On the Performance of Multilayer Perceptron in Profiling Side-channel Analysis

Léo Weissbart[1,2], Stjepan Picek[1], and Lejla Batina[2]

[1] Delft University of Technology, The Netherlands
[2] Digital Security Group, Radboud University, The Netherlands

Abstract. In profiling side-channel analysis, machine learning-based attacks nowadays offer the most powerful performance. This holds especially for techniques stemming from the neural network family: multilayer perceptron and convolutional neural networks. Convolutional neural networks are often favored as state-of-the-art results suggest better performance, especially in scenarios where targets are protected with countermeasures. Multilayer perceptron receives much less attention and researchers seem less interested in this technique, narrowing the results in the literature to comparisons with convolutional neural networks. Yet, a multilayer perceptron has a much simpler structure, which enables easier hyperparameter tuning, and hopefully, could contribute to the explainability of this neural network inner working.
In this paper, we investigate the behavior of a multilayer perceptron in detail in the context of the side-channel analysis of AES. By exploring the sensitivity of multilayer perceptron hyperparameters over the performance of the attack, we aim at providing a better understanding of successful hyperparameters tuning, and ultimately, the performance of this algorithm. Our results show that MLP (with a proper hyperparameter tuning) can easily break implementations having a random delay or masking countermeasures.

1 Introduction

Side-channel analysis (SCA) exploits weaknesses in physical implementations of cryptographic algorithms rather than the mathematical properties of the algorithms [17]. There, SCA correlates secret information with unintentional leakages like timing [14], power dissipation [15], and electromagnetic (EM) radiation [25]. One common division of SCA is into non-profiling (direct) attacks and profiling (two-stage) attacks. Profiling SCA is the worst-case security analysis as it considers the most powerful side-channel attacker with access to a clone device (where keys can be chosen and/or are known by the attacker). During the past few years, numerous works showed the potential and strength of machine learning in the context of profiling side-channel analysis. Indeed, across various targets and scenarios, researchers were able to show that machine learning can outperform other techniques considered as the state-of-the-art in the SCA community [16,2]. More interestingly, some machine learning techniques are successful even on implementations protected with countermeasures [13,2]. There, in the spotlight are

techniques from the neural network family, most notably, multilayer perceptron (MLP) and convolutional neural networks (CNNs).

For most of the research in the SCA domain, the inner working of a machine learning method is still considered as a black-box, as it cannot be naturally visualized. By tuning a set of variables before the learning process (i.e., the hyperparameters), it is possible to influence the performance of the attack. Still, the features learned during this process are difficult to explain and there exists no proof showing that a machine learning method cannot perform even better. Similarly, there are no proofs if there exist countermeasures that would make a machine learning process impossible.

When considering the attack success, commonly we take into account only the performance as measured by the number of traces needed to obtain the key. While this is an important criterion, it should not be the only one. For instance, attack complexity (complexity of tuning and training a model) and interpretability of the attack are also very important but much less researched. For instance, CNNs are often said to perform better than MLPs in the context of SCA, as they make the training of a model more versatile and remove the feature engineering process. On the other side, while MLP has a simpler structure and is probably easier to understand than CNNs, less attention is raised around its performance in SCA. Consequently, this raises an interesting dilemma: do we consider profiling SCA as a single-objective problem where the attack performance is the only criterion or should it be a multi-objective problem where one considers several aspects of "success". We believe the proper approach is the second one as without a better understanding of attacks we cannot make better countermeasures, which is an integral part of the profiling SCA research. Still, before diving into the interpretability of MLP, we must better understand its performance and robustness as measured by the attack performance and tuning difficulty.

In this paper, we experimentally investigate the performance of MLP when applied to real-world implementations protected with countermeasures and explore the sensitivity of the hyperparameter tuning of a successful MLP architecture. We emphasize that this work does not aim to compare the performance of different techniques, but rather to explore the capabilities of multilayer perceptron. To achieve this, we use two datasets containing different AES implementations protected with random delay countermeasure and masking countermeasure. Our results show that we require larger architectures only if we have enough high-quality data. Hence, one can (to a certain degree) overcome the limitation in the number of hidden layers by providing more perceptrons per layer or vice versa. Finally, while our experiments clearly show the difference in the performance concerning the choice of hyperparameters, we do not notice that MLP is overly sensitive to that choice. This means it is possible to conduct a relatively short tuning phase and still expect not to miss a hyperparameter combination yielding high performance.

The rest of this paper is organized as follows. In Section 2, we discuss the related works and in Section 3, profiling side-channel analysis and multilayer perceptron algorithm. Section 4 gives details on our experimental setup. In Section 5, we present the experimental results while in Section 6, we give a discussion on results. Finally, Section 7 concludes the paper with several possible future research directions.

2 Related Work

The corpus of works on machine learning and SCA so far is substantial, so here we concentrate only on works considering multilayer perceptron. Yang et al. considered neural networks and back-propagation as a setting for profiling SCA [30]. They indicated that "...neural network based power leakage characterization attack can largely improve the effectiveness of the attacks, regardless of the impact of noise and the limited number of power traces." Next, Zeman and Martinasek investigated MLP for profiling SCA where they mentioned the machine learning algorithm simply as "neural network" [18]. They considered an architecture with only a single hidden layer and experimented with several possible numbers of neurons in that layer. Finally, they used only one activation function – sigmoid. After those, there have been several papers using MLP with good results, but usually comparable with other machine learning techniques [8,19,12,11]. Still, the hyperparameter tuning was often not sufficiently explored and the datasets were not so realistic. In 2016, Maghrebi et al. conducted the first experiments with convolutional neural networks for SCA and they compared their performance with several other techniques (including MLP) [16]. Their results indicated that, while MLP is powerful, CNNs can perform significantly better. From that moment on, we observe a number of papers where various deep learning techniques are being considered, see, e.g., [24,23,21,10].

Pfeifer and Haddad considered how to make additional types of layers for MLP to improve the performance of such profiling SCA [20]. B. Timon investigated the "non-profiled" deep learning paradigm where he first obtained the measurements in a non-profiled way, which are then fed into MLP or CNN [29]. Interestingly, he reported even better results with MLP than CNNs. Finally, Picek et al. connected the Universal Approximation Theorem and performance of the side-channel attack where they stated if the attacker has unlimited power (as it is usually considered), then most of the MLP-based attacks could (in theory) succeed breaking implementation with only a single measurement in the attack phase [22].

