devices with more blocks per SM. Higher-level programming languages such as python frameworks relies on CUDA to call computation every low level function on GPU. We will see that it is possible to exploit this feature to perform side-channel a analysis of the size of data processed by the GPU.

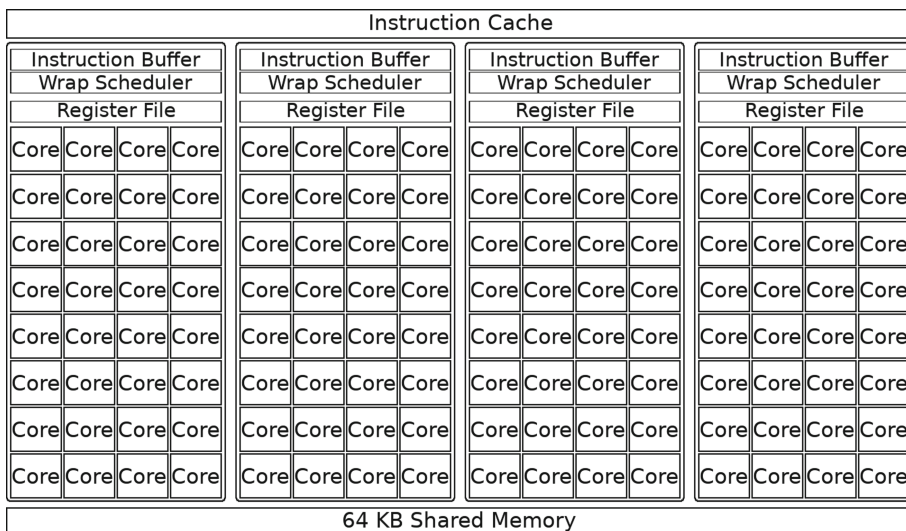


Fig. 3. Maxwell streaming multiprocessor architecture

3 Threat Model

The main goal of this attack is to reverse engineer the neural network architecture using only side-channel information. In this scenario, we consider an attacker with no insight of the inputs type, source or the implementation of the machine learning algorithm. Currently, to the best of our knowledge, there is no public implementation deploying side-channel countermeasure. We consider a passive and non-invasive attacker who can only acquire side-channel measurement while operating “normally” the target device and cannot control the flow of operation.

A suitable use case for this attack is considering an attacker who acquired a legal copy of the network in a black-box setting and aims to recover its internal details for IP theft. The attacker controls the inputs and performs side-channel measurement during the inference phase of the neural network. The goal is to reverse engineer the following information about the neural network architecture: number of layers, number of outputs, and activation functions in the network.

If successful, this attack can have severe monetary repercussions for companies investing significant resources to develop customized machine-learning models to create highly valuable IPs [22]. A successful attacker that is able to steal such models can offer similar services at much lower cost than the investing companies.

4 The Target and Network Implementation

The target is an Nvidia Jetson Nano [34], a module computer embedding a Tegra X1 SoC [35] combining an ARM Cortex-A57 CPU and a 128-core GPU

with Maxwell architecture suitable for AI applications such as image classification, object detection, segmentation, and speech processing. Specifically, modules similar to this one are used for real application in automotive visual computing, namely for Nvidia drive CX and PX computer platforms. The Jetson Nano Tegra X1 SoC contains a GPU with one Maxwell Streaming Multiprocessor (SMM) (see Fig. 3). The SMM is partitioned into four distinct 32-CUDA core processing blocks (128 CUDA cores total), each with its own dedicated resources for scheduling and instruction buffering.

The neural network is a convolutional neural network (CNN) implemented using the PyTorch python framework [23]. The dataset used to train the network is the CIFAR10 dataset [12], a 60 000 32×32 color images dataset representing 10 classes. The reference CNN architecture consists of two convolutional layers (of 6 and 16 filters of size 5) with max-pooling and three linear Fully Connected (FC) layers, all regulated with the ReLU activation function, and the final FC output layer. The input is a three-channel image of size 32×32 , and the output is a 10-sized vector of each class of the classification problem. This architecture, together with the corresponding SPA, is presented in Fig. 4.

