Strength in Numbers: Improving Generalization with Ensembles in Profiled Side-channel Analysis

Guilherme Perin[1,2], Lukasz Chmielewski[1] and Stjepan Picek[2]

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

Abstract. The adoption of deep neural networks for profiled side-channel attacks provides powerful options for leakage detection and key retrieval of secure products. When training a neural network for side-channel analysis, it is expected that the trained model can implement an approximation function that can detect leaking side-channel samples and, at the same time, be insensible to noisy (or non-leaking) samples. This outlines a generalization situation where the model can identify the main representations learned from the training set in a separate test set.

In this paper, we first discuss how output class probabilities represent a strong metric when conducting the side-channel analysis. Further, we observe that these output probabilities are sensitive to small changes, like the selection of specific test traces or weight initialization for a neural network. Next, we discuss the hyper-parameter tuning, where one commonly uses only a single out of dozens of trained models, where each of those models will result in different output probabilities. We show how ensembles of machine learning models based on averaged class probabilities can improve generalization. Our results emphasize that ensembles increase the performance of a profiled side-channel attack and reduce the variance of results stemming from different groups of hyper-parameters, regardless of the selected dataset or leakage model.

Keywords: Side-channel Analysis · Neural Networks · Model Generalization · Ensemble Learning

1 Introduction

The implementation of secure products considers the threat imposed by side-channel attacks (SCAs). The growing markets of embedded computing, and especially Internet-of-Things, require large amounts of confidential data to be processed on electronic devices. Cryptographic algorithms are usually implemented as part of those systems and, if not properly protected, are vulnerable to side-channel analysis. Depending on the level of access and control of the target device, side-channel analysis can be categorized as profiled (e.g., template attacks [CRR02], linear regression [SLP05], machine learning [LMBM13, MHM13]), or non-profiled attacks (e.g., DPA [KJJ99], CPA [BCO04], MIA [GBTP08], clustering [BGL09]). Profiled side-channel attacks consider a scenario where the adversary has full or sufficient control over a device that is identical to the target device. The adversary then learns statistics from the device under control and tries to match them on other target devices.

In the last few years, deep learning, in its supervised setting, has been intensively considered as an important method for profiled side-channel analysis [MPP16, CDP17, CCC+19]. With the adoption of deep neural networks, and especially the availability of open-source frameworks and datasets, their applications to side-channel analysis have

improved the community understanding of what are the main capabilities of this type of non-invasive attacks. The works of Cagli et al. [CDP17] and Kim et al. [KPH+19] proposed the application of techniques to improve generalization of deep neural networks for profiled SCA. While [CDP17] considered data augmentation to deal with trace misalignment, [KPH+19] used noise addition to the input traces as a regularization artifact. As a consequence, test generalization is improved, which can be confirmed by inspecting the guessing entropy of the attacked traces. While providing good indications on how to deal with the generalization problem in SCA, these works focus on fixed neural network architectures where the relation between hyper-parameters and generalization is left for future research. The work from Prouff et al. [PSB+18] proposes experiments in order to verify what are the most promising hyper-parameters for specific trace sets. While this gives valuable information for side-channel analysis, it is still challenging to use the same conclusions for different trace sets, as they are measured from different devices and with different equipment.

The selection of hyper-parameters is a crucial factor in deep neural network generalization, as can be deduced by observing the effects of underfitting or overfitting. One way to solve this issue is to implement a strong hyper-parameter tuning, but this can be very expensive in the profiled side-channel analysis context. A realistic scenario would assume that up to hundreds of different hyper-parameters combinations could be tested in a reasonable time. This is particularly difficult in the case when attacking protected AES implementations where hundreds of thousands of traces need to be used in the training set.

In this paper, we demonstrate that the reference metric to select the best model(s) from the hyper-parameter search is the validation key rank or guessing entropy. To provide more details about machine model metrics and how they are related to the side-channel analysis metrics, we take a closer look at the output probabilities obtained either in the validation or in the test phases. More precisely, the output layer of a neural network contains a number of neurons (or units) that is equivalent to the number of classes (or the range of labels) defined for datasets. The number of classes is directly derived from the selected leakage model. For every tested trace, this output layer provides class probabilities if the activation function is *softmax*. By ranking these output class probabilities by their size for each trace and each key candidate, we observe that these ranked class probabilities can be seen as a valid distinguisher or metric for side-channel analysis. We emphasize that the generalization to the validation set can be significantly different from the validation to the test set. While both sets represent a sample of data we do not use for training, the validation set is often used to evaluate the performance of the training set, and thus, it indirectly affects the model.

While crucial to the success of an attack, output probabilities are sensitive to even small changes in the setup, e.g., weight initialization, or choice of hyper-parameters. As such, we see that the question of how to conduct a successful side-channel attack connects with a question on how to make the output class probabilities robust. To that end, we consider the hyper-parameters tuning where instead of simply selecting the best model out of H searches, we take an ensemble of a group of best models. Our results emphasize that this procedure ensures a significantly higher success rate when compared to the performance of a single best model.

1.1 Related Works

As this work tackles the problem of generalization of neural network models and metrics in side-channel analysis, we provide an overview of works published in the last few years that are related to this problem. We identify the following main trends of research within the field:

- **Deep learning for profiled side-channel attacks**. Bypassing misalignment and masking countermeasures on AES [MPP16, CDP17] and public-key [CCC+19, WPB19] implementations with the application of deep convolutional neural networks.

Different techniques for regularization, like data augmentation through random shifts [CDP17, PSB+18, Mag19] or noise addition [KPH+19] have been tested. The authors of [ZS19] investigate the efficiency of deep learning with respect to misalignment in side-channel traces. In [YLMZ18], it is shown the results of convolutional neural networks on time-frequency representations of traces. The work of [HGG18] proposes the usage of cryptographic information (plaintext and ciphertext) as additional inputs to the first dense layer in a convolutional neural network in order to improve the key enumeration. Some types of neural networks work well against SCA countermeasures as convolutional neural networks have inherent resilience against misalignment in traces due to their spatial invariance [CDP17]. Multilayer perceptrons, on the other hand, mimic the effect of higher-order attacks by combining the features in the fully-connected layers, which makes such neural networks a natural choice when dealing with masking countermeasures [CDP17, KPH+19].

- **Trained neural networks as a side-channel distinguisher**. The work proposed by [Tim19] considers a DPA-like deep learning attack where models are trained for each key byte candidate in an AES implementation. Training and validation metrics are then considered for reference to distinguish between correct and incorrect key byte candidates.
- **Model interpretability** and *leakage assessment*. In this group, several works propose different techniques to verify what a neural network learns from side-channel traces. Different works proposed the evaluation of input activation gradient [MDP19b], occlusion techniques [HGG19], and layer-wise back propagation [PEC19] as a metric to assess what input features (or points of interest) the neural network selects as the most important for its decisions. Additional steps into the interpretability of neural networks are done with the introduction of the guessing entropy bias-variance decomposition [vdVP19]. By doing so, it is possible to verify what are the points of interest or features that a trained model considers in order to make its classification decision. Next, the work of [MDP19a] investigates the performance of neural networks through information theory. Finally, while not belonging to the interpretability direction, we also mention the work that considers the explainability of neural networks for SCA, where the authors consider the layer-wise activation function comparison [vdVPB19].
- **Relationship of deep learning and side-channel metrics**. The work proposed in [PHJ+19] concludes that there is an inconsistency between accuracy, recall, and precision when compared to common side-channel metrics like success rate and guessing entropy. To the best of our knowledge, this work is the only one that evaluates the meaning of deep learning metrics for side-channel analysis.
- **Deep learning for leakage assessment**. The authors of [WMM19] provide a methodology for leakage assessment with deep neural networks to leverage the benefits of deep learning in terms of location, alignment, and statistical order of the leakage.

1.2 Contributions and Motivation

By examining the above list of publications, we conclude there is still a lack of technical explanations to justify why a trained model can provide successful key recovery even when deep learning accuracy is close or below random guessing. The work of [PHJ+19] provides valuable information through practical experiments to show the existence of such inconsistencies with several deep learning metrics, but does not give a detailed analysis of output class probabilities. In this paper, we study how class probabilities influence attack performance. What is more, we propose to use ensembles to improve the output class probabilities predictions and, consequently, make the attacks more powerful. More precisely, our main contributions are:

1. **A didactic analysis of output class probabilities and their relation to successful key recovery:** Motivated by the idea of exploring metrics discrepancy, in Section 3, we explore the information contained in the output probabilities from the neural network's output layer as valid information for verifying the performance of a trained neural network for SCA. We show how output class probabilities change with respect to the dataset selection, but also to the machine learning model.
2. **How to improve generalization through with ensembles:** We demonstrate how a significantly better attack performance can be obtained by considering an ensemble of several trained neural network models instead of simply selecting the best model from, e.g., hyper-parameters search (Section 4). Interestingly, we show that ensembles work well not only when including only the best obtained models, but also when a large number of non-optimal models is used.

