Neural Network Model Assessment for Side-Channel Analysis

Guilherme Perin, Baris Ege and Lukasz Chmielewski

Riscure BV, The Netherlands, surname@riscure.com

Abstract. Leakage assessment of cryptographic implementations with side-channel analysis relies on two important assumptions: leakage model and the number of side-channel traces. In the context of profiled side-channel attacks, having these assumptions correctly defined is a sufficient first step to evaluate the security of a crypto implementation with template attacks. This method assumes that the features (leakages or points of interest) follow a univariate or multi-variate Gaussian distribution for the estimation of the probability density function. When trained machine learning or neural network models are employed as classifiers for profiled attacks, a third assumption must be taken into account that it the correctness of the trained model or learning parameters. It was already proved that convolutional neural networks have advantages for side-channel analysis like bypassing trace misalignments and defeating first-order masking countermeasures in software implementations. However, if this trained model is incorrect and the test classification accuracy is close to random guessing, the correctness of the two first assumptions (number of traces and leakage model) will be insufficient and the security of the target under evaluation can be overestimated. This could lead to wrong conclusions in leakage certifications. One solution to verify if the trained model is acceptable relies on the identifying of input features that the neural network considers as points of interest. In this paper, we implement the assessment of neural network models by using the proposed *backward propagation path* method. Our method is employed during the profiling phase as a tool to verify what the neural network is learning from side-channel traces and to support the optimization of hyper-parameters. The method is tested against masked AES implementation. One of the main results highlights the importance of *L2 regularization* for the automated points of interest selection from a neural network.

Keywords: Side-Channel Analysis · Neural Networks · Model Assessment · Hyper-Parameters Optimization

1 Introduction

Embedded systems containing cryptographic implementations are known to be vulnerable to side-channel analysis. These attack methods explore the leakage of information that is conveyed through unintended side-channels where usually power consumption, electromagnetic emanation and time are considered. Profiled (e.g., template attacks[1], linear regression[2], machine learning[3][4]) and non-profiled attacks (e.g., DPA[5], CPA[6], MIA[7], clustering[8]) are considered to assess the levels of security of a cryptographic implementation.

With the progress that has been made in the field of big data analysis and neural networks, deep learning has been considered as a powerful attack method for side-channel analysis. In many different domains where large amounts of data are available, like natural language processing, computer vision, medical data analysis, etc., a deep neural network

is being able to achieve high levels of accuracy in classification and recognition tasks. Side-channel analysis also falls into the big data scenario, because tenths of millions of side-channel traces need to be processed in a (conventional) hardware or software security assessment.

The application of multiple-layer perceptron (MLP) for side-channel analysis has been proposed in [4][9]. Recently, researchers have demonstrated that deep neural networks are able to break protected software AES implementations [10][11][12]. The elimination of jitter and other misalignment effects is not a straight-forward task and these effects directly affect the performance of first-order or profiled side-channel attacks. When these effects are strongly present in side-channel traces, it can lead to the (precipitate) conclusion that the target is sufficiently robust against high-potential side-channel attacks, as the leakage assessment phase is not detecting important leakages. Convolutional neural networks (CNNs) can deal with misaligned traces, as shown in [11]. Their results demonstrate that CNNs overcome misalignment and jitter-based countermeasures with the application of data augmentation techniques. Besides the confirmed advantages, the advent of deep neural networks for profiled side-channel analysis leaves some open questions that still need to be properly addressed. in [13], the authors investigate how machine learning metrics should be interpreted in order to estimate the success rate of an attack. The conclusion from that paper is that accuracy, precision or recall are not able to replace success rate or guessing entropy.

Another topic that needs more clarification is how to interpret the trained model parameters in order to assume that this model can be used to estimate the leakage of information from the target under evaluation. In [14], the authors proposed to use input gradient activations in order to identify what are the samples in side-channel traces that a neural network presents major sensitivity with respect to the cost function. A similar analysis has been presented in [12]. The authors of [15] investigate a layer-wise propagation method which is based on the relevance of a neuron with respect to the next layer in the network. Therefore, the methods presented in [14], [12] and [15] can all be used to verify which input features have more impact in the neural network output probabilities.

Main contributions: In this paper, we call the techniques presented in [14], [12] and [15] as *model assessment* and we propose an extension of the layer-wise propagation method called *backward propagation path*. For every side-channel trace, the proposed method is able to point out the sample(s) with more contribution to the output layer activation decision. We test the proposed method against unprotected and protected AES software implementations (simulations and ASCAD database). The obtained results highlight that *neural networks are able to recognize second-order leakages*, which is sufficient to defeat the masked AES implementation proposed in [16]. Next, we provide experiments to demonstrate how model assessment can be used to optimize neural networks hyper-parameters. In particular, we show how small variations in the regularization L2 directly affect the leakage detection by the neural network. An important conclusion from our experiments is that as soon as the L2 regularization hyper-parameter is wrongly set (i.e, too small or too large), the rest of hyper-parameters are very difficult to be found with respect to the precision of points of interest identification by the neural network. Therefore, the identification of a good L2 value has strong impact in the network's capability for leakage detection. We also enforce that different model assessment techniques may provide different results and that the usage of different methods can lead to more accurate conclusions about the trained model.

The rest of the paper is organized as follows. Section 2 provides definitions and terminologies that are referred along the paper. Section 3 provides an overview about neural networks and side-channel analysis. In particular, in this section, we address the main open points with respect to the application of neural networks in side-channel analysis. We start by separating the main open problems into four main groups and then

we propose a discussion on them. For one of them, model assessment, this paper focuses its main contributions. Discussion about model assessment techniques for neural networks is provided in Section 4. Section 5 describes the proposed *backward propagation path* method. Experiment results for different AES implementations are provided in Section 6. Section 7 provides an investigation on hyper-parameters optimization from model assessment. Finally, conclusions are presented in Section 8.

2 Background

This section provides definitions of terms and notations that will referred along the text. Following, a description of profiled side-channel analysis is explained in 2.2.

2.1 Terms and Notations

In this paper, we will refer to $x_{i,t}$ as being a sample t ($0 \leq t < S$) in a trace i belonging to a trace set T_N containing N traces. Thus, $T(i) = [x_{i,0}, ..., x_{i,S-1}]$ refers to the $i-$th trace from the trace set. Every observation x from a side-channel trace can be seen as a leakage observation.

A leakage model defines a function that quantifies an internal variable being processed during the execution of a cryptographic algorithm. In this paper, all the tests are performed on AES-128 and we consider Hamming weight of S_{box} output, $\ell_i = HW(S_{box}(p_i \oplus k_i))$, as the leakage model. Therefore, the term ℓ_i defines the label that is attributed to each side-channel trace i.

The term a_j^l defines the activation value for a neuron j in a layer l. The value $\omega_{i,j}^{l-1}$ defines the weight connection from a neuron i in layer $l-1$ to a neuron j in layer l. For this case, the layer can be input, dense or output. The activation of a neuron takes into consideration an activation function $f(\cdot)$, its weight connections $\omega_{i,j}^{l-1}$ and bias b^{l-1} from layer $l-1$. The term $len(l)$ defines number of neurons in a layer l.

For convolutional layers, the term $a_{f_{q,j}}^l$ defines a feature in the feature map that is obtained as a result from a convolution of a set of filter weights with a input activations. A filter in a convolutional layer l is defined as $\omega_u^{r,l} = [\omega_0^{r,l}, \cdots, \omega_{ks-1}^{r,l}]$, where ks is the kernel size or filter length. We must consider two scenarios:

- Input Layer $\rightarrow$ Convolutional Layer: every filter weight $\omega_u^{r,l}$ is associated to a filter index r and its position u inside the filter vector. The lenght of the filter is given by the kernel size ks and number of filters is given by R;

- Convolutional Layer $\rightarrow$ Convolutional Layer: every filter weight $\omega_{q,u}^{r,l}$ is associated to a filter index r in layer $l-1$, a filter index q in layer l and its position (q,u) inside the filter array. This filter array is represented as $\Omega_{R \times ks}$, where R is the number of filters in layer $l-1$.

