IACR Transactions on Cryptographic Hardware and Embedded Systems
ISSN 2569-2925, Vol. 2020, No. 3, pp. 147–168. DOI:10.13154/tches.v2020.i3.147-168

Revisiting a Methodology for Efficient CNN Architectures in Profiling Attacks

Lennert Wouters, Victor Arribas, Benedikt Gierlichs and Bart Preneel

imec-COSIC, KU Leuven Kasteelpark Arenberg 10, B-3001 Leuven-Heverlee, Belgium
firstname.lastname@esat.kuleuven.be

Abstract. This work provides a critical review of the paper by Zaid et al. titled "Methodology for Efficient CNN Architectures in Profiling attacks", which was published in TCHES Volume 2020, Issue 1. This work studies the design of CNN networks to perform side-channel analysis of multiple implementations of the AES for embedded devices. Based on the authors' code and public data sets, we were able to cross-check their results and perform a thorough analysis. We correct multiple misconceptions by carefully inspecting different elements of the model architectures proposed by Zaid et al. First, by providing a better understanding on the internal workings of these models, we can trivially reduce their number of parameters on average by 52%, while maintaining a similar performance. Second, we demonstrate that the convolutional filter's size is *not* strictly related to the amount of misalignment in the traces. Third, we show that increasing the filter size and the number of convolutions actually *improves* the performance of a network. Our work demonstrates once again that reproducibility and review are important pillars of academic research. Therefore, we provide the reader with an online Python notebook which allows to reproduce some of our experiments[1] and additional example code is made available on Github.[2]

Keywords: Side-Channel Analysis · Machine Learning · Deep Learning

1 Introduction

Machine learning techniques gained a substantial interest in the recent literature on side-channel analysis [PSB+18, CCC+19, Tim19, ZBHV20, PHJ+19]. This rapid increase in research volume in an emerging topic creates the need for a critical review of these results. In this work, we take a critical look at a recent manuscript published in Transactions on Cryptographic Hardware and Embedded Systems. Specifically, we evaluate the work by Zaid et al. titled "Methodology for Efficient CNN Architectures in Profiling attacks" [ZBHV20]. The goal of their work is to propose a general methodology for designing Convolutional Neural Networks (CNN) depending on the nature of the target side-channel measurements. More precisely, they provide a few CNN design guidelines to decide on the filter size and the number of convolutional blocks to employ.

We demonstrate that these design guidelines are largely based on misconceptions and misattributions of empirical gains, and that even the authors themselves do not always adhere to their own design guidelines. The aim of this paper is certainly not to devalue the research results of [ZBHV20], but to build on previous results and instigate the progress of our field. Note that, it is common in the machine learning community to correct or point out misattributions in already-published papers [LS19, MDB17, DBK+18].

[1] https://colab.research.google.com/drive/1S4ixlEoLm9HqtP3Ku0vqZxm0-S9mq-Cw
[2] https://github.com/KULeuven-COSIC/TCHES20V3_CNN_SCA

Received: 2020-01-15 Accepted: 2019-03-15 Published: 2020-06-19

We hope that our paper clarifies some of the misunderstandings from the original work, and that it helps to start a discussion on where the future of machine learning based SCA research should be headed.

Contributions. The contributions of our work can be summarised as follows:

- **Preprocessing techniques.** We demonstrate how the first convolutional layer proposed by Zaid et al. can be replaced by classical preprocessing techniques. By doing so, we trivially reduce the model's complexity on average by 52%. We provide extensive experiments using several preprocessing techniques and demonstrate how they can influence the model's performance.
- **Model design parameters.** Through experimentation, we study the effect of network parameters such as filter size and the number of convolutional blocks. By performing these investigations, we demonstrate that the design guidelines outlined in the manuscript by Zaid et al. do not form a generally applicable methodology. By alleviating some of their misconceptions we provide a more intuitive reasoning which can aid towards the design of a general methodology.
- **Reproducibility.** By providing the reader with a Python notebook we ensure that the experiments outlined in this paper are easily reproducible. Furthermore, our implementations can be used to build and improve upon our work.

The remainder of this paper is structured as follows. In Sect. 2, we provide some of the necessary background on the specific topics discussed in this work. In Sect. 3, we investigate different preprocessing techniques and how they influence the performance of the trained networks. Different visualization techniques to assign attribution to certain input samples are discussed in Sect. 4. In Sect. 5 and 6, we provide an intuition as to why the models proposed by Zaid et al. worked and show that the filter size is not solely related to the amount of misalignment. Finally, in Sect. 7 we conclude and provide the reader with topics for future research.