This paper is structured as follows. Section 2 discusses the datasets we investigate, performance metrics in machine learning and SCA, and ensembles. Section 3 presents an experimental analysis of output class probabilities and their relevance. In Section 4, we discuss how ensembles of neural networks can significantly improve the attack performance. In Section 5, we discuss possible future research directions, and we conclude the paper. We provide information about notation, machine learning techniques, and key rankings for the ASCAD dataset example in Appendices A, B, and C, respectively.

2 Datasets, Performance Metrics, and Generalization in SCA

2.1 Datasets

This work considers several datasets for the practical experiments, where all of them are available online. All datasets consist of side-channel analysis measurements where power analysis is used as the source of leakage. In this section, we provide details about the adopted datasets. All considered datasets consist of power side-channel measurements. However, the analysis and contributions in this paper can be extended to other sources of side-channel analysis, e.g., electromagnetic analysis. The details about investigated datasets are provided in Table 1.

Table 1: Publicly available datasets.

Dataset	Training	Validation	Test	Features	Countermeasure
Piñata SW AES	6 000	1 000	1 000	400	No
DPAv4	34 000	1 000	1 000	2 000	RSM
ASCAD	200 000	500	500	1 400	Masking
CHES CTF 2018	43 000	1 000	1 000	2 000	Masking

2.1.1 Piñata SW AES

This dataset refers to a software AES-128 implementation (32-bit STM microcontroller) where no countermeasures are implemented against side-channel analysis. The traces contain 400 features (or samples) each, and this interval corresponds to the processing of the S-box operation during the first AES encryption round. Training, validation, and test sets correspond to side-channel traces with the **fixed encryption key** (the key is the same for training/validation/test sets). The details about the target are given in [Ris20].

2.1.2 DPAv4

DPAv4 database contains trace sets collected from an AES-256 RSM (rotate shift masking) implementation [TEL14]. The training set consists of 34 000 traces with a **fixed key**. For our experiments, test and validation sets contain 1 000 traces each (those sets have the same key as for the training set). For convenience, we attack only the first key byte of an AES-256 implementation. We trimmed the traces to contain 2 000 features corresponding to the processing of the S-box operation during the first AES encryption round. [1]

2.1.3 ASCAD

This dataset contains measurements from an 8-bit micro-controller software implementation of AES-128, where Boolean masking is implemented as a countermeasure. It has 200 000 traces with random keys and an additional trace set with 1 000 traces with a fixed key [PSB+18]. Note, the set with 1 000 traces, and with the fixed key is split into two sets of 500 traces to be used as validation, and test sets[2]. The randomly generated masks for every AES encryption are also provided with the database metadata. From these mask values, we could confirm that in the first encryption round, the intermediates associated with key bytes 0 and 1 are unprotected because the mask values that would generate a masked S-box output byte value are set to zero for all traces. For the rest of the 14 key bytes (from index 2 until 15), the mask bytes are random.

2.1.4 CHES CTF 2018

This database refers to the CHES Capture-the-flag (CTF) AES-128 trace set, released in 2018 for the Conference on Cryptographic Hardware and Embedded Systems (CHES). The traces refer to masked AES-128 encryption running on a 32-bit STM microcontroller. In our experiments, we consider 43 000 traces for the training set, which contains a **fixed key**. The validation and test set consist of 1 000 traces each. The key for the training and validation set is different from the key configured for the test set. Each trace consists of 2 200 features. [3]

2.2 Profiled Side-channel Analysis and Deep Learning

As a profiled side-channel attack, the application of deep learning requires a training set of size N for the learning or profiling phase. Considering x_i as a vector representing a side-channel trace, where $x_{i,f}$ is a feature (point of interest) in x_i, each trace is then labeled according to a function $l = f(k, pk)$ that represents the side-channel leakage model. Ideally, the training set should be composed of side-channel traces where each trace is measured with random input data (ciphertext or plaintext) pk and random key k. Thus, the leakage model defines the number of classes in the training set. The assumption of a profiled attack is that the adversary has a device that is identical to the target one, and this adversary has (in the best-case scenario) full control of the device.

In addition to the training set, the adversary also collects a validation set of size V from the device under control.[4] After training the neural network, the validation set is evaluated in order to check the generalization capacity of this trained model. If validation metrics indicate a sufficient level of generalization (we will see later that this sufficient level is not easy to estimate for SCA), the adversary has obtained a potentially good enough model to apply to test traces of size Q collected (from an other identical device) with an unknown secret key k^*. We assume that test traces are drawn from the same underlying

[1] This dataset is available at http://www.dpacontest.org/v4/.
[2] This dataset is available at https://github.com/ANSSI-FR/ASCAD.
[3] This dataset is available at https://chesctf.riscure.com/2018/news.
[4] The keys k_i and inputs pk_i are also known for the validation set.

distributions defining training and validation sets (while this may not always be the case due to the portability setting [BCH+19]).

2.3 Performance Metrics and Generalization in SCA

The conventional deep learning metrics taken into account in this work are accuracy, and loss (or error). *Accuracy* indicates the ratio between correctly predicted data and the total number of predictions. *Loss* indicates the overall error for the evaluated set. The machine learning metrics obtained during the training step may indicate the presence of different phases that can occur when the parameters (weights and biases) are learned from the training dataset. Here, we assume the presence of three main phases (that does not occur necessarily in this order, even if usually it happens in this order):

1. Underfitting: it occurs when the approximation function defined by the neural network model is unable to fit both the training and validation sets. The error is significantly high for the training set.
2. Generalization: during this phase, the neural network achieves an approximation function that can fit the training and validation sets with an acceptable error. The metric results are sufficient to solve the classification problem under question. This is the ideal scenario in a machine learning setting. In the side-channel analysis, a *good enough* generalization is what we aim to obtain. The concept of *good enough* will be clarified when we discuss output class probabilities in the next section. Still, note that the generalization on the validation set does not necessarily mean generalization for the test set. As already said, the validation set is used to update the hyper-parameter values, and consequently, it affects the training phase.
3. Overfitting: this phase happens when the neural network can fit the training set with very high accuracy and very low error, however it cannot fit the validation set. The metrics obtained when the validation set is evaluated are insufficient to solve the machine learning (or classification) problem. Typically, it is relatively easy to overfit the model for a training set in the side-channel analysis. The number of traces is usually limited in size (up to a few million), and an over-parameterized neural network has conditions to overfit this training set. Usually, what happens is a large *generalization gap*, i.e., a difference between a model's performance on training data and its performance on unseen data drawn from the same underlying distribution.

Ideally, we should always train a neural network until it achieves the maximum quality in terms of generalization with respect to the validation set. If this happens, we should be able to assess whether the model is in the generalization phase or not. This seems to be an easy task. There are quite some difficulties in the interpretation of metrics to identify what phase the model belongs to while the training evolves. An interesting observation would be the detection of the boundaries between two of the phases above. These boundaries may be detected with the observation of conventional metrics (loss, accuracy, recall, or precision) using validation data.

Common metrics in SCA are success rate and guessing entropy [SMY09]. These metrics are not aimed only at predicting correct labels as is the case with machine learning metrics, but also to reveal the secret key. In particular, let us assume that given Q amount of traces in the attacking phase, an attack outputs a key guessing vector $\mathbf{g} = [g_1, g_2, \ldots, g_{|\mathcal{K}|}]$ in decreasing order of probability with $|\mathcal{K}|$ being the size of the keyspace. The success rate is defined as the average empirical probability that g_1 is equal to the secret key k^*. The guessing entropy is the average position of k^* in $\mathbf{g}$.

Moreover, different trace sets will require different hyper-parameters for the achievement of a good generalization. However, as practically demonstrated in [PHJ+19], conventional deep learning metrics are not very consistent in the context of side-channel analysis. To verify if a model is actually generalizing to the validation or even to a separate test set, one requires a different understanding of training and validation metrics. Usually, side-channel

traces collected from modern cryptographic implementations provide a limited leakage due to the state-of-the-art countermeasures. In this paper, we attack AES implementations where the successful results cannot be represented by deep learning metrics. For example, the trained model may present test accuracy close to random guessing, and still, the target key can be recovered from the side-channel test set. The trained model indicates a level of generalization that is limited but sufficient for the considered case. We define this limited level of generalization as *good enough*. To improve generalization, one can use some of the techniques as indicted in Section 1.1, but in this paper, we propose to use the ensembles as a method of choice to improve SCA performance.

2.4 Ensembles in Machine Learning

Ensemble learning is a technique that combines the predictions from several models in order to reduce the generalization error. The expected result is that the prediction obtained from model ensembles is superior in comparison to a single model. This is motivated by the statistical intuition that averaging measurements (or predictions) can lead to a more stable and reliable estimate because they reduce the influence of random fluctuations in a single measurement. The main methods for ensemble learning are:

- **Boosting:** it is also considered as a sequential ensemble learning method. This type of ensembles assumes the construction of weak learners (machine learning models that do not have enough capacity to model the relationships in data) in order to achieve the same performance as a strong learner. Weak learners are faster to train because they consider a small portion of the training data. In the loop process, every iteration is a weak model training, and the training data is drawn from the original distribution. Thus, the error is evaluated and, before a new iteration, the distribution that defines the new training data is adjusted based on certain criteria. In the end, there will be a model with optimal metrics based on the selection of training data. Boosting is used in profiled SCA in [PHJ+17].
- **Bagging:** this methodology considers an averaging or linear combination (or weighted sum) of predictions from all single models. The main goal of bagging ensembles is to reduce the variance of different models. Bagging is not uncommon in SCA [LMBM13, MPP16], as for instance, random forest uses bagging mechanism.
- **Stacking:** in this particular case, several single models are trained, and the predictions for each model from a validation set are used as new training data for a stacked model. The stacked model could be a neural network, for instance.

