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

Lichao Wu and Guilherme Perin

Delft University of Technology, The Netherlands

Abstract. In recent years, the advent of deep neural networks opened new perspectives for security evaluations with side-channel analysis. Specifically, profiling attacks now benefit from capabilities offered by convolutional neural networks, such as dimensionality reduction, the absence of manual feature selection, and the inherent ability to reduce 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. Finally, with a larger pooling stride and size, a neural network can be reduced in size, favoring training performance.

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 [13,19,3,8].

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 [8,3], 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 profiling 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' combination increase exponentially. Although some researchers are trying to set general design rules [28,25] or apply neural architecture search to find the best-performing network automatically [26,21], 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 its depth and the number of input features. We also give guidelines on how to choose the hyperparameters in different cases. We confirm that the pooling layer's addition can help reduce the network size while keeping excellent attack performance. 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, the dataset is divided 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, the vector of learnable parameters is denoted with $\boldsymbol{\theta}$, while the set of hyperparameters defining the profiling model f is denoted 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 model uses examples of input-output pairs. When considering 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 $\boldsymbol{\theta}'$ minimizing 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 $Q \times c$. Each probability value denotes how likely a certain measurement should be classified into a specific class (thus, $\mathbf{p}_{i,v}$ represents the probability that a specific 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:

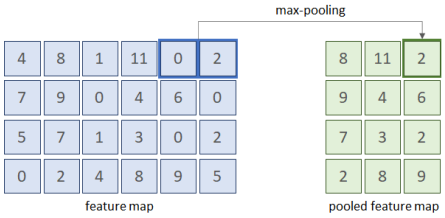
$$S(k) = \sum_{i=1}^{Q} \log(\mathbf{p}_{i,v}). \tag{1}$$

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 metrics like guessing entropy (GE) [23], representing the average position of k^* in a key guessing vector $\mathbf{g} = [g_1, g_2, \ldots, g_{|\mathcal{K}|}]$. As common, 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.

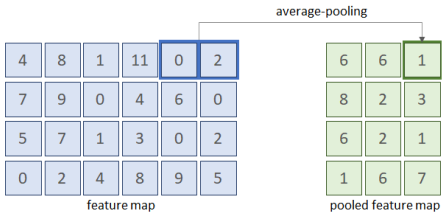
2.3 Convolutional Neural Networks

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. As we treat uni-dimensional side-channels, all convolution and pooling operations are 1D operations. Figure 1 illustrates the different types of pooling operations over a feature map (output of a convolution layer). As we see in the example from this figure, 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. Example for pooling size and pooling stride of 2.



(b) Average-pooling operation. Example for pooling size and pooling stride of 2.

Fig. 1: Pooling types

Finally, it is worth noting the difference between the parameters and hyperparameters for machine learning algorithms. Hyperparameters are all configuration variables corresponding to the model architecture, e.g., the convolution size or the type of pooling layer. The parameters are the configuration variables whose values can be estimated from data. The examples of parameters are the weights and bias in a neural network. When discussing tuning a neural network (or its pooling layer), we mean tuning its *hyperparameters*.

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]. More precisely, the ASCAD dataset contains 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 where the attacker's goal is:

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

From the information-theoretic point of view, template attack (TA) [4] represents the most powerful profiling SCA. There, one uses the probability density function (PDF) as templates to perform the attack. In an ideal 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) [13,3,8] emerged as the most 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 [28]. 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 Section 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; ASCAD fixed key dataset (ASCAD_f) and ASCAD random keys dataset (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 [26,22]. 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 the ASCAD paper [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.

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

Table 1: CNN architectures used in the experiments.

To evaluate the profiling attack performance, we consider four SCA-based 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.
- L_m: the correlation between the ideal key rank vector and the key rank (or guessing entropy) vector calculated from the attack [27].
- ACC: the classification accuracy on the validation traces.

GE and L_m metrics are derived from guessing entropy, aiming at evaluating the key recovery capacity of trained neural networks by setting a limited number of attack traces.. Specifically, 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 circumstance, we can better estimate the attack performance by evaluating the number of attack traces to reach it Implementation details and benefits of L_m metric in profiling SCA are provided in [27]. This metric can indicate attack performance even if the number of attack traces is limited and other metrics, such as GE or T_{GE0}, cannot precisely describe the key recovery capacity from the profiling model. Although related works indicate a low correlation between validation accuracy and success of an attack [17], the ACC metric shows that a higher validation accuracy could still mean a lower GE [26,21]. Therefore, the validation accuracy is also taken into consideration.

In the experimental results, we first try to understand the influence of data standardization on the attack performance for the ChipWhisperer dataset. After understanding this, 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. Additionally, we vary the pooling settings in different layers to understand the correlation between the pooling hyperparameter variation and layer depth. Finally, we explore the contribution of the pooling layer by training a profiling model with and without the last pooling layer.

5 Experimental Results

The experiments start with ChipWhisperer as this dataset is easily breakable even with a small CNN architecture. The required amount of 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

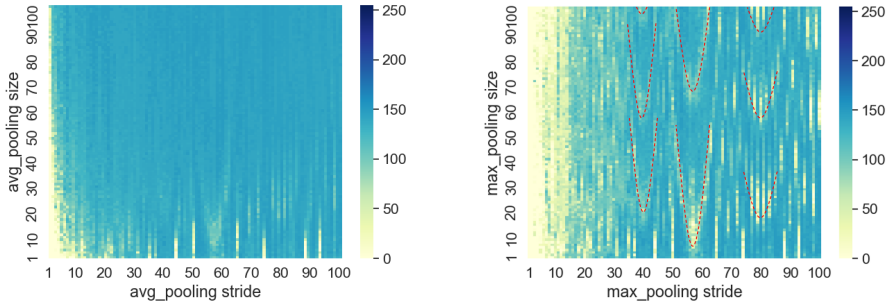
two versions of a dataset: original (no preprocessing) and standardized (forcing the amplitude ranges from -1 to 1).

