Convolutional Neural Networks with Data Augmentation against Jitter-Based Countermeasures

Profiling Attacks without Pre-Processing

Eleonora Cagli[1,2,4], Cécile Dumas[1,2], Loïc Masure[1,2,4], and Emmanuel Prouff[3,4]

[1] Univ. Grenoble Alpes, F-38000, Grenoble, France
[2] CEA, LETI, MINATEC Campus, F-38054 Grenoble, France
{eleonora.cagli,cecile.dumas,loic.masure}@cea.fr
[3] ANSSI, France emmanuel.prouff@ssi.gouv.fr
[4] Sorbonne Universitès, UPMC Univ Paris 06, POLSYS, UMR 7606, LIP6, F-75005, Paris, France

Abstract. In the context of the security evaluation of cryptographic implementations, *profiling attacks* (aka *Template Attacks*) play a fundamental role, and are often implemented under Gaussian assumption for the leakage distribution. Nevertheless this approach suffers from the difficulty to validate the software and hardware countermeasures that have been conceived to create traces misalignment. This forces the evaluator to perform critical preprocessing phases, such as the selection of the points of interest and the realignment of measurements. We propose an end-to-end strategy based on the Convolutional Neural Networks (CNN): it greatly facilitates the evaluation roadmap, since it does not require such preprocessings. To significantly increase the performances of the CNN, we moreover propose to equip it with the data augmentation technique that is classical in other applications of Machine Learning.

Keywords: Side-Channel Attacks, Convolutional Neural Networks, Data Augmentation, Machine Learning, Jitter, Trace Misalignment, Unstable Clock

1 Introduction

To prevent Side-Channel Attacks (SCA), manufacturers commonly implement countermeasures that create misalignment in the measurements sets. The latter countermeasures are either implemented in hardware (unstable clock, random hardware interruption, clock stealing) or in software (insertion of random delays through dummy operations [8, 9], shuffling [31]). Until now two approaches have been developed to perform security evaluations in presence of enhanced misalignment. The first one simply consists in adapting the number of side-channel acquisitions (usually increasing it by a factor which is linear in the misalignment effect). Eventually an integration over a range of points [20] can be made, which guarantees the extraction of the information over a single point, at the cost of a linear increase of the noise, that may be compensated by the increase of the acquisitions. The second one, which is usually preferred, consists in applying realignment techniques in order to limit the desynchronization effects. Two realignment techniques families might be distinguished: a signal-processing-oriented one (*e.g.* [24, 30]), more adapted to hardware countermeasures, and a probabilistic-oriented one (*e.g.* [11]), conceived for the detection of dummy operations, *i.e.* software countermeasures.

Among the SCAs, *profiling attacks* (aka Template Attacks, TA for short) play a fundamental role in the context of the security evaluation of cryptographic implementations. Indeed the profiling attack scenario allows to evaluate their worst-case security, admitting the attacker is able to characterize the device leakages by means of a full-knowledge access to a device identical to the one under attack. Such attacks work in two phases: first, a leakage model is estimated during a so-called *profiling phase*, then the profiled leakage model is exploited to extract key-dependent information in the proper *attack phase*. Approximating the information leakage by a Gaussian distribution is today the most popular approach for the profiling. Nevertheless the performances of such a classical TA highly depend on some preliminary phases, such as the traces realignment or the selection of the Points of Interest (PoIs).

We propose the use of a Convolutional Neural Network (CNN) as a comprehensive profiling attack strategy. Such a strategy, divided into a learning phase and a proper attack phase, can replace the entire roadmap of the state of the art attacks: for instance, contrary to a classical TA, any trace

preprocessing such as realignment or the choice of the PoIs are included in the learning phase. Indeed we will show that the CNNs implicitly perform an appropriate combination of the time samples and are robust to trace misalignment. This property makes the profiling attacks with CNNs efficient and easy to perform, since they do not require critical preprocessings. Moreover, since the CNNs are less impacted than the classical TA by the dimension of the traces, we can *a priori* expect that their efficiency outperforms (or at least perform as well as) the classical TAs. Indeed, CNNs can extract information from a large range of points while, in practice, Gaussian TAs are used to extract information on some previously dimensionality-reduced data (and dimensionality reduction never raises the informativeness of the data [10]).

To compensate some undesired behaviour of the CNNs, we propose as well to embed them with *Data Augmentation* (DA) techniques [28, 32], recommended in the machine learning domain for improving performances: the latter technique consists in artificially generating traces in order to increase the size of the profiling set. To do so, the acquired traces are distorted through plausible transformations that preserve the label information (*i.e.* the value of the handled sensitive variable in our context). Actually, we propose to turn the misalignment problem into a virtue, enlarging the profiling trace set *via* a random shift of the acquired traces and another typology of distortion that together simulate a clock jitter effect. Paradoxically, instead of trying to realign the traces, we propose to further misalign them (a much easier task!), and we show that such a practice provides a great benefit to the CNN attack strategy.

We propose as well the use of a visualization tool, allowing interpreting the succeeding results obtained by a CNN architecture. In particular, the visualization tool can be used to highlight the PoIs of each single trace belonging to a set of misaligned traces. We remark that with classical PoIs detecting statistics, *e.g.* the SNR or the T-Test, that are estimated in a point-wise manner over an entire dataset, the trace-wise visualization of PoIs is not possible, and in case of misalignment such statistics are corrupted by the misalignment noise. On one hand, visualizing the PoIs automatically extracted by the trained CNN architecture serves, in this paper, to validate the CNN claim of robustness to misalignment. On the other hand they are extremely useful in the optic of strengthening the embedded security, since they allow evaluators to localize the leaking parts of the cryptographic implementation, and to transmit useful warnings to the developers.

This contribution makes part of the transfer of methodology in progress in last years from the machine learning and pattern recognition domain to the side-channel analysis one. In last years, the strict analogy between TAs and the supervised classification problem as been depicted [14], while the deployment of Neural Networks (NNs) [22, 21, 33] and CNNs [19] to perform profiled SCAs has been proposed.

This communication focuses over the effectivness of CNNs in presence of misalignment, thus considerations about other kinds of countermeasures, such as masking, are left apart. Nevertheless, the fact that CNNs usually applies non-linear functions to the data makes them potentially (and practically, as already experienced in [19, 26]) able to deal with such a countermeasure as well.

The paper is organized as follows: in Sec. 2 we recall the classical TA roadmap, we introduce the CNN family of NNs, we describe the NN-based SCA, we discuss the practical aspects of the training phase of an NN, we give the description of the Data Augmentation and the visualization method. In Sec. 3 we present the results of the visualization technique on the ASCAD database [26], where software countermeasures' effects are simulated. Experiments against hardware countermeasures are described in Sec. 4, proving that the CNN are robust to deformations due to the jitter.

