# Learning when to stop: a mutual information approach to prevent overfitting in profiled side-channel analysis

Guilherme Perin[1,2], Ileana Buhan[1] and Stjepan Picek[2]

[1] Riscure BV, The Netherlands
[2] Delft University of Technology, The Netherlands

**Abstract.** Today, deep neural networks are a common choice for conducting the profiled side-channel analysis. Such techniques commonly do not require pre-processing, and yet, they can break targets protected with countermeasures. Unfortunately, it is not trivial to find neural network hyper-parameters that would result in such top-performing attacks. The hyper-parameter leading the training process is the number of epochs during which the training happens. If the training is too short, the network does not reach its full capacity, while if the training is too long, the network overfits, and is not able to generalize to unseen examples. Finding the right moment to stop the training process is particularly difficult for side-channel analysis as there are no clear connections between machine learning and side-channel metrics that govern the training and attack phases, respectively.

In this paper, we tackle the problem of determining the correct epoch to stop the training in deep learning-based side-channel analysis. We explore how information is propagated through the hidden layers of a neural network, which allows us to monitor how training is evolving. We demonstrate that the amount of information, or, more precisely, mutual information transferred to the output layer, can be measured and used as a reference metric to determine the epoch at which the network offers optimal generalization. To validate the proposed methodology, we provide extensive experimental results that confirm the effectiveness of our metric for avoiding overfitting in the profiled side-channel analysis.

**Keywords:** Side-channel Analysis · Neural Networks · Overfitting · Mutual Information · Information Bottleneck

## 1 Introduction

Profiled side-channel attacks (SCAs) determine the worst-case security bounds of protected cryptographic implementations. The attack model assumes an adversary with full control of a device identical to the target one. Various techniques like template attacks [CRR02], linear regression [SLP05], and machine learning [LMBM13, MZVT15] belong to the class of profiled SCAs and can achieve strong performance. Recently, deep learning techniques proved to be even more potent as 1) they do not need to use the pre-processing phase to select the points of interest, and 2) they perform well even in the presence of noise and countermeasures [CDP17, KPH$^+$19].

Although the application of deep learning for profiled SCA became popular, there are many open questions like the selection of hyper-parameters for successful side-channel attacks. For hyper-parameters like architectural details (e.g., number of layers, neurons), recent works provide certain directions to follow [ZBHV19]. At the same time, the ability to select the right moment to stop the training phase is left to instinct (and sometimes luck) as there does not seem to be a direct connection between the machine learning metrics and the performance of a side-channel attack [PHJ+19].

While the lack of a clear connection may not be deemed crucial, the end of the training phase depends on metrics like accuracy, loss, or recall, and their performance on a validation set. If the training phase finishes too late, the machine learning model overfits. As a consequence of overfitting, the model will not generalize to unseen data, and the attack phase will fail. [1]

One way to analyze the generalization of a deep neural network is through the lens of information theory. In [ST17], the authors propose a new methodology to interpret the training of a multilayer perceptron (MLP) through a theory called the *Information Bottleneck* (IB) [TZ15]. They demonstrate that the training of an MLP provides two distinct phases - *fitting* and *compression*. These phases are determined by computing the mutual information between the intermediate representations (activations from hidden layers), and input (raw data) and output (labels). The output of a hidden layer can be seen as a summary of statistics containing information about the input and output. The fitting phase is usually very fast, requiring only a few epochs, while the compression phase lasts longer. The compression phase is also the one responsible for the generalization of the neural network, i.e., its ability to perform on unseen data. We consider the mutual information between output layer activation's and the data labels (as given by the leakage model) as a metric to identify the epoch at which the neural network achieves its optimal generalization capacity. Our results show that training a network for too many epochs has a negative effect on generalization, and early stopping based on the mutual information metric is a reliable technique to avoid this scenario. We test our metric against three masked AES implementations and show that compared to the typical metrics like accuracy, recall, or loss our metric provides superior results.

To the best of our knowledge, this is the first result providing a reliable attack performance metric different from conducting an actual attack (key ranking). While key ranking is a reliable validation metric to optimize the generalization of a deep neural network for side-channel attacks [CDP17], it brings significant computational overhead when using large validation sets. Mutual information, on the other hand, offers remarkable performance at a fraction of the computational cost as it does not have to be computed for all key hypotheses. To facilitate reproducible research, we make the source code publicly available [Ano20].

## 1.1 Related Works

In this section, we review the known results related to the problem of optimizing the training of a neural network and information theory for deep learning in the context of side-channel analysis. From the first paper considering convolutional neural networks for SCA [MPP16], deep learning gained a significant recognition in the SCA community as the paradigm to follow for profiled SCA. Despite good results, even when considering protected targets [CDP17, KPH+19], there are open questions. For instance, progress on topics like interpretability and explainability of neural networks is difficult in general, and as such, there are not many results for SCA also [vdVPB19, vdVP19, MDP19].

A second important research direction explored by the SCA community is how to tune hyper-parameters [KPH+19, ZBHV19]. Common options for hyper-parameter exploration are number/types of layers/neurons, activation functions, and the number of epochs.

Determining the number of training epochs is relevant for any application domain, but is a particularly challenging problem in SCA [PHJ+19]. Common options are the use of early stopping or a predetermined number of epochs for training [PSB+18, HGG19,

[1] It is also possible for a machine learning model to underfit if the training stopped too early. Still, this is usually of less concern as the resulting machine learning model would generalize to unseen data but just not use its full potential, i.e., the attack would not be as powerful as possible.

KPH$^{+}$19, ZBHV19]. In both cases, as the performance is observed through machine learning metrics, it is difficult to know if the training stopped at the right moment.

To circumvent this problem, Robissout et al. [RZC$^{+}$20] conducted a success rate evaluation for every epoch of training and validation. This solution leads to a significant time overhead for large datasets. More importantly, there is no consensus on how to calculate the key rank for training sets that have random keys, as best practice recommends. Finally, the work proposed in [MDP20] states that maximizing the perceived information in a profiled side-channel attack is equivalent to minimizing the loss functions based on negative log-likelihood or cross-entropy.

## 1.2 Contributions

This work provides two main contributions, as follows:

1. A new metric to select the epoch where a deep network achieves its best performance for profiled side-channel analysis. We show that the information transferred to the output layer about the labels can be measured and used as a reliable metric to determine when to stop the training phase. The new metric offers four distinct improvements compared to existing results. First, the new metric is precise and can accurately predict the epoch at which the network achieves its best performance, while the alternative, validation key ranking, will give a range of values for the best epoch. Second, the computational overhead for computing the information transferred to the output layer is much smaller compared to performing a full attack to obtain the key ranking at the end of each epoch. Third, the new metric is suitable to use for training sets with randomized key, as recommended by best practices, while there is no consensus on how to calculate the key ranking for this type of training sets, as far as the authors are aware. Forth, we extend [ST17], where the authors show how to calculate the information path for MLP architectures to CNN architectures.
2. The new metric consistently offers good performance. We test the new metric on three publicly available datasets, using two leakage models, and two different architectures. In the experimental section, we thoroughly compare its performance to the four conventional metrics: validation loss, validation accuracy, validation recall, and validation key rank and conclude that in all cases the use of the new metric will lead to better generalization.

The rest of this paper is organized as follows. Section 2 provides background information about the profiled side-channel analysis, deep learning, and information theory. The theory of information bottleneck and information plane for deep learning is provided in Section 3, where we describe the main contribution of this work. Section 4 provides empirical validation of the proposed analysis based on information theory to increase the performance of deep learning-based side-channel attacks. Finally, conclusions and possible future research directions are given in Section 5.

# 2 Background

This section provides information about profiled side-channel attacks and deep learning-based SCA. Afterward, the basic concepts of information theory are defined.

## 2.1 Profiled Side-channel Attacks

Side-channel analysis (SCA) can be divided into non-profiled and profiled side-channel analysis. Standard non-profiled side-channel attacks like correlation power analysis [BCO04] or differential power analysis [KJJ99] explore the leakage from univariate statistics. These attacks are usually limited by the selection of a function defining the leakage model. They are also defined as the first-order side-channel attacks. If more than one variable needs to be explored from the non-profiled scenario, higher-order attacks can combine several samples, and the analysis usually requires more complex attacks.

Profiled attacks assume that the adversary has an open cryptographic device, which can be used to learn a model by freely setting the key and the input data (plaintext or ciphertext). These attacks also explore the multivariate nature of the leakage and usually require features/points of interest selection for dimensionality reduction. The learned model is tested against side-channel traces collected from an identical target device. This second part is known as the matching or attack phase, and it returns the key hypothesis with the maximum likelihood estimation. Standard profiled attacks are template attacks [CRR02], linear regression [SLP05], and machine learning [LMBM13, MHM13].

The adoption of deep neural networks as a profiled attack paradigm can provide automated points of interest selection where recent results suggest such techniques are less sensitive to trace misalignment and countermeasures based on the first order-masking for software AES implementations [CDP17, KPH$^+$19, PSB$^+$18].

A usual approach to assess the attacker's performance is to use metrics that denote the number of measurements required to obtain the secret key $k^*$. Common examples of such metrics are guessing entropy (GE) and success rate (SR) [SMY09]. *Guessing entropy* represents the average number of key candidates an adversary needs to test to determine the correct key, denoted with $k^*$, after conducting a side-channel analysis attack. More specifically, given $N$ amount of traces in the attacking phase, an attack outputs a vector $g = [g_1, g_2, ..., g_{|K|}]$ in decreasing order of probability. The guessing entropy is the average position of $k^*$ in $g$ over several experiments (commonly 50 or 100). The *success rate* is defined as the average empirical probability that $g_1$ equals the secret key $k^*$.

## 2.2 Deep Learning in the Context of Side-Channel Analysis

A deep neural network can be seen as a system that accepts as input a random variable $X$ (a side-channel trace) and provides as output a probability estimation $\hat{Y}$, based on the class label $Y$ estimated by a decision rule. The size of the input vector $X$ is the same as the number of samples/features in the side-channel trace. The size of the vector $Y$ is directly derived from the leakage model $l$ selected according to the target cryptographic function (e.g., S-box output in the first encryption or decryption AES round). In total, there are $M$ traces that are used in the training phase.

In the profiled SCA, the trained neural network is then tested against a set $N$ of side-channel traces, in which the secret key is unknown, and the key recovery methodology assumes that the correct key is the one that maximizes the summation probabilities $S_k$ for each key byte candidate:

$$S_k = \sum_{i=1}^{N} (p_{i,j}). \tag{1}$$

The value (i.e., the probability) $p_{i,j}$ is an element of a matrix $P$ with size *number of traces* $\times$ *number of classes.* This matrix is the output class probabilities obtained by predicting the trained model with a test or validation set. Thus, $p_{i,j}$ is the probability element obtained as a function of the attack trace $x_i$, leakage model $l$, and input data $p_{k_i}$ for every possible key guess $k$: $f_k(x_i, p_{k_i}, l)$. In the context of neural networks, $p_{i,j}$ represents the neuron's activation value for trace $i$ from the *Softmax* output layer. The process of training a deep neural network is an optimization problem stemming from the minimization of the selected loss function based on the stochastic gradient descent algorithm. For side-channel analysis, we are interested in a machine learning model that is capable of compressing the input relevant information from $X$ (i.e., a side-channel trace), while discarding the irrelevant information as noise or irrelevant samples.

