A Literature Study into Hyperparameter Systematization for Deep Learning-based Side-Channel Analysis

Doreen Mulder

Delft University of Technology
d.mulder@student.tudelft.nl

June 21, 2020

Abstract

With the recent increase in computational power, deep learning is being applied in many different fields. Deep learning has produced promising results in the field of side-channel analysis. However, the algorithms used to construct deep neural networks remain black boxes, which makes it hard to fully employ the capabilities of attacks performed with these techniques.

This study explores methods to systematize the deep learning techniques used in profiled side-channel analysis. We do so by conducting a literature review of the state-of-the-art. Our observations show that while the process of choosing an architecture and hyperparameters have a great influence on the performance of an attack, not all authors thoroughly document this process. In lights of further improvement of the state-of-the-art, we also propose several promising techniques for hyperparameter optimization in side-channel analysis.

Keywords— machine learning, deep learning, side-channel attack, hyperparameter optimization

1 Introduction

Side-channel attacks exploit leakages in physical channels to extract secret information from devices [15]. They pose a threat to even the most robust cryptographic systems, as they are often easy to implement, have a low cost per device, and may be difficult to detect given their non-invasiveness [10].

With the recent growth in computational power, deep learning became more accessible. Nowadays, many fields benefit from these powerful techniques. Side-channel analysis benefits from deep learning too, as deep learning techniques have broken protected implementations where other, non deep learning based side-channel attack methods are much less effective [14].

Side-channel analysis profits from the implementation of deep learning techniques. However, the black-box nature of deep learning algorithms makes it hard to understand how they solve classification problems, and why the results are accurate. This makes it hard to decide exactly which model architecture and hyperparameters are necessary for optimal performance. Systematization of these techniques would take away the need to reveal the exact workings of black-box deep learning algorithms, and make these techniques more accessible for researchers who do not

have extensive knowledge about the algorithms they intend to use. This makes systematization interesting especially in the field of side-channel analysis, as the design of attacks - and thus, design of countermeasures - is a combined effort between both software and hardware experts.

Because the usage of deep learning in profiled side-channel attacks is relatively new, there has been little research done into the systematization of such attacks. This study explores methods to systematize deep learning techniques used in profiled side-channel analysis. To achieve this, we introduce deep learning and explain why systematization is challenging. Deep learning techniques used in side-channel analysis are summarized and investigated. We explore recent efforts in deep learning systematization, most notably hyperparameter optimization. We relate these works to the field of side-channel analysis, and discuss them.

In Section 2, we explain our methodology. Section 3 shows the basics of deep learning, why the hyperparameter search problem is challenging, and some common approaches to solve it. Section 4 describes profiled side-channel attacks in both their traditional and deep learning-based variants. Section 5 shows more advanced hyperparameter optimization techniques, and explains why these would be interesting to see in a side-channel analysis context. When applicable this section also shows known attempts of their implementations in side-channel analysis.

2 Methodology

In this study we explore the state-of-the-art and formulate new hypotheses regarding the systematization potential of deep learning techniques used to perform side channel analysis. For this, a large quantity of experiments is needed. To gather the appropriate amount of data needed to form insights on the topic, we will conduct a literature review of state-of-the-art research. "Breaking Cryptographic Implementations Using Deep Learning Techniques" by Maghrebi et al. [14] was the starting point of our research. We thoroughly analyzed it, and used it to gather keywords and to create a concept map. This concept map was based on the main topics covered in the study, such as side-channel analysis, machine learning, and deep learning. Smaller subtopics, such as the different architectures that were covered, were then connected to these main topics. Then, we looked into all subtopics, and decided whether to investigate them more or not, possibly expanding the concept map from there. The concept map can be found in Appendix A.

The initial data collection was performed by querying academic databases with the keywords we formulated, and searching for papers that cite the aforementioned study. This provided us with 137 records. It was important that we found works that provide insights into the automation of deep learning, deep learning in profiled side channel attacks, or a combination of the two. To query this, we read the abstracts of all the gathered papers, discarding papers that headed into different directions than what was useful for our literature study. After this selection was made, we read the introductions and conclusions of all remaining papers, and ended up with 55 papers that were to be read in-depth. The PRISMA analysis of this process can be found in Appendix B.

It became apparent that many deep learning related side-channel analysis works built on each other. However, little attention was paid in most works to explain the details of the used architecture and configuration, and the reasoning behind why these were chosen. To the best of our knowledge, there existed no well-documented system for selecting the right architecture configuration for the side-channel analysis problem. Therefore, the scope of our research branched out to hyperparameter optimization in deep learning, in hopes to find techniques that are also applicable in a side-channel analysis context.

After collecting potentially interesting studies, we assessed the quality of the works. We required the works to be peer reviewed. By consulting the Journal Quality List we gauged if the journals publishing these works are reputable. Special attention was paid to influential works in the field with a high citation count.

We made the decision against using statistical analysis since our sources perform experiments with different algorithms, datasets and hyperparameters. A qualitative synthesis is more appropriate in this case, as this allows us to gain a better insight in how the studies are related to one

another.

In our synthesis, the discussion, we present the most interesting finds in our sources. We point out where studies differ and where they overlap. We pay attention to the setup and reproducibility of experiments, and criticize them when needed. Experimental results are analyzed. We explicitly mention when snowballing was used to find related works, as this relationship might be important for the origins of ideas and conclusions of experiments.

To avoid confusion, it will be explicitly mentioned when conclusions and interpretations are provided by the authors of original papers, or if it is a conclusion based on research done in this study.

Some difficulties were encountered when synthesizing our findings, most of them stemming from the fact that part of the listed techniques have yet to be tried in a side-channel analysis context. This made it harder to gauge if these techniques would benefit the performance of side-channel analysis models.

3 Deep Learning

In this section, we will go over the basic concepts found in deep learning. We explain the importance of good hyperparameter selection and how this plays a role in the generalization of a model.

3.1 Basics

Deep learning is a subclass of machine learning. It utilizes artificial neural networks, which are loosely modeled after the structures in a biological brain. Like a brain, these neural networks consist of a collection of connected nodes, called neurons. Each neuron typically performs an operation on the input it receives, after which it forwards its output to the next neuron [22].

Neurons are organized in layers. The data samples are delivered in the first layer, also known as the input layer. Each interesting variable, referred to as feature, is represented by its own neuron in the input layer. The last layer is called the output layer. In a network trained to perform classification tasks, each class has its own neuron in the output layer. The output layer can therefore be used to provide the probability for each class that the input is labeled with [17]. The layers in between are hidden. Depending on the type of layer used, these layers perform various operations on the data that gets forwarded through them.

The function that a neuron computes on its input is called an activation function. We can describe such an activation function as $\sigma(w_1x_1 + ... + w_qx_q + b)$, where σ is used to denote the function, x_i the i^{th} input, w_i the weight of that input, q the number of inputs that neuron receives, and b the bias of the neuron [22]. The weight and bias are parameters of the neuron. In each training iteration, or epoch, these parameters are adjusted to optimize the network's performance.