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. Finally, we also investigate the role of the pooling layer from the aspects of the network size and attack performance. This experiment is launched by training and comparing the model with and without the last pooling layer.

5.1 Case Study: the ChipWhisperer Dataset

The results for GE are shown in Figure 2. Since GE remain zero for all hyperparameter combinations when attacking the standardized dataset, we only present the GE value for the original dataset. As mentioned, 2 000 traces are used for the attack. First, we can conclude that the data standardization increases the model's resilience towards the pooling layers' hyperparameter variation. As shown in Figure 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, there are cases when a large pooling stride can achieve outstanding attack performance (i.e., pooling stride equal to 45, 50, 75, 85). Meanwhile, a large pooling stride can effectively reduce the 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.

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 (i.e., at max-pooling stride: 57, 80) that occur periodically. The corresponding patterns are also marked by a red dashed line in Figure 3b. 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



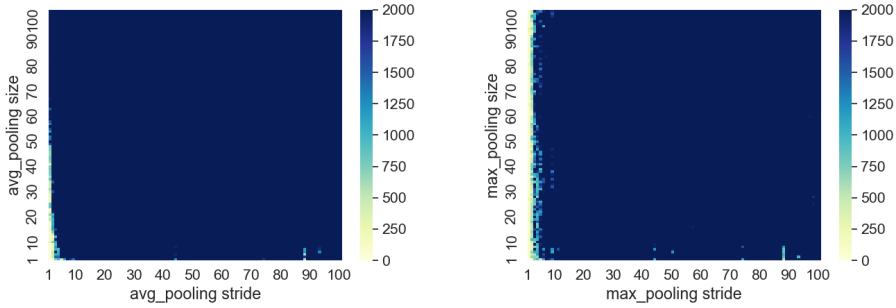
(a) GE: original dataset with average-pooling (min: 0; max: 150)

(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.

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 Figure 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.



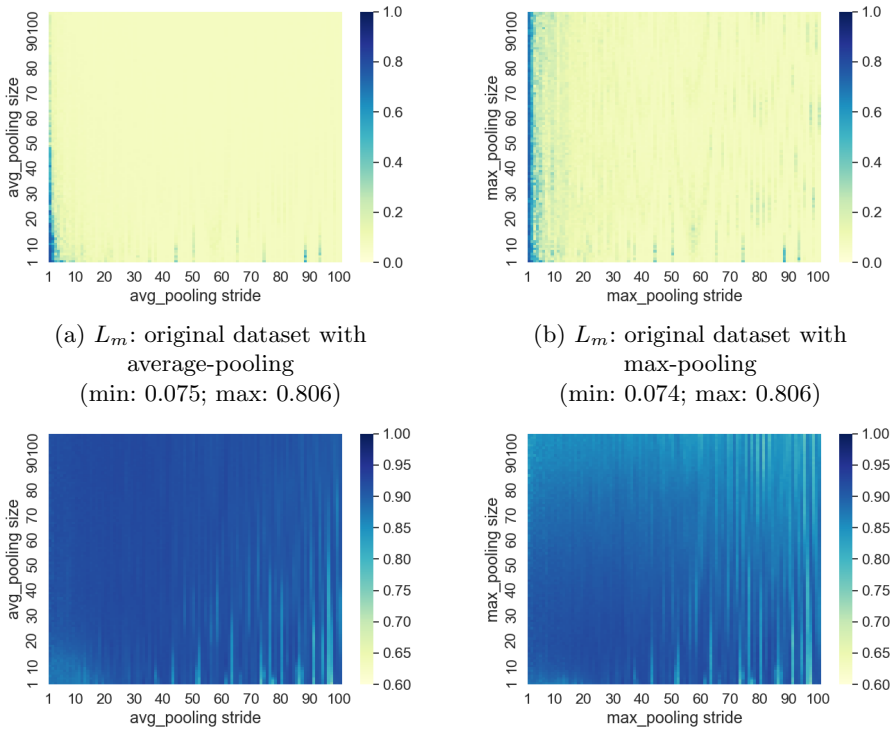
(a) T_{GE0}: original dataset with average-pooling (min: 31; max: >2 000)

(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.

Next, we apply the L_m metric and depict results in Figure 4. In line with the observations for the GE metric, the standardized dataset is easier to attack than the original one without preprocessing. Although L_m reaches a higher value with smaller pooling strides (less than ten) for the original dataset, L_m reaches above 0.5 for all hyperparameter combinations when applying the standardization to the dataset. Additionally, for the original dataset, we again observe that the max-pooling layer is more resilient to the pooling size variation, ensuring a large number of setting combinations for a successful attack.

Recall that the secret can be obtained with only one trace with the standardized dataset, so limited information can be acquired by evaluating GE and T_{GE0}. With the help of L_m, we observe the influence of the hyperparameter variation: max-pooling performs slightly better than the average-pooling. Indeed, only 52 average-polling setting combinations lead to the L_m value greater than 0.5. When using the max-pooling layer, this value increases to 174.