To better measure the execution of the neural network, we use a power trigger. The Jetson Nano handles General Purpose Input/Outputs (GPIOs). We use one GPIO pin to implement a trigger around the forward loop of the neural network to be sure only to measure while the GPU is active. It is to be noticed that, the neural network is already trained and the gradient operation is disabled to prevent the backward loop from happening.

To record the EM traces we removed the heatsink of the target and placed a Riscure Low Sensitivity (LS) EM probe⁴ above the main chip package. The best position of the probe is empirically chosen to maximize the leaking signal. We manually searched the position with a grid scanning above the chip for multiple locations and chose the most promising position based on visual inspection of the traces. This best location is presented in Fig. 5.

The oscilloscope in our experiment is the Teledyne Lecroy WaveRunner 8404M. We used it in two configurations, one for characterization with 5×10^9 samples/s and at most 32×10^6 samples and the second one for simple EM with 10^9 samples/s and at most 10^7 . We used greater sampling rate in the first configuration because we performed frequency analysis and we needed to be able to record a signal up to the GPU maximum clock frequency, namely 900 MHz, in good quality; for simple EM we do not need that high accuracy. The oscilloscope has TCP/IP support for both controlling and downloading measurements, which helps to automatize the entire process.

We acquired and analyzed using Riscure's Inspector software package⁵.

The goal of the neural network used in this paper is not meant for high efficiency or presenting a challenging classification task, but rather to show the methods and principles side-channel analysis can bring in to extract information from a neural network closed implementation.

⁴ https://www.riscure.com/uploads/2017/07/inspector_brochure.pdf.

⁵ <https://www.riscure.com/security-tools/inspector-sca>.

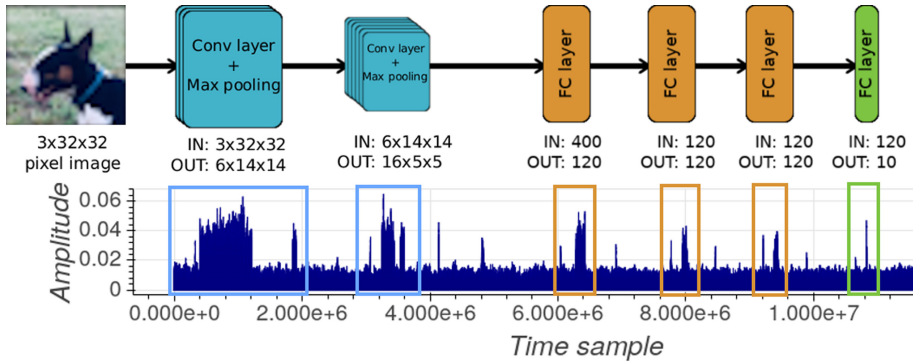


Fig. 4. SPA of CIFAR10 convolutional neural network

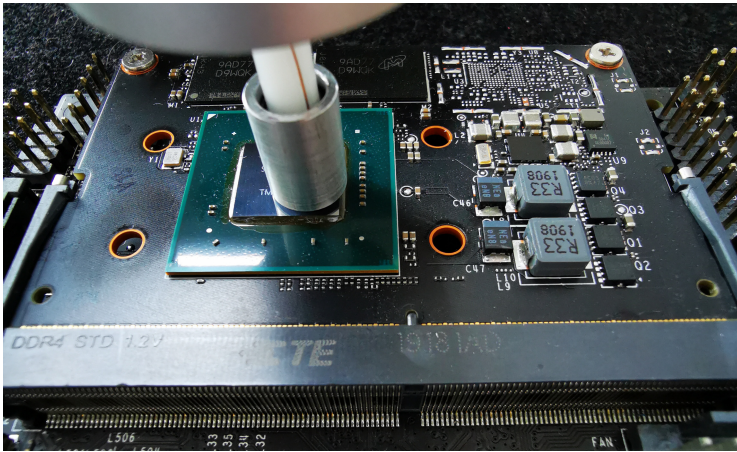


Fig. 5. Experimental setup: the EM probe location

5 Reverse Engineering

5.1 Characterization

In Fig. 4, the architecture of the neural network is showed next to an EM trace measured during its execution on the target. From the EM trace, we can observe that every different step of the forward loop of the NN is distinguishable. The two convolutional blocks are identifiable by a first activity corresponding to the convolutional operation followed by a smaller activity corresponding to the pooling operation. The layers of the MLP, namely, the FC layers, are also detectable by single peaks.

