Everything All at Once: Deep Learning Side-Channel Analysis Optimization Framework

Gabriele Serafini(✉), Léo Weissbart, and Lejla Batina

Radboud University, Nijmegen, The Netherlands
{gabriele.serafini,leo.weissbart,lejla.batina}@ru.nl

Abstract. Deep learning is becoming an increasingly proficient tool for side-channel analysis. While deep learning has been evolving around the tasks of image and speech recognition for decades, it is still lacking maturity for side-channel analysis. One of the challenges to train a good model is the fine-tuning of its hyperparameters. Many methods have been developed for Hyperparameter Optimization, but a few have been applied for deep learning side-channel analysis.

We study the use of sampling algorithm and early-stopping mechanism in the hyperparameter optimization search for deep learning side-channel analysis models. We also offer a scalable deep learning framework to extend results obtained for other problems and datasets.

Our results show that hyperparameter optimization methods can save time and resources while leading to models that can lead to the best possible output and at the same time are providing more confidence whether to look for a better model.

Keywords: Deep Learning · Side-Channel Analysis · Hyperparameter Optimization · Pruning

1 Introduction

From personal smart gadgets to sophisticated industrial machinery, devices embedded with sensors, software, and other technologies are becoming an integral part of our daily lives. These devices, collectively referred to as the Internet of Things (IoT), communicate over a common network, and the security of this network depends on the reliability of the least secure device. However, as the integration of IoT devices expands, the need for secure cryptographic implementations grows bigger. One of the most critical concerns is the potential vulnerability of these devices to cyberattacks, which can lead to unauthorized access, data theft, or even taking control over the device's functions, thus posing significant risks to users' privacy and security.

While strong cryptographic designs can give guaranties on the security of an application, there remains numerous possibles threats. Side-Channel Analysis (SCA) is a set of techniques aimed to explore the possible vulnerabilities of

M. Andreoni (Ed.): ACNS 2024 Workshops, LNCS 14586, pp. 195–212, 2024.
https://doi.org/10.1007/978-3-031-61486-6_12

implementations of secure designs on physical devices. These techniques entail the analysis of information contained in unintentional leakages from electronic devices during operations involving sensitive secrets. This analysis can provide insights into the device's internal workings, exposing vulnerabilities to security breaches.

The assessment of side-channel leakages is a part of standard evaluations for electronics in commercial use, and typically involves the use of classic techniques e.g., Differential Power Analysis [10] or Template attack [6]. While research on SCA becomes more mature, new analysis methods are developed that add up to the standard techniques for side-channel evaluators, such as Deep-Learning SCA.

Among the various deep learning techniques, Convolutional Neural Networks (CNNs) turned out to be particularly suited for in profiling SCA. CNNs are designed to automatically and adaptively learn spatial hierarchies of features, which make them effective at recognizing subtle and complex patterns in side-channel data. When trained on a side-channel dataset, a CNN model can learn to associate the secret key to certain power leakage of a cryptographic operation by extracting the important features it contains.

A fundamental challenge when using CNN, as a profiling SCA tool, is the tuning of its hyperparameters. Hyperparameters define the configuration of a model, and have a significant impact on a model performance. The sheer size of the hyperparameter space to explore to find a fitting model and the cost of evaluating each configuration make the hyperparameter optimization (HO) task computationally expensive and time-consuming. Moreover, there exists no guaranty for a CNN model that, if after the training phase the underlying attack is not successful, no better configuration that would make the attack succeed could be found. Thus, to ensure that an implementation is secure against CNN-based SCA, an exhaustive HO search, exploring all possible configurations of hyperparameters remains the best method.

The use of optimization techniques to tune hyperparameters can help to ease the HO search to some extends. One such method is the use of sampling based on optimization algorithms e.g., Bayesian optimization [25] or Genetic algorithms [12], for exploring the search space of hyperparameter configurations in an educated manner, and increase the chances that the best configurations are explored the earliest possible during the search. Another method used to enhance deep learning is Early Stopping. Based on the results of previous models during the training phase, this method allows for the early termination of an unpromising model training based on an intermediate results [2]. This method can save computational resources and allows the HO process to allocate more time to explore other, potentially better hyperparameter configurations.

In this paper, we aim to provide a first guideline towards a framework that integrates efficiently HO for CNN-based side-channel analysis. We demonstrate how the conjunction of early-stopping and optimization method, namely median Early Stopping and tree-structured Parzen Estimator (TPE), can computationally ease the HO search. The following points provide a more detailed overview of the study's focal areas:

1. We give an introductory analysis to enhance the efficiency of hyperparameter optimization in CNN-based SCA. We show it is possible to speed up the HO process while minimizing the compromise on the quality of results. The integration of sampling method and early-stopping offers more flexibility in dealing with the exploration-exploitation trade-off during hyperparameter tuning.
2. We introduce the use of Guessing Entropy (GE) inside sampling and early-stopping to reach faster convergence to good fitting models.
3. We demonstrate how the use of patience early stopping together with median early stopping can save resources by preventing long training of models that appeared to have already converged and won't learn further, and models that appear to have little likelihood of improving previously run models.
4. We share the code used for model training and HO. The utilization of several machine learning packages simplifies the hyperparameter optimization process and is specifically crafted for SCA, thus paving the way for further exploration with other optimization algorithms or machine learning frameworks in the SCA setting.