(a) L_m: original dataset with average-pooling (min: 0.075; max: 0.806)

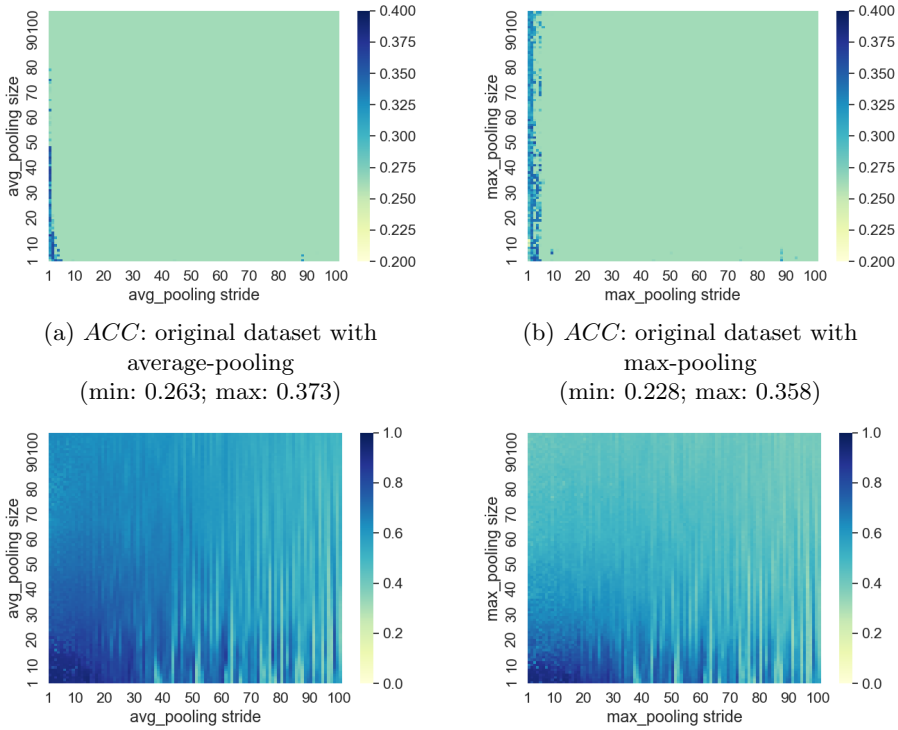
(b) L_m: original dataset with max-pooling (min: 0.074; max: 0.806)

(c) L_m: standardized dataset with average -pooling (min: 0.748; max: 0.929)

(d) L_m: standardized dataset with max-pooling (min: 0.748; max: 0.929)

Fig. 4: L_m 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 Figure 5, 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.



(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)

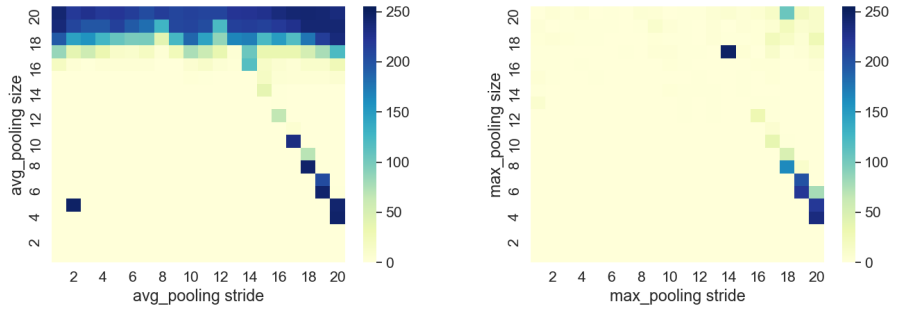
Fig. 5: Accuracy for the original/standardized dataset with average-/max-pooling layer for the HW leakage model on ChipWhisperer.

5.2 ASCAD with a Fixed Key (ASCAD_f)

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 dataset, we only attack the standardized dataset.

Tuning the first/last pooling layer. First, we evaluate the attack performance of each setting in combination with the GE metric. The results are shown in Figure 6. 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 classification efficiency. 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.



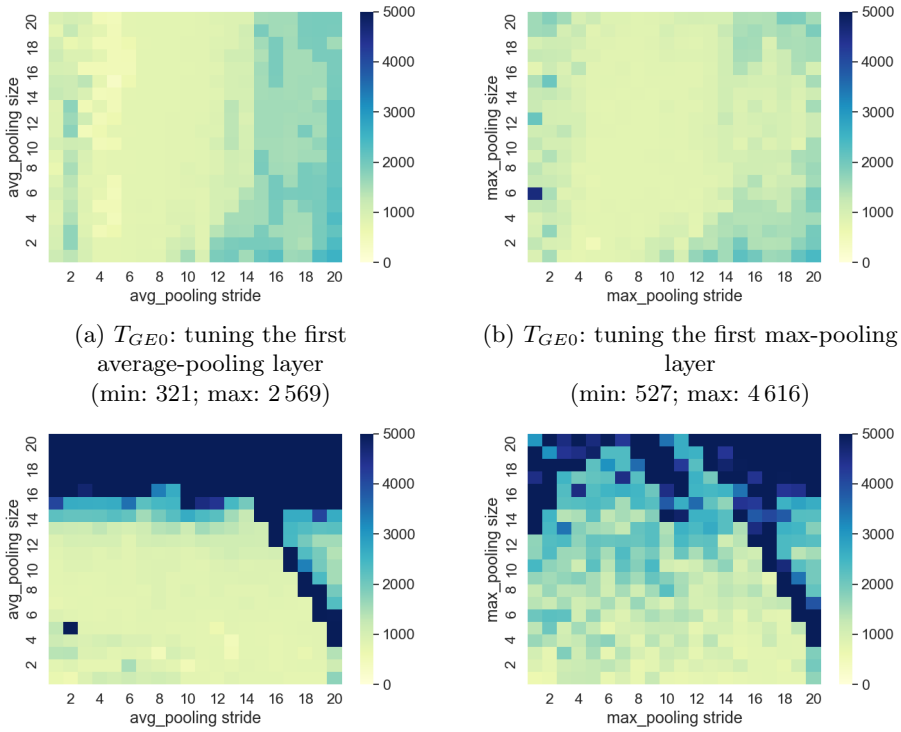
(a) GE: tuning the last average-pooling layer
(min: 0; max: 248)

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

Fig. 6: 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} (Figure 7), some unique patterns can be observed even when tuning the first pooling layer. From Figures 7a and 7b, 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 Figure 6. 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 Figures 7a and 7c or Figures 7b and 7d, 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.