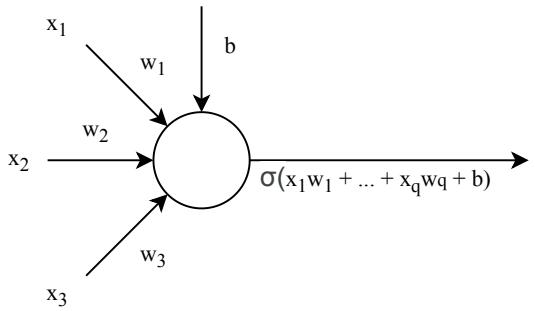


Figure 1: Visualization of a neuron, adapted from [22].

The learning algorithm used in deep learning is backpropagation. In each epoch, the backpropagation algorithm calculates the gradient of the loss function of the model. An optimizer algorithm then takes this as input and adjusts the parameters of the neurons in the network to minimize the loss function. In other words, when training a neural network, it learns by trying to minimize the error rate of the model.

3.2 Architectures and Hyperparameters

There are a number of deep learning architectures, with each of them having its own set of layer types and hyperparameters. Before training can begin, the architecture and hyperparameters have to be selected.

It is important to realize that hyperparameters are different from the model parameters. Hyperparameters can be seen as tuning options for the learning algorithm that are external to the training process, whereas the model parameters are the parameters that are being altered during the training process.

Hyperparameter tuning is an important problem, because the hyperparameter values used influence the performance of the resulting predictive model [20]. Using the most favorable hyperparameters ensures that the learning process performs optimally, which will result in a model of good quality. However, finding optimal hyperparameters for deep learning algorithms is challenging for a number of reasons.

- The deep learning algorithm is a black box, which makes it hard to try and verify hyperparameter combinations [21].
- The ideal hyperparameter configuration for a model depends on the dataset. For example, architectures and configurations that work well on image datasets are not guaranteed to perform well on non-image datasets [9].
- Hyperparameter search space grows exponentially relative to the number of hyperparameters [16].
- Hyperparameters can have values in different domains, for example real-valued (e.g. learning rate), integer-valued (number of layers), binary (use early stopping or not), or categorical (choice of optimizer algorithm). When the search space is discontinuous, objective evaluation may fail. The space may be noisy and non-deterministic, or flat, where many configurations have similar performance [9]. See Figure 2.
- Hyperparameters may also be conditional: a hyperparameter may only be relevant if another (combination of) hyperparameter(s) takes on a certain value [6].
- As there is no way to exhaustively try all possible hyperparameter configurations, there is no guarantee that the best hyperparameter values are obtained [22].

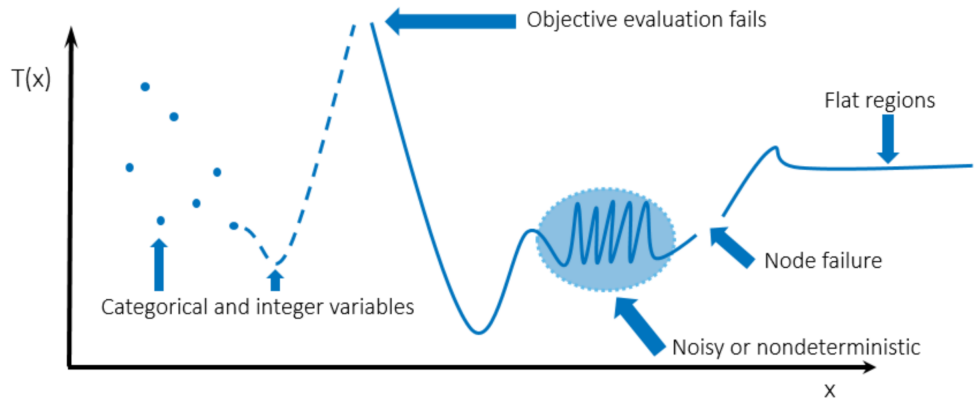


Figure 2: Challenges in hyperparameter tuning, from [9].

3.3 Generalization

When training a neural network, the goal is to achieve generalization in the resulting predictive model. This means that the model can perform correct classification on inputs it has not seen before: the neuron weights are tuned in such a way that it works on unseen input too.

Neural networks, especially ones with a large number of hyperparameters, are prone to overfitting, also known as memorization. In this case the neuron weights are tuned to the training data in such a precise way that it cannot correctly classify unseen input. There are several regularization techniques that combat overfitting. Some examples of this are adding or removing information, such as noise, penalizing complexity in a model, or stopping the training when overfitting is suspected to happen [22]. On the other hand, underfitting might happen when the model is trained too little. In this case, the model has not learned enough, resulting in an inability to capture the trend in the training data, which also results in not being able to make accurate predictions in unseen data.

It is useful to detect when underfitting becomes generalization, when generalization becomes overfitting, and if underfitting becomes overfitting while skipping the generalization part, since this can relax the search of some hyperparameters. Finding the end of the generalization phase and the start of the overfitting phase can be used as a guideline to decide when to stop training, in other words, the maximum number of epochs [17], whereas finding where underfitting goes over into generalization is useful to decide the minimum amount of epochs. Detecting if underfitting directly goes into overfitting, skipping generalization phase, can help determine the aggressiveness of the learning rate.

The learning rate influences the speed of the learning phase. A learning rate that is too low will take very long to converge to a local minimum, while a learning rate that is too high means that the weights are updated too heavily, resulting in an oscillating loss rate of the output [22], or the solution diverging completely from the optimal solution [9]. The aforementioned phenomena illustrate just how intertwined the performance of a deep neural network and its hyperparameters are.

3.4 Simple approaches for hyperparameter optimization

Common approaches to find suitable hyperparameters include grid search and random search. In a grid search, a set of values to be studied is selected for each hyperparameter of interest. These values are spaced out evenly. The model is then trained and assessed for the Cartesian products of the sets of values. It is a simple technique, however, a grid search is quite computationally intensive.

This is because the amount of possible combinations, and thus the amount of function evaluations, grows exponentially with regards to the number of hyperparameters in the model [9].

A random search also selects a set of values to study for each hyperparameter, but does so randomly, so these values are not spaced out evenly. Not all hyperparameters are equal in influence, as they may have little to no effect on a model for certain datasets. A random search is therefore less costly, as it does not evaluate all possible combinations of hyperparameter values [9]. Initializing hyperparameters through a random search is a useful method for beginning the search process, as it explores the entire configuration space and thus often finds settings with reasonable performance. However, it usually takes longer than guided search methods [6].

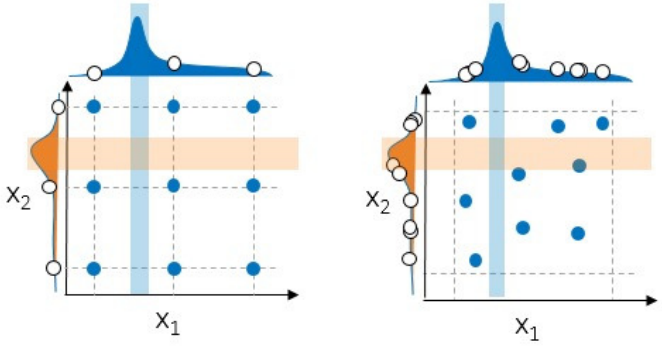


Figure 3: To the left, a grid search, to the right a random search. From [9].