## 2.3 On the Generalization Ability of Neural Networks

For a deep learning-based SCA to be successful, the trained model has to generalize well, which calls for a minimal error on the test set. Next, we define the generalization interval in SCA for neural networks.

**Definition 1.** Generalization Interval in SCA. Given an ensemble $(X_{train}, Y_{train})$ where $X_{train}$ represents the input training data, and $Y_{train}$ represents a set of training labels, the generalization interval defines the epochs where a successful key recovery can be obtained using Eq. (1).

Techniques used to reduce test errors are commonly known as regularization techniques, among which *early-stopping* is one of the best know techniques [GBC16]. Early stopping works under the assumption that the neural network achieved the best generalization and will start overfitting and deteriorate the generalization after this point (of best generalization), which is an undesired behavior. Consequently, determining the pre-specified number of iterations (epochs) is a hyper-parameter selection process.

For side-channel analysis, machine learning metrics have demonstrated to be inconsistent as reference for the validation process [PHJ+19]. As such, one can infer that implementing early stopping based on, e.g., loss function or accuracy could lead to conflicting results.

## 2.4 Datasets

We consider three publicly available datasets, instances of software AES implementations protected with the first-order Boolean masking. The first one is the widely used ASCAD database in the side-channel community for deep learning research [PSB+18]. The traces were measured from an implementation consisting of a software AES implementation running on an 8-bit microcontroller. The AES is protected with the first-order masking, where the two first key bytes (index 1 and 2) are not protected with masking (e.g., masks are set to zeros) and the key bytes 3 to 16 are masked. A trimmed trace set corresponding to the processing of key byte 3 (S-box operation in the first encryption round) is provided by the database. We consider the trace set containing 200 000 AES-128 encryption traces where the plaintext and key are **randomly** defined for each separate encryption. This trace set is used as a training and validation set. A second fixed-key trace set, consisting of 1 000 measurements is split into validation and test sets, having 500 traces each one. In this dataset, each trace contains 1 400 features. This dataset is available at https://github.com/ANSSI-FR/ASCAD.

The second dataset is the DPA Contest v4 (DPAv4) database [TEL14]. DPAv4 database provides trace sets collected from an AES-256 RSM (rotate shift masking) implementation. The training set consists of 34 000 traces with a **fixed key**. The test and validation sets contain 2 000 traces each. For convenience, we attack only the first key byte of an AES-256 implementation. Each trace consists of 2 000 features. This dataset is available at http://www.dpacontest.org/v4/.

The third dataset refers to the CHES Capture-the-flag (CTF) masked AES-128 encryption trace set, released in 2018 for the Conference on Cryptographic Hardware and Embedded Systems (CHES). In our experiments, we consider 43 000 traces for the training set with a **fixed key**. The validation and test set consist of 1 000 traces each. The keys for the training set are different from the key configured for the validation and test sets. Each trace consists of 2 200 features. This dataset is available at https://chesctf.riscure.com/2018/news.

## 2.5 Information Theory

In information theory, the (marginal) entropy $H(X)$ of a random variable $X$ is defined as the average information obtained by observing $X$, and it can be quantitatively defined as:

$$H(X) = -\sum_{x\in X} p(x)\log_2 p(x), \tag{2}$$

where $p(x)$ represents the probability of variable $X$ taking value $x$. The *conditional entropy* of $X$ given $Y$ is defined as follows:

$$H(X|Y) = -\sum_{x\in X} p(x)\sum_{y\in Y} p(x|y)\log_2 p(x|y). \tag{3}$$

This can be considered as the entropy of $X$ when knowing $Y$. Finally, the *mutual information* defines the dependence between variables $X$ and $Y$, and it can be defined using entropy and conditional entropy values as follows:

$$I(X;Y) = H(X) - H(X|Y). \tag{4}$$

An important property of mutual information in this context is the Data Processing Inequality (DPI), which states that for any three variables $X, Y, Z$, which form a Markov chain, $X \rightarrow Y \rightarrow Z$, the mutual information between the variables can only decrease and $I(X;Y) \geq I(X;Z)$.

# 3 Information Theory of Deep Neural Networks

Shwartz-Ziv and Tishby [ST17] showed that information theory could be used to visualize the training phase of a deep network to compare the performance of different network architectures. Intuitively, when training a network, each layer is getting its information from the layer before and transforming it by using matrix multiplication of nonlinear functions. Their insight was to treat each layer (the hidden activation functions) in the deep network as a random variable fully described by the information captured about the input data and the labels. Modeling each layer in the deep network as a random variable gives an alternative view of a deep network as a Markov chain. Each variable represents the nonlinear activation function, which successively transforms the input data into the label space. Using the mutual information between the layers, the input data, and the labels, we can visualize the transformation of the input data into the label space.

**Definition 2.** Information Path. Given en ensemble $(X, Y, T_i)$ where $X$ represents the input data, $Y$ represents a set of labels and $T_i$ is a hidden layer in an $n$-layered network, described as $X, Y \rightarrow T_1 \rightarrow ... \rightarrow T_n$, the information path is defined as the set of points $\{[I(X;T_i), I(T_i;Y)] | i \in \{1, n\}\}$.

The *information path* is a record of the information each hidden layer preserves about the input data $X$ and the output variables $Y$. It is typically computed for each epoch during the training phase. The information is plotted in a two-dimensional coordinate system referred to as the *information plane*. The coordinates of the information plane quantify the bits of information layer $T_i$ has about the input data $X$ as $I(X;T_i)$, and the bits of information layer $T_i$ has about the labels $Y$ as $I(T_i;Y)$. We can view the variable $T_i$ as a compressed representation of the input $X$, and $I(X;T_i)$ calculated based on the value $p(x)p(t_i|x)$, which measures how compact the representation of $X$ is. The maximum value for $I(X;T_i)$ is $H(X)$, which is the Shannon entropy that corresponds to the case where $T_i$ copies $X$ and there is no compression. The minimal value for $I(T_i;X)$ is 0 and corresponds to the case where $T$ has one value.

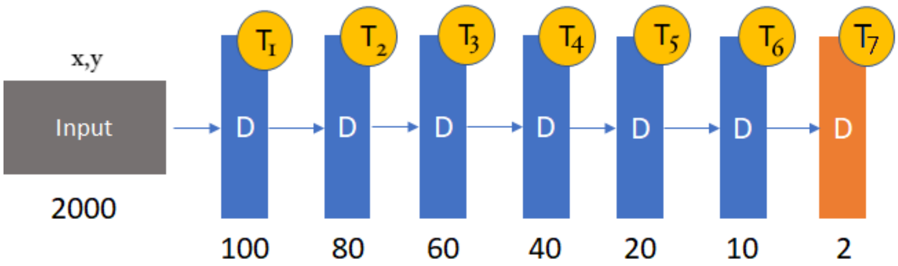


**Figure 1:** MLP with five hidden layers ($T_{2:6}$). The letter "D" denotes the dense (fully-connected) layer while the labels $T_1$ and $T_7$ corresponds input and output layers, respectively, with the *Tanh* activation function. The output layer $T_7$ has has *Softmax* activation function. The numbers under the layers indicate the amount of neurons.

**Lemma 1.** *Information Path Uniqueness. For each ensemble* $(X, Y, T_i)$*, where* $X$ *represents the input data,* $Y$ *represents a set of labels, and* $T_i$ *is a layer in an* $n$*-layered network described as* $X, Y \rightarrow T_1 \rightarrow ... \rightarrow T_n$ *there exists a unique information path that satisfies the following two inequalities:*

$$H(X) \geq I(X; T_1) \geq ... \geq I(X; T_n) \geq I(X; \hat{Y}), \tag{5}$$

*where* $\hat{Y}$ *are the labels predicted by the network, and*

$$I(X; Y) \geq I(T_1; Y) \geq \ldots \geq I(T_n; Y) \geq I(\hat{Y}; Y). \tag{6}$$

*Proof.* The proof for this lemma follows immediately by applying the DPI principle. □

### 3.1 The Information Bottleneck Principle

Shwartz-Ziv and Tishby observed that stochastic gradient descent (SGD) optimization defines two distinct phases during training [ST17]. The first one is the *fitting* phase, where both $I(X; T_i)$ and $I(T_i; Y)$ increase fast as the training progresses. During the fitting phase, the deep network layers increase the amount of information about the input data and the labels. The second phase is the *compression* phase, where the network starts to compress or forget information about the input data and slowly increases its generalization capacity by retaining more information about the labels.

The behavior of the network during the compression phase has been linked to the form of the activation functions [CHO19]. This happens due to a random diffusion-like behavior of the SGD algorithm if double-sided saturating[2] nonlinear activation function such as *Tanh* is employed. More precisely, Shwartz-Ziv and Tishby provided results for *Tanh* and show how information about the labels increases in the compression phase [ST17]. On the other hand, Saxe et al. demonstrated that the non-saturating activation functions like *ReLU* provide a different behavior in the compression phase as there is no causal connection between generalization and compression [SBD+18].

Figure 2 gives an overview of the information path of the deep network architecture described in Figure 1 at epochs: 1, 20, 100, and 200 during the training process. Each figure contains the coordinates of the information $[I(X; T_i), I(T_i; Y)]$ where $i$ represents the i-th layer. The information is captured from the five hidden layers ($T_{2:6}$) plus an input layer ($T_1$) and an output layer ($T_7$).

For the above example, we see that information changes only for the last two hidden layers $T_5$ and $T_6$, and the output layer $T_7$. The plot contains mutual information

[2] A saturating activation function squeezes the input data, i.e., the output is bounded to a certain range.

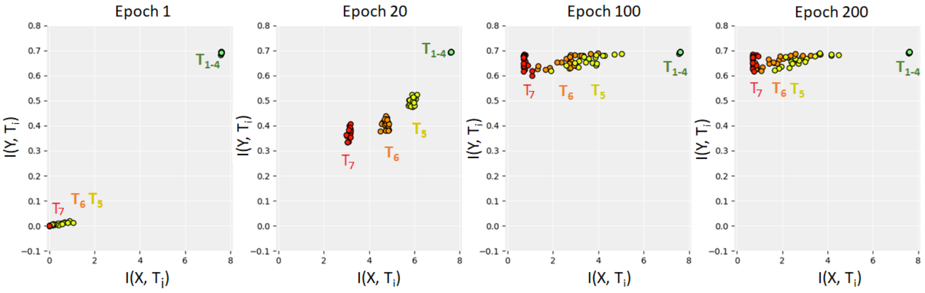


**Figure 2:** The information flow captured at epoch 1, 20, 100, and 200 for the network architecture depicted in Figure 1 when using the DPAv4 dataset (training set).