3 Background

In this section, we start by providing background information about profiling side-channel analysis. Afterward, we discuss the multilayer perceptron algorithm.

3.1 Profiling Side-channel Analysis

Cryptographic devices execute functions that handle both public and secret information. An adversary aims to steal the secret information by exploiting the physical leakage of the device. More precisely, these leakages are caused by the electrical properties of CMOS transistors that store the information in a device. When changing state to store or load information, CMOS consumes energy, which is a behavior that the attacker can observe and link to information that is processed.

Profiling side-channel analysis is an efficient set of methods where one works under the assumption that the attacker is in full control of an exact copy of the targeted device. By estimating leakage profiles for each target value during the profiling step (also known as the training phase), the adversary can classify new traces obtained from the targeted device by computing the probabilities of each target value to match the profile. There are multiple approaches to compute these probabilities, such as template attack [3], stochastic attack [26], multivariate regression model [28], and machine learning models [16,13].

When profiling the leakage, one must choose the appropriate leakage model, which will then result in a certain number of classes (i.e., possible outputs). The first common model is the intermediate value leakage model, which results in 256 classes as we consider the AES cipher that has 8-bit S-box:

$$Y(k) = \texttt{Sbox}[P_i \oplus k].$$

The second common leakage model is the Hamming weight (HW):

$$Y(k) = \texttt{HW}(\texttt{Sbox}[P_i \oplus k]).$$

The Hamming weight leakage model results in 9 classes. Note that the distribution of classes is very imbalanced, which can lead to problems in the classification process [23].

3.2 Multilayer Perceptron

The multilayer perceptron (MLP) is a feed-forward neural network that maps sets of inputs onto sets of appropriate outputs. MLP has multiple layers of nodes (artificial neuron as depicted in Figure 1) in a directed graph, where each layer is fully connected to the next layer. The output of a neuron is a weighted sum of m inputs x_i evaluated through a (nonlinear) activation function A:

$$Output = A(\sum_{i=0}^{m} w_i \cdot x_i). \tag{1}$$

An MLP consists of at least three types of layers: an input layer, an output layer, and one or more hidden layers [5]. If there is more than one hidden layer, the architecture can be already considered as deep learning. In Figure 2, we

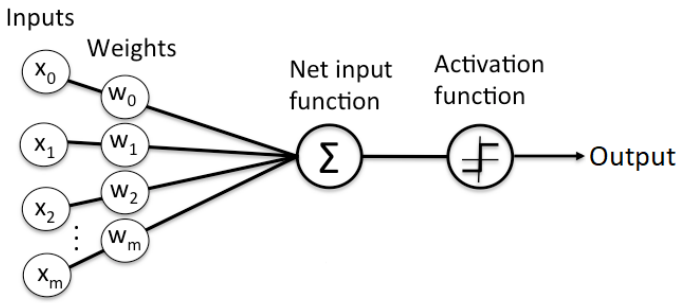


Fig. 1: Artificial neuron model (perceptron). A network structure with m inputs $(x_0, x_1, \ldots x_m)$ connected to a neuron with weights $(w_0, w_1, \ldots, w_m)$ on each connection. The neuron sums all the signals it receives where each signal is multiplied by its associated weights on the connection. The output is passed through an activation function, A, that is usually nonlinear to give the final output.

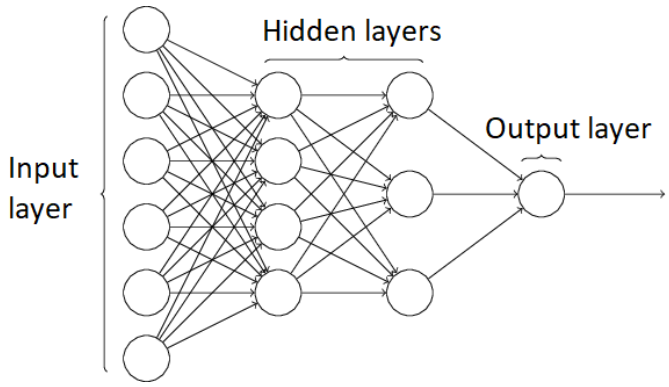


Fig. 2: Multilayer perceptron. The input layer has 6 inputs (features) that are then passed through 2 hidden layers, first one with 4 perceptrons and the second one with 3 perceptrons. Finally, the output layer has one perceptron.

depict an MLP with 2 hidden layers. To train the neural network, the backpropagation algorithm is used, which is a generalization of the least mean squares algorithm in the linear perceptron [9].

The multilayer perceptron has many hyperparameters one can tune but we concentrate our attention on the following ones:

1. The number of hidden layers. The number of hidden layers will define the depth of the algorithm and consequently, how complex relations can the MLP model handle.
2. The number of neurons (perceptrons) per layer. The number of neurons per layer tells us the width of the network and what is the latent space. Interestingly, there exists a well-known result in the machine learning community called the Universal Approximation Theorem that states (very informally) that a feed-forward neural network with a single hidden layer, under some assumptions, can approximate a wide set of continuous functions to any desired non-zero level of error [7].
3. Activation functions. Activation functions are used to convert an input signal to an output signal where if complex functional mappings are needed, one needs to use nonlinear activation functions.

When talking about machine learning algorithms, it is common to differentiate between parameters and hyperparameters. Hyperparameters are all those configuration variables that are external to the model, e.g., the number of hidden layers in a neural network. The parameters are the configuration variables that are internal to the model and whose values can be estimated from data. One example of parameters is the weights in a neural network. Consequently, when talking about tuning a machine learning algorithm, it means tuning its hyperparameters.

4 Experimental Setup

In this section, we first provide information about the datasets we use. Next, we briefly discuss the guessing entropy evaluation metric.

4.1 Datasets

We consider two datasets that we denote ASCAD and AES_RD. Both datasets are protected with countermeasures: the first one with masking and the second one with the random delay.

ASCAD Dataset This dataset implements a first-order Boolean masking countermeasure, which makes it more resilient against side-channel analysis and a widespread SCA protection mechanism. The principle of masking countermeasure is to introduce a random mask on the input of the algorithm to prevent an attacker to find a correlation between intermediate values and the secret key. As the mask is selected at random for every execution of the algorithm, the attacker cannot predict intermediate values. The targeted platform is an

8-bit AVR microcontroller (ATMega8515) running a masked implementation of AES-128. Measurements are made using electromagnetic emanation (EM). The dataset is partitioned into two sets: a profiling/training set of 50 000 traces and a testing/attack set of 10 000 traces. We use the original traces extracted without modification containing a pre-selected window of 700 relevant samples per trace corresponding to round three of a masked AES. We use the unprotected S-box output as a leakage model. The SNR for this dataset is around 0.8 if the mask is known and 0 if the mask is unknown. The trace set is publicly available at `https://github.com/ANSSI-FR/ASCAD/tree/master/ATMEGA_AES_v1/ATM_AES_v1_fixed_key`.