A way to verify a hyperparameter configuration is to train the model with that configuration and then performing a validation phase. To do so, the training data is split into a training set and a validation set. Performances of the (partially) trained model are evaluated over the validation set, computing a validation error, after which the hyperparameters are updated accordingly. Once the model has been validated, and the hyperparameters are definitely set, the real test error is evaluated over the test set [2].

Recently, efforts have been made to automate the process of training machine learning models. These techniques are known as AutoML. Such techniques open doors for those who are interested in applying machine learning, but do not have the resources to study the technologies behind it in order to make decisions about the architecture and hyperparameters [6]. In Section 5, we will go deeper into automated hyperparameter optimization, and discuss a variety of methods that are used in this field.

4 Components of a Profiled Side-Channel Attack

A side-channel attack is an attack that makes use of leakages in physical channels of a system in order to uncover secret information. These leakages are also called traces, and include power consumption, electromagnetic radiation, and timing [21].

Side-channel attacks come in two different classes. In a non-profiled side-channel attack, the attacker only has access to the traces leaked from the target device. On the other hand, an attacker performing a profiled side-channel attack has access to an additional device, a clone of the target device. This clone device is used to discover dependencies between the obtained data and the target device behavior. A profile is then constructed based on these findings, which the attacker uses to perform key-recovery attacks on the target device. Profiled side-channel attacks are considered to be the most powerful type of side-channel attack [22].

Profiled side-channel analysis consists of two phases: the profiling phase and the attack phase. During the profiling phase, a profile of the targeted device is created. This is done with help of

the clone device, which has similar or identical characteristics to the targeted device. The attacker gathers a set of profiling traces from the clone device. Because they have access to this clone device, they know the exact in-and-outputs of the operations, including the sensitive values [21]. From there, the attacker can calculate the leakage model [22]. The leakage model describes the relationship between the leaked samples and the sensitive data [18]. It is necessary to know this relationship, because usually the traces that are leaked do not directly contain the sensitive value itself, and some processing is needed before the desired value can be discovered.

During the attack phase, the adversary collects traces from the target device, and classifies these using the build profile. The success of such an attack relies on the quality of the profile, which is based on the amount of profiling traces, the quality of those traces, and the similarity of the clone device and the target device [21].

A common method to evaluate the performance of an attack is to use the guessing entropy, the average number of key guesses required to recover the key. Given the total number of traces in an attack phase, T_{tot}, and the total amount of keys in the keyspace, N, we can calculate a key guessing vector $g = [g_0, g_1, ..., g_{|N-1|}]$. This vector is sorted in decreasing order of probability. g_i denotes the probability that that key candidate is the true key. To calculate each element g_i over a number of samples, we use this formula:

$$g_i = \sum_{j=1}^{T_{tot}} log(\hat{p}_{ij})$$

where $\hat{p}_{ij}$ is the estimated probability for key candidate i, given sample j. The guessing entropy is then the index of the true key's rank.

4.1 Deep Learning-based Side Channel Analysis

In a deep learning-based side-channel attack, the attacker trains a model that is able to classify the traces of sensitive values [21]. Deep-learning based side-channel attacks have been successful on protected implementations where other attack techniques are much less effective [22, 21]. These attacks are more streamlined as the pre-processing of data is no longer mandatory [13]. Maghrebi notes in multiple works that deep-learning based point-of-interest selection is at least as good as state of the art leakage assessment methods [13, 12].

In a deep learning-based side-channel attack, traces are used as input data. The output values that the attacker aims to extract from the trace are used as labels [8]. A variety of deep learning architectures have been used in deep learning-based side-channel analysis. Fully connected networks such as multi-layer perceptrons, or MLP, feature extraction networks such as convolutional neural networks, or CNN, sum up the architectures that have been experimented with the most. Time dependency networks like recurrent neural networks, or RNN, and long-short term memory networks, or LSTM, have been tried too, although on a lesser scale than the former two architectures. But even though these experiments have produced promising results, introducing deep learning to side-channel analysis comes with a new set of problems. Explanations on model generalization and how to improve this in a side-channel analysis context are lacking, leading to little understanding on how to confirm whether a trained model has achieved generalization for a given side-channel analysis problem [17]. A large factor in this is the hyperparameter search problem that we discussed in 3.2 and 3.4. When performing a deep learning-based side-channel attack, the design of the network architecture has a substantial influence on the attack's success. However, there is only limited knowledge about which architecture configurations are most appropriate for the side-channel analysis problem [21].

5 Optimization Methods

The hyperparameter search problem plays a role in any neural network. Solving the problem through automated hyperparameter optimization removes the need to uncover the effects that

each hyperparameter has on model performance. However, this approach introduces a new set of challenges. The hyperparameter configuration space is often complex and high-dimensional, with a mix of continuous, categorical and conditional hyperparameters. It is not always clear which of an algorithm's hyperparameters need to be optimized, and in which ranges. There usually is no access to the gradients of the loss functions with respect to the hyperparameters, and one cannot directly optimize for generalization performance as training datasets are of limited size [6]. In this chapter, we go over some promising methods for automated hyperparameter search, and when data is available we show how they perform in a side-channel analysis context.

5.1 Learning Curve Extrapolation

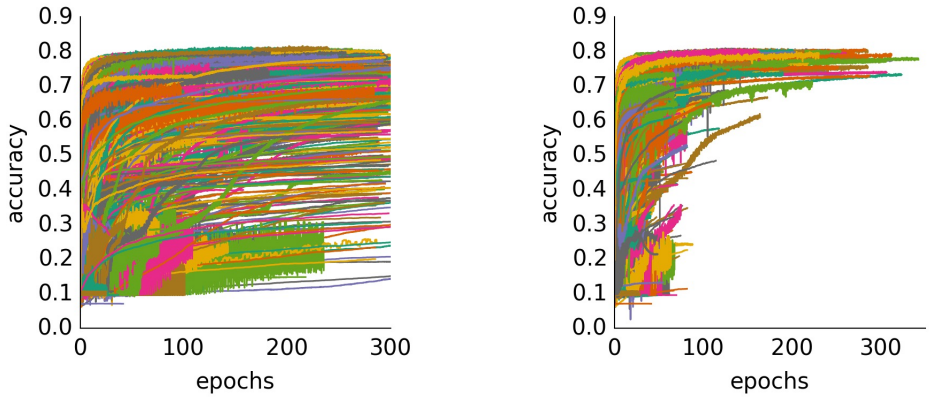
Rather than being a hyperparameter optimizer, a learning curve extrapolation algorithm may be used along hyperparameter optimizers in order to converge to the desired values faster. In learning curve extrapolation, the hyperparameter configuration on a certain algorithm is tested before the training process has completed. This can be done by only training the model for a few iterations before testing and deciding whether or not to stop the training process, by running it on a subset of features, or by only using few of the cross-validation folds [6]. Since the guessing entropy method is preferred over cross-validation in side-channel analysis, we will not focus on this concept.