2 Preliminaries

2.1 Notations

Throughout this paper we use calligraphic letters as $\mathcal{X}$ to denote sets, the corresponding upper-case letter X to denote random variables (random vectors if in bold $\mathbf{X}$) over $\mathcal{X}$, and the corresponding lower-case letter x (resp. $\mathbf{x}$ for vectors) to denote realizations of X (resp. $\mathbf{X}$). The i-th entry of a vector $\mathbf{x}$ is denoted by $\mathbf{x}[i]$. Side-channel traces will be viewed as realizations of a random column vector $\mathbf{X} \in \mathbb{R}^D$. During their acquisition, a target sensitive variable $Z = f(P, K)$ is handled, where P denotes some public variable, *e.g.* a plaintext, and K the part of secret key the attacker aims to retrieve. The value assumed by such a variable is viewed as a realization $z \in \mathcal{Z} = \{z^1, z^2, \ldots, z^{|\mathcal{Z}|}\}$ of a discrete finite

random variable Z. We will sometimes represent the values $z^i \in \mathcal{Z}$ via the so-called *one-hot encoding* representation, assigning to z^i a $|\mathcal{Z}|$-dimensional vector, with all entries equal to 0 and the i-th entry equal to 1: $z^i \rightarrow \mathbf{z}^i = (0, 0, \ldots, 0, \underbrace{1}_{i}, 0, \ldots, 0)$.

2.2 Profiling Side-Channel Attack

A profiling SCA is performed into two phases: a profiling (or characterization, or training) phase, and an attack (or matching) phase. During the profiling phase, the attacker exploits a *profiling set* $\{\mathbf{x}_i, z_i\}_{i=1,\ldots,N_p}$ of size N_p, which is a set of traces $\mathbf{x}_i$ acquired under known value z_i of the target, to estimate the probability:

$$\Pr[\mathbf{X}|Z = z]. \tag{1}$$

The potentially huge dimensionality of $\mathbf{X}$ lets such an estimation a very complex problem, and the most popular way adopted until now to estimate the conditional probability is the one that led to the well-established Gaussian TA [5]. To perform the latter attack, the adversary priorly exploits some statistical tests (*e.g.* SNR, T-Test [6]) and/or dimensionality reduction techniques (*e.g.* PCA, LDA, KDA [4]) to select a small portion of PoIs or an opportune combination of them. Then, denoting $\varepsilon(\mathbf{X})$ the result of such a dimensionality reduction, the attacker assumes that $\varepsilon(\mathbf{X})|Z$ has a multivariate Gaussian distribution, and estimates the mean vector $\boldsymbol{\mu}_z$ and the covariance matrix $\boldsymbol{\Sigma}_z$ for each $z \in \mathcal{Z}$ (*i.e.* the so-called templates). In this way the pdf (1) is approximated by the Gaussian pdf with parameters $\boldsymbol{\mu}_z$ and $\boldsymbol{\Sigma}_z$. The attack phase eventually consists in comparing the attack traces $\{\mathbf{x}_i\}_{i=1,\ldots,N}$ to the templates. The attacker sorts the key candidates $k \in \mathcal{K}$ with respect to their a-posteriori probability given by:

$$d_k = \prod_{i=1}^{N} \Pr[Z = f(p_i, k)|\mathbf{X} = \mathbf{x}_i] = \prod_{i=1}^{N} \frac{\Pr[\mathbf{X} = \mathbf{x}_i|Z = f(p_i, k)]}{\Pr[Z = f(p_i, k))]}, \tag{2}$$

where the last equation is obtained via the Bayes theorem under the hypothesis that acquisitions are independent, and omitting the prior probability of the variable $\mathbf{X}$ which is constant for each key hypothesis.

Traces misalignment affects the approach above. In particular, if not treated with a proper realignment, it makes the PoI selection harder, obliging the attacker to consider a wide range of points for his characterization (the more effective the misalignment, the wider the range), directly or after a previous integration, as proposed in [20]. Instead of an integration of points, other techniques that follow a dimension reduction approach might be applied, *e.g.* the PCA or the LDA. Unfortunately, the latter techniques suffer from the misalignment effect as well. As we will see in the next section, the neural networks, and in particular the CNNs, are able to efficiently address the PoI selection problem and the misalignment issue simultaneously. More precisely, they can be trained to search for informative weighted combinations of leakage points, in a way that is robust to traces misalignment.

2.3 Neural Networks

The classification problem is the most widely studied one in machine learning, since it is the central building block for all other problems, e.g detection, regression, etc. [2] It consists in taking an input, *e.g.* a side-channel trace $\mathbf{x}$, and in assigning it a label $z \in \mathcal{Z}$, *e.g.* the value of the target variable handled during the acquisition. In the typical setting of a supervised classification problem, a *training set* is available, which is a set of data already assigned to the right label. The latter set exactly corresponds to the profiling set in the side-channel context.

Neural networks (NN) are nowadays the privileged tool to address the classification problem. They aim at constructing a function $F\colon \mathbb{R}^D \rightarrow \mathbb{R}^{|\mathcal{Z}|}$ that takes data $\mathbf{x}$ and outputs vectors $\mathbf{y}$ of scores. The classification of $\mathbf{x}$ is done afterwards by choosing the label z^i such that $i = \text{argmax } \mathbf{y}[i]$. In general F is obtained by combining several simpler functions, called *layers*. An NN has an *input layer* (the identity over the input datum $\mathbf{x}$), an output layer (the last function, whose output is the scores vector $\mathbf{y}$) and all other layers are called *hidden* layers. The nature (the number and the dimension) of the layers is called the *architecture* of the NN. All the parameters that define an architecture, together

with some other parameters that govern the training phase, have to be carefully set by the attacker, and are called *hyper-parameters*. The so-called *neurons*, that give the name to the NNs, are the computational units of the network and essentially process a scalar product between the coordinate of its input and a vector of *trainable weights* (or simply *weights*) that have to be *trained*. Each layer processes some neurons and the outputs of the neuron evaluations will form new input vectors for the subsequent layer. The training phase consists in an automatic tuning of the weights and it is done *via* an iterative approach which locally applies the Stochastic Gradient Descent algorithm [12] to minimize a *loss function* quantifying the classification error of the function F over the training set. We will not give further details about this classical optimization approach, called *Backward Propagation*, and the interested reader may refer to [12].

2.4 Convolutional Neural Networks: Core Principles and Constructions

In this section we describe the layers that compose a Convolutional Neural Network (CNN), and we explain why the CNNs is able to be robust to misalignment.

