Online Performance Evaluation of Deep Learning Networks for Profiled Side-Channel Analysis

Damien Robissout[1], Gabriel Zaid[1,2], Brice Colombier[1], Lilian Bossuet[1], and Amaury Habrard[1]

[1] Univ Lyon, UJM-Saint-Etienne, CNRS Laboratoire Hubert Curien UMR 5516
F-42023, Saint-Etienne, France, firstname.lastname@univ-st-etienne.fr
[2] Thales ITSEF, Toulouse, France, firstname.lastname@thalesgroup.com

Abstract. Deep learning based side-channel analysis has seen a rise in popularity over the last few years. A lot of work is done to understand the inner workings of the neural networks used to perform the attacks and a lot is still left to do. However, finding a metric suitable for evaluating the capacity of the neural networks is an open problem that is discussed in many articles. We propose an answer to this problem by introducing an online evaluation metric dedicated to the context of side-channel analysis and use it to perform early stopping on existing convolutional neural networks found in the literature. This metric compares the performance of a network on the training set and on the validation set to detect underfitting and overfitting. Consequently, we improve the performance of the networks by finding their best training epoch and thus reduce the number of traces used by 30%. The training time is also reduced for most of the networks considered.

Keywords: Side-Channel Attacks · Metrics · Deep Learning · Underfitting · Overfitting

1 Introduction

Side-channel attacks are a class of cryptographic attacks in which an adversary exploit vulnerabilities of a system by analyzing its physical properties, such as the power consumption [8] or electromagnetic emanations [1], to reveal secret information. The implementation of a cryptographic algorithm involves the manipulation of sensitive variables which depend on the secret. This is the base concept behind side-channel attacks, among which we find *profiling attacks*. In this scenario, an adversary has access to a test device on which he can choose the plaintext and the secret key. With that information, he is able to estimate the conditional distribution associated with the sensitive variable of interest. On a target device containing a secret to retrieve, the adversary can then predict the actual sensitive value and reveal the secret. In 2002, the first profiling attacks, named *template attacks*, were introduced by Chari *et al.* [4] but their proposal was limited by its computational complexity.

Very similar to profiling attacks, deep learning algorithms were inevitably applied in the side-channel context. Indeed, some articles have shown the robustness of convolutional neural networks (CNNs) to the most common countermeasures, namely *masking* [10,11] and *desynchronization* [3,16]. One of their main advantages is that they do not require pre-processing of the traces. The training process, through which the network learns to solve one specific problem, consists in two phases [5]: the forward propagation and the backward propagation. Given an input, the aim of the forward propagation is to feed training examples to the network in the forward direction by processing successive linear and non-linear transformations in order to predict a value related to the input. Once this is done, the backward propagation measures the error between the predictions and the correct output and tries to reduce it by updating the parameters that compose the network.

To evaluate the training process and its performance, classical deep learning metrics can be used. One of the most popular is the accuracy. Unfortunately, as Picek *et al.* have shown [12], this metric is poorly suited in the context of side-channel analysis. Using the accuracy tends to favor the class with the highest output probability. This solution cannot be considered, in side-channel analysis, because the classifiers are often only loosely correlated with the true classification because of the very small leakage information present in the traces used for learning. Then, to perform a successful attack, the adversary must combine the classification results obtained for multiple traces to extract the estimate of the true class.

Contributions In this article, we evaluate the ability of a network to generalize the knowledge found in the learning samples. By comparing the performance on the training set, containing the examples used by the network to learn, and the validation set, containing examples the network has never seen before, we get an insight into how well the network performs on new examples. Our proposed metric, called $\Delta^{d}_{train,val}$, is derived from the sucess rate [14], commonly used in side-channel analysis. Defined as the number of succesful attacks over 100 realizations, the success rate is a suitable metric to evaluate the performance of attacks compared to the accuracy, which corresponds to taking into account only one trace to perform only one attack. By measuring the number of traces that are needed to get a successful d^{th} order success rate on the training and the validation sets, we can accurately evaluate the ability of the network to generalize its knowledge. We confirm the relevance of our metric by applying it on the ASCAD public dataset [13]. Using $\Delta^{d}_{train,val}$ has two benefits: during training, this metric can be used to detect the internal state of the network (underfitting/overfitting) and to find the best number of epochs to perform early stopping [5]. Furthermore, $\Delta^{d}_{train,val}$ helps to compare the performance between the networks once they are trained. Therefore, it allows us to optimize the performance of a network used for side-channel analysis.

Article Organization The article is organized as follows. Section 2 is dedicated to the neural networks and evaluation metrics used in the article. After defining the machine learning approach for evaluating the generalization capacity of a network, Section 3 defines a new evaluation metric, called $\Delta^{d}_{train,val}$, which measures this generalization in the context of side-channel analysis. This new metric is applied on the public dataset ASCAD [13] and its main CNN architecture in Section 4 and then compared against the only comparable existing metric, the guessing entropy bias variance decomposition [15]. Finally in Section 5, we discuss some future works that could be investigated and conclude on the results presented in the article.

2 Preliminaries

2.1 Notations

Let calligraphic letters $\mathcal{X}$ denote sets, the corresponding capital letters X (resp. bold capital letters) denote random variables (resp. random vectors $\mathbf{T}$) and the lowercase x (resp. $\mathbf{t}$) denote their realizations. The i-th entry of a vector $\mathbf{t}$ is written as $\mathbf{t}[i]$.

A side-channel trace is a random vector $\mathbf{T} \in \mathbb{R}^D$ where D defines the dimension of the trace. The targeted sensitive variable is $Z = f(P, K)$ where f denotes a cryptographic primitive, P $(\in \mathcal{P})$ denotes a public variable (*e.g.* plaintext or ciphertext) and K $(\in \mathcal{K})$ denotes a part of the key (*e.g.* byte) that an adversary tries to retrieve. Z takes values in $\mathcal{Z} = \{s_1, ..., s_{|\mathcal{Z}|}\}$. Let us denotes $\mathbf{k}^*$ the secret key used by the cryptographic algorithm.