(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)

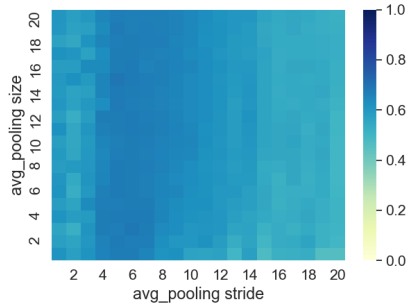
(c) T_{GE0}: tuning the last average-pooling layer (min:500; max: >5 000)

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

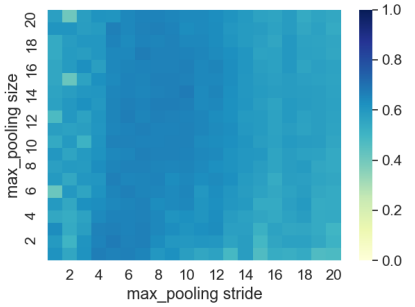
Fig. 7: T_{GE0} for the standardized dataset with average-/max-pooling layer for the HW leakage model on ASCAD_f.

Next, we evaluate the attack performance with L_m metric (Figure 8). From the results, we confirm the observations made with GE and T_{GE0}. First, tuning the first pooling layer has less impact on overall attack performance than varying the last pooling layer. Meanwhile, when the pooling stride is fixed for the first pooling layer, the pooling size variation causes less impact on L_m. For the last pooling layer, in contrast, the pooling stride becomes a less sensitive hy-

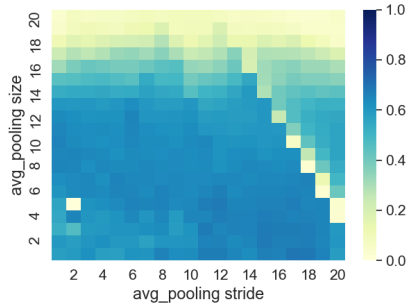
perparameter. When comparing the overall performance between average- and max-pooling, max-pooling performs slightly better than average-pooling when attacking the standardized ASCAD dataset: 255 average-pooling settings lead to L_m value greater than 0.5, while this value raises to 271 for the max-pooling. This observation is aligned with the conclusion drawn from the ChipWhisperer dataset.



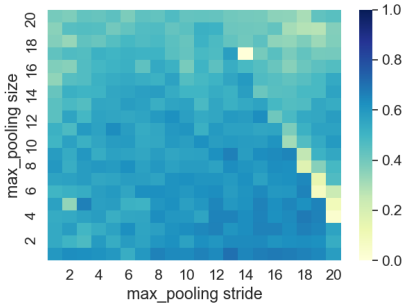
(a) L_m: tuning the first average-pooling layer
(min: 0.463; max: 0.688)



(b) L_m: tuning the first max-pooling layer
(min: 0; max: 0.691)



(c) L_m: tuning the last average-pooling layer
(min: 0.409; max: 0.676)

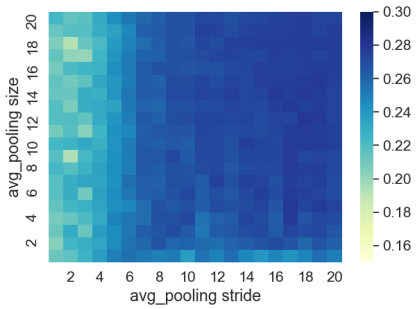


(d) L_m: tuning the last max-pooling layer
(min: 0; max: 0.706)

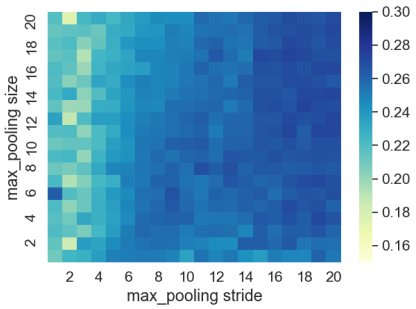
Fig. 8: L_m for the standardized dataset with average-/max-pooling layer for the HW leakage model on ASCAD_f.

Finally, we consider the *ACC* metric (Figure 9). Interestingly, the *ACC* metric presents similar patterns as the other three metrics but reversely. More specifically, the settings that reach better $GE/T_{GE0}/L_m$ 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

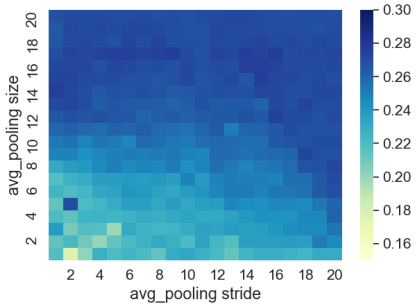
HW classes 3 and 5) regardless of the input. Following this, although the model may have higher validation accuracy and lower loss, the model's classification capability is degraded. Moreover, as can be seen from Figures 9a, 9b, and 9c, overfitting is more easily triggered with larger pooling settings, which is equivalent to smaller network sizes. For the max-pooling in the last layer (Figure 9d), a more uniform distribution of the ACC value can be seen, indicating its potential of reducing the network size while keeping good attack performance.



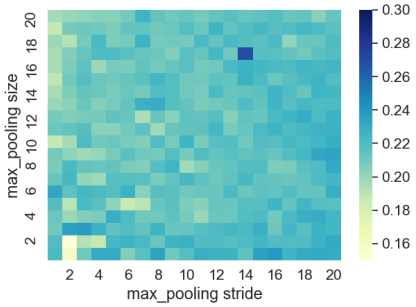
(a) ACC: tuning the first average-pooling layer
(min: 0.192; max: 0.280)



(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. 9: ACC for the standardized dataset with average-/max-pooling layer for the HW leakage model on ASCAD_f.