We note that while there are works that use ensembles in SCA, there, ensembles are a natural consequence of a selected machine learning method. For instance, random forest is designed as a bagging technique, and thus, it cannot be used without the bagging mechanism. In this paper, we are interested in ensembles of techniques that also work independently, like neural networks. This, to the best of our knowledge, was not investigated in SCA before. Finally, in this paper, we use the bagging ensemble technique.

3 Output Class Probabilities as an Indication of "Good Enough" Generalization

There are many domains where the application of a trained neural network requires a very high test accuracy in order to solve the problem. As a consequence, the generalization phase will start only after a sufficiently high accuracy is achieved for the training set. In the side-channel analysis, the generalization phase is directly related to the key recovery, and it may start very soon after the training starts because a low accuracy can already represent the turning point from underfitting to generalization. Unfortunately, as mentioned in the previous section, accuracy or recall may not be reliable metrics for the identification of this

minimum amount of epochs in order to have a successful key recovery. On the other hand, SCA metrics like key ranking, success rate, or guessing entropy are the metrics of use for neural networks in side-channel analysis, but they are also difficult to evaluate during the training phase. For instance, Robissout et al. suggest using the Euclidean distance between the training and validation success rate as a stopping metric. This metric is powerful, but it requires that success rate calculation is done after every training epoch [RZC+20]. Unfortunately, in practical scenarios, it is not obvious whether this is feasible as training sets can have millions of examples where they run for thousands of epochs. Additionally, it is not clear how to estimate the success rate for the training set if that set contains random keys.

When a neural network is trained to classify side-channel traces according to a chosen leakage model (e.g., the Hamming weight of an intermediate byte in the target cryptographic algorithm), the output layer of the network can provide the probability for each possible class. For that, the *softmax* activation function is defined for the output network layer. These probabilities indicate the likelihood that a specific test trace leaks an intermediate value according to a chosen leakage model. Then, a key ranking can be calculated, and this process is similar to a differential power analysis where all possible key byte values are tested for the same trace set, and a distringuisher indicates what the most likely key is. The assumption is that if the trained model achieves a *good enough* generalization, and if the number of test traces is sufficiently large, the correct key byte candidate provides a likelihood higher than the likelihood obtained from the incorrect key hypotheses.

Can a low accuracy (sometimes close to random guessing) still be associated with this *good enough* generalization phase? The problem is that the accuracy does not represent this generalization in SCA, which can only be represented by the way key ranking is computed from the output class probabilities. The accuracy is a metric that takes into account only the predicted class y' for each test trace. However, not only the output class probability associated with the predicted y' is taken into account for the key ranking calculation, but also the class probabilities that are not the highest ones, as they contain valuable information. This situation is directly related to the fact that deep learning-based SCA against protected targets is successful even if machine learning metrics indicate the opposite.

The outcome of predicting with a trained model on the test set is a two-dimensional matrix P with dimensions equal to *number of traces* $\times$ *number of classes*. Each element of this matrix is a vector of all possible class probabilities c for a specific trace i. Note that the number of classes does not need to be the same as the number of key guesses, which depends on the leakage model (e.g., for AES, the Hamming weight model has nine classes, while there are 256 key guesses). The summation probability $S(k)$ for each key byte candidate k is a valid distinguisher for the side-channel analysis, and it is given by:

$$S(k) = \sum_{i=1}^{Q} (p_{i,j}), \tag{1}$$

where Q is the number of test traces. The value $p_{i,j}$, an element of matrix P, is a function of the attacked trace i, the leakage model l, key guess k, and input pk_i.

Each row i of the matrix P contains the output class probabilities for a specific trace. For the correct key candidate k^*, Eq. (1) would sum up the highest output class probabilities from all rows only when the test accuracy is 100% (meaning that the guessed label is a correct one for every trace). In cases when accuracy is very low and insufficient to indicate the generalization, the output probabilities $p_{i,j}$ for the correct key k^* would not be the highest ranked ones in all rows of P. Still, if the summation for $S(k^*)$ is the largest value among all possible $S(k)$, $k \in |\mathcal{K}|$, then the predicted key is the correct one. As such, it is clear that not only the highest output class probability in a row i provides valuable classification information for side-channel analysis (which would consequently

mean that the trained model is still in a good enough generalization phase).

3.1 Output Class Probabilities from a Leaky Target: Piñata SW AES

We consider a trace set measured from an unprotected software AES-128 implementation. In this case, information leakage is significant. For this simple example, the training set contains 6 000 traces, and the validation and test sets contain 1 000 traces each. Each trace contains 400 points of interest or features. The leakage model is based on the Hamming weight of a byte in the S-box output from the first encryption round. This defines a maximum of nine classes (Hamming weights ranging from 0 to 8). As we target one round key byte at a time, it is necessary to train 16 models to recover the 16 round key bytes. To make it simple, we provide the results for a single round key byte.

As the training set is relatively small, we define a very simple multilayer perceptron (MLP) as the architecture of choice. For all the experiments in this paper, we considered *Keras* Python package for neural networks. To attack this unprotected AES, the configured model has three hidden layers containing 40 units or neurons each. The activation function for the hidden layers is *ReLU*, and the output layer, containing nine neurons (equivalent to the number of classes), has *softmax as the activation function*. The learning rate is 0.001 with *Adam* as the adaptive optimizer for the backpropagation algorithm. The loss function is computed from the *categorical cross-entropy*, which returns the cross-entropy between an approximation distribution y' (predictions) and a true distribution y (true values).

The model is trained for 200 epochs, and the achieved training, and validation accuracy are 83% and 52%, respectively (Figure 1a). For side-channel analysis, this validation accuracy already indicates a good level of generalization, and the key recovery will be successful if a reasonable amount of test traces is considered. In this case, the target key byte is recovered after the processing of approximately 20 traces. At the same time, the evaluated test accuracy is close to 48%. Notice how to model starts to overfit around epoch 40.

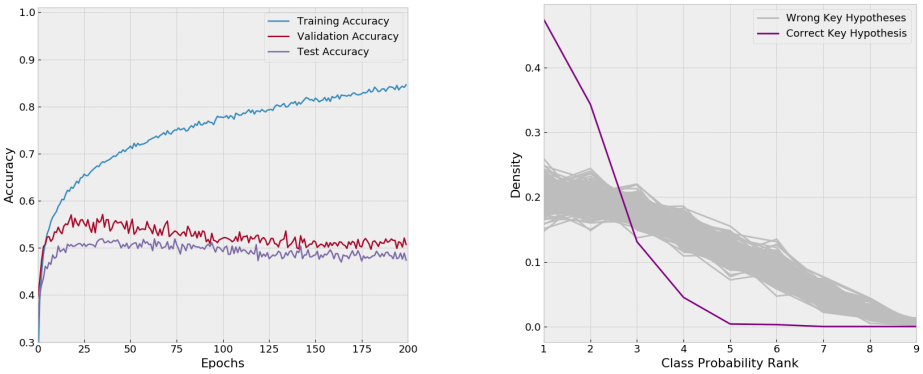
Now, let us analyze the relevant leakage that is captured in the output class probabilities for the test set. If the test accuracy would be 100%, for each test trace, the value of $p_{i,j}$ would be associated with the highest output value in the output network layer (the expected class is always equal to the predicted class). As our test accuracy is 48%, only 48% of the tested traces will contribute to the summation $S(k^*)$ with the highest class probability. The rest of the traces will contribute to the summation $S(k^*)$ with output class probabilities that are not ranked as the first (or that are not associated with the predicted class) in the output layer. Figure 1b shows the density distribution for the rank of output class probabilities for the correct (pink) and incorrect (grey) key byte candidates. By ranking the output probabilities in the last neural network layer from 1 (highest probability) to 9 (lowest probability), this figure indicates the density of output class probabilities. More precisely, it shows the number of times a certain class probability appeared for 1 000 attack traces.

For the correct key candidate (pink line), the best guess is the correct one for around 50% of the test traces. Afterward, we observe a relatively steep slope of the curve, which indicates that our trained model is sure in its predictions. What is more, we see that the model does not make errors where the correct key would be predicted as the lowest probable ones. On the other hand, for the wrong key candidates (gray region, which consists of 255 lines - one for each wrong key guess), the rank of output probabilities tends to be distributed more evenly for all wrong key candidates. Still, we see that even the wrong key candidates have a relatively high density for the top ranks. This happens due to the Hamming weight leakage model as when the classifier is wrong, it tends to predict Hamming classes 3, 4, and 5, which occur most often.