AES_RD Dataset The targeted platform for this dataset is an 8-bit AVR microcontroller (ATmega16) running an implementation of AES-128 protected with random delays as described by Coron and Kizhvatov [6]. This dataset has a random delay countermeasure, which introduces process interruption of random lengths at random times during the execution of the algorithm, introducing misalignment of important features in the collected traces, making the attack more difficult. In this attack, we target the first S-box operation. The data consists of 50 000 traces of 3 500 features each. We use 40,000 traces for training and 10,000 traces for attacking. The SNR has a maximum value of 0.0556. In this attack, we target the first S-box operation. The trace set is publicly available at `https://github.com/ikizhvatov/randomdelays-traces`.

4.2 Evaluation Metric

The capability of a profiling model is inversely proportional to the number of traces an attacker needs to use to recover the key. A common metric in SCA is Guessing Entropy (GE) [27]. GE defines the average position of the correct key guess in the guessing vector. In other words, when considering N attack traces, each of which results in a guessing vector $\mathbf{g} = [g_1, g_2, \ldots, g_{|K|}]$ containing the probabilities of each key guess in the keyspace K, ordered by decreasing probability, the guessing entropy is the average position of correct key candidate among $\mathbf{g}$.

The training and validation accuracy of the training phase of a neural network is often used as the only metrics to asses a good fitting of a model. When all hyperparameters have been evaluated, the best model is selected and an SCA metric is applied only on the best model to asses the power of the analysis. While somewhat related to other SCA metrics, accuracy can be deceiving and irrelevant towards measuring the quality of an attack, because it evaluates only one trace at a time whereas SCA metric takes several traces into account [23].

5 Results

All the implementations considered in the paper are software AES implementations. Depending on the implementation, the leakage might be higher for a

certain representation of the output (e.g., the Hamming weight (HW) when considering software implementation). In the analysis of our attacks, we will provide results with power leakage models of both the S-box output and its HW representation.

It is possible to increase the efficiency of the attack using feature selection or reduction methods (e.g., Pearson, PCA, LDA) to select interesting time samples on the dataset. Besides considering the raw traces (i.e., no pre-processing and/or feature engineering), we apply the Difference-of-Means (DoM) as a feature selection method.

Next, we give a detailed study of the MLP performance concerning 3 different hyperparameters:

- The number of perceptrons, with a fixed number of layers.
- The number of layers in an MLP, with a fixed number of perceptrons.
- The activation function used for the perceptrons in the hidden layers.

As it is not possible to explore the entire hyperparameters space, we study a small range of combinations that we consider representative of common architectures found in the related works and aim to explore the close range of the hyperparameters around successful settings from the literature. Additionally, we aim to show the sensitivity of these results when the neural network uses the same dataset but with a reduced number of features (selected Points of Interest (POIs) with the Difference-of-Mean (DoM) technique). In Table 1, we list all hyperparameter combinations we test. We run our experiments in Keras [4] and we use 200 epochs for training with a learning rate of 0.001.

Table 1: List of evaluated hyperparameters.

Hyperparameter	Range
Activation function	$ReLU, Tanh$
Number of layers	$1, 2, 3, 4, 5, 6$
Number of perceptrons per layer	$10, 20, 30, 40, 50, 100, 150, 200, 250, 300$

From Table 1, we consider $n_{act} * n_l * n_p = 2 * 6 * 10 = 120$ MLP architectures. To give an idea of the span of our exploring hyperparameters space, we estimated the time for computing one forward loop during the training phase of an MLP (n_f) which is on average 2.5 seconds. This time is the observed average time for one forward loop of a batch of 100 traces of ASCAD (700 features) using a single GPU. This time depends on many other parameters including the architecture of the GPU unit and the number of trainable weights of the MLP as well as the hyperparameters chosen for the exploration. Taking into account the above consideration, we can estimate the time to train all MLPs in the exhaustive search (within the ranges we defined): $n\_mlp\_to\_train = n_{act} * \sum_{i=0}^{n_l} (n_p)^i$, which leads to a total number of 5 640 MLP architecture to explore for only one dataset and leakage model.

As information obtained from the hyperparameter tuning is difficult to interpret from a single figure, we give the results for each dataset (ASCAD, AES_RD) in three figures: one figure represents the guessing entropy of the attack for all trained MLP architectures (every combination of hyperparameters). This figure shows the evolution of the attack when considering an increasing number of attack traces. The other figures represent the final guessing entropy (considering that we use the maximum number of attack traces we have at our disposal) of MLPs represented for the two activations functions explored. Each figure gives the guessing entropy for one MLP against the number of layers and the number of perceptrons per layer. These figures can be used to visualize the relation that exists between the hyperparameters and the efficiency of the attack.

5.1 ASCAD

The authors of the ASCAD dataset report the best performance using an MLP with the following hyperparameters: activation function: ReLU, number of layers: 6, number of perceptrons per layer: 200, and number of epochs for training: 200. We use these hyperparameters as a baseline and refer to the pre-trained MLP provided along with the database as MLP_{best}. Note, the authors of the ASCAD dataset do not claim these hyperparameters are the best possible ones, but the best out of those they tested.

