To Overfit, or Not to Overfit

Improving the Performance of Deep Learning-Based SCA

Rezaeezade, Azade; Perin, Guilherme; Picek, Stjepan

DOI
10.1007/978-3-031-17433-9_17

Publication date
2022

Document Version
Final published version

Published in
Progress in Cryptology - AFRICACRYPT 2022 - 13th International Conference on Cryptology in Africa, AFRICACRYPT 2022, Proceedings

Citation (APA)
Rezaeezade, A., Perin, G., & Picek, S. (2022). To Overfit, or Not to Overfit: Improving the Performance of Deep Learning-Based SCA. In L. Batina, & J. Daemen (Eds.), *Progress in Cryptology - AFRICACRYPT 2022 - 13th International Conference on Cryptology in Africa, AFRICACRYPT 2022, Proceedings* (pp. 397-421). (Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics); Vol. 13503 LNCS). Springer. https://doi.org/10.1007/978-3-031-17433-9_17

To Overfit, or Not to Overfit: Improving the Performance of Deep Learning-Based SCA

Azade Rezaeezade[1(✉)], Guilherme Perin[1,2], and Stjepan Picek[1,2]

[1] Delft University of Technology, Delft, The Netherlands
a.rezaeezade-1@tudelft.nl
[2] Radboud University, Nijmegen, The Netherlands

Abstract. Profiling side-channel analysis allows evaluators to estimate the worst-case security of a target. When security evaluations relax the assumptions about the adversary's knowledge, profiling models may easily be sub-optimal due to the inability to extract the most informative points of interest from the side-channel measurements. When used for profiling attacks, deep neural networks can learn strong models without feature selection with the drawback of expensive hyperparameter tuning. Unfortunately, due to very large search spaces, one usually finds very different model behaviors, and a widespread situation is to face overfitting with typically poor generalization capacity.

Usually, overfitting or poor generalization would be mitigated by adding more measurements to the profiling phase to reduce estimation errors. This paper provides a detailed analysis of different deep learning model behaviors and shows that adding more profiling traces as a single solution does not necessarily help improve generalization. We recognize the main problem to be the sub-optimal selection of hyperparameters, which is then difficult to resolve by simply adding more measurements. Instead, we propose to use small hyperparameter tweaks or regularization as techniques to resolve the problem.

Keywords: Side-channel analysis · Deep learning · Overfitting · Generalization

1 Introduction

Side-channel analysis (SCA) encompasses non-invasive attacks exploring unintentional information leakage from cryptographic devices. Consequently, it is of interest of manufacturers to evaluate the robustness of their products against threats posed by multiple side-channel attacks. Such attacks are classified as profiling [9] and non-profiling attacks [15]. The latter appeared first in late 1990s s and rapidly after publication, protection mechanisms (countermeasures) started to be proposed [8,19] and implemented to mitigate differential power attacks and their variations [5,12]. On the other hand, profiling attacks allow a more

L. Batina and J. Daemen (Eds.): AFRICACRYPT 2022, LNCS 13503, pp. 397–421, 2022.
https://doi.org/10.1007/978-3-031-17433-9_17

formal security evaluation of a device by implementing a profiling model from side-channel measurements obtained from a device identical to the device under evaluation. The model is later tested on a separate target as a secret recovery or attack phase. Since an evaluator can learn a statistical model from the existing leakages, several countermeasures against non-profiling attacks may become ineffective against profiling attacks, such as Boolean masking [17,18]. This depends on how much knowledge an adversary has about the target implementation (e.g., secret random shares and source code). This addition of knowledge in profiling attacks allows the estimation of worst-case security scenarios. If profiling attacks cannot reduce the secret's entropy under these circumstances and by considering a sufficient number of measurements, the target can be considered as secure.

Theoretically, the Gaussian template attack [9] is assumed to be the strongest profiling method in SCA. This is true if 1) an adversary can extract the most informative (leaking) points of interest (features) from side-channel measurements, 2) the true leakage distribution (which is unknown) follows a normal distribution, and 3) the adversary has an unlimited number of measurements to build the model. In recent years, deep learning appeared as a powerful alternative to implementing profiling models [17]. In practice, a deep neural network may skip feature selection and still provide strong models that defeat Boolean masking countermeasure. Additionally, the ability of Convolutional Neural Networks to bypass trace desynchronization effects [7] also became very attractive to the SCA community. The main drawback in deep learning (across all domains and not only SCA) is the expensive hyperparameter tuning process. Different solutions have been investigated to do this process in profiling attacks including grid search [30], random search [21], Bayesian optimization [28], genetic algorithms [17], and reinforcement learning [26]. Such solutions converge to different model behaviors, and usually, the best model is selected from a limited number of search attempts.

As the search spaces are usually very large, even for a few hyperparameters, the search mechanism can easily find multiple sub-optimal models for the problem. In other words, a large number of deep learning models will show overfitting or underfitting effects, leading to poor generalization ability. Finding an optimal model during the search is theoretically possible by assuming an unbounded adversary in terms of training resources, which is never the case in practice [24]. For profiling attacks in general, it is common to assume that a model is sub-optimal or wrong if the assumptions about the leakage model contain errors or the number of side-channel measurements is not sufficient [11]. This is because we aim to find the actual statistical distribution of the leakage in the profiling phase. Since this distribution is unknown, we must approximate it using density estimation techniques. The estimation process is aligned with assumption and estimation errors. The former is the consequence of incorrect assumptions about the leakage model, and the latter is the consequence of insufficient side-channel measurements [6]. Reducing any of those errors will result in the improvement of the model performance.

Intuitively, a deep learning model showing sub-optimal generalization would start to converge to a stronger model by increasing the number of profiling traces and re-training. The expectation is that fewer estimation errors (i.e., more profiling traces) would suppress eventual assumption errors made by the model (due to hyperparameters combination and leakage model).

This paper examines this common profiling attack assumption that more profiling traces always improve model generalization. Motivated by the deep double descent phenomenon [20], we experimentally show that poor generalization (typically justified as overfitting and underfitting) is much more a matter of specific hyperparameters choice and not the lack of profiling traces. We generate models ranging from few thousand to millions of trainable parameters, and we verify that in several cases, independent of leakage model and deep learning model size, adding more profiling traces does not change (or even reduce) the model's generalization ability. We investigate simple techniques based on limited hyperparameter changes and regularization to override this unexpected effect. Additionally, understanding what causes unexpected model behaviors in deep learning (such as overfitting) allows evaluators to draw more consistent conclusions about the target's security, reducing the chances of overestimating the security of a device due to assumptions errors of the model. Our main contributions are:

1. We experimentally verify that the assumption that "adding more profiling traces increases the generalization of a model" is not always true for deep learning-based SCA. We investigate in detail the overfitting effect in profiling attacks. We validate that in many cases, the overfitting effect can be reduced by adding more profiling traces during training if the chosen hyperparameters combination does not result in large assumption errors.
2. We run experiments for software and hardware AES datasets and demonstrate that this behavior is not dataset dependent.
3. Considering the first contribution, we show that when increasing the profiling set size does not improve generalization, using regularization techniques and applying small changes in hyperparameters can improve the performance by reducing the assumption errors.

This work does not aim to find any specific model that will break a target with fewer measurements than state-of-the-art. Rather, it provides insights into a commonly observed problem in deep learning-based SCA: overfitting. Instead of simply concluding that a model does not work due to overfitting, as done in most related works, we try to find the underlying reasons for such behavior and how to mitigate it.

2 Background

2.1 Profiling SCA

Profiling attacks are conducted in two main phases: profiling and attack. The main goal of the profiling phase is to implement a classifier by learning or computing a set of parameters. The first method for profiling attacks is the *template*

attack, where an adversary assumes that the leakage follows a multi-variate Gaussian distribution [9,10]. The profiling stage consists then of computing the Gaussian components (mean and covariance) as approximate statistics by assuming that the target device provides leakages that follow a Gaussian distribution. This allows an adversary to obtain a probability density function representing the distribution of side-channel leakages.

In machine learning (including deep learning) SCA, the statistics of the unknown leakage distribution are automatically learned from the profiling set. This time, the classifier parameters are learned from side-channel leakages rather than computed, meaning that the adversary does not necessarily need to assume the statistical distribution that describes the leakages well.