2 Background

Several papers on the topic of machine-learning based side-channel attacks introduce the basic concepts [PSB+18, Tim19]; therefore we do not see a need to repeat those here. Profiled side-channel attacks correspond to Time Series Classification (TSC) problem. Several works regarding this topic are published and provide valuable information, from which we highlight the review manuscripts by Bagnall et al. [BLB+17] and Fawaz et al. [FFW+19]. They provide a thorough inspection of the most recent advances within the TSC domain, defining a solid base for successful machine learning based side-channel attacks. For an introduction to the topic of profiled side-channel attacks using neural networks based methods, we point to the work by Benadjila et al. [PSB+18], who conduct a broad investigation on deep learning algorithms, discussing the relationship with classical template attacks. Additionally, the paper provides an open-source dataset known as ASCAD, enabling reproducibility, and easing comparison with future results. Finally, as our work is a thorough analysis of the work by Zaid et al. we recommend the reader to read their manuscript [ZBHV20].

2.1 Datasets

Throughout this paper, we will use the same datasets as those used in the work by Zaid et al. All datasets are split equally, where 90% of the profiling traces are used for training,

and the remaining 10% are used for validation. Finally, the attack (or test) set is used as provided.

1. **ASCAD:** Was introduced by Benadjila et al. with the goal of having a public dataset which can be used to evaluate machine learning techniques [PSB+18]. The dataset consists of traces captured from a masked AES implementation running on an 8-bit ATmega8515 smart card with an operating frequency of 4 MHz. The raw traces cover the first round of AES and consist of 100k samples each. The traces are the result of acquiring the electromagnetic (EM) radiations with a sample rate of 2 GS/s samples per second.

   Benadjila et al. pre-selected 700 samples, therefore greatly simplifying the classification task. These 700 samples contain information on the first round S-Box processing of the third byte. Two additional datasets are generated which introduce a random shift to evaluate the robustness of machine learning techniques on misaligned traces. Each of the three variants consists of 50k profiling traces and 10k attack traces. We refer to the dataset without desynchronization, as ASCAD_N0, and the datasets with a maximum shift of 50 and 100 as ASCAD_N50, and ASCAD_N100, respectively.

2. **DPA-contest v4:** Was released as part of the fourth edition of the DPA contest. The dataset provides traces of a (rotating S-boxes) masked AES-256 software implementation on an 8-bit ATmega163 smart card controlled by a SASEBO-W board [NSGD12]. The EM measurements were acquired using a sampling rate of 500 MS/s. Each measurement consists of 435,002 samples covering the first round of AES.

   In their paper, Zaid et al. [ZBHV20], knowing the mask value, consider this implementation unprotected. Furthermore, a subset of 4000 samples per trace is used. The sensitive variable for this dataset is the first S-Box output XORed with the mask. In its reduced form the dataset consists of 4,500 profiling traces and 500 attack traces. It is unclear to us where this dataset originates from and how the subset was created. Nevertheless, we use the dataset as provided by Zaid et al. and will refer to it as DPAv4_mod.

3. **AES_HD:** Was introduced by Picek et al. [PHJ+19] and targets an unprotected AES-128 implemented on a Xilinx Virtex-5 FPGA as part of a SASEBO GII SCA evaluation board. The EM measurements consist of 1250 samples each. The leakage model targets the Hamming distance of the register overwrite in the last round. The profiling set contains 50k traces and the attack set consists of 25k traces.

4. **AES_RD:** The last dataset was provided by Coron and Kizhvatov [CK09]. The target is an implementation of AES-128 using a random delay countermeasure on an ATmega16 8-bit AVR microcontroller. The raw power traces where reduced to 3500 samples (one sample per CPU clock) covering the first round of AES. The target label is the output of the first S-Box. Both the profiling set and attack set consist of 25k traces.

2.2 Preprocessing techniques

Profiled side-channel attacks can be considered a time series classification task. The goal of the model is to predict the class (target sensitive variable) of a side-channel measurement.

In the field of time series classification, it is common practice to preprocess the data before training the machine learning model [BLB+17, FFW+19]. For example, the data provided as part of the UCR time series classification archive has been standardized to have zero mean and unit variance to remove the effects of offset and scale [DKK+18, DBK+18].

Dau et al. provide an intuitive reasoning (using the `GunPoint` dataset as an example) as to why the data has been standardized [DBK+18].

Multiple examples in the SCA literature demonstrate the use of standardization techniques for template attacks. For example, Montminy et al. demonstrated the efficacy of zero-mean unit-variance normalization in the context of portable multivariate template attacks [MBTL13]. Lemke-Rust and Paar also note the use of standardization in their work on multivariate Gaussian mixture models for higher-order profiled attacks [LP07]. Choudary et al. document differences in measurements across different measurement campaigns with the same device [CK18]. However, for some reason, the use of normalization or standardization has not been widely adopted or reported in the context of machine learning based side-channel attacks.

