On the Importance of Pooling Layer Tuning for Profiling Side-Channel Analysis

Wu, Lichao; Perin, Guilherme

DOI
10.1007/978-3-030-81645-2_8

Publication date
2021

Document Version
Final published version

Published in
Applied Cryptography and Network Security Workshops - ACNS 2021 Satellite Workshops, AIBlock, AIHWS, AIoTS, CIMSS, Cloud S and P, SCI, SecMT, and SiMLA, 2021, Proceedings

Citation (APA)
Wu, L., & Perin, G. (2021). On the Importance of Pooling Layer Tuning for Profiling Side-Channel Analysis. In J. Zhou, C. M. Ahmed, L. Batina, S. Chattopadhyay, O. Gadyatskaya, C. Jin, J. Lin, E. Losiouk, B. Luo, S. Majumdar, M. Maniatakos, D. Mashima, W. Meng, S. Picek, M. Shimaoka, C. Su, & C. Wang (Eds.), *Applied Cryptography and Network Security Workshops - ACNS 2021 Satellite Workshops, AIBlock, AIHWS, AIoTS, CIMSS, Cloud S and P, SCI, SecMT, and SiMLA, 2021, Proceedings* (pp. 114-132). (Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics); Vol. 12809 LNCS). Springer. https://doi.org/10.1007/978-3-030-81645-2_8

On the Importance of Pooling Layer Tuning for Profiling Side-Channel Analysis

Lichao Wu(✉) and Guilherme Perin

Delft University of Technology, Delft, The Netherlands

Abstract. In recent years, the advent of deep neural networks opened new perspectives for security evaluations with side-channel analysis. Profiling attacks now benefit from capabilities offered by convolutional neural networks, such as dimensionality reduction and the inherent ability to reduce the trace desynchronization effects. These neural networks contain at least three types of layers: convolutional, pooling, and dense layers. Although the definition of pooling layers causes a large impact on neural network performance, a study on pooling hyperparameters effect on side-channel analysis is still not provided in the academic community. This paper provides extensive experimental results to demonstrate how pooling layer types and pooling stride and size affect the profiling attack performance with convolutional neural networks. Additionally, we demonstrate that pooling hyperparameters can be larger than usually used in related works and still keep good performance for profiling attacks on specific datasets.

Keywords: Side-channel analysis · Deep learning · Convolutional neural networks · Pooling

1 Introduction

The processing of confidential and secret information in embedded or electronic devices, in general, requires protection against different types of physical attacks. Encryption methods implement various algorithms to provide data protection for sensitive information, including cryptographic keys. An algorithm that proved to be mathematically secure is not necessarily implementation-secure. Side-channel analysis (SCA) is a class of non-invasive attacks where an adversary can record the unintended leakages, such as electromagnetic (EM) radiation [20] or power dissipation [9], and use those leakages to obtain secret information [14].

SCA can be divided into two categories based on the attack setting or security evaluation purpose (e.g., chip certification or security assessment). When an attacker can only access physical leakages captured on the target device, a non-profiled SCA, such as differential power analysis (DPA) [9], correlation power analysis (CPA) [2], and mutual information analysis (MIA) [5], could be used to retrieve the secret information. On the other hand, profiling SCA assumes an adversary with full control of a clone device (i.e., by changing the key or installing malicious software) that is identical to the target device. On that device, the attacker can profile the side-channel leakages. This allows the adversary to learn statistics from leakages and build profiling models. The commonly used methods include template attack [4] and supervised machine learning-based attacks [3,8, 13,19].

Supervised machine learning-based attacks have drawn great attention within the SCA community in recent years due to their effectiveness in breaking targets and high applicability to different attack scenarios. Among different types of neural networks, convolutional neural networks (CNNs) are the most adopted method in coping with countermeasures due to their spatial invariance property [3,8], making these models appropriate to bypass countermeasures such as noise and side-channel trace desynchronization. While profiling models based on deep neural networks actively threaten the security of cryptographic devices in profiled settings, there are still severe limitations and unknowns.

Neural network hyperparameter selection is one of the biggest obstacles. Taking CNNs as an example, they usually consist of three types of layers (convolution layer, pooling layer, and dense layer), where each layer has at least two configurable hyperparameters. When an attacker tries to enhance the network capability by applying more layers, the hyperparameters' combinations increase exponentially. Although some researchers are trying to set general design rules [25,27] or applying neural architecture search to find the best-performing network automatically [21,26], the results are far from definitive. Indeed, the generality of such hyperparameter tuning methods is usually dataset-specific, but they demonstrate that deep neural networks are powerful methods that can be tailored to different datasets.

This paper focuses on the pooling layer of CNNs, which is, to the best of our knowledge, an analysis not done before. We experimentally investigate the influence of a pooling layer's hyperparameters variation on the attack performance. To achieve this, we use two models, one with a single pooling layer and the other with multiple pooling layers. The former is used to target an unprotected dataset; the latter is optimized for two datasets containing different AES implementations protected with masking countermeasure. Our results clearly show that the type of pooling layer should be selected based on the neural network depth and the number of input features. We also give guidelines on how to choose the hyperparameters in different cases. Finally, our results show that pooling hyperparameter tuning is important and can result in significantly different attack performance even when not considering other layers or hyperparameters.

2 Preliminaries

2.1 Notation

We use calligraphic letters $\mathcal{X}$ to represent sets. The upper-case letters (X) represent random variables and random vectors $\mathbf{X}$ over $\mathcal{X}$. The realizations of X and $\mathbf{X}$ are represented by lower-case letters x and $\mathbf{x}$, respectively.

A dataset $\mathbf{T}$ constitutes a collection of side-channel traces (measurements) $\mathbf{t}_i$ associated with an input value (plaintext or ciphertext) $\mathbf{d}_i$ and a key candidate $\mathbf{k}_i$ ($k \in \mathcal{K}$ where k^* is the correct key). As common in deep learning-based SCA, we divide the dataset into three parts: a profiling set of N traces, a validation set of V traces, and an attack set of Q traces. In terms of a deep learning-based profiling model, we denote the vector of learnable parameters with $\boldsymbol{\theta}$ and the set of hyperparameters defining the profiling model f with $\mathcal{H}$.