- *The Fully-Connected layers* (FC for short), denoted by λ, are linear layers expressible as follows: denoting $\mathbf{x} \in \mathbb{R}^D$ the input of an FC, its output is given by $A\mathbf{x} + \mathbf{b}$, being $A \in \mathbb{R}^{D \times C}$ a matrix of weights and $\mathbf{b} \in \mathbb{R}^C$ a vector of biases. These weights and biases are the trainable weights of the FC layer.[5]
- *Convolutionnal layers* (CONV for short) are linear layers, denoted by γ, that share weights across space. To apply a CONV to an input of size $D \times V$, where the initial depth V is one, for 1D-data, n_{filter} small matrices, called *convolutional filters*, of size $W \times V$ (where W is called *kernel size*) are slid over the length dimension of the input by a chosen amount of units, called *stride*. The filters form a window, called *patch* in ML language, which defines a linear transformation of $W \times V$ consecutive points of the data into new matrices of size $1 \times n_{\text{filter}}$, arranged in such a way that n_{filter} is the depth of the layer output. The length dimension of the output of a convolutional layer depends on several parameters: the input length, the stride, and the *padding*. The two most common ways to pad the input are called *same padding* and *valid padding*: with the *same padding* the input is padded with some zeros at the beginning and at the end, in such a way that, for a stride equal to 1, the output has the same length than the input, for a stride equal to 2 the input length is exactly halved, for a stride equal to 3 it is exactly divided by 3, etc. The *valid padding* consists on the contrary to avoid any kind of padding. Only proper data points are used as input, and output length is adjusted: typically, for a stride equal to 1, the output length equals $D - W + 1$, where D is the input length. The coordinates of the window are among the trainable weights of the model. They slid over the input, so they are multiplied by different parts of the datum, but they are constrained to keep unchanged while sliding, *i.e.* to behave in the same way no matter the position of the input entries on the global input datum. This constraint aims to allow the CONV layer to learn shift-invariant features, *i.e.* characteristics of the datum for which the position is not discriminant. Shift-invariant features are largely present in image recognition context, which drove the development of CNNs. For examples the eyes, the nose and the mouth of a person in a picture, are discriminant features for the person no matter their position in the image. The ability at learning shift-invariant features makes CNNs robust to geometrical deformations [17] or to temporal deformation when considering side-channel signals. For this reason they are adequate to counteract misalignment-based countermeasures.
- *Pooling layers* (POOL for short) are non-linear layers, denoted by δ, that reduce the spatial size by making some filters slide across its input. The filter is 1-dimensional, characterised by a length W, and usually the stride is chosen equal to its length. In contrast with convolutional layers, the pooling filter does not contain trainable weights; they only slid across the input to select a segment, then a pooling function is applied: the most common pooling functions are the *max-pooling*, which outputs the maximum values within the segment, and the *average-pooling*, which outputs the average of the coordinates of the segment.

[5] They are called *Fully-Connected* because each i-th input coordinate is *connected* to each j-th output via the $A[i, j]$ weight. FC layers can be seen as a special case of the linear layers in general *Feed-Forward* networks, in which not all the connections are present. The absence of some (i, j)-th connections can be formalized as a constraint for the matrix A consisting in forcing to 0 its (i, j)-th coordinates.

– *Activation layers* (ACT for short) are composed of a single non-linear real function that is applied independently to each coordinate of its input. Several activation functions have been used in Deep Learning and currently the so-called ReLU is preferred. It processes $\max(0, x)$ to each coordinate x. The ACT layer preceded by the Batch Normalization layer below is denoted α.
– *Batch Normalization layers* (BN for short) have been introduced in [15] by Ioffe Szegedy to reduce the so-called *internal covariate shift* in neural networks and to eventually fasten the gradient descent.
– The *softmax*[6] layer (SOFT for short), denoted by s, applies the following processing to each coordinate of its input $\mathbf{x}$: $\mathsf{s}(\mathbf{x})[i] = \frac{e^{\mathbf{x}[i]}}{\sum_j e^{\mathbf{x}[j]}}$.

Eventually, the main block of a CNN is a CONV layer γ directly followed by a BN layer and an ACT layer α. The former locally extracts information from the input thanks to filters and the latter increases the complexity of the learned classification function thanks to its non-linearity. After some $(\alpha \circ \gamma)$ blocks, a POOL layer δ is usually added to reduce the number of neurons: $\delta \circ [\alpha \circ \gamma]^{n_{\text{conv}}}$. This new block is repeated n_{blocks} times in the neural network until obtaining an output of reasonable size. Then, n_{dense} FC layes λ are introduced in order to obtain a global result which depends on the entire input. To sum-up, a common convolutional network can be characterized by the following formula:[7]

$$F(\mathbf{x}) = \mathsf{s} \circ [\lambda]^{n_{\text{dense}}} \circ [\delta \circ [\alpha \circ \gamma]^{n_{\text{conv}}}]^{n_{\text{blocks}}}(\mathbf{x}) = \mathbf{y} \ . \tag{3}$$

The role of the *softmax* is to renormalize the output scores in such a way that they approximate a probability distribution $\mathbf{y} = \Pr[Z|\mathbf{X} = \mathbf{x}]$. In this way, the computed output does not only provide the most likely label to solve the classification problem, but also the a-posteriori probability of the remaining $|\mathcal{Z}| - 1$ other labels. In the profiling SCA context, this form of output allows us to enter it in (2) to rank key candidates, since (3) may be viewed as an approximation of the pdf in (1). We can thus rewrite (2) as:

$$d_k = \prod_{i=1}^{N} F(\mathbf{x}_i)[f(p_i, k)]. \tag{4}$$

We refer to [18] for an (excellent) explication over the relationship between the softmax function and the Bayes theorem.

2.5 Practical Aspects of the Training Phase and Overfitting

The goal of the training phase is to tune the weights of the NN. The latter ones are first initialized with random values and are afterwards updated by applying several times the same process: a batch of traces chosen in random order goes through the network to obtain its score, the loss function is computed from this score and finally the loss is reduced by modifying the trainable parameters. An iteration over the entire training set is called *epoch*. To monitor the training of an NN and to evaluate its performances it is a good practice to separate the labelled data into 3 sets:

– the proper *training set*, which is actually used to train the weights (in general it contains the greater part of the labelled data)
– a *validation set*, which is observed in general at the end of each epoch to monitor the training
– a *test set*, which is left unobserved during the training phase and involved to finally evaluate the performances of the trained NN. For our experiments we will use the attack traces as test set.

The accuracy is the most common metric to both monitor and evaluate an NN classifier. It is defined as the successful classification rate reached over a dataset. The *training accuracy*, the *validation accuracy* and the *test accuracy* are the successful classification rates achieved respectively over the training, the validation and the test sets. At the end of each epoch it is useful to compute and to compare the training accuracy and the validation accuracy. For some trained models we will measure in this paper (see *e.g.* Table 2) the following two quantities:

[6] To prevent underflow, the log-softmax is usually preferred if several classification outputs must be combined.
[7] where each layer of the same type appearing in the composition is not to be intended as exactly the same function (e.g. with same input/output dimensions), but as a function of the same form.

– the *maximal training accuracy*, corresponding to the maximum of the training accuracies computed at the end of each epoch
– the *maximal validation accuracy*, corresponding to the maximum of the validation accuracies computed at the end of each epoch.

In addition to the two quantities above, we will also evaluate the performances of our trained model, by computing a *test accuracy* (see *e.g.* Table 1).

On the Need to also Consider the Guessing Entropy. The accuracy metric is perfectly adapted to the machine learning classification problem, but corresponds in side-channel language to the success rate of a Simple Attack, *i.e.* an attack where a single attack trace is available. When the attacker can acquire several traces for varying plaintexts, the accuracy metric is not sufficient alone to evaluate the attack performance. Indeed such a metric only takes into account the label corresponding to the maximal score and does not consider the other ones, whereas an SCA through (4) does (and therefore exploits the full information). To take this remark into account, we will always associate the test accuracy to a side-channel metric defined as the minimal number $N^\star$ of side-channel traces that makes the *guessing entropy* (the average rank of the right key candidate) be permanently equal to 1 (see *e.g.* Table 1). We will estimate such a guessing entropy through 10 independent attacks.

As we will see in the sections dedicated to our attack experiments, applying Machine Learning in a context where at the same time (1) the model to recover is complex and (2) the amount of exploitable measurements for the training is limited, may be ineffective due to some overfitting phenomena.