The rest of the paper is organized as follows. In Sect. 2, we provide a brief overview of existing research in the area of CNN-based side-channel analysis and hyperparameter optimizations for this type of attacks. In Sect. 3, we provide necessary information on both SCA and deep learning attack. In Sect. 4, we present the methodology we applied to create experiments, and especially the datasets and framework we use. In Sect. 5, we show the results obtained from our experiments with efficiency of the best obtained model, and saving computational resource compared to other results from the literature. Finally, in Sect. 6, we sum up our results and identify some future research directions.

2 Related Works

The studies of deep learning for side-channel analysis have gained great interest in recent years [21]. CNN-based deep learning have been shown many times to be particularly efficient for attacks against masking countermeasure in cryptography [11,15,17,22,28,30]. Wouters et al. in [28] managed to reduce the complexity of the CNNs proposed by Zaid et al. [30] by an average of 52% while maintaining similar performance. The authors argue that increasing the filter size can actually improve the performance of the network. The review also emphasizes the importance of pre-processing side channel traces, and shows that with proper pre-processing the first convolutional block proposed in [30] to extract features can be omitted, which contributes to reducing the complexity of the models. Their work highlights the need for proper evaluation of the hyperparameter search space to conclude on the security of a given dataset.

In an effort to build a universal framework for deep-learning SCA to guaranty better result reproducibility, Perin et al. [20] and Brisfors et al. [5] introduce two publicly available frameworks designed to streamline deep learning based side channel analysis. In these frameworks, the objective is to provide a basis for SCA

evaluations with deep learning, and provide useful functionalities published in the literature.

Early-stopping and hyperparameter tuning for deep-learning SCA in discussed in [18,24]. The authors of [24] introduces the six sigma methodology for choosing the best possible hyperparameters. The method is a derivative from a well known methodology for various engineering problems often used to reduce the variability of industrial processes.

In [18], Paguada et al. introduces an early-stopping method designed for SCA model training. The early-stopping of the training is evaluated based on the guessing entropy of the resulting attack, using the newly defined patience and persistence of the guessing entropy. Our approach also uses optimized computation of the guessing entropy and combines an automated search for the best hyperparameters with a tree-structured Parzen optimization. Wu et al. in [29], also suggest an automated method for modifying the hyperparameters, but using Bayesian optimization. This paper also compares the tracking evolution of the guessing entropy, accuracy, and leakage difference distribution of the resulting attack to choose the best fitting hyperparameters, but do not implement early-stopping methods.

3 Background

3.1 Profiling SCA

Profiling SCA is a type of SCA that is opposed to non-profiled SCA (i.e., Single Power Analysis and Differential Power Analysis). Contrary to non-profiled SCA in which an attacker is assumed to have physical access to the device under attack, profiling SCA makes the assumption that an attacker also has access to and full control of an identical copy of the device under attack, and can learn about leakage to profile before accessing the device under attack. This technique is commonly divided in two phases:

1. A profiling phase: during this phase, the attacker collects side channel traces from a copy of the target device while running cryptographic operations with known secret. This data is then used to build a predictive model (i.e. a profile) of the device's behavior.
2. An attack phase: after the profiling phase, the attacker applies the developed model to the target device while it is executing the same cryptographic operations with unknown key. If the model developed during the profiling phase is accurate, the attacker can infer the secret key.

3.2 CNN and Profiling SCA

Convolutional Neural Networks (CNNs) are a specific type of feed-forward neural network that have become standard for tasks involving image and signal processing. A CNN is generally composed of multiple convolutional layers followed by fully connected layers. The input of a convolution layer is multiplied

by several filters (also known as kernels) which are shifted along the input data, performing a dot product operation at each position, and creating a so-called feature map. During this convolution operation, the kernels act as feature extractor, each learning to identify a different pattern in the input data. The patterns can range from simple edges to complex shapes or textures, and they emerge automatically during the training process. Once the features are extracted, the output of the convolution layers goes to a fully-connected network that acts as a classifier.

A CNN, as well as other deep learning algorithms, is divided in two stages: a training and an evaluation phase. During training, the neural network is challenged with raw input data and is followed by backpropagation correcting the differential between the predicted and known labels. In the evaluation phase, the network is challenged with a never previously used input data, and outputs a probability distribution to classify the input based on the trained task.

Because of the similar two-stage structure of profiling SCA and deep learning classification, the later can be applied as a profiling method in SCA. In such a framework, the aim is to classify observed side-channel leakages into corresponding secret information. Given this formulation, machine learning techniques emerge as a natural choice for enhancing the efficiency and efficacy of profiled side channel attacks.