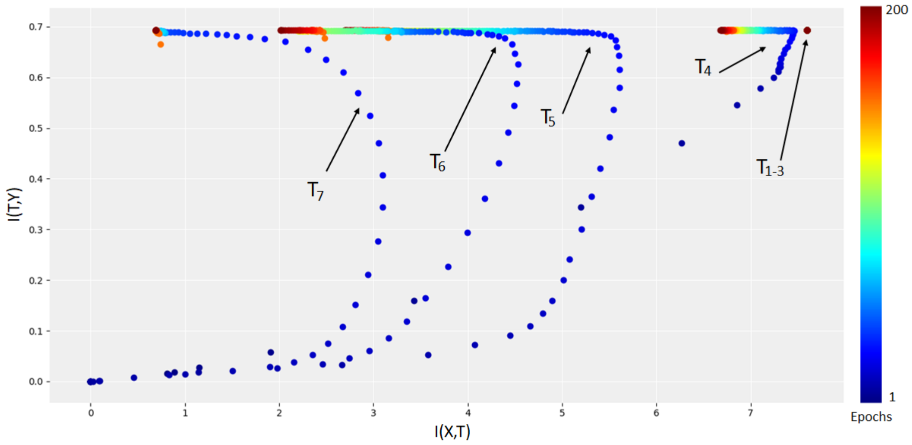


**Figure 3:** Information plane from the DPAv4 dataset (training set).

results for twenty training experiments (i.e., twenty dots for each layer). These results demonstrate that at the beginning of the training phase, the mutual information quantities $[I(X;T_i), I(T_i;Y)]$ are at a minimum level for hidden layers $T_5$ and $T_6$, and for the output layer $T_7$. As the training progress (epochs 20 and 100), the mutual information values increase until the $[I(X;T_i), I(T_i;Y)]$ reaches its maximum for all layers, including the output layer. If we continue the training process, the compression phase starts to happen as $I(X;T_i)$ starts to decrease and $I(T_i;Y)$ stays at a maximum level. In Figure 2, this information path is clearly observed for hidden layers $T_5$ and $T_6$, and the output layer $T_7$.

In Figure 3, we show the evolution of the information path for all training epochs for the DPAv4 dataset, for the same architecture given in Figure 1. The training evolution provides two distinct phases (fitting and compression), as discussed in [ST17]. In the first phase, the layers (mostly visible for hidden layers $T_{4:6}$ and output layer $T_7$) are fitting the training data. The information of an inner state $T_i$ or layer increases for the input $X$ and output $Y$[3]. In the second phase, the information about the output stays high, but the information about the input starts to decrease. From the figure, it is clear that the second phase starts before epoch 100.

[3] Note, information plane figures show different layers, but it is not possible to recognize a specific layer by just "observing" the graph, i.e., there is no pre-specified behavior for a specific layer. We store and plot data for each layer separately.

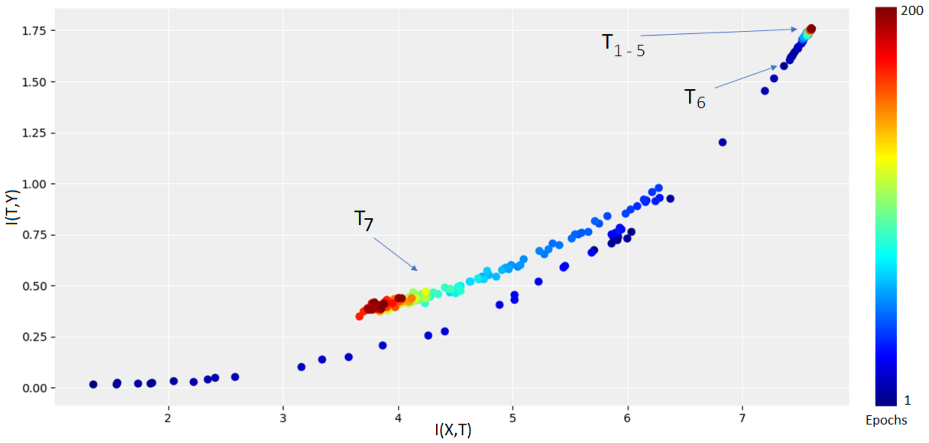


**Figure 4:** Information plane from the DPAv4 dataset (validation set).

## 3.2 The Information Path for Side-channel Analysis Data

To asses whether a machine learning model generalizes well, commonly, we check its performance on previously unseen data, i.e., validation set as defined in Definition 1. Similarly, to investigate the generalization from the information path, we must asses its behavior on the validation set. More precisely, we aim to find the generalization interval as defined in Definition 3.

**Definition 3.** SCA Generalization Interval via Information Path.
Given an ensemble $(X_{train}, Y_{train}, T_i)$ where $X_{train}$ represents the input training data, $Y_{train}$ represents a set of training labels, and $T_i$ is a hidden layer in an $n$-layered network, the generalization interval for SCA defines the interval of training epochs where the quantities $[I(X;T_i), I(T_i;Y)]$ reach the maximal values and we can obtain successful key recovery with Eq. (1) by predicting on the dataset $X_{test}$ with the trained neural network.

The results in Figure 4 show that it is possible to observe a different "movement" of the points $[I(X;T_i), I(T_i;Y)]$ in the information path when using the validation set. The fitting phase is clearly seen, as $I(X;T_i)$ and $I(T_i;Y)$ increase with the processing of first epochs (for the validation set, this movement is observable for hidden layer $T_6$ and output layer $T_7$).

The compression phase is different from the information plane observed in Figure 3. For the validation set, the points $[I(X;T_i), I(T_i;Y)]$ reach the maximum value for each hidden layer, and, later, both quantities decrease with the processing of more epochs. This indicates the overfitting scenario for the given trained machine learning model. More specifically, this happens because the generalization in difficult side-channel analysis problems (i.e., masked or protected AES) is minimal when given in terms of deep learning metrics (accuracy, loss, recall). At the same time, we aim to capture the machine learning model at an epoch when the best possible generalization occurs. From our observations, the epoch at which the generalization is optimal is given by the moment when $I(T_n;Y)$ reaches a maximum value.

## 3.3 Improving the Generalization in Deep Learning-based SCA

Recall, for a deep learning-based side-channel analysis to be successful, the trained model must generalize well to previously unseen data (validation/test set). Given a deep neural network defined by a set of hyper-parameters $\theta$, the internal representations $T_i$, $i \in 1, n$, (where $T_1$ and $T_n$ are the input and output layers, respectively) should inform about the labels $Y$ and input $X$ [AG18].

As observed from Figures 3 and 4, we stop the training when we reach the maximum value for $I(T_n; Y)$. As such, we assume:

1. During the training, the intermediate representation $T_i$ will be compressed to estimate $Y$ correctly.
2. The intermediate representation $T_i$ should be robust such that small addition of noise should not affect this compressed internal representation.
3. Only the information transferred to the output network layer is important for measuring the generalization [CHO19].

Our investigation suggests that the maximum value for $I(T_n; Y)$ for the output layer happens when the fitting phase is finished, and the compression started. This means that the training does not need to go through the full compression phase to achieve the best generalization. This is also aligned with findings from Shwartz-Ziv and Tishby as they observe that the beginning of the compression phase coincides with the best generalization [ST17], and Saxe et al. who show empirical results demonstrating that the compression phase does not necessarily improve generalization [SBD+18].

Note that the calculation of $I(T_n; Y)$ gives minimal overheads during the training process since we need to make the computation for a small fraction of the validation set. We estimate that the time overhead to compute $I(T_n; Y)$ at the end of each epoch equals less than 2%.

## 4 Experimental Validation

In this section, we empirically show that the mutual information between the output layer $T_n$ and the desired output $Y$, $I(T_n; Y)$ provides a consistent metric to determine an optimal number of epochs for the training phase. In our experiments, we consider both the Hamming weight leakage model and the identity value leakage model.

### 4.1 Estimating Mutual Information

The first step of calculating mutual information, Eq. (4), is *density estimation* [KSG04], which aims to construct an approximation of the density function (denoted with $p$ in Eqs. (2) and (3)) using observed data. There are two main approaches to density estimation. The first approach is *parametric*, where we assume the observed data to be drawn from a known family of distributions (e.g., normal distribution), while the second approach, *non-parametric*, makes no assumption with respect to the distribution of the observed data. For our setting, we consider the non-parametric approach to be more suitable as we have little information about the distribution of the underlying data.

Common approaches for non-parametric estimation are simple discretization methods such as *equal interval binning*, (or histogram estimator), which divides the observed data into equal-sized bins, or *equal frequency intervals*, which divides the observed data into bins with an equal number of samples [DKS95]. The price for the generic approach is a user-supplied parameter such as the bin width in case of equal-sized bins or the number of samples in each bin for the frequency-based binning. A variation of the discretization techniques described above is the *kernel density estimator*, where a kernel function replaces the “box” of the bin estimators. The user-supplied parameter is the kernel bandwidth, and its choice will have a significant impact on its performance.

Finding the optimal value for the user-supplied parameter is nontrivial but important as its value could have a direct effect on the estimator error. The quality of an estimator is evaluated by its bias and variance, and generic formulae for all estimators mentioned above are known [Sil98]. However, the formulae require as input the value of the distribution from which the data is observed, which in our case is not known.

Adaptive estimators use a recursive algorithm to determine the optimal bin width [CHO19]. Such methods use entropy as a measure of disorder in the observed data and determine the bin width that minimizes the entropy function over all possible thresholds. Although they offer good performance, we found this last method lacking for our purpose as it is very time-consuming. Our investigation showed that it would be faster to calculate the key rank for every epoch than to use the entropy-based binning.

As there is no optimal solution for the non-parametric estimation and low computational overhead is an important requirement in our case, we decided to use the histogram estimator. We thoroughly tested its reliability by running two types of experiments. In the first, we simply considered all possible values for the bin widths. In the second, we considered the different known rules for determining the bin widths. For both experiments, we consider all three datasets, described in 2.4 and provide detailed results in Appendix A.

We conclude from the first experiment that although the bin width of the histogram estimator has an impact on the achieved performance, the new metric is stable, and we observe the same behavior for several values of this parameter. More precisely, we observe that any value between 25 and 170 will give similar results. Therefore, we selected the value of 100 for the number of bins as it is in the middle of the interval. This observation holds for both the Hamming weight and the identity leakage models.