Although the Gaussian template attack theoretically represents the strongest profiling model due to the nature of side-channel leakages that tend to follow a Gaussian distribution, deep learning methods offer infinite opportunities to learn a classifier and, therefore, in practice, can reach more efficient results. If points of interest can be selected to build Gaussian templates, a security evaluator can evaluate the worst-case security scenarios by estimating the required number of side-channel measurements to recover the secret. On the other hand, deep learning methods require no feature selection as the points of interest are implicitly selected during the deep learning training phase.

The quality of a profiling classifier is given by its generalization ability to an attack set. For that, information-theoretic metrics like guessing entropy or success rate are computed from the output probabilities obtained from the classifier. Differing from the template attack where the adversary or the evaluator does not need to conduct hyperparameter tuning, deep learning methods suffer from the difficulty of defining those hyperparameters. Therefore, one needs to be careful when interpreting the outcome metrics from a profiling model and, consequently, the conclusions about the security of the implementation. The information-theoretic metrics help the evaluator to understand how to find the best possible deep learning model, and for that, it is important to understand when the metrics indicate overfitting, underfitting, and generalization.

2.2 Overfitting and Underfitting

In supervised learning, when we train a model, we fit it on a set of labeled data, and then we expect it to be able to decide about a set of unseen data from the same distribution. According to that, overfitting describes a model that fits the training data well but performs poorly on test data. It can be caused for various reasons, traditionally summarized as using a too complex model or using an inappropriate training set. In the same way, we can define underfitting as the model that does not learn the underlying function in training data, and its performance in test data is poor accordingly.

One suggested way to limit the effect of overfitting or improve the generalization power of a model is to expand the training set by collecting more data or running data augmentation [29]. However, there are limitations as traditional machine learning (and small neural network) models' performance saturates at

some point, and we cannot see significant changes from that point ahead as we increase the training set size. In the case of deep neural networks, performance variation regarding increasing training set size depends on the proportion between the network's capacity[1] and the number of traces in the training set. Although the exact behavior of the neural models is still ongoing research, it has been shown that when the size of the training set and model's capacity is comparable, increasing the training set size can hurt the model's performance [20].

Deep learning models commonly have many trainable parameters [2], and such models can easily fit training data with empirical risk (average of the loss on the training set) close to zero. This simplifies the optimization problem so that we can use very complex models for simple problems and still reach extremely good generalization [31].

2.3 Deep Double Descent Phenomenon

The deep double descent phenomenon was first discovered by Belkin et al. [3]. The challenge started with the common claim in modern machine learning that "larger models are always better" [13,16,27], while standard statistical machine learning theory predicts that larger models are candidates for overfitting. Belkin et al. unified the classical bias-variance trade-off and the modern practice of larger models' advantage. They showed how increasing models' capacity beyond the size of the smallest model that can fit the training data with zero empirical risk could lead to performance improvement.

Bias-variance trade-off is the concept traditionally used to describe underfitting as a result of high bias and overfitting as a result of high variance. The conventional bias-variance trade-off is shown on the left of Fig. 1. Choosing the function class $\mathcal{F}$, like neural networks, and minimizing their training risk using an objective function, the training and test risk varies based on the capacity of $\mathcal{F}$. In this trade-off, there is a "sweet spot" where a model with a specific capacity has the best performance for the given problem. Before the "sweet spot", performance increases with increasing the capacity of the models. After the "sweet spot", the performance of the models in the training set keeps increasing, but performance in the test set decreases.

More recently, deep double descent replaced the conventional bias-variance trade-off with the neural model performance below the "interpolation point", where the interpolation point is the capacity of the smallest model from function class $\mathcal{F}$ that can fit a training set with n traces and zero training loss. Belkin et al. showed that although models with a capacity near the interpolation point have a high test risk (average of the loss in the test set), if we keep increasing the model's capacity, the risk will decrease again, and we can even reach models with better performance than the models in the "sweet spot". One can see these double descent behavior in the right of Fig. 1. All the models beyond the interpolation point fit the training data perfectly and have zero training risk.

[1] We consider the capacity to be the size of the network in trainable parameters.

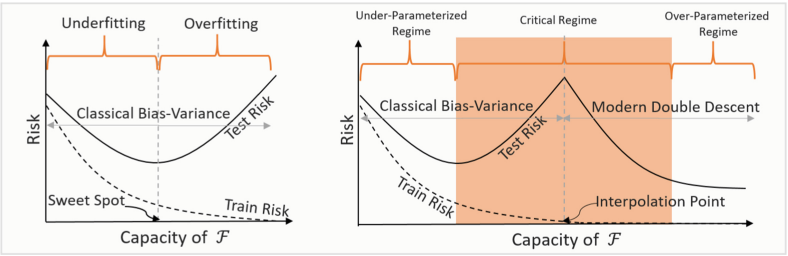

Fig. 1. Curves for training risk (dashed line) and test risk (solid line). Left: The classical U-shaped risk curve. Right: The Double Descent risk curve.

Considering this deep double descent phenomenon, a model f from function class $\mathcal{F}$ works in one of the following regimes, depending on the proportion between the capacity of the model and the number of measurements in the training set (n):

- Under-parameterized regime: where the capacity of f is sufficiently smaller than n. The models working in this regime show an increase in performance by increasing the training set size, i.e., increasing the number of measurements in the training set until the saturation point (where increasing the training set size does not improve the model's performance anymore).
- Over-parameterized regime: where the capacity of f is sufficiently larger than n. The models working in this regime show an increase in performance by increasing training set size unless the increase can change the regime that the model is in.
- Critically-parameterized regime: where the capacity of f and the size of the training set n are comparable. Models working in this regime may show an increase or decrease in performance by increasing the training set size. In the critical regime, the chosen model barely fits the training data, so it is fragile[2]. In the right of Fig. 1, by increasing the model's capacity, the test risk initially increases in the critical regime, and just as the model reaches the interpolation point and can fit with zero training risk, it undergoes the second descent. In many research works, this effect cannot be observed as it is avoided through early stopping or other regularization techniques[3].

In the right of Fig. 1, the critical regime is denoted in orange color. As one can see, the test risk reaches its peak at the interpolation point. Meanwhile, the training risk reaches zero for the first time. After that point, the training risk remains zero, and the test risk starts to decrease again. Increasing the training set size pushes the double descent curve downward by decreasing test risk. However, since a larger training set requires larger models to fit, increasing the training

[2] Small changes in the model's hyperparameter can invalidate it.
[3] Regularization are techniques used to mitigate overfitting by reducing the complexity of the models.

set also shifts the interpolation point (and the peak of test risk) to the right [20]. When we increase the training set size, these two effects combine, and we may observe that training a model with a fixed capacity with less training data shows better performance than training it with a larger training set. Consequently, increasing the training set size can hurt the performance of the model.

2.4 Overfitting and Generalization in Side-channel Analysis

To decide about the generalization power of neural network models in the side-channel domain, we cannot rely on accuracy and loss for the training and validation set. As discussed in [23], such machine learning metrics are not suitable in SCA as they limit the final prediction to individual decisions for each trace. Therefore, we need a metric that can accumulate the small biases of the model prediction toward the correct key. To overcome this shortcoming, besides the accuracy and loss of the model, we inspect the evolution of guessing entropy for an attack set. Following a standard differential power analysis setting, we accumulate information about each key candidate as $S_k = \sum_{i=1}^{Q} \log p(x_i, y)$, where Q is the number of attack traces and $p(x, y)$ indicates the probability of a trace x_i to be represented by an intermediate (in our case, a label) y. For example, the intermediate y can be the output of an S-box operation in the AES cipher. Namely, $y = Sbox(d \oplus k)$, where d is a plaintext byte and k is the key byte candidate. The best key candidate is the key that maximizes S_k. If we sort the key candidates according to their S_k values, guessing entropy (GE) is the average rank of the actual key among all the candidate keys after multiple attacks to an attack set. The attack is successful if it places the actual key first, i.e., $GE = 1$.