As mentioned, this attack succeeds for already around 20 traces. While the required number of traces cannot be read from Figure 1b, we can deduce the attack will easily

succeed since the pink line has much higher values for most probable class probabilities, which means that Eq. (1) has a much higher value for the correct key guess than any wrong key guess.

This analysis shows that output class probabilities contain valuable information and that they can be considered as an important metric for side-channel analysis and generalization. Still, as we considered a very simple example here, one can question do we see the same trend when attacking more difficult targets. Next, we evaluate a protected AES implementation, where, as we will see, output class probabilities behave very differently when accuracy is close to random guessing.



(a) Training, validation, and test accuracy.

(b) Rank of output class probabilities.

Figure 1: Machine learning metrics and class probability ranks for the Piñata SW AES implementation.

3.2 Output Class Probabilities from a Protected Target: ASCAD AES

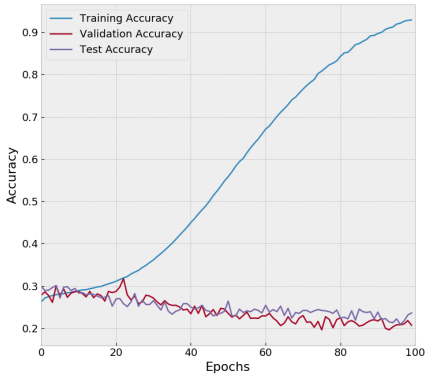
In this section, we provide an analysis of the output class probabilities on a masked AES implementation (ASCAD database). We develop our analysis for the processing of key byte with index 2 in the first encryption round (as this is the first masked byte). For training, validation, and test phases, the selected leakage model is the Hamming weight of S-box output *without taking into account the mask values and the knowledge of countermeasures*, i.e., a black-box scenario.

The implemented neural network model is more complex when compared to the previous example (Section 3.1). Now, we define a multilayer perceptron with five hidden layers, where each layer contains 200 neurons, and the activation function is *ReLU*. The rest of the hyper-parameters are the same as the ones considered for the leaky AES in the previous subsection. Note that this architecture is similar to the one given in [PSB+18], with a small difference that we use one less hidden layer. We train our architecture for 100 epochs.

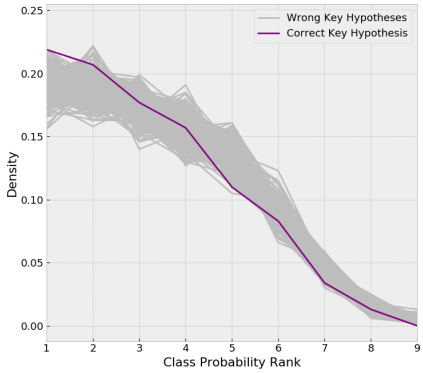
In Figure 2a, we observe that while the accuracy grows during the full training phase, the validation accuracy slowly decreases from the beginning, which indicates that our neural network is not learning how to generalize to unseen data. If we had to identify the main training phases (underfitting, generalization, and overfitting) from the accuracy metric, we would assume that the neural network model starts in an underfitting scenario (up to 10 epochs) and then it starts to fit the training set until the overfitting scenario would be achieved if more than 100 epochs are processed. Unfortunately, from the validation and test accuracy, we can assume that the generalization phase never happened. This is somewhat counter-intuitive, as the actual attack reveals that this target can be broken with around 1 000 traces (the correct key candidate k^* is ranked as the first).

Next, we consider the ranks for the output class probabilities for the correct and wrong key candidates (Figure 2b). Differing from Figure 1b, now the probabilities for the correct and wrong key candidates are similar, which means that our trained model is less certain in its predictions. Additionally, we see that for the correct key candidate, the best ranked guess has a lower density, which is expected as the classifier predicts it less often. Even for the correct key candidate, we observe that sometimes we obtain the correct guess among the least probable ones. Finally, we observe that the highest probability rank is still somewhat higher for the correct key than for any wrong key.

The interpretation of the grey region for all wrong keys is somewhat more difficult as it is not possible to distinguish among various keys. First, we observe that for the lowest probable guesses, the grey region appears above the pink line, which means that the wrong keys have predictions more spread out, i.e., for wrong keys, the classifier is less certain how to classify. Next, between the ranks 4 and 8, the grey region is above the pink line, which would indicate that some of the wrong guesses could have the final probability very close to the correct key (or even surpass it, depending on specific values). Still, this is not the case because our analysis shows that the wrong key with high probability for a certain key rank (e.g., rank 1) has sudden jump for next ranks (e.g., ranks 2 and 3), which means, that in the end, the summed probability for any wrong key is smaller than the summed probability for the correct key.



(a) Training, validation and test accuracy.



(b) Rank of output class probabilities.

Figure 2: Machine learning metrics and class probability ranks for the ASCAD database (1 000 traces).

Next, in Figure 3, we depict the class probability rankings for different number of test traces: 250, 500, 750, and 1 000 traces. It is clear that with 250 traces, the correct key guess is far from the predicted one, as there is a significant grey region above the pink line even for the highest probable ranks. As the number of traces increases, we can observe how pink line improves, first by becoming more reliable for ranks 2 and 3, and then finally, for 1 000 traces reaching one of the top values for the highest probable rank. In Appendix C, we give details about the probabilities behavior for the correct key and the best wrong guess. Interestingly, we can see that the result with 1 000 traces somewhat differs from the one given in Figure 2b (especially for the correct key guess). This indicates that while the summed output probability is the correct metric for assessing the SCA performance, it is also quite sensitive even to small changes, like a random selection of test traces or different weight initialization (which are the differences here).

While the weight initialization procedure and selection of test traces are important, we commonly assume that, on average, the performance should be similar for any subset of traces drawn from the same distribution and any reasonable weight initialization procedure. Next, we ask the question of how sensitive are output class probabilities to hyper-parameter

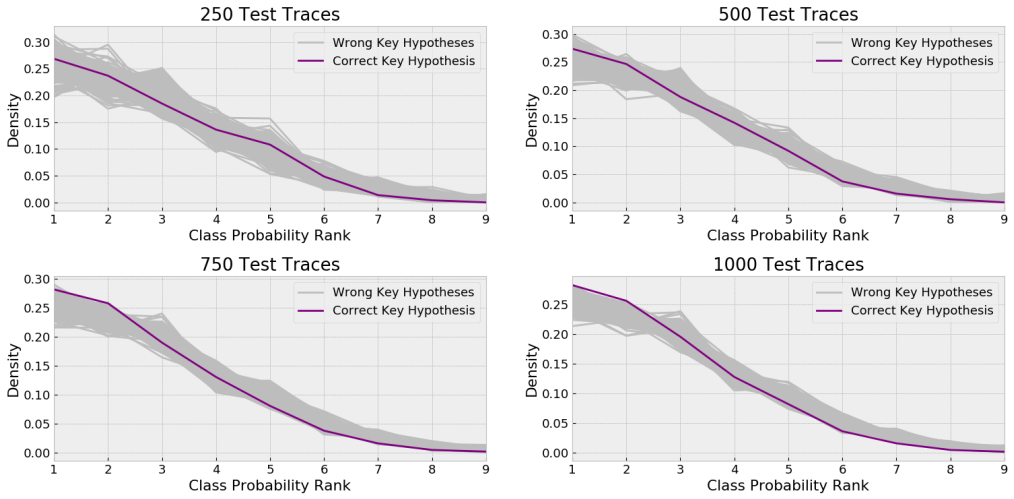


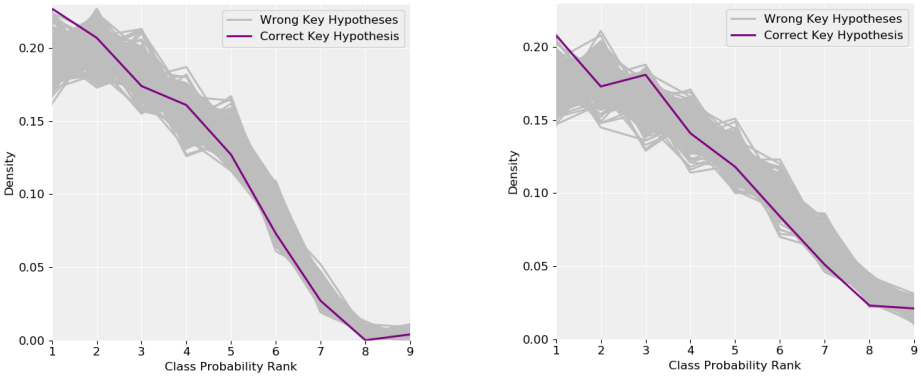
Figure 3: Rank of output class probabilities for the test set by considering correct and incorrect key candidates for different amount of test traces (ASCAD).

tuning? Note, in realistic scenarios, when one does not know what the optimal classifier to use, and due to the No Free Lunch theorem [Wol02], there is no single best classifier for all the problems.