It is possible to verify whether the leakage is effectively coming from the GPU activity by observing the leakage in the frequency domain. Because the GPU maximum clock frequency is 900 MHz, the computation made on the device

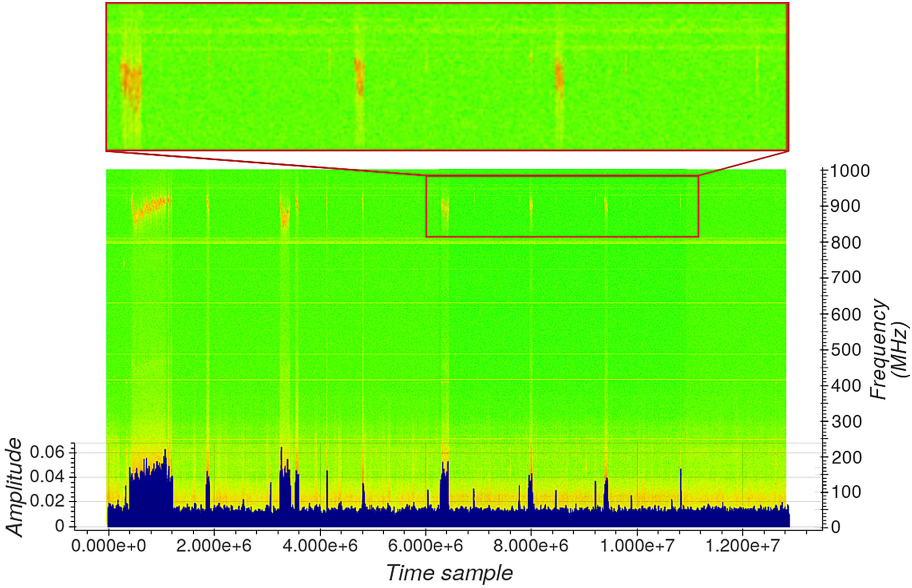


Fig. 6. A single EM trace (in the blue color at the bottom) and the corresponding spectrogram (middle) with the MLP activation zoomed-in (top)

will emit leakage in the same range of frequencies. In Fig. 6, we represented the absolute value of the leakage together with the spectrogram plot of the leakage. We can see that the detected leakage correspond to the frequency range of the GPU. It would be possible to continue the same analysis using this frequency signal, but in this paper, we will only focus on EM analysis in the time domain.

In the following analyses, we either use raw traces or apply an averaged window resampling method on the absolute value of the signal. The averaged window resampling reduces the number of features of the trace by averaging samples in a fixed-size window shifted without overlapping across all samples of the trace. This processing makes alignments on specific patterns faster and easier.

5.2 Reverse Engineering the Number of Layers

In this section we investigate how to recover the number of hidden layers in the MLP from the SCA during the inference phase.

Since the dataflow of a NN is such that layers are processed sequentially, the analysis of different number of fully connected layer is trivial from the EM traces. We measured three different implementations of a neural network, with a different number of fully connected layers and observe their leakage. From the reference neural network model, we change the number of fully connected layers from 4 to 6. The number of neurons in each of the additional layers is the same as the second fully connected layer from the reference model (i.e., 120 neurons).