2.2 Profiled Side-Channel Analysis

Standard non-profiled approaches like CPA or DPA are limited by the necessity of a leakage model and a selection function. They are also univariate nature: each time sample is being attacked independently. Therefore, these univariate attacks might be not sufficient if the assumed leakage model and selection function are too far from reality or a leakage is exploitable only if several time samples are combined together.

In these cases so-called profiled side-channel analysis attacks, in particular, template attacks[1], linear regression[2], and machine learning[3][4], might be successful due to their multi-variate nature (i.e., they can combine multiple samples for the same leakage).

In general, the profiled attacks aim to answer the following question: "when possessing a device, which is under a full control of the attacker, how to make the best use of it to attack another device for which the key is unknown?". The assumption is that both devices differ only in the keys that they use. Besides that they are identical with respect to hardware and software.

Usually a profiled attack consists of two phases. The first phase, called *learning* or *profiling*, requires a *profiling* device, which is under full control of the attacker, on which they can characterize leakage of the device. The result of the learning phase can be seen as a set of templates where each template represents a piece of information about a class of population. These templates together let the adversary to distinguish one class apart from another in the second phase. In the second phase, called *attacking*, the attacker matches the result of the learning phase to the leakage coming from the attacked device (which stores an unknown secret)[1].

Different profiled attacks have different requirements with respect to points of interest selection and modelling the leakage, which is usually performed using probability density function (*pdf*). In particular, with respect to deep learning attack, neural networks learn the *pdf* function from the traces and this process is significantly affected by the choice of network hyper-parameters. Therefore, to implement the proper neural network to correctly learn from traces, we need to correctly define hyper-parameters and for that we need to assess the neural network model.

3 Neural networks and side-channel attacks

Research and developments of side-channel attacks allows the academic community and chip manufacturers to provide high-end secure solutions for embedded systems with respect to unintended side-channel leakages. Recent publications have considered software AES implementations to provide evidences that deep neural networks poses serious threats for certain targets even when the collected trace sets contains jitter or misalignments or when the AES is protected with first-order masking schemes. In the context of public-key cryptography, [17] provides security evaluation of protected RSA implementation by using neural networks. Therefore, secure cryptographic implementations (both, hardware accelerators and software crypto libraries) are now threatened by the possibility that deep neural networks can bypass countermeasures that were sufficient to repel the possibility of leakage detection with state-of-art side-channel attack methods.

When neural networks are considered for profiled side-channel attacks, the good results were obtained for open-source trace sets (ASCAD [16] and DPAContest). However, there are several open points that the research community has to address in order to make neural networks as a solid and trustful methodology for side-channel analysis evaluations. This is important not only to understand what are sufficient countermeasures for the life cycle of the product, but also to improve security based on the information that neural network models can provide.

Well-known machine learning metrics are also employed for neural networks: accuracy, recall, precision and cost function. In [13], the authors provide a comprehensive evaluation of machine learning metrics for side-channel attacks, where the main focus is the imbalance class problem for leakage models defined for symmetric-key algorithms (Hamming distance and Hamming weight). The main conclusion is that side-channel attacks cannot only rely on machine learning metrics as they do not succeed in replacing success rate or guessing entropy metrics. When the analysis in conducted against a symmetric algorithm (e.g., AES), the key ranking after test phase is conducted in a DPA-like approach with the processing of many traces with a fixed key. For that, the trained model provides output class probabilities for each processed test trace. These class probabilities from the output

[1] Usually for the sake of simplicity, only a single device is used during security evaluations.

network layer are then employed in the key ranking. However, from our experience, for some cases even when test accuracy is close to a random guessing, the key ranking can still return the correct key candidate. This does not refer to the fact that metrics are not important, but indicates that output probabilities in fact do contain relevant information for side-channel attacks.

For a given training data set, we want to configure a neural network that can achieve good enough generalization in its optimization problem. The network parameters (weight connections and biased values in network layers) are learned based on the configurable hyper-parameters (e.g., number of layers, elements per layer, activation functions, back-propagation algorithm, etc.). Solutions for this difficult problem include genetic algorithm, simulated annealing or Bayesian optimization. This hyper-parameters search algorithms are very time-consuming and, to the best of our knowledge, there are no publications with practical results about the application of optimization algorithms in neural networks for side-channel attacks. Model assessment techniques that verify what and how the neural network is learning its parameters can be a good alternative as an information gathering to optimize the hyper-parameters. Some publications [14][15][12] have proposed different techniques to visualize what the network identifies as main input features (points of interest) in order to make its decisions. In [14], the authors also consider input activation gradient as a metric for the selection of points of interest for template attacks. In [12] the sensitivity analysis (with is also based on input activation gradients) is employed as a distinguisher for non-profiled attacks. It is interesting to note that both mentioned references consider the same database (ASCAD) for the experiments, however successful results are obtained against a masked AES implementation with different neural networks configurations. While in [14] the experiments are conducted for deep convolutional neural networks with up to 7 convolutional layers, in [12] the authors define MLPs or CNNs with up to 2 dense layers or 2 convolutional plus pooling layers, respectively. This opens the question related to the importance of correctly choosing the hyper-parameters and what is the importance of each one of them in the leakage detection. Additionally, we would like to find some conclusions on what hyper-parameters have more or less influence in the identification of points of interests from side-channel traces.

In this paper, we provide a contribution on model assessment for side-channel analysis by proposing an alternative solution based on the identification of backward (layer-wise) propagation paths in neural networks. We provide proof-of-concept on unprotected and protected AES implementations and show how model assessment can give interesting insights about the effect of hyper-parameters in the leakage detection.

4 Model Assessment Techniques for Side-Channel Analysis

Deep neural networks can implement highly complex models. For every trace set collected from a specific target-of-evaluation, there will be different sets of hyper-parameters that define several *local minima* and a *global minima* in the *landscape* (defined by the error/loss/cost function). It is generally assumed that as soon as the learning model finds a local minima, the neural network would be actually learning from data-dependent leakages. Ideally, the network should also discard non-leaking samples from its decision process. Of course, this task is accompanied by the correct selection of a leakage model. The task of making the model to converge to one of the local minima or, ideally, to the global minima, is difficult due to the large amount of possible combinations of hyper-parameters. However, training metrics (accuracy, loss function) could indicate such performances during training and validation. The remaining question is related to what exactly the network has learned from side-channel traces.

If the work relies on identifying what are the optimal values for two or three of the hyper-parameters, a grid search could cover all possibilities. However, if more hyper-parameters need to be found, the usage of an optimization algorithms would be necessary, such as evolutionary algorithms. However, optimization algorithms can lead to unrealistic situations where the convergence to a good set of hyper-parameters is unfeasible within a reasonable time. When evaluating the security of cryptographic implementations, leakage certifications or even security assessments on the manufacturer's side, the available time can not afford for a very extensive hyper-parameters search.

To clearly understand how a trained model is actually performing the distinction between data-dependent samples from non-leaking samples, we need to appeal for model assessment techniques. In neural networks, it also refers to the identification of input features that are decisive for the classification task. Some works [18] give some directions in showing how information is conveyed through the hidden layers, and others try to explain how the features in the input data affect the activations in hidden layers [19]. Techniques that highlight what convolution layers learn from images, allows the localization of objects in input data. Some techniques, based on occlusion sensitivity [19, 20], are useful in the direction of identifying how the convolution layers treat separate classes from the input data. For side-channel analysis, the works presented in [14], [15] and [12] provide good coverage of model assessment techniques for side-channel analysis. Next, we provide a brief description of these techniques.

4.1 Occlusion techniques

The author of [19] proposes different visualization techniques for object detection for computing vision. One of the techniques is called *occlusion* and it refers to input (in case, two-dimensional data) manipulation in order to identify the features that contributes more for a neural network decision.

For side-channel analysis, this technique can be used to identify the samples associated to neural network activations after training stops. The associated training traces can be processed, one-by-one, with parts of traces occluded. The occlusion is nothing more than zeroing a sample interval in the entire training set and asking the neural network to perform the classification. By observing the metrics after every occlusion, it is possible to identify the intervals that, once occluded, reduces the classification accuracy of the trained model or even affect key ranking results.