3.3 Output Class Probabilities from a Protected Target: CHES CTF 2018

Next, we give an analysis of the CHES CTF 2018 dataset (43 000 traces for the training phase and 1 000 for validation and testing). We consider two CNN architectures that differ only in the number of dense layers (2 vs. 3 dense layers). The selected leakage model is the Hamming weight of the S-box output in the first AES encryption round. As it can be observed in Figure 4, a small change in hyper-parameters results in different output probabilities, for both correct and wrong key guesses. What is more, we can see that both attacks will be successful as the pink line is above the grey region for the highest probability ranks. This means that while output probabilities represent a strong distinguisher when summed, they are not robust to changes in the architectures (for both correct and wrong guesses). Again, note that while the grey region is above the pink line for several ranks, the grey region represents 255 wrong guesses, and if we consider the output probability for any single wrong key guess, it is lower than the one for the correct key guess.

Now, a natural progression is to try to make the output class predictions more robust, which should result in a better attack performance and more stable behavior in the presence of small changes. Since the hyper-parameter selection is a difficult task that rarely results in the best possible hyper-parameters (and consequently, machine learning models), we propose the usage of ensembles to improve the SCA performance. There, one would select several good machine learning models instead of simply taking a single better (where better denotes the best out of the tested machine learning models, but still a model that is most likely not the optimal one) model according to a validation metric. This approach is motivated by the idea of increasing the distance of the summation in Eq. (1) for the correct key candidate k^* with respect to the wrong key candidates.



(a) CNN with 1 conv. layer (32 filters, ks = 10, stride = 1) and 3 dense layers (400 neurons). **(b)** CNN with 1 conv. layer (32 filters, ks = 10, stride = 1) and 2 dense layers (400 neurons).

Figure 4: Rank of output class probabilities for two different groups of CNN hyper-parameters on the same dataset (CHES CTF 2018).

4 Improving Generalization in SCA with Ensembles

A deep neural network can be understood as a learning algorithm that needs to be tuned with to perform well. This tuning, i.e., selection of hyper-parameters, can be considered as one stage of a bi-level optimization problem, where the first objective refers to the learning of neural network parameters (weights and biases), and the second objective is the performance with regards to the selected hyper-parameters. In deep learning-based profiled SCA, we could even consider a third objective that is the selection of a leakage model. The latter defines the way the datasets are labeled, and this also needs to be selected in a proper way for an attack to be successful. Depending on the available time, several deep neural network configurations can be tested, and the output metrics are monitored to adjust the hyper-parameters. Such a hyper-parameter optimization process generates many trained models and is usually concluded by selecting a model according to the minimum (cross-validation) generalization error.

Let us assume a neural network model h and a set of different groups of hyper-parameters $\Lambda = (\lambda_0, \lambda_1, \ldots, \lambda_{z-1})$. The model h is trained with training set t_{train} and validated with validation set t_{val}, $\forall \lambda \in \Lambda$, producing $H = (h_{\lambda_0}, h_{\lambda_1}, \ldots, h_{\lambda_{z-1}})$ trained models. As suggested in [HKV19], the best model is selected according to:

$$h_{best} = \underset{\lambda \in \Lambda}{\operatorname{argmin}} \, L_{val}(h_\lambda, t_{train}, t_{val}), \tag{2}$$

where $L_{val}(h_\lambda, t_{train}, t_{val})$ return the validation loss for a model h, which is configured with a set of hyper-parameters λ, trained with t_{train}, and validated with t_{val}.

In this work, the set of H trained models is a result of random hyper-parameter selection in the predefined ranges of hyper-parameter values, as detailed in Section 4.2. Because the training of each separate model can take a considerable amount of time, commonly in SCA, the size of the set H has to be limited to achieve results in a reasonable time. As an example, the training of a deep neural network from a dataset containing millions of traces may take, in an optimistic scenario (using parallel GPUs), several minutes for a single key byte. If an optimization process for hyper-parameters requires evaluation of hundreds of different hyper-parameter groups, a profiled attack on a 16-key bytes AES implementation will take weeks or even months.

For this reason, it sounds reasonable to take the most out of the hyper-parameter tuning phase, and explore whether one can use more than a single machine learning model

obtained during the tuning phase. We emphasize that one acquires numerous machine learning models due to the tuning phase, so there is no added computational complexity. The only addition stems from the fact that one needs to combine multiple models in the attack phase, but the complexity of that grows linearly in the number of used models (we only require the additional summation of output probabilities over all trained models). Finally, we will demonstrate that using even a small number of models in ensembles, e.g., 10, improves performance significantly.

4.1 Model Ensembles for Profiled Side-channel Analysis

Ideally, for neural networks, the hyper-parameters for each model should vary in order to learn different features from the same training set. If the models have equal configurations, it is likely that the neural network will learn similar representations from the training set and provide similar classification results for the same test set. Here, the main goal of ensemble learning is to improve the success rate of profiled side-channel attacks. This will be confirmed by comparing the convergence of guessing entropy in comparison to the convergence observed for the single selected best model. When a deep learning-based side-channel attack is successful, the main reason for that comes from the summation probability (see Eq. (1)) being larger for the correct key candidate in the middle and lower ranks (see for example Figure 2b where the lower ranks are more concentrated for the correct key candidate). As a result, successful ensembles should increase the summation probability for the correct key candidate while averaging out the variations for the incorrect candidates.

Note, however, that we are not assuming that ensemble learning is *always* better if compared to a single learning model resulting from an optimal hyper-parameter search. The main goal of this analysis is to demonstrate that the chances of success in terms of key recovery are higher for ensembles when compared to single models elected from a limited number of hyper-parameters groups, which is confirmed for several public side-channel datasets. For a very large training set, one expects that a complex deep neural network needs to be defined. This actually requires a careful selection of hyper-parameters, and to try each new configuration may take a significant amount of time and computation power. Moreover, because several models are combined into one, if few of the models contain hyper-parameters that do not provide good learnability from training sets, the fluctuations introduced by these models will be removed by having models that are generalizing. The experiments provided in this section demonstrate that ensembles are more likely to succeed if compared to single learning models for a limited number of H trained models.

In a hyper-parameter search, the large number of possible hyper-parameter combinations would naturally create a number of neural network architectures that are unable to generalize to a separate test set. To be able to generalize over the space of hyper-parameters Λ, it is crucial to have an ensemble that does not contain a majority of bad classifiers.

Eq. (2) indicates that the selection of a good model from hyper-parameter optimization process takes into account the loss function. This is obvious, as the model with the lowest validation loss would indicate the best generalization case. However, as empirically demonstrated in the previous sections and also highlighted in [PHJ+19], the key rank, which is obtained from output class probabilities, is a more informative metric with respect to generalization for profiled side-channel analysis. Therefore, instead of using the original proposition from [HKV19], we modify Eq. (2) to select the best model according to:

$$h_{best} = \underset{\lambda \in \Lambda}{\operatorname{argmin}}\, GE(h_\lambda, t_{train}, t_{val}), \tag{3}$$

where $GE_{val}(h_\lambda, t_{train}, t_{val})$ refers to guessing entropy computed over the **validation** trace set, t_{val}, after training the model h_λ with a training trace set t_{train}.

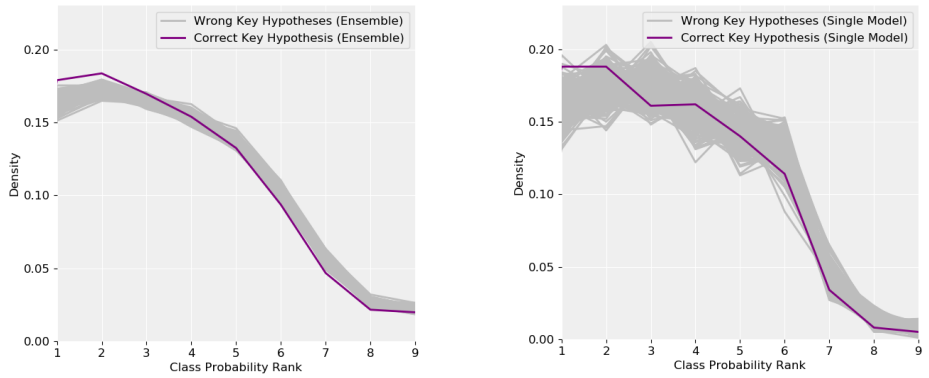
The selected best model, h_{best} provides output class probabilities $p_{h_{best},i,j}$, where i refers to trace i and j is the class probabilities obtained for leakage model l, key guess k, and input pk_i. Instead of selecting simply the best model, we compute an ensemble of best models by summing up the output class probabilities from several trained models. In this work, we consider the bagging option to build ensembles. The applied method computes the new likelihood for each key candidate by summing up the probabilities from all individuals models. The summation probabilities for ensemble learning $S_e(k)$ are then computed for each key byte candidate k as follows:

$$S_e(k) = \sum_{m=1}^{Z} \sum_{i=1}^{Q} (p_{m,i,j}). \tag{4}$$

In Eq. (4), the term Z (where $Z \leq H$) refers to the number of machine learning models. The term $p_{m,i,j}$ refers to the output class probability vector for model m and trace i according to the key guess k, leakage model l, and input pk_i. After building the ensembles, we expect an improvement in the key ranking convergence, which, in the end, comes from the ensemble summation probabilities. The main goal of this analysis is to compare ensemble learning in SCA when:

- Ensembles are built from all single trained models.
- Ensembles are built from E_b best trained single models. The best models are selected based on key rank for the validation set according to Equation (3).