2.2 Profiling attacks

A profiling attack is performed in two stages: a profiling phase and a matching phase. During the profiling phase, an adversary has access to a test device on which he can control the input and the secret key of the cryptographic algorithm. He uses this knowledge to find the relevant leakages depending on the sensitive variable Z. The adversary builds a model $F : \mathbb{R}^D \rightarrow \mathbb{R}^{|\mathcal{Z}|}$ that estimates the probability $Pr[\mathbf{T}|Z = z]$ from a profiling set $\mathcal{T} = \{(\mathbf{t}_0, z_0), \ldots, (\mathbf{t}_{N_p-1}, z_{N_p-1})\}$ of size N_p.

Once the model F is built, in the matching phase, the adversary estimates which intermediate value is processed. By predicting this sensitive variable and knowing the public variable used during the encryption, the adversary can compute a score vector, based on $F(\mathbf{t}_i), i \in [\![0, N_a - 1]\!]$, for each trace included in a dataset of N_a attack traces. The key candidate with the highest values will be defined as the recovered key.

To evaluate the performance related to the estimations, we can classify all the key candidates into a vector of size $|\mathcal{K}|$, denoted $\mathbf{g} = (g_1, g_2, ..., g_{|\mathcal{K}|})$, following their resulting probability. We consider g_1 as the most likely candidate and $g_{|\mathcal{K}|}$ as the least likely one. Let us denote $\mathbf{g}(\mathbf{k}^*[b])$ the actual position of the b^{th} byte of the secret key in $\mathbf{g}$. This position is called *rank*. In side-channel analysis, a common metric, called guessing entropy (GE) [14], defines the average rank of a byte b of $\mathbf{k}^*$, denoted $\mathbf{k}^*[b]$, among all key hypotheses. We consider an attack as successful, using N_a traces, when the guessing entropy is equal to 1.

The rank of the correct key gives us an insight into how well our model performs. A related metric is the success rate, the probability that an attack succeeds in recovering $\mathbf{k}^*[b]$ among all the hypotheses. A success rate of p means that p attacks, over 100 realizations, succeed to retrieve $\mathbf{k}^*[b]$. In [14], Standaert *et al.* propose to extend the notion of success rate to an arbitrary order d. Let $A_{E_k,L}$ be an adversary trying to attack a cryptographic computation E_k using to a leakage model L. The adversary has to conduct some experiments $Exp^d_{A_{E_k,L}}$ in order to exploit the relevant information that leaks. The output of the attack is a guessing vector $\mathbf{g}$ of length d that is composed of the key candidates sorted according to the attack result. If $\mathbf{k}^*[b] \in \mathbf{g}$, then we consider the attack as a success and $Exp^d_{A_{E_k,L}} = 1$. Thus, the d^{th} order success rate can be defined as:

$$SR^d_{A_{E_k,L}} = Pr[Exp^d_{A_{E_k,L}} = 1].$$

In other words, the d^{th} order success rate is defined as the probability that the target secret $\mathbf{k}^*[b]$ is ranked among the d first key guesses in the score vector. In the rest of the article, the d^{th} order success rate is denoted SR^d.

2.3 Neural networks

Neural networks have risen in popularity over the past ten years due to the increase in computing power and the democratization of GPUs. They proved to be very efficient at solving a large variety of problems like classification or feature extraction. It is for these reasons that the application of machine learning techniques was eventually explored in side-channel analysis [6,2,9] and soon after followed the application of deep learning and the use of neural networks [10,3]. For a classification task, a neural network aims at constructing a function $F : \mathbb{R}^D \rightarrow \mathbb{R}^{|\mathcal{Z}|}$ that computes an output called a *prediction* represented as a vector of size $|\mathcal{Z}|$, the number of possible classes. To solve a classification problem, the function F has to find the right prediction y associated with the input $\mathbf{t}$ with high confidence. To approximate the optimal solution, a neural network must be trained given a profiling set of N_p pairs $(\mathbf{t}_i, y_i)$ where $\mathbf{t}$ is the i-th profiling input and y_i is the label associated with the i-th input. To construct F, neural networks are built from several layers composed of unit blocks called neurons. These neurons perform operations to select the relevant features that allow for an efficient classification of $\mathbf{t}$.

One special kind of network is the convolutional neural network. The particularity of CNNs is the use of filters for improving the pattern recognition. The main advantage of the filters and convolutional layers is their time-invariance property that allows the network to be robust against desynchronization (*e.g.* shifting, jitter) [3,16]. Therefore, the resynchronization pre-processing is not necessary anymore. However, Zhou and Standaert [17] have shown that resynchronization still helps the network during the learning phase and improves the network performance.

Once the architecture of the network is fixed, the training can begin but we need to be able to evaluate *how well* a network is learning. In order to do so, some evaluation metrics have been developed.

2.4 Evaluation metrics

In machine learning, to accurately evaluate the networks, it is common to look at the progression of different metrics that can be decomposed into two categories:

- **The learning metrics**, such as the *empirical risk*, which is the average of a *loss function*, over all the examples of the training set, estimating the classification error. They help the network to update its trainable parameters (*i.e.* weights) in order to optimize F.

The classification error is defined by a comparison between a label y_i and the related predicted value $\hat{y}_i$. The function measuring this error is the loss function. The goal of the training is to minimize the loss in order to reduce the errors made by the network on the training examples. The most commonly used loss is the *categorical cross-entropy* [5]. Minimizing the categorical cross-entropy reduces the dissimilarity between the correct distributions and the predicted distributions for a set of inputs. Thus, the evaluation of this learning metric helps to interpret the training error of a model. It can be visualized after the training to better understand how well the network learned.

- **The performance metrics**, such as the *accuracy*, that define the performance of a network for a given input. This metric computes the number of good predictions for a set of traces. In side-channel analysis, it corresponds to a first order success rate using only one trace [12].

The performance metrics are exploited in order to evaluate the internal state of a network [5] and used to detect both *underfitting* and *overfitting.*

Underfitting typically describes the moment of the learning phase where the network has not seen enough training examples to extract relevant information from them. It is therefore not able to make correct predictions. It can also be a sign that the architecture of the network is not complex enough to properly estimate the underlying function. To prevent underfitting, it is possible to increase the number of training examples, the number of epochs and the complexity of the network [5].