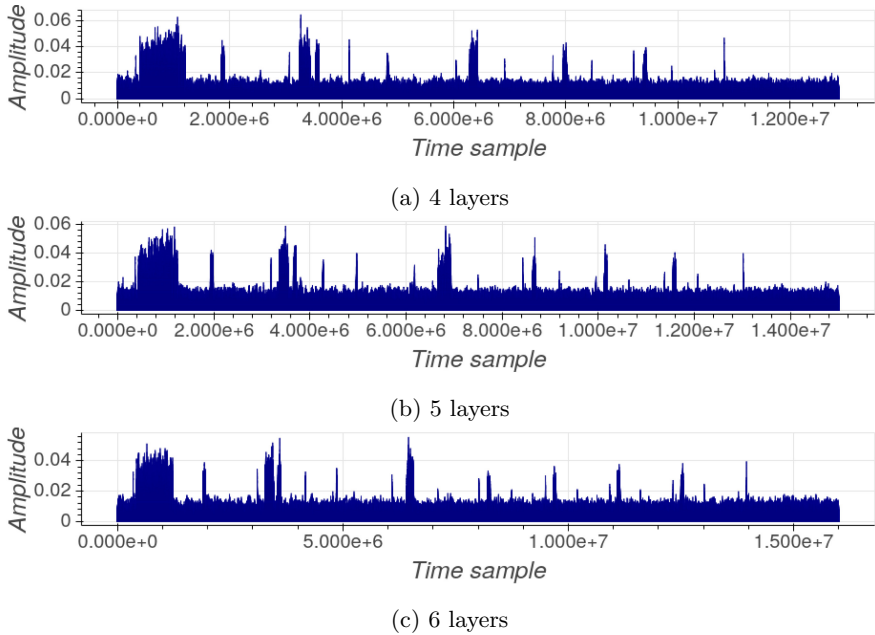


Fig. 7. Differences in the number of fully connected layers

The resulting EM measurements are represented in Fig. 7. From the three plots, the two first convolutional blocks and the fully connected layers are clearly identifiable. While the plots are aligned according to the first convolutional layer, the timing of the execution is not consistent. Many process interruptions occur during the computation, leading to misalignments in the traces. However, the additional layers do appear in the EM measurement and are easily identifiable.

5.3 Reverse Engineering the Number of Neurons

Now we investigate how to recover the number of neurons in a hidden MLP layer.

In a GPU implementation, every neuron operation is processed in parallel. However, the parallelization degree depends on the size of the inputs and number of neurons, as there is a limit on the number of floating-point operation that can be computed in parallel. For example, given N GPU threads, each capable of computing one floating-point operation per clock cycle, the GPU scheduler can compute N operations per clock cycle. If $n_{inputs} \times n_{neurons} > N$ then the number of neurons will partially leak, and if $n_{inputs} \geq N$ then the number of neurons will entirely leak as every neuron computation would require more than one cycle.

From the execution of the linear operation in the fully connected layers, it is possible to recover the number of perceptrons per layer using timing side-channel. In Fig. 8 different models are analyzed. Here, we control the number of