To better explain the difference in performance between ensembles and a single best model, we consider the class probability ranks for the CHES CTF 2018 dataset in Figure 5. There are two main effects we observe 1) for the correct key guess, ranks 2 to 4 improve, which means that the correct key guess will have higher summed probability, and 2) for wrong key guesses, we observe that they behave more stable and the top values for the highest ranks are significantly reduced. This means that the wrong key guesses will have smaller summed probabilities. Consequently, the difference between the correct and wrong key guesses will be larger, and the target will be broken with fewer measurements.



(a) Ensemble results. **(b)** Single best model results.

Figure 5: Rank of output class probabilities computed from CHES CTF 2018 dataset. Ensemble of 50 models.

4.2 Results on Publicly Available Datasets

This section provides experimental results for the bagging ensemble methodology. We also experimented with the stacking method, but it provided poor performance for the considered datasets. Experimental analysis of boosting will be reserved for future works.

The boosting method is a sequential learner where each step in this process is modified based on the errors provided by the previous learner. If the hyper-parameters are the same for each new iteration in the boosting process, the subset of training data needs to be optimized based on a metric or criteria. However, if the hyper-parameters are not optimal, the weak learner could provide limited performance, even considering an optimal subset of the training set. As a consequence, the number of iterations could be very large, significantly increasing the time complexity. On the other hand, if each new iteration considers a different hyper-parameters set, we would need to solve another optimization problem.

Next, we consider three publicly available datasets. We do not give results for the Piñata SW implementation as it is very easy to break. We consider both the Hamming weight and identity leakage models, but we provide the results for the latter only for the ASCAD dataset as this is the only case where there are random keys in the training phase. For other scenarios, i.e., when using the identity leakage models with a dataset that has the same key for the training and testing (DPAv4), our results show it can be easily broken. On the other hand, when there is a single key for training and a single, but different key for testing (CHES CTF 2018), we cannot break it with the identity leakage model (which is not surprising as the neural network never learned to generalize for different keys).

For each dataset, we compute the ensemble guessing entropy and success rate on a single AES key byte. For that, the hyper-parameter search consists of training $H = 50$ different models. These experiments are repeated for MLP and CNN. For each trained model, the hyper-parameters are randomly selected from predefined ranges defined in Tables 2 and 3 for MLP and CNN, respectively. Note that these ranges are selected on the basis of results from related works where we aimed to have a wide selection of options, but still options that are known to be able to result in a good attack performance. The unchanged hyper-parameters are *optimizer*, in which the adaptive Adam optimizer is selected, and weight initialization that adopts the *random uniform* method. Furthermore, the models are always trained for 50 epochs, as this was an amount leading to better generalization.

Table 2: Hyper-parameter search space for multilayer perceptron.

Hyper-parameter	min	max	step
Learning Rate	0.0001	0.001	0.0001
Mini-batch	100	1 000	100
Dense (fully-connected) layers	2	8	1
Neurons (for dense or fully-connected layers)	100	1 000	100
Activation function (all layers)	ReLU, Tanh, ELU, or SELU		

4.2.1 AES256 - DPAv4

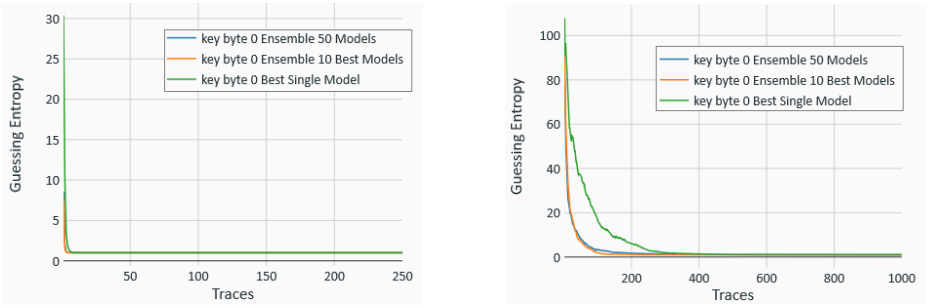
In Figure 6, we provide results for the Hamming weight leakage model and DPAv4 with guessing entropy. First, we observe that MLP performs better than CNN and that for MLP, there is almost no difference between using an ensemble of all available models, an ensemble of ten best models, or a single best model. Still, a small improvement (albeit not relevant from a practical perspective) can be seen for the ensemble of ten best models. On the other hand, when using CNN, we see that ensembles are significantly better than selecting a single best model. At the same time, there is no difference between the setting with 50 models in the ensemble or ten models in the ensemble.

In Figure 7, we consider the same setting, but this time, with the success rate metric. For MLP, the behavior is similar to the guessing entropy scenario. For CNN, we see more

Table 3: Hyper-parameters search space for convolutional neural network.

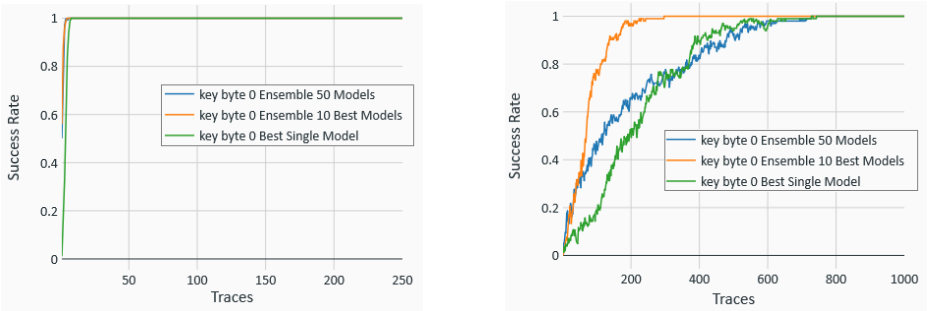
Hyper-parameter	min	max	step
Learning Rate	0.0001	0.001	0.0001
Mini-batch	100	1 000	100
Convolution layers	1	2	1
Filters	8	32	4
Kernel Size	10	20	2
Stride	5	10	5
Dense (fully-connected) layers	2	3	1
Neurons (for dense or fully-connected layers)	100	1 000	100
Activation function (all layers)	ReLU, Tanh, ELU or SELU		

significant differences where the ensemble of ten best models is significantly better than the ensemble of 50 models or a single best model. We note that with an ensemble of ten models, we require around 350 traces to reach a success rate of 1, while for the same performance for a single best model, we require around 750 traces.



(a) MLP **(b)** CNN

Figure 6: Guessing entropy for DPAv4 for the Hamming weight leakage model.

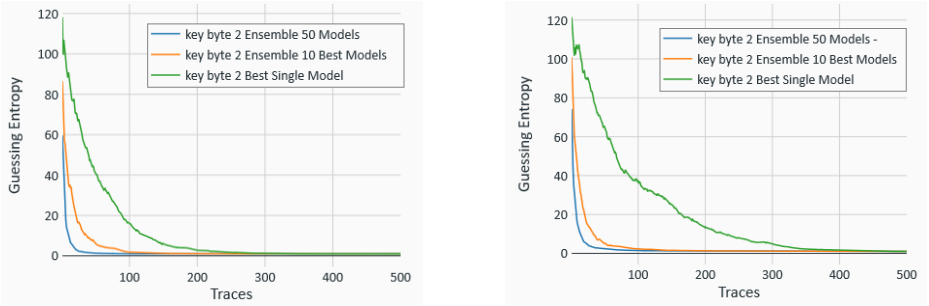


(a) MLP results. **(b)** CNN results.

Figure 7: Success rate for DPAv4 for the Hamming weight leakage model.

4.2.2 AES128 - ASCAD

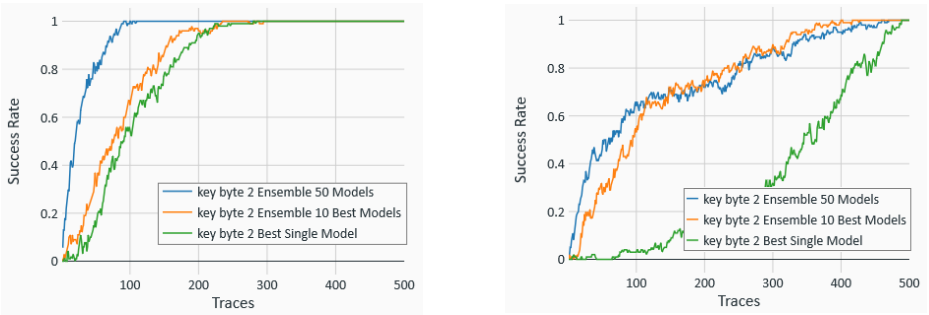
In Figures 8 and 9, we depict results for the ASCAD dataset in the Hamming weight leakage model, for guessing entropy and success rate, respectively. When considering guessing entropy, we see very similar behavior for both MLP and CNN. Interestingly, here the best option is to use an ensemble of all 50 models, which indicates that more than ten best models provided strong information to reduce the attack variance. In both cases, selecting only a single best model performs the worst, where the difference in the performance is around one order of magnitude when compared with ensembles of 50 models.