The main possible limitation of occlusion techniques for side-channel analysis is related to leakage order. As already stated in this paper, neural networks implement very complex models and are possibly able to learn from high-order leakages. If the occluded interval suppress some of the leakages, the classification accuracy can still be high enough because the network is detection leakages from different trace samples.

4.2 Input Activation Gradients

In [14], the authors proposed the visualization of input activation gradients as a technique to characterize the automated selection of points of interests by deep neural networks. The result is a vector of gradients that are computed by the back propagation algorithm as the derivative of the cost function with respect to the input activation. Basically, when the neural network is trained in a correct way, the gradient visualization (GV) method indicates which samples have more influence in the network predictions. In [12], the authors proposed the same solution, defined as *sensitivity analysis*, and they used it in the context of non-profiled side-channel analysis where input activation gradients are basically used as distinguishers.

4.3 Layer-wise Relevance Propagation

The authors of [15] compared different techniques for the assessment of trained neural networks. One of the methods, called layer-wise relevance propagation (LRP) implements a recursive process of the activations in the neural network until the input layer is reached. The method computes the relevance of a neuron in layer $l-1$ with respect to a neuron l. In this paper, we propose a variant of the LRP where the relevance is excluded from the recursion. As will be described in the next section, the main idea of out proposed model assessment method is to identify the neurons with greater influence in the network's decision by suggesting what would happen if that neuron wouldn't exist in the network or, in other words, if that neuron would have activation zero for a particular input side-channel trace.

5 Backward Propagation Path

Once a neural network is trained, the training traces are again provided to the network and, one by one, a neuron which is activated with the higher value in the output layer (after *softmax* activation function) is taken into account. This single neuron represents one of the classes defined in the input data set. For side-channel analysis, it is well-known that the number of classes is derived from the selected leakage model and the number of neurons in the output layer is equal to this number of classes.

A neuron in the output layer is fully connected to all the neurons in the previous layer through a set of connection weights. Furthermore, the activation value for each neuron depends on all their previous weights connections and activation values of the previous layer. Every connection between two neurons is defined by a weight value, which remains unchanged after training stops as well as the bias value for every layer. The values of the input data (or, input trace) are processed by all the neuron connections, producing different activation values for all the neurons inside the network. The main goal of training a neural network for side-channel analysis is to learn the parameters (weights and biases) in a way that the network can distinguish input trace samples that represent data dependent leakage. If the neural network is able to create activation paths based on input samples that are actually *points of interest* (i.e., samples that contains data dependent information) we then assume that our model is trained in a way that it is learning the actual leakages from side-channel traces.

We are interested in identifying exactly the sample(s) from each input trace that the network is selecting as the most important one(s) for its classification task. By doing this assessment, we are able to observe whether the trained model is actually selecting leaking samples (points of interest). The proposed method can be applied to training data as well as validation data. When the method is applied to training data, it is possible to visualize how the network is fitting the data. By processing validation (or test) data with backward propagation path method it is possible to visualize the generalization capabilities as an additional information to the conventional metrics (accuracy, recall or loss function).

5.1 Method Description

This section provide details about the proposed method. The adopted solution is a layer-wise process. Neural networks can present different types of layer structures. Therefore, we describe all the possible cases for MLPs and CNNs. For CNNs, the only requirement is that the network is always structured with the following configuration: input layer, convolution + pooling layers, dense layers and output layer.

Dense to dense layer: Let us represent a neuron by its activation value a_j^l, where j is the index of the neuron in layer l. We are interested in identifying what neuron in layer $l-1$ has more importance for the activation value a_j^l. This procedure is done by

analyzing what multiplication result $\omega_{i,j}^{l-1}.a_i^{l-1}$ provides the major contribution in the resulting activation value $a_j^l = f\big(\sum_{i=0}^{len(l-1)} \omega_{i,j}^{l-1}.a_i^{l-1} + b^{l-1}\big)$, where $\omega_{i,j}^{l-1}$ is the weight connection between a neuron i in layer $l-1$ and a neuron j in layer l, b^{l-1} is the bias of layer $l-1$ and $f(\cdot)$ is the activation function. Therefore, for $i = [0, ..., len(l-1)]$, we compute:

$$a_j^l(m) = f\Big(\sum_{i=0, i\neq m}^{len(l-1)} \omega_{i,j}^{l-1}.a_i^{l-1} + b^{l-1}\Big) \tag{1}$$

The index of the neuron in layer $l-1$ that contributes more for the activation a_j^l in layer l is then given by:

$$z^{l-1} = indexOf\Big(\underset{0\leq m<len(l-1)}{\operatorname{argmin}} \Big(a_j^l - a_j^l(m)\Big)\Big) \tag{2}$$

This procedure can also be understood as a backward propagation path identification. Fig. 1 illustrates an example for what an activation path looks like. In this illustrative example, it was identified that $a_1^2\omega_{1,0}^2$ is the most contributory term for the activation a_0^3 in layer 3, which is, from (2), $z^2 = 1$.

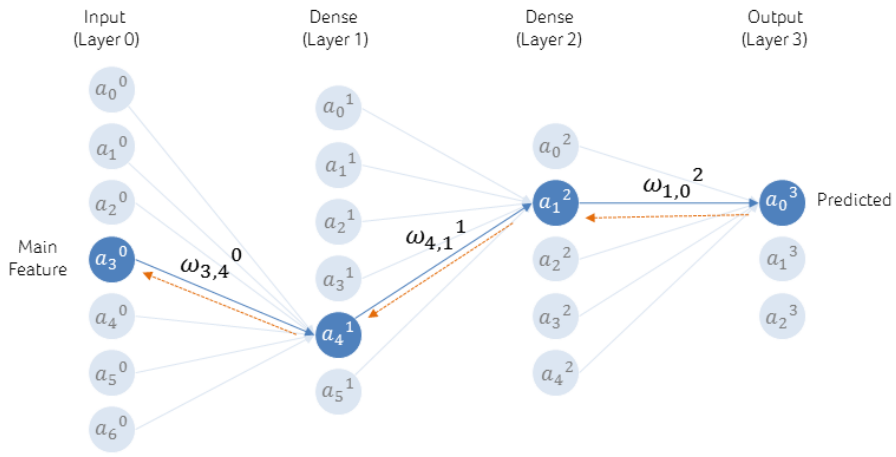

Figure 1: Illustration of selected backward propagation path for MLPs.

Convolution to dense layer: So far, the backward propagation path is described for multiple-layer perceptrons. The method can also be extended to convolutional neural networks. For MPLs, the task of identifying the backward propagation path is quite simple, because we only need to find the more contributory term $\omega_{i,j}^{l-1}a_i^{l-1}$ for the activation of a neuron when there are two consecutive dense layers. However, for CNNs, we have to identify the more contributory terms for connections from pooling to dense layers and from convolutional to convolutional layers.

In the connection from a pooling to a dense layer, the procedure is quite simple, because the output of a pooling layer is flattened and connected to this dense layer. The only difference refers to the identification of the feature in the pooled feature map and what filter is responsible for this feature result. Fig. 2 illustrates this procedure. The goal is to identify which feature (and which filter) from the feature map of layer 2 contributes more for the activation a_4^3 in dense layer 3. The example determines that the term $a_{f_{1,2}}^2\omega_{f_{1,2}}^2$ contributes more to the activation a_4^3. This can be done by following Equations (1) and

(2). Next, we select what features from the feature map of the convolution layer 1 were considered to generate this feature $a^2_{f_{1,2}}$. As we can see, two features ($a^1_{f_{1,4}}$, $a^1_{f_{1,5}}$) are then identified in the backward propagation path when the convolution layer 1 is reached.

