ÉCOLE
CENTRALE LYON

Institute for Computing
and Information Sciences
Radboud University

Master's Thesis

Deep neural network reverse engineering through side channel information leakage

Author :
Thomas BUZER

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

September 16, 2022

FINAL YEAR INTERNSHIP

Academic year 2021-2022

This form should be inserted after the cover page of the FYI Report. It certifies that the Report has been validated by the firm and can be submitted as it stands to the Registrar's Office of Ecole Centrale de Lyon.

COMPANY VALIDATION OF FYI REPORT

Final Year Internship references

Student's name: BUZER Thomas

Report title: Deep neural network reverse engineering through side channel information leakage

Firm: Radboud University - Institute for Computing and Information Sciences

Name of Company Tutor: BATINA Lejla

Name of School Tutor: MARCHAND Cédric

The company acknowledges having read the report mentioned above and authorizes its transmission to the Ecole Centrale de Lyon.
The company authorizes the distribution of the report on the catalog of the library of the Ecole Centrale de Lyon:

- information on the TFE (title – author – keywords – summary…) will be public
- Access to the pdf report, upon authentication, will be limited to students and staff of the Ecole Centrale de Lyon.

Firm's representative
Name: BATINA LEJLA
Position: PROFESSOR

Date: 16-09-2022
Signature and stamp:
L. Batina

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 the vulnerability of 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 the Deep Processing Unit (DPU) accelerator from Xilinx implemented on the ZCU104 board from the same manufacturer.

Résumé

Le développement des applicaitons d'apprentissage profond se fait dans tous les domaines. Les voitures autonomes comme les caméras de sécurité profitent du développement de ces technologies intégrées. Afin d'optimiser les performances des implémentations de réseaux de neuronnes, des plateformes dédiées FPGA ou GPU ont été développées. L'enjeu de sécurité est ici la propriété intellectuelle que représente ces produits. Les modèles dintelligences artificielles se sont déjà montrés vulnérables à ce genre d'attaques en dévoilant des paramètres des réseaux de neurones utilisés tels que le nombre de neurones ou le type de couche. Cette étude porte sur les fuites électromagnétiques des paramètres des réseaux de neurones implémentés et des poids concernant l'architecture Deep Processing Unit (DPU) développée par Xilinx pour ses plateformes FPGA ZCU104.

Contents

List of Figures

List of Tables

Introduction

Object recognition, natural language processing, autonomous driving tasks or art are some of the 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 training data, the training itself and the development of efficient network are hard and expensive labors requiring expert knowledge and a lot of resources. That is why IP vendors are to keep their models and data 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, layer sizes and activation function of small NNs. 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 NNs. To solve this problem, Yu et al. [Yu+20b] presented a black-box methodology using EM SCA to infer possible architectures of the networks. They recovered some hyperparameters of the networks and used them to guess the possible networks commonly used. After training the remaining possible networks, the research team used adversarial learning techniques in order to efficiently train the best performing architecture to achieve performances equivalent to the original network. This attack focuses on binarized NN, with binary weights and activation, which are commonly used for small devices because of their small memory size.

This master's thesis aims at measuring the information leakage coming from FPGA implementation of neural networks. Focusing on the Ultrascale+ ZCU104 platform and the Deep Processing Unit (DPU) from Xilinx which is detailed in Section 2.2, neural networks will be implemented in order to measure the

electromagnetic emissions of the computing chip. First tests will be performed in order to evaluate if the board is producing any EM noise. Then a timing attack will be performed to recover the size of hidden convolution layers. Finally machine learning attacks will be implemented to recover hyperparameters and weights of running convolution networks.

Main acronyms

ASIC : Application Specific Integrated Circuit

CEMA : Correlation Electromagnetic Analysis

CMOS : Complementary Metal Oxide Semiconductor

CNN : Convolutional Neural Network

CPA : Correlation Power Analysis

DEMA : Differential Electromagnetic Analysis

DPA : Differential Power Analysis

DPU : Deep Processing Unit

EM : Electromagnetic

FPGA : Field Programmable Gate Array

GPU : Graphics Processing Unit

IC : Integrated Circuit

IP : Intellectual Property

MLP : Multi Layer Perceptron

MOSFET : Metal Oxide Semiconductor Field-Effect Transistor

NLP : Natural Language Processing

SCA : Side Channel Analysis

SEMA : Simple Electromagnetic Analysis

SPA : Simple Power Analysis

1 State of the Art

1.1 FPGA

Field Programmable Gate Arrays are integrated circuit which can be configured to fit the designers needs after manufacturing. Using Hardware Description Language (HDL), one can use logic blocks, memory blocks and other functions to construct complex computing chips. The flexibility of this platform gives it several advantages over GPUs and its possible performances enhancement makes it a good replacement for micro controllers.

Their flexibility and ease of reprogramming also reduces the "time to market" of FPGA products compared to ASICS. Combined with the increase in efficiency which now rivals those of ASICS, FPGA are to continue developping in every domain where specific computation is needed.

1.2 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 variables 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. Figure 1 shows the experimental setup used in by Batina [Bat+19a] to recover the result of a multiplication inside a microcontroller.

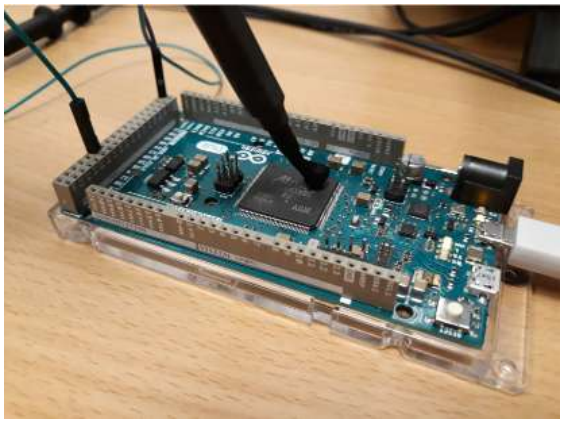


Figure 1: Experimental setup of SCA on ATmega2560 of an Arduino Mega [Bat+19a]

1.2.1 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.

1.2.2 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.

1.2.3 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.

1.2.4 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.

1.2.5 Profiling attacks

A profiling attack consists in collecting data about a device identical to target which can be controlled by the attacker. With the clone device, the attacker models the device depending on the controlled inputs and observed outputs. When the attacker obtains access to the real target, the model will be used to predict secrets from the device with a single or multiple traces.

Machine learning attacks have been a focus of the research [GHO15] [LBM14] as it allows the attacker to create a template without any assumption on the data distribution and can bypass some countermeasures such as masking. Some machine learning attacks also come with the benefit of not needing trace alignment to work properly. CNN are specifically designed to match patterns wherever they are inside the trace. This makes the whole process of recording and analysing the traces easier.

1.2.6 Countermeasures

Since the rise of side-channel attacks, multiple countermeasures have been developed to make real world implementations safer [OST06] [CG00] [Vey+12]. There are two main categories of countermeasures against SCA :

- Reducing leaked signal
- Reducing correlation between leaked signal and computed values

In the first category, a simple solution can be found with jamming the leaking channel with noise. This does not eliminate the correlation between the leaked signal and the secret but it means that the attacker will require more traces to recover the secret.

The second category contains solutions as constant timing computation. Because computation time sometimes depend on the secret, this opens to the

attacker the possibility of timing attacks. In order to mitigate this problem, a random delay can be added or the computation can be done in a constant time. Even though this technique is not always easy to implement, it also reduces the performances of the device.

Most countermeasures for SCA come with surface, time or energy trade offs.

1.3 Deep Neural Network recovery

1.3.1 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.

1.3.1.a Neuron

Each node, called neuron, of the network is linked to the previous layers and its value determined by the following equation where w_i are the weights of the layer, b is the bias and f is the non linear activation function. The representation of this architecture can be found on Figure 2

$$output = f(\sum w_i * x_i + b) \tag{1}$$

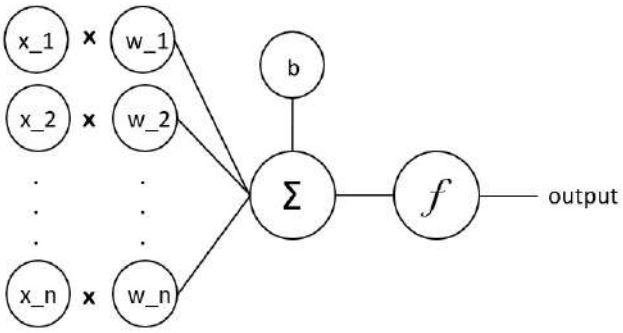


Figure 2: Representation of a neuron

1.3.1.b Activation function

The activation function is a non linear function which allows the network to approximate non-linear functions. The most popular ones are tanH, ReLu, sigmoid or leakyReLu or SoftMax. They have different properties and use-case which are application dependent. Figure 3 shows the most used ones. Their shape directly influences the learning process and how the network evolves during learning.

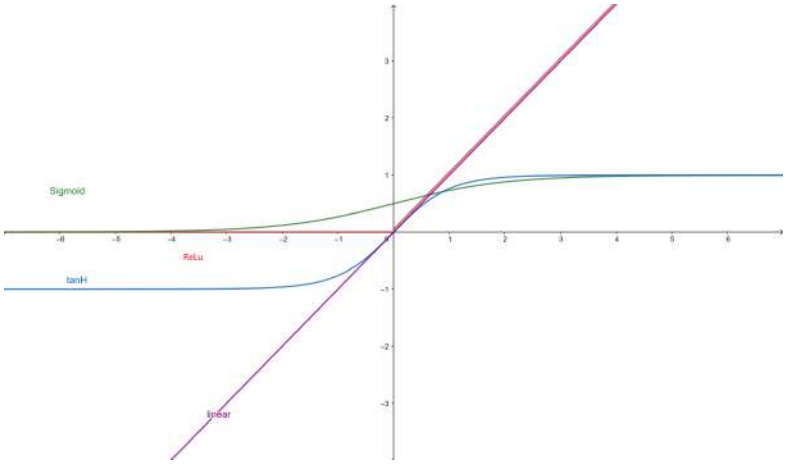


Figure 3: Representation of four usual activation functions

1.3.1.c Multi Layer Perceptron

Multi Layer Perceptron (MLP) is the simplest kind of neural network possible. It is composed of one input layer, one output layer and several hidden layers. These layers are composed of neurons connected to the previous and next layers. The architecture of a MLP with two hidden layers is shown on Figure 4.

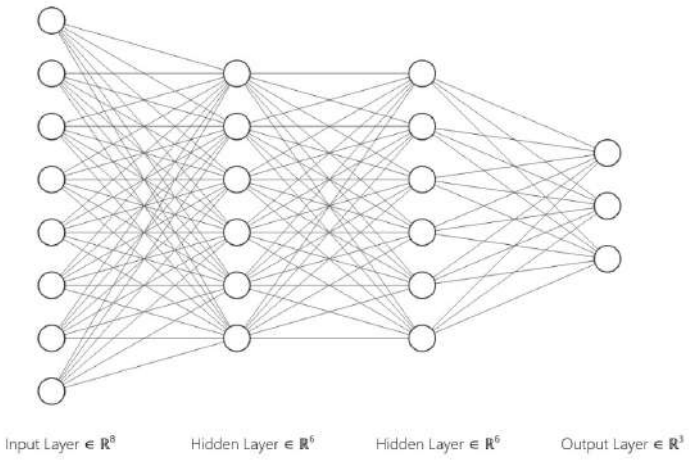


Figure 4: Architecture of a MLP with two hidden layers

MLP are only made of fully-connected layers but there are also other kind of layers with different goals.

1.3.1.d Training process

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. This algorithm estimates the error made at each node of the network based on the inputs and outputs. After changing the weights based on these estimations, the training proceeds with another input. Because this process is heavy to run, it is usual to proceed in 100 inputs batches and only update the weights of the network every batch.

The number of time this whole process is repeated is call epochs. The more epochs are done, the more a neural network can learn from the training data until overfitting. The network is called overfitted when the accuracy on training data is good but it cannot perform as good on the testing data. It basically learns to recognise every specific input which is not the goal of classification tasks.

1.3.1.e Convolution layers

Convolution Neural Networks (CNN) use a specific kind of layer which convolve the input with a small matrix. They are usually used in image processing tasks. This kind of layer is useful to extract subfeatures of the images.

Because the matrix used in the convolution is small, it allows processing the images with a few parameters : usually 5x5 matrices. The use of fully connected layers in this kind of application in not practical because of the number of neurons required to process an image.

This behaviour also allows the network to be shift invariant, allowing the features to be extracted wherever they are on the original image.

1.3.1.f Binary Neural Networks

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.

1.3.1.g Hyperparameters

The hyperparameters of layers are all the parameters linked to this layer which are crucial for a good training process. In a fully connected layer, it would be the number of neurons. In a convolution layer, there are many more parameters such as the matrix size, number of matrix used, stride or padding.

1.3.2 Adversarial attack method

Adversarial Learning (AL) aims at selecting special data for training in order to optimize the learning process. It relies on finding data 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.

1.4 Leakage evalutation

In order to test for information leaks, the Welch's t-test statistical tool is used to asses if two sets of data are different or not. This is done by computing a t-value which is computed by the formula 2 with X_1 and X_2 being the two test sets of

size n and m. The t-value can show if the means of two sets are significantly different or not.

$$t\_value = \frac{\overline{X_1} - \overline{X_2}}{\sqrt{\frac{Var(X_1)}{n} + \frac{Var(X_2)}{m}}} \tag{2}$$

The higher the t-value, the more unlikely the two sets are to have the same mean. A value of 4.5 is accepted as already showing significant difference with a confidence of 99.99% for sets with more than 100 traces. [WO19].

1.5 Related work

1.5.1 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.