Detecting Overfitting and Underfitting in SCA. The inability of a deep neural network to generalize in profiling SCA is usually attributed to model overfitting or underfitting. In practice, overfitting is characterized by a model that memorizes all training data but cannot generalize to different (unseen) attack traces. The main outcome is a model that mostly fits noise instead of existing leakages. This scenario becomes even more critical in realistic profiling attacks where the attack traces are collected from a separate device. Underfitting happens when the model is trained for an insufficient number of iterations (i.e., epochs), and training metrics indicate a model that cannot even memorize training data.

A profiling model is considered sub-optimal if there are assumption or estimation errors [11]. In the case of deep learning, assumption errors also include hyperparameter combinations that lead to insufficient generalization. However, there are cases when the trained model still eventually recovers the correct key at the price of the increased number of attack traces. Thus, it is common to observe the following scenarios:

– Model overfits but still recovers the correct key information: in this case, training accuracy reaches 100% (resp. training loss approaches zero) at the end of the training process. The overfitting is identified by checking the validation accuracy, which is typically close to random guessing, and the validation loss,

which tends to grow as training continues. However, when processing a sufficient number of attack traces during the GE calculation, it is still possible to rank the correct key as first (or among the first ones). Of course, as the model overfits and the GE is usually verified for side-channel measurements obtained from the same device (ignoring portability [4]), one cannot assume the model provides sufficient generalization.
- Model *tends* to overfit but still recovers the correct key byte: the only difference from the previous situation is that the model shows a continuous improvement in training accuracy (resp., decreasing in loss), but the training stops before it reaches 100%. As this stopping decision happens before the complete overfitting, the tendency is a slightly better (but still far from optimal) generalization.
- Model overfits or tends to overfit and does not recover the correct key byte: this is a case when the evaluator selects a combination of hyperparameters that creates a model that cannot fit existing leakages and predominantly fits noise from side-channel measurements.

The final profiling models achieved through any of these scenarios are defined as sub-optimal models. Even the scenarios that recover the secret information are far from ideal for security evaluations. The problem is to assume that a sub-optimal model represents the existing leakage distribution of the target. The direct consequence is a false sense of security, usually overestimating the security of the implementation. *Additionally, it remains unclear if the increase in the number of profiling traces, which reduces estimation errors, always ensures the reduction of negative effects due to the model's sub-optimality.*

Finding the Best Possible Models. A typical procedure in deep learning-based profiling attacks is to run hyperparameter search methods to find the best possible models using available resources (processing power, time, and memory). To speed up the hyperparameter search process, an alternative is to reduce the number of profiling traces during the search and select the best possible model. Once the best model is found, one can increase the number of profiling traces again and re-train the model to improve its generalization. In essence, for some models, adding more profiling traces requires small changes in hyperparameters to accommodate the larger profiling set and the expected improvement in generalization. As it is more likely that we are dealing with sub-optimal models (we do not follow the worst-case security settings), generalization will be limited and will also happen in models showing overfitting.

3 Experimental Setup

3.1 Datasets

We use two publicly available datasets that are large enough to run our experiments. This will allow us to inspect the influence of the increasing number of traces on the neural network model behavior.

ASCAD Random Keys[4]**.** This dataset was released in 2019 [1] and contains side-channel measurements from an AES-128 software implementation running on an 8-bit AVR microcontroller. Details about the cryptographic design are provided in [25]. The AES implementation is protected with a first-order Boolean masking countermeasure. This dataset contains 200,000 traces with random keys and random plaintexts, which is then considered as a profiling set. Additional 100,000 traces with a fixed key are considered for attack and validation sets. Each side-channel measurement contains 250,000 features, and we consider the trimmed version of the dataset containing 1,400 features. This target interval represents the processing of the third byte (the first masked one) of the S-box in the first encryption round. Therefore, in our experiments, this dataset is labeled according to this intermediate byte, i.e., $Y^{(i)} = Sbox[P_3^{(i)} \oplus k_3^{(i)}]$ (Identity (ID) leakage model) and the Hamming weight of this intermediate byte (Hamming weight (HW) leakage model).

AES_HD[5]**.** This dataset was introduced in [23] and contains power side-channel measurements from an unprotected AES_128 implementation running on an FPGA platform. This dataset contains 500,000 side-channel traces (in [23], the authors considered a smaller version of the dataset) with a fixed key. In our case, we split this dataset in a profiling set containing 400,000 traces and the remaining 100,000 are used for attack and validation phases. Each measurement in this dataset contains 1,250 features. We label this dataset according to the Hamming distance (HD) between output ciphertext byte and the corresponding byte before S-box in the last encryption round, $Y^{(i)} = Sbox^{-1}[C_j^{(i)} \oplus k_j^{(i)}] \oplus C_{j'}^{(i)}$, where $C_j^{(i)}$ and $C_{j'}^{(i)}$ are related according to the ShiftRows operation. In our experiments, $j = 10$ and $j' = 6$.

3.2 Analysis Methodology

In profiling SCA, an analysis that makes correct assumptions about the leakage model and leakage distributions can increase models' performance by increasing the number of profiling traces. In Sect. 2.3, we discussed that recent deep learning results show the existence of a critical regime, where increasing the number of training data leads to an unexpected decrease in model generalization. Not surprisingly, this also happens in the side-channel domain.

This paper explores the effects mentioned above in the case of deep learning-based SCA. Our analysis methodology aims to verify the possibilities of reducing the overfitting effects in profiling models and understanding if a deep learning model trained with more data should always deliver better generalization. Our approach consists of the following steps:

[4] https://github.com/ANSSI-FR/ASCAD/tree/master/ATMEGA_AES_v1/ATM_AES_v1_variable_key.

[5] https://github.com/AISyLab/AES_HD_Ext.

1. A search space $\mathcal{S}$ is created for hyperparameter options for MultiLayer Perceptron (MLP) and Convolutional Neural Networks (CNN)[6].
2. 500 different hyperparameters combinations are generated for MLP and CNN neural network topology and leakage model (Hamming weight, Identity, and Hamming distance) combinations and two datasets (ASCAD Random Keys and AES_HD). Thus, the total number of generated models equals 3,000.

Table 1. *MLP* and *CNN* models hyperparameters.

Hyperparameters	Range
Dense layers	
Number of neurons	[100, 900], step = 100
Number of layers	[1, 8], step = 1
Convolution layers	
Number of layers	[1, 4], step = 1
Number of kernels	[4, 20], step = 1
First layer's filter size	[2, 4, 8, 12, 16]
$i^{(}th)$ layer filter size	$((i-1)filter\_size)^2$
Pooling	"Average", "Max"
Pooling size	[2, 10], step = 2
Pooling stride	[2, 10], step = 2
Learning hyperparameters	
Optimizer	"Adam", "RMSprop"
Weight initialization	"random_uniform", "glorot_uniform", and "he_uniform"
Activation function	"relu", "selu", and "elu"
Batch size	$[100, 900]$, $step = 100$
Learning rate	$[0.005, 0.001, 0.0005, 0.0001, 0.00005, 0.00001]$

To generate the models, we considered the hyperparameters and ranges depicted in Table 1. For both MLP and CNN dense layers, we used the ranges shown in *Dense layers* part of Table 1. The number of epochs for all settings is fixed and equals 200. The hyperparameters' ranges are being selected based on the ranges reported in the previous researches [1,22,30].

3. In the first phase, each randomly generated neural network is trained with a portion of the available profiling set. We start with 50,000 profiling traces, and the performance of each profiling model is estimated with the average over ten different validation sets, where each has 10,000 traces. For guessing

[6] Due to the limitation on the number of pages of the paper, we avoid describing MLP and CNN architectures here. We refer interested readers to many available papers in the literature describing these neural network models like [25,30].

entropy and the required number of attack traces, we repeat the attack 100 times in each validation set for 5,000 randomly selected traces and report the average over those ten sets.