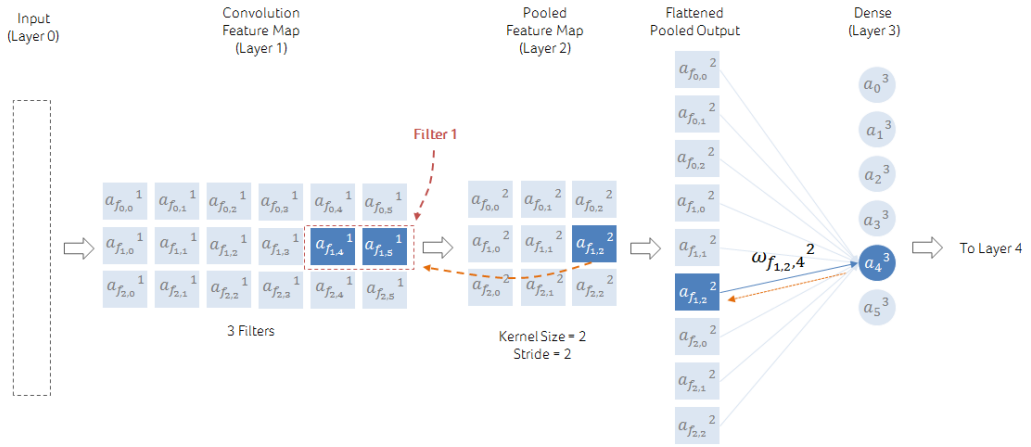

Figure 2: Illustration of selected backward propagation path for the connection between pooling and dense layer.

Convolution to convolution layer: Let us assume that we have a CNN where the first layers are, in this order: input (layer 0), convolutional (layer 1), convolutional (layer 2) and pooling (layer 3). Fig. 3 illustrates this example by showing the output feature maps of each of this layers. At this point, we assume that we already know what are the features in layers 2 and 3 the contribute more for the activation of the neuron in next layer (layer 4) that, in our case, is a dense layer (e.g., see Fig. 2). For two consecutive convolutional layers in a network, we are interest in identifying what are the features in layer 1 (and corresponding filter) that contribute more for the generation of specific features in layer 2 (and corresponding filter). In the illustrative example from Fig. 3, the features $a^1_{f_{2,4}}$ and $a^1_{f_{2,6}}$ (related to filter 2 in layer 1) are the ones that contribute more for the generation of features $a^2_{f_{4,4}}$ and $a^2_{f_{4,5}}$ (related to filter 4 in layer 2), respectively.

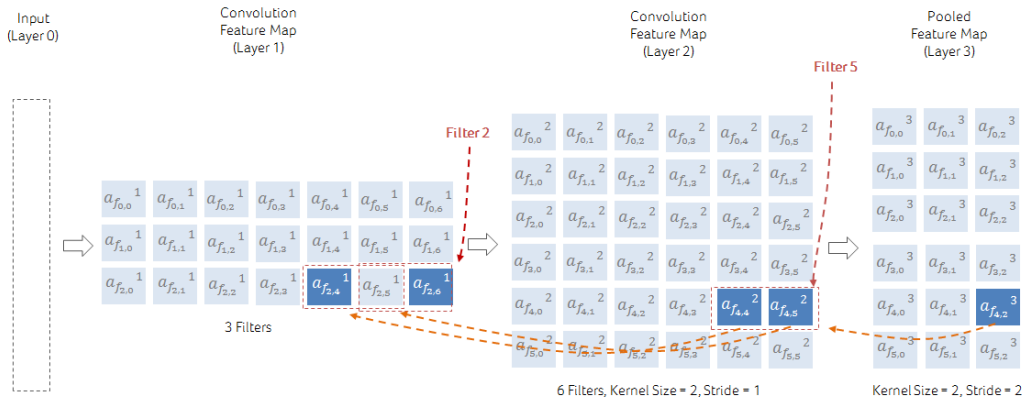

Figure 3: Illustration of selected backward propagation path for the connection between convolution layers.

Considering ks and st as the kernel size and stride, respectively, of the convolutional layer l. Also, we consider r and q as being the filter indexes of layers $l-1$ and l, respectively,

and R and Q are the number of filters in layers $l-1$ and l, respectively. A filter weight is given by $\omega_{r,u}^{q,l}$. Therefore, a feature $a_{f_{q,j}}$ in the feature map of layer l is computed as follows:

$$a_{f_{q,j}}^{l} = f\Big(\gamma_{j}^{q,l} + b_{r}^{l-1}\Big) \tag{3}$$

and $\gamma_{j}^{q,l}$ is the convolution result for the calculation of the feature $a_{f_{q,j}}$ in layer $l-1$ with respect to the filter q in this layer $l-1$ and it is given by:

$$\gamma_{j}^{q,l} = A_{1\times ks} \times \Omega_{R\times ks} = \left(\begin{bmatrix} a_{f_{r,j.st}}^{l-1} & \cdots & a_{f_{r,j.st+ks-1}}^{l-1} \end{bmatrix} \begin{bmatrix} \omega_{0,0}^{q,l} & \cdots & \omega_{0,ks-1}^{q,l} \\ \omega_{1,0}^{q,l} & \cdots & \omega_{1,ks-1}^{q,l} \\ \vdots & \cdots & \ddots \\ \omega_{R-1,0}^{q,l} & \cdots & \omega_{R-1,ks-1}^{q,l} \end{bmatrix} \right) \tag{4}$$

Eq. 4 basically describes how all the filters $\Omega_{R\times ks}$ in layer l combines information from every feature map row output from layer $l-1$. Following the illustrative example from Fig. 3, equation (4) becomes:

$$\gamma_{4}^{4,2} = A_{1\times 2} \times \Omega_{3\times 2} = \left(\begin{bmatrix} a_{f_{2,4}}^{1} & a_{f_{2,5}}^{1} \end{bmatrix} \begin{bmatrix} \omega_{0,0}^{4,2} & \omega_{0,1}^{4,2} \\ \omega_{1,0}^{4,2} & \omega_{1,1}^{4,2} \\ \omega_{2,0}^{4,2} & \omega_{2,1}^{4,2} \end{bmatrix} \right) \tag{5}$$

and:

$$\gamma_{5}^{4,2} = A_{1\times 2} \times \Omega_{3\times 2} = \left(\begin{bmatrix} a_{f_{2,5}}^{1} & a_{f_{2,6}}^{1} \end{bmatrix} \begin{bmatrix} \omega_{0,0}^{4,2} & \omega_{0,1}^{4,2} \\ \omega_{1,0}^{4,2} & \omega_{1,1}^{4,2} \\ \omega_{2,0}^{4,2} & \omega_{2,1}^{4,2} \end{bmatrix} \right) \tag{6}$$

We are interested in computing the terms $a_{f_{r,j}}^{l-1}(r,u)$ (for $r = 0, ..., R-1$ and $u = 0, ..., ks$) by setting the element $\omega_{r,u}^{q,l}$ of Ω to zero. Finally, the feature $a_{f_{r,j}}^{l-1}$ from layer $l-1$ that contributes more for the activation of feature $a_{f_{q,j}}^{l}$ in layer l is given by:

$$a_{f_{r,j}}^{l-1} = indexOf\Big(\underset{0\le u<ks, 0\le r<R}{\operatorname{argmin}} \Big\{ a_{f_{r,j}}^{l-1} - a_{f_{r,j}}^{l-1}(r,u) \Big\} \Big) \tag{7}$$

As a result, we can identify which feature u in filter r from layer $l-1$ contributes more for the activation of a feature in layer l.

Convolution to input layer: after the more contributory feature in convolution layer 1 is identified, the method identify what is the input feature (from input side-channel trace) that contributes more for the activation of this feature in layer 1.

Considering ks and st as the kernel size and stride, respectively, of the convolutional layer 1. Also, r and R are the filter index and the number of filters, respectively, of this layers 1. A filter weight is given by $\omega_{u}^{r,l}$. Therefore, a feature $a_{f_{r,j}}$ in the feature map of layer l with respect to filter r is computed as follows:

$$a_{f_{r,j}}^{1} = f\Big(\sum_{i=j.st}^{j.st+ks-1} a_{f_{r,i}}^{0} \omega_{i-j.st}^{r,1} + b_{r}^{1} \Big) \tag{8}$$

where $f(\cdot)$ is the convolution layer activation function and b_{r}^{1} is layer bias for the convolution filter r. We are interested in computing the following values:

$$a_{f_{r,j}}^{1}(m) = f\Big(\sum_{i=j.st, i\neq m}^{j.st+ks-1} a_{f_{r,i}}^{0} \omega_{i-j.st}^{r,1} + b_{r}^{1} \Big) \tag{9}$$

for all $r \in [0, ..., R-1]$. The input feature that contributes more for the activation of feature $a^0_{f_{r,j}}$ is given by:

$$a^0_{f_{r,j}} = indexOf\Big(\underset{0 \leq i < ks,}{\operatorname{argmin}} \Big\{ a^0_{f_r,j} - a^0_{f_r,j}(m) \Big\}\Big) \tag{10}$$

Therefore, following this layer-wise method it is possible to identify the input feature (from input layer) that has more contribution in the activation of a specific neuron in the output layer. Next, a proof-of-concept is provided with respect to first and second-order leakages from simulated AES traces. After, the proposed method is applied to the ASCAD database and it is demonstrated how this model assessment mechanism can be used to improve leakage detection or the automatic selection of points of interests from side-channel traces.