Tuning the dense layer with/without the last pooling layer. We investigate the role of the last pooling layer and the following dense layers, trying to find a direction to reduce the network size (complexity). Again, four evaluation metrics, GE, T_{GE0}, L_m, and ACC, are applied to interpret the attack results.

Results are shown in Figure 10. Note that each unit of the dense layer size represents 64 neurons. For example, for a dense layer with 64 units: 4 096 neurons are available in the dense layer.

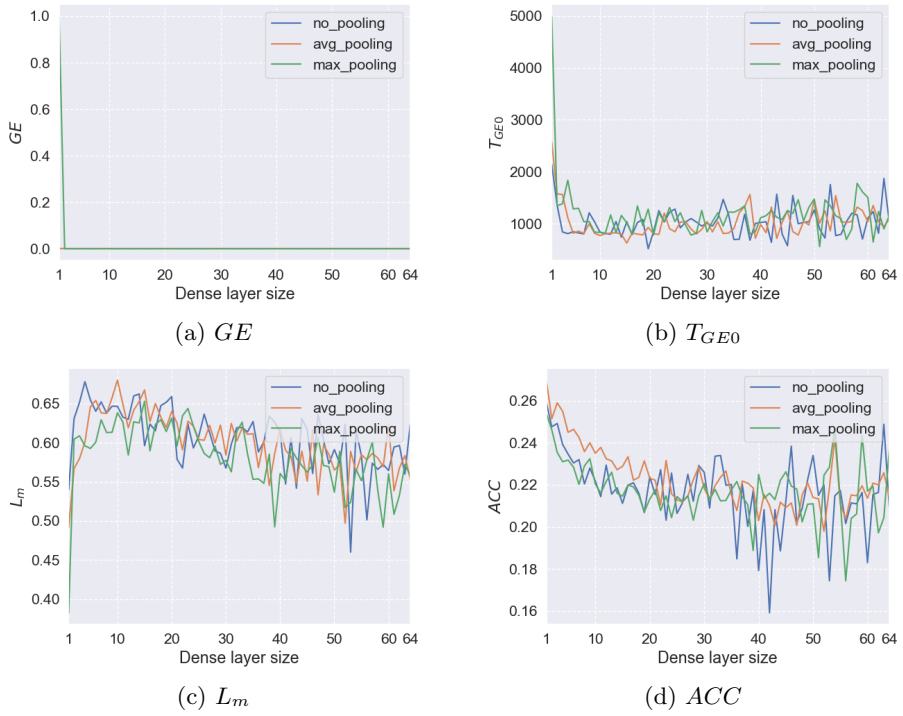
By fixing the hyperparameters of the pooling layer to default (two) and only tuning the size of the dense layer, a quick drop of GE and T_{GE0} and rise of L_m can be observed when increasing the dense layer size from one to three (64 neurons to 192 neurons). Moreover, in Figures 10b and 10c, a network without the last pooling layer (before the first dense layer) requires less dense neurons to reach the top performance, which can be attributed to the contribution of more features being used for classification. On the other hand, the pooling layer reduces the complexity of the network by averaging/selecting max value over multiple features. As a trade-off, more neurons are required in the dense layer to reach similar attack performance.

When further increasing the dense layer size, the values decrease for both L_m and ACC. Indeed, although the network's classification capability could be increased by increasing the complexity of the dense layer, the training effort to learn from the dataset is also increased. Therefore, to balance the attack performance and model complexity, the small size of the dense layer with a pooling layer could be optimal.

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. Note, the dense layer's size is varied to reduce the network size while keeping a good attack performance.

Tuning the first/last pooling layer. First, we apply the GE metric to interpret the results shown in Figure 11. Interestingly, we again confirm the conclusion drawn 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 setting combinations. This tendency becomes more significant when investigating the first layer: for the max-pooling layer, 21% 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 68%. 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



(a) GE (b) T_{GE0}

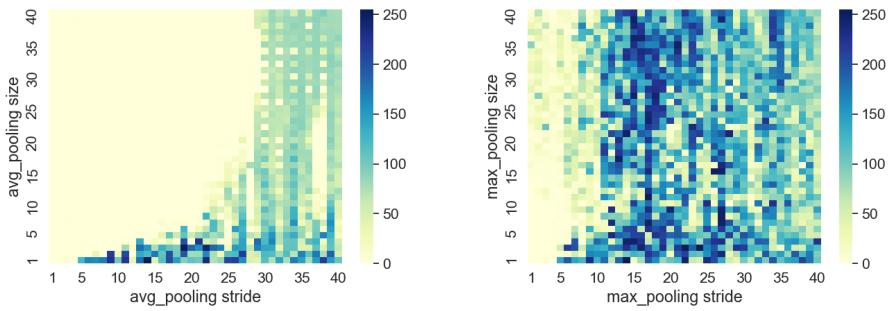
(c) L_m (d) ACC

Fig. 10: Tuning the dense layer with/without the average/max last pooling layer attacking ASCAD_f.

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.

The performance deviations of average- and max-pooling become more pronounced when considering T_{GE0} as depicted in Figure 12. Specifically, from Figure 12b, only 22 setting combinations (out of 400) 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 271. For the last pooling layer, the differences between the two pooling methods are reduced. Still, average-pooling has more tolerance (276 good settings) to the hyperparameter variation than max-pooling (179 good settings).