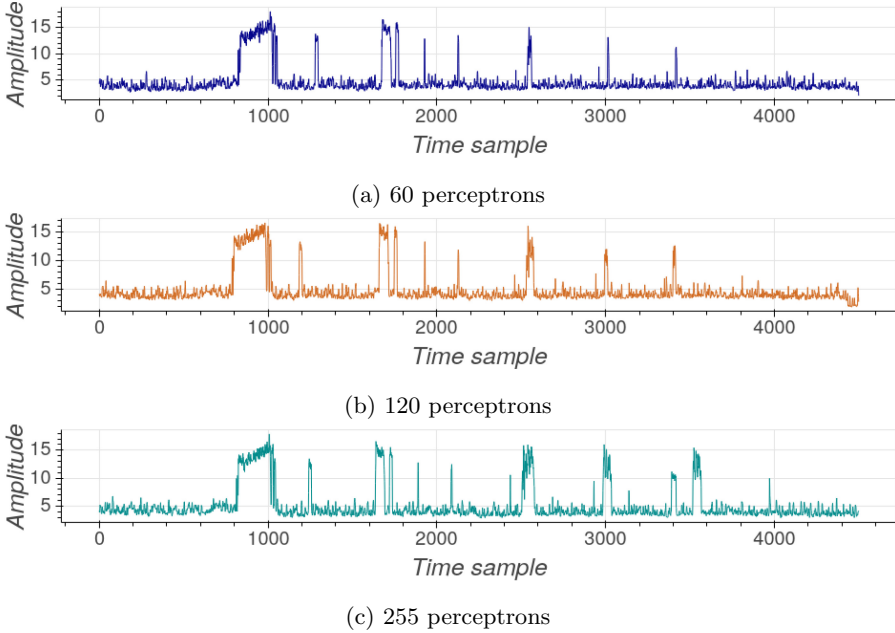


Fig. 8. Difference in number of perceptrons inside fully connected layers

neurons within the hidden layers. We can see that when the number of neurons in the layers increases, the execution time of each layer also increases.

Recovering the exact number of neurons in a layer would require to be capable to distinguish a single neuron difference. In Fig. 9, the timing of the first fully connected layer activity with an increasing number of perceptrons from 30 to 100 is represented. For every number of perceptron, we averaged fifty measurements and align the traces according to the desired pattern to measure the execution time of the specific layer. While the relation shows a linear behavior, the measurements noise and re-alignment, still make it difficult to distinguish a single perceptron change. However, approximate recovery is possible with a relatively low error margin.

5.4 Reverse Engineering the Type of Activation Function

Nonlinear functions are essential to approximate a linearly non-separable problem. The use of these functions also helps to reduce the number of network nodes. With the knowledge of the type of nonlinear function used in a layer, an attacker can deduce the behavior of the entire neural network using the input values.

We analyze the side-channel leakage from different commonly used activation functions, namely ReLU, Sigmoid, Tanh, and Softmax [6, 20]. The activation function applied to the first convolutional layer of the network is changed in different measurements. We measured the EM leakage of the execution of the layer

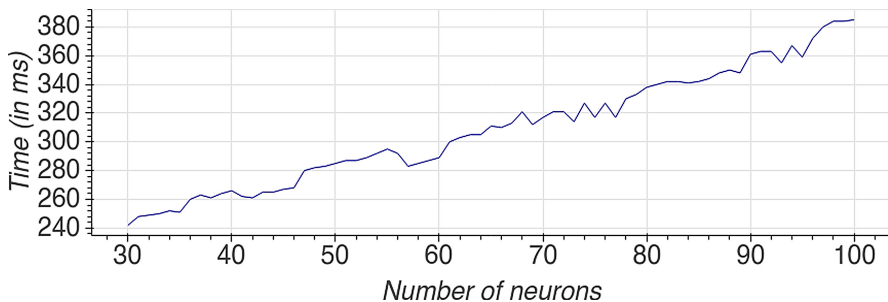


Fig. 9. Differences in the number of perceptrons (from 30 to 100 units)

Table 1. Statistical analysis on computation time (in μ s).

Activation function	Mean	Maximum	Minimum
ReLU	33.5	34	33
Tanh	36.0	37	34
Sigmoid	43.3	46	41
Softmax	124.5	127	123

computation and the activation function for random input and is represented in Fig. 10. We identify the execution of the convolutional layer from 0 to 520 time samples and it is identical for all sub-figures. The execution of the activation function presents differences among all different activation functions. We can notice for example that the execution of the ReLU activation function is the shortest and that the Softmax function is the longest by far. The timing differences between ReLU, Tanh, and Sigmoid activation functions are smaller.

The computation time of the activation function does depend on the size of the input. Therefore, to identify the type of activation function, one should first recover the number of inputs. We measured fifty executions of the activation function after the first convolutional layer. The input of this activation function is of the size of the output of the convolutional layer before the pooling layer and is of size $6 \times 28 \times 28 = 4704$. All measurements are done on random data, and we draw a statistical analysis of the timing pattern for all types of activation functions considered in Table 1. It can be observed that each activation function stands out, and thus it is possible to recover trivially the type of activation function from a neural network implementation.