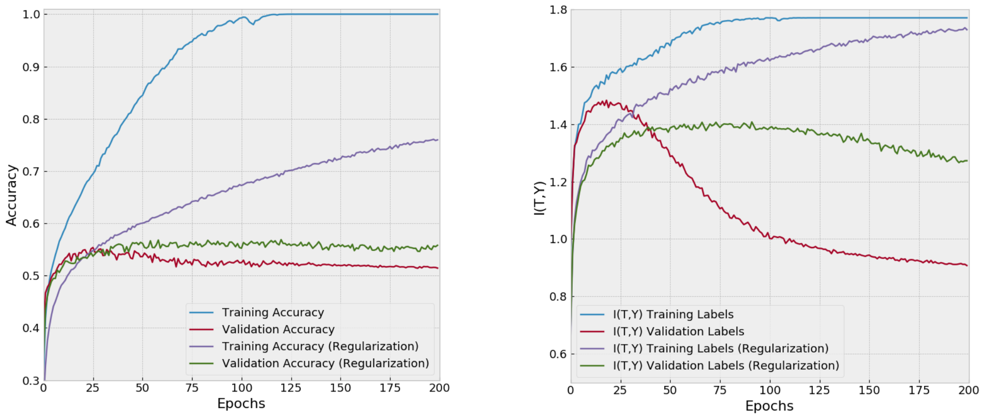
In the second experiment, we use plug-in estimates to determine the value for the bin-width. These estimates work by making assumptions on the distribution of the observed data and are known as empirical rules for determining the bin width. The results are depicted in Appendix A, where we can observe the bin width obtained with different rules result in very similar attack performance. Both our experiments confirm that the mutual information metric is not highly sensitive to the histogram bin size, which makes it a robust procedure for practical applications where one needs to consider different datasets, leakage models, and neural network architectures. Based on this observation, our strategy is to choose the bin value as the average value with good performance for all tested scenarios.

## 4.2 On the Length of the Generalization Interval

As already stated, we aim to reach the generalization interval and then, stop the training. By doing so, we ensure that the trained machine learning model will generalize to unseen data. Still, the question remains how difficult it is to stop at generalization interval. Intuitively, the shorter the interval, the easier it would be to miss it. Ideally, we aim to have a neural network that reaches the generalization interval relatively fast and stays in that interval for a longer period. Naturally, before discussing how to obtain long generalization interval, we must ensure it happens and that we simply do not go to overfitting from underfitting phase.

Regularization techniques can help prevent a deep neural network from overfitting during the training process. To check the impact of the regularization on the neural network and its generalization interval, we use the information plane as it provides a visual indication for the relationship between $I(X;T_n)$ and $I(T_n;Y)$. The maximum value of $I(T_n;Y)$ during training indicates an epoch at which the neural network should be inside the generalization interval for the training process, as defined in Definition 3. When the network does not implement any regularization technique in its hyper-parameter configuration, the trained model has a higher chance of overfitting the training data.

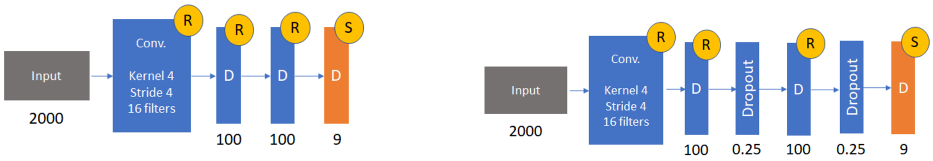
In Figure 5, we depict results for a convolutional neural network with and without regularization (dropout). This experiment is conducted on a proprietary unprotected software AES implementation (STM32 microcontroller). For that, we considered 6 000 traces for the training set and 1 000 traces for the validation set, both having fixed keys. The traces contain 400 features. Observing Figure 5b, for the case without regularization, we see



**(a)** Accuracy with and without regularization (dropout).

**(b)** $I(T_n, Y)$ with and without regularization (dropout).

**Figure 5:** Convolutional neural network configurations (learning rate = 0.001, *Adam* optimizer, batch size = 400, randomly uniform initialized weights).



**(a)** CNN

**(b)** CNN with dropout

**Figure 6:** Convolutional neural network configurations (learning rate = 0.001, *Adam* optimizer, batch size = 400, randomly uniform initialized weights).

that the mutual information $I(T_n; Y)$ reaches a maximum value (where the distributions $T_n$ and $Y$ are obtained from the validation set) and after that, $I(T_n; Y)$ for validation decreases continuously while $I(T_n; Y)$ for the training stays at a maximum value. Additionally, note how $I(T_n; Y)$ indicates that the generalization phase lasts shorter than one would infer from accuracy, as illustrated in Figure 5a.

Figures 5a and 5b also show the accuracy and $I(T_n; Y)$, respectively, for training and validation labels sets obtained from a regularized convolutional neural network with dropout. After processing 200 epochs, the training accuracy has not reached 100%, the desired outcome for a regularized neural network. At the same time, the validation accuracy reaches approximately 56%, which is a significantly higher value compared to 51% without regularization, as shown in Figure 5a. The mutual information $I(T_n; Y)$ for the validation set (see Figure 5b) reaches its maximum value and stays longer at this level. This indicates that the same generalization level is kept until at least epoch 100. Consequently, as the value of $I(T_n; Y)$ stays high for more training epochs, the neural network provides better generalization for those epochs. Again, accuracy is not able to indicate the same phenomenon, as its value remains stable (albeit of different magnitude for validation set) for both regularized and non-regularized networks.

The neural network configurations (with and without dropout) are illustrated in Figure 6. The "R" and "S" labels refer to *ReLU* and *Softmax*, respectively. The number under the layer block indicates the number of neurons in dense layers ("D") and the dropout rate for dropout layers.

## 4.3 Results for the Publicly Available Datasets

In this section, we compare the performance of five metrics (validation loss, validation accuracy, validation recall, key rank for the validation set, and $I(T_n;Y)$) to select the epoch $t$ at which the machine learning model achieves the best performance. Besides these five metrics, we depict the results when we do not use early stopping regime, but rather, allow the full number of training epochs. Those results are denoted as "GE all epochs" and "SR all epochs" for guessing entropy and success rate, respectively. Note, when giving results for guessing entropy and success rate, we conduct one training phase and 100 testing phases per metric to reach the average performance values. The best success rate value equals 100%, and the best guessing entropy value is 1. On the other hand, when giving results with distributions (e.g., Figure 8), we repeat experiments 100 times, i.e., there are 100 training phases to be able to build distributions.

For the three tested datasets (ASCAD, DPAv4, and CHES CTF), the results are obtained by attacking one key byte in the first AES encryption round. The selected leakage model is the Hamming weight of an S-box output. We conduct a tuning phase for hyper-parameters, where we experiment with varying CNN and MLP architectures. We emphasize that we do not claim these architectures to be optimal, as finding optimal architectures was not the goal of this work. The final neural network configurations use the *Adam* optimizer for the backpropagation algorithm, and the learning rate is always set to 0.001. Initial weights are initialized at random using *random uniform* method. The selected loss function is the categorical cross-entropy provided by *Keras* library. All the experiments were performed on a computer equipped with a GPU Nvidia RTX 2060. The details about selected convolutional neural networks hyper-parameters are listed in Table 1. For MLP, we use architecture with five dense layers containing 600 neurons each. We verified that the selected MLP provides good results for the ASCAD database, and the selected CNNs provide strong results for all three considered datasets.

For ASCAD, we additionally give results for the identity leakage model. For DPAv4, we give only the Hamming weight results as the identity leakage model enables an easy attack where there are no significant differences among neural network architectures or validation metrics. For CHES CTF, we again give only the Hamming weight leakage model as the currently available dataset contains only 43 000 traces, which is not enough to break the target in the identity leakage model.

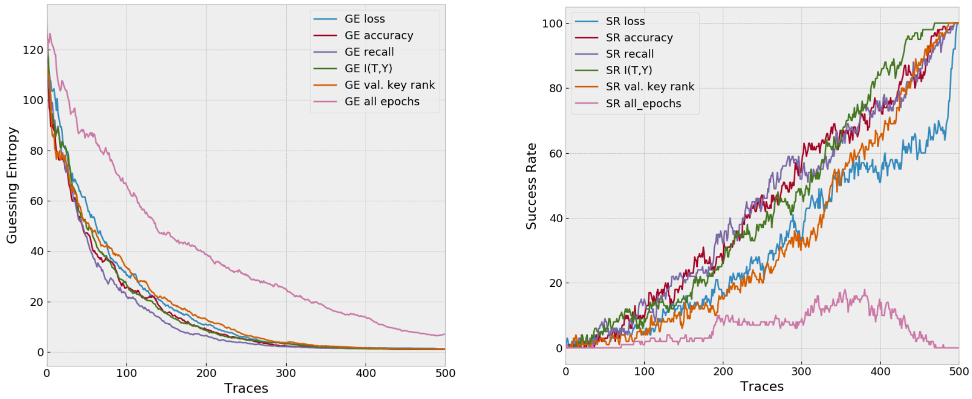
**Table 1:** Hyper-parameters for convolutional neural networks.

| Dataset | DPAv4 | ASCAD | CHES CTF 2018 |
|---|---|---|---|
| Learning Rate | 0.001 | 0.001 | 0.001 |
| Optimizer | Adam | Adam | Adam |
| Mini-batch | 400 | 2 000 | 400 |
| Convolution layers | 1 | 1 | 1 |
| Filters, Kernel Size, Stride | 16, 10, 10 | 16, 10, 10 | 16, 4, 4 |
| Dense (fully-connected) layers | 4 | 3 | 2 |
| Neurons (for dense or fully-connected layers) | 400 | 200 | 100 |
| Activation Function (all layers) | ReLU | ReLU | ReLU |

### 4.3.1 AES-128 Encryption - ASCAD (Random Keys)

The empirical validation on the ASCAD database (key byte 3) considers 200 000 traces for training, 500 traces for validation, and 500 traces for the test. Both validation and test sets have a fixed key. The selected CNN architecture is trained for 50 epochs. After identifying the best epoch for each of the five metrics, the corresponding machine learning models are applied to the test set.

Figure 7 shows the guessing entropy and success rate for the test set obtained for each validation metric for the Hamming weight leakage model. From Figure 7b, the best success rate is achieved when the machine learning model is selected from the epoch when the metric is the maximum value of $I(T_n;Y)$. More precisely, around the processing of 460 traces, the success rate reaches 100% if the model is selected from the epoch determined by the maximum $I(T_n;Y)$ value. The lines "GE all epochs" and "SR all epochs" correspond to the results when evaluating GE and SR after the processing of 50 epochs. We can see that those lines also depict the worst attack performance as in those cases, due to too many training epochs, the machine learning models overfit and do not generalize for the test set. Figure 7a shows no significant differences among most of the metrics (except the scenario where we do not use early stopping), and for both SCA metrics, we see that mutual information works well and gives consistently strong attack performance.