5.1.1 Predictive Termination

When a human expert evaluates a poor hyperparameter configuration, this expert is able to quickly estimate whether the resulting network will perform poorly. They are then able to terminate the training process, saving time compared to completing the training process and only evaluating then [4]. Efforts have been made to automate this process. When using predictive termination, a learning curve model is used to extrapolate the learning curve of a partially trained model for a given hyperparameter configuration. The training process is then terminated if the predicted learning curve is not estimated to perform better than a certain threshold [6].

A possible predictive termination method was given in by the authors in [4]. For each hyperparameter configuration, a deep neural network is partially trained. Their algorithm keeps track of the best performance found so far. This best performance is initialized as $-\infty$, and is updated with the new best performance each run. The algorithm then predicts the probability that the neural network, after training for m intervals, will exceed the best performance. If this probability is above a certain threshold, the training will continue. Otherwise, training is terminated and the expected validation error is returned. Figure 4 shows the effects of this predictive termination algorithm on the learning curve of fully connected networks trained on the CIFAR-10 dataset. The figure shows how poor runs are terminated fairly early in the training process.

When looking into predictive termination in side-channel analysis, we were not able to find whether such a technique has been tried or not. This implies that either it has been tried and the effects have not been documented, or it has not been tried as of yet.



(a) Without predictive termination

(b) With predictive termination

Figure 4: Learning curves of fully connected networks trained on CIFAR-10. The plots contain all learning curves from 10 runs with the SMAC hyperparameter optimizer, as well as 10 runs with the TPE hyperparameter optimizer. From [4].

5.2 Bayesian Optimization

Because of their efficiency in objective function evaluations, Bayesian optimization and related techniques are currently a topic of interest [3]. Bayesian optimization has been proven to perform well in deep neural network tuning for image classification, speech recognition, and neural language modeling [6]. It is an iterative algorithm that uses a probabilistic surrogate model together with an acquisition function to decide which target function configuration to evaluate next. The surrogate model is used to predict the learning curve of the resulting model. The acquisition function then uses this model to acquire new candidate configurations. Bayesian optimization usually initializes the target function's parameters with a random search. Then, in each iteration, the surrogate model is fitted to all the observations made so far by the target function. The acquisition then uses the surrogate model to predict the performance of the candidate configurations to study next. Figure 5 shows Bayesian optimization on a one-dimensional function.

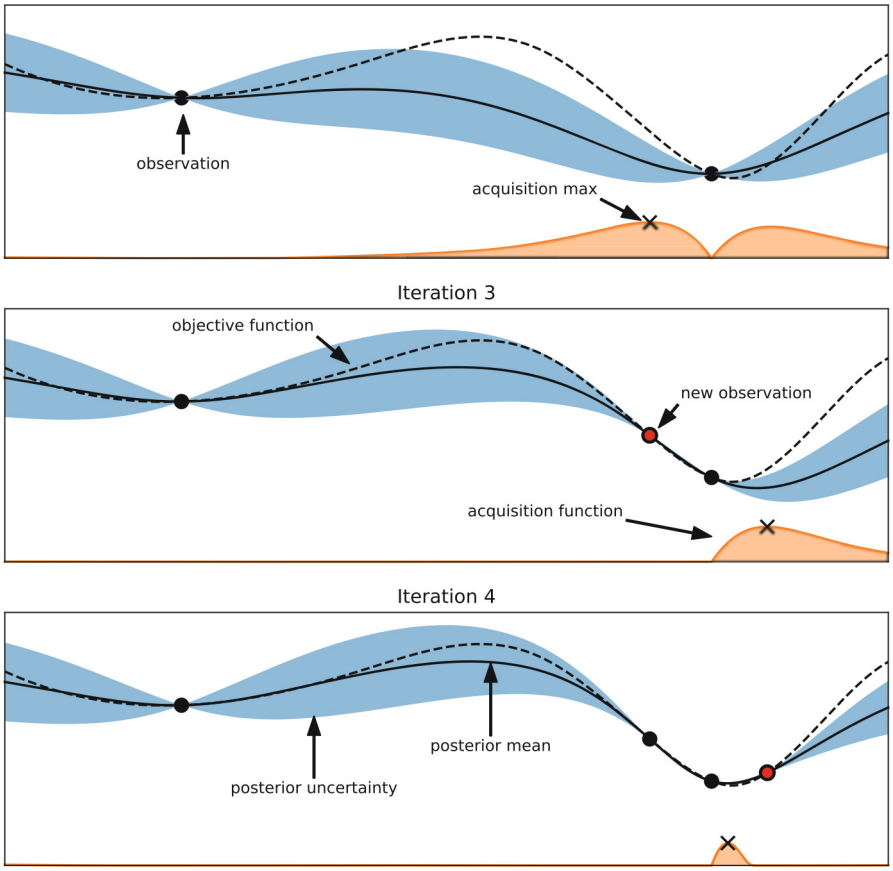


Figure 5: Bayesian optimization on a one-dimensional function. The black line shows the Gaussian process surrogate function, with the blue shape around it representing the uncertainty. The dashed line shows the actual objective function, which should be minimized. This is done by maximizing the acquisition function, which is represented by the orange curve. The acquisition function selects new values at points where the the predicted function value is relatively low and the uncertainty is relatively high. Eventually, this converges to a function minimum, as can be seen on the bottom image. From [6].

Compared to training and evaluating the target function, the acquisition function is cheap to compute. Bayesian optimization has also been shown to outperform manual search by human experts and random search on its own. However, to maximize its optimization performance, it is important to select an appropriate acquisition function. This makes it difficult for non-experts to utilize the potential of this method [16].

Traditionally, Bayesian optimization use Gaussian processes as acquisition function to model the target function. One downside of the standard Gaussian processes is their poor scalability, given that they scale cubically with the number of data points.

Again, there is a lack of information on if Bayesian optimization has been tried in a side-channel analysis context.

5.2.1 Freeze-Thaw Bayesian Optimization

Freeze-Thaw Bayesian optimization is a Bayesian optimization algorithm that maintains a set of frozen models throughout the process of optimization [20]. The algorithm uses information-theoretic criteria to determine which models to thaw and train further. It relies on the assumption that the training loss during when the model is being fitted roughly follows an exponential decay towards a final value that is unknown. This method is interesting because it is able to dynamically stop and restart experiments, reducing the amount of operations the acquisition function has to perform, which is beneficial in cases when computationally expensive algorithms are used, like the aforementioned Gaussian process.

5.3 Transfer Learning

In transfer learning, trained models are used as starting points for models with similar functions. The authors of [25] explain that neural networks perform exceptionally well in transfer learning. The source model can be used as a starting point for both the architecture and hyperparameters of the target model. This yields a fully initialized, pre-trained model which can then be fine-tuned on the data of the new training set. There have been experiments with RNNs modifying their own weights, and LSTMs being used as meta-learners to train MLPs. Unfortunately it appears that these methods have not been tried yet in the field of side-channel analysis.