6 Experiments

In this section, we provode some experimental results to validate the proposed method. We start by defining the neural network configuration that is considered for the experiments. Ee provide results for simulated AES traces in order to have a proof-of-concept. Next, we show the application of the proposed method to the ASCAD database.

6.1 Neural network configuration

For the experiments provided in this section, we defined a convolutional neural network with the following hyper-parameters:

- Convolutional layer: 10 filters, kernel size (ks) (or filter length) of 10, stride (st) of 5 and *Relu* activation function;
- Dense Layer: 40 neurons, *Relu* activation function;
- Dense Layer: 40 neurons, *Relu* activation function;
- Output layer: 9 neurons, *Softmax* activation function.

The mini-batch size is 128, learning rate is set to 0.01 and the chosen optimizer is *nesterovs* with *momentum* equal to 0.9. The only considered regularization technique is $L2 = 0.05$. For each experiment, the network parameters (weights) are initialized with *xavier* method. No learning rate decay is considered during training. We consider two scenarios. First, we evaluate the method against simulated AES traces. Next, we apply the method to ASCAD database.

6.2 Method Validation on Simulated AES Traces

In order to validate the proposed backward propagation path method, this section provides results for simulated AES traces. The main idea is to demonstrate that this model assessment can be used to verify how a neural network identifies first-order leakages or either combines input features for side-channel analysis in a second-order context.

6.2.1 Simulated AES traces

As a proof-of-concept we first provide results for simulated traces. The simulated side-channel traces simply contain random samples values $s \in [0, ..., 8]$, except for the sample 68 which contains exactly the same value of the target leakage (e.g., the Hamming weight of the $S_{box}(k_i \oplus p_i)$). Fig. 4 shows results for the application of the backward propagation

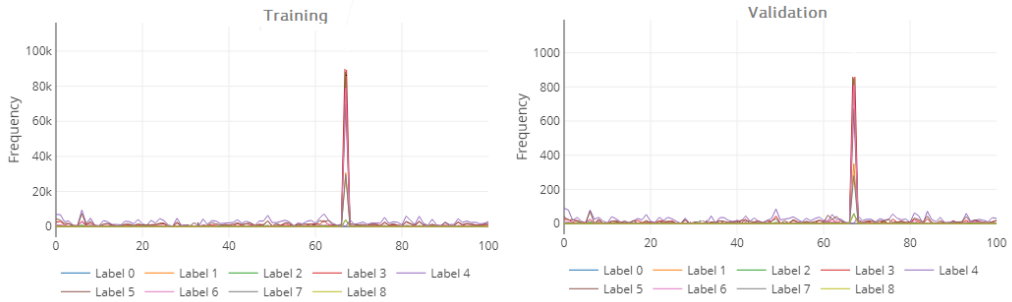

Figure 4: backward propagation path results for simulated AES traces. Result for training (left) and validation (right) traces.

path on training and validation traces. As the figure indicates, sample 68 is recognized as the most contributory sample for the output activations. For this experiment, 100,000 traces were generated, where 99,000 were used for training and 1,000 for validation.

The y-axis in Fig. 4 indicates the frequency that an input sample is selected as the most important feature for the output network activation in the processed trace set.

6.2.2 Masked Simulated AES Traces

The generation of masked AES traces follows the implementation described in [16], which is the masked AES implementation for the ASCAD database. Every simulated traces contain 100 samples. Each trace contain the following values for specific sample positions:

- Sample 32 contains the Hamming weight of $state0[i] \leftarrow p[i] \oplus r[i]$, where $p[i]$ is a plaintext byte and $r[i]$ is a random masking byte.
- Sample 33 contains the Hamming weight of a random masking byte: $state1[i] \leftarrow r[i]$.
- Sample 67 contains the Hamming weight of $state0[i] \leftarrow (state0[i] \oplus key[i] \oplus r_{in}) \oplus state1[i]$, where r_{in} is random mask value;
- Sample 68 contains the Hamming weight of $state0[i] \leftarrow S_{box} * [state0[i]]$, where $S_{box}*$ is a masked substitution table.
- Sample 69 contains the Hamming weight of $state0[i] \leftarrow (state0[i] \oplus state1[i]) \oplus r_{out}$, where r_{out} is a random mask value.

All other samples contain random values ranging from 0 to 8. Fig. 5 shows the results for the backward activation method against the simulated masked AES traces. Again, 100,000 traces were generated, where 99,000 were used for training and 1,000 for validation. As we can see, the trained network considers samples 33 and 68 as the main input samples for the activation of the output layer. These two samples are the mask $r[i]$ and the masked $S_{box}*$ output value, meaning that the trained neural network is actually combining these two samples for the output activation. These could be understood as a second-order leakage detection by the neural network.

6.3 Method Validation on ASCAD Database

We are interested in assessing the learnability of a neural network during training with respect to automatic selection of points of interest. A good enough trained model would be the one that classifies input traces by activating a neuron in the output layer because

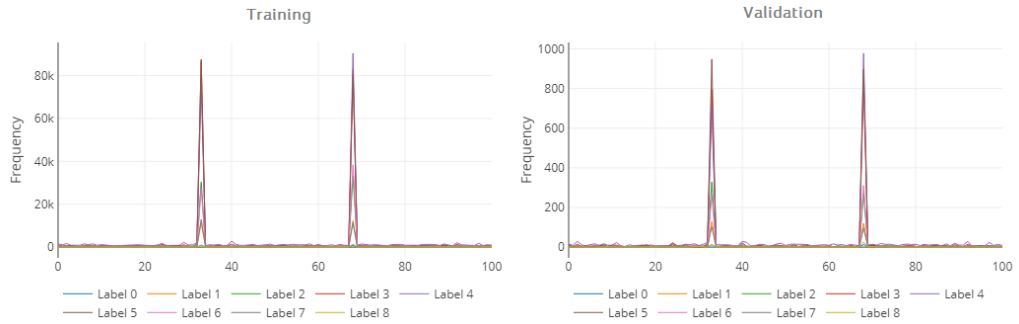

Figure 5: backward propagation path results for simulated masked AES traces. Result for training (left) and validation (right) traces.

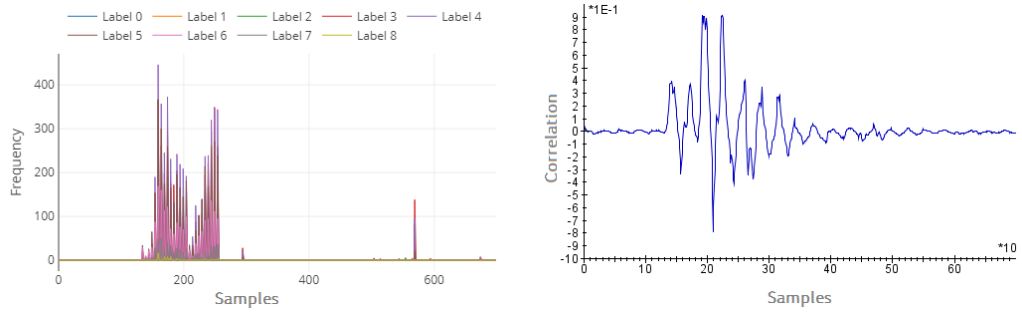

Figure 6: Backward propagation path result (left) and correlation with S_{box} output values for $k1$ (right).

it identifies a feature in the input layer that contains information related to the selected leakage model.