Overfitting happens when the network is starting to learn features from the training examples that are not relevant for generalization. Thus it is loosing its generalization power which means it is better at predicting training examples but performs poorly on the validation set. The consequence is that the network learns the training examples by heart, learning features that are not useful for classification purposes. For example, in side-channel analysis, we can assume that the network learns noise patterns from the training set, where the noise is considered to be independent from the intermediate value Z. Since those patterns are random, they will most likely negatively influence the prediction of new examples. Another factor can be an overly complex architecture. As a consequence, the network is able to estimate functions F much more complex than the optimal one. To reduce the impact of overfitting, some techniques can be used such as data augmentation [3], noise addition [7], regularization or a more fitting architecture [16].

As mentioned before, the accuracy is not a suitable metric for evaluating the network performance for side-channel attacks. It differs from the paradigm of the side-channel attacks where one considers and uses a set of traces to accumulate information about the secret key. According to Picek et al. [12], it is more relevant to use the success rate when a side-channel developer wants to evaluate the performance related to his network. Indeed, contrary to the accuracy, the success rate is based on the accumulation of information over several traces.

2.5 Related work on metrics for side-channel analysis

The guessing entropy is a suitable metric to evaluate the performance of a network. However, this tool does not give an insight about the internal state of the network. Indeed, the guessing entropy evaluates the performance associated with a set of traces but does not compare the performance between the training dataset and the validation one. Then, it is difficult for an evaluator to identify the appropriate moment where the model starts overfitting. Detection of overfitting is an important problem to consider since it lowers the performance of the network. Moreover, an early detection of the overfitting can bring a substantial gain in terms of training time by reducing the number of epochs the network has to train for. Therefore, to solve this problem, van der Valk and Picek [15] introduced the guessing entropy bias variance decomposition (GEBVD). By doing so, they are able to separately study the evolution of the bias and the variance of the guessing entropy and draw conclusions on the influence of some hyperparameters on the performance of the network. A high bias may indicate that the network is underfitting and a high variance that it is overfitting.

Our approach is different in the sense that we evaluate the generalization capability of a given architecture by studying its performance at training and validation. If the network can be improved, *e.g.* when having much better performance on training than on validation, then regularization can be applied to improve the training of the network and reach better performance. We also study the link between good performance at training and good performance at validation.

In order to do that, we propose a new metric called $\Delta^{d}_{train,val}$. The aim of this new metric is to characterize the generalization power of a network dedicated to side-channel analysis.

3 $\Delta^{d}_{train,val}$: a deep learning evaluation metric for side-channel analysis

$\Delta^{d}_{train,val}$ uses a common side-channel metric, namely the success rate, to evaluate the performances of a network both on the training set and on the validation set. Therefore, $\Delta^{d}_{train,val}$ allows the attacker to draw conclusions on the internal state of the network, namely underfitting or overfitting.

3.1 $\Delta^{d}_{train,val}$: internal state detection

Let a *model* be the result function F of the training of an architecture for a given amount of epochs. We define $N^{d}_{train}(model)$ and $N^{d}_{val}(model)$ as the minimal number of traces that a model needs in order to reach an d^{th}-order success rate:

$$N^{d}_{train}(model) = min\{n_{train} \mid \forall n \geq n_{train},\ SR^{d}_{train}(model(n)) = 90\%\}$$

and,

$$N^{d}_{val}(model) = min\{n_{val} \mid \forall n \geq n_{val},\ SR^{d}_{val}(model(n)) = 90\%\}.$$

An d^{th} order success rate means that the attacker has at most d key guesses to test after the attack in order to recover the correct one.

By comparing the performances of the attacks on training and on validation, we obtain information on how well the network is able to generalize its knowledge. The choice of the euclidean norm seemed the most natural to compare two number of traces. Therefore, $\Delta^{d}_{train,val}$ is computed as follow:

$$\Delta^{d}_{train,val} = |N^{d}_{val} - N^{d}_{train}|.$$

The computation of N^{d}_{train} and N^{d}_{val} is based on the existing side-channel metrics that are known to exploit the full information available. By comparing them, we combine the machine learning and the side-channel approaches to evaluate any network.

Our proposal has the advantage that it is possible to evaluate the internal state of the network during the training and afterwards by visualizing this new metric. Indeed, we are able to efficiently visualize when our network is in an underfitting, good or overfitting state. The choice of a success rate of 90% was made, instead of 100%, to bring more stability to the values of N^{d}_{train} and N^{d}_{val}.

3.2 Detection of overfitting/underfitting

The evolution of $\Delta^{d}_{train,val}$ and the internal state of a model are illustrated with three areas in Figure 1, showing an example of the evolution of $\Delta^{1}_{train,val}$ during the training of a network:

- **Underfitting**: as mentioned in Section 2.4, underfitting occurs when the network has not learned enough information from the training set. Thus, the attack cannot be performed. The values related to N^d_{train} and N^d_{val} are often not defined or are both very high. In other words, when the number of training epochs is low, $\Delta^d_{train,val}$ is also not defined or its value is very high (area on the left of Figure 1). Such cases call for an augmentation of the number of epochs or the amount of training data to reach a success rate of 90%.

- **Good trade-off**: when the network is able to learn enough relevant information from the training set, the value of $\Delta^d_{train,val}$ converges towards $N_{bias}(\mathcal{T}_{train})$, which represent the minimal difference between N^d_{train} and N^d_{val} given a training set $\mathcal{T}_{train}$. Let us denote e the number of epochs needed to reach a good trade-off, we have:

$$\Delta^d_{train,val} \xrightarrow[epoch \to e]{} N_{bias}(\mathcal{T}_{train}). \tag{1}$$

  A good trade-off occurs at the number of epochs for which $\Delta^d_{train,val}$ is close to $N_{bias}(\mathcal{T}_{train})$. The network generalizes the relevant information for performing as well on the training set as on the validation set (area in the middle of Figure 1). This metric gives information on the ability of the network to generalize well. In the following, we use best trade-off to describe the best network we were able to train without guaranteeing that it is optimal.