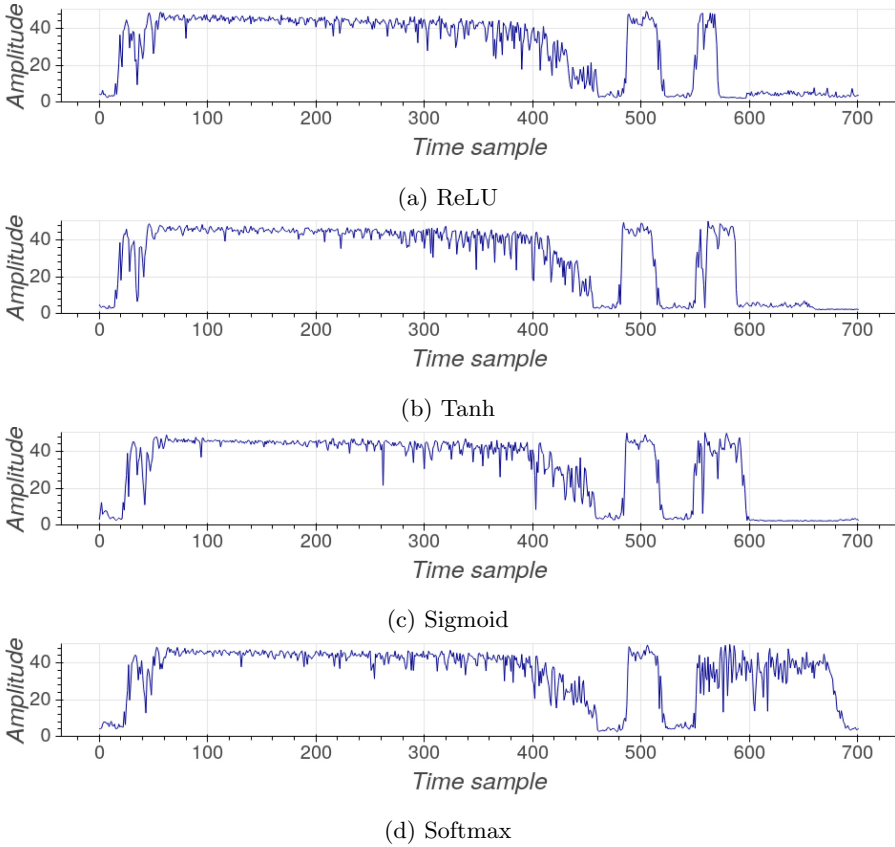


Fig. 10. Differences in the type of activation function applied on layer output

6 Conclusions and Future Work

Side-channel analysis have already been proven capable to reverse engineering a neural network implemented on a microcontroller architecture. While microcontrollers can be the hardware of choice for some small edge computing applications, GPU stays the most popular platform for deep learning. In this paper, we show the possibility to recover key parameters of a GPU implementation of a neural network. We can recover the number of layers and number of neurons per layer of a multi-layer perceptron with simple power analysis. We can also identify different types of activation functions with single power analysis for a given number of inputs. We can conclude that we have managed to recover all the secret information that can be achieved using only simple EM analysis.

For the reverse engineering of a complete neural network, the weights of all layers for both MLP and CNN networks and network hyperparameters for CNNs should be recovered too. We consider these tasks to be future work, but we envision that the weights can be recovered using correlation or differential

power analysis on float vector multiplication similarly to [1]. However, the main challenges, besides noise and misalignment, would be the lack of information on how the multiplication is performed and the parallel aspect of GPU computation (i.e., multiple intermediate values might be computed at the same time). Recovery of CNN hyperparameters would be probably also a hard task and we suspect that it would require a template attack.

Acknowledgements. The authors would like to thank Carlos Castro, Louiza Papachristodoulou, and Yuriy Serdyuk from NavInfo for very informative discussions on using neural networks in the automotive context. Moreover, we would like to thank the anonymous reviewers for their insightful comments.

Lukasz Chmielewski is supported by European Commission through the ERC Starting Grant 805031 (EPOQUE) of P. Schwabe.