(a) MLP results. **(b)** CNN results.

Figure 8: Guessing entropy for ASCAD for the Hamming weight leakage model.

When considering the success rate, we can observe that the differences in performance are much smaller. This is because the success rate considers only the best key guess, which limits its power when compared to guessing entropy. With MLP, an ensemble of 50 models is the best, while an ensemble of ten models and one single best model perform significantly worse. For CNN, the situation is even more interesting as we see that all considered techniques reach a success rate of 1 after processing approximately the same number of traces. Both ensemble options perform better than a single model where this difference is especially striking if using a smaller number of traces in the training phase.

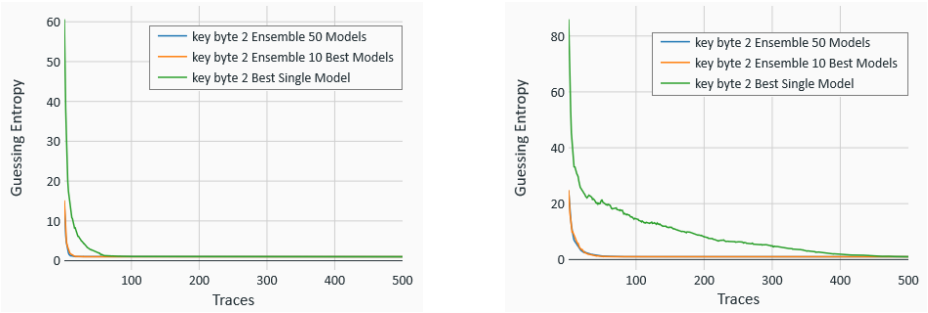


(a) MLP results. **(b)** CNN results.

Figure 9: Success rate for ASCAD for the Hamming weight leakage model.

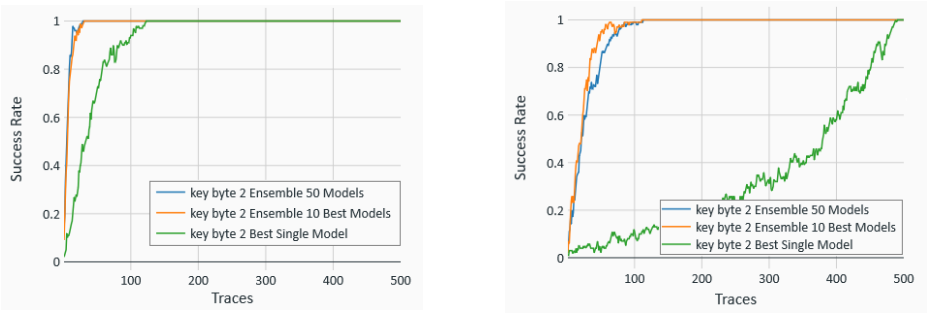
Next, we show the results for the ASCAD dataset for the identity leakage model (l =S-box(k, pk)) where the training set contains random keys. Figure 10 gives results for the guessing entropy metric. For both MLP and CNN, we observe that ensembles work better than a single best model, where the difference is especially pronounced for CNN (order of difference in the number of traces needed to reach guessing entropy of 0 for ensembles vs. the best model). Success rate results (Figure 11) show similar behavior where ensembles are better than a single model. This difference is rather small for MLP

but quite significant for CNN. More precisely, we see that with ensembles, we require around 100 traces to reach a success rate of 1, while with the best model, we need 500 traces.



(a) MLP results. **(b)** CNN results.

Figure 10: Guessing entropy for ASCAD for the identity leakage model.



(a) MLP results. **(b)** CNN results.

Figure 11: Success rate for ASCAD for the identity leakage model.

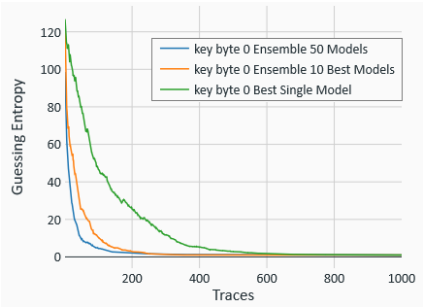
4.2.3 AES128 - CHES CTF 2018

Finally, in Figures 12 and 13, we show results for the CHES CTF dataset, the Hamming weight leakage model, for guessing entropy and success rate, respectively. Here, ensembles again outperform a single best model, which confirms our previous results. What is more, we see that 50 models work better than a subset of 10 best models. Differing from previous settings, here, CNN outperforms MLP as it requires half the traces to reach guessing entropy of 0 for the best performing method.

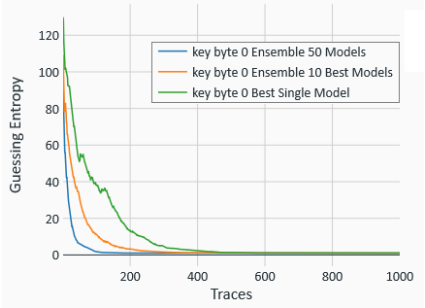
5 Conclusions and Future Works

This paper proposes an analysis of the deep learning model generalization in the SCA context. First, we provide an analysis of the output class probabilities from the test phase of a deep learning-based side-channel analysis. We demonstrate how output class probabilities contribute to the successful attack with all the probabilities (ranks) and not only the highest ones. Next, we show that those output class probabilities are sensitive to small changes in the neural network architectures.

To strengthen the performance of side-channel attacks and make the output class probabilities more stable, we propose to use ensembles of neural networks. Our results

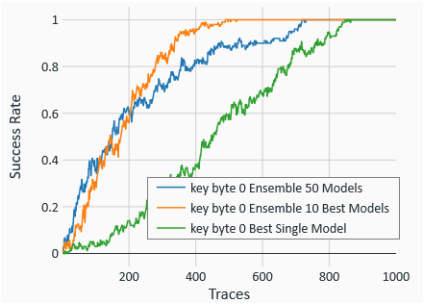


(a) MLP results.

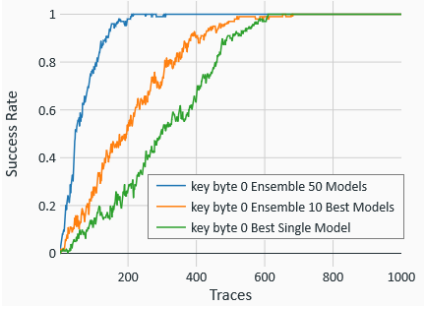


(b) CNN results.

Figure 12: Guessing entropy for CHES CTF 2018 for the Hamming weight leakage model.



(a) MLP results.



(b) CNN results.

Figure 13: Success rate for CHES CTF 2018 for the Hamming weight leakage model.

confirm that ensembles perform very well and, on average, better than single (best) models. The main conclusions from the use of ensembles in the deep learning-based SCA are:

- Ensembles are good alternatives to make better usage of several hyper-parameter groups tested for a single key byte. Combining the outcome of several trained models increases attack performance.
- The use of ensembles relaxes the need for a careful selection of a strong group of hyper-parameters in neural networks. Still, we *do not* conclude that ensembles replace an optimal hyper-parameter tuning method.
- We *do not* assume or conclude that ensembles improve the correct learnability of neural networks from side-channel traces. If the model is configured and trained in a way that the generalization is very poor, likely, that ensembles will not improve the generalization.
- The ensembles tend to provide a success rate that is at least good as the best single model among several hyper-parameters configuration options. In fact, our results showed that a single model never outperformed ensemble.

As mentioned in Section 2, ensembles are built with different approaches. As future work, we plan to investigate the stacking ensemble methodology. Moreover, we aim to investigate the benefits of ensemble learning in combination with regularization techniques.

References

[BCH+19] Shivam Bhasin, Anupam Chattopadhyay, Annelie Heuser, Dirmanto Jap, Stjepan Picek, and Ritu Ranjan Shrivastwa. Mind the portability: A warriors

guide through realistic profiled side-channel analysis. *IACR Cryptology ePrint Archive*, 2019:661, 2019.

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

[BGL09] Lejla Batina, Benedikt Gierlichs, and Kerstin Lemke-Rust. Differential cluster analysis. In Christophe Clavier and Kris Gaj, editors, *Cryptographic Hardware and Embedded Systems - CHES 2009, 11th International Workshop, Lausanne, Switzerland, September 6-9, 2009, Proceedings*, volume 5747 of *Lecture Notes in Computer Science*, pages 112–127. Springer, 2009.

[CCC+19] Mathieu Carbone, Vincent Conin, Marie-Angela Cornelie, François Dassance, Guillaume Dufresne, Cécile Dumas, Emmanuel Prouff, and Alexandre Venelli. Deep learning to evaluate secure RSA implementations. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(2):132–161, 2019.

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

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

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

[GBTP08] Benedikt Gierlichs, Lejla Batina, Pim Tuyls, and Bart Preneel. Mutual information analysis. In Elisabeth Oswald and Pankaj Rohatgi, editors, *Cryptographic Hardware and Embedded Systems - CHES 2008, 10th International Workshop, Washington, D.C., USA, August 10-13, 2008. Proceedings*, volume 5154 of *Lecture Notes in Computer Science*, pages 426–442. Springer, 2008.