Neural network metrics like accuracy, precision or recall may indicate how well the neural network is being able to fit from training data and to generalize to validation data. Even if these metrics are very useful in demonstrating the amount of learning, they cannot represent if the neural network is actually detecting leakages from side-channel traces. For that, we consider the ASCAD database that allows us to validate the proposed method against unprotected and protected AES traces.

6.3.1 Software AES - ASCAD Database

In [16] the authors provide a full description of the AES implementation that was used to generate the ASCAD database for side-channel traces and it is available on *GitHub* (link). The authors suggest the usage of 50k traces for training and 10k traces for validation.

The AES implementation is protected with masking countermeasure and the side-channel traces contain information of electromagnetic emanation related to the first round. The operations on sub-key bytes 1 and 2 are not masked. Therefore, we select the trace interval containing 700 samples with respect to the processing of the $k1$. Fig. 6 provides the correlation analysis for the S_{box} output (right) and results for the backward propagation method (left) for the trained network over this interval. As we can obverse, the backward propagation method indicate that the trained network is actually selecting the samples that are actually point of interest (high correlation) as the main input features. In this case, we can assume that the network is actually recognizing with a high precision the location of points of interest.

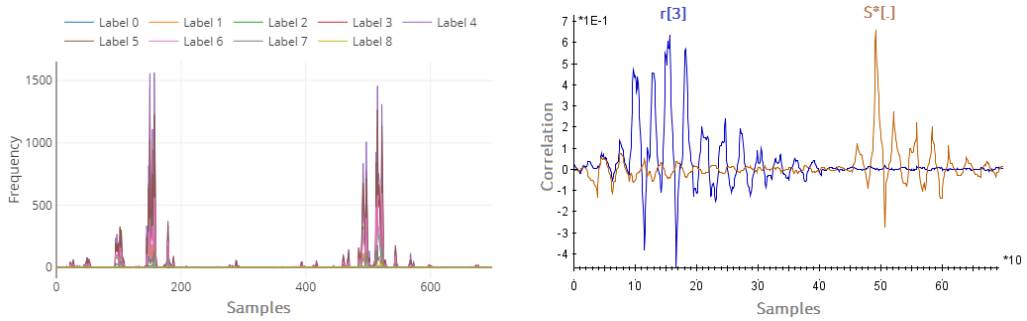

Figure 7: Backward propagation path result (left) and correlation with $r[3]$ and masked S_{box} output (right).

6.3.2 Software AES with Masking - ASCAD Database

AES rounds from sub-key bytes 3 to 16 are masked and the corresponding mask values are also available for each trace in the database. This information allows the calculation of the masked S_{box} output value. Based on that, we select the interval representing the processing of sub-key byte $i = 3$ and compute the correlation for $r[3]$ (mask for $k3$) as well as the masked S_{box} output value. Fig. 7 (right side) shows these correlation results. Next, we train a CNN over these data set and also implement the model assessment with backward propagation method. The analysis is performed in a black-box scenario, where the training traces are labelled based on the Hamming weight of S_{box} output without any masking into consideration: $\ell_{p_i,k_i} = HW(S_{box}(p_i \oplus k_i))$.

Fig. 7 (left side) provides results for this model assessment. Comparing both figures, it is possible to verify that the neural network considers the input samples related to the mask $r[3]$ and the masked S_{box} output as the main features. This confirms that the trained neural network is actually combining two or more samples for every trace, as a second-order attack.

It is important to highlight that the definition of a neural network that selects points of interests as main input features requires the tuning of hyper-parameters. Section 7 describes how small changes in some of these hyper-parameters completely changes these results. In particular, it is demonstrated how small variations in the regularization L2 completely modifies these points of interests selection by the neural network.

7 Optimizing neural network hyper-parameters from model assessment

So far, this paper has demonstrated that model assessment techniques are able to provide valuable information in order to visualize how trained neural networks identify the points of interest automatically. A further question in this investigation would be how to use model assessment techniques to improve model learnability by modifying hyper-parameters. The report presented in [16] provides several experiments for different hyper-parameters on the ASCAD database. Here, the idea is to compare different hyper-parameters and see how they influence the leakage detection by the neural network. We start by assessing how regularization L2 has an important impact on the learning process against a masked AES implementation.

7.1 Assessing the effect of L2 regularization

L2 regularization (also known as *weight decay* or *ridge regularization*) imposes a penalty during the parameters (weight connections) updates in the back propagation algorithm with stochastic gradient descent. This penalty prevents the weights to become too large and, as a consequence, the network has less chances to overfit the training data. The following equation helps us to understand how the regularization is employed in the cost or error function:

$$E(x)_{L2} = E(x) + \lambda \sum_{i,j} |\omega_{i,j}|^2 \tag{11}$$

The error $E(x)$ is based on an equation that evaluates the global distance between the actual labels and the predicted labels. Examples for that are mean squared error, negative log-likelihood and cross-entropy. By adding a regularization $\lambda \sum_{i,j} |\omega_{i,j}|^2$ to the error function (where λ is the regularization term and $\omega_{i,j}$ are the weights), the back-propagation interprets the error as being already too large and, as a consequence, the weights become smaller to compensate such an effect.

In this section we demonstrate how the same neural network architecture learns differently when different amounts of regularization are considered. It is well-known that the increase in the number of training traces also provide a inherent regularization to the network. However, this comes with increased training time which can be very critical for side-channel analysis. In [11], as already discussed in this paper, the authors considered random shifts over the training set during training phase as a regularization technique for convolutional neural networks. Their results prove that CNNs learn more from input traces if data augmentation techniques are considered against misaligned traces. In [21] the authors adopted the noise addition technique during training as a method to avoid overfitting. Moreover, the mini-batch size also provide an inherent regularization to the network, specially if larger mini-batch sizes are chosen. Here, we provide results for different L2 regularization values while keeping the rest of the hyper-parameters unchanged. Moreover, we show that a change in the number of training traces or the mini-batch size would require another optimal value for λ.

The main idea of this section is to emphasise the neural networks can learn more accurately from points of interests if a proper L2 regularization value is considered. We test the regularization on the masked AES traces from ASCAD database. The analyses are conducted over the sample interval representing the processing of S_{box} for $k3$ ($i = 3$ in [16]).

Section 6.3.1 presents successful results in terms of points of interest selection by the neural network. However, before achiving this good result, it was tested several different hyper-parameters and we verified that the L2 regularization was providing major effects in the points of interest identification. Here, we provide results for three different values for λ while the rest of hyper-parameters remain unchanged. First, λ is set to 0.01 and the backward propagation method results are shown in Fig. 8 (left). As the figure shows, the network learned from different places and it is not feasible to conclude that this trained model is actually performing its activation by having the points of interest (processing of mask $r[3]$ and masked S_{box} output values) as main input features. A preliminary conclusion is that the network is learning basically every sample in the input traces, which would lead to overfitting and, as a consequence, provide poor generalization.

A totally different result is achieved when λ is to set 0.05 and it is shown in Fig. 7 (left). It is impressive how a small change in λ makes the network to select almost exactly the points of interest as main features for its activations. Now, we raised λ to 0.1 and the backward propagation method results are shown in Fig. 8 (right). In this case, the network is inconsistently identifying main input features for its activations.

Figure 8: Backward propagation path method result for $\lambda = 0.01$ (left) and $\lambda = 0.1$ (right).

As expected, for $\lambda = 0.05$, the key ranking for $k3$ indicated a successful attack, even if the test accuracy and recall were close to a random guessing (0.285 and 0.12, respectively). When $\lambda = 0.01$ or $\lambda = 0.1$, the key recovery was not successful.

7.2 Number of training traces and L2

ASCAD database for the masked AES implementation contains 60,000 traces. As the authors of the database suggest [16], 50,000 traces are sufficient to train a neural network that is able to recover the key from this protected implementation when processing up to 10,000 traces in the test or validation phase. Similar results are proposed in [11], [14], [12], [15] and [21]. In this section, we demonstrate that the attack can be successful with less traces for both training and test phases. Additionally, we would like to emphasize that even with less traces for the training phase, it is possible to make the network to recognize leakages from the intervals containing actual points of interests, as demonstrated in Section 6.3.1. To achieve such results, we keep the network hyper-parameters unchanged, except for the L2 regularization term. Results clearly indicate the importance of the correct selection of this hyper-parameter in the learnability of the neural network parameters. Fig. 9 illustrates results after training with 20,000 and 40,000 training traces and with λ set to 0.09 and 0.075, respectively. This figure shows results for the application of the backward propagation path (top) and gradient visualization (bottom). The latter is evaluated for every epoch during training. The key recovery was successful for both cases.