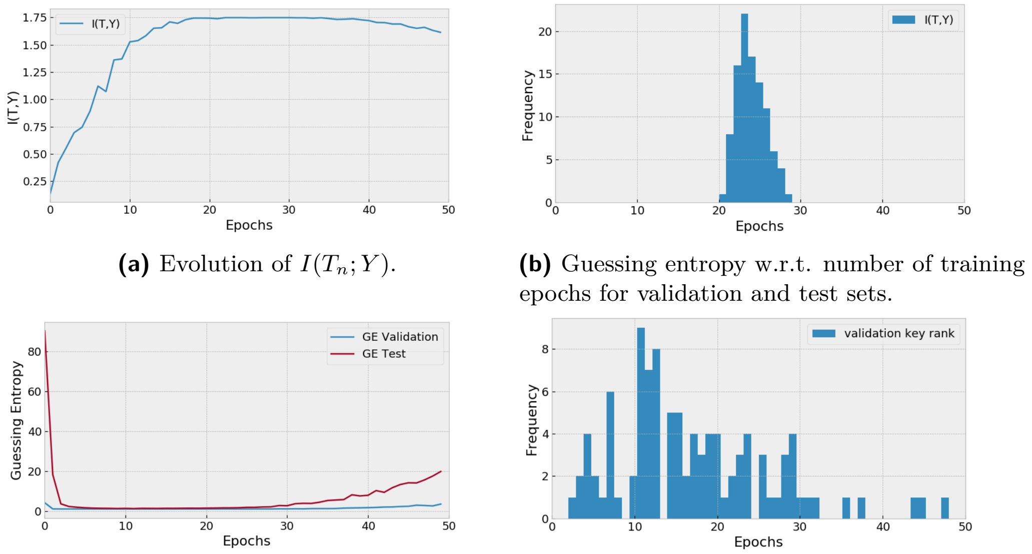
**(a)** Guessing entropy.

**(b)** Success Rate.

**Figure 7:** Results on ASCAD for the Hamming weight leakage model, CNN architecture.

Figure 8 shows the results for 100 experiments with the same CNN architecture. Figure 8a gives the $I(T_n;Y)$ evolution for the processed epochs. On average, the highest $I(T_n;Y)$ values are achieved between epochs 20 and 30 (the highest for epoch 24), as indicated by the plot distribution in Figure 8b. Figure 8c shows the test and validation guessing entropy results for the number of epochs (training phase) and $I(T_n;Y)$ metric. We see there is an interval (epochs between 8 and 26) in which the key rank is low and, consequently, generalization is satisfactory. This indicates that guessing entropy reaches good values even before $I(T_n;Y)$ becomes maximal. Still, allowing more epochs does increase $I(T_n;Y)$, while keeping GE minimal. Figure 8d shows the distribution of the best epochs based on the validation key rank metric. This histogram contains the results of 100 experiments (unchanged hyper-parameters) and indicates that the best validation key rank may happen at different epochs. More precisely, we see the highest value already around epoch 12, which explains why validation key rank is also among the worst performing metrics for both SR and GE. While this could sound counter-intuitive, there is a simple explanation for such behavior. As we use a validation set often (whenever evaluating whether to stop training), the validation set indirectly influences the trained model. Consequently, it is possible to observe some differences when applying trained models to the test set, which was never evaluated before. As we can observe, $I(T_n;Y)$ metric seems to be less sensitive to this issue, and as such, it represents a more suitable choice for early stopping metric.

Next, we repeat the experiments for the ASCAD dataset in the Hamming weight model,



**(a)** Evolution of $I(T_n;Y)$.

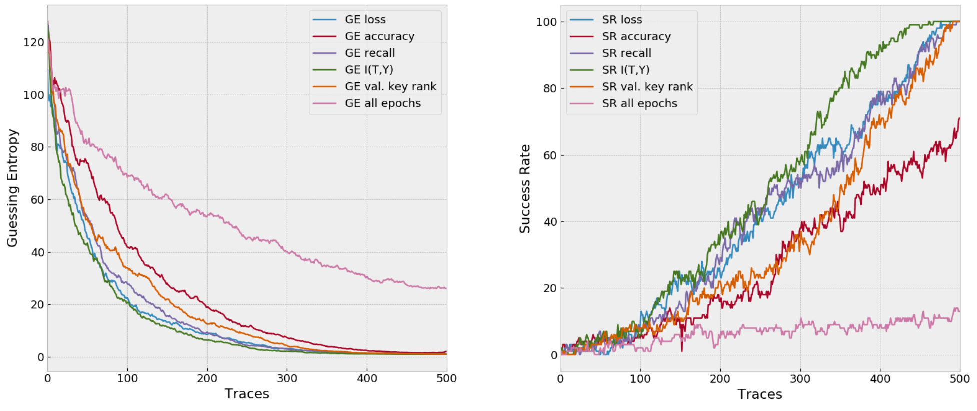
**(b)** Guessing entropy w.r.t. number of training epochs for validation and test sets.

**(c)** Distribution of "early stopping" epoch having maximum $I(T_n;Y)$ as a metric.

**(d)** Distribution of "early stopping" epoch having validation key rank as metric.

**Figure 8:** Results on ASCAD for the Hamming weight leakage model, CNN architecture.

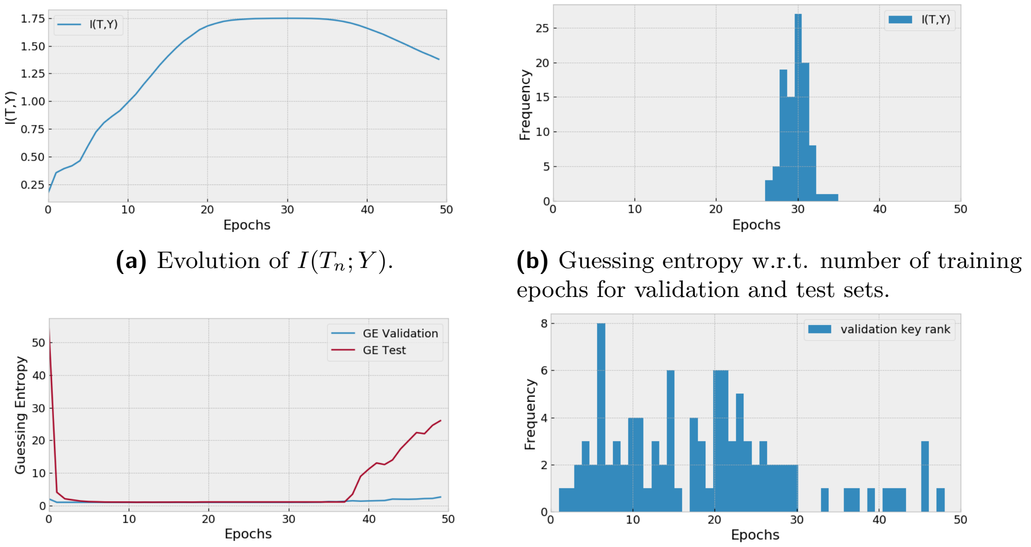
but now, we use MLP architecture. From Figures 9a and 9b, we can see $I(T_n;Y)$ as being the most successful metric, followed closely by loss. Again, we see if we do not use early stopping, neural network overfits, which results in poor attack performance. Figure 10a indicates that $I(T_n;Y)$ reaches the best performance for epochs 25 to 35. This is confirmed in Figure 10b where we observe the highest frequency for epoch 31. Considering the validation and test set behavior when using $I(T_n;Y)$ to indicate stopping, a number of epochs give good behavior (from 10 to 35). This is in line with the behavior for CNN, as GE can indicate a successful attack even before $I(T_n;Y)$ reaches the maximal value. Finally, Figure 10b gives insight into the performance of the validation key rank, where we see a number of epochs to have high frequency, but the highest value happens around epoch 5, which is too early as confirmed when evaluating the attack performance (Figure 9 where the validation key rank performs much worse than $I(T_n;Y)$).



**(a)** Guessing entropy.

**(b)** Success Rate.

**Figure 9:** Results on ASCAD for the Hamming weight leakage model, MLP architecture.



**(a)** Evolution of $I(T_n;Y)$.

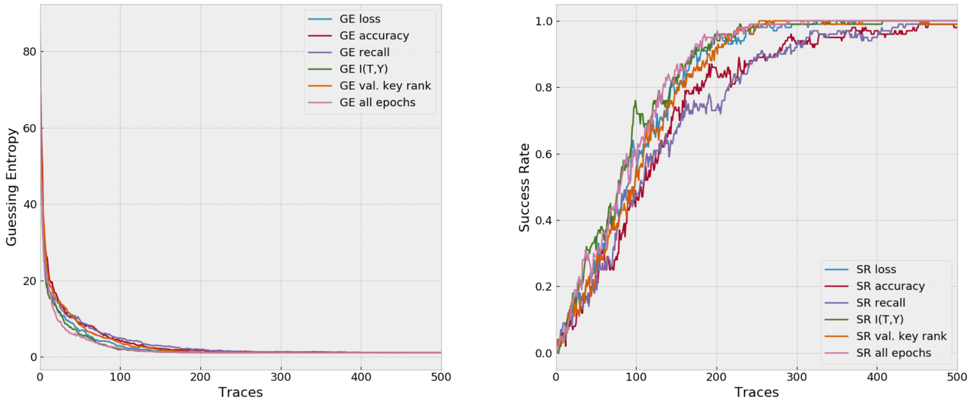
**(b)** Guessing entropy w.r.t. number of training epochs for validation and test sets.

**(c)** Distribution of "early stopping" epoch having maximum $I(T_n;Y)$ as a metric.

**(d)** Distribution of "early stopping" epoch having validation key rank as metric.

**Figure 10:** Results on ASCAD for the Hamming weight leakage model, MLP architecture.

Next, we consider the identity leakage model for the ASCAD dataset. First, in Figure 11, we depict the results for guessing entropy and success rate. The differences among attack performances are very small, but for both guessing entropy and success rate, the mutual information metric gives good results. Here, no early stopping mechanism does not affect attack performance. This behavior is expected, as now there are more classes, so neural networks need more epochs to fit the data into the model (and naturally, to overfit).