Overfitting. Often the training accuracy is higher than the validation one. When the gap between the two accuracies is excessive, we assist to the *overfitting* phenomenon. It means that the NN is using its weights to *learn by heart* the training set instead of detecting significant discriminative features. For this reason its performances are poor over the validation set, which is new to it. Overfitting occurs when an NN is excessively complex, *i.e.* when it is able to express an excessively large family of functions. In order to keep the NN as complex as wished and hence limiting the overfitting, some *regularization* techniques can be applied. For example, in this paper we will propose the use of the *Data Augmentation* (DA) [28] that consists in artificially adding observations to the training set. Moreover, for experiments described in Sec. 4 we will take advantage of the *early-stopping* technique [25] that consists in well choosing a stop condition based on the validation accuracy or on the validation loss (*i.e.* the value taken by the loss function over the validation set).

2.6 Data Augmentation

As pointed out in Sec.2.5, it is sometimes necessary to manage the overfitting phenomenon, by applying some regularization techniques. As we will see in Sec. 4 this will be the case in our experiments: indeed we will propose a quite deep CNN architecture, flexible enough to manage the misalignment problems, but trained over some relatively small training sets. This fact, combined with the high number of weights exploited by our CNN implies that the latter one will *learn by heart* each element of the training set, without catching the truly discriminant features of the traces.

Among all regularization techniques, we choose to concentrate priorly on the Data Augmentation [28], mainly for two reasons. First, it is well known that the presence of misalignment forces to increase the number of acquisitions. In other terms, misalignment may provoke a lack of data phenomenon on the adversary side. In the machine learning domain such a lack is classically addressed thanks to the DA technique, and its benefits are widely proved. For example, many image recognition competition winners made use of such a technique (*e.g.* the winner of ILSVRC-2012 [16]). Second, the DA is controllable, meaning that the deformations applied to the data are chosen, thus fully characterized. It is therefore possible to fully determine the addition of complexity induced to the classification problem. In our opinion, other techniques add constraints to the problem in a more implicit way, *e.g.* the dropout [13] or the ℓ_2-norm regularization [2].

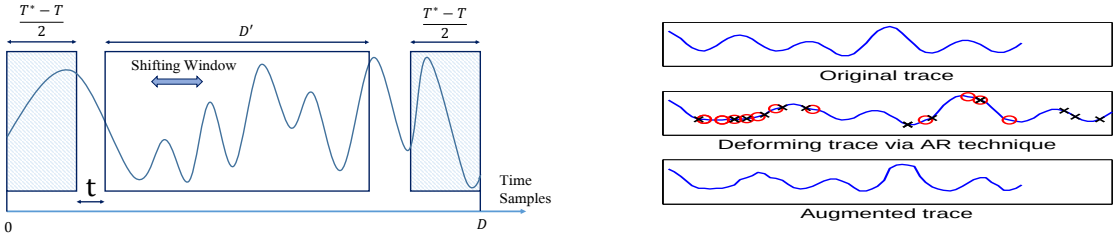

Fig. 1: Left: Shifting technique for DA. Right: Add-Remove technique for DA (added points marked by red circles, removed points marked by black crosses).

Data augmentation consists in artificially generating new training traces by deforming those previously acquired. The deformation is done by the application of transformations that preserve the label information (*i.e.* the value of the handled sensitive variable in our context). We choose two kinds of deformations, that we denote by *Shifting* and *Add-Remove*.

Shifting Deformation ($\mathrm{SH}_{T^\star}$*)* simulates a random delay effect of maximal amplitude $T^\star$, by randomly selecting a shifting window of the acquired trace, as shown in Fig. 1. Let D denote the original size of the traces. We fix the size of the input layer of our CNN to $D' = D - T^\star$. Then the technique $\mathrm{SH}_{T^\star}$ consists (1) in drawing a uniform random $t \in [0, T^\star]$, and (2) in selecting the D'-sized window starting from the t-th point. For our study, we will compare the SH_T technique for different values $T \leq T^\star$, without changing the architecture of the CNN (in particular the input size D'). Notably, $T \lneq T^\star$ implies that $T^\star - T$ time samples will never have the chance to be selected. As we suppose that the information is localized in the central part of the traces, we choose to center the shifting windows, discarding the heads and the tails of the traces (corresponding to the first and the last $\frac{T^\star - T}{2}$ points).

Add-Remove Deformation (AR*)* simulates a clock jitter effect (Fig. 1). We will denote by AR_R the operation that consists (1) in inserting R time samples, whose positions are chosen uniformly at random and whose values are the arithmetic mean between the previous time sample and the following one, (2) in suppressing R time samples, chosen uniformly at random.

The two deformations can be composed: we will denote by $\mathrm{SH}_T\mathrm{AR}_R$ the application of a SH_T followed by a AR_R.

2.7 Visualization method for Deep Learning and SCA

In this section, we present how to use a technique that enables to visualize, for the measured traces, the relevance of each point with respect to the attack accuracy. The aim is to replace characterization techniques such as SNR in order to become more robust against jittering countermeasures. Formally, we want to build a D-sized vector $\mathbf{r}_i$ for each input trace $\mathbf{x}_i$.

A former approach: the Signal-to-Noise Ratio (SNR) The SNR is a powerful statistical tool able to show the informative regions on leakage traces. The formal definition of the SNR at instant t is the following:

$$\mathrm{SNR}[t] \triangleq \frac{\mathrm{Var}_z(\mathbb{E}[\mathbf{X}[t]|Z=z])}{\mathbb{E}_z[\mathrm{Var}(\mathbf{X}[t]|Z=z)]} \tag{5}$$

As the name mentions, the numerator is the *signal term* while the denominator is the *noise term*. In informative regions the variance should be more dependent on the signal variations due to the variable Z manipulation, than in other regions.

The success of this tool relies on the fact that it can be used before an attack to perform a PoI selection. However, jittering countermeasures will not let SNR concentrate information in narrow regions but spread it over several points. As an example, a random shift of each trace on a maximum of 100 points will require 100 times more traces to get the same quality of SNR [7].

Backward Propagation towards the input traces. Computer Vision scientists have proposed several methods to interpret the decision made by a Deep Neural Network. The simplest one [29] consists in computing the gradient of the loss function with respect to each input trace as a relevance map, defined as the following:

$$\mathbf{r}_i[t] \triangleq \frac{\partial l}{\partial \mathbf{x}_i[t]} \tag{6}$$

where l is the loss function.

The hot zones, *i.e.* the regions where the relevance map assumes highest values, localize the most significant input trace coordinates: indeed the presence of high gradient indicates that a slight variation of the signal strongly influences the loss function. This method is well adapted in contexts where relevant information is localized in the input, as it is commonly assumed in the SCA domain [3].

The principle of this method is based on the fact that the backward propagation enables to efficiently compute the gradient with respect to all the model learning parameters by propagating an error term from the top layers to the first ones. Even if the gradient computation with respect to the input traces is not relevant in the learning phase (as the data is assumed to be constant, not trainable), it is necessarily computed as a side effect of the backward propagation. Thus, the computation of the relevance map is done for free.