The results for the L_m metric are shown in Figure 13. They consolidate the observations from the previous two metrics but also provide new information. For instance, when looking at Figures 13b and 13d, we find more setting combinations with potential to reach a high L_m value, eventually leading to a successful attack. Recall that overfitting represents the main reason for the degradation of the attack performance for the ASCAD_f. From Figure 13d, the sub-optimal L_m



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

(b) *GE*: tuning the first max-pooling layer
(min: 0; max: 245)

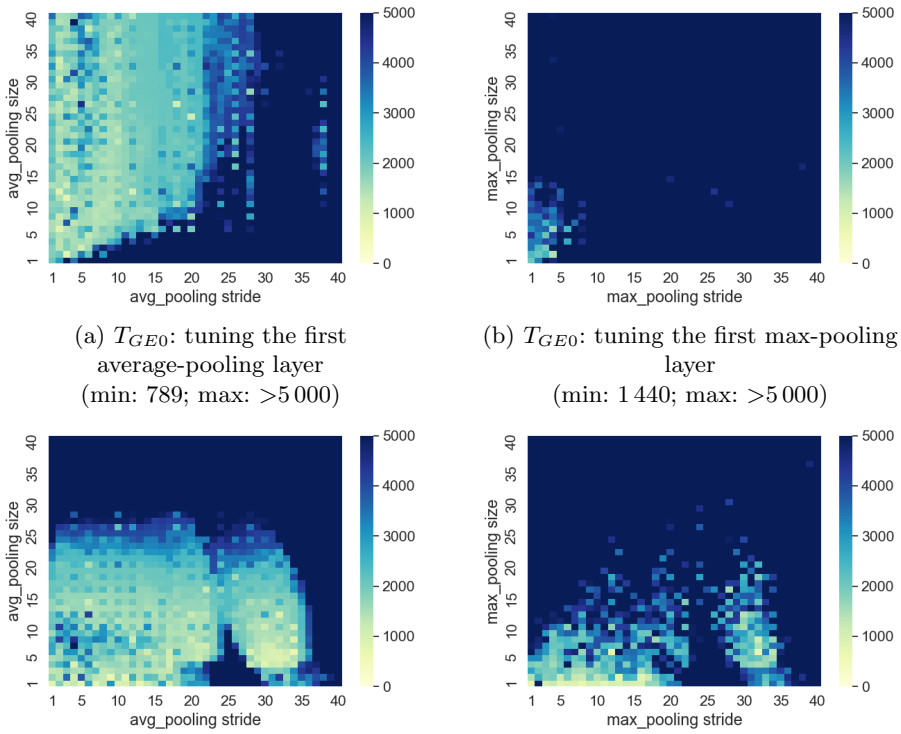
(c) *GE*: tuning the last average-pooling layer
(min: 0; max: 96)

(d) *GE*: tuning the last max-pooling layer
(min: 0; max: 249)

Fig. 11: *GE* for original/standardized dataset with average-/max-pooling layer for the HW leakage model on ASCAD_r.

values (0.2) are more concentrated in the middle of the graph, indicating a huge reduction of the model size. Therefore, training longer could be a solution to enhance the attack performance. Again, in general, average-pooling outperforms max-pooling in both shallower and deeper layers.

We analyze the attack results with the *ACC* metric in Figure 14, which are similar to ASCAD_f (see, e.g., Figure 14c). 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 (Figures 14b and 14d), the *ACC* values distribute more uniformly, indicating the possibility of the trained model to be underfitting. Combined with the observations for ASCAD_r: a model equipped with max-pooling layers may require more training effort, and additional training epochs may help enhance the attack performance.



(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)

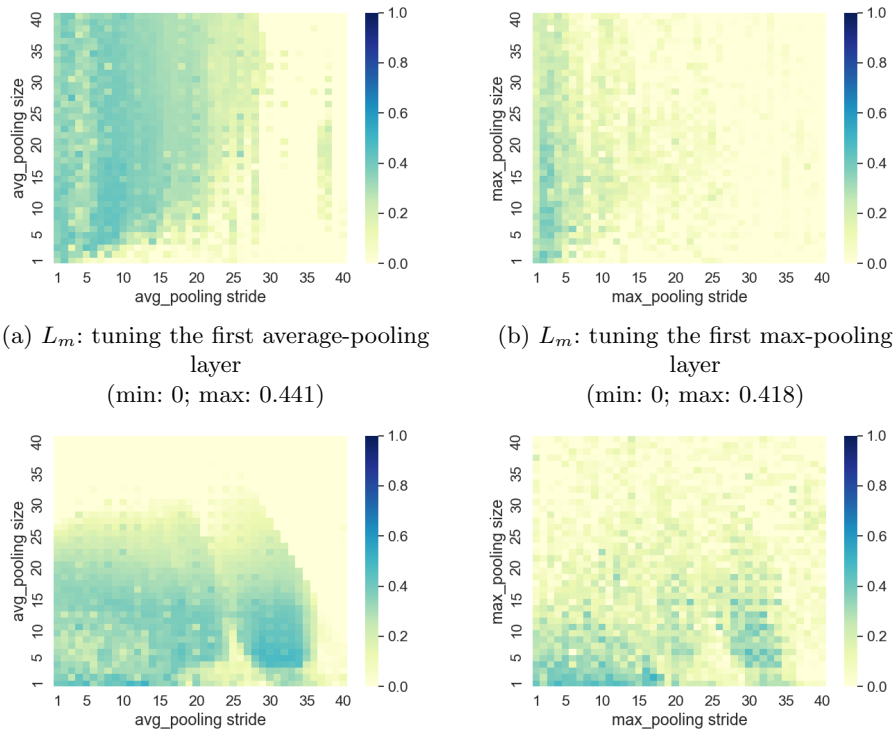
(c) T_{GE0}: tuning the last average-pooling layer (min:668; max: >5 000)

(d) T_{GE0}: tuning the last max-pooling layer (min: 746; max: >5 000)

Fig. 12: T_{GE0} for original/standardized dataset with average-/max-pooling layer for the HW leakage model on ASCAD_r.