4. If $GE = 1$, the neural network model is selected as a *good* model. Otherwise, the model is discarded. To avoid subjective interpretation of the term *good model*, we clarify that this is not an optimal profiling model but a trained neural network that can recover the target key with the considered number of traces (i.e., 5,000).
5. For all good models, we analyze the training metrics to verify what regime the model is working in and if the profiling model overfits (or tends to overfit). The schematics of the overfitting or tending to overfit behaviors of the deep models are illustrated in Fig. 2.
6. All good models are re-trained with an increased number of profiling traces. We re-train the good models with 100,000, 150,000, and 200,000 traces for the ASCAD Random Keys dataset. For the AES_HD dataset, we repeat the training process with 100,000 up to 400,000 traces with steps of 50,000 traces. Note that the models are always re-initialized with the same trainable parameters from step 2. This will keep all the other hyperparameters and settings the same and provide us with the possibility to observe the effect of profiling size increments.
7. After re-training the good models with more profiling traces, we verify if the required number of attack traces to reach $GE = 1$ is reduced when compared to the baseline case (model trained with 50,000). We assume that this reduction in the required number of traces reflects a generalization improvement. This can be justified from two viewpoints. First, from a deep learning perspective, the improvement in the validation accuracy results in assigning the largest probability to the correct key more frequently than in cases when the model shows less accuracy in the validation set. From a side-channel perspective, we are interested in models that can rank the correct key first with fewer attack traces, and this can be a measure to compare the generalization power of different models.
8. For models where adding more profiling traces improves their generalization (this situation is illustrated in the top part of Fig. 3), one of the following situations might happen:
   (a) The baseline model was working in an under-parameterized regime, and the hyperparameter combination error (assumption error) was small compared to the neural network model's estimation error. As a result, increasing the number of profiling traces leads to decreasing estimation error and better generalization.
   (b) The baseline model was working in an over-parameterized regime, and the assumption error was small compared to the estimation error. As a result, an increasing number of profiling traces leads to decreasing estimation error and better generalization.
   (c) The baseline model was working in a critical regime. Increasing traces in the training set did not invalidate the neural network model, and the hyperparameter combination error did not increase with an increasing

number of profiling traces. Instead, estimation error decreased by adding more traces to the training set, leading to decreasing estimation error and better generalization.

9. For models where adding more profiling traces does not improve the generalization (this situation is illustrated in bottom part of Fig. 3), one of the following situations might happen:
   (a) Regardless of the regime the baseline model is working in, the specific hyperparameters combination leads to assumption errors after increasing the profiling set size. The model needs to be adjusted to reduce the assumption error effects.
   (b) The baseline model is working in the critical regime, and adding more traces to the profiling set skewed the model and decreased its performance.

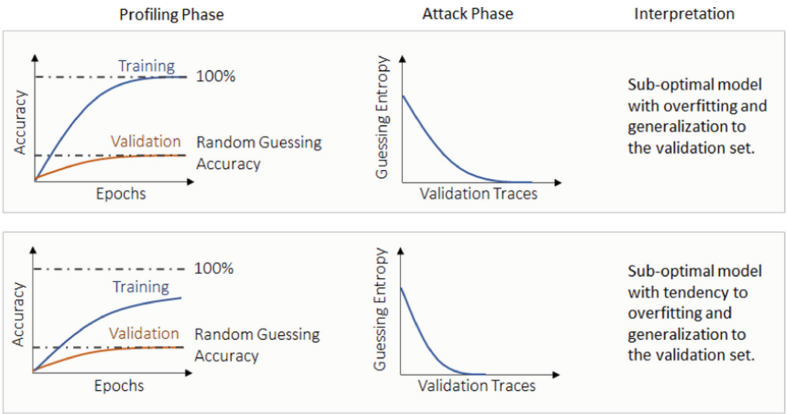

Fig. 2. Common model behaviors.

Figure 2 illustrates a typical scenario identified in [23] for sub-optimal deep learning models when machine learning metrics do not reflect the performance of a model concerning guessing entropy. The fact that we illustrate accuracy as a reference machine learning metric does not imply that we are ignoring training and validation loss to decide if a model shows the potential behaviors from Fig. 2. As proposed in [18], minimizing the cross-entropy loss should be the equivalent of solving side-channel analysis optimization. However, this principle usually does not apply to sub-optimal models with too many assumption errors, as is the case of models showing overfitting. It is important to note that for most of the models considered good models, the situation in the top part of Fig. 3 happens more often. This means that for those models, overfitting is also caused by the lack of profiling traces, leading to insufficient profiling. On the other hand, if a model shows the behavior from the bottom part of Fig. 3, we can modify some hyperparameters that are unrelated to the model size (e.g., learning rate, batch size, training epochs, dropout, and activation functions), which can cause the model to reduce its assumption errors.

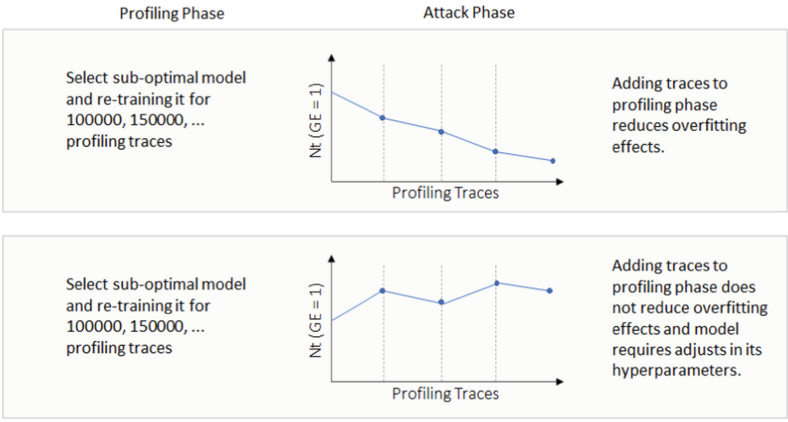

Fig. 3. Interpreting the model behaviors.

4 Experimental Results

Considering the neural network capacity and the size of the profiling set, we should specify the regime that the neural network model is working on. Unfortunately, we cannot clearly specify the borders between these regimes. This is because the interpolation point definition and "sufficiently larger" and "sufficiently smaller" terms in the definition of over-parameterized and under-parameterized regimes cannot give a clear separation. Additionally, there is no widely accepted definition for the capacity of the models in deep learning, which can justify the different performance of neural network models with a similar number of trainable parameters but different topology and output layer size. To tackle these ambiguities, we selected "good" neural network models with different numbers of trainable parameters ranging from a few thousand to a few million. Then, we consider the training and validation accuracy and loss evolution to roughly specify the interpolation point for different neural network and leakage model combinations. Finally, we can roughly specify the working regime for every selected neural network model. Still, considering that the specified interpolation points are valid for the specific profiling size, by increasing the profiling size, the working regime of the model may change [20]. In the tables, every row represents a randomly selected neural network model, denoted with numbers. Observe that there are some models that have the same number of trainable parameters (denoted as "Params"). This simply means that different architectures have the same capacity. For various settings (datasets, leakage models), we give various numbers of models based on the number of models that converge and have different capacities (to provide good coverage from small to large model sizes).

4.1 ASCAD Random Keys Dataset

Four different combinations of MLP and CNN topologies with the random structure declared in Sect. 3, and the Hamming weight (HW) and Identity (ID) leakage models are investigated for the ASCAD dataset. The results of experimenting with the first combination (MLP topology and the HW leakage model) are shown in Table 2a. To compare the performance of a neural network model trained with four different profiling sets (50,000, 100,000, 150,000, and 200,000), we consider the required number of attack traces for each profiling set to place the actual key in the first place. Then refer to them as NT_1[7] for profiling set with 50,000 traces to NT_4 for profiling set with 200,000 traces for each neural network model. As mentioned in Sect. 2.4, the performance of the models that need fewer traces to reach $GE = 1$ is considered to be better.

Table 2. Results for multilayer perceptron.