The main advantage of this method is that it gives, for each trace, a customized PoIs localization. Since the relevance map is computed trace-wise, the desynchronization of the trace set does not affect such a computation for a given trace. This is in contrast with the SNR tool, for which the statistical estimation is necessarily done over several samples, eventually desynchronized. A second advantage of such a visualization is that it highlights the fact that the CNN applies an intrinsic extraction of informative regions from misaligned datasets, which explains the attack successness. A characteristic of this visualization approach is that it needs a successfull CNN attack to be performed. It seems a sort of *post-mortem* characterization of the target vulnerabilities, while usually (in the TA context) the PoI search acts as a preliminary phase. Nevertheless, for evaluation purposes, this visualization post-analysis is essential to identify the sources of vulnerability, necessary to develop adapted countermeasures.

This method has also some drawbacks. The main one, as mentioned in [23], is that the use of very deep network architectures leads to the training of very complex models, and to the building of not very smooth functions. Therefore, the relevance map may show some high peaks even in absence of relevant information. At the same time, very deep networks may be more performant than simpler ones. As a direct consequence, there is a tradeoff to find in the choice of the CNN architecture between light architecture that will be more interpretable and deeper and heavier architectures that will perform better.

3 Application on the ASCAD database

In this section we present a first experiment on the ASCAD database [26] in order to verify that the CNNs are able to manage the misalignment and to apply the previous visualization method.

We considered the case where no masking is used, so we chose the target value to be:

$$Z = \mathsf{SBox}(p[3] \oplus k[3]) \oplus r_{out} \ . \tag{7}$$

The ASCAD database contains $50,000$ labelled traces, used for the training/profiling phase (more precisely, $40,000$ are used for the training, and $10,000$ for the validation), and $10,000$ traces, divided in 10 subsets used for the attack phase. The traces are composed of 700 time samples. Moreover the traces have been artificially desynchronized with a random shift between 0 and 100 points. To illustrate the desynchronization effect, we depict in Fig. 2 the SNR estimated over $50,000$ traces for the synchronized (left) and desynchronized (right) datasets. In the second case (misaligned traces), we remark the estimated SNR is not able to highlight some informative points.

For this experiment we chose the following CNN architecture:

$$\mathsf{s} \circ [\lambda]^1 \circ [\delta \circ [\alpha \circ \gamma]^1]^5, \tag{8}$$

i.e. $n_{\text{dense}} = n_{\text{conv}} = 1$ and $n_{\text{blocks}} = 5$.

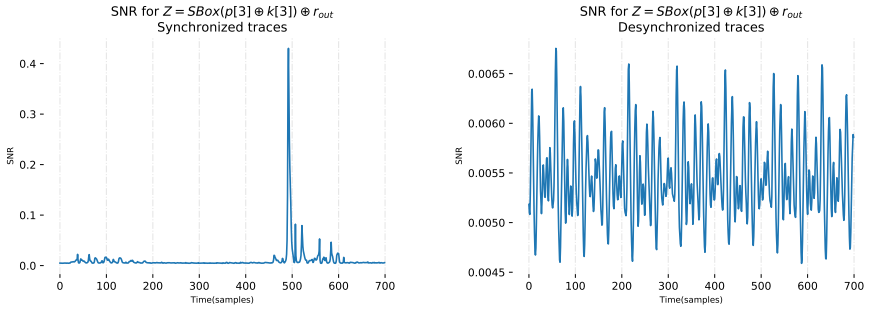

Fig. 2: Characterization with SNR for synchronized (left) and desynchronized (right) data.

We performed the training during 40 epochs and we obtained the results reported in Table 1.

Metrics	Value
Test accuracy	2.5 %
Guessing Entropy (one trace)	38.13
Min. traces required ($N^\star$)	3.00

Table 1: Performance metrics evaluated on the ASCAD test set.

The performance metrics are much better than a random prediction on the test set. Therefore we conclude the model has learned to predict the target value. In order to validate the NN extracts information from the actual region of interest, we perform the visualization technique described in Sec. 2.7. In Fig. 3 the loss function gradient with respect to several input traces is plotted. Each line shows a pattern of peaks similar to the SNR one, drawn in the left part of Fig. 2, but located at different abscisses. Globally it reveals a band of peaks that corresponds to the main peak of the SNR up to the maximum shift of 100 points. This stengthen the idea the CNN has correctly learned how to discriminate the traces in terms of their target value. Thus, visualizing such gradients can help the evaluator to appreciate the performance of its CNN architecture on one hand, and to measure the effect of jittering counter-measure on another hand.

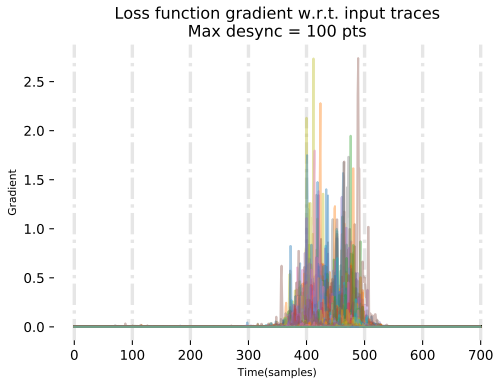

Fig. 3: Characterization with the gradient with respect to each input trace.

4 Application to hardware countermeasures

A classical hardware countermeasure against side-channel attacks consists in introducing instability in the clock. This implies the cumulation of a deforming effect that affects each single acquired clock cycle, and provokes traces misalignment on the adversary side. Indeed, since clock cycles do not have the same duration, they are sampled during the attack by a varying number of time samples. As a consequence, a simple translation of the acquisitions is not sufficient in this case to align w.r.t. an identified clock cycle. Several realignment techniques are available to manage this kind of deformations, *e.g.* [30]. The goal of this paper is not to compare a new realignment technique with the existing ones, but to show that we can get rid of the realignment pre-processing exploiting the end-to-end attack strategy provided by the CNN approach.

4.1 Performances over Artificial Augmented Clock Jitter

In this section we present the results that we obtained over two datasets named *DS_low_jitter* and *DS_high_jitter*. Each one contains $10,000$ labelled traces, used for the training/profiling phase (more precisely, $9,000$ are used for the training, and $1,000$ for the validation), and $100,000$ attack traces. The traces are composed of $1,860$ time samples. The two datasets have been obtained by artificially adding a simulated jitter effect over some synchronized original traces. The original traces were measured on an Atmega328P microprocessor. We verified that they originally encompass leakage on 34 instructions: 2 *nops*, 16 loads from the NVM and 16 accesses to look-up tables. For our attack experiments, it is assumed that the target is the first look-up table access, *i.e.* the 19th clock cycle. Our CNN has been trained to classify the traces according to the Hamming weight of the Sbox output; namely, our labels are the 9 values taken by $Z = \mathrm{HW}(\mathrm{Sbox}(P \oplus K))$.[8] To simulate the jitter effect each clock pattern has been deformed[9] by adding r new points if $r > 0$ (resp. removing r points if $r < 0$), with $r \sim \mathcal{N}(0, \sigma^2)$.[10] For the *DS_low_jitter* dataset, we fixed $\sigma^2 = 4$ and for the *DS_high_jitter* dataset we fixed $\sigma^2 = 36$. As an example, some traces of *DS_low_jitter* are depicted in the left-hand side of Fig.4: the cumulative effect of the jitter is observable by remarking that the desynchronization raises with time. Some traces of *DS_high_jitter* are depicted as well in the right-hand side of Fig. 4. For both datasets we did not operate any PoI selection, but entered the entire traces into our CNN.