- **Overfitting**: as mentioned in Section 2.4, overfitting occurs when the network is starting to learn the training features by heart. As a consequence, the network looses its generalization power on the validation set to obtain better performance on the training set. The value of N^d_{train} approaches 1 and more generally converges to a very small value. At the same time, N^d_{val} increases towards a value $N_{max}(\mathcal{T}_{train})$, *i.e.*:

$$\Delta^d_{train,val} \xrightarrow[epoch \to \infty]{} N_{max}(\mathcal{T}_{train}). \tag{2}$$

  This value represents the maximal number of traces needed by the network to reach a success rate of 90% once the training has stabilized, *i.e.* the network cannot improve its performance on the training set anymore. Thus, the update of its weights does not change the prediction made on the validation set and the performance of the overfitted model stays low (area on the right of Figure 1) .

With that knowledge, we can optimize the training of the network by performing early stopping.

3.3 $\Delta^d_{train,val}$: a suitable metric for early stopping

Early stopping consists in monitoring the learning of the network and stopping the training when the learning metrics, usually the accuracy and the loss, are

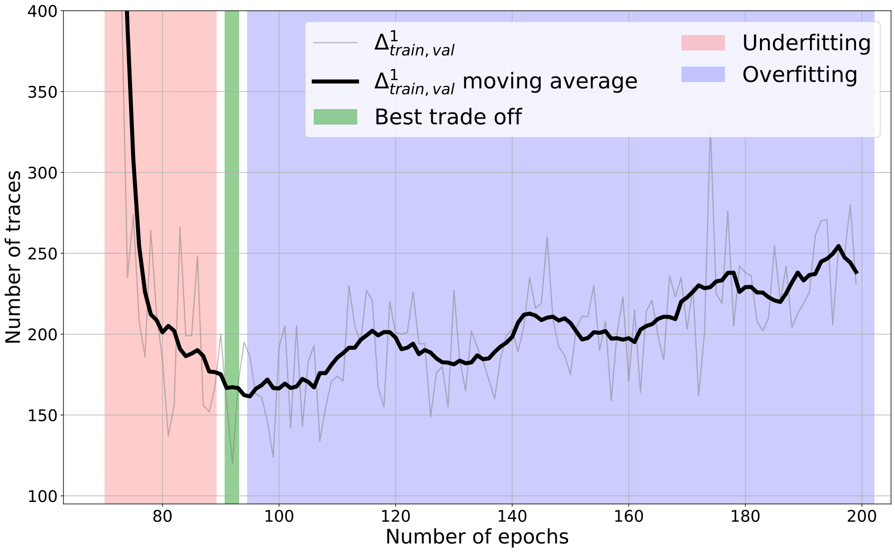


Fig. 1: Evolution of $\Delta^1_{train,val}$ for different number of epochs. The plot of $\Delta^1_{train,val}$ is done using a moving average of size 10.

optimal, *i.e.* just before the network starts to overfit. As mentioned in [5], early stopping has other effects on the network. It is a mean of applying regularization without having to penalize weights and therefore can be used in parallel without other methods of regularization. It can also be considered as an additional hyperparameter, the number of training steps or number of epochs, that is tuned during the training of the network using the computation of its associated metrics. All in all, it is recommended to perform early stopping as long as there is an appropriate metric to use in combination. The learning metrics are computed both on the training set and on a validation set to be able to properly tell whether or not the network performs well. The comparison between performance at training and on validation yields important information about the network. In our case, $\Delta^d_{train,val}$ gives us the information needed to identify when to stop the training (see Figure 1). If our metric does not grow for a given number of epochs then we can assume the optimal state is reached and stop the training.

The computation of $\Delta^d_{train,val}$ can be done in parallel to the training of the next epoch. As illustrated on Figure 2, as long as the training time of an epoch is superior to the time it takes to compute $\Delta^d_{train,val}$, there is no time overhead. In the cases where it takes longer than an epoch, for example when the training set is small, the computation of $\Delta^d_{train,val}$ can be done once every few epochs to prevent time overhead as shown in Figure 3.

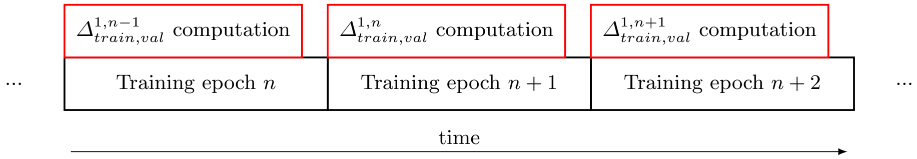

$\Delta^{1,n-1}_{train,val}$ computation
$\Delta^{1,n}_{train,val}$ computation
$\Delta^{1,n+1}_{train,val}$ computation
...
Training epoch n
Training epoch $n+1$
Training epoch $n+2$
...
time

Fig. 2: Computation of $\Delta^1_{train,val}$ for different consecutive epochs when the computation is shorter than the training of one epoch

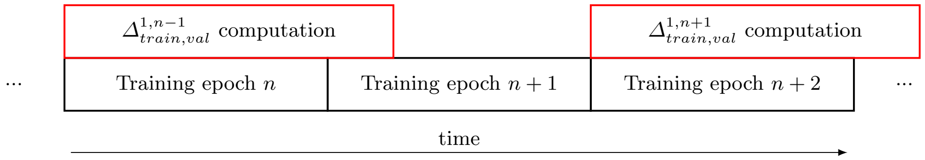

$\Delta^{1,n-1}_{train,val}$ computation
$\Delta^{1,n+1}_{train,val}$ computation
...
Training epoch n
Training epoch $n+1$
Training epoch $n+2$
...
time

Fig. 3: Computation of $\Delta^1_{train,val}$ for different consecutive epochs when the computation is longer than the training of one epoch

4 Experimental results

For all the experiments presented in this section we computed $\Delta^1_{train,val}$ on neural networks during their training phase to evaluate their best capacity. The networks are trained using the ASCAD[3] database, introduced in [13] to be a common database for researchers. The device used to acquire the electromagnetic measurements is an 8-bit AVR ATMega8515 running an AES implementation secured against first order side-channel attacks. The dataset is composed of a training set of 50000 traces and a test set of 10000 traces both coming from the same device. From the raw traces, 700 points are selected which contain leakage of the mask and the masked value of the third key byte in the first round. The leakage model associated with the traces is:

$$Y(\mathbf{k}^*) = \mathit{Sbox}(\mathbf{p}[3] \oplus \mathbf{k}^*[3]),$$

where $\mathbf{p}$ is the plaintext and $\mathbf{k}^*$ the correct key.