SCA often involves extracting useful information from noisy, high-dimensional data, a process that CNNs are exceptionally well-equipped to handle. The properties of CNN align with the requirements of side channel analysis for extracting features in a high dimensional power trace and overcome invariance of a leakage position. The learned features are more representative and discriminative, enabling them to effectively identify and classify patterns in side channel data.

The property of translation invariance of CNNs is useful in the context of side channel analysis. This property ensures that once a feature from a leakage, such as a specific pattern, has been learned, the CNN model can recognize the same pattern regardless of its position in any new side channel trace, making CNNs effective tools to work effectively with raw, unprocessed data. This is particularly beneficial in side channel analysis, where preprocessing steps such as trace alignment can be challenging and time-consuming.

3.3 SCA Metrics

Precision, recall and accuracy, often fall short in deep-learning SCA due to the binary nature of their result. These metrics focus on whether a single prediction is correct, overlooking an attacker's strategy of narrowing down potential keys across many traces. A more informative metric in SCA is Guessing Entropy (GE), which measures the average number of keys as attacker would need to guesses before finding the correct one.

Guessing entropy can be modeled as follows:

$$GE = E\left[rank_{k^*}(\mathbf{g})\right] \tag{1}$$

In this context, $rank_{k^*}$ denotes the position of the correct key k^* in an ordered vector of candidate keys. The vector $\mathbf{g}$ represents the output array of keys, sorted in an order determined by the likelihood of each candidate key being the correct one according to the classifier. The expectation is computed by averaging the rank value over multiple experiments.

3.4 Leakage Model

A leakage model is a critical component that links the observed side-channel information to the intermediate values of the cryptographic operation being performed. The leakage model thus forms a fundamental bridge between the physical world observations and the mathematical properties of the cryptographic algorithm, allowing for an attacker to make informed deductions about secret data.

The basic premise behind leakage models is that the side-channel emissions of a device are not random, but depend on the device's internal state and operations. In the case of a cryptographic device, these operations are influenced by the data being processed and the secret key. The leakage model aims to describe this relationship, essentially serving as a predictor of side-channel emissions based on the known intermediate values of the cryptographic algorithm.

In this paper, only the Identity Leakage Model (ID) is considered.

This model assumes a direct relationship between the observed side-channel leakage and the secret intermediate values during cryptographic operations.

Under the ID model, the observed leakage is assumed to be the identity of the intermediate value. This suggests that the leakage directly reveals the intermediate value, without any noise or distortion.

Mathematically, this can be expressed as:

$$L = V \tag{2}$$

where, L denotes the observed leakage and V represents the secret intermediate value.

In the context of the Advanced Encryption Standard (AES), for instance, the intermediate value could be the result of the application of the S-box in the first round, which can be expressed as $V = Sbox(key + input)$.

This would result in 256 possible classes (corresponding to the 256 possible values of an 8-bit output), each representing a distinct intermediate value.

3.5 Hyperparameter Optimization

Hyperparameter optimization (HO) is a critical step in the development of ML algorithms. It entails determining an optimal set of hyperparameters, that minimizes or maximizes an established fit function, and so improves the model's performance. The hyperparameters are chosen before training a model and cannot be changed later. Moreover, it is not possible to predict that a configuration of hyperparameters will lead to a good model.

Learning rate, number of layers in a neural network, number of hidden units in each layer, type of activation functions, and many more parameters are examples of hyperparameters. When we initially apply a model, the ideal values for these hyperparameters are often not obvious, and they might be highly problem-dependent.

To find a suitable set of hyperparameters, one needs to explore different configurations of a given model. Common methods for exploring hyperparameters are:

- Grid Search: This is the simplest strategy, in which we establish a subset of the hyperparameter space and methodically attempt all combinations within the defined grid.
- Random Search: Unlike grid search, random search draws hyperparameter values at random from a given space. When the number of hyperparameters is considerable, this method may be more efficient than grid search.
- Bayesian Optimization: This method creates a posterior distribution of functions that best describes the function to be optimized, and then uses it to select the most promising hyperparameters to evaluate in the true function.

3.6 Tree-Structured Parzen Estimator

The process of selecting the hyperparameters that optimize a model's performance can be time-consuming and computationally expensive. It is, nonetheless, critical for developing strong and accurate ML models. In this study, we use the Tree-structured Parzen Estimator (TPE) [4], a Sequential Model-Based Optimization (SMBO) approach.

TPE constructs a model of the objective function with the goal of suggesting more promising hyperparameters for evaluation.

TPE models two probability distributions: $P(x|y)$ and $P(y)$, where x represents the hyperparameters, and y is the corresponding objective function. Through this modelling, TPE assigns greater weight to regions of the hyperparameter space it deems promising.

The operational procedure of TPE consists of the following steps:

1. TPE starts its process by randomly selecting n hyperparameters configurations from the search space and evaluating the objective function for them.
2. These hyperparameters are then fitted into two distinct distributions: one for those that yielded better outcomes of the objective function (termed "good" values), and another for the remainder of the hyperparameters (termed "bad" values).
3. In subsequent iterations, TPE samples more from regions having a high ratio of good to bad hyperparameter values. This ratio is determined by a parameter, γ, which is a quantile threshold separating the good and bad hyperparameters based on their objective function outcomes.