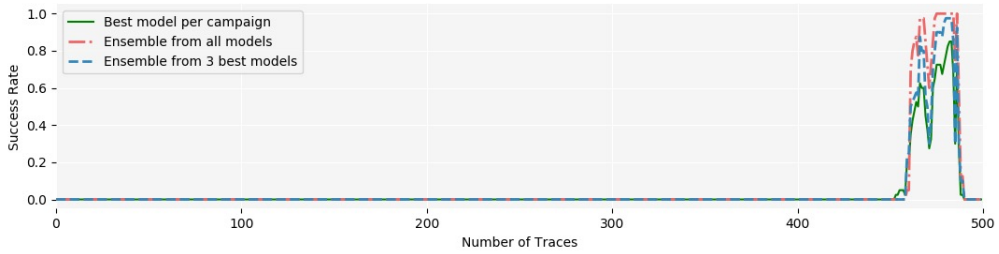
5.3.1 Ensemble Learning

Only choosing a single hyperparameter configuration can be wasteful when an optimization algorithm yields multiple usable configurations [6]. Ensemble learning is a type of transfer learning that combines multiple configurations into one. Configurations may be homogeneous, where hyperparameters are the same for the base models, or heterogeneous, where they are different. Combining models can be done in multiple ways. Wang et al. introduces three popular methods, namely Bagging, Boosting and Stacking [23]. Bagging is short for bootstrap aggregating. It uses different training data subsets (with replacement) from the entire training data to train different models on. These models are called base learners. To create the final ensemble, the base learners are combined by a majority vote. The Boosting method creates different base learners by repeatedly training a base learner to modified versions of the training dataset, resulting in a set of different base learners. These base learners are weighted according to their error. The final ensemble is then made using a linear combination of the base learners. Finally, Stacking uses a meta-learner to combine different base learners that were produced by differing algorithms.

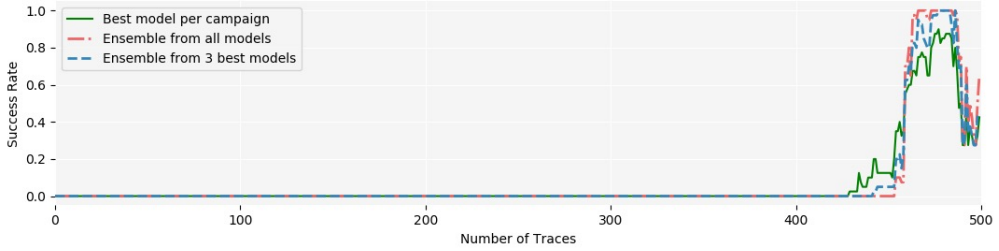
Ensemble models are expected to perform better classification compared to the performance of a single model. This stems from the idea that in statistics, combined measurements can lead to more reliable estimations, because the influence of outliers and random fluctuations in the single measurements is reduced. Ideally, ensembles are performed on systems with a large and diverse configuration space [6], with every model having a different set of hyperparameters in order to learn different features from the same training set [17].

Guilherme Perin conducted a study on ensemble learning in side-channel analysis and obtained promising results [17]. In this study, ensembles created with the bagging method produced more stable results for differing training set sizes, and tend to have a success rate that is at least as good as the best single model, as can be seen in Figures 6 and 7.

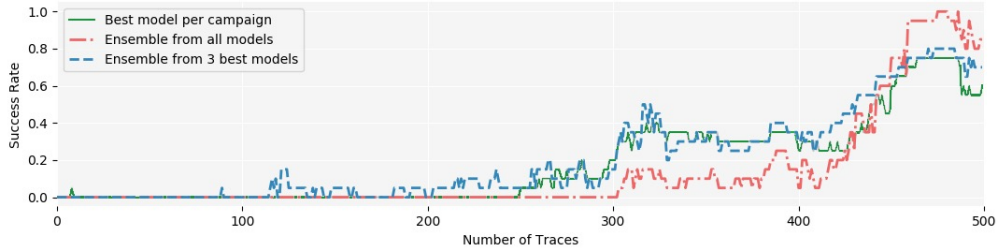
Although the use of ensembles relaxes the need to carefully select hyperparameters, it cannot replace hyperparameter search methods. Perin recommended that future works also look into the boosting and stacking methods to create ensembles.