2.2 Deep-Learning Based Profiling Side-Channel Analysis

The goal of supervised machine learning is to learn a function f mapping an input to the output ($f : \mathcal{X} \rightarrow Y$)). To accomplish this, the function f uses examples of input-output pairs. In supervised learning for profiling SCA, the input-output pairs are represented by leakage traces and the corresponding intermediate data. The profiling stage is equivalent to the training phase in supervised learning, while the attack phase is equivalent to testing in supervised learning. Formally, the profiling SCA is executed in the following stages:

– Profiling stage: learn $\boldsymbol{\theta}'$ that minimizes the empirical risk represented by a loss function L on a profiling set of size N.
– Attack stage: predict the classes $y(x_1, k^*), \ldots, y(x_Q, k^*)$, where k^* represents the secret (unknown) key on the device under the attack.

By applying attack traces to the profiling models, probabilistic deep learning algorithms output a matrix of probabilities P of size $Q \times c$ (where c denotes the number of output classes). Each probability value denotes how likely a certain measurement should be classified into a specific class v (thus, $\mathbf{p}_{i,v}$ represents the probability that the class v is predicted). The class v is obtained from the key and input through a cryptographic function and a leakage model l. Every row of the matrix P is a vector of all class probabilities for a specific trace $\mathbf{x}_i$ ($\sum_v^c \mathbf{p}_{i,v} = 1, \forall i$). The probability $S(k)$ for any key byte candidate k is the maximum log-likelihood distinguisher:

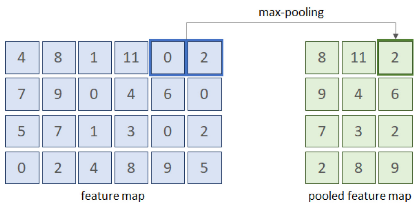
$$S(k) = \sum_{i=1}^{Q} \log(\mathbf{p}_{i,v}). \tag{1}$$

As common in SCA, an adversary aims to obtain the secret key k^* with the minimum attack effort. To evaluate this effort, it is common to use a metric like guessing entropy (GE) [23] that represents the average position of k^* in a key guessing vector $\mathbf{g} = [g_1, g_2, \ldots, g_{|\mathcal{K}|}]$. Here, g_1 represents the most likely key candidate, while $g_{|\mathcal{K}|}$ represents the least likely key candidate. Note that this represents a significant difference from the machine learning settings where one would commonly consider validation accuracy as a metric of success.

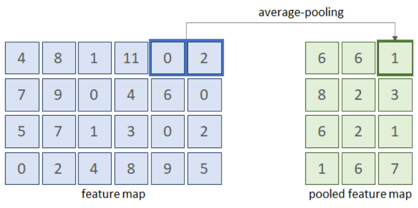
2.3 Convolutional Neural Networks

Convolutional neural networks (CNNs) are widely used neural networks in many domains, including SCA. They commonly consist of three types of layers:

- **Convolutional layer**: this layer computes neurons' output connected to local regions in the input, each computing a dot product between their weights and a small region they are connected to in the input volume.
- **Pooling layer**: this layer aims at decreasing the number of extracted features by performing a down-sampling operation along the spatial dimensions. It is common to consider convolution and pooling layers to form a convolution block. Two main types of pooling layers are considered in this paper: *average-pooling* and *max-pooling*. Average-pooling layers perform the average of a pooling block concerning the *pooling size* (i.e., the number of elements covered with a single pooling operation). Max-pooling layers return the maximum element from a block concerning pooling size. Figure 1 illustrates the different types of pooling operations over a feature map (output of a convolution layer). As we see in this example, the selection of the pooling type can be crucial for the model performance, as each type of pooling returns different results. *Pooling stride* refers to the pooling step over the feature map.
- **Fully-connected layer**: the dense layers are normally applied after convolution layers and pooling layers. The goal of this layer is to compute either the hidden activations or the class scores.



(a) Max-pooling operation with pooling size of 1x2 and pooling stride of 2.



(b) Average-pooling operation with pooling size of 1x2 and pooling stride of 2.

Fig. 1. A demonstration of max-pooling and average-pooling operations. The feature map is reduced from 4×6 to 4×3 after pooling.

2.4 Datasets

ChipWhisperer Dataset. The Chipwhisperer dataset is designed to evaluate various algorithms by providing a standard comparison base [15]. The dataset we consider contains 10 000 side-channel power traces measured by the ChipWhisperer CW308 target running an unprotected AES-128 implementation. Each trace contains 5 000 sample points (features). In our experiment, we use 7 500 traces for profiling and 2 000 traces for the validation. We use key byte two as the target secret data.

ASCAD Datasets. ASCAD datasets represent a common target for profiling SCA as they contain measurements protected with masking and settings with fixed or random keys [1]. The ASCAD dataset is the measurements from an 8-bit AVR microcontroller running a masked AES-128 implementation. Currently, there are two versions of this database: one that uses a fixed key for both profiling and attack dataset, and the other one with random keys in the profiling set. The datasets are available at https://github.com/ANSSI-FR/ASCAD.

The first dataset version has a fixed key, and it consists of 50 000 traces for profiling and 10 000 for the attack. From 50 000 traces in the profiling set, we use 45 000 traces for profiling and 5 000 for validation. Each trace has 700 features (preselected window corresponding to the processing of key byte 3, the first masked key byte). We denote this dataset as ASCAD_f.

The second version has random keys, with 200 000 traces for profiling and 100 000 for the attack. We use 45 000 traces for profiling and 5 000 traces from the attack set for validation (note that the attack set has a fixed but a different key from the profiling set). Each trace has 1 400 features (preselected window corresponding to the processing of key byte 3, the first masked key byte). We denote this dataset as ASCAD_r.

3 Related Works

The profiling SCA can be considered as a classification task on one-dimensional data. In general, the attacker's goals are:

– to classify the traces containing unknown but fixed information (i.e., encryption subkeys),
– by using the classification results and knowledge about the plaintexts/ciphertexts, retrieve the secret information.

From the information-theoretic point of view, template attack (TA) [4] represents the most powerful profiling SCA if the theoretical model and reality fully match. There, one uses the probability density function (PDF) as templates to perform the attack. In an ideal (but unrealistic) case where the attacker has an unlimited number of traces and the noise follows the Gaussian distribution, TA can reach its full attack capability [11].