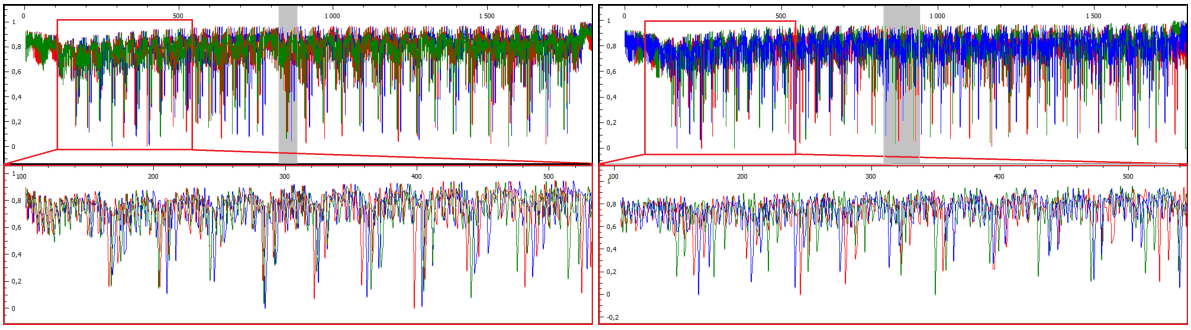

Fig. 4: Left: some traces of the *DS_low_jitter* dataset, a zoom of the part highlighted by the red rectangle is given in the bottom part. Right: some traces (and the relative) of the *DS_high_jitter* dataset. The interesting clock cycle is highlighted by the grey rectangular area.

For the CNN architecture, we chose the form

$$\mathsf{s} \circ [\lambda]^1 \circ [\delta \circ [\alpha \circ \gamma]^1]^4, \tag{9}$$

[8] The Hamming weight model is known to be particularly relevant to model the leakage occurring during register writing in Atmega328P [1].

[9] The 19th clock cycle suffers from the cumulation of the previous 18 deformations

[10] This deformation is not the same of the proposed AR technique for the DA.

i.e. $n_{\text{dense}} = n_{\text{conv}} = 1$ and $n_{\text{blocks}} = 4$.

We assisted to a strong overfitting phenomenon and we successfully reduced it by applying the DA strategy introduced in Sec.2.6. We applied both the shifting deformation SH_T with $T^\star = 200$ and $T \in \{0, 20, 40\}$ and the add-remove deformation AR_R with $R \in \{0, 100, 200\}$, training the CNN model using the 9 combinations SH_TAR_R. We performed a further experiment with much higher DA parameters, *i.e.* $\text{SH}_{200}\text{AR}_{500}$, to show that the benefits provided by the DA are limited: as expected, too much deformation affects the CNN performances (indeed results obtained with $\text{SH}_{200}\text{AR}_{500}$ will be worse than those obtained with *e.g.* $\text{SH}_{40}\text{AR}_{200}$).

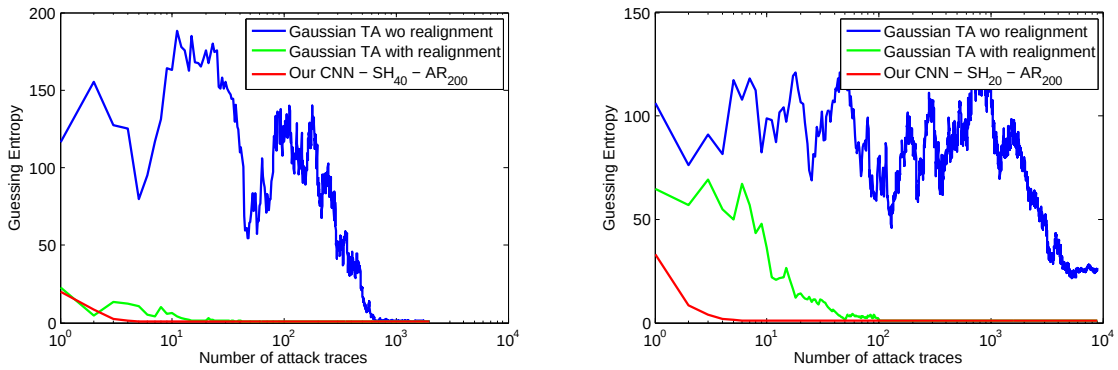

Fig. 5: Comparison between a Gaussian template attack, with and without realignment, and our CNN strategy, over the *DS_low_jitter* (left) and the *DS_high_jitter* (right).

The results we obtained are summarized in Table 2. Case SH_0AR_0 corresponds to a training performed without DA technique (and hence serves as a reference suffering from the overfitting phenomenon). It can be observed that as the DA parameters raise, the validation accuracy raises while the training accuracy decreases. This experimentally validates that the DA technique is efficient in reducing overfitting. Remarkably in some cases, for example in the *DS_low_jitter* dataset case with $\text{SH}_{100}\text{AR}_{40}$, the best validation accuracy is higher than the best training accuracy. To interpret this phenomenon we observe that the training set contains both the original data and the augmented ones (*i.e.* deformed by the DA) while the validation set only contains non-augmented data. The fact that the achieved training accuracy is lower than the validation one, indicates that the CNN does not succeed in learning how to classify the augmented data, but succeed to extract the features of interest for the classification of the original data. We judge this behaviour positively. Concerning the DA techniques we observe that they are efficient when applied independently and that their combination is still more efficient.

According to our results in Table 2, we selected the model issued using the $\text{SH}_{200}\text{AR}_{40}$ technique for the *DS_low_jitter* dataset and the one issued using the $\text{SH}_{200}\text{AR}_{20}$ technique for the *DS_higher_jitter*. In Fig. 5 we compare their performances with those of a Gaussian TA possibly combined with a realignment technique. To tune this comparison several state-of-the-art Gaussian TA have been tested. In particular for the selection of the PoIs two approaches have been applied: first we selected from 3 to 20 points maximising the estimated instantaneous SNR, secondly we selected sliding windows of 3 to 20 consecutive points covering the region of interest. For the template processing, we tried (1) the classical approach [5] where a mean and a covariance matrix are estimated for each class, (2) the *pooled* covariance matrix strategy proposed in [6] and (3) the stochastic approach proposed in [27]. In this experiment, the leakage is concentrated in peaks that are easily detected by their relatively high amplitude, so we use a simple method that consists in first detecting the peaks above a chosen threshold, then keeping all the samples in a window around these peaks. The results plotted in Fig. 5 are the best ones we obtained (via the stochastic approach over some 5-sized windows). Results show that the performances of the CNN approach are much higher than those of the Gaussian templates, both with and without realignment. This confirms the robustness of the CNN approach with respect to the jitter effect: the selection of PoIs and the realignment integrated in the training phase are effective.

Table 2: Results of our CNN in presence of artificially-generated jitter countermeasure, with different DA techniques. For each technique 4 values are given: in position a the maximal training accuracy, in position b the maximal validation accuracy, in position c the test accuracy, in position d the value of $N^\star$ (see Sec. 2.5 for definitions).