References

1. Batina, L., Bhasin, S., Jap, D., Picek, S.: CSI NN: reverse engineering of neural network architectures through electromagnetic side channel. In: 28th USENIX Security Symposium (USENIX Security 2019), pp. 515–532 (2019)
2. Chabanne, H., Danger, J.L., Guiga, L., Kühne, U.: Side channel attacks for architecture extraction of neural networks. *CAAI Trans. Intell. Technol.* **6**(1), 3–16 (2021)
3. Chari, S., Rao, J.R., Rohatgi, P.: Template attacks. In: Kaliski, B.S., Koç, K., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 13–28. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-36400-5_3
4. Fujiyoshi, H., Hirakawa, T., Yamashita, T.: Deep learning-based image recognition for autonomous driving. *IATSS Res.* **43**(4), 244–252 (2019)
5. Gao, Y., Zhou, Y., Cheng, W.: How Does Strict Parallelism Affect Security? A Case Study on the Side-Channel Attacks against GPU-based Bitsliced AES Implementation. *IACR Cryptology ePrint Archive 2018/1080* (2018)
6. Haykin, S., Network, N.: A comprehensive foundation. *Neural Netw.* **2**(2004), 41 (2004)
7. Jiang, Z.H., Fei, Y., Kaeli, D.: A complete key recovery timing attack on a GPU. In: 2016 IEEE International Symposium on High Performance Computer Architecture (HPCA), pp. 394–405. IEEE (2016)
8. Jiang, Z.H., Fei, Y., Kaeli, D.: A novel side-channel timing attack on GPUs. In: Proceedings of the on Great Lakes Symposium on VLSI 2017, pp. 167–172 (2017)
9. Kober, J., Bagnell, J.A., Peters, J.: Reinforcement learning in robotics: a survey. *Int. J. Rob. Res.* **32**(11), 1238–1274 (2013). <https://doi.org/10.1177/0278364913495721>
10. Kocher, P., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 388–397. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48405-1_25
11. Kocher, P.C.: Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 104–113. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-68697-5_9
12. Krizhevsky, A.: Learning multiple layers of features from tiny images. Technical report, University of Toronto (2009)