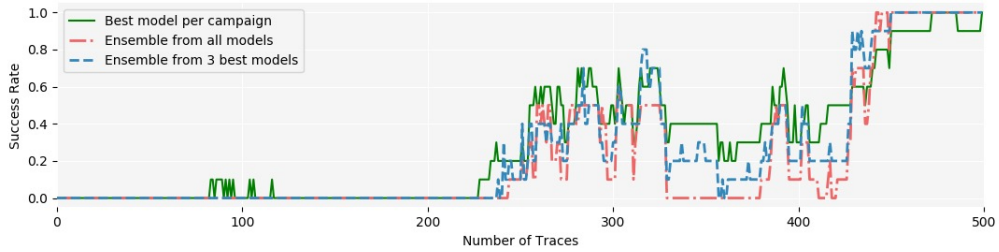
(a) 10,000 traces per single model



(b) 25,000 traces per single model

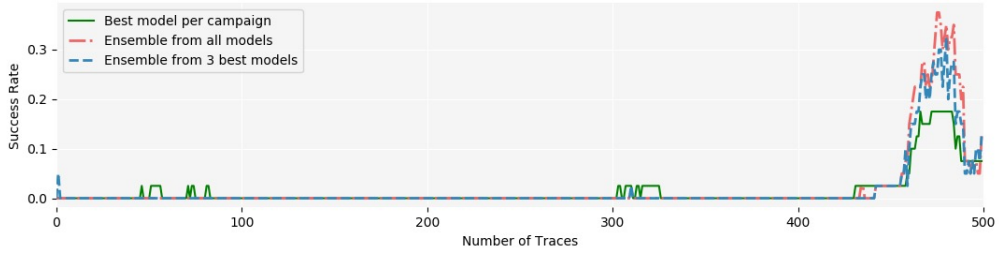


(c) 50,000 traces per single model

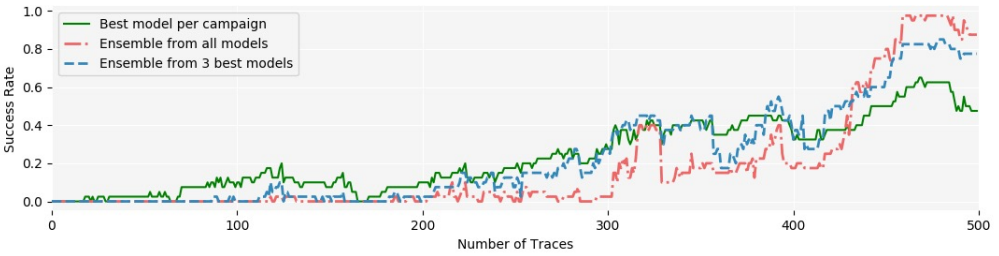


(d) 75,000 traces per single model

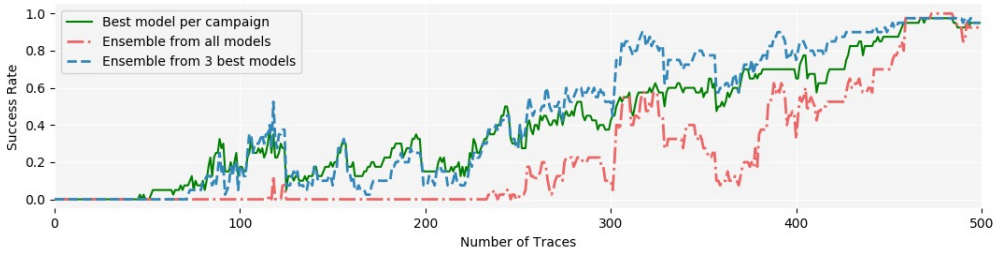
Figure 6: Perin's succes rates when averaging ensembles on the ASCAD database of masked AES traces. These experimental results are of the ensemble of 10 single homogeneous models. We can observe that often, the ensembles perform equally or better than the best single model. From [17].



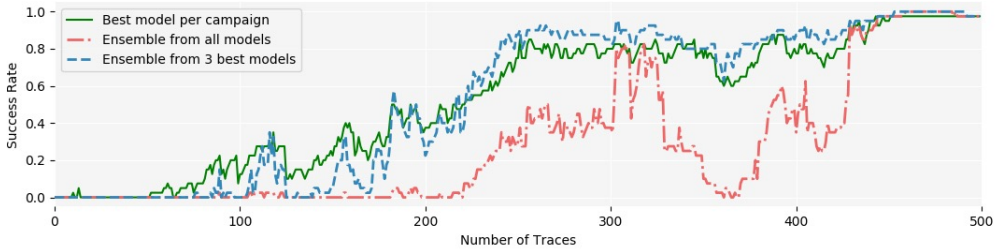
(a) 10,000 traces per single model



(b) 25,000 traces per single model



(c) 50,000 traces per single model



(d) 75,000 traces per single model

Figure 7: Perin's succes rates when averaging ensembles on the ASCAD database of masked AES traces. These experimental results are of the ensemble of 10 single heterogeneous models. Although the ensemble from all models performs poorly compared to the best single model, an ensemble constructed from the three best models performs equally or better than the best single model. From [17].

6 Discussion

One might wonder about the ethics of conducting research into simplification of the design process of malicious attacks. However, if the scientific community is not aware of how an attack is constructed, it becomes impossible to design appropriate countermeasures. We acknowledge the risks this type of research brings, but we encourage researchers to use this information responsibly.

The design of the used network architecture has a great influence on the performance of a deep learning-based side-channel attack. However, there is a surprising lack of information and documentation on which architectures are most appropriate for the side-channel analysis problem, and the reasoning behind this. This problem is not only present in side-channel analysis, but it is a reoccurring theme within the scientific community. Therefore we recommend authors in both side-channel analysis and other fields to take the time to document and clarify their architectural decisions. This will help researchers working on new works that are based on these studies decide which settings to use, and will create a better overall understanding of the influences these decisions have on the resulting models.

The hyperparameter search problem is a significant problem in deciding how to configure a deep neural network architecture. We emphasized the importance of doing research into automated hyperparameter optimization, as this removes the need to uncover the effects that each hyperparameter has on model performance. For this reason, automated hyperparameter configuration will also positively affect researchers who are interesting in using certain deep neural network algorithms, but do not have the extensive knowledge required to select appropriate hyperparameters for optimal model performance. This is especially useful in the field of side-channel analysis, as this field requires a combined effort of software and hardware experts to construct attacks and countermeasures. In this study, we discussed various techniques for automated hyperparameter optimization, and when the information was available, showed how they perform in a side-channel analysis context.

One of the discussed techniques is learning curve extrapolation. The hyperparameter configuration on a certain algorithm is tested before the training process has completed, after which a decision is made on whether to terminate training or not. This saves time compared to completing the training process and only evaluating then. To mimic the human capability of quickly determining whether a model is going to perform well or not with a certain hyperparameter configuration, an predictive termination algorithm may be used [4]. Since knowledge is still limited about which architecture configurations are most appropriate for the side-channel analysis problem [21], this makes it possible to quickly test and gauge the quality many configurations. Therefore we recommend future work in the side-channel analysis field to look into these techniques.

We also discussed Bayesian optimization, an iterative algorithm that uses a probabilistic model together with an acquisition function to decide which hyperparameter configurations to evaluate next. It comes with its own set of challenges to maximize the optimization performance, which makes it difficult for non-experts to utilize the potential of this method. To the best of our knowledge, no data has been documented on the performance of Bayesian optimization in a side-channel analysis context. However, we think it is a very promising method, as results show that the method performs well in deep neural network tuning for image classification, speech recognition, and neural language modeling. These fields all deal with wildly different datatypes, and therefore we think that it is worth it to spend effort into finding out if Bayesian optimization performs well in side-channel analysis too.

If this is the case, we recommend looking into Freeze-Thaw Bayesian optimization as well. It is an interesting method because it is able to dynamically stop and restart experiments, reducing the amount of operations that need to be performed, which is beneficial in cases where computationally expensive algorithms are used. If Bayesian optimization performs well in side-channel analysis, we definitely recommend exploring Freeze-Thaw Bayesian optimization as well.

When multiple different deep learning models with the same purpose exist, transfer learning might be interesting to look into. In transfer learning, trained models are used as starting points for models with similar functions. We think that using these techniques in a side-channel analysis

context is interesting, as there exists a number of trained machine learning models in this field already. We found one study that tried a type of transfer learning in side-channel analysis, ensemble learning, a type of transfer learning that combines multiple configurations into one. They perform better on average. This is based on the idea that in statistics, combined measurements can lead to more reliable estimations, because the influence of outliers and random fluctuations in the single measurements is reduced. These techniques are also interesting because they allow researchers to build upon already established knowledge, possibly discovering patterns in hyperparameter values when multiple models are combined into one. The Bagging ensembling method has already been tried in side-channel analysis, and produced promising results [17]. The author recommended future work to also try the Boosting and Stacking methods. Based on these results, we agree that it is interesting to also try the Boosting and Stacking methods in future works.

7 Conclusions

Although the performance of a deep learning-based side-channel attack stands or falls with the quality of its neural network, there is a lack of documentation on how to create a qualitative model. Appropriate hyperparameter selection contributes to good performance. For this reason we recommend researchers to include the reasoning and methods behind their hyperparameter selection in their works. When this becomes standard in the field, it will be easier to build upon each other's works, as it will be known which methods and techniques work well in the field, and which ones don't.

In this literature review we discussed several techniques that are worth looking into when we consider the side-channel analysis problem. One of the discussed techniques is learning curve extrapolation. Methods like early termination make it possible to quickly test and gauge many hyperparameter configurations, which is beneficial since knowledge is still limited about which architecture configurations are most appropriate for applying to side-channel analysis.