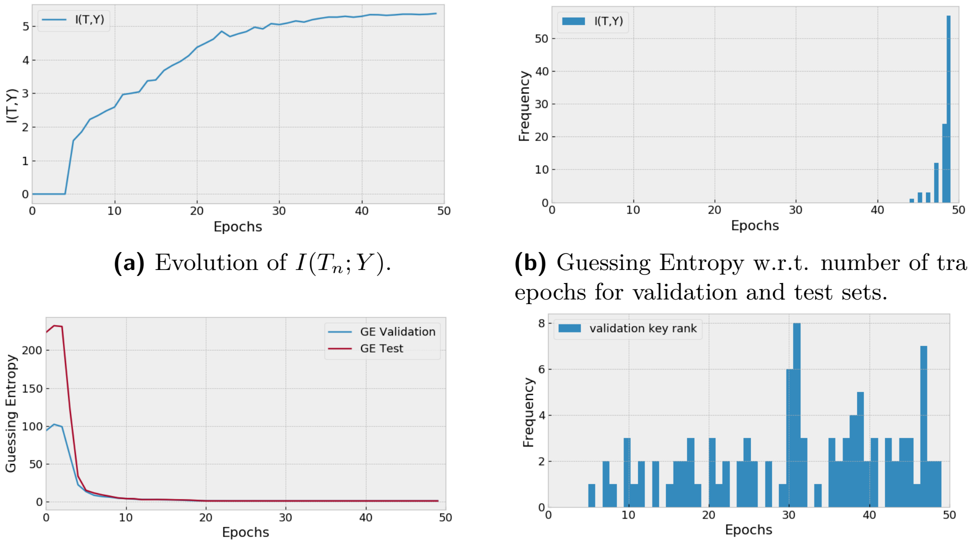


**(a)** Guessing entropy

**(b)** Success Rate

**Figure 11:** Results on ASCAD for the identity leakage model, CNN architecture.

Figure 12a displays the $I(T_n;Y)$ evolution over 50 epochs. The mutual information increases with the number of epochs and reaches a steady level around epoch 47. This is confirmed in Figure 12b, where we can indeed observe that the epochs 47 to 49 give the best results. When considering guessing entropy, both validation and test set values indicate strong performance when having more than 15 epochs. Using the validation key rank as the early stopping metric shows a number of epochs as suitable to stop the training process (Figure 12d). Still, the two highest peaks are observed around epochs 32 and 48. As the validation key rank and $I(T_n; Y)$ point to similar epochs to stop the training, the results in Figure 11 are as expected – no significant difference in the attack performance.



**(a)** Evolution of $I(T_n; Y)$.

**(b)** Guessing Entropy w.r.t. number of training epochs for validation and test sets.

**(c)** Distribution of "early stopping" epoch having maximum $I(T_n; Y)$ as a metric.

**(d)** Distribution of "early stopping" epoch having validation key rank as metric.

**Figure 12:** Results on ASCAD for the identity leakage model, CNN architecture.

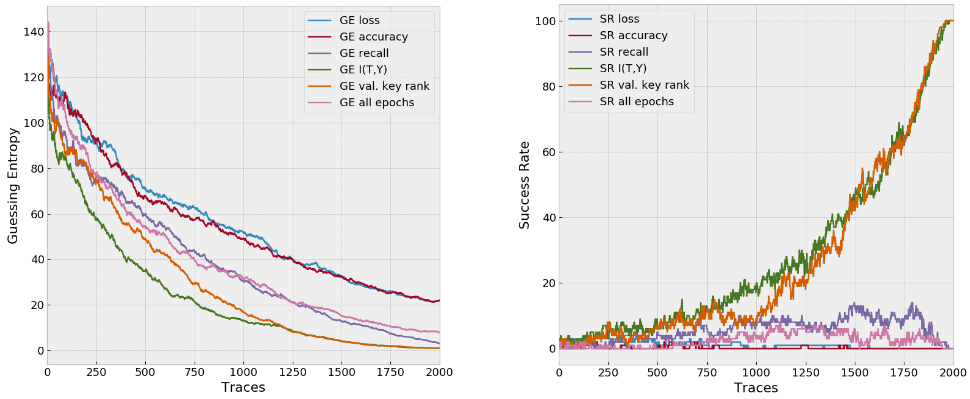
### 4.3.2 AES-256 Encryption - DPAv4

For the DPAv4 dataset, we consider 34 000 traces in the training set and 2 000 traces in the validation set. An additional 2 000 traces are used as a test set. These results were obtained from the 100 training runs on CNN configured with unchanged hyper-parameters. Figure 13 shows the guessing entropy and success rate obtained from the selected metrics (accuracy, recall, loss, key rank, and maximum $I(T_n; Y)$) from the validation set. Again, selecting the model at an epoch with the maximum $I(T_n, Y)$ for the validation set provides the best results for both success rates and guessing entropy. Similar to the ASCAD dataset in the Hamming weight leakage model, for a small number of attack traces, $I(T_n; Y)$ gives better results than the validation key rank. Again, this happens due to the influence of the validation set to the trained model. Interestingly, here we can observe that allowing training for all 50 epochs leads to overfitting, but the same behavior happens if we would stop training on the basis of loss, recall, and accuracy.

As we can see from Figures 14a and 14b, the network achieves its maximum $I(T_n; Y)$ value between epochs 10 and 16. Figure 14c confirms that we require around 10 epochs to reach guessing entropy of 1. Additionally, the behavior stays relatively stable up to epoch 38 (where there is no deterioration up to epoch 15, and afterward, there are slight changes in GE). Finally, in Figure 14d, the validation key rank agrees with $I(T_n; Y)$ by reaching the maximal frequency values for epochs 11 to 15 (cf. Figure 14b).

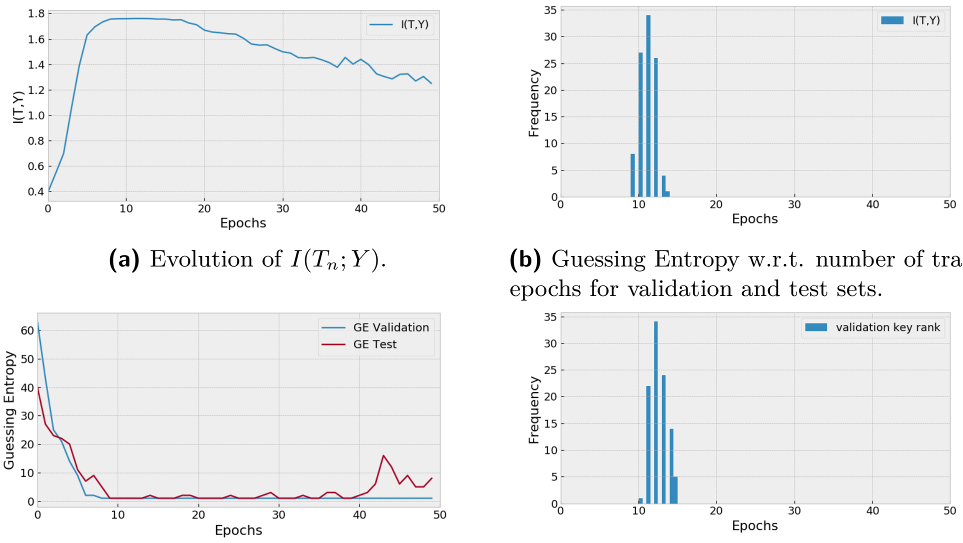
### 4.3.3 AES-128 Encryption - CHES CTF 2018

For the CHES CTF dataset, we consider 43 000 traces in the training set and 1 000 traces in the validation set. Additional 1 000 traces are used as a test set. These results were obtained from the 100 training runs on CNN configured with the unchanged hyper-parameters.



**(a)** Guessing entropy **(b)** Success Rate

**Figure 13:** Results on DPAv4 for the Hamming weight leakage model, CNN architecture.



**(a)** Evolution of $I(T_n;Y)$.

**(b)** Guessing Entropy w.r.t. number of training epochs for validation and test sets.

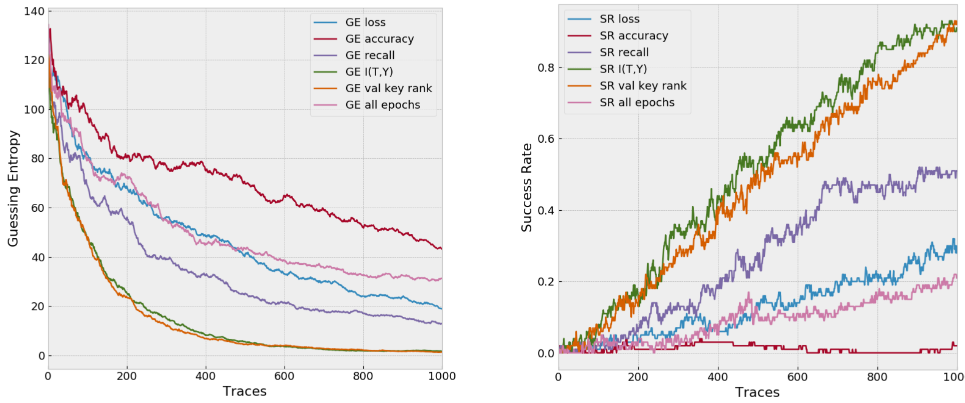
**(c)** Distribution of "early stopping" epoch having maximum $I(T_n;Y)$ as a metric.

**(d)** Distribution of "early stopping" epoch having validation key rank as metric.

**Figure 14:** Results on DPAv4 for the Hamming weight leakage model, CNN architecture.

Figure 15 shows the guessing entropy and success rate for the five considered metrics. We can observe that using the training model at the epoch with the maximum $I(T_n;Y)$ provides the best success rate and guessing entropy (followed closely by the validation key rank). At the same time, retrieving the model at epochs indicated by the best validation accuracy, loss, or recall leads to significantly worse success rate and guessing entropy results. Similarly, if there is no early stopping, the attack performance is also poor.

Figure 16a provides the mutual information value $I(T_n;Y)$ for training phase and every epoch. The maximum $I(T_n;Y)$ is reached between epochs 10 and 18. Figure 16b gives similar indication with epoch 14 having the highest frequency. Those results are confirmed in Figure 16c, where epochs 10 to 15 have the lowest guessing entropy. What is more, after epoch 18, the neural network starts to degrade its generalization capacity as it starts to overfit on the training set. On the other hand, the generalization capacity before epoch 7 also provides, on average, poor generalization since the network is inside the fitting phase,

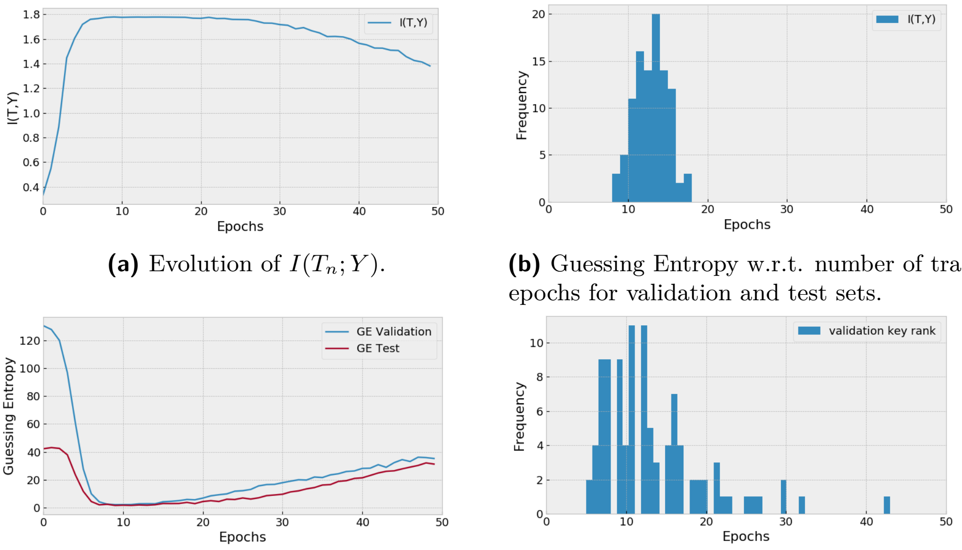


**(a)** Guessing entropy

**(b)** Success Rate

**Figure 15:** Results on CHES CTF 2018 for the Hamming weight leakage model, CNN architecture.

where satisfactory generalization is not achieved yet (i.e., the network underfits). Finally, in Figure 16d, the validation key rank gives similar results (thus, we have similar SCA metrics results in Figure 15). Still, again the validation key rank indicates to stop the training a little bit earlier than $I(T_n;Y)$.