Through this process, TPE continuously adapts its focus towards the promising areas of the hyperparameter space, while diminishing attention on areas known to yield inferior results.

3.7 Early Stopping Strategies in Hyperparameter Optimization

Early stopping strategies in HO to the process of halting training of non-promising hyperparameter configuration as early as possible during the tuning process, saving computational resources. These techniques are particularly useful for tuning complex machine learning architectures, where the computational cost can be quite high [27]. Among early stopping strategies, the most notable are: median early-stopping, asynchronous successive halving algorithm [13], and hyperband algorithm [9,14]. These strategies monitor a model training performance and use it to decide whether to stop the training prematurely. All these strategies are adaptive early-stopping mechanisms that adapt the rules to trigger the training stop based on the tracked monitored value. Early stopping strategies were initially viewed as HO strategies when applied to grid search or random search, but it has been demonstrated how these strategies can actually enhance Optimization algorithms such as Bayesian Optimization [26,27].

In particular, we explore median early-stopping in this paper. Median-early stopping will stop the training of a model when the objective value by the current epoch is worse than the median value of the running averages of all completed previous trials objective values. Another early-stopping rule that can be applied is the Patience-Early stopper. This method can be used to wrap an early-stopping rule with a patience parameter that will trigger the wrapped rule after a fixed number of epochs where no improvement can be observed.

Despite their computational benefits, these strategies bear the risk of prematurely discarding configurations that may start slow but eventually outperforms others. This issue can augment with the double descent phenomena [16], potentially leading to suboptimal final configurations.

3.8 Warm-Up Values in Early Stopping Strategies

Warm-up values refer to the initial set of iterations or epochs during which the performance of the hyperparameter configurations is not evaluated for pruning. This grace period allows the configurations to stabilize and exhibit their potential before any pruning decisions are made. In our experiments, we utilize these warm-up values in conjunction with the Median-Early-Stopper. The Median-Early-Stopper operates by comparing the performance of each configuration against the median performance of all configurations at similar epochs. If a configuration's performance falls below the median, it is stopped, allowing resources to be focused on more promising configurations. To understand the impact of warm-up values on this process, we experiment with warm-up periods of 30, 50, 75, and 100 epochs. By varying these values, we aim to analyze how different lengths of the warm-up period affect the overall hyperparameter optimization process, particularly in terms of the timing and effectiveness of early stopping decisions.

4 Methodologies

4.1 Study Settings

The experiments utilize both the ASCADf and the ASCADr datasets [3]. Those datasets have been introduced in an initiative to provide a ground base for side-channel analysis of smartcard masked AES implementation. ASCADf dataset was the first published, and contains traces collected from encryption operations with a fixed key. The ASCADr dataset contains traces collected from encryptions with random keys and was meant to be more challenging than ASCADf. For both, we use the label obtained by using the Identity Leakage Model.

For the computational aspect, all the deep learning models were trained on an NVIDIA GeForce GTX 1080 TI Graphics Processing Unit (GPU).

The implementation of the experiment was conducted using Python. The TensorFlow [1] and Keras [7] libraries served as the foundation for constructing and training the deep learning models, while Optuna [2] was employed for the optimization of hyperparameters. For the purpose of visualizing results and monitoring the progress of our experiments, we used Matplotlib [8], TensorBoard, and Optuna's built-in visualization features. These tools allowed us to effectively track, analyze, and present the outcomes of our research in a clear and intuitive manner.

4.2 CNN Model Architecture

In this research, we leveraged the benefits of sampling optimization with early-stopping to perform hyperparameter optimization (HO) experiments on two SCA datasets: ASCADr and ASCADf, with the identity leakage model.

The experiments are executed each for 100 models. In these experiments, the HO process is guided using a customized objective function based on GE.

The CNN-based models can be decomposed in a feature extraction part (i.e., one or more convolutional blocks) followed by a multi-layer perceptron that acts as a classifier to recover the guessed intermediate value of an input trace. One convolutional block always is formed with a convolution layer and a pooling layer. An additional normalization layer is inserted every two blocks between the convolution and pooling layers to add regularization of parameters. A representation of the obtained models is showed in Fig. 1. The Table 1 lists the different hyperparameters chosen to explore and their range for this study.

We combined the capabilities of Optuna and Keras to dynamically construct CNN models, the structure, and parameters of which were determined by the Tree-structured Parzen Estimator (TPE) Sampling Algorithm used in the HO process.