The main advantage of using this database is to compare our results to the ones presented in the ASCAD reference article.

4.1 Early stopping on the ASCAD database

To perform early stopping during the training, we computed $\Delta^1_{train,val}$ at the end of each epoch. We started by applying this method to the architecture CNN_{best} presented in Appendix A, Table 2. This is the best performing CNN architecture presented in [13]. It uses the categorical cross-entropy (CCE) loss, the RMSprop optimizer with a learning rate of 10^{-5} and a base number of epochs for training of 75.

[3] `https://github.com/ANSSI-FR/ASCAD`

For readability, we show, in Figure 4, a moving average of $\Delta^1_{train,val}$ using a window of size 3. This is done to smooth out the curve and better see its global shape and explains why the minimal value of the metric is not always at the minimum of the curve. The moving average is thus not taken into account while computing the minimal value of $\Delta^1_{train,val}$. The evolution of N_{train} for a whole training is also shown and the attacks are performed on a set of synchronized traces, called $Desync_0$. The addition of the performance at training helps to understand the state of the network. At around 30 epochs, the training of the network allows it to reach a success rate of 90% using 4900 traces from the validation set. From that point on, the value of $\Delta^1_{train,val}$ quickly decreases to reach a minimal value at 47 epochs. At this point, it is able to reach a success rate of 90% with around 800 traces. It then slowly increases again to stabilize in an unstable regime at $N_{max}(\mathcal{T}_{train}) \approx 3000$ traces as mentioned in Equation 2. The original article introducing this network [13] recommended 75 epochs of training to reach the best performances but here we find that, after 75 epochs of training, the success rate of 90% is reached with around 1150 traces. Performing early stopping using $\Delta^1_{train,val}$ allowed for an improvement of 30% of the performances of the network. The time it takes to train the network also went down from 1 hour to 40 minutes which is a 33% decrease in computation time compared to [13]. We can clearly identify the learning phase where the network is underfitting that lasts until epoch 47. At this point the network is at its best capacity given its hyperparameters and training set. Due to its great complexity, the network starts to overfit after the next epoch. Therefore, $\Delta^1_{train,val}$ shows us that there is no benefit in continuing the training after epoch 47. It can also be seen that the value of N_{train} decreases much faster than $\Delta^1_{train,val}$ and its value is very low even at the best capacity of the network. This is a sign of overfitting even though it does not yet impact negatively the performances on validation. Possible solutions to fix this problem would be to change the complexity of the network or to add regularization [5].

4.2 Comparison between GEBVD and $\Delta^1_{train,val}$

In this section, we take a look at the guessing entropy bias variance decomposition as introduced in [15]. The goal of this decomposition is to separate the bias from the variance in the performance of a network. This allows to evaluate the state of the network because a high bias is typically linked to underfitting while a high variance implies some overfitting. GEBVD therefore aims at separating the bias and the variance of the guessing entropy. Van der Valk *et al.* manage to do so by estimating the bias by the mean of the guessing entropy and the variance to be its variance. We try to compare both $\Delta^d_{train,val}$ and the GEBVD regarding the information they offer on the state of a CNN.

For that, we use CNN_{BV} presented in [15] for the tests. A description of the architecture of the network can be find in Appendix A, Table 3. This network uses the mean squared error (MSE) as a loss function, the Adam optimizer with a learning rate of 10^{-4} and the base number of epochs for training is 50. As described in the original article, it has a varying complexity depending on the

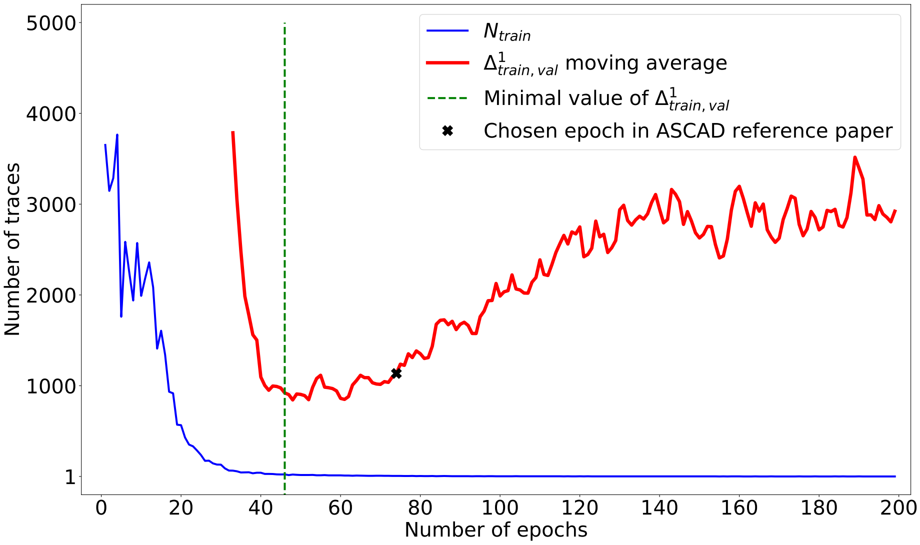


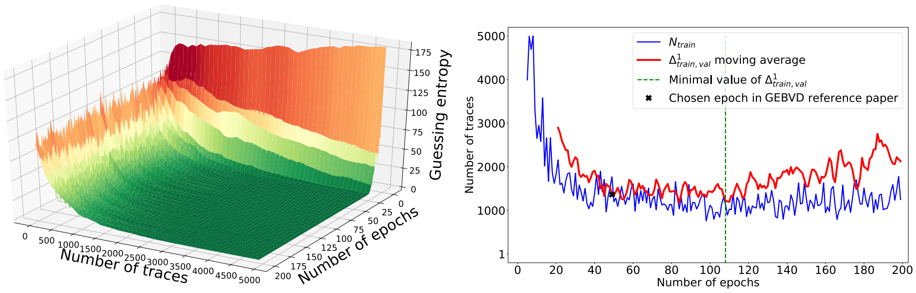
Fig. 4: Evolution of $\Delta^1_{train,val}$ for different number of epochs for CNN_{best} on $Desync_0$, the training set with no desynchronization of the ASCAD database [13], and comparison with their choice of number of epochs