DS_low_jitter									
a	b	SH_0		SH_{20}		SH_{40}		SH_{200}	
c	d								
AR_0		100.0%	68.7%	99.8%	86.1%	98.9%	84.1%		
		57.4%	14	82.5%	6	83.6%	6		
AR_{100}		87.7%	88.2%	82.4%	88.4%	81.9%	89.6%		
		86.0%	6	87.0%	5	87.5%	6		
AR_{200}		83.2%	88.6%	81.4%	86.9%	**80.6%**	**88.9%**		
		86.6%	6	85.7%	6	**87.7%**	**5**		
AR_{500}								85.0%	88.6%
								86.2%	5
DS_high_jitter									
a	b	SH_0		SH_{20}		SH_{40}		SH_{200}	
c	d								
AR_0		100%	45.0%	100%	60.0%	98.5%	67.6%		
		40.6%	35	51.1%	9	62.4%	11		
AR_{100}		90.4%	57.3%	76.6%	73.6%	78.5%	76.4%		
		50.2%	15	72.4%	11	73.5%	9		
AR_{200}		83.1%	67.7%	**82.0%**	**77.1%**	82.6%	77.0%		
		64.0%	11	**75.5%**	**8**	74.4%	8		
AR_{500}								83.6%	73.4%
								68.2%	11

4.2 Performances on a Secure Smartcard

As a last (but most challenging) experiment we deployed our CNN architecture to attack an AES hardware implementation over a modern secure smartcard (secure implementation on 90nm technology node). On this implementation, the architecture is designed to optimize the area, and the speed performances are not the major concern. The architecture is here minimal, implementing only one hardware instance of the SubByte module. The AES SubByte operation is thus executed serially and one byte is processed per clock cycle. To protect the implementation, several countermeasures are implemented. Among them, a hardware mechanism induces a strong jitter effect which produces a high traces desynchronization. The bench is setup to trig the acquisition of the trace on a peak which corresponds to the processing of the first byte. Consequently, the set of traces is aligned according to the processing of the first byte while the other bytes leakages are completely misaligned. To illustrate the effect of this misalignment, the SNR characterizing the (aligned) first byte and the (misaligned) second byte are computed (according to the formula given in [4]) using a set of $150,000$ traces labelled by the value of the SubByte output (256 labels). These SNRs are depicted in the top part of Fig. 6. The SNR of the first byte (in green) detects a quite high leakage, while the SNR of the second byte (in blue) is nullified. A zoom of the SNR of the second peak is proposed in the bottom-left part of Fig. 6. In order to confirm that the very low SNR corresponding to the second byte is only due to the desynchronization, the patterns of the traces corresponding to the second byte have been resynchronized using a peak-detection-based algorithm, quite similar to the one applied for the experiments of Sec.4.1. Then the SNR has been computed onto these new aligned traces and has been plot in red in the top-left part of Fig. 6; this SNR is very similar to that of the first byte. This clearly shows that the leakage information is contained into the trace but is efficiently hidden by the jitter-based countermeasure.

We applied the CNN approach onto the rough set of traces (without any alignement). First a $2,500$-sized window of the trace has been selected to input CNN. The window, identified by the vertical cursors in the bottom part of Fig. 6, has been selected to ensure that the pattern corresponding to the leakage of the second byte is inside the selection. At this step, it is important to notice that

such a selection is not at all as meticulous as the selection of PoIs required by a classical TA approach. The training phase has been performed using 98,000 labelled traces; 1,000 further traces have been used for the validation set. We performed the training phase over a desktop computer equipped with an Intel Xeon E5440 @2,83GHz processor, 24Gb of RAM and a GeForce GTS 450 GPU. Without data augmentation each epoch took about 200s.[11] The training stopped after 25 epochs. Considering that in this case we applied an early-stopping strategy that stopped training after 20 epochs without validation loss decrement, it means that the final trainable weights are obtained after 5 epochs (in about 15 minutes). The results that we obtained are summarized in Table 3. They prove not only that our CNN is robust to the misalignment caused by the jitter but also that the DA technique is effective in raising its efficiency. A comparison between the CNN performances and the best results we obtained over the same dataset applying the realignment-TA strategy in the right part of Fig.6. Beyond the fact that the CNN approach slightly outperforms the realignment-TA one, and considering that both case-results shown here are surely non-optimal, what is remarkable is that the CNN approach is potentially suitable even in cases where realignment methods are impracticable or not satisfying. It is of particular interest in cases where sensitive information does not lie in proximity of peaks or of easily detectable patterns, since many resynchronization techniques are based on pattern or peak detection. If the resynchronization fails, the TA approach falls out of service, while the CNN one remains a further weapon in the hands of an attacker.

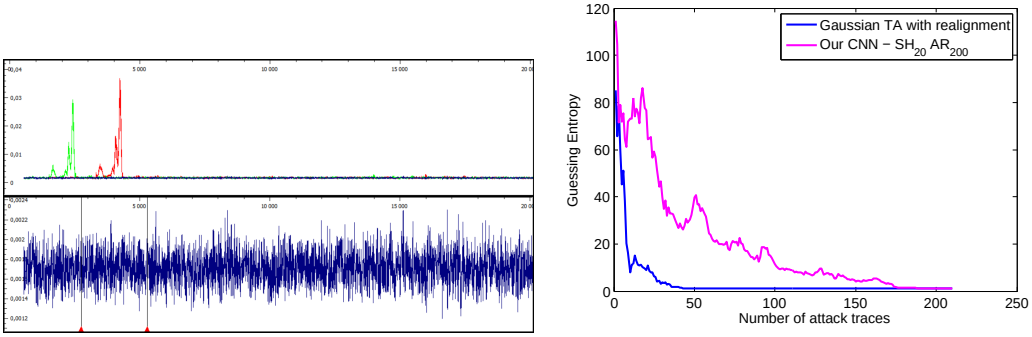

Fig. 6: Top Left: in green the SNR for the first byte; in blue the SNR for the second byte; in red the SNR for the second byte after a trace realignment. Bottom Left: a zoom of the blue SNR trace. Right: comparison between a Gaussian template attack with realignment, and our CNN strategy, over the modern smart card with jitter.

		SH_0AR_0		$SH_{10}AR_{100}$		$SH_{20}AR_{200}$	
a	b	35.0%	1.1%	12.5%	1.5%	**10.4%**	**2.2%**
c	d	1.2%	137	1.3%	89	**1.8%**	**54**

Table 3: Results of our CNN over the modern smart card with jitter.

Conclusions and Perspectives

In this paper we have proposed an end-to-end profiling attack approach for evaluation, based on the CNNs. We claimed that such a strategy would be effective still in presence of trace misalignment. We successfully verified our claim by performing the CNN-based attack over different kind of misaligned data. This property represents a great practical advantage compared to the nowadays state-of-the-art profiling attacks, that require a meticulous trace realignment in order to be efficient. It represents also a solution to the problem of the selection of points of interest issue: CNNs efficiently manage high-dimensional data, allowing the evaluator to simply select large windows. Moreover, thanks to the visualization tools we proposed, the evaluator, who needs to characterize and identify the vulnerabilities allowing CNN attack to succeed, is able to retrieve the PoIs in a trace-wise manner. In order to deal with the risk of overfitting phenomenon, we proposed two Data Augmentation techniques adapted to

[11] raising to about 2000s when $SH_{20}DA_{200}$ data augmentation is performed (data are augmented online during training)

misaligned side-channel traces. All the experimental results we obtained have proved that they provide a great benefit to the CNN strategy.