Tuning the dense layer with/without the last pooling layer. Next, we set the pooling layer's hyperparameter to default (two) and tune the size of the dense layer. The results are shown in Figure 15. Different from the observations in Figure 10 (ASCAD_f), by increasing the dense layer size, we see an improved attack performance for T_{GE0}, L_m, and ACC, indicating the potential to further increase the attack performance by using larger dense layer size and more dense layers. Besides, compared with the model with the pooling layers, removal of the last pooling layer tends to have less variation when increasing the dense layer size, but more trainable parameters are used as a trade-off (the output of the last convolution layer is directly flattened and fully connected with the first dense layer). In general, the model with or without pooling performs equally well, but pooling layers are still needed to construct a CNN model that reduces the model size while keeping a good attack performance.

Based on those observations, we list the conclusions for all three datasets:



(a) L_m: tuning the first average-pooling layer
(min: 0; max: 0.441)

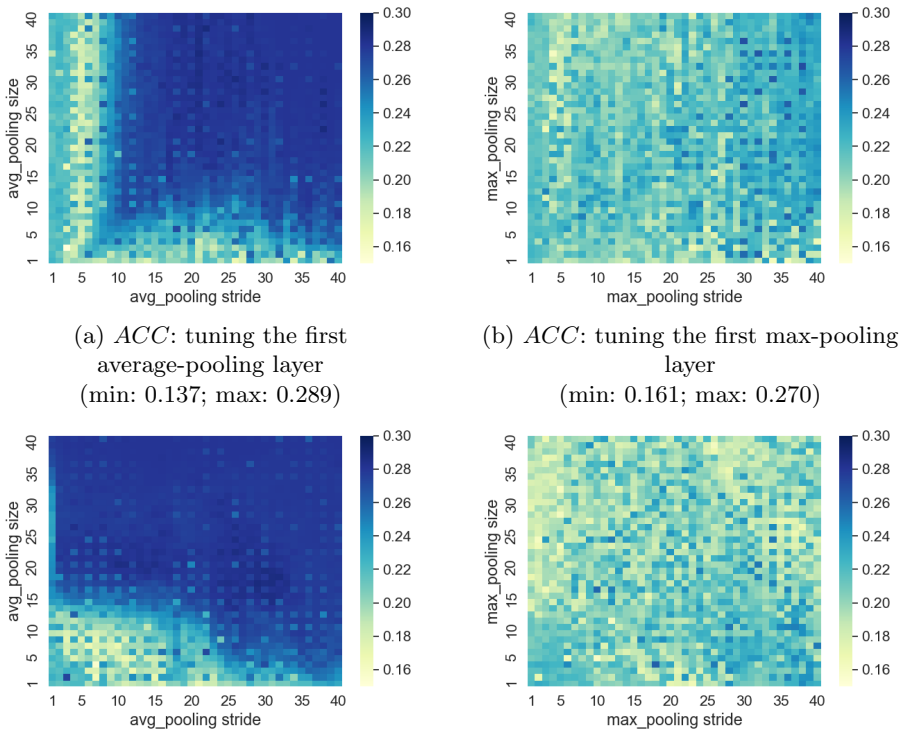
(b) L_m: tuning the first max-pooling layer
(min: 0; max: 0.418)

(c) L_m: tuning the last average-pooling layer
(min: 0; max: 0.510)

(d) L_m: tuning the last max-pooling layer
(min: 0; max: 0.443)

Fig. 13: L_m for original/standardized dataset with average-/max-pooling layer for the HW leakage model on ASCAD_r

– Data standardization can significantly improve the attack performance.
– When the input data has limited features, varying a pooling layer in the shallow layer causes less influence on the attack performance than in the deeper layer.
– For the deeper pooling layers, if the input features are limited, the max-pooling layer is preferable. Otherwise, an average-pooling layer could lead to better performance.
– Smaller pooling strides are required for the shallower pooling layers. At the same time, the smaller pooling sizes are preferable for 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.
– The removal of some pooling layers may increase the robustness of the model towards the dense layer variation. Nevertheless, we recommend using the last



(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)

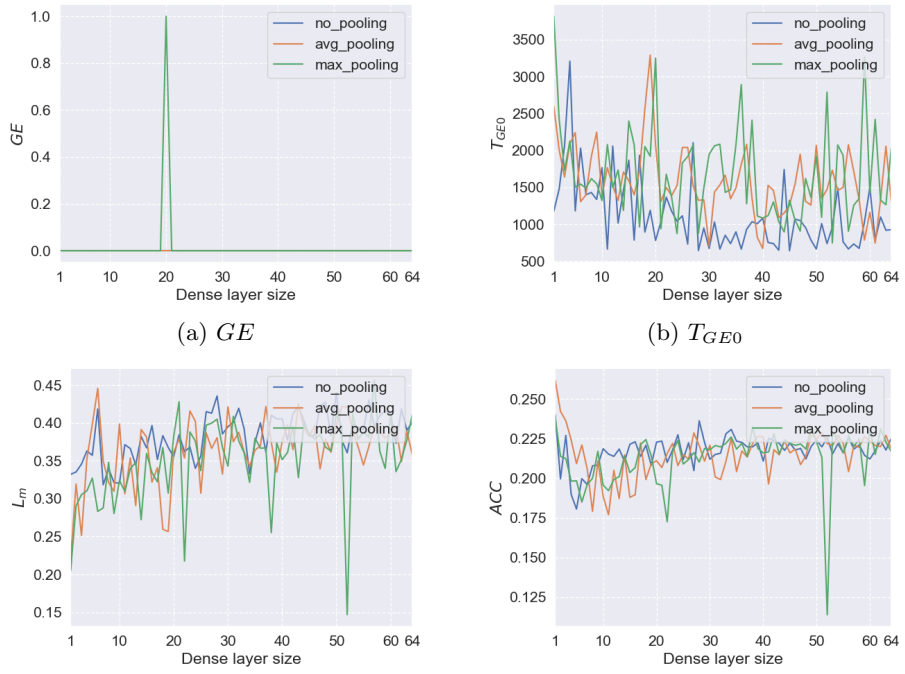
(d) *ACC*: tuning the last max-pooling layer (min: 0.162; max: 0.262)

Fig. 14: *ACC* for original/standardized dataset with average-/max-pooling layer for the HW leakage model on ASCAD_r.

pooling layer as part of the model to efficiently reduce the network size while keeping a good attack performance.