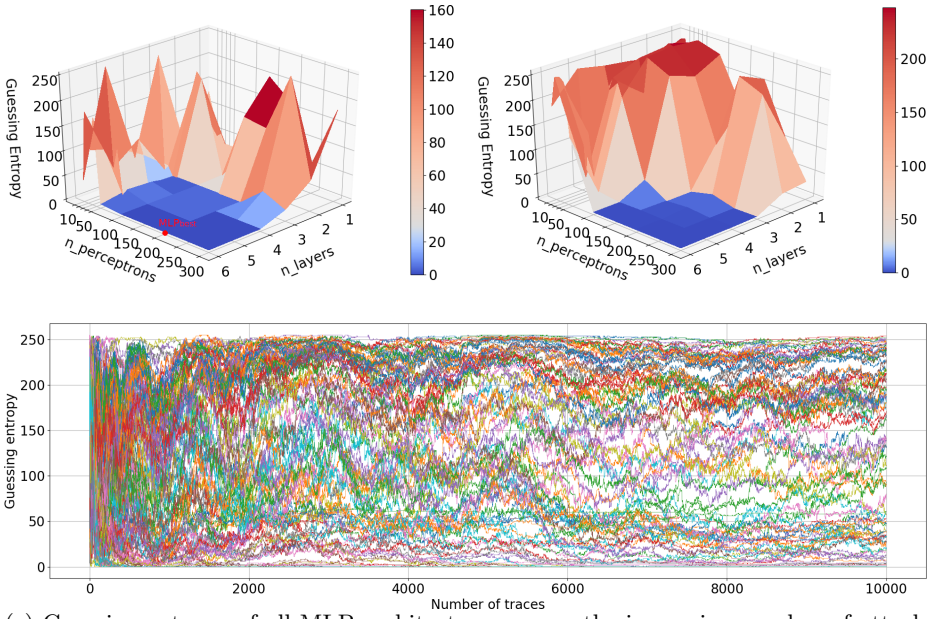
In Figure 3, we depict the influence of all combinations of hyperparameter choices for the ReLU and Tanh activation functions when considering the intermediate value leakage model. We can see that for both choices of activation functions, there exist MLP architectures that reach a guessing entropy of 0 within 1 000 attack traces. Still, the space of MLP architectures that achieve a guessing entropy lower than 30 is wider for ReLU than for Tanh. On the other hand, Tanh seems to behave more stable as we do not observe so many peaks in the landscape, i.e., the behavior is more uniform across a number of settings. We can notice that several MLP architectures with the ReLU activation function and a low number of perceptrons perform well, but only reaches GE above 30.

As it can be observed in Figure 3a, for setting with 200 perceptrons per layer, all MLPs with more than 2 hidden layers converge approximately equally fast to GE of 0. Further, we notice that increasing the number of perceptrons from the MLP_{best} architecture increase GE (i.e., decrease the performance of the attack), and that very similar results can be obtained in the close range of this MLP architecture. Finally, we see from Figure 3c many settings are reaching GE of 0 but also having bad performance even with 10 000 attack traces. We interpret this as expected sensitivity to the hyperparameter tuning.

We now reduce the number of features, i.e., the number of points in traces. To do so, we use Difference-of-Mean between the average trace for each different label/intermediate value. After this procedure, we use the 50 most important features. We then train different MLPs with the traces that have a reduced number of features. We apply the same reduction for the attack dataset and compute guessing entropy and we show the results in Figures 4.

Fig. 3: ASCAD Guessing Entropy for the intermediate leakage model.

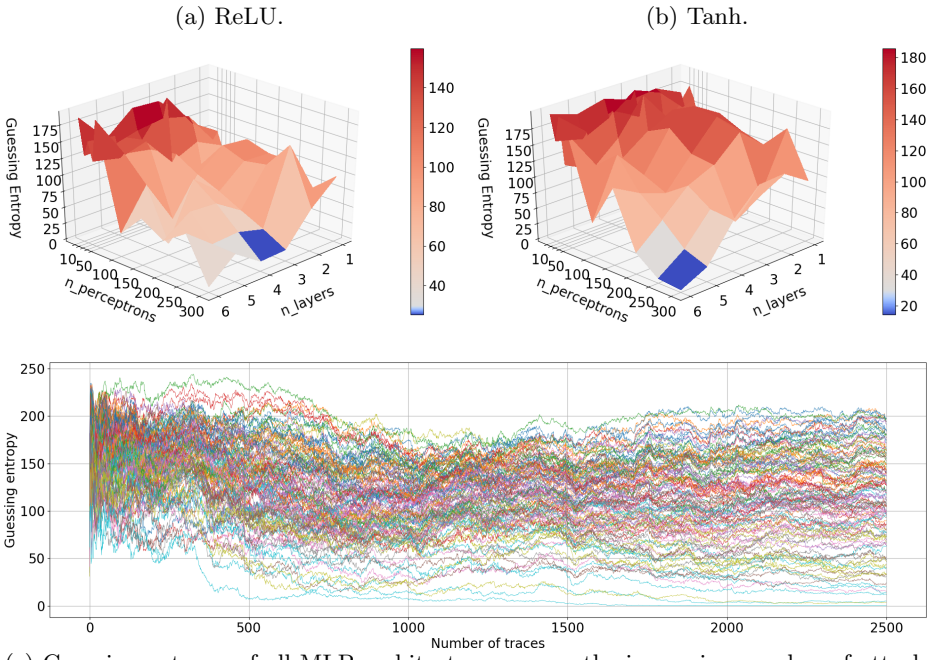
(a) ReLU. (b) Tanh.



(c) Guessing entropy of all MLP architectures versus the increasing number of attack traces.

We can see that the area where GE goes below 30 is now smaller. For the ReLU activation function, this area is located around 3 and 4 layers with 250 and 300 perceptrons per layer. For the Tanh activation function, it is located above 5 layers and 250 perceptrons per layer. Interestingly, we can notice that the highest score in Figure 4a is not obtained for the highest number of layers. For both activation functions, the hyperparameters leading to a good attack performance is shifted toward larger hyperparameter values, which indicates that when starting with less information (i.e., fewer features), we require deeper MLP to be able to reach the same performance level. Figure 4c shows sensitivity to hyperparameter tuning similar to the case with no feature selection.

Fig. 4: ASCAD Guessing Entropy with a reduced number of features and the intermediate leakage model.