In this work, CNNs are applied in a classification-oriented manner: the CNN model is used to classify traces with respect to a sensitive variable, then the outcomes of several classification are jointly exploited to perform an Advanced Attack (aka Differential Power Attack) in order to retrieve a key chunk. Future works might concern the definition of a new ML task, to replace the classification one in order to perfectly fit advanced attack scenarios, and the identification of specialized metrics (*e.g.* loss functions, evaluation metrics more adequate than the accuracy) to solve such a new task.

In general, we are convinced of the importance of further exploring DL techniques in side-channel context. At the same time we are aware of the lack of clear theoretical foundations that would give guides about the choice of the hyper-parameters that influence the performances of the DL architectures. For this reason we believe that researchers should share their efforts in developing and analysing *ad hoc* methodologies to tune side-channel-oriented neural networks. To this aim, a publication appeared in the *Cryptology ePrint archive* on January 2018 [26] proposing a fully-reported set of benchmarks performed over some electromagnetic emanation acquisitions. The whole acquisitions database were published as well, including all the sources of the target implementation. We wish this open platform may serve as a common basis for researchers willing to compare their new architectures or their improvements of existing models. This kind of public databases have been central tools in the development of deep learning solutions in many other domains, for example in image recognition context.

References

1. Sonia Belaïd, Jean-Sébastien Coron, Pierre-Alain Fouque, Benoît Gérard, Jean-Gabriel Kammerer, and Emmanuel Prouff. Improved side-channel analysis of finite-field multiplication. In *Cryptographic Hardware and Embedded Systems - CHES 2015 - 17th International Workshop, Saint-Malo, France, September 13-16, 2015, Proceedings*, pages 395–415, 2015.
2. Christopher M.. Bishop. *Pattern recognition and machine learning*. Springer, 2006.
3. Eleonora Cagli, Cécile Dumas, and Emmanuel Prouff. *Enhancing Dimensionality Reduction Methods for Side-Channel Attacks*, pages 15–33. Springer International Publishing, Cham, 2016.
4. Eleonora Cagli, Cécile Dumas, and Emmanuel Prouff. Kernel discriminant analysis for information extraction in the presence of masking. In *International Conference on Smart Card Research and Advanced Applications*, pages 1–22. Springer, 2016.
5. Suresh Chari, JosyulaR. Rao, and Pankaj Rohatgi. Template attacks. In Burton S. Kaliski, Cetin K. Koc, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002*, volume 2523 of *Lecture Notes in Computer Science*, pages 13–28. Springer Berlin Heidelberg, 2003.
6. Omar Choudary and Markus G Kuhn. Efficient template attacks. In *Smart Card Research and Advanced Applications*, pages 253–270. Springer, 2014.
7. Christophe Clavier, Jean-Sébastien Coron, and Nora Dabbous. Differential power analysis in the presence of hardware countermeasures. In *Cryptographic Hardware and Embedded Systems - CHES 2000, Second International Workshop, Worcester, MA, USA, August 17-18, 2000, Proceedings*, pages 252–263, 2000.
8. Jean-Sébastien Coron and Ilya Kizhvatov. An efficient method for random delay generation in embedded software. In *Cryptographic Hardware and Embedded Systems-CHES 2009*, pages 156–170. Springer, 2009.
9. Jean-Sébastien Coron and Ilya Kizhvatov. Analysis and improvement of the random delay countermeasure of ches 2009. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 95–109. Springer, 2010.
10. Thomas M Cover and Joy A Thomas. *Elements of information theory*. John Wiley & Sons, 2012.
11. François Durvaux, Mathieu Renauld, François-Xavier Standaert, Loic van Oldeneel tot Oldenzeel, and Nicolas Veyrat-Charvillon. Efficient removal of random delays from embedded software implementations using hidden markov models. In *International Conference on Smart Card Research and Advanced Applications*, pages 123–140. Springer, 2012.
12. Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.
13. Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012.
14. Gabriel Hospodar, Benedikt Gierlichs, Elke De Mulder, Ingrid Verbauwhede, and Joos Vandewalle. Machine learning in side-channel analysis: a first study. *Journal of Cryptographic Engineering*, 1(4):293–302, 2011.

15. Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.
16. Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.
17. Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.
18. Henry W Lin and Max Tegmark. Why does deep and cheap learning work so well? *arXiv preprint arXiv:1608.08225*, 2016.
19. Houssem Maghrebi, Thibault Portigliatti, and Emmanuel Prouff. Breaking cryptographic implementations using deep learning techniques. In *International Conference on Security, Privacy, and Applied Cryptography Engineering*, pages 3–26. Springer, 2016.
20. Stefan Mangard. Hardware countermeasures against dpa–a statistical analysis of their effectiveness. In *Topics in Cryptology–CT-RSA 2004*, pages 222–235. Springer, 2004.
21. Zdenek Martinasek, Jan Hajny, and Lukas Malina. Optimization of power analysis using neural network. In *International Conference on Smart Card Research and Advanced Applications*, pages 94–107. Springer, 2013.
22. Zdenek Martinasek and Vaclav Zeman. Innovative method of the power analysis. *Radioengineering*, 22(2):586–594, 2013.
23. Grégoire Montavon, Wojciech Samek, and Klaus-Robert Müller. Methods for interpreting and understanding deep neural networks. 73:1–15.
24. Sei Nagashima, Naofumi Homma, Yuichi Imai, Takafumi Aoki, and Akashi Satoh. Dpa using phase-based waveform matching against random-delay countermeasure. In *Circuits and Systems, 2007. ISCAS 2007. IEEE International Symposium on*, pages 1807–1810. IEEE, 2007.
25. Lutz Prechelt. *Early Stopping — But When?*, pages 53–67. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.
26. Emmanuel Prouff, Remi Strullu, Ryad Benadjila, Eleonora Cagli, and Cecile Dumas. Study of deep learning techniques for side-channel analysis and introduction to ascad database. Cryptology ePrint Archive, Report 2018/053, 2018. `https://eprint.iacr.org/2018/053`.
27. Werner Schindler, Kerstin Lemke, and Christof Paar. A stochastic model for differential side channel cryptanalysis. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 30–46. Springer, 2005.
28. Patrice Y Simard, David Steinkraus, John C Platt, et al. Best practices for convolutional neural networks applied to visual document analysis. In *ICDAR*, volume 3, pages 958–962. Citeseer, 2003.
29. Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *CoRR*, abs/1312.6034, 2013.
30. Jasper GJ van Woudenberg, Marc F Witteman, and Bram Bakker. Improving differential power analysis by elastic alignment. In *Cryptographers Track at the RSA conference*, pages 104–119. Springer, 2011.
31. Nicolas Veyrat-Charvillon, Marcel Medwed, Stéphanie Kerckhof, and François-Xavier Standaert. Shuffling against side-channel attacks: A comprehensive study with cautionary note. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 740–757. Springer, 2012.
32. Sebastien C Wong, Adam Gatt, Victor Stamatescu, and Mark D McDonnell. Understanding data augmentation for classification: when to warp? In *Digital Image Computing: Techniques and Applications (DICTA), 2016 International Conference on*, pages 1–6. IEEE, 2016.
33. Shuguo Yang, Yongbin Zhou, Jiye Liu, and Danyang Chen. Back propagation neural network based leakage characterization for practical security analysis of cryptographic implementations. In *International Conference on Information Security and Cryptology*, pages 169–185. Springer, 2011.