**(a)** Evolution of $I(T_n;Y)$.

**(b)** Guessing Entropy w.r.t. number of training epochs for validation and test sets.

**(c)** Distribution of "early stopping" epoch having maximum $I(T_n;Y)$ as a metric.

**(d)** Distribution of "early stopping" epoch having validation key rank as metric.

**Figure 16:** Results on CHES CTF 2018 for the Hamming weight leakage model, CNN architecture.

When attacking a protected target, like the public databases consisting of the first-order masked AES implementations, model generalization is very limited, and validation or test metrics are close to random guessing. For side-channel analysis, a sufficient generalization is given by a low guessing entropy or high success rate. As we can observe from the results given in Section 4, the trained model at each epoch provides different key rank results, and the over-training easily leads to deterioration of the model's generalization. This problem can be addressed by using an appropriate metric to save the trained model at the epoch that provides the best success rate or guessing entropy. Our experimental analysis shows that having the maximum value of $I(T_n;Y)$ as a metric to select the model at the best epoch provides better success rate and guessing entropy results when compared to machine learning metrics like loss, recall, or accuracy. Additionally, we observe that $I(T_n;Y)$ works especially well in settings where other metrics could have problems, as is the Hamming weight leakage model that suffers from data imbalance. The $I(T_n;Y)$ metric works even better than the validation key rank, where we notice that the key rank validation indicates somewhat earlier to stop the training. Based on the obtained results, we give several observations for deep learning-based SCA:

1. It is necessary to implement early stopping regularization.
2. Early stopping based on mutual information consistently gives the best results.
3. Validation key rank seems to be somewhat more conservative in its estimate than the mutual information.
4. Guessing entropy reaches good values even before $I(T_n;Y)$ becomes maximal. Still, waiting for $I(T_n;Y)$ to become maximal has the effect of reaching the most stable attack behavior.
5. Mutual information metric is a computationally simple technique and is not sensitive to the histogram estimation procedure.

## 5 Conclusions and Future Work

The selection of correct performance metrics can lead to a better interpretation of deep neural networks. In particular, the usage of deep learning for profiled side-channel attacks on the masked AES implementations is very sensitive to the number of processed epochs (i.e., how long is the training). In this paper, we demonstrate that using the mutual information between output layer activations (i.e., the output *Softmax* probabilities) and the true labels $I(T_n;Y)$ of a validation set leads to better generalization for separate test sets. We compared $I(T_n;Y)$ metric against conventional machine learning metrics (accuracy, recall, and loss), and we verified that mutual information is a much more reliable metric to detect an epoch at which the trained neural network is inside a generalization interval. As such, we can conclude that mutual information should be considered as a metric of choice when conducting a deep learning-based side-channel analysis.

In future work, we aim to investigate the usage of mutual information metric as a reference for the selection of other hyper-parameters. Additionally, we would like to investigate the behavior of this metric in cases when the traces contain misalignment, and consequently, the generalization is even more difficult. Such analysis is also essential to improve the portability capabilities of trained deep neural networks for side-channel attacks.

## Acknowledgements

This work was supported by the European Union's H2020 Programme under grant agreement number ICT-731591 (REASSURE).

## References

[AG18] Rana Ali Amjad and Bernhard C. Geiger. How (not) to train your neural network using the information bottleneck principle. *CoRR*, abs/1802.09766, 2018.

[Ano20] Anonymous for TCHES submission. Source code for mutual information approach for stopping the training), 2020. https://gitlab.com/ches_submission_dl_mia/ches_submission_dl_mia.

[BCO04] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In Marc Joye and Jean-Jacques Quisquater, editors, *Cryptographic Hardware and Embedded Systems - CHES 2004: 6th International Workshop Cambridge, MA, USA, August 11-13, 2004. Proceedings*, volume 3156 of *Lecture Notes in Computer Science*, pages 16–29. Springer, 2004.

[CDP17] Eleonora Cagli, Cécile Dumas, and Emmanuel Prouff. Convolutional neural networks with data augmentation against jitter-based countermeasures - profiling attacks without pre-processing. In Wieland Fischer and Naofumi Homma, editors, *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, volume 10529 of *Lecture Notes in Computer Science*, pages 45–68. Springer, 2017.

[CHO19] Ivan Chelombiev, Conor Houghton, and Cian O'Donnell. Adaptive estimators show information compression in deep neural networks. In *International Conference on Learning Representations*, 2019.

[CRR02] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template attacks. In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, volume 2523 of *Lecture Notes in Computer Science*, pages 13–28. Springer, 2002.

[DKS95] James Dougherty, Ron Kohavi, and Mehran Sahami. Supervised and unsupervised discretization of continuous features. In Armand Prieditis and Stuart J. Russell, editors, *Machine Learning, Proceedings of the Twelfth International Conference on Machine Learning, Tahoe City, California, USA, July 9-12, 1995*, pages 194–202. Morgan Kaufmann, 1995.

[FR14] Aurélien Francillon and Pankaj Rohatgi, editors. *Smart Card Research and Advanced Applications - 12th International Conference, CARDIS 2013, Berlin, Germany, November 27-29, 2013. Revised Selected Papers*, volume 8419 of *Lecture Notes in Computer Science*. Springer, 2014.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[HGG19] Benjamin Hettwer, Stefan Gehrer, and Tim Güneysu. Profiled power analysis attacks using convolutional neural networks with domain knowledge. In Carlos Cid and Michael J. Jacobson Jr., editors, *Selected Areas in Cryptography – SAC 2018*, pages 479–498, Cham, 2019. Springer International Publishing.

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.

[KPH$^+$19] Jaehun Kim, Stjepan Picek, Annelie Heuser, Shivam Bhasin, and Alan Hanjalic. Make some noise. unleashing the power of convolutional neural networks for profiled side-channel analysis. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(3):148–179, 2019.

[KSG04] Alexander Kraskov, Harald Stögbauer, and Peter Grassberger. Estimating mutual information. *Physical Review E*, 69(6), Jun 2004.

[LMBM13] Liran Lerman, Stephane Fernandes Medeiros, Gianluca Bontempi, and Olivier Markowitch. A machine learning approach against a masked AES. In Francillon and Rohatgi [FR14], pages 61–75.

[MDP19] Loïc Masure, Cécile Dumas, and Emmanuel Prouff. Gradient visualization for general characterization in profiling attacks. In Ilia Polian and Marc Stöttinger, editors, *Constructive Side-Channel Analysis and Secure Design - 10th International Workshop, COSADE 2019, Darmstadt, Germany, April 3-5, 2019, Proceedings*, volume 11421 of *Lecture Notes in Computer Science*, pages 145–167. Springer, 2019.

[MDP20] Loïc Masure, Cécile Dumas, and Emmanuel Prouff. A comprehensive study of deep learning for side-channel analysis. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(1):348–375, 2020.

[MHM13] Zdenek Martinasek, Jan Hajny, and Lukas Malina. Optimization of power analysis using neural network. In Francillon and Rohatgi [FR14], pages 94–107.

[MPP16] Houssem Maghrebi, Thibault Portigliatti, and Emmanuel Prouff. Breaking cryptographic implementations using deep learning techniques. In Claude Carlet, M. Anwar Hasan, and Vishal Saraswat, editors, *Security, Privacy, and Applied Cryptography Engineering - 6th International Conference, SPACE 2016, Hyderabad, India, December 14-18, 2016, Proceedings*, volume 10076 of *Lecture Notes in Computer Science*, pages 3–26. Springer, 2016.

[MZVT15] Zdenek Martinasek, Ondrej Zapletal, Kamil Vrba, and Krisztina Trasy. Power analysis attack based on the MLP in DPA contest v4. In *38th International Conference on Telecommunications and Signal Processing, TSP 2015, Prague, Czech Republic, July 9-11, 2015*, pages 154–158. IEEE, 2015.

[PHJ$^+$19] Stjepan Picek, Annelie Heuser, Alan Jovic, Shivam Bhasin, and Francesco Regazzoni. The curse of class imbalance and conflicting metrics with machine learning for side-channel evaluations. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(1):209–237, 2019.

[PSB$^+$18] Emmanuel Prouff, Remi Strullu, Ryad Benadjila, Eleonora Cagli, and Cécile Dumas. Study of deep learning techniques for side-channel analysis and introduction to ASCAD database. *IACR Cryptology ePrint Archive*, 2018:53, 2018.

[RZC$^+$20] Damien Robissout, Gabriel Zaid, Brice Colombier, Lilian Bossuet, and Amaury Habrard. Online performance evaluation of deep learning networks for side-channel analysis. Cryptology ePrint Archive, Report 2020/039, 2020. `https://eprint.iacr.org/2020/039`.

[SBD$^+$18] Andrew M. Saxe, Yamini Bansal, Joel Dapello, Madhu Advani, Artemy Kolchinsky, Brendan D. Tracey, and David Daniel Cox. On the information bottleneck theory of deep learning. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018.

[Sil98] Bernard. W. Silverman. *Density Estimation for Statistics and Data Analysis*. Chapman and Hall, 1998. https://doi.org/10.1201/9781315140919.

[SLP05] Werner Schindler, Kerstin Lemke, and Christof Paar. A stochastic model for differential side channel cryptanalysis. In Josyula R. Rao and Berk Sunar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2005, 7th International Workshop, Edinburgh, UK, August 29 - September 1, 2005, Proceedings*, volume 3659 of *Lecture Notes in Computer Science*, pages 30–46. Springer, 2005.