1.5.2 Test setup

In their articles, Batina et al. [Bat+19b] and Yu et al. [Yu+20b] recorded the EM field thanks to a electromagnetic 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.

1.5.3 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]. Figure 5 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 with the control over the input of a neuron.

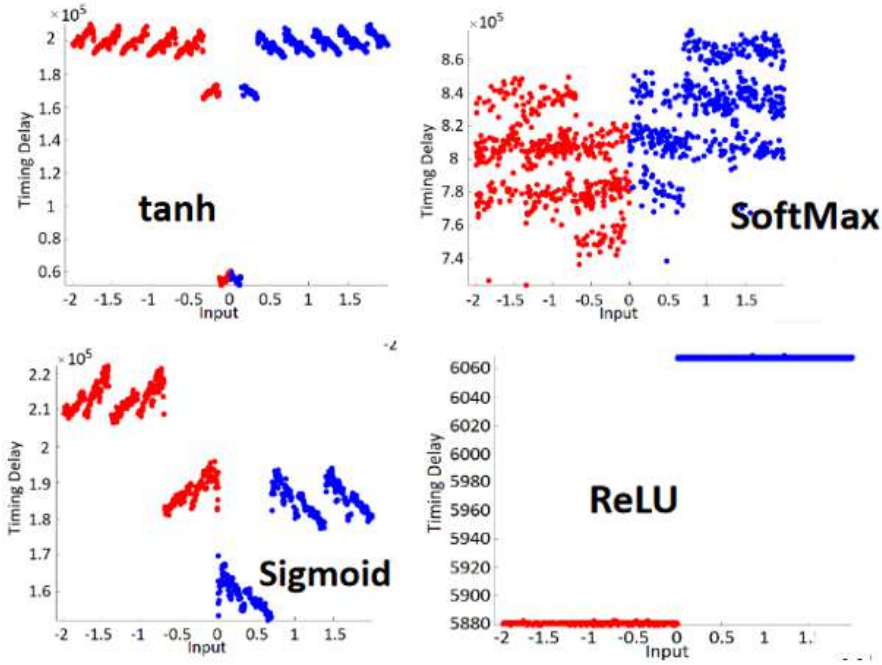


Figure 5: Influence of activation function on processing delay (ns)[Bat+19b]

1.5.4 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 Figure 6. Noting $b_{31}...b_0$ the bits of the stored number a, the value of this number can be recovered with this formula:

$$a = (-1)^{b_{31}} \times 2^{(b_{30}...b_{23})_2 - 127} \times (1.b_{22}...b_0)_2 \tag{3}$$

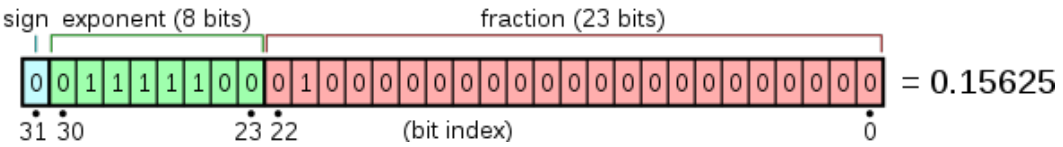


Figure 6: Example of 32 bits float : `en.wikipedia.org/wiki/IEEE_754`

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. Figure 7 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 otherwise (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.

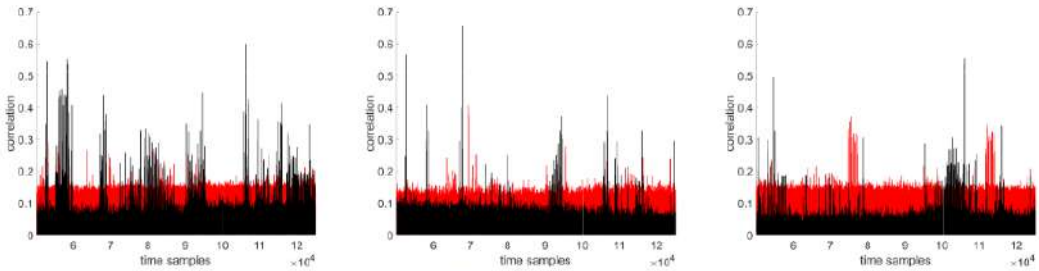


Figure 7: Correlation of different weights candidate on multiplication operation [Bat+19b]

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 $[0; 2^7 - 1]$. Figure 8 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.

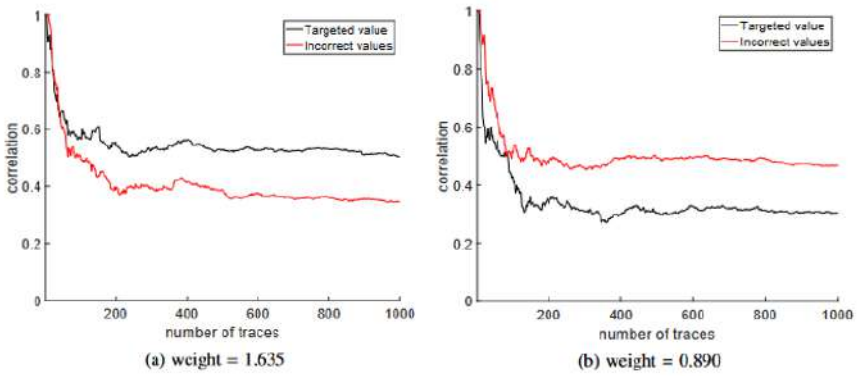


Figure 8: Correlation comparison between correct and incorrect mantissa of the weight [Bat+19b]

1.5.5 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. Figure 9 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 9: 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 Figure 10.

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

Figure 10: 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 Figure 11. 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.

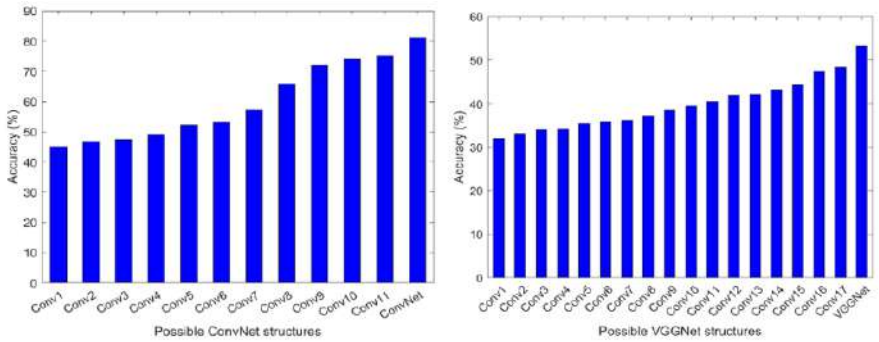


Figure 11: Performances of ConvNet and VGGNet possible architectures [Yu+20b]

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 Figure 12 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 12: Accuracy on test sets. RM = Random, FeatureAdversary = FA, FeatureFool = FF [Yu+20b]

2 Attack Setup

2.1 Trigger signal

2.1.1 Trigger setup

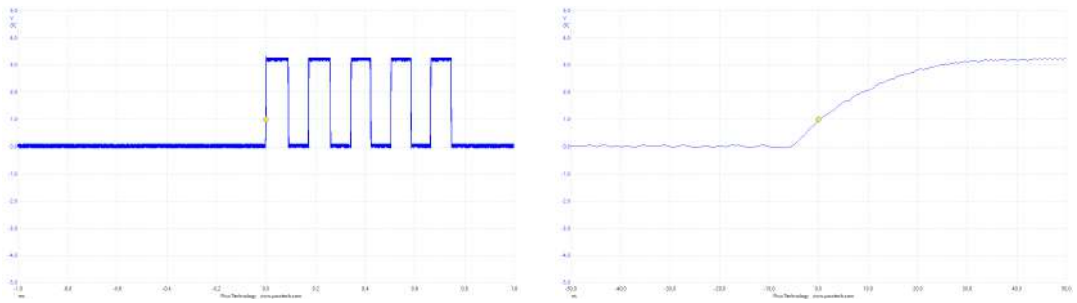
In order to synchronise the computation and the measurements from the oscilloscope, a trigger signal needs to be implemented with the GPIOs ports available on the ZCU104 board.

Using the Vivado, Petalinux and Vitis tools from Xilinx, the original boot image for the Xilinx ZCU104 board can be modified to allow the use of the 4 LEDs mounted on the board. A thin wire can then be soldered to the LED so that it acts as a trigger signal for the oscilloscope.

In order to toggle the trigger state, a 0 or a 1 needs to be written to a file. This method is not the fastest which exists to control a pin state but the next section covers the performances and shows that they are good enough for this use-case.

2.1.2 Trigger performances

With a simple loop program, the maximum frequency and raise time of the trigger signal can be measured. With values of $6.6kHz$ and $50ns$, the trigger can be considered as good enough for our purpose here because the maximum speed of image processing by the board is around 2000 $images/s$. The short raising time ensure that the trigger of the oscilloscope will be reliable, precise and will not induce time variations.



(a) Max frequency of the trigger signal **(b)** Raise time of the trigger signal

Figure 13: Max frequency and raise time of the trigger signal

2.2 Deep Processing Unit

The Deep Processing Unit (DPU) is an intellectual property (IP) core from Xilinx which allows parallel computing for Machine Learning on UltraScale+ FPGAs from the same manufacturer. This IP comes in different sizes allowing different amount of computing power with the FPGA depending on the space required on the chip by other parts of the customer's design. The DPU is implemented at the same stage as the trigger in the Vitis tool. The following Figure 14 is extracted from the Xilinx documentation and explains the top-level block diagram of the DPU IP. It shows its relation with the external memory and processing unit. The high performance computation is done in the Processing Engine (PE) blocks which size and number vary depending on the size of the DPU chosen by the user.

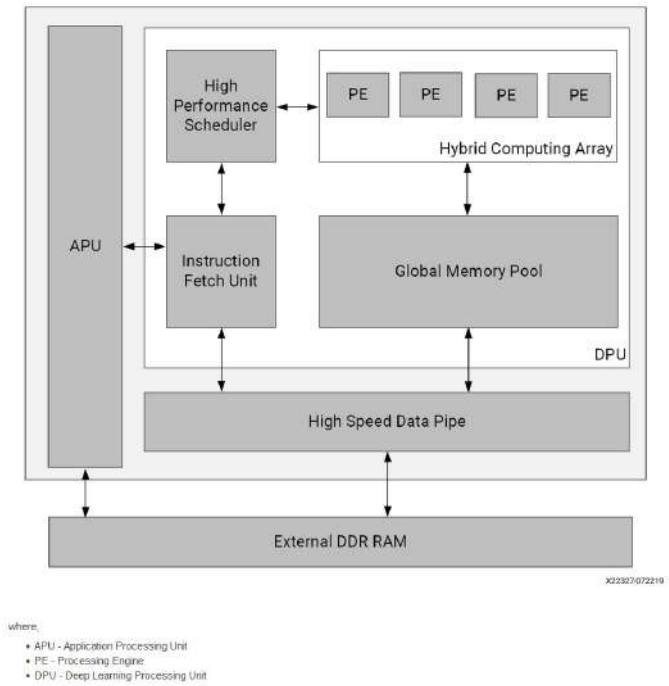


Figure 14: DPU top-level diagram : Xilinx PG338 documentation

The DPU comes with the Vitis-AI environment which allows users to efficiently implement any Neural Network from TensorFlow, Pytorch, or Caffe. For example, Vitis-AI takes the pytorch *.pth* model as an input and gives a python app to run on the board along with a *.xmodel* file which contains all the weights and parameters of the model.

As the DPU only works with integers weights, the Vitis-AI tool has to process the network before creating the final *.xmodel* file. A process called quantizing is in charge of changing the weights of the network to integers without loosing accuracy. The compiling step changes the human readable *.xmodel* file into another one which can only be read by the FPGA when initializing the network . The Figure 15 shows the different steps to required to translate a pytorch model to a compiled file able to run on the ZCU104 board.

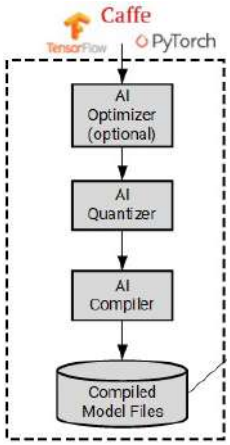


Figure 15: Vitis AI flow to compile a model

2.2.1 Performances

The amount of parallel computations depends on the size of the DPU implemented. For our experiments, the DPU implemented is the biggest available. It allows 16 input channels, 16 output channels, and 8 pixels to be processed at the same time. These numbers are shown is the Figure 16 for the *B*4096 architecture which brings the number of operations to 4096 per clock cycle.

Table: Parallelism for Different Convolution Architectures

DPUCZDX8G Architecture	Pixel Parallelism (PP)	Input Channel Parallelism (ICP)	Output Channel Parallelism (OCP)	Peak Ops (operations/per cycle)
B512	4	8	8	512
B800	4	10	10	800
B1024	8	8	8	1024
B1152	4	12	12	1150
B1600	8	10	10	1600
B2304	8	12	12	2304
B3136	8	14	14	3136
B4096	8	16	16	4096

Figure 16: Performances for different DPU sizes available

The DPU unit requires two clocks in order to run properly, one of them being twice the frequency of the first. During the setup of the DPU, these frequencies have been set to 100 and 200MHz.

2.2.2 Black Box

As the DPU unit is an IP from Xilinx, it is a true black box. Its working principles are briefly described in the Xilinx documentation but most of the inner workings of the system are kept secret. Although this might be considered as an obstacle to the attack conducted further in this thesis, it also means that no shortcuts have been taken.

2.3 Probe and measurements

As the two clock frequencies given to the DPU unit are 100 and 200MHz, a picoscope 5203 can be used for measurements as its bandwidth goes up to 250MHz.