number of convolutional layers used. The networks used to obtain the following results have zero (0CONV) or one (1CONV) convolutional layers. Finally, the leakage model used is the following:

$$Y(\mathbf{k}^*) = HW(Sbox(\mathbf{p}[3] \oplus \mathbf{k}^*[3])),$$

where $\mathbf{p}$ and $\mathbf{k}^*$ are the same as before. This reduces the number of output classes to 9 instead of the previous 256.

Figure 5 shows a comparison of the two metrics on CNN_{BV} with no convolution. On the left, Figure 5a shows the evolution of the guessing entropy at each epoch of the training for different numbers of attack traces while Figure 5b shows the evolution of $\Delta^1_{train,val}$ and N_{train}. From Figure 5a, we see that around epoch 25, the networks is able to consistently reach a success rate of 90% with around 3800 traces which seems to indicate that is it well suited for the problem at hand. On the second picture though, it appears that the learning period lasts until epoch 40. Between epoch 25 and epoch 100, the network is in an unstable regime, meaning there are a lot of unpredictable variations from one epoch to the next, both for N_{train} and N_{val}. This directly impacts the evolution of $\Delta^1_{train,val}$ making it harder to locate the best capacity of the network. Indeed, the best-performing network requires 109 epochs of training and only takes around 2100 traces to reach a success rate of 90%. This is roughly 50% better than the network after 25 epochs and 25% better than the network after 50 epochs, which is the number of epochs recommended in [15], that needs around 3000 traces to obtain the same success rate. This behavior can be attributed to underfitting



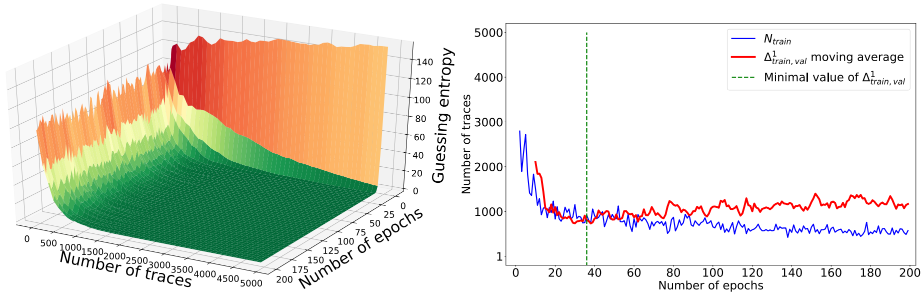
(a) Guessing entropy evolution

(b) Evolution of $\Delta^1_{train,val}$

Fig. 5: Evolution of metrics during training for CNN_{BV} with 0 convolutional block using the MSE loss

due to a lack of complexity of the network and can be an indicator that the model chosen will not converge. It can also be linked to the choice of the loss function and learning rate.

To test this last hypothesis, we trained the same network using the categorical cross-entropy instead of the mean squared error. Figure 6 shows the result of this training. We can see an overall increase in the performance of the network on the validation set as well as a smaller variance in the value of N_{train} for a large number of epochs. With $\Delta^1_{train,val}$, we reach the best capacity for the network at the epoch 37 where the network needs around 1300 traces to reach a success rate of 90%. Then, the value of $\Delta^1_{train,val}$ keeps on increasing until the end of the training. This early stopping brings significant improvements compared to the network trained for 50 epochs, used in [15], that needed around 1800 traces to reach a success rate of 90%. It is also better in terms of training time since we go from 800 seconds for 50 epoch to 618 seconds for 37 epochs. The increase of $\Delta^1_{train,val}$ after the epoch 37 is linked to a stabilization of the performance on validation while the performance of the attacks on training continues to improve. It seems that the low complexity of the CNN reduces the impact of overfitting on the performance on validation. This is further confirmed in Figure 7 representing the evolution of the guessing entropy in Figure 7a and of $\Delta^1_{train,val}$ in Figure 7b for the same architecture but with one layer of convolution. We can see that it tends to overfit much faster by the shape of N_{train} and this tendency has a great impact on the performance on validation which are much worse. The best capacity, given by the minimum value of $\Delta^1_{train,val}$, is reached at the epoch 42 with around 1650 traces needed to reach a success rate of 90%. It is still an improvement compared to the epoch 25 and the epoch 50, used in [15], which require respectively 2700 and 2150 traces. The same metrics were computed for this network using the MSE loss but the architecture in combination with this loss could not consistently reach a success rate of 90% within 5000 traces throughout the training.

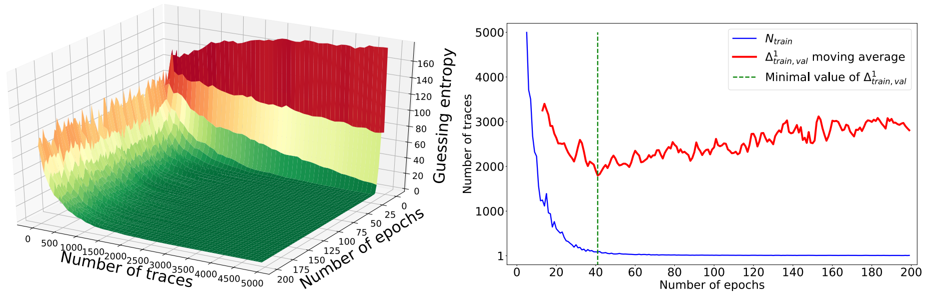


(a) Guessing entropy evolution

(b) Evolution of $\Delta^1_{train,val}$

Fig. 6: Evolution of metrics during training for CNN_{BV} with 0 convolutional block using the CCE loss

On the one hand, low complexity slows down the overfitting but it does not necessarily means the network will perform better in the end. On the other hand, higher complexity allows the network to find more links between the input and output which leads to overfitting if not regulated. This is why regularization techniques are applied to neural networks to prevent this phenomenon from happening.