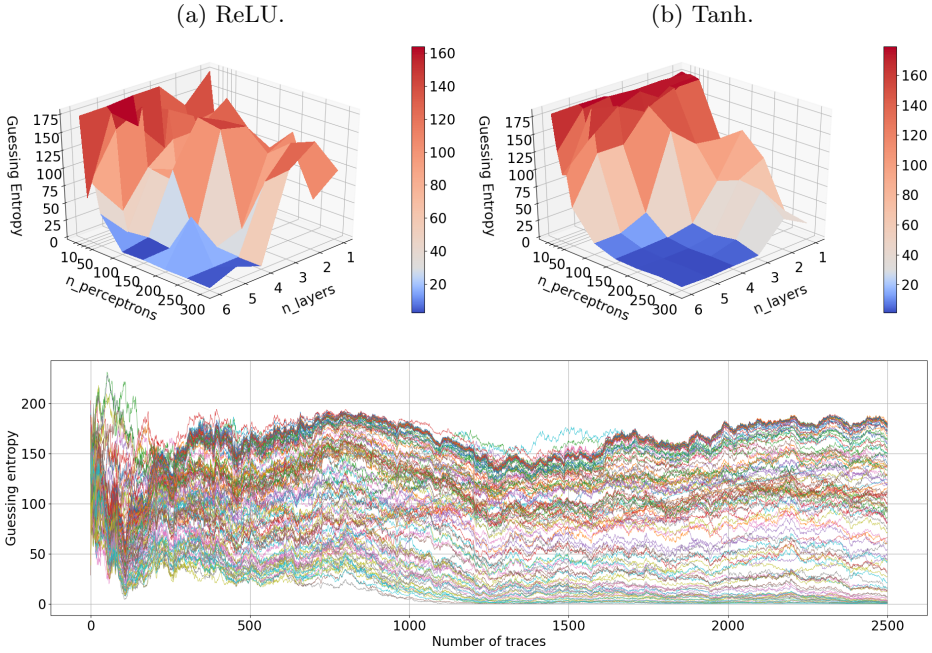
(c) Guessing entropy of all MLP architectures versus the increasing number of attack traces.

Next, we consider the Hamming Weight (HW) leakage model. From Figure 5, we see similar results when compared to the intermediate value leakage model. Still, in Figure 5b, the number of perceptrons per layer has a stronger influence on the guessing entropy than the number of layers. We believe this happens as more perceptrons per layer give more options on how to combine features while deeper networks would contribute to being able to have more complex mappings between input and output, which is not needed for HW (simpler) leakage model.

Interestingly, again we observe than Tanh behaves more stable than the ReLU activation function.

Notice a better behavior for MLP with a small number of layers when compared to the intermediate value scenario. This can be explained by the fact that the classification problem is reduced by having only 9 classes against 256 in the previous setting. We can also see a stable area for a number of perceptrons above 150 and a number of layers above 3. In this area, the hyperparameters choice does not influence anymore the performances of the MLP. Similar to the intermediate value leakage model, the sensitivity to the hyperparameter tuning (Figure 5c) is as expected with many settings reaching top performance, but also many performing poorly.

Fig. 5: ASCAD Guessing Entropy in the Hamming weight leakage model.



(c) Guessing entropy of all MLP architectures versus increasing number of attack traces.

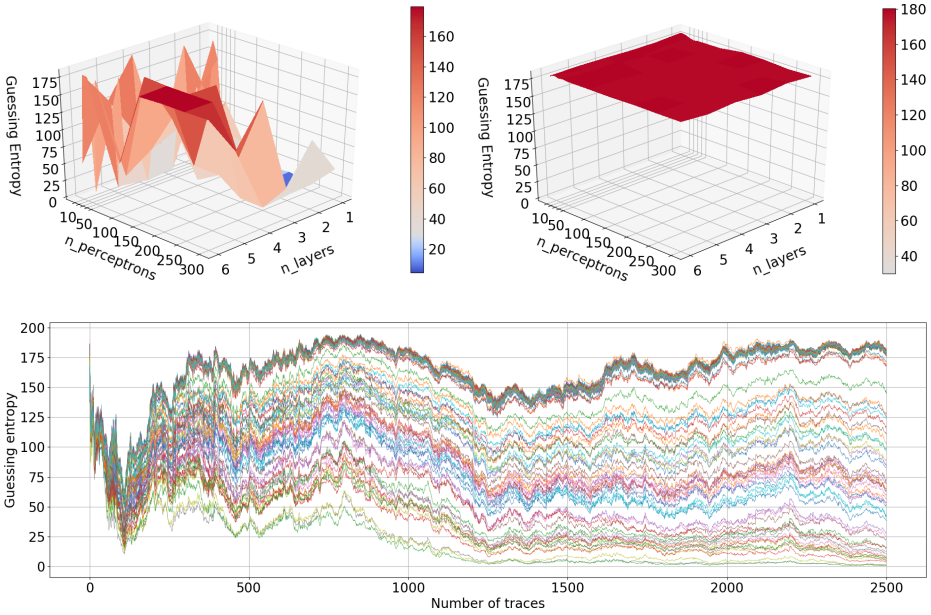
Now, we use the reduced number of feature representation of the dataset and apply the Hamming weight leakage model. We can see in Figure 6c that many MLP architectures differ significantly with a GE spread between 0 and 175. In Figure 6b, we see that no MLP with the Tanh activation function succeed in the attack. Finally, in Figure 6a, we see that MLP with ReLU reaching GE of 0 has only 1 hidden layer and when the number of layer increase, the performance decrease. Based on the ruggedness of the landscape for ReLU, it is clear that the

choice of the number of layers/perceptrons plays a significant role. In Figure 6c, slightly differing from previous cases, we see more groupings in the GE performance. This indicates that a reduced number of features in the weaker leakage model is less expressive so more architectures reach the same performance.

Fig. 6: ASCAD Guessing Entropy with a reduced number of features and the Hamming weight leakage model.

(a) ReLU.

(b) Tanh.



(c) Guessing entropy of all MLP architectures versus increasing number of attack traces.