To enhance the efficiency of our experiments and explore the potential benefits of early stop strategies, we conducted four tests for each dataset. Each test uses the Median-Early-Stopping and Patience-Early-Stopping techniques with varying degrees of intensity, adjusted by manipulating the warm-up values. For ASCADf, an additional experiment was conducted without employing any early

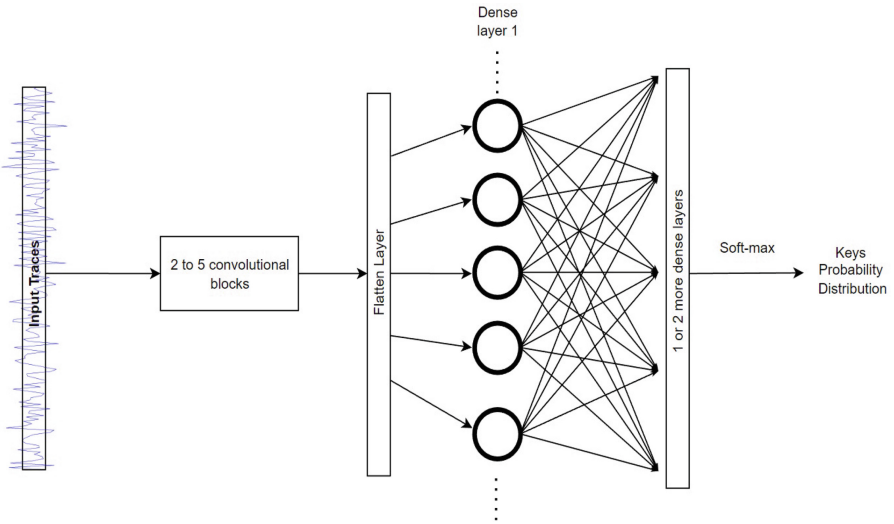

Fig. 1. General overview of how the convolutional neural networks are built

Table 1. Hyperparameter space explored in the optimization process

Hyperparameter	Range	Step
batch_size	$[100, \ldots, 1000]$	100
learning_rate	[1e-5] (Fixed)	None
activation_function	[relu, selu]	None
filters	$[2, \ldots, 64]$	$\times 2$
num_conv_layers	$[2, \ldots, 5]$	1
kernel_size	$[2, 3, 5, 7, 11, 17]$ (for each conv layer)	None
num_fc_layers	$[1, 2, 3]$	1
size_fc_layers	$[64, \ldots, 512]$ (for each fc layer)	64
epochs	200 or 300 (fixed)	None
strides	1 (fixed)	None

stopping method, providing a benchmark against which to assess the impact of these resource-saving techniques.

Finally, we assessed the experiments by selecting the best models from each and comparing their performance metrics. We compared the different models using GE.

4.3 Hyperparameter Optimization Process

Optuna was selected as the primary tool for conducting a series of HO experiments. The Tree-structured Parzen Estimator (TPE) sampling algorithm, a

Bayesian optimization algorithm provided by Optuna, was employed for the hyperparameter configuration selection.

An important component of the approach was the use of the averaged GE as the objective function to guide the HO process:

$$GE_{average} = \frac{GE(1) + GE(2) + ...GE(n)}{n} \tag{3}$$

where n is the size of the subset of traces taken from the validation set to estimate the key rank, which correspond to the number of used traces to compute the GE during the training. Utilizing the averaged GE as a metric during the HO process is helpful in comparing the efficiency between different models, since this metric gives an estimation of both the amount of traces needed to reach a low GE, and the amount of attempts needed for guessing the correct value. Optuna ultimately utilized this number as an objective value to guide the Hyperparameter selection of the future models based on TPE Algorithm.

5 Results

This section of our research delves into the results from our analysis of the two distinct datasets: ASCADf and ASCADr. We focus primarily on the hyperparameter optimization experiments conducted at various warm-up values for the Median Early Stopper, methodically studying their impact on resource use, measured in number of epochs, and the performance of the best model. This examination allows us to determine the effects of early stopping on the best models' result for each unique experiment.

5.1 Experiments for ASCADf

Table 2 encapsulates the impact of various warm-up periods on resource utilization during hyperparameter optimization processes. The columns labeled Median-Early-Stopper and Patience-Early-Stopper show the percentages of epochs spared due to the implementation of the respective early stopping strategies. These savings are calculated in comparison to the total number of epochs that would have been required if no early stopping mechanism were employed. This comparison highlights the efficiency gains achieved by employing these early stopping strategies in our hyperparameter optimization process. By saving epochs, we are able to evaluate more configurations using fewer resources, thus enhancing the overall efficiency and scalability of the optimization process.

Specifically, the Median Early Stopper column denotes the percentage of epochs saved because of the application of this strategy, the Patience Early Stopper column represents the percentage of saved epochs attributed to early stopping based on patience. Lastly, the Sum column aggregates these savings, providing an overall view of the resource conservation.