The goal is to recover information from electromagnetic emissions of the board. A $RF - U2.5 - 2$ near field probe is used to capture the electromagnetic signal coming from different parts of the board. The signal from the probe is then pre-amplified by a $PA303$ Amplifier. This signal then pass though an impedance matcher to the oscilloscope. The setup is shown on the Figure 17. Each component has been highlighted in color : dark green is the picoscope, light green is the probe stand, red is the near field probe, yellow is the trigger probe, purple is the remote connection from the board to the computer, dark blue is the impedance adapter and light blue is the $PA303$ amplifier.

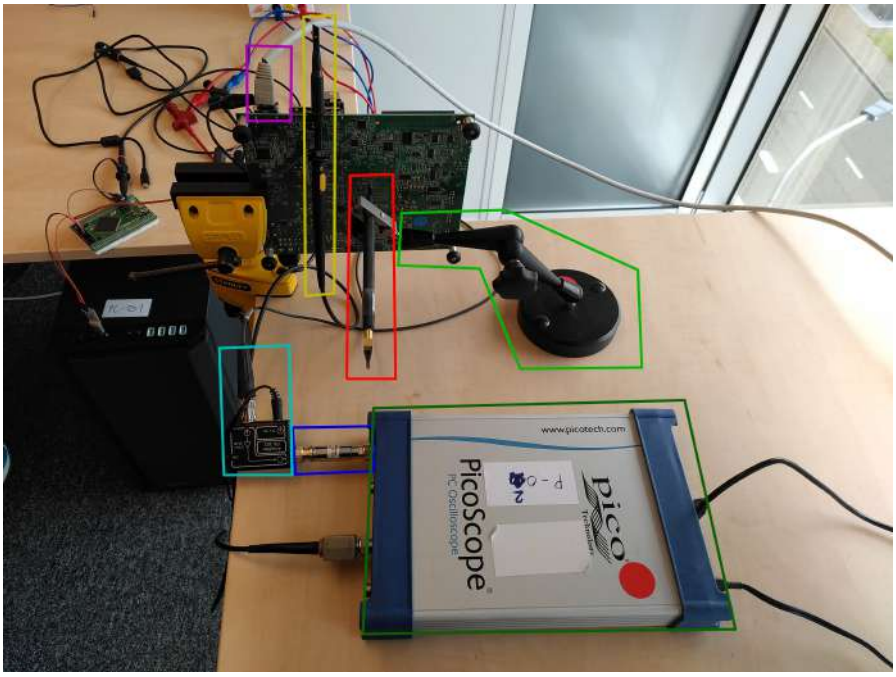


Figure 17: Measurements setup

2.3.1 Near field probe placement

In order to find a good measuring spot on the board, a loop of inference has been implemented with the DPU and the probe was placed at different strategic locations on the board. By recording the amplitude and shape of the measurements, the best leaking spot can easily be found in a few minutes. The strategic location are mainly decoupling capacitors on the opposite side of the main chip and traces going in and out of the chip. It was found that one capacitor on the back of the board had little noise and good amplitude which is optimal for side channel analysis. This capacitor and exact probe placement can be seen on Figure 18

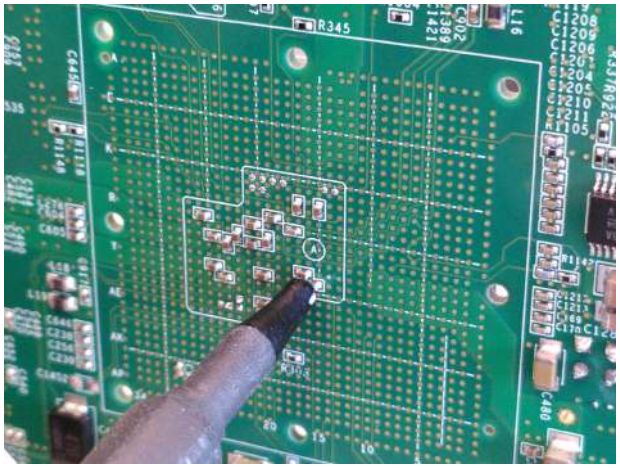


Figure 18: Probe placement

2.3.2 Background noise

Measurements of the background noise and frequencies were conducted. These measurements were done away from the board and any noise-generating device. The Figure 19 shows in blue the measurement of the probe and its frequency spectrum. A few spikes caused by local radios can be seen around the interesting $100MHz$ frequency but they are not overlapping with it.

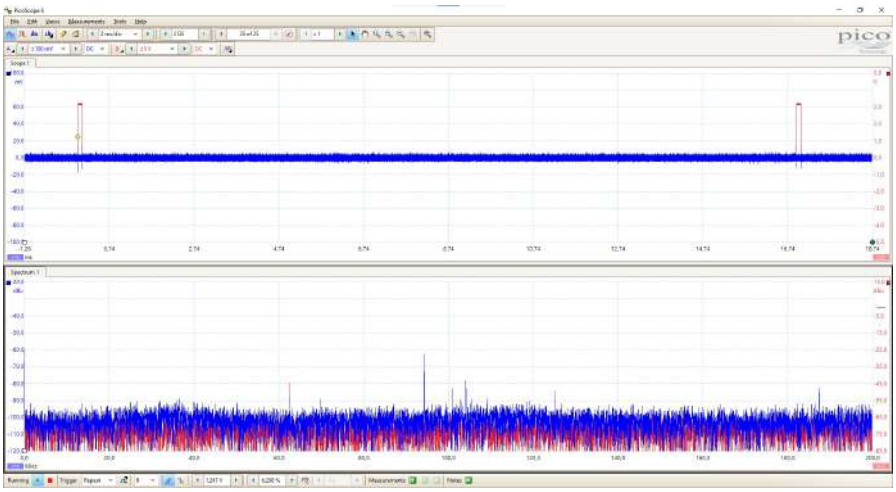


Figure 19: Background ambient noise and spectrum

After choosing the location of the near field probe, the effect of running board can be seen on the measurements. The Figure 20 has been captured without

any task running on the DPU unit but with the board turned on. The signals measured come from the DPU rest noise. The 100 and 200MHz frequencies can clearly be identified with high levels. Some noise between 20 and 60MHz also appear on the spectrum.

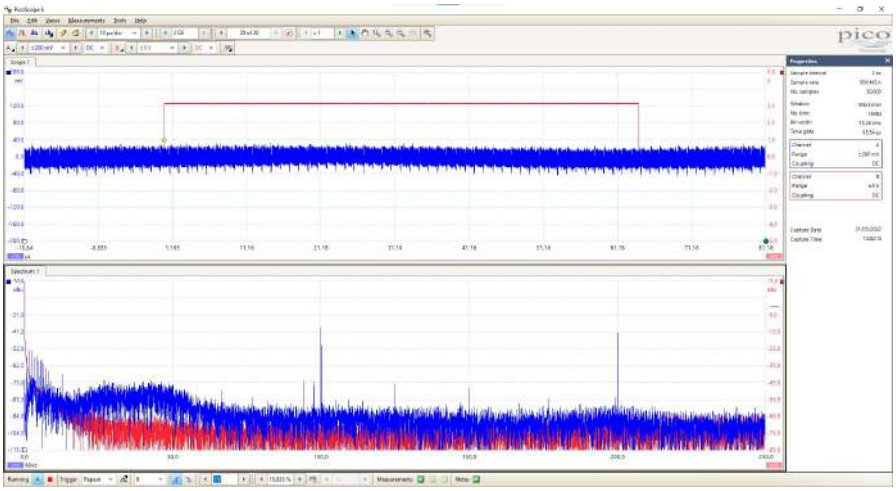


Figure 20: Measurement of electromagnetic field without DPU activity

3 Simple Electromagnetic Analysis

3.1 Shape identification

3.1.1 First Measurements

After synchronising the trigger signal and the DPU computation, traces were recorded when a simple CNN was running. The measurements can be found in Figure 21. The red trace is the trigger signal which goes high when the DPU computation is called in the python app and goes down when the results come back. Compared to Figure 20 without DPU activity, this trace contains a lot more noise. At $1ms$, a peak of activity appears for a short time. Zooming on this peak reveals 13 peaks as shown on Figure 22. These peaks vary in length, amplitude and spacing but some groups of peaks can be identified as similar to each other.

The spectrum shows a lot of noise near the two frequencies of the DPU and in the low frequency range. This clearly indicates that the DPU activity is affecting the current running through the capacitor under measurement. As the CNN under test only has 5 convolution layers, each of the 13 peaks does not correspond immediately to a layer being processed. The link between peaks and layers will be investigated later.

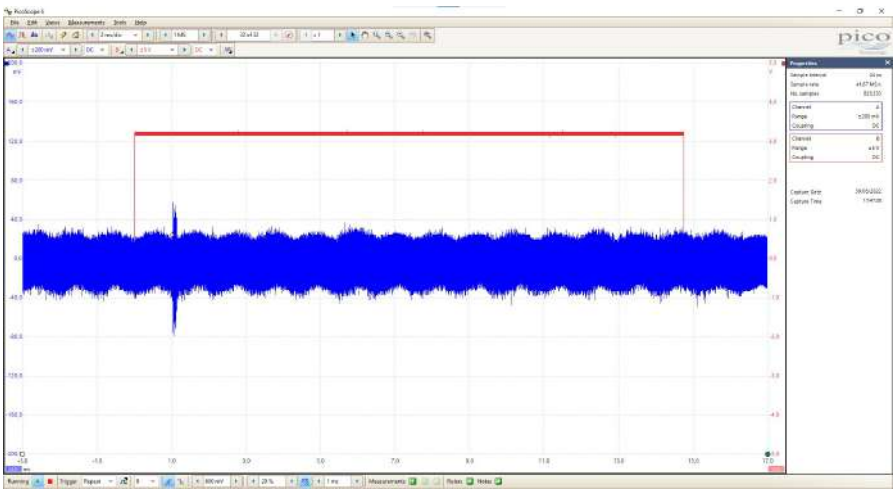


Figure 21: Measurement of electromagnetic field with DPU activity

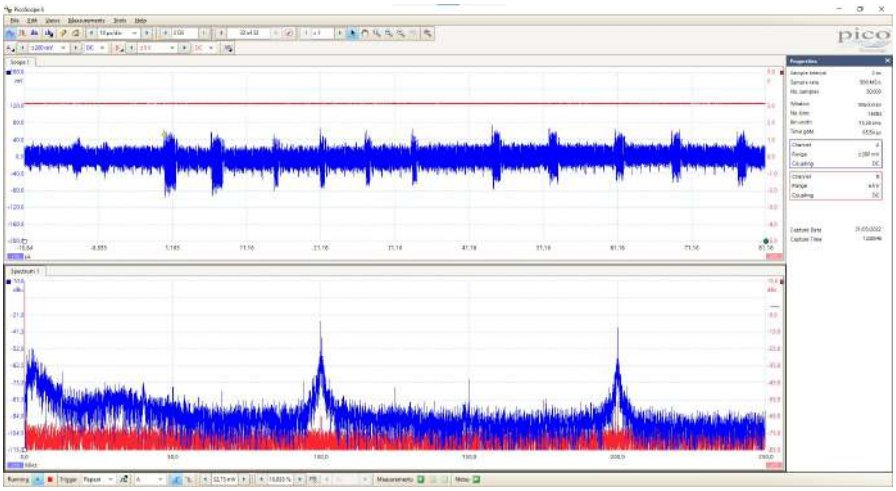


Figure 22: Measurement of electromagnetic field with DPU activity - Zoomed

3.1.2 Effect of number of channels

The effect of the number of channels on the computation has been measured using three similar CNN architectures. The representation of the architecture is given on Figure 23 and the pytorch code for it is the following :

```
import torch
import torch.nn as nn

network = nn.Sequential(
## Layer 1
nn.Conv2d(1,16, kernel_size=5, stride=2, padding=1),
nn.BatchNorm2d(16),
nn.ReLU(inplace=True),

## Layer 2
nn.Conv2d(16,**32**, kernel_size=5, stride=2, padding=1),
nn.BatchNorm2d(**32**),
nn.ReLU(inplace=True),

## Layer 3
nn.Conv2d(**32**,10, kernel_size=3, stride=3, padding=1),

nn.BatchNorm2d(10),
nn.Flatten
```

```
20 )
```

Where the **32** value is either 16, 32, or 64 depending on the architecture.