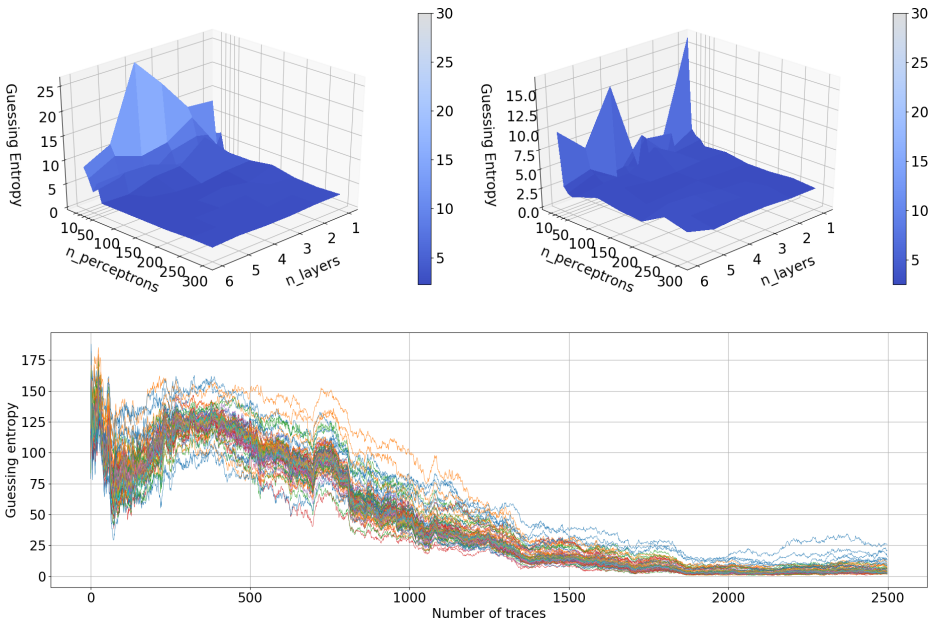
5.2 AES_RD

Given the intermediate value model (Figure 7), all MLP architectures including the smallest ones (1 hidden layer with 10 perceptrons) are capable of reaching GE below 30 within 2 500 attack traces. Increasing the number of layers does not seem to have an impact on the ReLu activation function and for the Tanh activation function, it even seems to increase GE. For both activation functions, increasing the number of perceptrons per layer does make GE decrease. Still, from Figure 7c, we see that regardless of the architecture chosen, all MLP settings converge within the same amount of attack traces. This indicates that there is not enough useful information that larger networks can use and as such, using

Fig. 7: AES_RD Guessing Entropy for the intermediate leakage model.

(a) ReLU.

(b) Tanh.

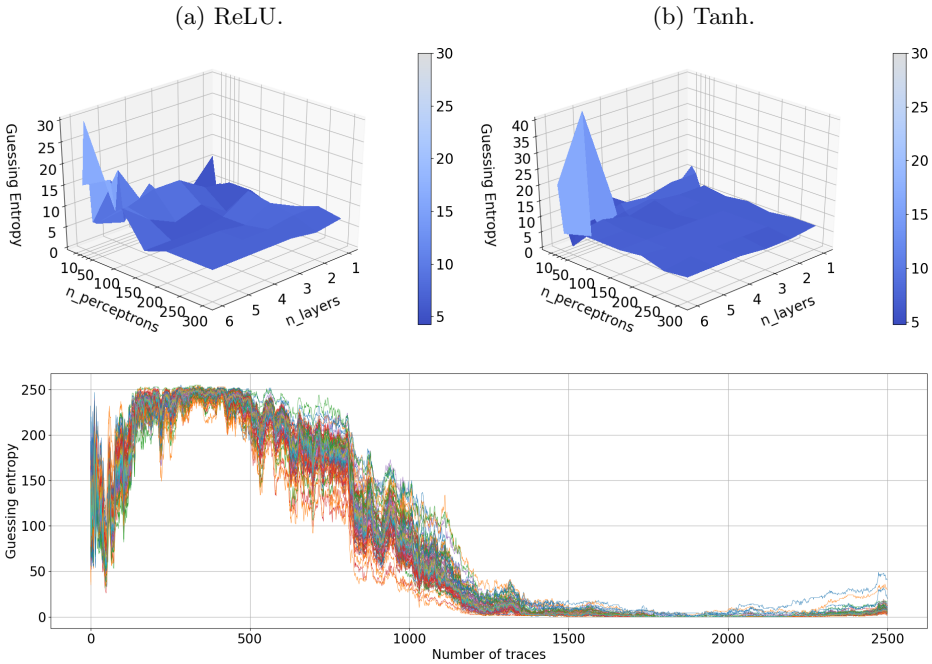


(c) Guessing entropy of all MLP architectures versus increasing number of attack traces.

them brings no performance gain (and as such, there is not much benefit from detailed hyperparameter tuning).

In Figure 8, we observe a similar performance when training MLPs with a reduced number of features for the AES_RD dataset and the intermediate leakage model (containing only 50 selected features). Again, this implies there is no useful information in additional features and that is why MLP cannot perform better even if we use larger/deeper architectures. This is following the expected behavior for the random delay countermeasure as the features are not aligned. Finally, we see that the landscape is smoother for the Tanh activation function than for the ReLU activation function (similar to the ASCAD dataset but differing from the AES_RD with all features).

Fig. 8: AES_RD Guessing Entropy with a reduced number of features and the intermediate leakage model.



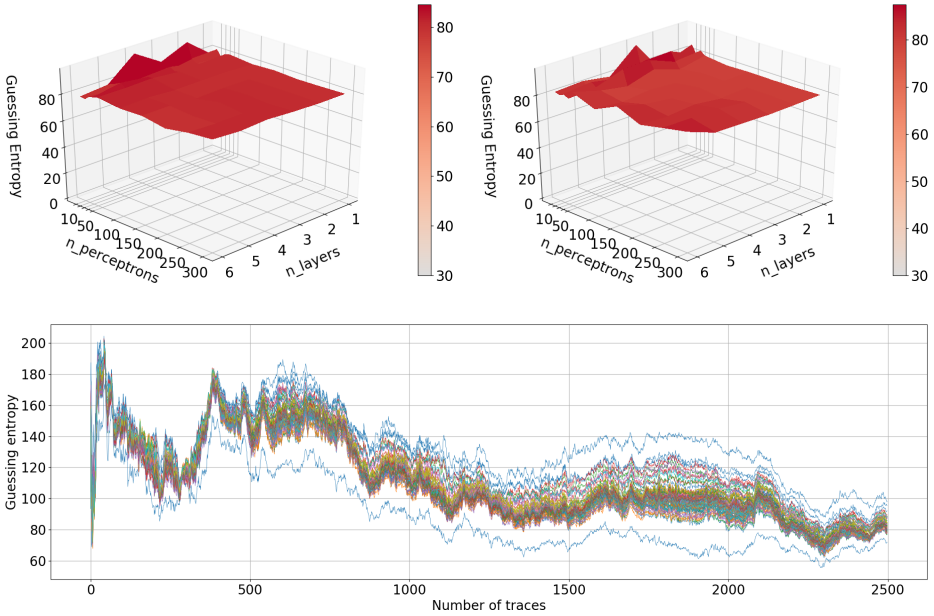
(c) Guessing entropy of all MLP architectures versus increasing number of attack traces.

When considering the HW model for the AES_RD dataset, we notice that even after 2 500 traces, the attack is still unsuccessful. More precisely, in Figure 9, we see that no MLP can reach a GE below 60. Again, this can be expected as before we required around 1 500 traces to succeed in the attack and now we use a "weaker" leakage model that needs more measurements. Interestingly, all architectures behave relatively similar as evident in Figure 9c.