In terms of machine learning-based profiling SCA, various approaches, such as random forest [10] and support vector machines [7] have been adopted first. More recently, multilayer perceptron (MLP) [6,18] and convolutional neural networks (CNN) [3,8,13] emerged as more powerful approaches.

Specifically, CNNs demonstrated to be capable of coping with various countermeasures due to their spatial invariance property [3,8]. Thus, they became one of the most powerful approaches for deep learning-based SCAs. However, a CNN optimized for one dataset is not necessarily applicable to other datasets, thus raising difficulties in implementing such attacks. To allow customization and optimization of CNN designs, Zaid et al. proposed a methodology to select hyperparameters related to the size of layers in CNNs [27]. This work is further improved by Wouters et al. [25] with the help of data standardization. In terms of neural architecture search, Bayesian optimization is adopted by Wu et al. to find optimal hyperparameters for MLP and CNNs [26]. Rijsdijk et al. used reinforcement learning to design CNNs that show strong attack performance with a small number of trainable parameters [22]. Several works consider tuning of specific CNN hyperparameters: Li et al. investigated the influence of weight initialization techniques [12] while Perin and Picek considered different optimizers [16].

4 Experimental Setup

In this section, we present our strategy to evaluate the performance of two types of commonly-used pooling layers: average-pooling and max-pooling. The analysis is conducted on three publicly available datasets described in Sect. 2.4. The default CNN models used to test the pooling layer are described in Table 1. Specifically, $CNN_{chipwhisperer}$ is used to attack the Chipwhisperer dataset. The ASCAD fixed key (ASCAD_f) and ASCAD random keys datasets (ASCAD_r) are profiled with CNN_{ascad} [1]. We consider only the HW leakage model as the conclusions drawn from the pooling layer with one leakage model can be easily extended to other leakage models. Also, considering the related work, the HW leakage model performs well for the considered datasets [22,26]. In terms of hyperparameters, we show the number of filters in the table for convolution layers. The convolution stride is set to 11 for both models following the network design from [1]. Pooling layers follow each convolution layer, and the pooling size and stride are set to two by default. For both models, *ReLU* is used as the activation function. The optimizer is *RMSProb* with a learning rate of 1e-5.

Table 1. CNN architectures used in the experiments.

Test models	Convolution layer	Pooling layer	Dense layer
$CNN_{chipwhisper}$	Conv(8)	avg(2, 2)	128 * 2
CNN_{ascad}	Conv(64, 128, 256, 512, 512)	avg(2, 2)*5	4 096 * 2

To evaluate the profiling attack performance, we consider three evaluation metrics:

– Guessing Entropy (GE): the averaged correct key rank after applying the maximum number of attack traces.
– T_{GE0}: the number of traces required to reach GE equal to zero.
– ACC: the classification accuracy on the validation traces.

GE aims at evaluating the key recovery capacity of trained neural networks by setting a limited number of attack traces. The second metric T_{GE0} is designed for cases that the models require few traces to retrieve the secret key. In this case, even if GE equals zero for different settings, we can better estimate the attack performance by evaluating the number of attack traces to reach it. For the *ACC* metric, although related works indicate a low correlation between validation accuracy and success of an attack [17], a higher validation accuracy could still mean a lower *GE* [21,26]. Therefore, the validation accuracy is also taken into consideration.

In the experimental results, we first investigate the influence of data standardization (by zeroing the mean and scaling to unit variance) on the attack performance for the ChipWhisperer dataset. Then, we perform extensive analysis towards the impact of two main configurable hyperparameters: pooling size and pooling stride, within a pooling layer with different evaluation metrics. Finally, we vary the pooling settings in different layers to understand the correlation between the pooling hyperparameter variation and layer depth.

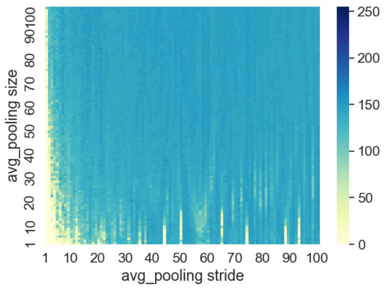
5 Experimental Results

The experiments start with ChipWhisperer as this dataset is easily breakable even with a small CNN architecture. The required time to train a CNN model for this dataset is relatively low, and, therefore, we can tune the model's hyperparameter with smaller steps and a larger range. In terms of the evaluation aspects, with the $CNN_{chipwhisperer}$ specified in Table 1, we focus on tuning the pooling size and stride of the only available pooling layer. With such an analysis, we aim to understand the pooling hyperparameters' influence on the general performance of the model. Here, we experiment with both average-pooling and max-pooling methods by setting the range for pooling size and stride from 1 to 100 with a step of 1 and test all combinations (10 000 combinations in total). Besides, we investigate the link between the data standardization and the pooling layer's hyperparameters selection. As such, the experiments are performed with two versions of a dataset: original (no preprocessing) and standardized (forcing the amplitude ranges from −1 to 1).

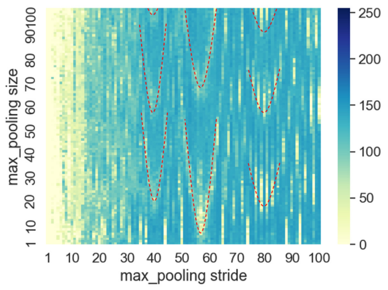
CNN_{ascad} is used as the profiling model for standardized ASCAD_f and ASCAD_r. Compared with $CNN_{chipwhisperer}$, this model's complexity is increased to overcome the masking countermeasure. Note there are five pooling layers in the CNN_{ascad} model. When perturbing all pooling layers simultaneously, the variation range of the pooling layer is limited. Therefore, we only focus on varying the hyperparameters of the first and the last pooling layers. Due to the traces length differences, for ASCAD_f, we tune the pooling hyperparameters ranging from 1 to 20, while for ASCAD_r, we double this range (1 to 40). The step equals one for both datasets.