7.3 Mini-batch size, learning rate and L2

When the learning rate or the mini-batch size is modified, there will be a different optimal range of values for λ in order to make the network to generalize (successful key recovery) and to learn from points of interests. Usually, to achieve similar and high validation accuracy, a modification in the mini-batch size or in the learning rate would require different number of epochs. This is one of the most costly parts of training neural networks, because the modification of one hyper-parameter affects the others. Assuming that we want our trained neural network to be successful against the ASCAD masked AES implementation in terms of key recovery as well as in terms of correct points of interest detection (a network that learns second-order leakages), we define a limit of 100 epochs and a learning rate of 0.01. By doing so, we need to identify the correct balance between mini-batch size and λ. Fig. 10 shows model assessment results using backward propagation path and gradient visualization when the mini-batch size is set to 64 and $\lambda = 0.01$ and also when the mini-batch size is set to 256 and $\lambda = 0.075$. As we can see, similar results in terms of points of interest identification were obtained by balancing the relationship between

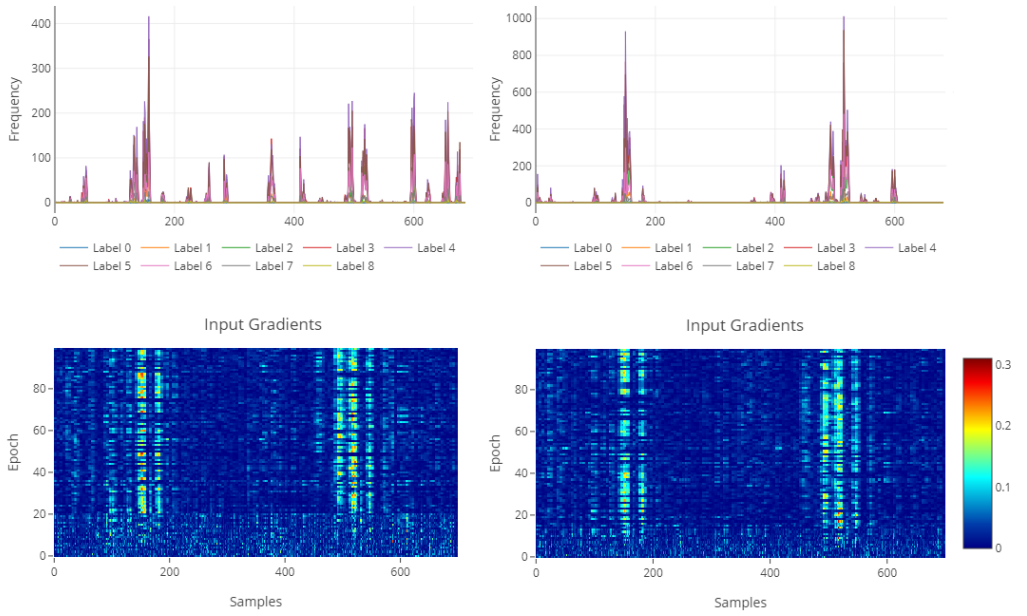

Figure 9: Results from backward propagation and gradient visualization methods for 20,000 training traces and $\lambda = 0.09$ (left) and 40,000 training traces and $\lambda = 0.075$ (right).

mini-batch size and λ. For both scenarios, the training phase considered 18,000 traces and key recovery was successful with less than 1000 validation traces.

A preliminary conclusion from these experiments indicates that the automated points of interest detection by the neural network is strongly related to regularization. Here, we mainly tested the effect of the L2 regularization against different mini-batch sizes and different number of traces. It is possible to achieve neural network configurations with very similar leakage detection capabilities for different combinations of hyper-parameters. As we mentioned before, the results presented in [14], [12] and in this paper provide similar results for leakage detection from neural network with very different configurations. Therefore, we can assume that for the evaluated traces, it is possible to make the network to identify points of interests from side-channel traces by optimizing global network hyper-parameters (mini-batch size, learning rate, regularization L2, optimizer) rather than looking for the optimal layer hyper-parameters (number of layers, number of neurons, kernel size, stride and activation functions).

8 Conclusion

In this paper, we extend the assessment of neural network models for side-channel analysis with a proposed technique called backward propagation path. After adopting neural networks as a tool to identify and classify leakages for profiled side-channel attacks, we concluded that model assessment, that verifies which input features the network is assuming as the most important ones for its decisions, is a very informative metric for the improvement and validation of neural networks models. The proposed method indicates, according to the selected number of backward paths, the input trace samples that contribute more for activation in the output layer. We provided results for unprotected and protected AES implementations. For reference results, we evaluated the protected AES traces from the ASCAD database and it was demonstrated how the network automatically selects points of interest if the hyper-parameters are correctly chosen. During th execution of this

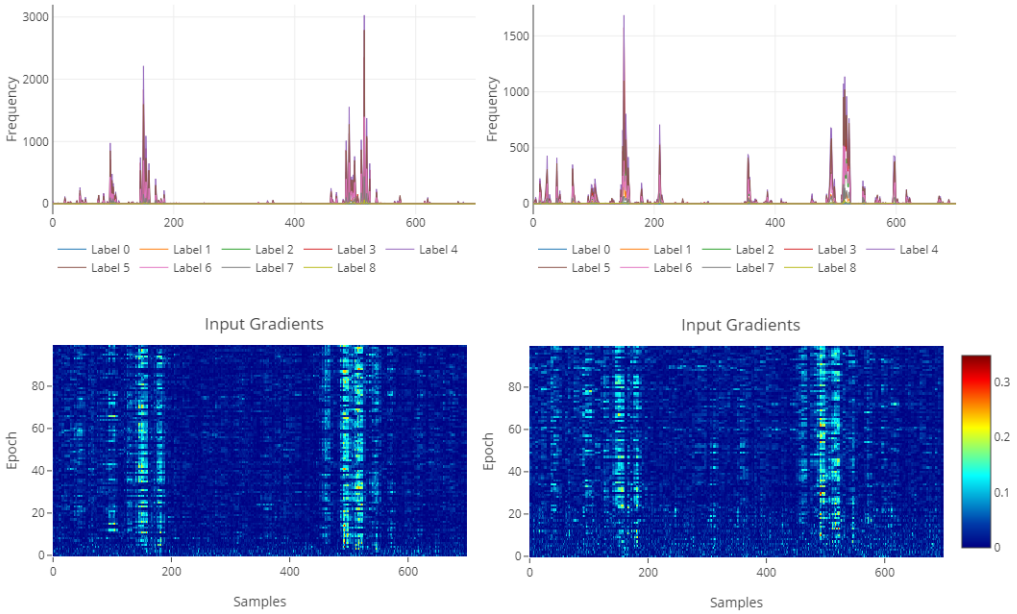

Figure 10: Results from backward propagation and gradient visualization methods for mini-batch size of 64 with $\lambda = 0.01$ (left) and for mini-batch size of 256 with $\lambda = 0.075$ (right).

research, we tested several layer-wise hyper-parameters (for convolutional and dense layers) as well as global hyper-parameters related to the back propagation algorithm. We verified that the L2 regularization (and its relation to number of training traces, learning rate and mini-batch size) plays an important role in the prevention of overfitting, allowing the network to learn from leaking samples with more precision if this value is correctly chosen for a given amount of training traces. Additionally, we suggest that hyper-parameter search is an important mechanism to train successful models as soon as we identify what are the most affecting hyper-parameters for the leakage detection. Here, we verified that L2 regularization can be tuned to an optimal value while keeping the rest of hyper-parameters unchanged. The obtained results confirmed that neural networks are able to combine two or more input samples in order to implement models that can learn second-order leakages from masked AES implementations.

9 Acknowledgements

This work was supported by the European Union's H2020 Programme under grant agreement number ICT-731591 (REASSURE).