6 Conclusions and Future Work

In this paper, we considered the effect of a pooling layer towards profiling side-channel analysis. 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 four 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. Besides, we evaluated the importance of the last pooling layer in terms of attack performance



(a) GE (b) T_{GE0}

(c) L_m (d) ACC

Fig. 15: Tuning the dense layer with/without the average/max last pooling layer attacking ASCAD_r.

and network complexity. As a trade-off of model size reduction, the implementation of the pooling layer leads to omitting of some features. However, the attack performance is still comparable to the one without the last pooling layers.

We plan to explore the influence of the pooling layer's hyperparameter choice in various input sizes and profiling models for future work. 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 in future work.

References

1. Benadjila, R., Prouff, E., Strullu, R., Cagli, E., Dumas, C.: Deep learning for side-channel analysis and introduction to ASCAD database. J. Cryptographic Engineering **10**(2), 163–188 (2020). https://doi.org/10.1007/s13389-019-00220-8, `https://doi.org/10.1007/s13389-019-00220-8`
2. Brier, E., Clavier, C., Olivier, F.: Correlation power analysis with a leakage model. In: International Workshop on Cryptographic Hardware and Embedded Systems. pp. 16–29. Springer (2004)

3. Cagli, E., Dumas, C., Prouff, E.: Convolutional neural networks with data augmentation against jitter-based countermeasures. In: International Conference on Cryptographic Hardware and Embedded Systems. pp. 45–68. Springer (2017)
4. Chari, S., Rao, J.R., Rohatgi, P.: Template Attacks. In: CHES. LNCS, vol. 2523, pp. 13–28. Springer (August 2002), San Francisco Bay (Redwood City), USA
5. Gierlichs, B., Batina, L., Tuyls, P., Preneel, B.: Mutual information analysis. In: International Workshop on Cryptographic Hardware and Embedded Systems. pp. 426–442. Springer (2008)
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 - Breaking Cryptographic Devices Using Support Vector Machines. In: Schindler, W., Huss, S.A. (eds.) COSADE. LNCS, vol. 7275, pp. 249–264. Springer (2012)
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 Transactions on Cryptographic Hardware and Embedded Systems pp. 148–179 (2019)
9. 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
10. Lerman, L., Medeiros, S.F., Bontempi, G., Markowitch, O.: A Machine Learning Approach Against a Masked AES. In: CARDIS. Lecture Notes in Computer Science, Springer (November 2013), berlin, Germany
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: International Workshop on Constructive Side-Channel Analysis and Secure Design. pp. 20–33. Springer (2015)
12. Li, H., Krček, M., Perin, G.: A comparison of weight initializers in deep learning-based side-channel analysis. In: Zhou, J., Conti, M., Ahmed, C.M., Au, M.H., Batina, L., Li, Z., Lin, J., Losiouk, E., Luo, B., Majumdar, S., Meng, W., Ochoa, M., Picek, S., Portokalidis, G., Wang, C., Zhang, K. (eds.) Applied Cryptography and Network Security Workshops. pp. 126–143. Springer International Publishing, Cham (2020)
13. Maghrebi, H., Portigliatti, T., Prouff, E.: Breaking cryptographic implementations using deep learning techniques. In: International Conference on Security, Privacy, and Applied Cryptography Engineering. pp. 3–26. Springer (2016)
14. 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/
15. O'Flynn, C., Chen, Z.D.: Chipwhisperer: An open-source platform for hardware embedded security research. In: International Workshop on Constructive Side-Channel Analysis and Secure Design. pp. 243–260. Springer (2014)
16. Perin, G., Picek, S.: On the influence of optimizers in deep learning-based side-channel analysis. IACR Cryptol. ePrint Arch. **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 Transactions on Cryptographic Hardware and Embedded Systems

2019(1), 209–237 (Nov 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 Transactions on Cryptographic Hardware and Embedded Systems **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: International Conference on Security, Privacy, and Applied Cryptography Engineering. pp. 157–176. Springer (2018)
20. 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)
21. Rijsdijk, J., Wu, L., Perin, G., Picek, S.: Reinforcement learning for hyperparameter tuning in deep learning-based side-channel analysis. Tech. rep., Cryptology ePrint Archive, Report 2021/071, 2021. https://eprint. iacr. org ... (2021)
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.) Advances in Cryptology - EUROCRYPT 2009. pp. 443–461. Springer Berlin Heidelberg, Berlin, Heidelberg (2009)
24. Tran, N.Q., Nguyen, H.Q.: Efficient cnn-based profiled side channel attacks. Journal of Computer Science and Cybernetics **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 Transactions on Cryptographic Hardware and Embedded Systems **2020**(3), 147–168 (Jun 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. Wu, L., Weissbart, L., Krček, M., Li, H., Perin, G., Batina, L., Picek, S.: On the attack evaluation and the generalization ability in profiling side-channel analysis. Cryptology ePrint Archive, Report 2020/899 (2020), https://eprint.iacr.org/2020/899
28. Zaid, G., Bossuet, L., Habrard, A., Venelli, A.: Methodology for efficient cnn architectures in profiling attacks. IACR Transactions on Cryptographic Hardware and Embedded Systems **2020**(1), 1–36 (Nov 2019). https://doi.org/10.13154/tches.v2020.i1.1-36, https://tches.iacr.org/index.php/TCHES/article/view/8391