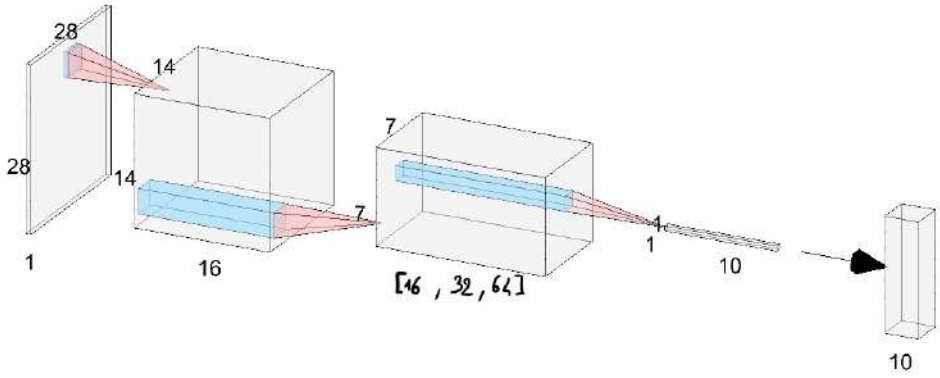
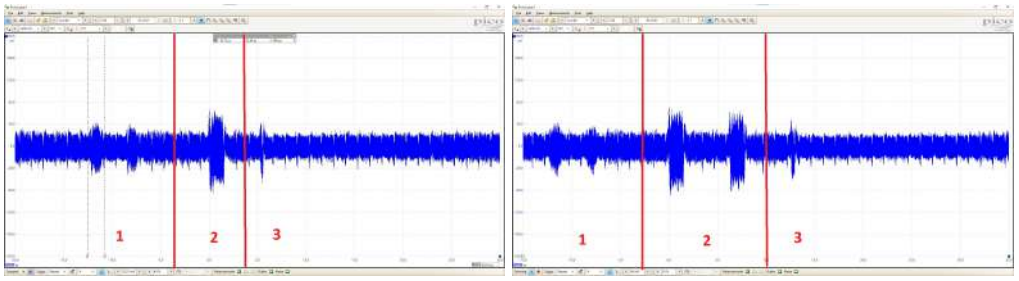
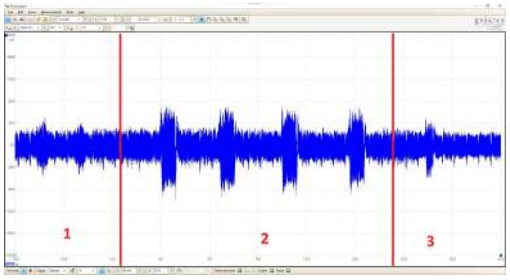


Figure 23: Visual representation of the CNN used to measure the influence of number of channels



(a) 16 channels **(b)** 32 channels



(c) 64 channels

Figure 24: Influence of the number of channels on the signal

The traces of the three different architectures can be seen on Figure 24. Comparing differences and similarities, the three layers can be distinguished from one-another as only the second and third layer are to be influenced by the change in number of channels. The approximate boundary between the three layers have been marked on the three traces.

The DPU can run 16 channels (input and output) in parallel, it splits the second layer computation into 2 or 4 blocks for the second and third architecture. These blocks are easily detected for the second layer but also appear on the third layer within the small peaks. The difference of separation for the second and third layer can be explained by the number of clock cycles involved in these two calculations. This number can be calculated with the following equation and are found in Table 1:

$$n_{clk} = \frac{n_{pixels} * n_{channels}}{parallelism_{pixel} * parallelism_{channels}} \tag{4}$$

$n_{channels}$	$parallelism_{channels}$	n_{pixels}	$parallelism_{pixel}$	n_{clk}
16	16	49	8	7
32	16	49	8	13
64	16	49	8	25

Table 1: Number of clock cycle for the third layer of each architecture

3.1.3 Number of pixel

Even though the DPU unit provide some parallelisation of the pixels, the time of execution of each layer is still dependent on the total number of output number of pixels of this layer. Figures 25b and 25a show the theoretical and real time of execution for different image sizes. The following single layer network was implemented with different images sizes to measure the execution times. 100 traces were recorded for each layer and the execution time of these layers was measured on these traces.

```
network = nn.Sequential(
nn.Conv2d(16,16, kernel_size=5, stride=1, padding=0)
)
```

The theoretical execution time is computed by this formula :

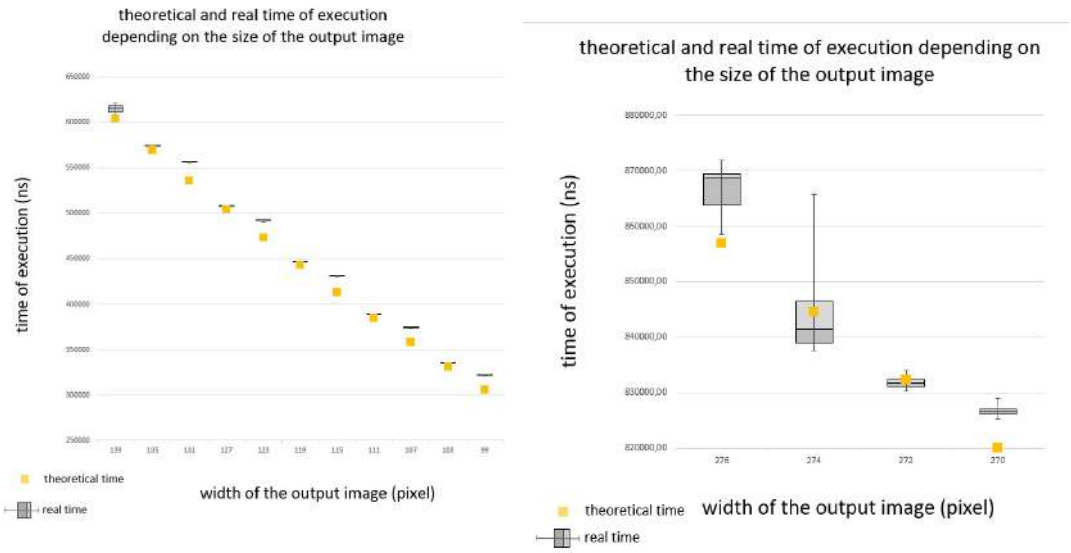
$$t_{theoretical} = \frac{outputsize^2 * kernelsize^2}{clock * parallelism} \tag{5}$$

where

$$outputsize = \frac{inputsize - kernelsize + 2 * padding}{stride} \tag{6}$$

The graphs clearly show that the output size can be determine through the measurement of the execution time. Figure 25a shows that even small differences of 2 pixels can be detected. The box plots show that the execution time is constant across varying input and that the time does not vary much between traces.

These graphs show the link between execution time and image size. This is to be expected but it confirms that Xilinx did not implement any countermeasures in order to decorrelate execution time and image sizes. Because the size of the output image depends on the input image size and hyperparameters of the networks layers, in a scenario where the attacker knows the size of the input layer, information about the first convolution layer can be recovered.



(a) pixel size 139 - 99 **(b)** pixel size 276 - 270

Figure 25: Influence of the number of pixels on the execution time

3.1.4 Identification of the end of a layer

Because the DPU unit often separates the operations of one layer into several blocks, it makes it harder to distinguish the beginnings or ends of layers. The idea of this part of the attack is to train a machine learning algorithm to recognise the end of a layer.

The data used for training the neural network comes from the traces collected on a a convolution network which contains a layer composed of 32 blocks followed by two layers composed of only one block. One of the traces from this architecture can be seen on Figure 26.

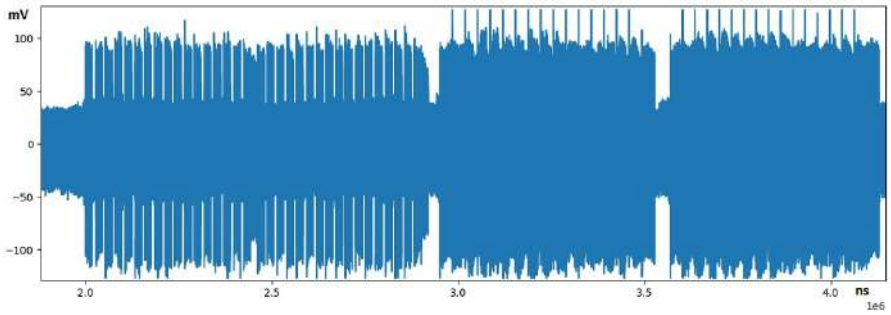


Figure 26: A trace used to train end of layer recognition

The input of the recognition network is 8000 samples traces which are extracted from the original traces. This window width corresponds to $8\mu s$ which is smaller than the smallest space between peaks on the traces. This size has been chosen to avoid the network to recognise falling edges quickly followed by rising edges. Then the analysed window only contains one rising edge, one falling edge, or no edge at all. The traces have been labeled with either 0 if it does not contain the end of a layer or 1 if it does. Figure 27 show some examples of the dataset with their labels. By human eye, the windows containing a falling edge are really similar and do not appear different. Because the data used contain much more 0 labels than 1, a lot of it have been removed from the dataset to bring the percentage of 0 labels in the dataset from 99% to 90%.

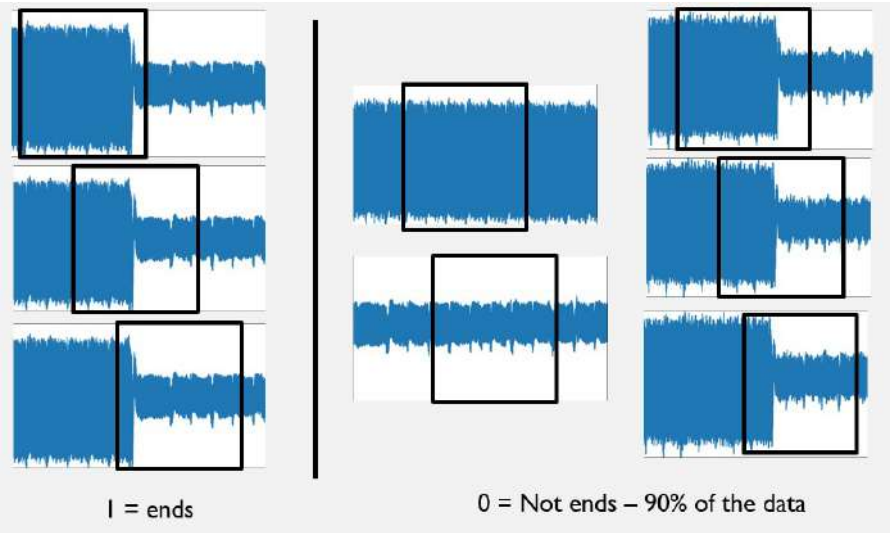


Figure 27: Examples taken from the dataset for end of layer recognition

After training the model with pytorch, it reaches an accuracy of 98.8%. Figure 28 shows the testing accuracy with increasing epochs compared to random guess and proportion of 0 in the dataset. Because the input of the network is a $8\mu s$ window, a python script can be written to infer the model on different part of the original trace in order to detect the separation of the layers.

To improve accuracy, the input of the network is set as a sliding window moving $0.8\mu s$. This means that the falling edge of the end of a layer will be analysed 10 times. To differentiate real ends from wrong guesses from the network, the program will search for 8 to 10 predictions close together.

This attack aims at slicing big traces into layers for further processing. In itself, this attack is able to determine the number of convolution layers contained in a network.

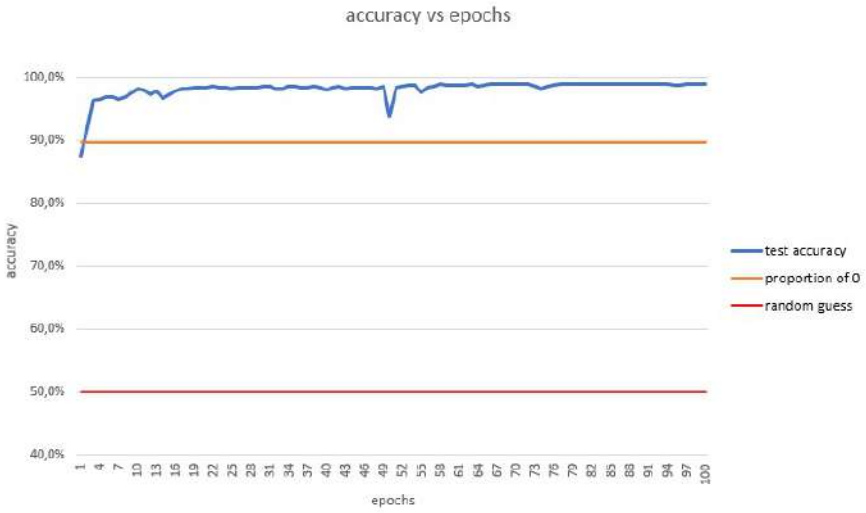


Figure 28: Training accuracy with epochs on the detection of end of layers

3.2 Machine Learning Hyperparameters reverse engineering

3.2.1 Threat model

In this profiling attack, the attacker is supposed to have access to a copy of the targeted device where he can control the networks implemented on the board. After recording traces of different layers and training a neural network to classify the traces, the attacker feeds an unknown trace to this network and is able to recover hyperparameters of this convolution layer.

Put to use with the previous attack with end of layer recognition, one could reverse engineer all the convolution layers of a network one by one.

3.2.2 Datasets

The training dataset is made of traces from 189 different convolution layers with different parameters. They are all based on this pytorch architecture :

```
network = nn.Sequential(
nn.Conv2d(nchannels_in, nchannels_out, kernel_size, stride,
    padding=0)
)
```

where the parameters are chosen is the list bellow:

- nchannels_in = [1, 2, 4, 8, 16, 32]
- nchannels_out = [2, 4, 8, 16, 32, 64]
- kernel_size = [1, 3, 5]
- stride = [1, 2, 3]

All possible combinations which have $nchannels\_in < nchannels\_out$ have been implemented. These hyperparameters have been chosen because they are among the most commonly used in convolution layers.

These layers have a time of execution between $1ms$ and $25ms$. Because the frequencies of the DPU unit are 100 and $200MHz$, and to have a safety margin with the Nyquist criteria, the sampling rate of $1GS/s$ has been chosen. Because this choice causes each trace to be 25 Million points, the amount of data created by the 189 different layers is enormous. To manage this data, each trace is preprocessed to reduce its size to 250k points. To achieve this reduction, the absolute value of the traces is resampled with averaging a window of $1\mu s$ and a step size of $0.1\mu s$. While greatly reducing the amount of data handled by the classifier network, this technique allows to preserve as much information as possible from the original traces.

The dataset is composed of 18900 traces, 100 for each different layer, which have been randomly separated in 80% for training purposes and 20% for testing.

Figure 29 shows 4 traces used for training the network. The overall shape of the trace varies a lot with the architecture. These traces show the big difference in timing between small and big convolution layers.