#	*Params*	NT_1	NT_2	NT_3	NT_4
1	14,109	4,856	4,721	1,932	2,352
2	14,549	2,419	5,000	520	631
3	28,209	3,141	1,577	1,225	1,035
4	31,149	3,615	5,000	622	5,000
5	42,309	4,045	2,215	1,029	972
6	44,169	4,336	893	654	561
7	58049	4,034	2,259	1,662	1,324
8	70,509	4,435	1,712	899	745
9	78,159	3,385	1,601	1,306	1,044
10	85,809	4,742	2,575	1,906	1,286
11	141,009	3,859	1,671	1,020	801
12	161,209	4,070	1,792	1,063	939
13	181,409	3,357	1,895	1,917	1,830
14	282,009	3,088	1,420	1,078	912
15	322,209	4,391	2,032	1,148	907
16	402,609	2,559	1,601	891	1,139
17	523,209	4,419	2,515	1,322	1,241
18	693,909	3,698	1,426	1,140	1,046
19	724,409	4,276	2,238	1,511	1,368
20	884,809	3,217	2,508	1,848	1,606
21	964,809	1,279	870	1,016	1,355
22	1,206,009	3,117	1,551	1,158	970
23	1,526,409	3,535	1,780	1,298	1,305
24	1,707,009	3,205	2,619	5,000	1,111
25	1,957,509	4,751	4,504	2,143	1,911
26	2,208,009	4,842	2,620	3,038	2,278
27	2,458,509	3,450	1,744	1,072	893

(a) *MLP* trained for the *HW* leakage model.

#	*Params*	NT_1	NT_2	NT_3	NT_4
1	68,176	4,840	810	225	123
2	186,156	4,999	1,185	337	168
3	196,256	4,865	1,373	514	241
4	226,556	4,769	1,814	469	248
5	331,656	4,896	4,450	1,742	808
6	371,856	4,991	4,775	3,609	1,615
7	371,856	4786	3208	1370	757
8	412,056	4,937	1,837	543	325
9	412,056	4,650	1,877	464	282
10	497,356	4,912	3,114	1,300	1,263
11	497,356	4,864	3,161	1,104	433
12	587,656	4,717	3,093	1,625	770
13	663,056	4,821	3,406	1,343	713
14	663,056	4,937	3,048	954	498
15	823,456	4,989	2,549	1,121	576
16	828,756	4,858	2,675	1,129	615
17	828,756	4,900	3,543	1,620	732
18	1039,156	4,719	1,047	458	281
19	1079,256	4,798	4,005	2,132	1,105
20	1,129,456	3,544	1,244	561	426
21	1,129,456	2,571	1,026	1,108	402
22	1,129,456	4,829	3,413	2,601	1,333
23	1,329,756	4,986	4,672	3,673	2,720
24	1,329,756	4,836	3,759	2,440	1,357
25	1,580,256	5,000	3,351	2,581	1,355
26	158,0256	4,651	2,928	905	4,818
27	1,785,856	2,271	5,000	5,000	5,000

(b) *MLP* trained for the *ID* leakage model.

[7] NT: Average of the minimum number of attack traces that the model needs to place the actual key first.

Observe how all the neural network models ranked the correct key in the first place when trained with 50,000 traces. In many cases, by increasing the profiling size for each MLP model, the required number of traces to reach $GE = 1$ follows the classical machine learning belief that more data is better (for example, #6). Still, in some cases, the performance of the neural network model (the required number of attack traces) decreased when increasing the profiling set size. A part of this behavior has been predicted by deep double descent, and the proportion between the model's size and profiling set size, especially when increasing profiling size, changes the regime the neural network works in (e.g., #13 in Table 2a). This model works near over-parameterized regime boundaries with 50,000 profiling traces since its performance in the profiling set is $\approx 97\%$. Then, by increasing the profiling set size to 200,000, its performance decreases to $\approx 60\%$. Considering the proportion between the model's size and the profiling set size, this model is working in the critical regime for 100,000, 150,000, and 200,000 profiling traces. Still, the irregular increases and decreases in the required number of attack traces considering the model's complexity are not fully justified by the double descent phenomenon. For example one can consider #4 in Table 2a. The accuracy of the baseline model is $\approx 45\%$, and for the rest of the profiling set sizes, its profiling accuracy is less than 30%. Considering the proportion between this model size and the profiling set sizes, and the profiling accuracy, the model works in an under-parameterized regime, and an increasing number of profiling traces does not change the working regime. However, when we train the model with 100,000 and 200,000 profiling traces, it cannot reduce the key entropy at all as the model does not fit the leakage anymore.

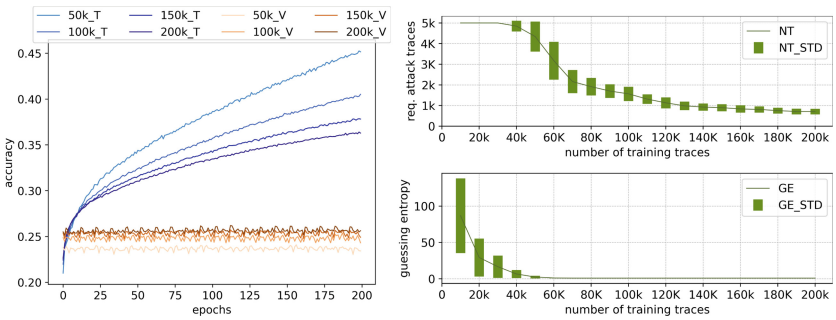

Fig. 4. Increasing the profiling set size and accuracy in the training (T) and validation (V) set and the required number of attack traces to reach $GE = 1$, #8 in Table 2a.

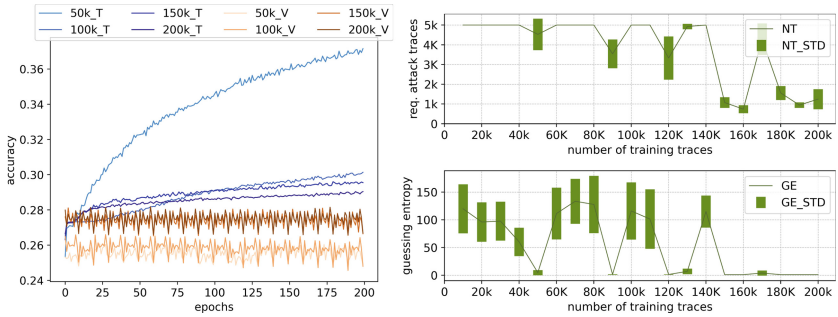

Fig. 5. Increasing the profiling set size and accuracy in the training (T) and validation (V) set and the required number of attack traces to reach $GE = 1$, #2 in Table 2a.

One can observe the changes in the performance for two MLP architectures in Figs. 4 and 5. The lighter colors in the left plot show accuracy for profiling (blue color) and validation (orange color) sets for MLPs trained with smaller profiling sets, and sequentially darker colors show these metrics after increasing the profiling set sizes to 100,000, 150,000, and 200,000 traces. Considering the profiling and validation accuracy and the proportion between the model's size and the profiling set size for #8 in Fig. 4, we can see that the baseline model works in the critical regime, and increasing the profiling set size pushes this model toward the under-parameterized regime. To eliminate the influence of a biased attack set on the required number of attack traces, we evaluated 10 different validation sets and provided mean and standard deviation values. In the right and top of Fig. 4, one can see the mean and variance of the required number of attack traces for MLP #8 to decrease GE to 1. In the right and bottom of Fig. 4, one can see the average of GE that the model managed to reach with 5,000 traces[8]. In Fig. 4, increasing profiling size results in better performance concerning the accuracy, the required number of attack traces, and the final GE that the model managed to reach with 5,000 attack traces. We can even detect small changes in the required number of attack traces from 150,000 to 200,000 profiling traces, which can be a sign of saturation of the model.

MLP #2 is another example that works in the under-parameterized regime. In Fig. 5, the accuracy of the MLP model shows normal (more profiling traces increase accuracy) behavior when increasing profiling set size. The training accuracy of the baseline model is $\approx 38\%$ and decreases to 28% for the profiling set with 200,000 traces. This model's test accuracy varies from 24% for the baseline model to $\approx 28\%$ when we train it with 200,000 traces. These accuracy percentages and the proportion between the model's size and the profiling set sizes indicate that this model works in the under-parameterized regime for all profiling set sizes. Still, this model cannot reduce the key entropy when trained with 200,000 traces.