5.1 Case Study: The ChipWhisperer Dataset

The results for *GE* are shown in Fig. 2. Since *GE* remain zero for all pooling layer's hyperparameter combinations (pooling stride and pooling size) when attacking the standardized dataset, we only present the *GE* value for the original dataset. As mentioned, 2 000 traces are used for the validation. First, we can conclude that the data standardization increases the model's resilience towards the pooling layers' hyperparameter variation. As shown in Fig. 2, for both average- and max-pooling, the attack model is more sensitive to the pooling stride variation. Indeed, a larger pooling stride misses some critical features outputted by the previous convolution layer, finally causing degradation of the attack performance. However, we notice that there are several cases where a large pooling stride can achieve outstanding attack performance. Meanwhile, a large pooling stride can effectively reduce the number of outputted features, leading to a smaller model. This observation indicates the possibility of reducing the network size by using a large pooling stride and having a good understanding of the leakage measurements.



(a) GE: original dataset with average-pooling (min: 0; max: 150)

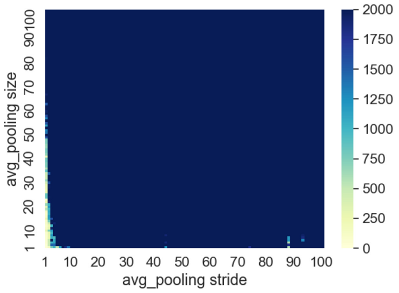


(b) GE: original dataset with max-pooling (min: 0; max: 151)

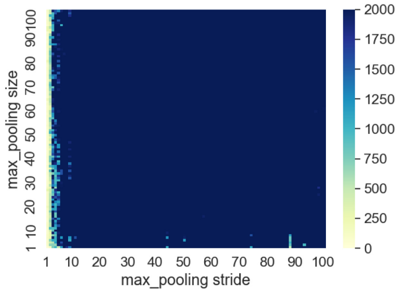
Fig. 2. GE for the original/standardized dataset with average-/max-pooling layer for the HW leakage model on ChipWhisperer.

Interestingly, when attacking the original dataset, the model equipped with the max-pooling layer performs better than the one with the average-pooling layer in general. Specifically, 97% of the average-pooling setting combinations lead to GE value larger than 50, while this value decreases to 85% when applying max-pooling. Additionally, when applying larger pooling size and pooling stride, max-pooling seems a better choice for a successful attack (GE converges or even decreases to zero). Simultaneously, we observe V-shaped patterns (e.g., at max-pooling stride: 57, 80) that occur periodically. The corresponding patterns are also marked by a red dashed line in Fig. 2b. A possible explanation could be that these (large) pooling hyperparameters accidentally cover the leakages appearing in specific locations. However, these critical features are most likely to be skipped, considering many unsuccessful setting combinations. This observation points out the importance of the leakage characterization: if an evaluator understands leakage positions (points of interest), he can confidently decrease the complexity of the attack model by increasing the stride of the pooling layer to a proper value. Similar conclusion is also drawn in [24].

Figure 3 provides results when evaluating the number of traces required to reach GE equal to zero (T_{GE0}). Since GE converges to zero with only a single trace with the standardized dataset, we only show the results attacking the original dataset in Fig. 3. Similar to the observation with the GE metric, the max-pooling layer seems more robust to the pooling size variation when the pooling stride is small.



(a) T_{GE0}: original dataset with average-pooling (min: 31; max: >2 000)

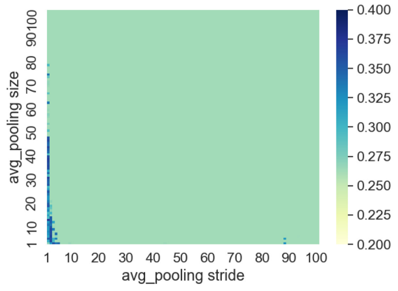


(b) T_{GE0}: original dataset with max-pooling (min: 15; max: >2 000)

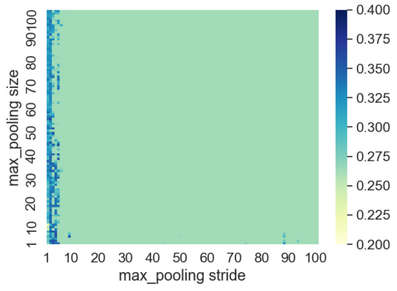
Fig. 3. T_{GE0} for the original/standardized dataset with average-/max-pooling layer for the HW leakage model on ChipWhisperer.

Finally, we analyze the attack performance with each hyperparameter combination with ACC. As shown in Fig. 4, aligned with the previous observation, attacks on the original dataset lead to low ACC, while for the standardized dataset, the accuracy is higher. When comparing the max-pooling and average-pooling layers, the former performs better, as it could lead to high ACC with more pooling setting combinations. Note that most of the ACC values in Figs. 4a and 4b are 0.263, which equals the number of traces labeled as the Hamming weight four (526) divided by the total number of validation traces (2 000). Thus, we conclude that the model is strongly influenced by the class imbalance problem [17] with the original dataset. Data standardization reduces the dominance of the feature in the biggest cluster and decrease the occurrence of overfitting.

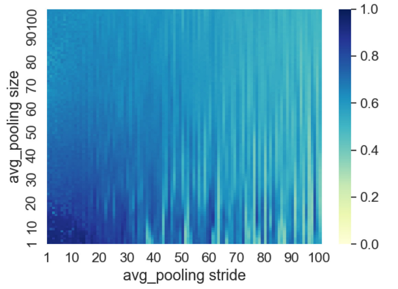
Utilizing the observations for the Chipwhisperer dataset, we postulate that the dataset standardization increases the attack efficiency. Simultaneously, it dramatically increases the model's resilience towards the variation of the pooling layer's hyperparameters. Therefore, for the ASCAD datasets, we only attack the standardized versions of the datasets.