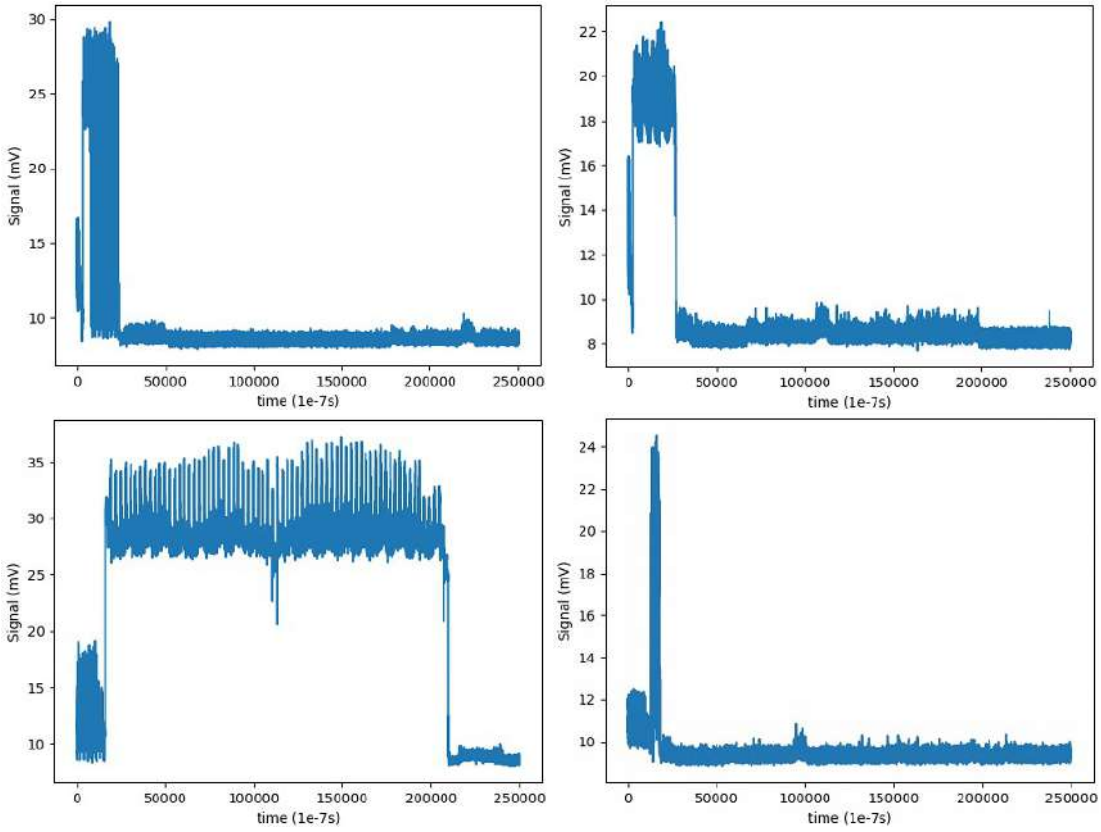


Figure 29: Examples of traces used for training the network

3.2.3 Classifying Network

The network trained to classify the different hyperparameters is the one depicted in the Firgure 30. It contains four convolution operations and three fully connected layers. This architecture has been chosen because of its simplicity and common use in simple classification tasks. The number of outputs depends on the number of classes of the targeted hyperparameter. For the number of input and output number of channels, there are 6 outputs and for the kernelsize and stride, there are 3. A multi-label approach to the problem, which consists in having only one network for the four hyperparameters, has been considered but having multiple networks allowed for different parameters for training and independent testing.

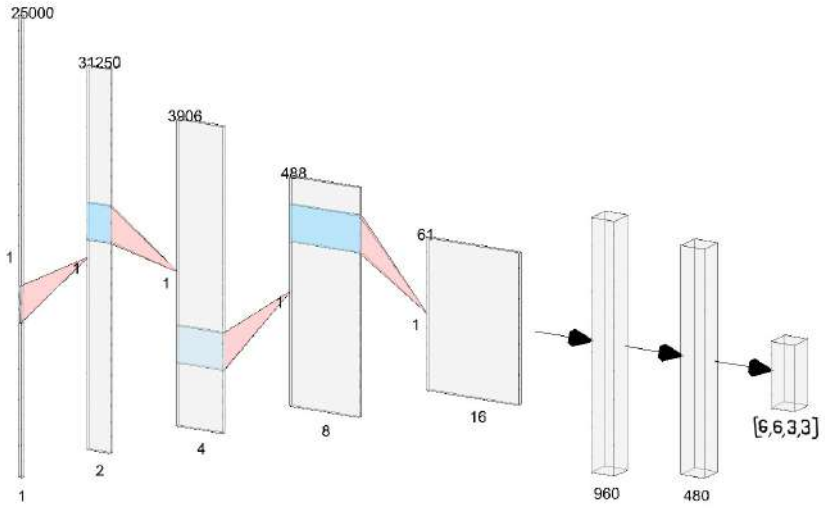


Figure 30: Visual representation of the CNN trained to classify the hyperparameters

3.2.4 Training

Two techniques have been tried for training the networks : classification and regression. The first one consists of the network having as many outputs as classes and choosing the biggest output as the "choice" of the network. Regression differs in the way that it outputs a single positive number. This number is considered as the output of the network. This means that the regression technique can output numbers which are not in the original training classes. This technique may allow more generalisation because the network can guess new classes but it is much more expensive to train.

Training was done on a shared 128cores CPU and took at most 2 days per network. For classification tasks, the training time was down to less than two hours.

3.2.4.a Regression

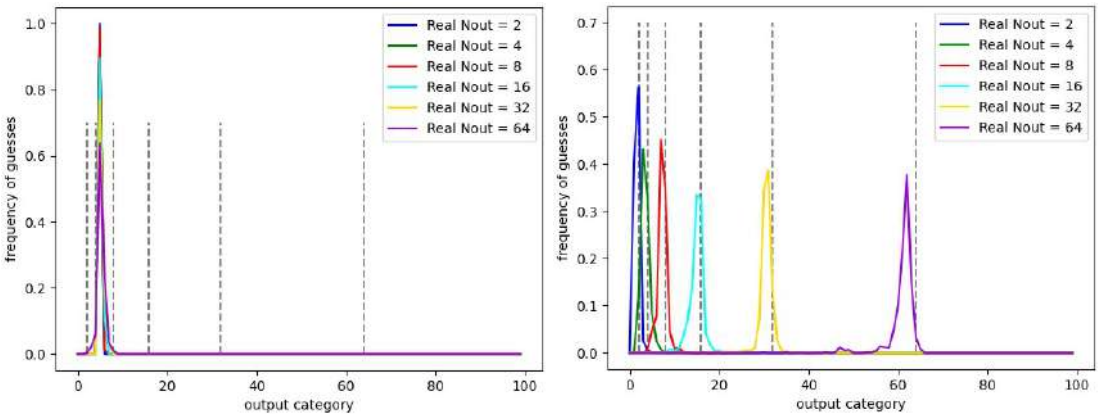
A regression approach to this problem has been tried. This technique consists of outputting a single number with the neural network and considering this as the output rather than having as many outputs as classes and taking the class with the biggest value as the output. This technique allows more generalisation

because the network can guess numbers between training classes but it is also much more expensive to train. Figure 31b shows the frequencies of the guesses from regression network for the 6 classes for the *nchannels_out* hyperparameter, each color representing a class.

At the beginning of training, the network is not able to separate the different classes. Figure 31a shows that the network mostly outputs one class for all traces. During the training, the network is able to separate the guesses to achieve classification. The guesses are not always on the exact number but are clearly separated from one another. Figure 31c shows the evolution of the guesses for the class 64 over several epochs during training.

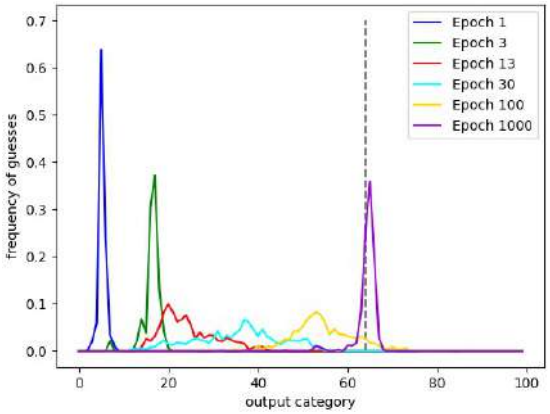
Because the network outputs a positive number, an accuracy definition has to be defined. It has been chosen to allow at most half of the space between classes as margin of error for the output to be counted as accurate. This choice has been made to prevent overlapping of the different training classes while allowing some margin of error. Taking the 8 category of the $nchannels_{out}$ as an example, to consider a guess as accurate, it will have to be in the 6 to 12 interval because the closest classes are 4 and 16.

The accuracy results and comparison with the classification method are held in Section 3.2.8.



(a) guesses of the network at epoch 3 for all classes

(b) guesses of the network at epoch 1000 for all classes



(c) class 64 guesses for different epochs

Figure 31: Regression technique performances

3.2.4.b Classification

The training implements some techniques to prevent over fitting such as data augmentation. This technique consists of adding noise to the training data while training. As mentioned, it reduces over fitting and improve generalisation. Figure 32 shows the training process for different noise amplitude. The test has been conducted with three different data augmentation parameters, 1, 3, and 10% of noise. It can clearly be seen that the right amount of noise can improve learning speed and testing accuracy. The 1 and 10% curves show to little or to much noise as the learning is chaotic and final accuracy is lower than the 3% one.

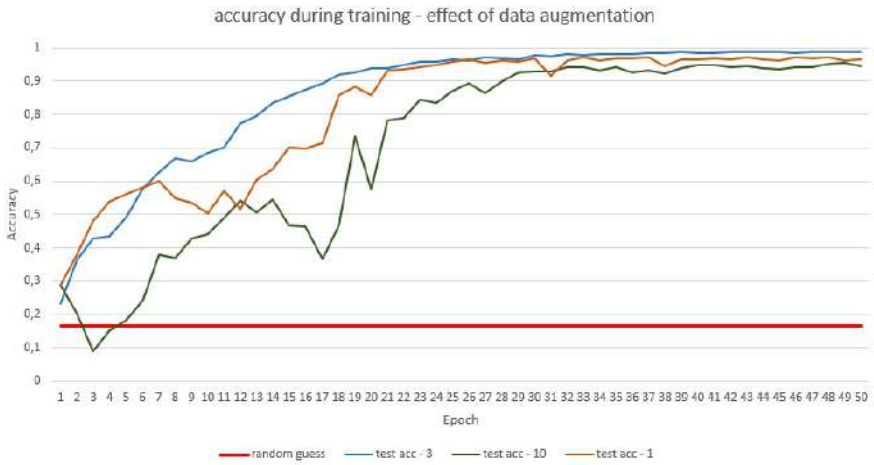
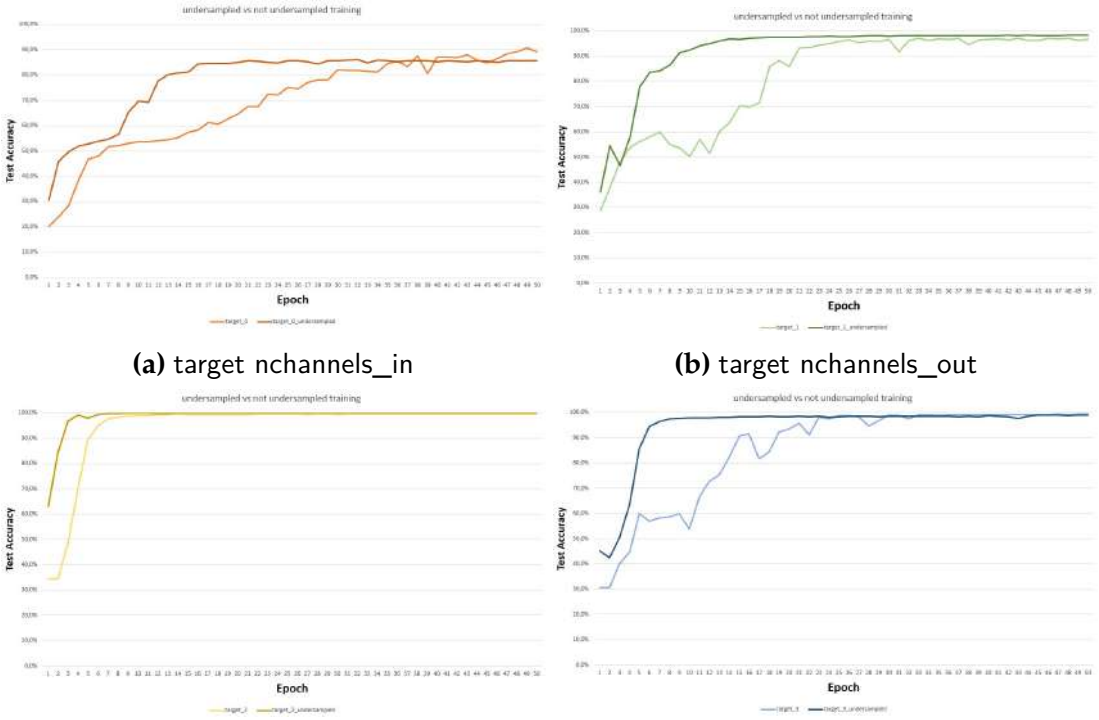


Figure 32: Training accuracy with different data augmentation parameters

3.2.5 Effect of undersampling

Because collecting traces at $1GS/s$ was really slow, the experiment has firstly been conducted using undersampling. Instead of averaging 100 samples to reduce the amount of data processed by the neural network, the sampling rate in set to $10MS/s$ which has the same effect on the amount of data acquired but drastically reduces transfer time between the scope and the computer. Compared to window resampling, undersampling adds a lot of noise to the data and is not able to conserve information on the higher frequencies.