Considering MLP #2 in Table 2a, while this MLP model converges to $GE = 1$ with on average 2,419 attack traces for a profiling set of 50,000 traces, it can-

[8] We took incremental steps of 10,000 in case of MLP #8 and MLP #2 to make the behavior tracking easier in Figs. 4 and 5.

not converge at all, i.e., cannot rank the correct key better than random guessing after increasing profiling size from 50,000 to 100,000 traces. The behavior is even more surprising when the model again converged to $GE = 1$ with on average 520 attack traces when it is trained with 150,000 profiling traces. This is an example of a scenario when the hyperparameter combination contains an assumption error and increasing the profiling size invalidates the model, i.e., changes the weights of the neural network model in a way that it cannot capture the underlying leakage model distribution and find the correct key. However, there are cases like #1 (works in the under-parameterized regime) and #26 (works in the over-parameterized regime), where increasing the profiling set size does not change the performance according to the required number of attack traces or sometimes makes it a bit worse compared to a smaller profiling set. In these cases, the assumption has an error, but it has been suppressed by increasing the profiling set size. MLP #23, #25, and #27 in Table 2a work in over-parameterized regime. Their profiling accuracy is more than 95% for all the profiling set sizes. Besides, the sizes of these models are at least six times larger than the largest profiling set size for ASCAD. In the case of MLP #24 and MLP#26, while the models have large sizes, the baseline model cannot reach more than 90% accuracy and, after increasing profiling size to 200,000 traces, the accuracy decreases under 70%. Thus, these models work in the critical regime for all the profiling sizes because the hyperparameter combination does not allow the models to fit the leakage perfectly.

Similar behavior is observed for multilayer perceptrons trained for the ID leakage model. In Table 2b, one can see the required number of attack traces for MLP models trained with 50,000, 100,000, 150,000, and 200,000 traces. There are many cases that follow the more data better generalization principle, but we can see cases that do not show a decrease in the required number of attack traces when we increase the profiling set size. Considering the size of the input layer, which is 1,400 (number of features for the ASCAD dataset), and the size of the output layer, which is 256 (number of classes for the ID leakage model), the smallest model that we managed to find that converges to $GE = 1$ has 68,176 trainable parameters and the baseline model has $\approx 10\%$ accuracy. Then the profiling accuracy decreases to $\approx 3\%$, which is still larger than the validation accuracy ($\approx 1\%$ but still larger than random guessing in the case of the Identity leakage model). MLP #23, #24, and #26 work in an over-parameterized regime for the same reasons as models in Table 2a.

We investigated CNNs and the Hamming weight and Identity leakage models to check if this behavior is still observable. The trained models and the required number of attack traces in this experiment can be analyzed in three regimes. For example, CNNs #1 to #7 in Table 3a work in under-parameterized regime. This can be concluded by considering the profiling accuracy of these models, which is less than 35%, and the proportion between the model sizes and the profiling set sizes. These models' accuracy and loss evolution for the profiling and validation sets do not show significant overfitting, meaning that these models do not have the capacity to memorize the noise in the profiling set. This can be considered as an indication that the number of trainable parameters does not represent the capacity of the models in a way that we can compare models from two different

families (cf. results for the MLP and the ID leakage model). To confirm this, we draw attention to the fact that the smallest successful models, in the case of CNNs, have less than 10,000 trainable parameters.

Neural network models like #3 in Table 2a, and #1 and #7 in Table 3a are working in under-parameterized regime and follow the "more data better generalization" principle, but MLP models like #1, #2, and #4 in Table 2a and CNN models like #2 and #5 in Table 3a show a decrease in their generalization power after increasing the profiling set size. Their hyperparameter combination imposes an assumption error on the final model, and the neural network model cannot capture the actual leakage model's distribution. To resolve this, we need to change the hyperparameters so that the final model's parameters can capture the actual leakage model's distribution. The results in Sect. 4.4 show that in a considerable number of cases, a small change in hyperparameters, like changing the activation function or the number of epochs, can result in a neural network model that can capture the underlying leakage model distribution better.

The observations for the over-parameterized regime for MLP also hold for CNN (#23 and #24 in Table 3a). In the case of CNN models in Table 3b, many baseline models reach 100% accuracy, but then this accuracy decreases significantly by increasing the profiling set sizes. Consequently, none of these models work in the over-parameterized regime for all profiling sets. The models are being pushed into the critical regime by increasing the profiling set size.

Table 3. Results for convolutional neural networks.

#	*Params*	NT_1	NT_2	NT_3	NT_4
1	2,959	629	320	286	248
2	5,689	2,670	2,718	801	5,000
3	6,579	2,885	1,616	883	934
4	9,781	1,319	654	481	521
5	15,327	4,821	5,000	5,000	5,000
6	15,439	1,862	671	460	502
7	19,209	3,958	1,303	897	766
8	24,433	813	474	4,141	477
9	31,113	3,196	1,300	5,000	5,000
10	44,905	3,149	1,855	1,283	982
11	57,943	3,660	1,450	1,411	803
12	63,865	2,712	1,270	1,253	756
13	75,813	1,587	1,663	1,497	745
14	94,661	2,625	2,586	2,068	2,604
15	100,113	3,673	2,930	2,337	1,501
16	124,585	4,905	3,393	1,458	1,596
17	189,233	4,730	5,000	2,783	2,588
18	243,129	3,983	4,814	2,929	2,487
19	391,521	2,233	1,721	1,470	1,295
20	434,121	3,605	2,296	2,677	1,964
21	541,243	4,833	3,467	1,162	441
22	570,753	2,441	1,830	1,165	894
23	984,649	3,525	2,068	1,672	1,661
24	1,211,881	4,175	3,419	1,554	1,600
25	1,409,249	3,540	1,867	1,165	905

(a) *CNNs* trained for the *HW* leakage model.

#	*Params*	NT_1	NT_2	NT_3	NT_4
1	6124	1,676	843	938	329
2	6,350	1,007	5,000	5,000	157
3	8,390	4,310	5,000	5,000	106
4	10,646	2,473	598	5,000	5,000
5	13,274	379	156	60	52
6	27,140	1,211	111	47	29
7	32,978	5,000	3,320	319	57
8	54,254	5,000	735	126	55
9	66,584	5,000	1,469	558	293
10	71,272	4,003	199	5,000	5,000
11	86,304	5,000	2,139	351	157
12	91,080	5,000	4,964	5,000	933
13	114,564	4,998	2,636	706	292
14	134,146	3,058	622	1,546	439
15	154,696	4,045	1,194	115	58
16	192,120	5,000	581	270	5,000
17	202,792	3,239	1,095	422	308
18	290,536	2,943	5,000	5,000	3,073
19	356,572	4,767	943	577	195
20	466,312	5,000	168	5,000	5,000
21	598,838	1,946	693	466	249
22	662,432	5,000	1,017	516	287
23	707,656	5,000	3,988	1,914	4,482
24	718,128	5,000	837	196	82
25	828,774	5,000	3,379	2,764	967

(b) *CNNs* trained for the *ID* leakage model.

4.2 AES_HD Dataset

In the case of the AES_HD dataset, we trained 500 random MLP and 500 random CNN models with the Hamming distance (HD) leakage model and again selected some of the models that converged to GE equal to 1. The AES_HD dataset has twice the number of traces in the profiling set compared to the ASCAD dataset, and this allows us to investigate the model's generalization power after increasing the number of profiling samples for a longer interval. As observed in Table 4, we trained MLP models with profiling sets including 50,000, 100,000, 150,000, 200,000, 250,000, 300,000, 350,000, and 400,000 traces. The required number of attack traces to reach GE equal to 1 for each neural network model shows that the model's generalization did not increase for many cases when we increased the number of profiling traces. The experiment has been repeated for the CNN models, and the same behavior was observed, so we omit those results.

In most cases in Table 4, the models converge to GE equal to 1 for a similar number of attack traces. This means that the increase in the number of profiling traces can change the parameters of the models, and since the output probabilities of the models will change as a result of this, we can see the changes in the required number of attack traces for different profiling set sizes. However, these changes are not improvements in the attack performance as they do not indicate a successful attack performance. On the other hand, the choice of attack traces can have the same effect (i.e., changes in the required number of attack traces). However, there are models like #5 that, for some increases (from 150,000 to 200,000 and from 250,000 to 400,000), cannot even converge. While the training curve of these models shows normal behavior, they cannot rank the correct key better than the random guessing (thus, accuracy cannot serve as an indication of the SCA performance).