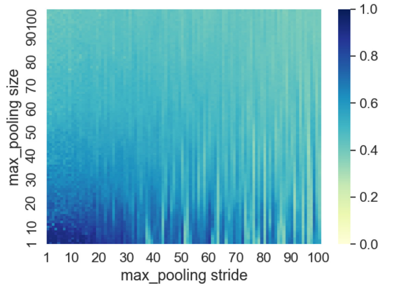
(a) *ACC*: original dataset with average-pooling (min: 0.263; max: 0.373)



(b) *ACC*: original dataset with max-pooling (min: 0.228; max: 0.358)



(c) *ACC*: standardized dataset with average-pooling (min: 0.339; max: 0.945)

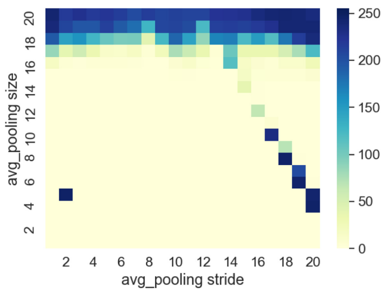


(d) *ACC*: standardized dataset with max-pooling (min: 0.322; max: 0.942)

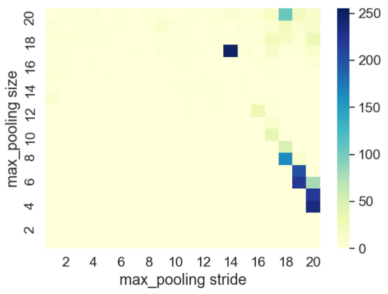
Fig. 4. Accuracy for the original/standardized dataset with average-/max-pooling layer for the HW leakage model on ChipWhisperer.

5.2 ASCAD with a Fixed Key (ASCAD_f)

First, we evaluate the attack performance of each setting in combination with the *GE* metric. The results are shown in Fig. 5. Here, we omit the tuning results for the first pooling layer because of the constant *GE* value (zero) for all setting combinations. On the other hand, when tuning the last pooling layer, the average-pooling method provides inferior performance with a large pooling size. When going to a larger pooling stride, although not so obvious, the models applying both the average- and max-pooling layers method on the last layer have reduced attack performance. For the average-pooling method, a larger pooling size could lead to these critical features being 'averaged' by other less relevant features, thus degrading the attack performance. For the max-pooling method, the unique features can be picked up even with a larger pooling size. Interestingly, we see a 'slash line' on the right part of the figure for both pooling methods. One possible reason could be that with these pooling settings, the critical features are completely missed.



(a) *GE*: tuning the last average-pooling layer (min: 0; max: 248)

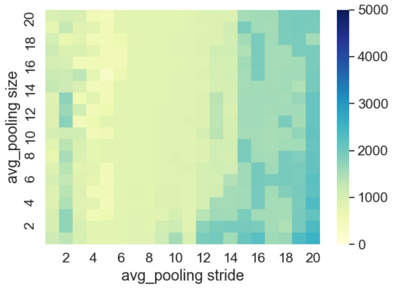


(b) *GE*: tuning the last max-pooling layer (min: 0; max: 248)

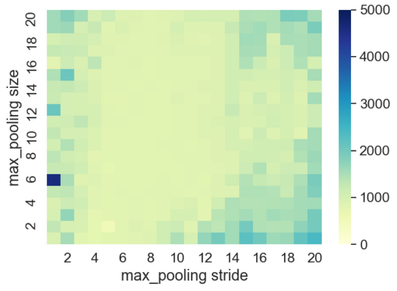
Fig. 5. *GE* for the standardized dataset with average-/max -pooling layer for the HW leakage model on ASCAD_f.

When analyzing the results with T_{GE0} (Fig. 6), some unique patterns can be observed even when tuning the first pooling layer. From Figs. 6a and 6b, we confirm that changing the pooling stride causes greater variation of T_{GE0} than the pooling size for both average-pooling and max-pooling methods. A possible reason could be that the features are still location-dependent after sampling by the first convolution layer. A smaller pooling stride could support capturing these important features. Meanwhile, comparing the results for average- and max-pooling, the latter method seems to enable more pooling settings with low-value T_{GE0}, which is aligned with the conclusion made in Fig. 5. Indeed, when counting the number of setting combinations that lead to T_{GE0} greater than 5 000, the values are 118 and 70 for the averaging-pooling and max-pooling method, respectively. Besides, when comparing Figs. 6a and 6c or Figs. 6b and 6d, the corresponding patterns seems to be rotated for 90 degrees. One explanation could be that the leakages in the deeper layers tend to distribute uniformly across the features. Thus, the selection of the pooling stride becomes less important than the pooling size.

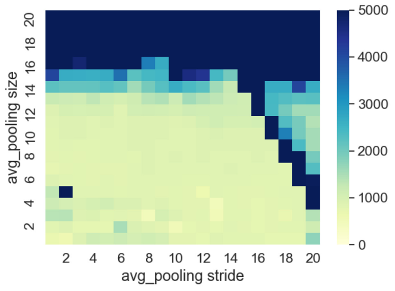
Finally, we consider the *ACC* metric (Fig. 7). Interestingly, the *ACC* metric presents similar patterns as the other metrics but reversely. More specifically, the settings that reach better GE/T_{GE0} values are worse with *ACC* and vice versa. With this observation, we can conclude that overfitting is the cause of the degraded performance. Indeed, the HW leakage model forces the dataset to follow a binomial distribution. Thus, the overfitted model tends to output high probabilities for the middle classes (i.e., the HW class 4 and then HW classes 3 and 5) regardless of the input. On the other hand, the overfitting may also be triggered by the dataset property as the same key is used for both training and attack. Indeed, the model can easily "learn" the correct key instead of successfully exploiting leakages with this setting. Following this, although the model may have higher validation accuracy and lower loss, the model's classification



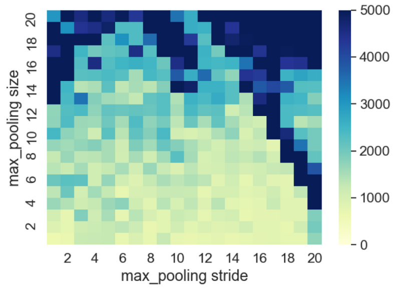
(a) T_{GE0}: tuning the first average-pooling layer (min: 321; max: 2 569)



(b) T_{GE0}: tuning the first max-pooling layer (min: 527; max: 4 616)



(c) T_{GE0}: tuning the last average-pooling layer (min:500; max: >5 000)



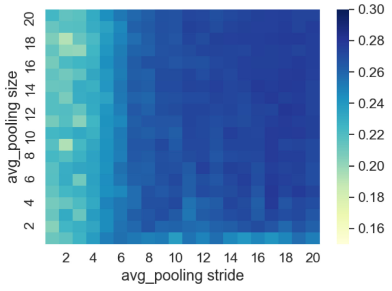
(d) T_{GE0}: tuning the last max-pooling layer (min: 638; max: >5 000)

Fig. 6. T_{GE0} for the standardized dataset with average-/max-pooling layer for the HW leakage model on ASCAD_f.