Consequently, as no MLP architecture can succeed in the attack for the HW model on the AES_RD dataset, we cannot conclude whether more layers or perceptrons would result in improved attack performance. Similarly, as the attack is not successful even when considering all the features, we do not give results for a reduced number of features as again, no MLP architecture can reach reasonably good GE.

Fig. 9: AES_RD Guessing Entropy for the Hamming weight leakage model.

(a) ReLU. (b) Tanh.



(c) Guessing entropy of all MLP architectures versus increasing number of attack traces.

6 Discussion

MLP can break the AES_RD implementation (AES with a random delay countermeasure) rather easily. The smallest models (1 layer, 200 perceptrons and 6 layers, 10 perceptrons) share the best outcome of all the models in the comparison. The same results are observed when using only the most important features or when choosing the Hamming weight leakage model. These results could be explained by an important leakage of the secret if the countermeasure would have been turned off. Although the random delays shift the first round S-box operation from the start of the encryption execution, a strong leakage of the

operation handling the secret information is still present. Consequently, using an MLP is enough to overcome this countermeasure. This indicates that the current consensus in the SCA community on MLP performance should change. Indeed, CNNs are considered especially good for random delay countermeasure and MLP for masking countermeasure. We see there is no reason not to consider MLP also good against the random delay countermeasure.

When selecting 50 POIs with a Difference-of-Mean method, the selected points represent only $50/3\,500 \simeq 1\%$ of the original traces in the dataset and the information about the leakage is also reduced. Still, the attack succeeds in the same way, which can be explained because the leakage only comes from the selected POIs.

When considering the ASCAD dataset, we observe that the best score obtained for MLP has the following hyperparameters: 6 number of layers and 200 perceptrons. Still, we see in Figures 3a and 3b that MLP with similar hyperparameters can perform equally good (where the red point represent the result obtain with the architecture of the best MLP MLP_{best} from the ASCAD paper).

When selecting POIs with the Difference-of-Mean method, we can observe that the performance decreases, meaning that the useful information is decreased. This is turn results in attacks not able to recover the full secret key. Still, some MLPs can obtain the secret key in the given number of traces and we observe that both the number of layers and the number of perceptrons influence their performance. Finally, the performance of MLPs with the Hamming weight leakage model gives better performance than for the intermediate value, but the range of hyperparameters that can achieve the best results is smaller than for the intermediate value leakage model.

To answer the question of how challenging is the tuning of MLP hyperparameters, we observe that there is nearly no influence using a (relatively) big or small MLP for AES_RD dataset. When considering the ASCAD dataset with the masking countermeasure, depending on the leakage model considered, the size of the MLP can play a significant role. There, either by increasing the number of perceptrons per layer or the number of layers with a fixed number of perceptrons, we can decrease the guessing entropy.

From the activation function perspective, we see that ReLU behaves somewhat better for the intermediate leakage model when compared to Tanh, i.e., it can reach the top performance with a smaller number of layers/perceptrons. For the Hamming weight scenario, Tanh seems to work better on average but still, ReLU reaches top performance with smaller architectures than Tanh. Finally, we notice that Tanh gives more stable behavior when averaged over all settings, i.e., with the Tanh activation function, the hyperparameter tuning seems to be less sensitive. To conclude, ReLU seems to be the preferred option if going for top performance or using smaller architectures while Tanh should be preferred if stability over a number of scenarios is required.

MLP is (or, at least can be) a deep learning algorithm that has a simple architecture and a few hyperparameters but can show good performance in the side-channel analysis. What is more, it can break implementations protected with

both masking or hiding countermeasures. We see that if there is no sufficient useful input information (as one would expect when dealing with the random delay countermeasure), a reasonable choice is to go with a relatively small architecture. For masked datasets, we see that either the number of perceptrons or the number of layers needs to be large but there the choice of the activation function also plays an important role. Finally, we see that in all considered scenarios, the MLP architectures are not overly sensitive to the hyperparameter choice, i.e., there does not seem to be a strong motivation to run very fine-grained hyperparameter tuning.

7 Conclusions and Future Work

In this paper, we considered the behavior of a multilayer perceptron for profiling side-channel analysis. We investigated two datasets protected with countermeasures and a number of different MLP architectures concerning 3 hyperparameters. Our results clearly show that the input information to the MLP plays a crucial role and if such information is limited, larger/deeper architectures are not needed. On the other hand, if we can provide high quality input information to the MLP, we should also use larger architectures. At the same time, our experiments revealed no need for very fine-grained hyperparameter tuning. While the results for MLP cannot compare with the state-of-the-art results for CNNs, we note that in many cases they are not far. If we additionally factor in that MLP is easier and faster to train, the choice between those two techniques becomes even more difficult to make and should depend on additional goals and constraints. For example, reaching the top performance is the argument for the usage of CNNs but if one requires small yet powerful architecture, a more natural choice seems to be MLP.

In this work, we concentrated on scenarios where each hidden layer has the same number of perceptrons. It would be interesting to investigate the performance of MLP when each layer has a different number of perceptrons. Naturally, this opens a question of what combinations of neurons/layers to consider as one could easily come to hundreds, even thousands of possible settings to explore. Similarly for activation functions, here we consider only the two most popular ones where all hidden layers use the same function. It would be interesting to allow different layers to have different activation functions. Recent experiments showed that MLP is even able to outperform CNNs when considering different devices for training and testing (i.e., the portability case) [1]. We plan to explore the influence of the hyperparameter choice in those scenarios. Finally, as we already mentioned, MLP architectures are usually simpler than CNNs, which should mean there are easier to understand. We aim to explore whether we can design stronger countermeasures against machine learning based-attacks based on how MLP is working.