In Table 4, MLP #1, #2, #3, and #4, are in the under-parameterized regime. Looking at the required number of attack traces for different profiling set sizes, we cannot see an absolute decrease in this metric. Since the number of trainable parameters in these models in comparison to even 50,000 profiling traces is small, they have learned the leakage model's distribution with a smaller number of profiling traces and gone to the saturation part of the learning curve where adding more traces to the profiling set does not improve the performance. Models like #19, #20, and #21 in Table 4 are working in an over-parametrized regime because they reach $\approx 100\%$ accuracy for all profiling set sizes. In many cases, these neural network models need more than 5,000 attack traces to place the correct key to the first rank. However, by observing the GE evolution, we see that these models could reduce the key entropy, and by adding more attack traces, they can rank the correct key in the first place. This observation shows that models working in an over-parameterized regime cannot perform as well as models working in the under-parameterized regime for the AES_HD dataset. Consequently, there is overfitting that damages the model's generalization from a side-channel perspective. For the models working in a critical regime, the deep double descent effect and assumption error combine. Thus, we can observe

(middle of Table 4) that for many cases, the models cannot rank the correct key in the first place with less than 5,000 attack traces.

4.3 Influence of Regularization Techniques

To check the influence of the increasing number of profiling traces in the presence of regularization on the required number of attack traces, we added dropout regularization to neural network models. More precisely, we added a dropout layer with a rate of 0.5 after every dense layer in MLP and CNN models that were selected in Sects. 4.1 and 4.2. The required number of attack traces in this experiment shows that using this technique can improve the performance of the neural models on average, especially for models with medium to large capacity, but still cannot negate the assumption error. In Table 5, one can see the influence of the increasing number of profiling traces on the MLP performance and the HD leakage model for the AES_HD dataset. The same behavior was captured for both CNN and MLP models for the ASCAD and AES_HD datasets.

Table 4. *MLP* trained for the *HD* leakage model.

#	*Params*	NT_1	NT_2	NT_3	NT_4	NT_5	NT_6	NT_7	NT_8	NT_9
1	12829	3294	2610	2549	2239	2041	2317	2220	2358	2011
2	26049	2795	2144	2128	2424	1904	2200	2526	2024	1938
3	26049	3268	2753	2060	2204	2470	2679	2816	2732	2608
4	27309	4700	3353	5000	3454	4384	3543	4014	3885	3965
5	60249	3681	3551	2366	5000	3224	5000	5000	2253	5000
6	136109	3445	3798	3321	3778	3391	3757	4478	3122	3118
7	196709	4918	3980	5000	4030	4293	4223	3495	2800	4222
9	292209	4527	5000	5000	5000	5000	5000	5000	5000	5000
10	378009	4867	5000	5000	4285	5000	4213	5000	4367	4288
11	378009	3706	5000	5000	5000	5000	5000	5000	5000	5000
12	412809	3735	3127	3459	2910	2854	2606	3066	2829	3297
13	468309	3880	4542	5000	5000	5000	3418	4001	3664	4732
14	824809	3456	4864	5000	5000	5000	5000	5000	5000	5000
15	824809	3533	3617	3978	5000	4235	4394	3674	4933	5000
16	880509	4339	5000	5000	5000	5000	5000	5000	5000	5000
17	985209	3771	4076	4502	4156	4601	3204	2614	3053	3219
18	1131009	2838	4824	5000	5000	5000	5000	5000	5000	5000
19	1381509	3401	5000	5000	5000	5000	5000	5000	4380	5000
20	1466409	2746	5000	5000	5000	5000	5000	4604	5000	5000
21	2383509	4134	3779	4496	5000	5000	5000	4137	5000	5000

Using dropout regularization influences the working regime of the models significantly. For both MLP and CNN models, almost all the re-trained models worked in the under-parameterized regime, even for the large models, like #9 in Table 5. This model is the counterpart of model #17 in Table 4, and its performance increased considerably. However, dropout regularization does not always improve the model's performance. For example, model #10 in Table 5 cannot converge at all with dropout. Its counterpart is #20 in Table 4 that was able to rank the actual key under 10 for many profiling sizes.

Table 5. *MLP* trained for the *HD* leakage model and the dropout regularization.

#	*Params*	NT_1	NT_2	NT_3	NT_4	NT_5	NT_6	NT_7	NT_8	NT_9
1	12829	5000	5000	4604	3496	3880	3578	3653	4133	3659
2	40599	4633	3283	3491	2758	3046	2854	2573	3430	2978
3	73209	3161	2574	2873	2600	2426	2548	2812	2411	2615
4	126009	3108	2677	2494	2254	2307	2765	2550	2608	2207
5	252009	5000	5000	5000	5000	5000	5000	5000	5000	5000
6	468309	2212	1740	1970	1942	1978	1784	2269	2291	2065
7	504009	3414	3258	2278	2733	2686	2496	2627	2753	2711
8	648909	1533	1462	1393	1402	1285	1455	1487	1405	1369
9	985209	957	920	902	1249	955	1084	1170	1232	1462
10	1466409	5000	5000	5000	5000	5000	5000	5000	5000	5000

4.4 Reducing Assumption Error by Changing Hyperparameters

Looking into Table 6, one can see the effect of changing a simple hyperparameter on the performance of a neural network model for a specific profiling size. For the sake of brevity, we provide only a few examples (one example from each combination of neural network topology and leakage models for the ASCAD dataset). Model #1 in Table 6, is the counterpart of model #4 in Table 2a. We trained this model for different numbers of epochs (shown in the "EPC" column). While the model that has been trained with 100,000 traces for 200 epochs cannot converge at all, if we train it for 100 epochs, it will be able to find the correct key with 647 attack traces. If we train it for 400 epochs, it will recover the key with 1,196 traces. Model #2 is the counterpart of model #13 in Table 2b. We changed this model's learning rate (shown in the "LR" column). As one can see in Table 6, this change improved its performance. Finally, models #3 and #4 are the counterpart of models #9 in Table 3a and model#2 in Table 3b. Changing the activation function (shown in the "Act. Func." column) in the case of these two models increased the performance considerably. In many other cases, changing hyperparameters led to the improvement of the models that could not converge to $GE = 1$ for a specific profiling set size.

Table 6. Results with small changes in hyperparameters.

#	*Params*	NT_1	NT_2	NT_3	NT_4	EPC	LR	Act. func
1	31149	3615	5000	622	5000	200	0.005	Adam
1	31149	3688	1196	5000	2434	400	0.005	Adam
1	31149	3856	647	5000	5000	100	0.005	Adam
2	663056	4821	3406	1343	713	200	0.0005	Adam
2	663056	4427	2053	886	496	200	0.0001	Adam
3	31113	3196	1300	5000	5000	200	0.005	Adam
3	31113	2415	5000	5000	621	200	0.0001	Adam
3	31113	3237	887	603	838	200	0.005	RMSprop
4	6350	1007	5000	5000	157	200	0.001	RMSprop
4	6350	675	133	2096	116	400	0.001	RMSprop
4	6350	1114	111	47	31	200	0.001	Adam

4.5 General Observations

- Increasing the profiling set size is not a guaranteed solution to increase the generalization power of a neural network model in deep learning-based SCA. The effects of the working regime on the performance of a model in the side-channel domain and the assumption error imposed by the hyperparameters combination cause the overall irregular behavior of the models regarding the increasing profiling set size.
- The under-parameterized models perform better than the critical and over-parameterized models in the SCA domain. While the theory indicates that over-parameterized models can reach better performance compared to under-parameterized models in a number of settings [2], this was not the case for SCA so far, especially for a noisy dataset like AES_HD.
- Compared to the neural network models working in the under-parameterized regime, the models working in the over-parameterized regime have a large capacity, and their number of trainable parameters is much larger than the number of traces in the profiling set. On average, such models need more traces to rank the correct key in the first place compared to the neural network models working in the under-parameterized regime. Still, it is possible to converge to $GE = 1$ with a small number of attack traces using large models and regularization.
- In many deep learning classification applications, we are interested in the final decision of the model, i.e., the class that the model assigns to measurement and not the probability that the model calculates to assign that measurement to a specific class. Contrary, in SCA, we are interested in the probabilities that a model assigns to each class for each measurement. Thus, small deviations in the estimated distribution function based on the profiling set will lead to significant changes in the model's performance from an SCA perspective. Since over-parameterized models have a large capacity and many parameters

shape the underlying leakage model distribution, they have more potential to contain assumption errors.
– A model working in a critical regime is very fragile, and in some cases, even a small change can decrease its performance noticeably. Thus, a cautionary solution could be to avoid using such models.
– Overfitting is a complex phenomenon, and we cannot trace its causes to only one source. Still, our results indicate that hyperparameter combinations play a more significant role than the profiling set size.