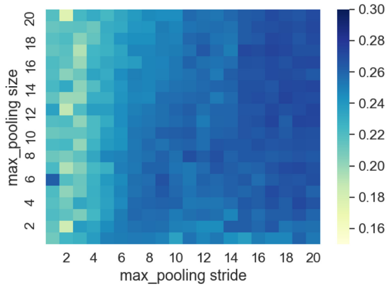
capability is degraded. Moreover, as can be seen from Figs. 7a, 7b, and 7c, overfitting is more easily triggered with larger pooling settings, which is equivalent to smaller network sizes. For the max-pooling in the last layer (Fig. 7d), a more uniform distribution of the *ACC* value can be seen, indicating its potential of reducing the network size while keeping good attack performance.

5.3 ASCAD with Random Keys (ASCAD_r)

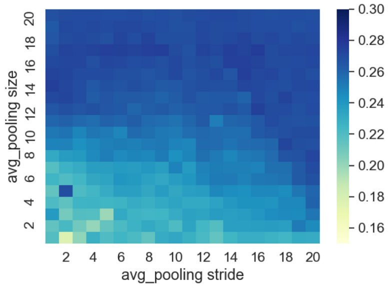
Compared with the ASCAD_f dataset, the length of a trace in the ASCAD_r dataset is doubled (1 400 features). Since the same CNN model (CNN_{ASCAD}) is used as the profiling model, the number of features available at the output of the last convolution layer (input of the last pooling layer) is also doubled, providing additional range to tune the hyperparameter of the pooling layer. Aligned with the experiments for the ASCAD_f dataset, we tune both average- and max-pooling layer and analyze the results with different metrics.



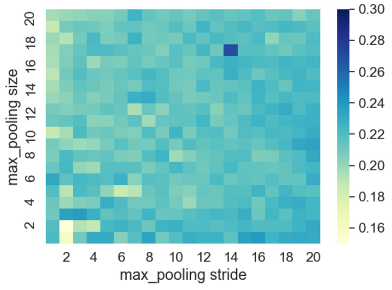
(a) *ACC*: tuning the first average-pooling layer (min: 0.192; max: 0.280)



(b) *ACC*: tuning the first max-pooling layer (min: 0.176; max: 0.274)



(c) *ACC*: tuning the last average-pooling layer (min: 0.177; max: 0.277)



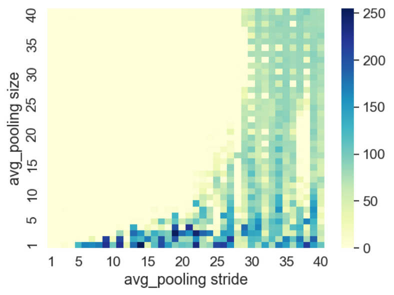
(d) *ACC*: tuning the last max-pooling layer (min: 0.149; max: 0.270)

Fig. 7. *ACC* for the standardized dataset with average-/max-pooling layer for the HW leakage model on ASCAD_f.

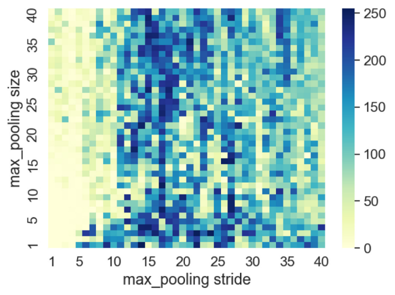
First, we apply the *GE* metric and give results in Fig. 8. Interestingly, we again confirm the conclusion made for the ASCAD_f dataset: for the pooling layer in the shallower layers, pooling stride is essential in extracting and down-sampling the features, while the pooling size should be more carefully tuned in the deeper layers. Meanwhile, average-pooling performs better than max-pooling for most of the setting combinations. This tendency becomes more significant when investigating the first layer: for the max-pooling layer, 29% of the pooling setting combinations lead to *GE* value below 50 with 5 000 attack traces. When using the average-pooling layer, this value increases to 72%. Recall the observations for the ChipWhisperer dataset: an average-pooling layer is more suitable for the standardized dataset, while the max-pooling layer works better for the original (non-standardized dataset). Here, we reach the same conclusion from the results when attacking the ASCAD_r dataset.

Compared with the conclusions for ASCAD_f, it seems that more input features lead to a better performance of the average-pooling layer than max-pooling.

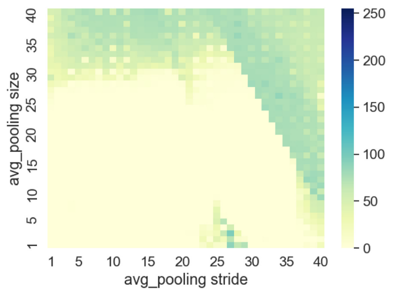
However, considering the different characteristics of the data, no definitive conclusions can be drawn.



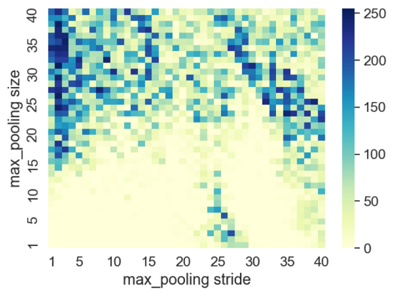
(a) *GE*: tuning the first average-pooling layer (min: 0; max: 255)



(b) *GE*: tuning the first max-pooling layer (min: 0; max: 255)



(c) *GE*: tuning the last average-pooling layer (min: 0; max: 96)

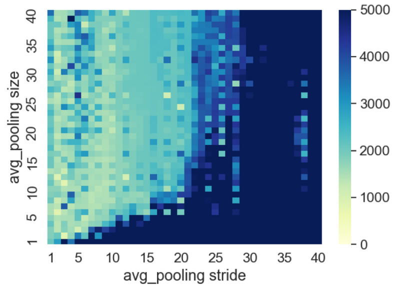


(d) *GE*: tuning the last max-pooling layer (min: 0; max: 252)

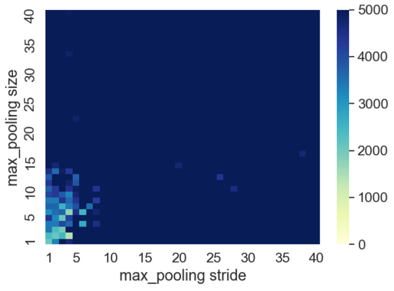
Fig. 8. *GE* for standardized dataset with average-/max-pooling layer for the HW leakage model on ASCAD_r.