[GD98] Matt W Gardner and SR Dorling. Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences. *Atmospheric environment*, 32(14-15):2627–2636, 1998.

[HGG18] Benjamin Hettwer, Stefan Gehrer, and Tim Güneysu. Profiled power analysis attacks using convolutional neural networks with domain knowledge. In Carlos Cid and Michael J. Jacobson Jr., editors, *Selected Areas in Cryptography - SAC 2018 - 25th International Conference, Calgary, AB, Canada, August 15-17, 2018, Revised Selected Papers*, volume 11349 of *Lecture Notes in Computer Science*, pages 479–498. Springer, 2018.

[HGG19] Benjamin Hettwer, Stefan Gehrer, and Tim Güneysu. Deep neural network attribution methods for leakage analysis and symmetric key recovery. *IACR Cryptology ePrint Archive*, 2019:143, 2019.

[HKV19] Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren, editors. *Automated Machine Learning - Methods, Systems, Challenges*. The Springer Series on Challenges in Machine Learning. Springer, 2019.

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

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

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

[Mag19] Houssem Maghrebi. Deep learning based side channel attacks in practice. *IACR Cryptology ePrint Archive*, 2019:578, 2019.

[MDP19a] Loïc Masure, Cécile Dumas, and Emmanuel Prouff. A comprehensive study of deep learning for side-channel analysis. *IACR Cryptology ePrint Archive*, 2019:439, 2019.

[MDP19b] Loïc Masure, Cécile Dumas, and Emmanuel Prouff. Gradient visualization for general characterization in profiling attacks. In Ilia Polian and Marc Stöttinger, editors, *Constructive Side-Channel Analysis and Secure Design - 10th International Workshop, COSADE 2019, Darmstadt, Germany, April 3-5, 2019, Proceedings*, volume 11421 of *Lecture Notes in Computer Science*, pages 145–167. Springer, 2019.

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

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

[PEC19] Guilherme Perin, Baris Ege, and Lukasz Chmielewski. Neural network model assessment for side-channel analysis. *IACR Cryptology ePrint Archive*, 2019:722, 2019.

[PHJ+17] S. Picek, A. Heuser, A. Jovic, S. A. Ludwig, S. Guilley, D. Jakobovic, and N. Mentens. Side-channel analysis and machine learning: A practical perspective. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 4095–4102, 2017.

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

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

[Ris20] Riscure. Piñata (training target), 2020. https://www.riscure.com/product/pinata-training-target/.

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

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

[SMY09] François-Xavier Standaert, Tal Malkin, and Moti Yung. A Unified Framework for the Analysis of Side-Channel Key Recovery Attacks. In *EUROCRYPT*, volume 5479 of *LNCS*, pages 443–461. Springer, April 26-30 2009. Cologne, Germany.

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

[Tim19] Benjamin Timon. Non-profiled deep learning-based side-channel attacks with sensitivity analysis. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(2):107–131, 2019.

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

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

[WMM19] Felix Wegener, Thorben Moos, and Amir Moradi. DL-LA: deep learning leakage assessment: A modern roadmap for SCA evaluations. *IACR Cryptology ePrint Archive*, 2019:505, 2019.

[Wol02] David H. Wolpert. The supervised learning no-free-lunch theorems. In Rajkumar Roy, Mario Köppen, Seppo Ovaska, Takeshi Furuhashi, and Frank Hoffmann, editors, *Soft Computing and Industry: Recent Applications*, pages 25–42, London, 2002. Springer London.

[WPB19] Leo Weissbart, Stjepan Picek, and Lejla Batina. One trace is all it takes: Machine learning-based side-channel attack on eddsa. *IACR Cryptology ePrint Archive*, 2019:358, 2019.

[YLMZ18] Guang Yang, Huizhong Li, Jingdian Ming, and Yongbin Zhou. Convolutional neural network based side-channel attacks in time-frequency representations. In Begül Bilgin and Jean-Bernard Fischer, editors, *Smart Card Research and Advanced Applications, 17th International Conference, CARDIS 2018, Montpellier, France, November 12-14, 2018, Revised Selected Papers.*, volume 11389 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 2018.

[ZS19] Yuanyuan Zhou and François-Xavier Standaert. Deep learning mitigates but does not annihilate the need of aligned traces and a generalized resnet model for side-channel attacks. *Journal of Cryptographic Engineering*, 04 2019.

A Notation

Table 4: Notation.

Symbol	Description
t_{train}, N	training set of size N
t_{val}, V	validation set of size V
t_{test}, Q	test set of size Q
x_i	side-channel measurement i
$x_{i,f}$	feature f in side-channel measurement i
l, HW	leakage model, Hamming weight
k, k^*, $\mathcal{K}$, $\lvert\mathcal{K}\rvert$	key guess, secret key, keyspace, keyspace size
pk	input data
GE, g	guessing entropy, guessing entropy vector
y, y'	true classes, predicted classes
P	probability matrix
$p_{i,j}$, $p_{m,i,j}$	probability vectors in P
L	loss function
H, h	set of machine learning models, machine learning model
E_b, Z	ensemble of best models, number of machine learning models
Λ, λ	set of hyper-parameters groups, group of hyper-parameters

B Machine Learning Algorithms

The multilayer perceptron (MLP) is a feed-forward neural network that maps sets of inputs onto sets of appropriate outputs [GD98]. MLP consists of multiple layers of nodes in a directed graph, where each layer is fully connected to the next one, and training of the network is done with the backpropagation algorithm. There are at least three layers: one input layer, one output layer, and one hidden layer. If there is more than one hidden layer, then such an architecture already represents deep learning.

Convolutional neural networks (CNNs) commonly consist of three types of layers: convolutional layers, pooling layers, and fully-connected layers. Convolution layer computes the output of neurons that are 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 decrease the number of extracted features by performing a down-sampling operation along the spatial dimensions. The fully-connected layer (the same as in MLP) computes either the hidden activations or the class scores. To avoid the overfitting, batch normalization layer, which normalizes the input layer by adjusting and scaling the activations, is commonly used.

C Class Probabilities vs. Number of Traces

In Tables 5 and 6, we depict the key rank probabilities for the ASCAD dataset and the Hamming weight leakage model. We consider the results if taking only the best rank, the sum of two best ranks, or the sum of the three best ranks. We give results for four different test set sizes. For convenience, we put the values in red color if the correct key is worse than the best wrong, and green color otherwise. In Figure 5, we consider the case where we compare the correct key with the best wrong key up to a certain ranking. This means that the best wrong key can change from setting to setting. First, we consider only the highest probability rank; this means, for instance, if the correct key is highest ranked for a trace, we add it to the sum. Otherwise, we add nothing. Going to the first two ranks, we add the probability if the correct key is in the first two ranks, otherwise, we add nothing. For wrong keys, we follow the same procedure, but we take the best wrong key (which means there is always a value to add to the sum of probabilities). By doing so, we aim to depict the behavior of wrong keys as from figure, it is not always easy to discern it since 255 lines overlap.

Considering the results, we can see that the correct key has lower rankings if we take a smaller number of traces or only the best rank. By adding more information (regardless if it is in the form of more traces or more ranks), the correct key improves, while the best wrong key becomes worse. If we consider only the best rank, we see that even 1 000 traces is not enough to reach guessing entropy 0. By adding the information from the second best ranking, we notice that 750 traces becomes enough to conduct a successful attack. Naturally, considering more ranks or taking more traces makes the distinction between the correct and the best wrong key even larger, and thus, the attack easier.

In Table 6, we consider a similar setting, but now, we take only the wrong key that was the best for the first key rank. We see a similar behavior as for the previous example. The difference is that now, reaching guessing entropy of 0 is easier. This is because the best wrong key for key rank 1 does not stay the best wrong key when taking ranks 1 and 2, etc.

Table 5: Density for class probability ranks 1, 2, and 3 for the ASCAD dataset in the Hamming weight model.

	rank 1		rank 1 + rank 2		rank 1 + rank 2 + rank 3	
Nr traces	Correct key	Best wrong key	Correct key	Best wrong key	Correct key	Best wrong key
250	0.268	0.314	0.505	0.572	0.690	0.755
500	0.274	0.299	0.520	0.547	0.708	0.732
750	0.282	0.290	0.540	0.535	0.730	0.725
1 000	0.283	0.284	0.539	0.529	0.735	0.725

Table 6: Density for class probability ranks 1, 2, and 3 for the ASCAD dataset in the Hamming weight model (**best wrong key selected from rank 1 only.**).

	rank 1		rank 1 + rank 2		rank 1 + rank 2 + rank 3	
Nr traces	Correct key	Best wrong key	Correct key	Best wrong key	Correct key	Best wrong key
250	0.268	0.314	0.505	0.529	0.690	0.744
500	0.274	0.299	0.520	0.547	0.708	0.720
750	0.282	0.290	0.540	0.526	0.730	0.698
1 000	0.283	0.284	0.539	0.519	0.735	0.701