(a) Guessing entropy evolution

(b) Evolution of $\Delta^1_{train,val}$

Fig. 7: Evolution of metrics during training for CNN_{BV} with 1 convolutional block using the CCE loss

All those conclusions can be deduced using $\Delta^1_{train,val}$ and are harder to see with the GEBVD because once the guessing entropy reaches 1, the variance reaches 0, therefore no information can be extracted from it. When comparing the Figures 6a and 7a, there is hardly any difference between the evolution of the guessing entropies. This contrasts with the evolution of $\Delta^1_{train,val}$ between the Figures 6b and 7b in which we can see a difference in the quality of the

Table 1: Summary of the results with choices made using $\Delta^1_{train,val}$ in **bold**

Networks	Nb of CONV layers	LOSS	Architecture reference	Nb epochs	N_{val}	$\Delta^1_{train,val}$	Time to train (seconds)	Difference in time and performance in comparison to the reference
CNN best	5	CCE	[13]	75	1151	1145	3600	Time: -33.3%
			This article	**47**	**802**	**779**	**2400**	N_{val}: -30.3%
CNN BV	0	MSE	[15]	50	2960	1449	970	Time: +98.5%
			This article	**109**	**2093**	**954**	**1926**	N_{val}: -29.3%
		CCE	[15]	50	1849	915	800	Time: -22.7%
			This article	**37**	**1331**	**413**	**618**	N_{val}: -28.6%
	1	CCE	[15]	50	2177	2136	956	Time: -20.4%
			This article	**42**	**1659**	**1575**	**761**	N_{val}: -23.7%

training as well as the performance of the networks. For the network without convolutional layers, the value of N_{train} and N_{val} are much closer which leads to a lower value of $\Delta^1_{train,val}$ and indicates a better learning phase with less overfitting. In addition, since $\Delta^1_{train,val}$ is based on the number of traces needed to reach a SR^1 of 90%, we are still able to evaluate and compare networks when all the attacks are successful, *i.e.* when the SR^1 reaches 100%. Indeed, GEBVD cannot be used when the variance is null because it gives no information on the state of the network. With $\Delta^1_{train,val}$, we consistently find the best training epoch and therefore perform early stopping which improves the capacity and the training time of the network. The results detailed in this section are summarized in Table 1, giving a comparison between the networks as mentioned in their respective article and the choices made using $\Delta^1_{train,val}$. Those results seem to confirm, as argued in [11] by Masure *et al.*, that the categorical cross-entropy is an appropriate loss to use in deep learning for side-channel analysis.

5 Conclusion

In this article, we introduced a new metric dedicated to deep learning for side-channel analysis. By comparing the efficiency of the attacks on the training set and on the validation set, this metric evaluates at the same time the performance of a given architecture and its potential for improvement. It also allows for a characterization of the state of the network and therefore for the detection of overfitting. This property makes it possible to use this metric during the training of a network to perform early stopping.

Consequently, we used $\Delta^d_{train,val}$ to evaluate the best CNN of the ASCAD public database. We found out by applying $\Delta^1_{train,val}$ for early stopping that it reaches its best performance around 47 epochs which is less than the 75 epochs mentioned in [13]. This early stopping of the training allowed for a reduction of 31% of the number of traces needed to reach a success rate of 90% and reduced the training time by 30%. The measurement of $\Delta^1_{train,val}$ showed that it was heavily overfitting the training data. To limit this overfitting, we suggest the use of normalization and regularization. We then compared $\Delta^1_{train,val}$ to the GEBVD as introduced in [15] by evaluating an architecture presented in the article. It showed that $\Delta^1_{train,val}$ gives a better insight on how well the network is performing as well as how much it is overfitting the training data, especially when the attacks are successful. We managed to improve the performance of the network by 20 to 30% while the training time was reduce by more or less the same percentage. This shows the importance of choosing the right number of epochs.

For future work, we plan to use $\Delta^d_{train,val}$ to evaluate the improvements that can bring normalization and regularization techniques. Finally, it can be interesting to adapt this metric (or a derivative) as a loss function in order to optimize the network in the side-channel context.

References

1. Agrawal, D., Archambeault, B., Rao, J.R., Rohatgi, P.: The EM side—channel(s). In: Kaliski, B.S., Koç, ç.K., Paar, C. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2002. pp. 29–45. Springer Berlin Heidelberg, Berlin, Heidelberg (2003)
2. Bartkewitz, T., Lemke-Rust, K.: Efficient template attacks based on probabilistic multi-class support vector machines. In: Mangard, S. (ed.) Smart Card Research and Advanced Applications - 11th International Conference, CARDIS 2012, Graz, Austria, November 28-30, 2012, Revised Selected Papers. Lecture Notes in Computer Science, vol. 7771, pp. 263–276. Springer (2012). https://doi.org/10.1007/978-3-642-37288-9_18
3. Cagli, E., Dumas, C., Prouff, E.: Convolutional neural networks with data augmentation against jitter-based countermeasures - profiling attacks without preprocessing. In: Fischer, W., Homma, N. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings. Lecture Notes in Computer Science, vol. 10529, pp. 45–68. Springer (2017). https://doi.org/10.1007/978-3-319-66787-4_3
4. Chari, S., Rao, J.R., Rohatgi, P.: Template attacks. In: Revised Papers from the 4th International Workshop on Cryptographic Hardware and Embedded Systems. pp. 13–28. CHES '02, Springer-Verlag, London, UK, UK (2003), `http://dl.acm.org/citation.cfm?id=648255.752740`
5. Goodfellow, I.J., Bengio, Y., Courville, A.C.: Deep Learning. Adaptive computation and machine learning, MIT Press (2016), `http://www.deeplearningbook.org/`
6. Hospodar, G., Gierlichs, B., Mulder, E.D., Verbauwhede, I., Vandewalle, J.: Machine learning in side-channel analysis: a first study. J. Cryptographic Engineering **1**(4), 293–302 (2011). https://doi.org/10.1007/s13389-011-0023-x
7. Kim, J., Picek, S., Heuser, A., Bhasin, S., Hanjalic, A.: Make some noise. unleashing the power of convolutional neural networks for profiled side-channel analysis. IACR Transactions on Cryptographic Hardware and Embedded Systems **2019**(3), 148–179 (May 2019). https://doi.org/10.13154/tches.v2019.i3.148-179