Figure 33 shows the testing accuracy for the four targets on undersampled and not undersampled data (undersampled being the darker one). It shows a smoother and quicker rise in accuracy for the undersapled data. This result is not expected because of the data loss induced by undersampling the traces. The main explanation for this is that the window used in order to reduce the size of the data was too large and did not allow for efficient conservation of the data.



(a) target nchannels_in

(b) target nchannels_out

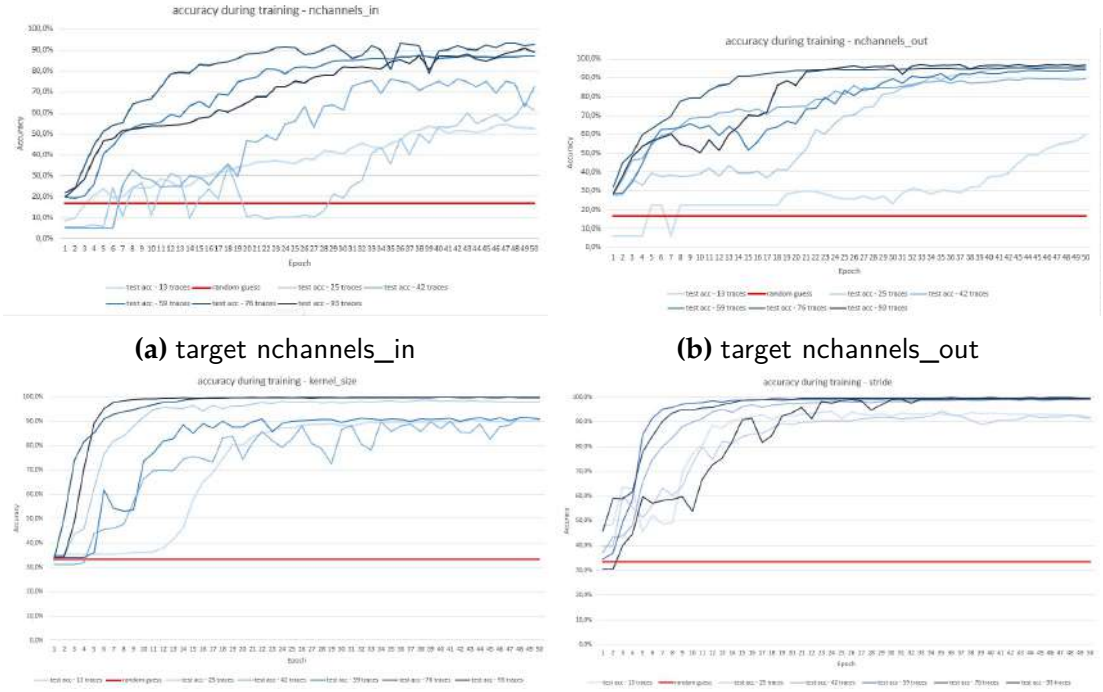
(c) target kernel_size

(d) target stride

Figure 33: accuracy vs epoch for undersampled traces

3.2.6 Effect of number of traces

Because collecting the 100 traces was done over several days, it was conducted in several steps. It allowed the training of the networks at different points of the collection in order to measure the effect of the number of traces in the dataset on the accuracy. This experiment has only been conducted using the classification technique. As expected, the more data the neural network gets as an input, the more accurate it gets and the faster it learns with each epochs. Even though this rule is generally true, there are a few outliers. These outliers might be due to poor starting conditions of the training. Figure 34 shows the testing accuracy of the training models for different sizes of dataset for all four targets. The darker the line, the more data was available for training and testing the model.



(a) target nchannels_in

(b) target nchannels_out

(c) target kernel_size

(d) target stride

Figure 34: accuracy vs epoch for different sizes of dataset

3.2.7 Evaluation on unknown layers

After training of the different networks with a good enough accuracy, traces of unknown architecture have been fed through the recognition networks to test the generalisation power of the networks. Because all the architectures with $nchannels\_in < nchannels\_out$ have been implemented for training and testing, it leaves all the architectures with equal or more input channels than output channels to test the models with for validation.

After implementing these architectures on the ZCU board and recording their traces with the exact same setup as the other architectures, the traces have been fed through the networks which have given the results given in Section 3.2.8.

3.2.8 Accuracy results

Figure 2 gives the accuracy of the networks for the four different hyperparameters with the classification and regression training techniques. The classification technique shows better results overall. This can be explained by the fact that the

regression technique is harder to implement and would need more hyperparameters tuning for an ideal training.

The accuracy obtained for hyperparameters 1, 3 and 4 are good compared to the random guess. Only the second hyperparameter does not hold accuracy from the testing dataset to the validation dataset. This might come from the fact that the testing set and validation set are too different from each-other for the network to generalize.

The use of several traces to remove outliers out of the guesses to classify is not critical in this case as the network is confident. Being right or wrong, more than 80% of all traces coming from the same architecture are being classified in the same class.

Tables 3 show number of guesses per class on the validation dataset for all four hyperparameters. This nummber of guesses has been decomposed over the real classes. These tables allows to determine the distance between the guesses and the real class. To take the *nchannels_in* hyperparameter (Table 3a) as an example, 9 guesses were made 2 classes away from the real one for 8 channels.

The ideal matrix being a diagonal one, Tables 3a, 3b, 3c show good results with the majority of the guesses being on the diagonal and very few guesses being far away from the diagonals.

Table 3d clearly shows the opposite behaviour where the network randomly guesses a class higher than the real one. For this particular hyperparameter, several network architectures and training features have been implemented to try to raise the accuracy but none were efficient.

The inacurracy of the network on the second hyperparameter can come from the choice the decomposition of all the possible architectures in the train-test dataset and validation set. Because the decomposition is based on a criteria based on *nchannels_in* and *nchannels_out*, this may have an effect on the generalisation of these classes.

method	dataset	n_in	n_out	kernel_size	stride
Classification	testing	89.2%	96.6%	99.6%	99.3%
	validation	85.6%	13.8%	82.1%	68.9%
	random guess	16.7%	16.7%	33.3%	33.3%
Regression	testing	65.2%	92.6%	92.4%	91.0%
	validation	38.1%	11.0%	82.1%	55.9%
	random guess	33.3%	33.3%	33.3%	33.3%

Table 2: Accuracy for Classification and Regression for testing and validation set

		Guessed class					
		1	2	4	8	16	32
Real class	1	0	0	0	0	0	0
	2	0	28	12	5	0	0
	4	0	13	68	9	0	0
	8	0	9	2	124	0	0
	16	0	10	1	9	160	0
	32	1	1	5	8	12	198

(a) nchannels_in results

		Guessed class		
		1	3	5
Real class	1	200	23	2
	3	0	207	18
	5	0	78	147

(b) kernel_size

		Guessed class		
		1	2	3
Real class	1	101	85	39
	2	7	153	65
	3	7	7	211

(c) stride

		Guessed class					
		2	4	8	16	32	64
Real class	2	0	55	49	27	55	39
	4	0	51	24	14	46	45
	8	0	0	32	14	40	49
	16	0	0	1	5	35	49
	32	0	0	0	0	5	40
	64	0	0	0	0	0	0

(d) nchannels_out results

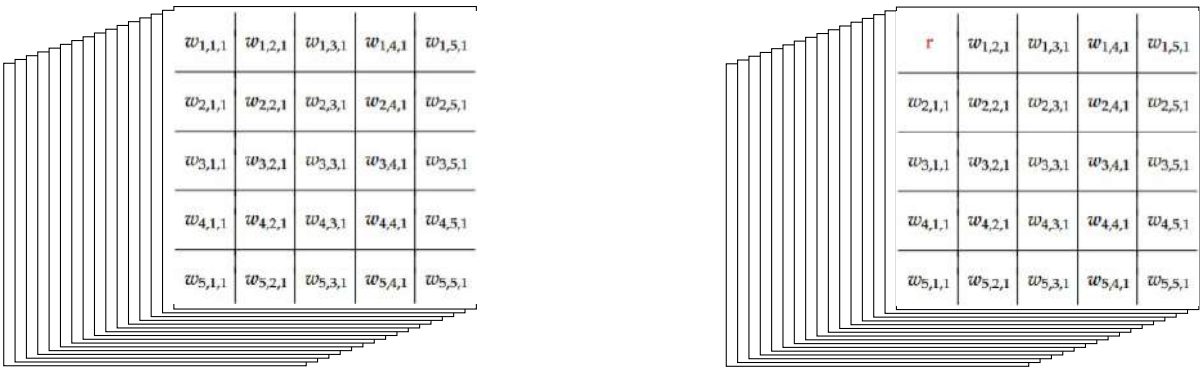
Table 3: Number of guesses per class on the validation dataset

4 Correlation Electromagnetic Analysis

This part will focus on the correlation Electromagnetic Analysis that have been performed on the target.

4.1 Single weight leak

In order to test single weight leakage, a single convolution layer has been implemented on the board. Two sets of traces where recorded : one set had all the weights fixed and the other set has the same weights except for the first one which is picked randomly. The two different sets of convolution kernels have the weights looking like Figure 35. Each convolution network is composed of 16 channels of 25 weights which combines into 400 weights in total.



(a) Set 1 with all fixed weights

(b) Set 2 with one random weight

Figure 35: Two subsets used to test single weight leakage

The traces are recorded with the same input. This corresponds to a threat model where the attacker has control over the input of the device. After recording 5 traces for each network on 5000 different networks (2500 for each set), the traces are processed to apply absolute value, window resampling and alignment. These processed traces are then fed into a ttest algorithm which can tell if two sets are statistically different or not.

Figure 36 shows the timings on which the t-value is statistically significant over the traces. The peaks seen on the grey traces correspond to the DPU activity clearly show higher t-values in blue which is to be expected because the weight values are only used during computation.

Because this is a statistical attack, the more traces are used the more significant the t-value becomes. Figure 37 show the evolution of the max t-value with the

number of traces used for the attack. With 2000 traces, the t-value already reaches significant values.

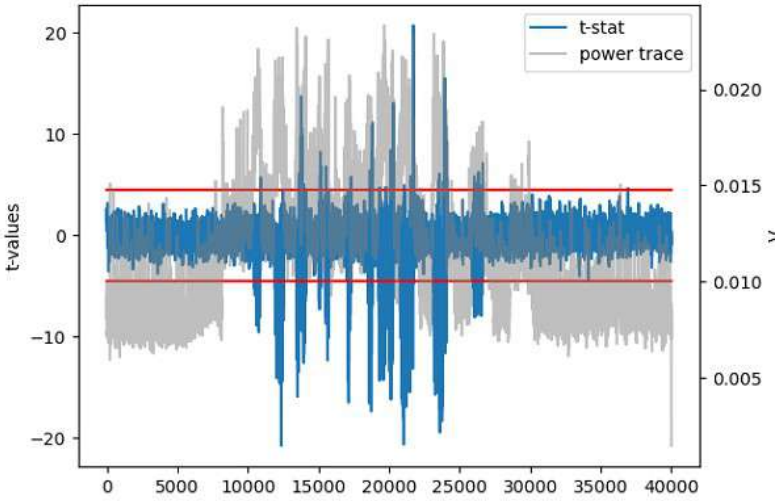


Figure 36: t-value across the traces

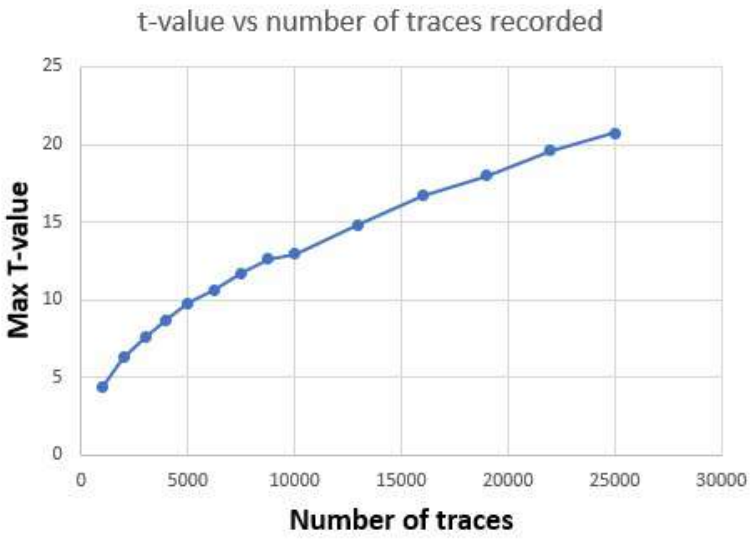


Figure 37: t-value depending on the number of recorded traces

4.2 Weight recognition

After confirming that the weights were leaking, an experiment was conducted to apply machine learning recognition to traces in order to recognise a single weight of the convolution layer. Because the weights are coded with 16 bits (2 bytes), there are 65536 possible network to run. In order to reduce this number while testing for leakage for the full set of 16 bits, it has been decided to train two different networks to recognise 256 weights each. The first one recognises weights between 0 and 256, checking for the 1rst byte leakage while the second one tests for the second byte leakage with the test of all the possible weights with the first byte set to 0. The two sets are represented on the following list.

- **First weights :** 0, 1, 2, 3, ..., 253, 254, 255.
- **Second weights :** -32768, -32512, -32256, -32000, ..., 32000, 32256, 32512.