5 Conclusions and Future Work

This work investigates overfitting in deep learning-based SCA. We challenge the common assumption that more profiling traces is better, and we show that, while this may be the case, it cannot be taken for granted. Indeed, our experiments showed a number of settings where adding more profiling traces makes the attack less powerful or even unsuccessful. A simple yet powerful option to fight against overfitting is to use regularization or tweak the neural network architecture. Unfortunately, using such techniques does not also provide a guarantee to avoid overfitting. Thus, it is necessary to carefully design the model and assess its performance with different settings to understand in what regime the model works and then use the most appropriate one (which seems to be the under-parameterized regime). Finally, our research provides the setup to be more precise when discussing the failure of deep learning models in SCA. Instead of "simply" saying there is overfitting, one should strive to give insights what are the causes of that behavior. In future work, it would be interesting to provide a systematic approach on how to tweak hyperparameter changes to improve the attack performance. Besides, we again saw the beneficial effects of regularization (see, e.g., [14]), so it is somewhat surprising that there are no systematic evaluations on the relevance of regularization techniques in SCA.

Acknowledgment. This work received funding in the framework of the NWA Cybersecurity Call with project name PROACT with project number NWA.1215.18.014, which is (partly) financed by the Netherlands Organisation for Scientific Research (NWO). Additionally, this work was supported in part by the Netherlands Organization for Scientific Research NWO project DISTANT (CS.019).

References

1. Agence nationale de la sécurité des systèmes d'information (ANSSI): ASCAD. Github repository (2018). https://github.com/ANSSI-FR/ASCAD
2. Bartlett, P.L., Montanari, A., Rakhlin, A.: Deep learning: a statistical viewpoint. CoRR abs/2103.09177 (2021). https://arxiv.org/abs/2103.09177
3. Belkin, M., Hsu, D., Ma, S., Mandal, S.: Reconciling modern machine-learning practice and the classical bias-variance trade-off. Proc. Natl. Acad. Sci. **116**(32), 15849–15854 (2019)

4. Bhasin, S., Chattopadhyay, A., Heuser, A., Jap, D., Picek, S., Shrivastwa, R.R.: Mind the portability: a warriors guide through realistic profiled side-channel analysis. In: 27th Annual Network and Distributed System Security Symposium, NDSS 2020, San Diego, California, USA, 23–26 February 2020. The Internet Society (2020)
5. 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
6. Bronchain, O., Hendrickx, J.M., Massart, C., Olshevsky, A., Standaert, F.-X.: Leakage certification revisited: bounding model errors in side-channel security evaluations. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019. LNCS, vol. 11692, pp. 713–737. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-26948-7_25
7. 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
8. Chari, S., Jutla, C.S., Rao, J.R., Rohatgi, P.: Towards sound approaches to counteract power-analysis attacks. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 398–412. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48405-1_26
9. 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
10. Choudary, O., Kuhn, M.G.: Efficient template attacks. In: Francillon, A., Rohatgi, P. (eds.) CARDIS 2013. LNCS, vol. 8419, pp. 253–270. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08302-5_17
11. Durvaux, F., Standaert, F.-X., Veyrat-Charvillon, N.: How to certify the leakage of a chip? In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 459–476. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-55220-5_26
12. 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
13. Huang, Y., et al.: Gpipe: efficient training of giant neural networks using pipeline parallelism. CoRR abs/1811.06965 (2018). https://arxiv.org/abs/1811.06965
14. 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. Cryptographic Hardware Embed. Syst. 148–179 (2019)
15. 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
16. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Bartlett, P.L., Pereira, F.C.N., Burges, C.J.C., Bottou, L., Weinberger, K.Q. (eds.) Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held 3–6 December 2012, Lake Tahoe, Nevada, United States, pp. 1106–1114 (2012)
17. 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
18. Masure, L., Dumas, C., Prouff, E.: A comprehensive study of deep learning for side-channel analysis. IACR Trans. Cryptographic Hardware Embed. Syst. **2020**(1), 348–375 (2020). https://doi.org/10.13154/tches.v2020.i1.348-375
19. Messerges, T.S., Dabbish, E.A., Sloan, R.H.: Power analysis attacks of modular exponentiation in smartcards. In: Koç, Ç.K., Paar, C. (eds.) CHES 1999. LNCS, vol. 1717, pp. 144–157. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48059-5_14
20. Nakkiran, P., Kaplun, G., Bansal, Y., Yang, T., Barak, B., Sutskever, I.: Deep double descent: where bigger models and more data hurt. In: 8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, 26–30 April 2020. OpenReview.net (2020). https://openreview.net/forum?id=B1g5sA4twr
21. Perin, G., Chmielewski, L., Picek, S.: Strength in numbers: improving generalization with ensembles in machine learning-based profiled side-channel analysis. IACR Trans. Cryptographic Hardware Embed. Syst. **2020**(4), 337–364 (2020). https://doi.org/10.13154/tches.v2020.i4.337-364
22. Perin, G., Picek, S.: On the influence of optimizers in deep learning-based side-channel analysis. In: Dunkelman, O., Jacobson, Jr., M.J., O'Flynn, C. (eds.) SAC 2020. LNCS, vol. 12804, pp. 615–636. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-81652-0_24
23. Picek, S., Heuser, A., Jovic, A., Bhasin, S., Regazzoni, F.: The curse of class imbalance and conflicting metrics with machine learning for side-channel evaluations. IACR Trans. Cryptographic Hardware Embed. Syst. **2019**(1), 209–237 (2019). https://doi.org/10.13154/tches.v2019.i1.209-237
24. Picek, S., Heuser, A., Perin, G., Guilley, S.: Profiling side-channel analysis in the efficient attacker framework. Cryptology ePrint Archive, Report 2019/168 (2019). https://ia.cr/2019/168
25. Prouff, E., Strullu, R., Benadjila, R., Cagli, E., Dumas, C.: Study of deep learning techniques for side-channel analysis and introduction to ASCAD database. IACR Cryptol. ePrint Arch. p. 53 (2018). https://eprint.iacr.org/2018/053
26. Rijsdijk, J., Wu, L., Perin, G., Picek, S.: Reinforcement learning for hyperparameter tuning in deep learning-based side-channel analysis. IACR Trans. Cryptographic Hardware Embed. Syst. **2021**(3), 677–707 (2021). https://doi.org/10.46586/tches.v2021.i3.677-707
27. Szegedy, C., et al.: Going deeper with convolutions. In: IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, 7–12 June 2015, pp. 1–9. IEEE Computer Society (2015). https://doi.org/10.1109/CVPR.2015.7298594
28. Wu, L., Perin, G., Picek, S.: I choose you: automated hyperparameter tuning for deep learning-based side-channel analysis. IACR Cryptol. ePrint Arch. p. 1293 (2020). https://eprint.iacr.org/2020/1293
29. Ying, X.: An overview of overfitting and its solutions. In: Journal of Physics: Conference Series, vol. 1168, p. 022022 (2019). https://doi.org/10.1088/1742-6596/1168/2/022022
30. Zaid, G., Bossuet, L., Habrard, A., Venelli, A.: Methodology for efficient CNN architectures in profiling attacks. IACR Trans. Cryptographic Hardware Embed. Syst. **2020**(1), 1–36 (2020). https://doi.org/10.13154/tches.v2020.i1.1-36
31. Zhang, C., Bengio, S., Hardt, M., Recht, B., Vinyals, O.: Understanding deep learning (still) requires rethinking generalization. Commun. ACM **64**(3), 107–115 (2021). https://doi.org/10.1145/3446776