13. Krizhevsky, A., Sutskever, I., Hinton, G.E.: ImageNet classification with deep convolutional neural networks. In: Pereira, F., Burges, C.J.C., Bottou, L., Weinberger, K.Q. (eds.) *Advances in Neural Information Processing Systems* 25, pp. 1097–1105. Curran Associates, Inc. (2012). <http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf>
14. Kučera, M., Tsankov, P., Gehr, T., Guarnieri, M., Vechev, M.: Synthesis of probabilistic privacy enforcement. In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017*, pp. 391–408. Association for Computing Machinery, New York (2017). <https://doi.org/10.1145/3133956.3134079>
15. Lundervold, A.S., Lundervold, A.: An overview of deep learning in medical imaging focusing on MRI. *Zeitschrift für Medizinische Physik* **29**(2), 102–127 (2019)
16. Luo, C., Fei, Y., Luo, P., Mukherjee, S., Kaeli, D.: Side-channel power analysis of a GPU AES implementation. In: *2015 33rd IEEE International Conference on Computer Design (ICCD)*, pp. 281–288. IEEE (2015)
17. Maji, S., Banerjee, U., Chandrakasan, A.P.: Leaky nets: recovering embedded neural network models and inputs through simple power and timing side-channels - attacks and defenses. *IEEE Internet Things J.* 1 (2021). <https://doi.org/10.1109/JIOT.2021.3061314>
18. Mitchell, T.M., et al.: *Machine Learning*. McGraw-Hill, New York (1997)
19. Naghibijouybari, H., Neupane, A., Qian, Z., Ghazaleh, N.A.: Side channel attacks on GPUs. *IEEE Trans. Dependable Secure Comput.* (2019)
20. Nair, V., Hinton, G.E.: Rectified linear units improve restricted Boltzmann machines. In: *ICML* (2010)
21. Nickolls, J., Buck, I., Garland, M., Skadron, K.: Scalable parallel programming with CUDA: is CUDA the parallel programming model that application developers have been waiting for? *Queue* **6**(2), 40–53 (2008)
22. Papernot, N., McDaniel, P., Sinha, A., Wellman, M.P.: SoK: security and privacy in machine learning. In: *2018 IEEE European Symposium on Security and Privacy (EuroS P)*, pp. 399–414 (2018). <https://doi.org/10.1109/EuroSP.2018.00035>
23. Paszke, A., et al.: Pytorch: an imperative style, high-performance deep learning library. *arXiv preprint arXiv:1912.01703* (2019)
24. Riscure: Automated Neural Network construction with Genetic Algorithm (blog post) (2018). <https://www.riscure.com/blog/automated-neural-network-construction-genetic-algorithm>
25. Takato, G., Sugawara, T., Sakiyama, K., Li, Y.: Simple electromagnetic analysis against activation functions of deep neural networks. In: Zhou, J., et al. (eds.) *ACNS 2020. LNCS*, vol. 12418, pp. 181–197. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-61638-0_11
26. Tariq, Z., Shah, S.K., Lee, Y.: Speech emotion detection using IoT based deep learning for health care. In: Baru, C., et al. (eds.) *2019 IEEE International Conference on Big Data (Big Data)*, Los Angeles, CA, USA, 9–12 December 2019, pp. 4191–4196. IEEE (2019). <https://doi.org/10.1109/BigData47090.2019.9005638>
27. Teuffl, P., Payer, U., Lackner, G.: From NLP (natural language processing) to MLP (machine language processing). In: Kotenko, I., Skormin, V. (eds.) *MMM-ACNS 2010. LNCS*, vol. 6258, pp. 256–269. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14706-7_20
28. Wei, J., Zhang, Y., Zhou, Z., Li, Z., Al Faruque, M.A.: Leaky DNN: stealing deep-learning model secret with GPU context-switching side-channel. In: *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pp. 125–137. IEEE (2020)

29. Wei, L., Luo, B., Li, Y., Liu, Y., Xu, Q.: I know what you see: power side-channel attack on convolutional neural network accelerators. In: Proceedings of the 34th Annual Computer Security Applications Conference, ACSAC 2018, pp. 393–406. Association for Computing Machinery, New York (2018). <https://doi.org/10.1145/3274694.3274696>
30. Xiang, Y., et al.: Open DNN box by power side-channel attack. *IEEE Trans. Circuits Syst. II Express Briefs* **67**(11), 2717–2721 (2020). <https://doi.org/10.1109/TCSII.2020.2973007>
31. Xu, Q., Arafin, M.T., Qu, G.: Security of neural networks from hardware perspective: a survey and beyond. In: Proceedings of the 26th Asia and South Pacific Design Automation Conference, ASPDAC 2021, pp. 449–454. Association for Computing Machinery, New York (2021). <https://doi.org/10.1145/3394885.3431639>
32. Yoshida, K., Kubota, T., Okura, S., Shiozaki, M., Fujino, T.: Model reverse-engineering attack using correlation power analysis against systolic array based neural network accelerator. In: 2020 IEEE International Symposium on Circuits and Systems (ISCAS), pp. 1–5 (2020). <https://doi.org/10.1109/ISCAS45731.2020.9180580>
33. Yu, H., Ma, H., Yang, K., Zhao, Y., Jin, Y.: DeepEM: deep neural networks model recovery through EM side-channel information leakage. In: 2020 IEEE International Symposium on Hardware Oriented Security and Trust (HOST), pp. 209–218. IEEE (2020)
34. NVIDIA Jetson Nano module Datasheet, April 2021. <https://developer.nvidia.com/embedded/dlc/jetson-nano-system-module-datasheet>
35. NVIDIA Tegra X1 White Paper, April 2021. <http://international.download.nvidia.com/pdf/tegra/Tegra-X1-whitepaper-v1.0.pdf>