4.2.1 First byte

After collecting 500 traces for the first 256 weights, a first test can be implemented in order to assess weight leakage within this set of data. A t-test similar to the one conducted previously is performed on the data for each weight. In total, 256 t-test have been conducted giving the following results in Table 4. The histogram of the distribution of the t-values across all classes is shown on Figure 38. Most of the values being above the 4.5 threshold, it can be considered that the weights are leaking and should be, at least partially recoverable.

	min	max	average
t-value	3.72	16.45	7.86

Table 4: T-value for the first byte of the weights

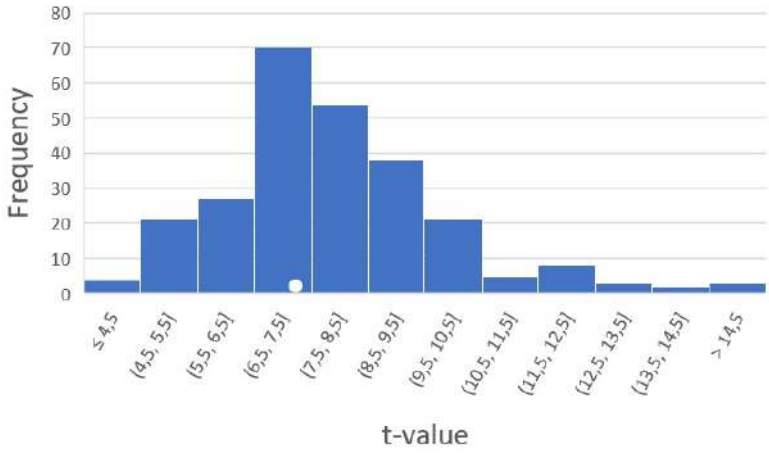


Figure 38: histogram of the t-value across all classes

With all the weights being between 0 and 255, a neural network has been implemented in order to classify the traces in 256 classes corresponding to each weight. Using the same training techniques as in Section 3.2, a neural network has been trained with the classification technique. The hyperparameters of the training are available in Table 5

layer type	parameters	activation
Conv1D	filters: 4, kernel-size: 10, strides: 2	ReLU
Batchnorm	-	-
Dropout	dropout rate: 0.3	-
Conv1D	filters: 8, kernel-size: 700, strides: 10	ReLU
Batchnorm	-	-
Max-pool	pool-size: 4, strides: 2	-
Flatten	-	-
Dense	neurons: 512	tanh
Dropout	dropout rate: 0.2	-

Table 5: Architecture and layer parameters for byte 1 classification

In order to visualise the performances of the neural network recognition, $256x256$ pixels images have been created which represent the distribution of the guesses of the neural network within the categories. The image created is shown in Figure 39. The horizontal axis represents the guessed class and the vertical one the original class. The color of the pixel depends on the frequency of the

guess of the neural network on the testing dataset. The best possible outcome is a diagonal going from $(0, 0)$ to $(255, 255)$.

Figure 39 clearly shows some learning of the neural network as a lot of guesses are made on the diagonal. The single trace accuracy is 7.06%. The results will be discussed in Section 4.2.3.

Figure 40 extracts from the previous Figure the maximum of each column. This maximum represents the best guess of the neural network across the 100 traces available per class.

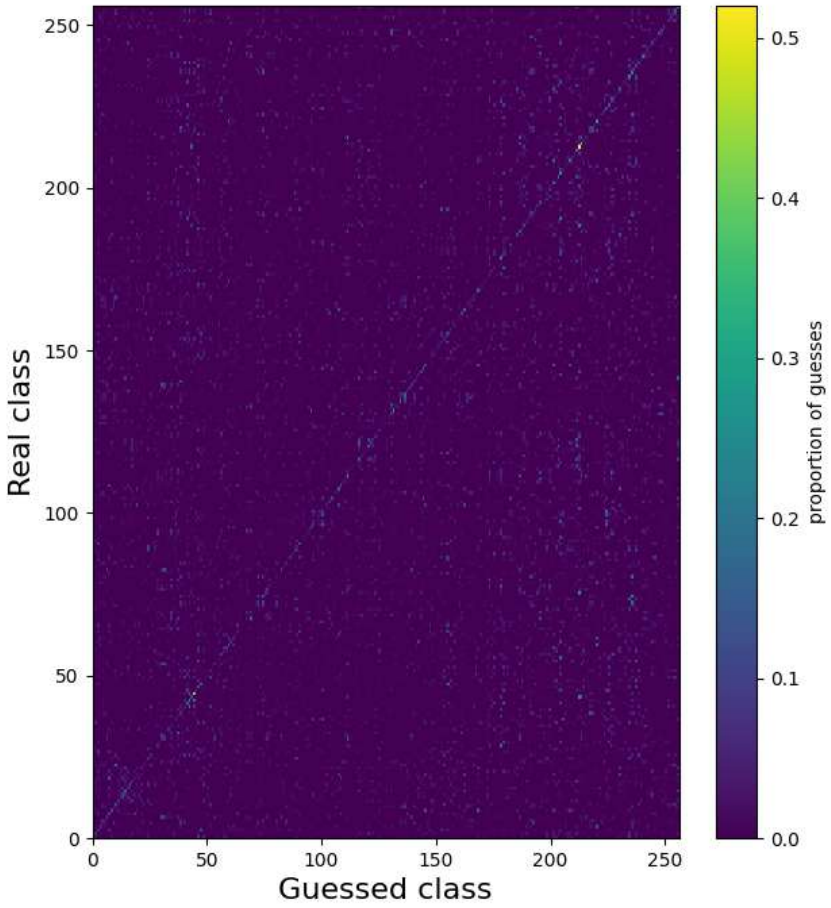


Figure 39: Colormap of the proportion of guesses for each class given the real class for first byte

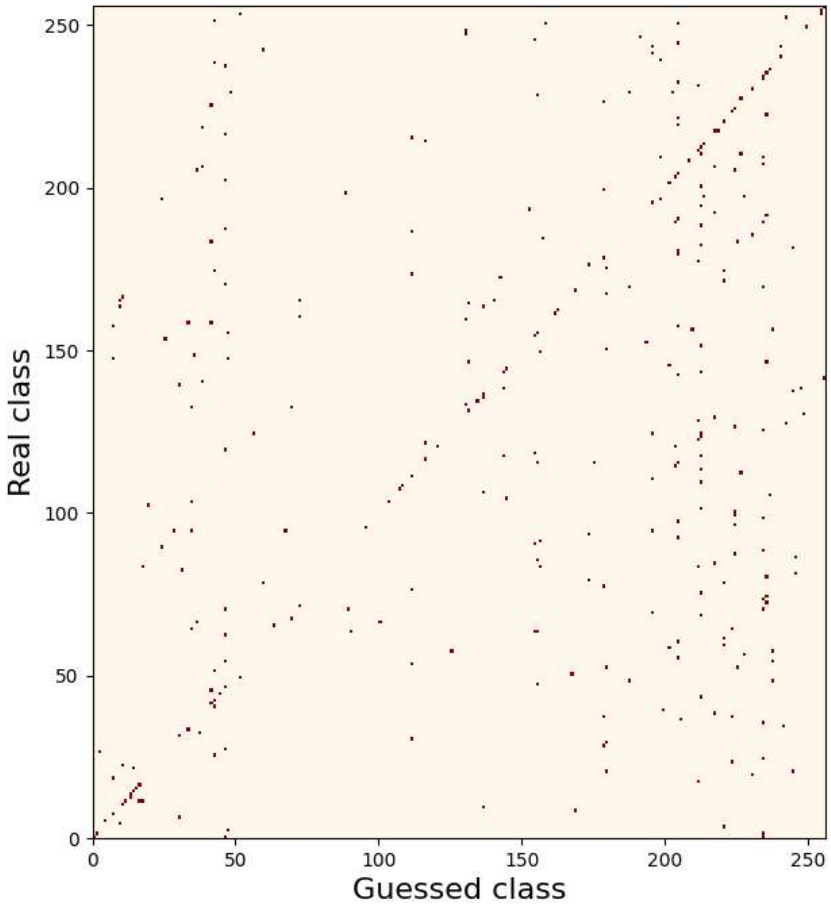


Figure 40: Position of the maximum per column for the first byte

4.2.2 Second byte

The same kind of experiment has been conducted for the second byte of the weights. With the same hyperparameters as in the first test, training has given the following results.

First, Table 6 shows higher t-values meaning that the average of each class must easier to recognise than for the first byte. The distribution of the values is not significantly different than for the first byte.

	min	max	average
t-value	8.30	22.41	15,11

Table 6: T-value for the second byte of the weights

The same colormap and max value representation for the second byte is available on Figure 41 and 42.

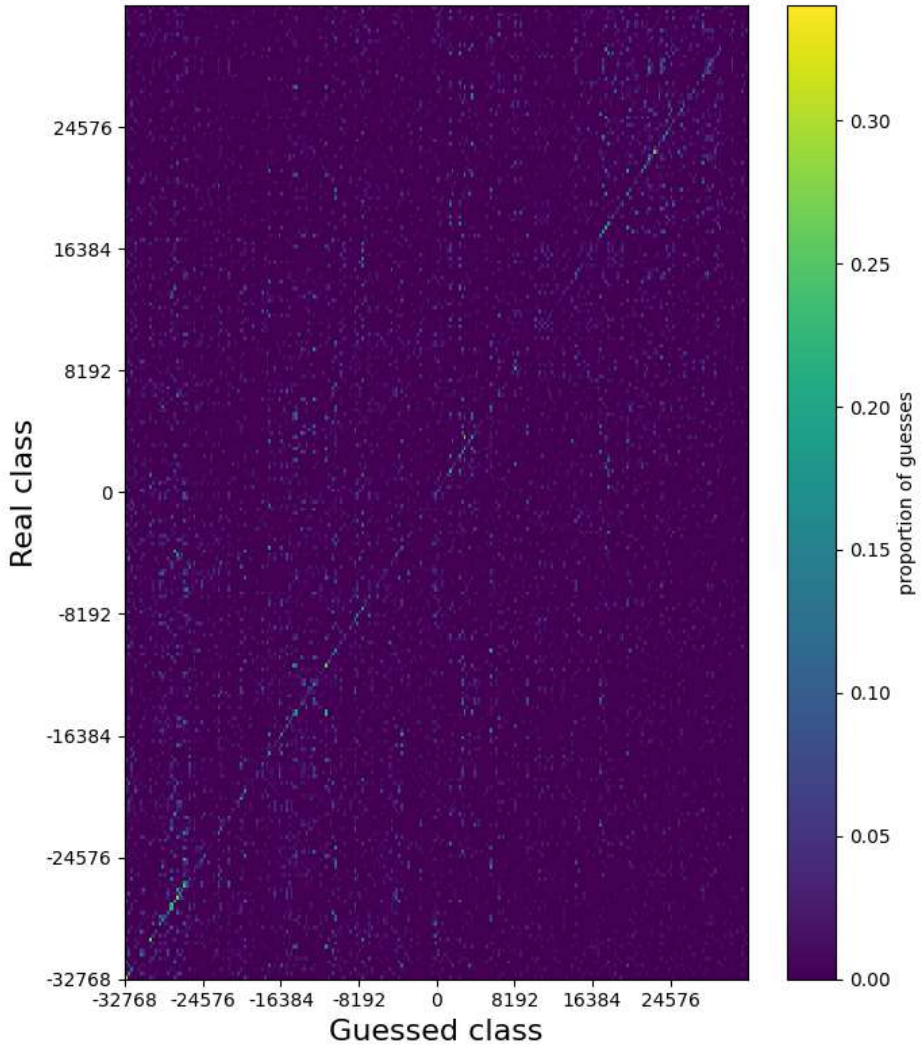


Figure 41: Colormap of the proportion of guesses for each class given the real class for the second byte

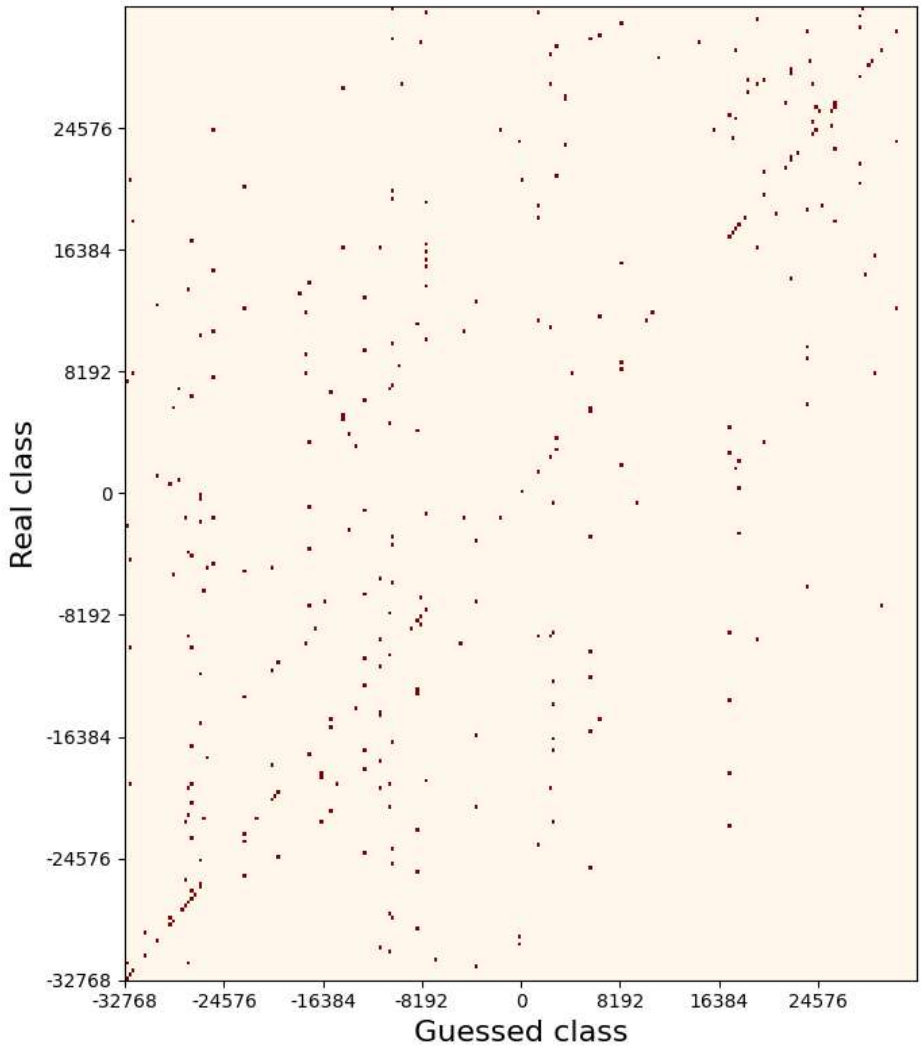


Figure 42: Position of the maximum per column for the second byte

4.2.3 Weight recovery accuracy

Table 7 summarize the accuracy of the neural network for single trace and 100 traces. While clearly being above random guess, the neural network is not able to consistently recover full bytes of the weights from a single trace. With a hundred traces, the accuracy reaches 20% and it is safe to say that adding more traces for both training and testing can improve the accuracy of the recovery.

	single trace	100 traces	random guess
first byte	7.06%	20.31%	0.39%
second byte	6.26%	20.31%	0.39%

Table 7: Accuracy of the weight recognition for single and 100 traces for full byte leakage

Both colormaps in Figure 39 and 41 show higher frequency of guess on the diagonal. When the neural network is wrong, the guesses are well distributed among the other classes. This allows the use of multiple traces in order to eliminate some of the uncertainty. This is why the maximum of each column shwon in Figure 40 and 42 are more likely to be on the diagonal.