Bayesian optimization performs well in deep neural network tuning for image classification, speech recognition, and neural language modeling. Because it has been so widely applied, it is interesting to apply this to side-channel analysis too. Unfortunately, it comes with its own set of challenges, making it difficult for non-experts to utilize the potential of this method. We consider it beneficial to keep looking into Bayesian optimization and try to find a relationship between good optimization performance and the effect its hyperparameters has on the resulting side-channel analysis model. For future work we recommend to explore this technique in the field of side-channel analysis.

The transfer learning technique uses already established models to create new ones. Ensemble learning is a type of transfer learning that combines multiple configurations into one. The Bagging ensembling method has already been tried in side-channel analysis, and produced promising results [17]. It will be interesting to see if this is also the case with the Boosting and Stacking methods.

During the literature study we encountered many other hyperparameter optimization techniques, such as Bayesian Optimization Hyperband (BOHB) [25, 5, 1], Bayesian and multi-armed bandit optimization [19], Gradient based optimization through Reversible Learning [11], Multi-node Evolutionary Neural Networks for Deep Learning (MENNDL) [24], Nelder-Mead [16] and HORD [7]. While the time constraints on our work prevented us from exploring all these different options, we would like to mention that these works exist and are worth investigating in the future.

8 Acknowledgements

With my finished Bachelor's thesis in front of me, I would like to take opportunity to thank all the people who have supported me.

First of all, I would like express my sincerest thanks to my supervisors, Stjepan Picek and Marina Krcek, for their guidance and assistance through every stage of this project. I am grateful for my peers Djoshua Moonen, Johannes Ijpma and Jim Kok for the discussions we had and the help we provided to each other. And even though he worked on a different graduation project, I would like to thank Rohan Sobha as well for the discussions we had and the new insights he provided. While the current pandemic made working together harder than what we had envisioned, I could not have wished for a better peer group.

I would like to thank my family and friends, with special thanks to my mother Indira van Beijnum, my sister Roseanna Mulder, and my boyfriend Erik Blokland. Four years ago I took the plunge and started my studies, and throughout these years you have never stopped believing in me. Now at the end, you helped me overcome what seemed to be a nearly impossible and overwhelming project.

Finally, I would like to thank my pet birds, whose cheerful chirps never failed to brighten up my mood.

Doreen Mulder
June 21st, 2020

A Concept Map

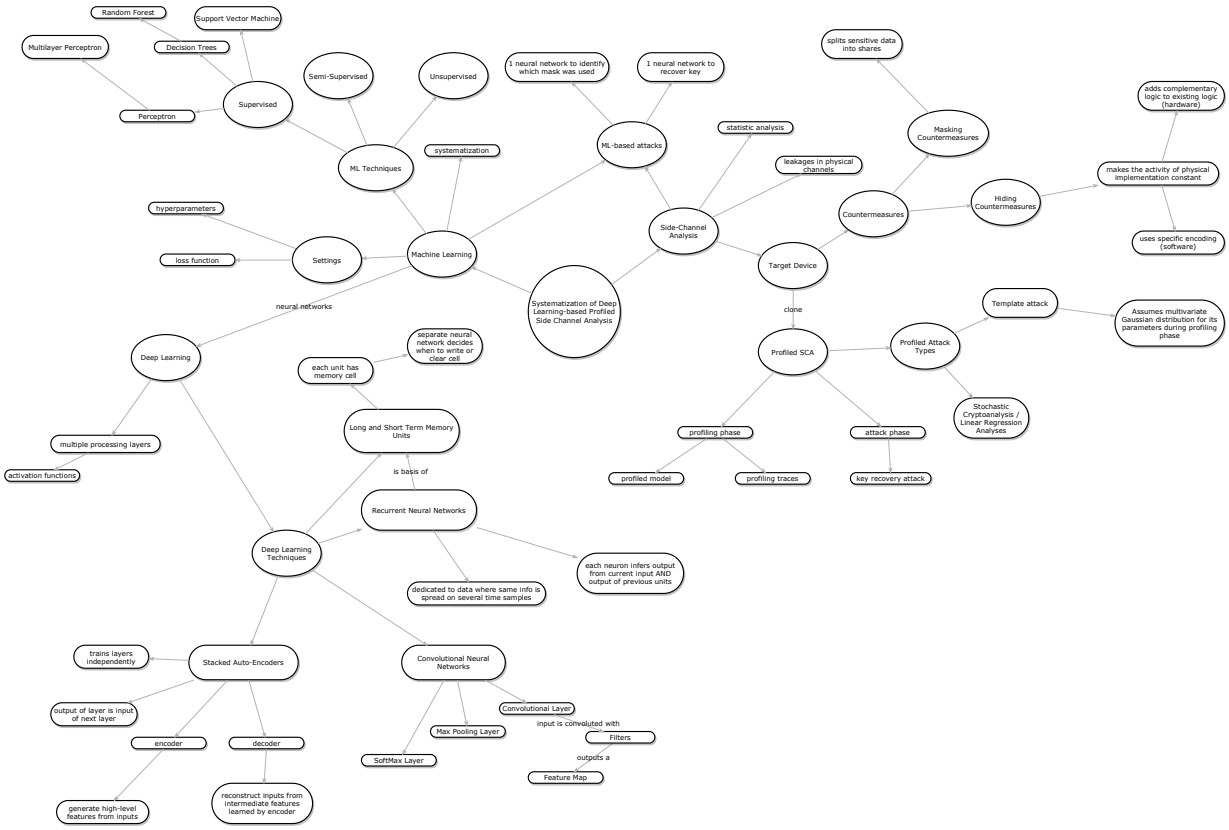


Figure 8: Concept map used to gather keywords.