[SMY09] François-Xavier Standaert, Tal G Malkin, and Moti Yung. A unified framework for the analysis of side-channel key recovery attacks. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 443–461. Springer, 2009.

[ST17] Ravid Shwartz-Ziv and Naftali Tishby. Opening the black box of deep neural networks via information. *CoRR*, abs/1703.00810, 2017.

[TEL14] TELECOM ParisTech SEN research group. DPA Contest ($4^{\text{th}}$ edition), 2013–2014. http://www.DPAcontest.org/v4/.

[TZ15] Naftali Tishby and Noga Zaslavsky. Deep learning and the information bottleneck principle, 2015.

[vdVP19] Daan van der Valk and Stjepan Picek. Bias-variance decomposition in machine learning-based side-channel analysis. Cryptology ePrint Archive, Report 2019/570, 2019. https://eprint.iacr.org/2019/570.

[vdVPB19] Daan van der Valk, Stjepan Picek, and Shivam Bhasin. Kilroy was here: The first step towards explainability of neural networks in profiled side-channel analysis. Cryptology ePrint Archive, Report 2019/1477, 2019. https://eprint.iacr.org/2019/1477.

[ZBHV19] Gabriel Zaid, Lilian Bossuet, Amaury Habrard, and Alexandre Venelli. Methodology for efficient cnn architectures in profiling attacks. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020(1):1–36, Nov. 2019.

## A Bin Size Estimators

To estimate the probability density, a critical step is to determine the bin width, which is the user-supplied parameter for the histogram estimator. The results of our experiments are shown in Figures 17, 18, and 19. As shown in the figures on the right, any bin width larger than 15 leads to a final key rank lower than 4 (key rank equal to 1 indicates the successful key recovery). The key rank is computed for a separate test set and is obtained by selecting the machine learning model at the epoch that gives the highest $I(T_n;Y)$ for each tested bin width. The plots on the left side of Figures 17, 18, and 19 show the value of $I(T_n;Y)$ w.r.t. the number of epochs for all tested bin sizes. As we can see, if the bin size is too small, the mutual information $I(T_n;Y)$ barely changes.

Figure 20 shows results for eight different plug-in estimators, vs. the choice of fixing the number of bins to 100. The plug-in estimators we tested are Freedman Diaconis ('fd'), 'sturges', 'auto' (which is the maximum of 'fd' and 'sturges' estimators), 'rice', 'scott', square-root estimator ('sqrt'), and 'doane'. The results show the guessing entropy resulting from early stopping by having $I(T_n;Y)$ as metric by testing different bin size estimators. As we can see, they all lead to successful key recovery with similar guessing entropy convergence for all tested datasets and leakage models.

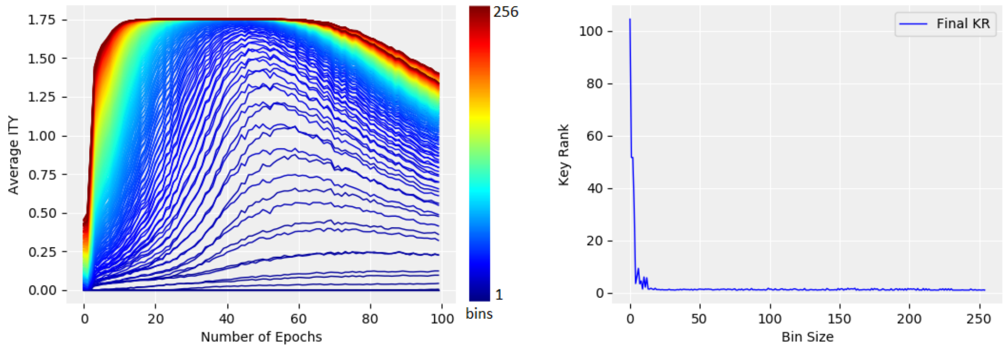


**Figure 17:** (to be viewed in colors) The influence of the number of bins in the calculation of $I(T_n;Y)$ for the ASCAD dataset. Average $I(T_n;Y)$ w.r.t. the number of epochs for different bin sizes (left). The final key ranking having the maximum value of $I(T_n;Y)$ as a reference metric for different bin sizes (1 to 256) (right).

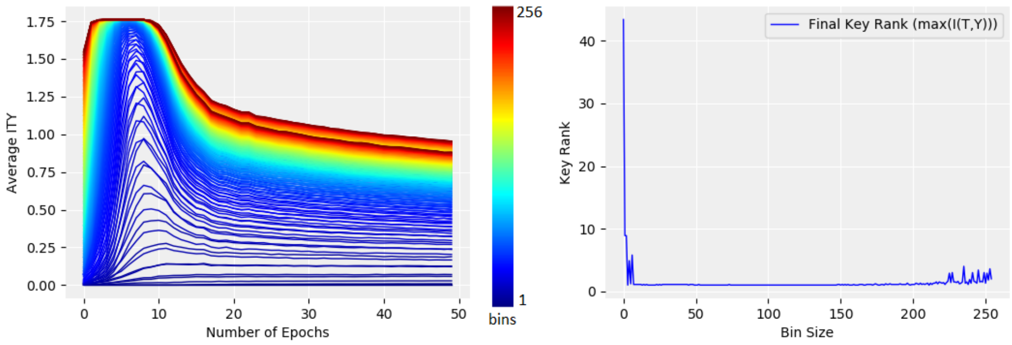


**Figure 18:** (to be viewed in colors) The influence of the number of bins in the calculation of $I(T_n;Y)$ for the DPAv4 dataset. Average $I(T_n;Y)$ w.r.t. the number of epochs for different bin sizes (left). The final key ranking having the maximum value of $I(T_n;Y)$ as a reference metric for different bin sizes (1 to 256) (right).

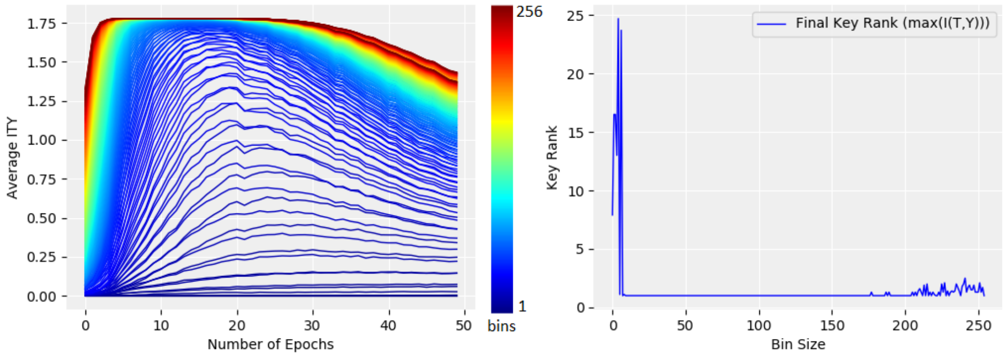
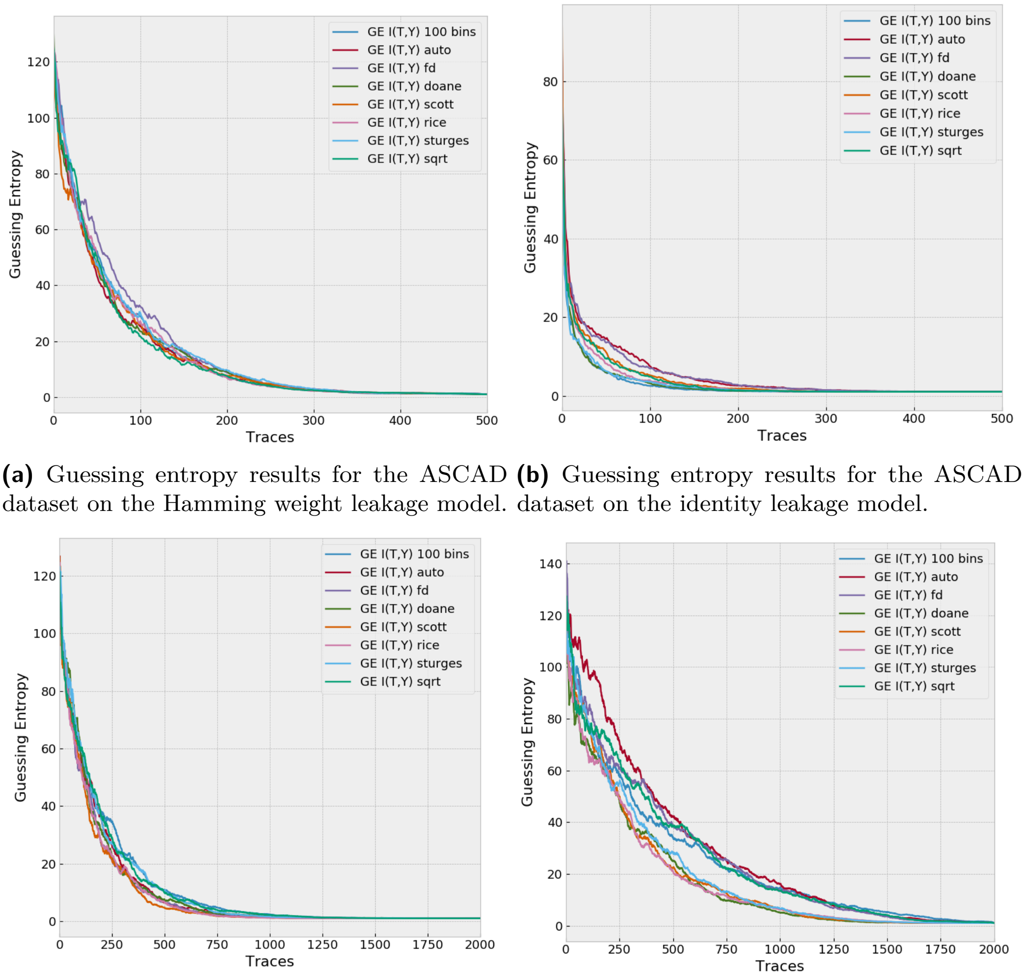


**Figure 19:** (to be viewed in colors) The influence of the number of bins in the calculation of $I(T_i,Y)$ for the CHES CTF 2018 dataset. Average $I(T_i,Y)$ w.r.t. the number of epochs for different bin sizes (left). The final key ranking having the maximum value of $I(T_i,Y)$ as a reference metric for different bin sizes (1 to 256) (right).



**(a)** Guessing entropy results for the ASCAD dataset on the Hamming weight leakage model.

**(b)** Guessing entropy results for the ASCAD dataset on the identity leakage model.

**(c)** Guessing entropy results for the CHES CTF dataset on the Hamming weight leakage model.

**(d)** Guessing entropy results for the DPAv4 dataset on the Hamming weight leakage model.

**Figure 20:** Guessing entropy results when early stopping is conducted with mutual information as a metric for different binning size estimators.