The performance deviations of average- and max-pooling become more pronounced when considering T_{GE0} as depicted in Fig. 9. Specifically, from Fig. 9b, only 66 setting combinations (out of 1 600) required less than 2 000 attack traces to retrieve the correct key. When using the average-pooling as the first pooling layer, this value increases to 997. For the last pooling layer, the differences between the two pooling methods are reduced. Still, average-pooling has more tolerance (889 good settings) to the hyperparameter variation than max-pooling (414 good settings).

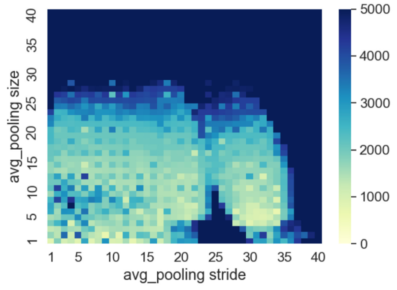
Finally, we analyze the attack results with the *ACC* metric (Fig. 10), which are similar to the one for ASCAD_f (see Fig. 10c). The model starts overfitting with a larger pooling stride and pooling size. Interestingly, this observation is more distinguishable for the average-pooling method. For the max-pooling layer



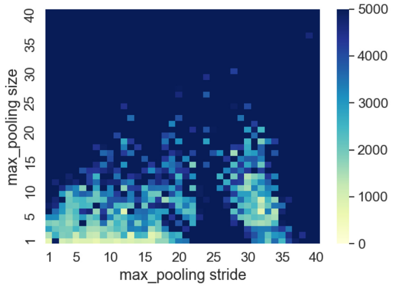
(a) T_{GE0}: tuning the first average-pooling layer (min: 789; max: >5 000)



(b) T_{GE0}: tuning the first max-pooling layer (min: 1 440; max: >5 000)



(c) T_{GE0}: tuning the last average-pooling layer (min:668; max: >5 000)



(d) T_{GE0}: tuning the last max-pooling layer (min: 771; max: >5 000)

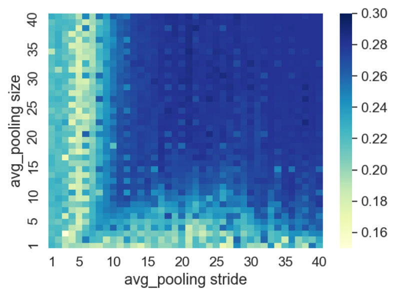
Fig. 9. T_{GE0} for standardized dataset with average-/max-pooling layer for the HW leakage model on ASCAD_r.

(Figs. 10b and 10d), the *ACC* values distribute more uniformly, indicating the possibility of the trained model to be underfitting. Together with the observations from ASCAD_f with *ACC* metric: a model equipped with max-pooling layers may require more training effort, and additional training epochs may help enhance the attack performance.

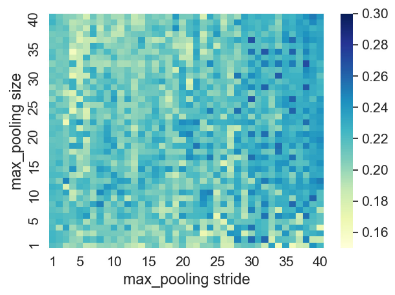
5.4 General Observations and Suggestions

Based on experiments from previous sections, testing on three different datasets, we provide the following observations:

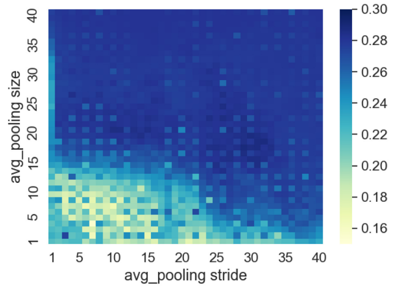
– Data standardization can be an effective tool to avoid the overfitting and improve the attack performance.
– When the input data has limited features, a pooling layer in the shallow part of the network is more sensitive to pooling stride variation. While in the deeper layer, the influence of pooling size becomes more significant.



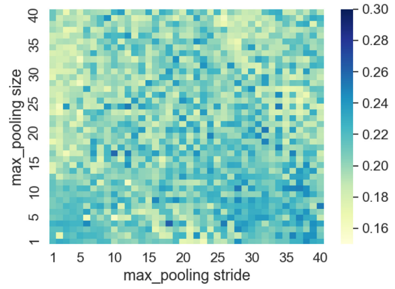
(a) *ACC*: tuning the first average-pooling layer (min: 0.137; max: 0.289)



(b) *ACC*: tuning the first max-pooling layer (min: 0.161; max: 0.270)



(c) *ACC*: tuning the last average-pooling layer (min: 0.155; max: 0.288)



(d) *ACC*: tuning the last max-pooling layer (min: 0.159; max: 0.262)

Fig. 10. *ACC* for standardized dataset with average-/max-pooling layer for the HW leakage model on ASCAD_r.

Following these observations, we give the following suggestions regarding the pooling layer's hyperparameter selection:

– Applying data standardization can significantly increase the robustness of the model in terms of pooling layer's hyperparameter variation.
– Although in some cases the max-pooling layer slightly outperforms its counterpart, an average-pooling layer is more preferable as it can consistently give good attack performance.
– For the shallower pooling layers, smaller pooling strides are required to avoid omitting the important features. At the same time, the smaller pooling sizes are preferable for intermediate/deeper pooling layers.
– For the network size reduction, larger pooling sizes could be applied for the shallower pooling layers. The deeper pooling layers could be used with larger pooling strides.

6 Conclusions and Future Work

In this paper, we considered the effect of a pooling layer towards CNN-based SCA. We investigated one unprotected dataset (ChipWhisperer) and two datasets protected with masking countermeasures (ASCAD_f and ASCAD_r). Two commonly used pooling methods, average-pooling, and max-pooling are tested with different hyperparameter settings. The results are evaluated through three metrics. Our results clearly show that the pooling method and the corresponding hyperparameters should be determined based on both the depth of the (pooling) layer and the size of input features.