B PRISMA Analysis

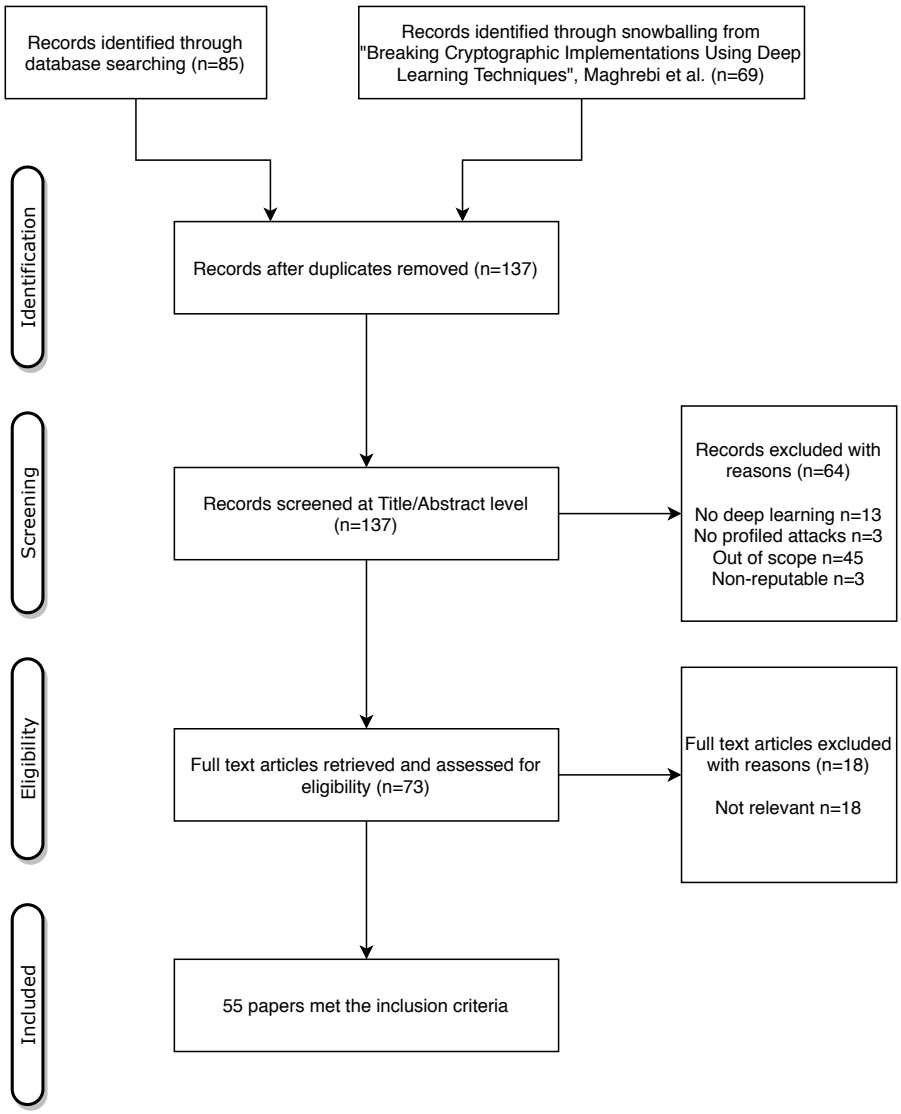


Figure 9: PRISMA analysis of the data collection process.

References

[1] Bertrand, H., Ardon, R., Perrot, M., and Bloch, I. Hyperparameter optimization of deep neural networks: Combining hyperband with bayesian model selection. In *Conférence sur l'Apprentissage Automatique* (2017).

[2] Cagli, E. *Feature Extraction for Side-Channel Attacks*. PhD thesis, Sorbonne Université, 12 2018.

[3] Claesen, M., and De Moor, B. Hyperparameter search in machine learning. *arXiv preprint arXiv:1502.02127* (2015).

[4] Domhan, T., Springenberg, J. T., and Hutter, F. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *Twenty-Fourth International Joint Conference on Artificial Intelligence* (2015).

[5] Falkner, S., Klein, A., and Hutter, F. Bohb: Robust and efficient hyperparameter optimization at scale. *arXiv preprint arXiv:1807.01774* (2018).

[6] Hutter, F., Kotthoff, L., and Vanschoren, J. *Automated Machine Learning*. Springer, 2019.

[7] Ilievski, I., Akhtar, T., Feng, J., and Shoemaker, C. A. Efficient hyperparameter optimization for deep learning algorithms using deterministic rbf surrogates. In *Thirty-First AAAI Conference on Artificial Intelligence* (2017).

[8] Jin, S., Kim, S., Kim, H., and Hong, S. Recent advances in deep learning-based side-channel analysis. *ETRI Journal 42*, 2 (2020), 292–304.

[9] Koch, P., Wujek, B., Golovidov, O., and Gardner, S. Automated hyperparameter tuning for effective machine learning. In *Proceedings of the SAS Global Forum 2017 Conference* (2017).

[10] Kocher, P., Jaffe, J., and Jun, B. Differential power analysis. In *Advances in Cryptology — CRYPTO' 99. CRYPTO 1999. Lecture Notes in Computer Science*, M. Wiener, Ed., vol. 1666. Springer, Berlin, Heidelberg, 1999, pp. 288–397.

[11] Maclaurin, D., Duvenaud, D., and Adams, R. Gradient-based hyperparameter optimization through reversible learning. In *International Conference on Machine Learning* (2015), pp. 2113–2122.

[12] Maghrebi, H. Deep learning based side channel attacks in practice. Tech. rep., IACR Cryptology ePrint Archive 2019, 578, 2019.

[13] Maghrebi, H. Deep learning based side-channel attack: a new profiling methodology based on multi-label classification. Cryptology ePrint Archive, Report 2020/436, 2020. `https://eprint.iacr.org/2020/436`.

[14] Maghrebi, H., Portigliatti, T., and Prouff, E. Breaking cryptographic implementations using deep learning techniques. In *Security, Privacy, and Applied Cryptography Engineering: 6th International Conference* (12 2016), pp. 3–26.

[15] Mangard, S., Oswald, E., and Popp, T. *Power Analysis Attacks: Revealing the Secrets of Smart Cards*. Springer, Graz, Austria, 2007.

[16] Ozaki, Y., Yano, M., and Onishi, M. Effective hyperparameter optimization using nelder-mead method in deep learning. *IPSJ Transactions on Computer Vision and Applications 9*, 1 (2017), 20.

[17] Perin, G. Deep learning model generalization in side-channel analysis analysing class probabilities , metrics and ensembles.

[18] Ramezanpour, K., Ampadu, P., and Diehl, W. Scaul: Power side-channel analysis with unsupervised learning. *arXiv preprint arXiv:2001.05951* (2020).

[19] Swearingen, T., Drevo, W., Cyphers, B., Cuesta-Infante, A., Ross, A., and Veeramachaneni, K. Atm: A distributed, collaborative, scalable system for automated machine learning. In *2017 IEEE International Conference on Big Data (Big Data)* (2017), IEEE, pp. 151–162.

[20] Swersky, K., Snoek, J., and Adams, R. Freeze-thaw bayesian optimization. arxiv 2014. *arXiv preprint arXiv:1406.3896* (2014).

[21] Tubbing, R. An analysis of deep learning based profiled side-channel attacks. Master's thesis, Delft University of Technology, 2019.

[22] van der Valk, D. Towards understanding of deep learning in profiled side-channel analysis: Similarity of predictors measured and explained. Master's thesis, Delft University of Technology, 2019.

[23] Wang, G., Hao, J., Ma, J., and Jiang, H. A comparative assessment of ensemble learning for credit scoring. *Expert systems with applications 38*, 1 (2011), 223–230.

[24] Young, S. R., Rose, D. C., Karnowski, T. P., Lim, S.-H., and Patton, R. M. Optimizing deep learning hyper-parameters through an evolutionary algorithm. In *Proceedings of the Workshop on Machine Learning in High-Performance Computing Environments* (2015), pp. 1–5.

[25] Zela, A., Klein, A., Falkner, S., and Hutter, F. Towards automated deep learning: Efficient joint neural architecture and hyperparameter search. *arXiv preprint arXiv:1807.06906* (2018).