References

1. Bhasin, S., Chattopadhyay, A., Heuser, A., Jap, D., Picek, S., Shrivastwa, R.R.: Mind the portability: A warriors guide through realistic profiled side-channel analysis. Cryptology ePrint Archive, Report 2019/661 (2019), https://eprint.iacr.org/2019/661
2. Cagli, E., Dumas, C., Prouff, E.: Convolutional Neural Networks with Data Augmentation Against Jitter-Based Countermeasures - Profiling Attacks Without Preprocessing. In: Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings. pp. 45–68 (2017)
3. Chari, S., Rao, J.R., Rohatgi, P.: Template attacks. In: International Workshop on Cryptographic Hardware and Embedded Systems. pp. 13–28. Springer (2002)
4. Chollet, F., et al.: Keras. https://github.com/fchollet/keras (2015)
5. Collobert, R., Bengio, S.: Links Between Perceptrons, MLPs and SVMs. In: Proceedings of the Twenty-first International Conference on Machine Learning. pp. 23–. ICML '04, ACM, New York, NY, USA (2004). https://doi.org/10.1145/1015330.1015415, http://doi.acm.org/10.1145/1015330.1015415
6. Coron, J.S., Kizhvatov, I.: An efficient method for random delay generation in embedded software. In: International Workshop on Cryptographic Hardware and Embedded Systems. pp. 156–170. Springer (2009)
7. Cybenko, G.: Approximation by superpositions of a sigmoidal function. Mathematics of Control, Signals and Systems **2**(4), 303–314 (Dec 1989). https://doi.org/10.1007/BF02551274, https://doi.org/10.1007/BF02551274
8. Gilmore, R., Hanley, N., O'Neill, M.: Neural network based attack on a masked implementation of AES. In: 2015 IEEE International Symposium on Hardware Oriented Security and Trust (HOST). pp. 106–111 (May 2015). https://doi.org/10.1109/HST.2015.7140247
9. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press (2016), http://www.deeplearningbook.org
10. Hettwer, B., Gehrer, S., Güneysu, T.: Profiled power analysis attacks using convolutional neural networks with domain knowledge. In: Cid, C., Jr., M.J.J. (eds.) Selected Areas in Cryptography - SAC 2018 - 25th International Conference, Calgary, AB, Canada, August 15-17, 2018, Revised Selected Papers. Lecture Notes in Computer Science, vol. 11349, pp. 479–498. Springer (2018). https://doi.org/10.1007/978-3-030-10970-7_22, https://doi.org/10.1007/978-3-030-10970-7_22
11. Heuser, A., Picek, S., Guilley, S., Mentens, N.: Lightweight ciphers and their side-channel resilience. IEEE Transactions on Computers **PP**(99), 1–1 (2017). https://doi.org/10.1109/TC.2017.2757921
12. Heuser, A., Picek, S., Guilley, S., Mentens, N.: Side-channel analysis of lightweight ciphers: Does lightweight equal easy? In: Radio Frequency Identification and IoT Security - 12th International Workshop, RFIDSec 2016, Hong Kong, China, November 30 - December 2, 2016, Revised Selected Papers. pp. 91–104 (2016)
13. 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, https://tches.iacr.org/index.php/TCHES/article/view/8292

14. Kocher, P.C.: Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In: Proceedings of CRYPTO'96. LNCS, vol. 1109, pp. 104–113. Springer-Verlag (1996)
15. Kocher, P.C., Jaffe, J., Jun, B.: Differential power analysis. In: Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology. pp. 388–397. CRYPTO '99, Springer-Verlag, London, UK, UK (1999), http://dl.acm.org/citation.cfm?id=646764.703989
16. Maghrebi, H., Portigliatti, T., Prouff, E.: Breaking cryptographic implementations using deep learning techniques. In: Security, Privacy, and Applied Cryptography Engineering - 6th International Conference, SPACE 2016, Hyderabad, India, December 14-18, 2016, Proceedings. pp. 3–26 (2016)
17. Mangard, S., Oswald, E., Popp, T.: Power Analysis Attacks: Revealing the Secrets of Smart Cards. Springer (December 2006), ISBN 0-387-30857-1, http://www.dpabook.org/
18. Martinasek, Z., Zeman, V.: Innovative method of the power analysis. Radioengineering **22**(2) (2013)
19. Martinasek, Z., Hajny, J., Malina, L.: Optimization of power analysis using neural network. In: Francillon, A., Rohatgi, P. (eds.) Smart Card Research and Advanced Applications. pp. 94–107. Springer International Publishing, Cham (2014)
20. Pfeifer, C., Haddad, P.: Spread: a new layer for profiled deep-learning side-channel attacks. Cryptology ePrint Archive, Report 2018/880 (2018), https://eprint.iacr.org/2018/880
21. Picek, S., Heuser, A., Alippi, C., Regazzoni, F.: When theory meets practice: A framework for robust profiled side-channel analysis. Cryptology ePrint Archive, Report 2018/1123 (2018), https://eprint.iacr.org/2018/1123
22. Picek, S., Heuser, A., Guilley, S.: Profiling side-channel analysis in the restricted attacker framework. Cryptology ePrint Archive, Report 2019/168 (2019), https://eprint.iacr.org/2019/168
23. 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 Trans. Cryptogr. Hardw. Embed. Syst. **2019**(1), 209–237 (2019). https://doi.org/10.13154/tches.v2019.i1.209-237, https://doi.org/10.13154/tches.v2019.i1.209-237
24. Picek, S., Samiotis, I.P., Kim, J., Heuser, A., Bhasin, S., Legay, A.: On the performance of convolutional neural networks for side-channel analysis. In: Chattopadhyay, A., Rebeiro, C., Yarom, Y. (eds.) Security, Privacy, and Applied Cryptography Engineering. pp. 157–176. Springer International Publishing, Cham (2018)
25. Quisquater, J.J., Samyde, D.: Electromagnetic analysis (ema): Measures and counter-measures for smart cards. In: Attali, I., Jensen, T. (eds.) Smart Card Programming and Security. pp. 200–210. Springer Berlin Heidelberg, Berlin, Heidelberg (2001)
26. Schindler, W., Lemke, K., Paar, C.: A stochastic model for differential side channel cryptanalysis. In: International Workshop on Cryptographic Hardware and Embedded Systems. pp. 30–46. Springer (2005)
27. Standaert, F.X., Malkin, T., Yung, M.: A Unified Framework for the Analysis of Side-Channel Key Recovery Attacks. In: EUROCRYPT. LNCS, vol. 5479, pp. 443–461. Springer (April 26-30 2009), Cologne, Germany
28. Sugawara, T., Homma, N., Aoki, T., Satoh, A.: Profiling attack using multivariate regression analysis. IEICE Electronics Express **7**(15), 1139–1144 (2010)

29. Timon, B.: Non-profiled deep learning-based side-channel attacks with sensitivity analysis. IACR Transactions on Cryptographic Hardware and Embedded Systems **2019**(2), 107–131 (Feb 2019). https://doi.org/10.13154/tches.v2019.i2.107-131, https://tches.iacr.org/index.php/TCHES/article/view/7387
30. Yang, S., Zhou, Y., Liu, J., Chen, D.: Back propagation neural network based leakage characterization for practical security analysis of cryptographic implementations. In: Kim, H. (ed.) Information Security and Cryptology - ICISC 2011. pp. 169–185. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)