References

[1] S. Chari, J. R. Rao, P. Rohatgi, Template attacks, in: B. S. K. Jr., Ç. K. Koç, C. Paar (Eds.), Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers, Vol. 2523 of Lecture Notes in Computer Science, Springer, 2002, pp. 13–28. doi:10.1007/3-540-36400-5_3.
URL https://doi.org/10.1007/3-540-36400-5_3

[2] W. Schindler, K. Lemke, C. Paar, A stochastic model for differential side channel cryptanalysis, in: J. R. Rao, B. Sunar (Eds.), Cryptographic Hardware and Embedded Systems - CHES 2005, 7th International Workshop, Edinburgh, UK, August 29 - September 1, 2005, Proceedings, Vol. 3659 of Lecture Notes in Computer Science, Springer, 2005, pp. 30–46. doi:10.1007/11545262_3.
URL https://doi.org/10.1007/11545262_3

[3] L. Lerman, S. F. Medeiros, G. Bontempi, O. Markowitch, A machine learning approach against a masked AES, in: Smart Card Research and Advanced Applications - 12th International Conference, CARDIS 2013, Berlin, Germany, November 27-29, 2013. Revised Selected Papers, 2013, pp. 61–75. doi:10.1007/978-3-319-08302-5_5.
URL https://doi.org/10.1007/978-3-319-08302-5_5

[4] Z. Martinasek, J. Hajny, L. Malina, Optimization of power analysis using neural network, in: Smart Card Research and Advanced Applications - 12th International Conference, CARDIS 2013, Berlin, Germany, November 27-29, 2013. Revised Selected Papers, 2013, pp. 94–107. doi:10.1007/978-3-319-08302-5_7.
URL https://doi.org/10.1007/978-3-319-08302-5_7

[5] P. C. Kocher, J. Jaffe, B. Jun, Differential power analysis, in: M. J. Wiener (Ed.), Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings, Vol. 1666 of Lecture Notes in Computer Science, Springer, 1999, pp. 388–397. doi:10.1007/3-540-48405-1_25.
URL https://doi.org/10.1007/3-540-48405-1_25

[6] E. Brier, C. Clavier, F. Olivier, Correlation power analysis with a leakage model, in: M. Joye, J. Quisquater (Eds.), Cryptographic Hardware and Embedded Systems - CHES 2004: 6th International Workshop Cambridge, MA, USA, August 11-13, 2004. Proceedings, Vol. 3156 of Lecture Notes in Computer Science, Springer, 2004, pp. 16–29. doi:10.1007/978-3-540-28632-5_2.
URL https://doi.org/10.1007/978-3-540-28632-5_2

[7] B. Gierlichs, L. Batina, P. Tuyls, B. Preneel, Mutual information analysis, in: E. Oswald, P. Rohatgi (Eds.), Cryptographic Hardware and Embedded Systems - CHES 2008, 10th International Workshop, Washington, D.C., USA, August 10-13, 2008. Proceedings, Vol. 5154 of Lecture Notes in Computer Science, Springer, 2008, pp. 426–442. doi:10.1007/978-3-540-85053-3_27.
URL https://doi.org/10.1007/978-3-540-85053-3_27

[8] L. Batina, B. Gierlichs, K. Lemke-Rust, Differential cluster analysis, in: C. Clavier, K. Gaj (Eds.), Cryptographic Hardware and Embedded Systems - CHES 2009, 11th International Workshop, Lausanne, Switzerland, September 6-9, 2009, Proceedings, Vol. 5747 of Lecture Notes in Computer Science, Springer, 2009, pp. 112–127. doi:10.1007/978-3-642-04138-9_9.
URL https://doi.org/10.1007/978-3-642-04138-9_9

[9] Z. Martinasek, O. Zapletal, K. Vrba, K. Trasy, Power analysis attack based on the MLP in DPA contest v4, in: 38th International Conference on Telecommunications and Signal Processing, TSP 2015, Prague, Czech Republic, July 9-11, 2015, IEEE, 2015, pp. 154–158. doi:10.1109/TSP.2015.7296242.
URL https://doi.org/10.1109/TSP.2015.7296242

[10] H. Maghrebi, T. Portigliatti, E. Prouff, Breaking cryptographic implementations using deep learning techniques, in: C. Carlet, M. A. Hasan, V. Saraswat (Eds.), Security,

Privacy, and Applied Cryptography Engineering - 6th International Conference, SPACE 2016, Hyderabad, India, December 14-18, 2016, Proceedings, Vol. 10076 of Lecture Notes in Computer Science, Springer, 2016, pp. 3–26. doi:10.1007/978-3-319-49445-6_1.
URL https://doi.org/10.1007/978-3-319-49445-6_1

[11] E. Cagli, C. Dumas, E. Prouff, Convolutional neural networks with data augmentation against jitter-based countermeasures - profiling attacks without pre-processing, in: W. Fischer, N. Homma (Eds.), Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings, Vol. 10529 of Lecture Notes in Computer Science, Springer, 2017, pp. 45–68. doi:10.1007/978-3-319-66787-4_3.
URL https://doi.org/10.1007/978-3-319-66787-4_3

[12] B. Timon, Non-profiled deep learning-based side-channel attacks with sensitivity analysis, IACR Trans. Cryptogr. Hardw. Embed. Syst. 2019 (2) (2019) 107–131. doi:10.13154/tches.v2019.i2.107-131.
URL https://doi.org/10.13154/tches.v2019.i2.107-131

[13] S. Picek, A. Heuser, A. Jovic, S. Bhasin, F. Regazzoni, The curse of class imbalance and conflicting metrics with machine learning for side-channel evaluations, IACR Trans. Cryptogr. Hardw. Embed. Syst. 2019 (1) (2019) 209–237. doi:10.13154/tches.v2019.i1.209-237.
URL https://doi.org/10.13154/tches.v2019.i1.209-237

[14] L. Masure, C. Dumas, E. Prouff, Gradient visualization for general characterization in profiling attacks, in: I. Polian, M. Stöttinger (Eds.), Constructive Side-Channel Analysis and Secure Design - 10th International Workshop, COSADE 2019, Darmstadt, Germany, April 3-5, 2019, Proceedings, Vol. 11421 of Lecture Notes in Computer Science, Springer, 2019, pp. 145–167. doi:10.1007/978-3-030-16350-1_9.
URL https://doi.org/10.1007/978-3-030-16350-1_9

[15] B. Hettwer, S. Gehrer, T. Güneysu, Deep neural network attribution methods for leakage analysis and symmetric key recovery, IACR Cryptology ePrint Archive 2019 (2019) 143.
URL https://eprint.iacr.org/2019/143

[16] E. Prouff, R. Strullu, R. Benadjila, E. Cagli, C. Dumas, Study of deep learning techniques for side-channel analysis and introduction to ASCAD database, IACR Cryptology ePrint Archive 2018 (2018) 53.
URL http://eprint.iacr.org/2018/053

[17] M. Carbone, V. Conin, M. Cornelie, F. Dassance, G. Dufresne, C. Dumas, E. Prouff, A. Venelli, Deep learning to evaluate secure RSA implementations, IACR Trans. Cryptogr. Hardw. Embed. Syst. 2019 (2) (2019) 132–161. doi:10.13154/tches.v2019.i2.132-161.
URL https://doi.org/10.13154/tches.v2019.i2.132-161

[18] R. Shwartz-Ziv, N. Tishby, Opening the black box of deep neural networks via information, CoRR abs/1703.00810. arXiv:1703.00810.
URL http://arxiv.org/abs/1703.00810

[19] F. Chollet, Deep Learning with Python, 1st Edition, Manning Publications Co., Greenwich, CT, USA, 2017.

[20] M. D. Zeiler, R. Fergus, Visualizing and understanding convolutional networks, CoRR abs/1311.2901. arXiv:1311.2901.
URL http://arxiv.org/abs/1311.2901

[21] J. Kim, S. Picek, A. Heuser, S. Bhasin, A. Hanjalic, Make some noise: Unleashing the power of convolutional neural networks for profiled side-channel analysis, IACR Cryptology ePrint Archive 2018 (2018) 1023.
URL https://eprint.iacr.org/2018/1023