8. Kocher, P.C., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M.J. (ed.) Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings. Lecture Notes in Computer Science, vol. 1666, pp. 388–397. Springer (1999). https://doi.org/10.1007/3-540-48405-1_25
9. Lerman, L., Poussier, R., Markowitch, O., Standaert, F.: Template attacks versus machine learning revisited and the curse of dimensionality in side-channel analysis: extended version. J. Cryptographic Engineering **8**(4), 301–313 (2018). https://doi.org/10.1007/s13389-017-0162-9
10. Maghrebi, H., Portigliatti, T., Prouff, E.: Breaking cryptographic implementations using deep learning techniques. In: Carlet, C., Hasan, M.A., Saraswat, V. (eds.) Security, Privacy, and Applied Cryptography Engineering - 6th International Conference, SPACE 2016, Hyderabad, India, December 14-18, 2016, Proceedings. Lecture Notes in Computer Science, vol. 10076, pp. 3–26. Springer (2016). https://doi.org/10.1007/978-3-319-49445-6_1
11. Masure, L., Dumas, C., Prouff, E.: A comprehensive study of deep learning for side-channel analysis. IACR Transactions on Cryptographic Hardware and Embedded Systems **2020**(1), 348–375 (Nov 2019). https://doi.org/10.13154/tches.v2020.i1.348-375
12. Picek, S., Heuser, A., Jovic, A., Bhasin, S., Regazzoni, F.: The curse of class imbalance and conflicting metrics with machine learning for side-channel evaluations. IACR Transactions on Cryptographic Hardware and Embedded Systems **2019**(1), 209–237 (Nov 2018). https://doi.org/10.13154/tches.v2019.i1.209-237
13. Prouff, E., Strullu, R., Benadjila, R., Cagli, E., Dumas, C.: Study of deep learning techniques for side-channel analysis and introduction to ASCAD database. IACR Cryptology ePrint Archive **2018**, 53 (2018), `http://eprint.iacr.org/2018/053`
14. Standaert, F., Malkin, T., Yung, M.: A unified framework for the analysis of side-channel key recovery attacks. In: Advances in Cryptology - EUROCRYPT 2009, 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cologne, Germany, April 26-30, 2009. Proceedings. pp. 443–461 (2009). https://doi.org/10.1007/978-3-642-01001-9_26
15. van der Valk, D., Picek, S.: Bias-variance decomposition in machine learning-based side-channel analysis. Cryptology ePrint Archive, Report 2019/570 (2019), `https://eprint.iacr.org/2019/570`
16. Zaid, G., Bossuet, L., Habrard, A., Venelli, A.: Methodology for efficient cnn architectures in profiling attacks. IACR Transactions on Cryptographic Hardware and Embedded Systems **2020**(1), 1–36 (Nov 2019). https://doi.org/10.13154/tches.v2020.i1.1-36
17. Zhou, Y., Standaert, F.X.: Deep learning mitigates but does not annihilate the need of aligned traces and a generalized resnet model for side-channel attacks. Journal of Cryptographic Engineering (04 2019). https://doi.org/10.1007/s13389-019-00209-3

A Networks

Table 2: **Network hyperparameters for** CNN_{best} [13]

Layer type	Hyperparameters
Trace input	700

Convolution 1D	Filter = 64,
	Filter length = 11,
	Activation = ReLU

Average Pooling	Pool length = 2

Convolution 1D	Filter = 128,
	Filter length = 11,
	Activation = ReLU

Average Pooling	Pool length = 2

Convolution 1D	Filter = 256,
	Filter length = 11,
	Activation = ReLU

Average Pooling	Pool length = 2

Convolution 1D	Filter = 512,
	Filter length = 11,
	Activation = ReLU

Average Pooling	Pool length = 2

Convolution 1D	Filter = 512,
	Filter length = 11,
	Activation = ReLU

Average Pooling	Pool length = 2
Flatten	-
Fully-connected	Neurons = 4096
Fully-connected	Neurons = 4096
Output	Softmax: 256 classes

Table 3: **Network hyperparameters for** CNN_{BV} [15]

Layer type	Hyperparameters
Trace input	700

Convolution 1D	Filter = 8,
	Filter length = 3,
	Activation = ReLU

Batch Normalization	-
Max Pooling (Optional)	Pool length = 2

Convolution 1D	Filter = 16,
	Filter length = 3,
	Activation = ReLU

Batch Normalization	-
Max Pooling (Optional)	Pool length = 2

Convolution 1D	Filter = 32,
	Filter length = 3,
	Activation = ReLU

Batch Normalization	-
Max Pooling (Optional)	Pool length = 2

Convolution 1D	Filter = 64,
	Filter length = 3,
	Activation = ReLU

Batch Normalization	-
Max Pooling (Optional)	Pool length = 2

Convolution 1D	Filter = 64,
	Filter length = 3,
	Activation = ReLU

Batch Normalization	-
Max Pooling (Optional)	Pool length = 2
Flatten	-
Dropout	Coefficient = 0.5
Fully-connected	Neurons = 512
Dropout	Coefficient = 0.5
Output	Softmax: 9 or 256 classes