The total savings (represented by the Sum column), which combines the impacts of both the different early stopping strategies, doesn't exhibit a direct

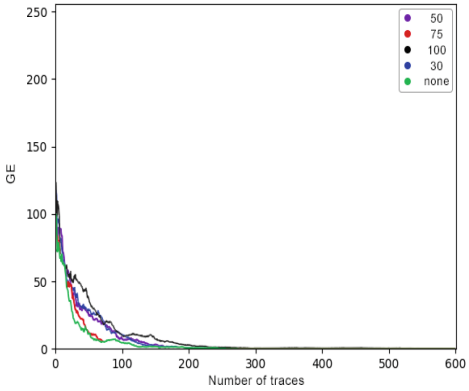

Fig. 2. Guessing entropy for the ASCADf experiments using warm up value of 30, 50, 75 and 100

Table 2. Summary of the effect of different warm-up values on the average number of epochs saved with early stopping during training on ASCADf (in percentage of a complete training)

Warm-up	Median-Early-Stopper	Patience-Early-Stopper	Sum
30	49.085	2.68	51.765
50	36.825	7.34	44.165
75	9.975	24.82	34.795
100	4.79	33.27	38.06

relationship with the warm-up period. This follows from the fact that a lighter early stop based on previous models lead to more space for the early stop mechanism based on the training patience to act.

From the data collected in Table 2 and corresponding performance drawn from Fig. 2, we can analyze the relations between the warm-up steps applied to the training and the performance outcomes of the optimal model in the HO process. All settings perform as well as or better than the run without early-stopping, confirming that early-stopping does not impact on the capacity of HO to find a good fitting model.

The setting with a minimal warm-up phase of 30 steps realizes significant resource savings, with nearly half of the epochs conserved. Notably, the model's performance under this condition doesn't compromise drastically as it recovers the key with approximately 200 traces, like the rest of the configurations. This suggests that even an aggressive early stop schedule can still result in optimal model performance.

Extending the warm-up phase to 50 or 75 steps results in a slight decrease in epoch savings. However, the performance of the model remains similar, requiring

around 200 traces for key recovery, which aligns with the performance exhibited by setting the warm up-value to 30.

Increasing the warm-up phase to 100 steps, or even having no early stop at all, maintains a consistent performance level, necessitating around 200 traces for key recovery. This finding reinforces the premise that strategic early stop doesn't compromise finding optimal configurations, but it might instead ensure efficient resource allocation and exploration-exploitation balance.

Table 3. Comparison with other papers of number of traces required to reach a GE of 0 for ASCADf, ID leakage Model

Attempt	Traces to reach GE = 0
[3]	1476
[30]	191
[23]	202
[29]	158
this study	230

5.2 Experiments for ASCADr

Table 4. Summary of the effect of different warm-up values on the average number of epochs saved with early stopping during training on ASCADr (in percentage of a complete training)

Warm-up	Median-Early-Stopper	Patience-Early-Stopper	Sum
30	33.3	14.44	44.74
50	35.63	18.09	53.72
75	24.0	38.12	62.12
100	0.0	39.11	39.11

The Table 4 summarizes the effects of different warm-up values on the 2 early stopping strategies for the ASCADr dataset.

When the warm-up value is set to 30, the Median Early Stopper contributes to 33.30% saving in epochs, while Patience-Early Stopping approximately 14.44%, reaching the total reduction of epochs to approximately 44.74%. Surprisingly, this figure is slightly lower than the reduction seen for higher warm up values (50 and 74), even though a lower warm up value would usually mean more aggressive pruning.

This counterintuitive result can be attributed to the fact that during this particular experiment, effective configurations were relatively quickly identified,

and the experiment showed consistent improvements as the models were executed. Considering the Median Early stop mechanism, this scenario naturally led to a decrease and delay of early stopping compared to other scenarios.

Although a lower warm up value generally implies more aggressive early stopping, it is essential to consider the dynamics of the optimization process and the quality of the configurations with the order in which they are found.

At a warm-up value of 50, Median Early Stopper models saved about 35.63% of the total epochs that could have been run, and the early stop saved approximately 18.09% of the epochs. The combined effect led to a reduction of around 53.72% in the number of epochs.

Increasing the warm-up value to 75, the impact of the Median Early Stopper savings decreases to 24.0%, but the effect of early stopping increases to 38.12%, leading to a total reduction of approximately 62.12%.

At a warm-up value of 100, the Median Early Stopper has no effect (0.0% reduction), while the impact of early stopping slightly increases to 39.11%, leading to the overall reduction in epochs to 39.11%.

It's important to explain that the elevated percentage of pruned epochs in the ASCADr experiments, compared to those in the ASCADf experiments, is primarily attributable to the baseline number of epochs set for each dataset. In the ASCADr experiments, the default number of epochs, was set to 300. In contrast, for the ASCADf experiments, this number was lower at 200.

This discrepancy means that, since the warm-up and early stopping values are kept constant across both sets of experiments, the resource-saving techniques, have a larger pool of epochs from which to cut in the ASCADr experiments. As a result, these techniques appear to prune a higher percentage of epochs in the ASCADr experiments compared to the ASCADf ones. This difference, however, is simply a reflection of the different default epoch settings for the two datasets, rather than indicating any inherent differences in the effectiveness of the early stopping strategies.