In future work, we plan to explore the influence of the pooling layer's hyperparameter choice for various input sizes and profiling models. Next, we aim to explore the role of the countermeasures when selecting and tuning the pooling layers. Finally, in this work, we concentrated on the HW leakage model only. It would be interesting to expand this to other leakage models.

References

1. Benadjila, R., Prouff, E., Strullu, R., Cagli, E., Dumas, C.: Deep learning for side-channel analysis and introduction to ASCAD database. J. Cryptograph. Eng. **10**(2), 163–188 (2020). https://doi.org/10.1007/s13389-019-00220-8
2. Brier, E., Clavier, C., Olivier, F.: Correlation power analysis with a leakage model. In: Joye, M., Quisquater, J.-J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 16–29. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-28632-5_2
3. Cagli, E., Dumas, C., Prouff, E.: Convolutional neural networks with data augmentation against jitter-based countermeasures. In: Fischer, W., Homma, N. (eds.) CHES 2017. LNCS, vol. 10529, pp. 45–68. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66787-4_3
4. Chari, S., Rao, J.R., Rohatgi, P.: Template attacks. In: Kaliski, B.S., Koç, K., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 13–28. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-36400-5_3
5. Gierlichs, B., Batina, L., Tuyls, P., Preneel, B.: Mutual information analysis. In: Oswald, E., Rohatgi, P. (eds.) CHES 2008. LNCS, vol. 5154, pp. 426–442. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85053-3_27
6. 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. IEEE (2015)
7. Heuser, A., Zohner, M.: Intelligent machine homicide. In: Schindler, W., Huss, S.A. (eds.) COSADE 2012. LNCS, vol. 7275, pp. 249–264. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29912-4_18
8. 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 Trans. Cryptograph. Hardware Embedded Syst., 148–179 (2019)
9. Kocher, P., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 388–397. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48405-1_25

10. Lerman, L., Medeiros, S.F., Bontempi, G., Markowitch, O.: A machine learning approach against a masked AES. In: Francillon, A., Rohatgi, P. (eds.) CARDIS 2013. LNCS, vol. 8419, pp. 61–75. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08302-5_5
11. Lerman, L., Poussier, R., Bontempi, G., Markowitch, O., Standaert, F.-X.: Template attacks vs. machine learning revisited (and the curse of dimensionality in side-channel analysis). In: Mangard, S., Poschmann, A.Y. (eds.) COSADE 2014. LNCS, vol. 9064, pp. 20–33. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21476-4_2
12. Li, H., Krček, M., Perin, G.: A comparison of weight initializers in deep learning-based side-channel analysis. In: Zhou, J., et al. (eds.) ACNS 2020. LNCS, vol. 12418, pp. 126–143. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-61638-0_8
13. Maghrebi, H., Portigliatti, T., Prouff, E.: Breaking cryptographic implementations using deep learning techniques. In: Carlet, C., Hasan, M.A., Saraswat, V. (eds.) SPACE 2016. LNCS, vol. 10076, pp. 3–26. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-49445-6_1
14. Mangard, S., Oswald, E., Popp, T.: Power Analysis Attacks. Springer, Boston (2007). https://doi.org/10.1007/978-0-387-38162-6
15. O'Flynn, C., Chen, Z.D.: ChipWhisperer: an open-source platform for hardware embedded security research. In: Prouff, E. (ed.) COSADE 2014. LNCS, vol. 8622, pp. 243–260. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10175-0_17
16. Perin, G., Picek, S.: On the influence of optimizers in deep learning-based side-channel analysis. IACR Cryptology ePrint Archive 2020, 977 (2020). https://eprint.iacr.org/2020/977
17. 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. Cryptograph. Hardware Embedded Syst. **2019**(1), 209–237 (2018). https://doi.org/10.13154/tches.v2019.i1.209-237. https://tches.iacr.org/index.php/TCHES/article/view/7339
18. 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. Cryptograph. Hardware Embedded Syst. **2019**(1), 1–29 (2019)
19. 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.) SPACE 2018. LNCS, vol. 11348, pp. 157–176. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-05072-6_10
20. Quisquater, J.-J., Samyde, D.: ElectroMagnetic analysis (EMA): measures and counter-measures for smart cards. In: Attali, I., Jensen, T. (eds.) E-smart 2001. LNCS, vol. 2140, pp. 200–210. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45418-7_17
21. Rijsdijk, J., Wu, L., Perin, G., Picek, S.: Reinforcement learning for hyperparameter tuning in deep learning-based side-channel analysis. Technical report, Cryptology ePrint Archive, Report 2021/071 (2021). https://eprint.iacr.org
22. Rijsdijk, J., Wu, L., Perin, G., Picek, S.: Reinforcement learning for hyperparameter tuning in deep learning-based side-channel analysis. Cryptology ePrint Archive, Report 2021/071 (2021). https://eprint.iacr.org/2021/071

23. Standaert, F.-X., Malkin, T.G., Yung, M.: A unified framework for the analysis of side-channel key recovery attacks. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 443–461. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-01001-9_26
24. Tran, N.Q., Nguyen, H.Q.: Efficient CNN-based profiled side channel attacks. J. Comput. Sci. Cybern. **37**(1), 1–22 (2021)
25. Wouters, L., Arribas, V., Gierlichs, B., Preneel, B.: Revisiting a methodology for efficient CNN architectures in profiling attacks. IACR Trans. Cryptograph. Hardware Embedded Syst. **2020**(3), 147–168 (2020). https://doi.org/10.13154/tches.v2020.i3.147-168. https://tches.iacr.org/index.php/TCHES/article/view/8586
26. Wu, L., Perin, G., Picek, S.: I choose you: Automated hyperparameter tuning for deep learning-based side-channel analysis. Cryptology ePrint Archive, Report 2020/1293 (2020). https://eprint.iacr.org/2020/1293
27. Zaid, G., Bossuet, L., Habrard, A., Venelli, A.: Methodology for efficient CNN architectures in profiling attacks. IACR Trans. Cryptograph. Hardware Embedded Syst. **2020**(1), 1–36 (2019). https://doi.org/10.13154/tches.v2020.i1.1-36. https://tches.iacr.org/index.php/TCHES/article/view/8391