4.2.4 Single bit leakage

The first two experiments try to recover a whole byte of the weights leaving 256 classes to chose from. An other possibility to attack the weights is to attack it bit by bit. With the same traces but by changing the labels to match the different bits of the weights, it is possible to train networks to recognise individual bits. The accuracy for individual bits are available in Table 8. This training has been done on a laptop with a cpu in only 1 hour per model.

In this experiment, the random guess is 50% because there are only two classes.

Because of time constraints, the training has been limited to 20 epochs which was not enough to fully train the networks, complete results may show better results.

	single trace	100 traces		single trace	100 traces
sign bit	78.99%	85.12%	7 bit	58.07%	71.86%
14 bit	60.90%	67.52%	6 bit	59.90%	66.41%
13 bit	51.65%	53.13%	5 bit	49.96%	49.96%
12 bit	52.24%	53.91%	4 bit	51.88%	53.91%
11 bit	52.90%	58.20%	3 bit	55.10%	58.20%
10 bit	49.98%	50.00%	2 bit	50.03%	50.03%
9 bit	49.96%	50.02%	1 bit	49.98%	50.00%
8 bit	50.03%	51.17%	0 bit	51.42%	52.34%

Table 8: Accuracy of single bit leakage for single and 100 traces

Single trace values above 51% have been colored green as this thresholds represents the rejection of the null hypothesis with a p-value of 0.5%. It can be noted that it is mostly the most significant bits which seem to be leaking the most.

The disparity of results may come from different parameters :

- **Leakage** : The different bits may leak differently because of the way multiplications are done in the DPU unit.
- **Incomplete training** : Constraints of time limited training to 20 epochs which may not be enough for the accuracy of some networks to rise.
- **Single recognition network** : The network architecture used was the same for all the bits. Although the leakage between bits must be somewhat similar, custom hyperparameters tuning for each bit may enhance accuracy.

5 Discussion

5.1 Limits of the attacks

Even though this thesis has shown FPGA implementation of NN vulnerable to information leakage, these attacks are limited. The weight recovery requires a huge amount of data and post-processing to recover a single weight. This leads to very little efficiency of the attack on large networks.

The attack on hyperparameters of the convolution layer has proven to be efficient and robust but still needs a lot of data to set up the profiling attack. Hyperparameters are usually kept in well-known range which makes it easier for the attacker to limit the search space but this limitation requires a lot of expertise on the problem.

As for every profiling attacks, the accuracy of the models will get worse if the traces captured for profiling and for inferring come from different boards.

5.2 Countermeasures

All the countermeasures presented here are well-known in the side-channel field and have proven to be efficient but expensive to implement.

Because the attacks rely on the identification of patterns, the random shuffling of the operations can be effective at hiding individual weights. This technique [Vey+12] consists in doing the matrix multiplication operation in a random order. Although this should be effective for single weight recovery, it is ineffective at hiding computing times which gives out hyperparameters of the layers. This solution comes with the drawback of having to implement a random number generator on the FPGA or having a dedicated component attached to it, adding again more surface and energy consumption to the overall design.

Adding software or hardware measures to make the computation constant time is a possible solution to mitigate architecture reverse engineering but it costs a lot of performances which was the primary reason for switching to FPGA platforms.

5.3 Integration of the chip

Because the studied FPGA was integrated on the ZCU104 development board, it can be assumed that its integration is well made and that custom boards made by customers would be more likely to emit EM signals. This integration could lead to greater leakage of sensible information.

5.4 Future research

This section will cover what is ahead in the domain of SCA on FPGA and their application to machine learning implementations.

Because this work is based on convolution layers, further research must be conducted on other types of layers. Fully connected layers are the most interesting target after convolution layers because of its major use in simple neural network applications. Even though the layer type is different, the leakage is expected to be similar because the DPU IP is not protected at all against side-channel attacks.

Other accelerators can be implemented on the FPGA chip from Xilinx. Research could compare the leakage of these other accelerators to this one. The comparison should take into account the number of logic gates used on the FPGA, the computational advantage given by the architecture and the leakage.

Traces have been captured without any changes on the board but some alterations might be useful to enhance data leakage. For example, removing some of the decoupling capacitors on the board might make the amplitude of the signal bigger. An other modification can be removing the metal case of the chip in order to put the probe directly above the silicon. This method is called delidding and allows the use of a different kind of setup which can position the probe more precisely to focus different part of the FPGA.

The last experiment recovering weights with machine learning did not have enough time to be properly finished. To continue this experiment, one would need to fully train all the network to asses their final accuracy. It would also be good to train a network to recognise the hamming weight of the weights as this information may be easier to recover.

Conclusion

The implementations of Neural Networks have proven to be vulnerable to theft attack even in a black-box methodology. Focusing on convolution layers, this work has proven that hyperparameters leak through the use of the Deep Processing Unit accelerator intellectual property which was implemented on a ZCU104 board from Xilinx. The shapes of the traces leak significant amount of information and can easily be captured with simple electromagnetic probes without any modification of the board.

The first experiment aimed at extracting hyperparameters of convolution layers with machine learning on electromagnetic traces. With three out of four hyperparameter being recognised with an accuracy higher than 68% on validation set, a machine learning approach has proven to be efficient.

Thanks to the Welch's t-test, the second experiment shows that single weights are vulnerable to statistical attack. In a fixed input scenario, it takes only 2000 recordings to distinguish a single weight out of a 400 weights convolution layer. The attack implementing machine learning against single weights did not show enough results to recover weights fully but paved the way for further research and attacks.

References

[CG00] Jean-Sébastien Coron and Louis Goubin. "On Boolean and Arithmetic Masking against Differential Power Analysis". In: *Cryptographic Hardware and Embedded Systems — CHES 2000*. Ed. by Çetin K. Koç and Christof Paar. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 231–237. ISBN: 978-3-540-44499-2.

[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.

[OST06] Dag Arne Osvik, Adi Shamir, and Eran Tromer. "Cache Attacks and Countermeasures: The Case of AES". In: *Topics in Cryptology – CT-RSA 2006*. Ed. by David Pointcheval. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 1–20. ISBN: 978-3-540-32648-9.

[Vey+12] Nicolas Veyrat-Charvillon et al. "Shuffling against Side-Channel Attacks: A Comprehensive Study with Cautionary Note". In: *Advances in Cryptology – ASIACRYPT 2012*. Ed. by Xiaoyun Wang and Kazue Sako. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 740–757. ISBN: 978-3-642-34961-4.

[LBM14] Liran Lerman, Gianluca Bontempi, and Olivier Markowitch. "Power analysis attack: an approach based on machine learning". In: *International Journal of Applied Cryptography* 3.2 (2014), pp. 97–115.

[GHO15] Richard Gilmore, Neil Hanley, and Maire O'Neill. "Neural network based attack on a masked implementation of AES". In: *2015 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE. 2015, pp. 106–111.

[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.

[WO19] Carolyn Whitnall and Elisabeth Oswald. *A Cautionary Note Regarding the Usage of Leakage Detection Tests in Security Evaluation*. Cryptology ePrint Archive, Paper 2019/703. https://eprint.iacr.org/2019/703. 2019. URL: https://eprint.iacr.org/2019/703.

[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.

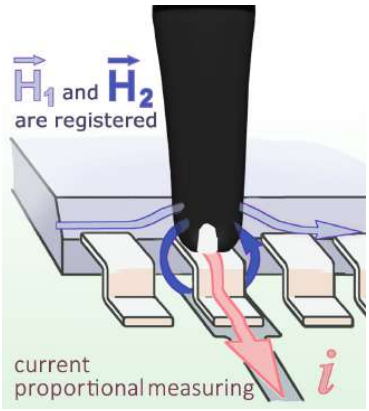
[Yu+20b] H. Yu et al. "DeepEM: Deep Neural Networks Model Recovery through EM Side-Channel Information Leakage". In: *IEEE International Symposium on Hardware Oriented Security and Trust (HOST)* (2020). https://doi.org/10.1109/HOST45689.2020.9300274.

Appendices

EM Probe Documentation

RF-U 2.5-2

H-Field Probe 30 MHz up to 3 GHz



Short description

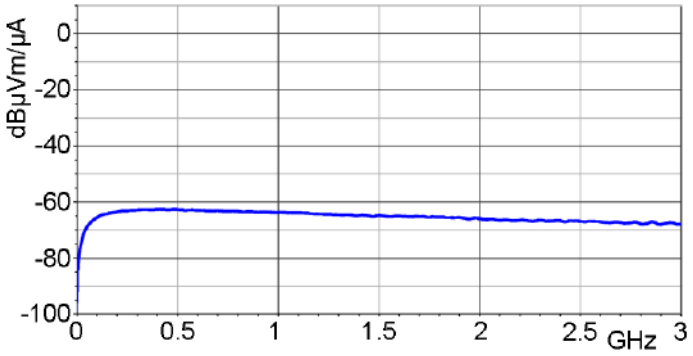
The RF-U 2.5-2 near-field probe is designed for the selective measurements of RF currents in conducting paths, component connectors, SMD components, and IC-pins. The probe head has a magnetically active gap with an approx. width of 0.5 mm. To use, the head should be positioned directly onto the measured object.

The RF-U 2.5-2 is a passive near-field probe that functions like the RF-U 5-2 probe, but is designed for SMD components (pins). The near-field probe is small and handy. It has a current attenuating steath and its upper side is electrically shielded. It can be connected to a spectrum analyzer or an oscilloscope with a 50 Ω input. The H-field probe does not have an internal terminating resistance of 50 Ω.

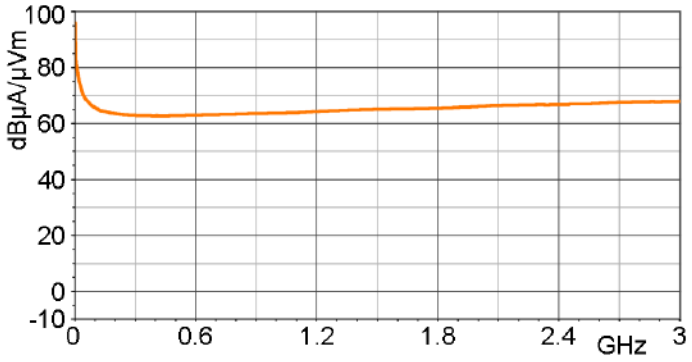
Technical parameters

Frequency range	30 MHz ... 3 GHz
Resolution	≈ 0.5 mm
Probe head dimensions	Ø ≈ 4 mm
Connector - output	SMB, male, jack

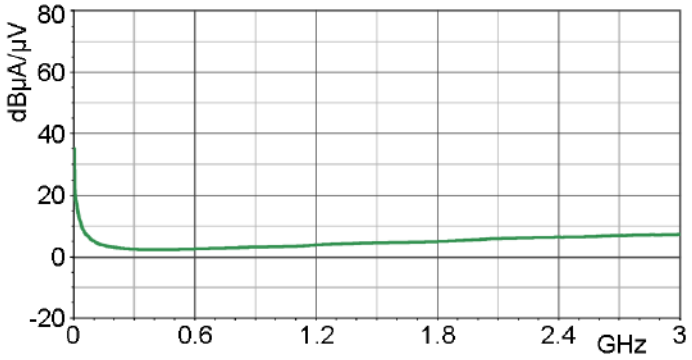
Frequency response [dBµV] / [dBµA/m]



H-field correction curve [dBµA/m] / [dBµV]



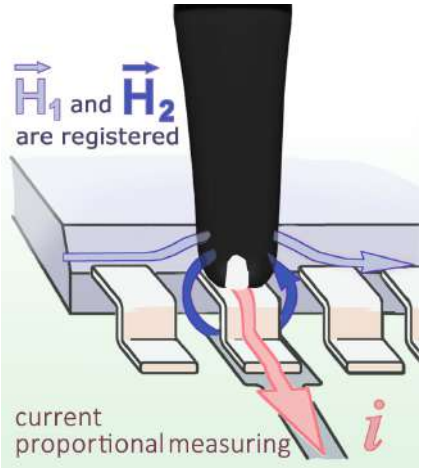
Current correction curve [dBµA] / [dBµV]



RF-U 2.5-2

H-Field Probe 30 MHz up to 3 GHz

Measuring principles



Probe head