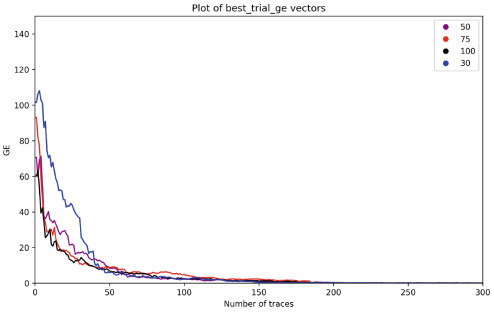

Fig. 3. Guessing entropy for the ASCADr experiments using warm up value of 30, 50, 75 and 100

Analyzing the outcomes of the best models' Guessing Entropy (GE) from various experiments of the ASCADr dataset, depicted in Fig. 3, reveals comparable results across the board for different warm-up values, specifically 30, 50, 75, and 100. Interestingly, the model from the experiment with a warm-up value of 30 exhibited slightly superior performance, reaching a GE of 0 with 170 traces. Meanwhile, the best models from experiments with warm-up values of 50, 75, and 100 also demonstrated commendable performance, requiring slightly more than 200 traces to achieve a GE of 0.

Once more, despite the amounts of computational resources saved across the different warm-up values, all experiments resulted in well performing models. These results suggest that the role of early stopping strategies can be effective in the HO process and can help to focus the process on more promising configurations and lead to better results.

Table 5. Comparison with other papers of number of traces required to reach a GE of 0 for ASCADr, ID leakage Model

Attempt	Traces to reach GE = 0
[19]	105
[23]	490
[29]	1568
this study	170

6 Conclusion and Future Work

In this paper, we investigated the role of HO in CNN-based SCA. We specifically showed how the integration of sampling and early-stopping strategies can help to find a good fitting model and save computing time and resources during the HO process.

Our results showed that the implementation of adaptive early-stopping techniques during the training phase could speed up the HO process without affecting performance. Across all warm-up values used in the experiments, the models consistently required around 200 traces for successful key recovery. This observation suggest that an efficient HO process, facilitated by early-stopping, does not compromise the effectiveness of the CNN models in SCA.

Furthermore, the use of TPE as a sampling algorithm and a GE-based objective value reduce the number of explored configurations during the HO process before finding a first good fitting model. These methods contributed to the effective selection and evaluation of hyperparameter configuration, leading to promising results shown in Tables 3 and 5.

In this work, we focused on the combination of TPE sampling and median early-stopping methods, but there exists many other sampling and early-stopping methods which could alleviate better the HO search. It would be interesting to

explore in more depth the different methods available in the machine learning literature and adapt it to the problem of SCA. This work also explored only the case of CNN-based deep learning, and only with a narrowed hyperparameter search space, adapted from the previous research for the given datasets we used for the experiments. When dealing with another dataset, a different search space or even different neural network architecture should be designed. The standardization of a common search space, taking into account width and depth limits of the architecture, could help to increase the confidence of security assessment of deep-learning based SCA.

References

1. Abadi, M., et al.: TensorFlow: large-scale machine learning on heterogeneous systems (2015). https://www.tensorflow.org/. software available from tensorflow.org
2. Akiba, T., Sano, S., Yanase, T., Ohta, T., Koyama, M.: Optuna: a next-generation hyperparameter optimization framework. In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 2623–2631 (2019)
3. Benadjila, R., Prouff, E., Strullu, R., Cagli, E., Dumas, C.: Deep learning for side-channel analysis and introduction to ASCAD database. J. Cryptogr. Eng. **10**(2), 163–188 (2020)
4. Bergstra, J., Bardenet, R., Bengio, Y., Kégl, B.: Algorithms for hyper-parameter optimization. In: Advances in Neural Information Processing Systems, vol. 24 (2011)
5. Brisfors, M., Forsmark, S.: DLSCA: a tool for deep learning side channel analysis. IACR Cryptol. ePrint Arch. 1071 (2019). https://eprint.iacr.org/2019/1071
6. Chari, S., Rao, J.R., Rohatgi, P.: Template attacks. In: Kaliski, B.S., Koç, C.K., Paar, C. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2002. Lecture Notes in Computer Science, vol. 2523, pp. 13–28. Springer, Berlin (2003). https://doi.org/10.1007/3-540-36400-5_3
7. Chollet, F., et al.: Keras (2015). https://keras.io
8. Hunter, J.D.: Matplotlib: a 2D graphics environment. Comput. Sci. Eng. **9**(3), 90–95 (2007). https://doi.org/10.1109/MCSE.2007.55
9. Jamieson, K., Talwalkar, A.: Non-stochastic best arm identification and hyperparameter optimization. In: Artificial Intelligence and Statistics, pp. 240–248. PMLR (2016)
10. Kocher, P., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M. (ed.) Advances in Cryptology — CRYPTO' 99. Lecture Notes in Computer Science, vol. 1666, pp. 388–397. Springer, Berlin (1999). https://doi.org/10.1007/3-540-48405-1_25
11. Kubota, T., Yoshida, K., Shiozaki, M., Fujino, T.: Deep learning side-channel attack against hardware implementations of AES. Microprocess. Microsyst. **87**, 103383 (2021). https://doi.org/10.1016/j.micpro.2020.103383
12. Li, C., et al.: Genetic algorithm based hyper-parameters optimization for transfer convolutional neural network. CoRR **abs/2103.03875** (2021). https://arxiv.org/abs/2103.03875
13. Li, L., et al.: A system for massively parallel hyperparameter tuning. Proc. Mach. Learn. Syst. **2**, 230–246 (2020)

14. Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., Talwalkar, A.: Hyperband: a novel bandit-based approach to hyperparameter optimization. J. Mach. Learn. Res. **18**(1), 6765–6816 (2017)
15. Maghrebi, H., Portigliatti, T., Prouff, E.: Breaking cryptographic implementations using deep learning techniques. In: Carlet, C., Hasan, M., Saraswat, V. (eds.) Security, Privacy, and Applied Cryptography Engineering. Lecture Notes in Computer Science(), vol. 10076, pp. 3–26. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-49445-6_1
16. Nakkiran, P., Kaplun, G., Bansal, Y., Yang, T., Barak, B., Sutskever, I.: Deep double descent: Where bigger models and more data hurt. J. Stat. Mech: Theory Exp. **2021**(12), 124003 (2021)
17. Paguada, S., Armendariz, I.: The Forgotten Hyperparameter: - introducing dilated convolution for boosting CNN-based side-channel attacks. In: Zhou, J., et al. (eds.) ACNS 2020. LNCS, vol. 12418, pp. 217–236. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-61638-0_13
18. Paguada, S., Batina, L., Buhan, I., Armendariz, I.: Being patient and persistent: optimizing an early stopping strategy for deep learning in profiled attacks. IEEE Trans. Inf. Forensics Secur. **17** (2022)
19. Perin, G., Chmielewski, Ł., Picek, S.: Strength in numbers: improving generalization with ensembles in machine learning-based profiled side-channel analysis. IACR Trans. Cryptogr. Hardw. Embedded Syst., 337–364 (2020)
20. Perin, G., Wu, L., Picek, S.: AISY - deep learning-based framework for side-channel analysis. Cryptology ePrint Archive, Report 2021/357 (2021). https://eprint.iacr.org/2021/357
21. Picek, S., Perin, G., Mariot, L., Wu, L., Batina, L.: SoK: deep learning-based physical side-channel analysis. ACM Comput. Surv. **55**(11), 1–35 (2023). https://doi.org/10.1145/3569577
22. Picek, S., Samiotis, I.P., Kim, J., Heuser, A., Bhasin, S., Legay, A.: On the performance of convolutional neural networks for side-channel analysis. In: Chattopadhyay, A., Rebeiro, C., Yarom, Y. (eds.) SPACE 2018. LNCS, vol. 11348, pp. 157–176. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-05072-6_10
23. Rijsdijk, J., Wu, L., Perin, G., Picek, S.: Reinforcement learning for hyperparameter tuning in deep learning-based side-channel analysis. IACR Trans. Cryptogr. Hardw. Embedded Syst., 677–707 (2021)
24. Rioja, U., Paguada, S., Batina, L., Armendariz, I.: The uncertainty of side-channel analysis: a way to leverage from heuristics. ACM J. Emerg. Technol. Comput. Syst. **17**(3), 1–27 (2021). https://doi.org/10.1145/3446997
25. Snoek, J., Larochelle, H., Adams, R.P.: Practical Bayesian optimization of machine learning algorithms. 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. 2960–2968 (2012). https://proceedings.neurips.cc/paper/2012/hash/05311655a15b75fab86956663e1819cd-Abstract.html
26. Wang, J., Xu, J., Wang, X.: Combination of hyperband and Bayesian optimization for hyperparameter optimization in deep learning. arXiv preprint: arXiv:1801.01596 (2018)
27. Wistuba, M., Schilling, N., Schmidt-Thieme, L.: Hyperparameter search space pruning – a new component for sequential model-based hyperparameter optimization. In: Appice, A., Rodrigues, P.P., Santos Costa, V., Gama, J., Jorge, A., Soares,

C. (eds.) ECML PKDD 2015. LNCS (LNAI), vol. 9285, pp. 104–119. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-23525-7_7
28. Wouters, L., Arribas, V., Gierlichs, B., Preneel, B.: Revisiting a methodology for efficient CNN architectures in profiling attacks. IACR Trans. Cryptogr. Hardw. Embedded Syst., 147–168 (2020)
29. Wu, L., Perin, G., Picek, S.: I choose you: automated hyperparameter tuning for deep learning-based side-channel analysis. IEEE Trans. Emerg. Top. Comput. (2022)
30. Zaid, G., Bossuet, L., Habrard, A., Venelli, A.: Methodology for efficient CNN architectures in profiling attacks. IACR Trans. Cryptogr. Hardw. Embedded Syst., 1–36 (2020)