Similarly to Dau et al. we can provide an intuitive reasoning as to why preprocessing the raw data can lead to better results in the context of side-channel attacks. Two sets of side-channel measurements (e.g. a profiling and attack set) can have subtle differences. Differences over multiple measurement campaigns occur because of environmental changes (e.g. temperature [CEM18, MMR20]), changes in the measurement setup (e.g. supply voltage and oscilloscope settings [MMR20, CK14]), or because of slight differences between target devices (i.e. the portability problem [MBTL13, CK14]). Compensating for this offset and scaling can make it easier for the model to learn a classification which generalises. Nevertheless, in some specific cases a model can benefit from the information provided by the raw measurements [DBK+18]. As shown later in Sect. 3, it is therefore important to experiment with multiple preprocessing strategies. Additionally, normalizing all input features of a neural network to lie within a similar range can greatly increase the convergence rate of the training process [LBOM12].

In the machine learning context, the inputs to a model are often referred to as input features, whereas in the specific case of side-channel analysis, these input features are the measurement samples. Throughout this work we use both terms interchangeably. We refer to feature based preprocessing techniques as those techniques that normalize the input data (side-channel traces) per sample. On the other hand, horizontal or instance based preprocessing techniques, will be applied on a per trace basis.

We explore the effects of commonly used preprocessing techniques in the context of deep learning based profiled side-channel attacks. We experiment with both feature and instance based techniques. Specifically, we will explore the effects of scaling between 0 and 1 (as performed by Zaid et al.), scaling between -1 and 1, and standardizing the data (zero-mean unit-variance).

2.3 Visualization techniques

Multiple visualization techniques have been proposed to alleviate the black-box nature of Neural Networks (NNs).

The goal of these techniques is to assign *attribution* to the input features [ACÖG18]. The attribution is calculated with respect to the specific target neuron related to the correct class for a given example in the classification task. The result of combining the attributions of every input feature is known as the *attribution map*, which has the same dimensions as the input.

Several approaches exist to designate the contribution of the multiple input features. On the one hand, *Perturbation-based Forward Propagation* relies on making perturbations to single neurons or inputs, and examining the impact on subsequent neurons in the network. On the other hand, *Backpropagation-Based* approaches propagate a signal backwards from an output neuron through the network until the inputs in a single pass.

Perturbation-based Forward Propagation. Perturbation-based methods compute the attribution map by occluding or altering the input before running a forward pass on the

new input. Several strategies have been proposed for this approach [ZF14, ZT15, ZCAW17]. Perturbation-based techniques provide good results, but tend to be slow as the total number of input features becomes larger. Moreover, provided the non-linear nature of NNs, the outcome is highly influenced by the number of features altered at each iteration [ACÖG18].

Backpropagation-based. Backpropagation-based approaches determine the attributions from every input feature in a single forward and backward pass through the network. There are four methods that are more prominent within this category:

1. **Layer-wise Relevance Propagation (LRP):** Bach et al. [BBM+15] presented the technique known as LRP, that proposes two approaches: on the one hand, a Taylor-type decomposition; on the other hand, layer-wise propagation relevance. It applies a propagation rule that distributes class relevance found at a given layer onto the previous layer. The layer-wise propagation rule is computed with a backward pass through the network.

2. **Gradient * Input:** Proposed by Shrikumar et al. [SGK17a], this technique aims to enhance the sharpness of the attribution maps. To do so, the partial derivatives of the output with respect to the input are multiplied with the input itself.

3. **Integrated Gradients:** Introduced by Sundararajan et al. [STY17], this method makes use of the derivatives as well, similarly to the previous proposal. However, instead of computing the attribution with a single derivative, evaluated at a single point x, Integrated Gradients computes the *average* gradient. Furthermore, the input ranges linearly from the average $\bar{x}$ to x.

4. **DeepLIFT:** Shrikumar et al.'s method [SGK17b] explains the difference in the output from some "reference" output with respect to the difference between a "reference" input and the input provided. The attribution accredited to each unit i represents the relative effect of the unit when activated by the provided input compared to the activation with the "reference" input.

Multiple alternative techniques have been proposed within this category [SVZ14, ZF14, SDBR15, SCD+17, MLB+17]. The side-channel community has also actively looked into this problem. The paper from Masure et al. [MDP19] proposes Gradient Visualization (GV), a method similar to the Gradient * Input previously presented. Hettwer et al. [HGG19] conclude that LRP was the most successful method in their experiments. Perin et al. [PEC19] propose the technique called backward propagation path, which is an extension of the LRP method.

However, we use the work by Ancona et al. [ACÖG18], that revisits multiple methodologies and proposes a unified framework called DeepExplain[3]. They prove that the LRP and DeepLIFT methods can also be defined as computing backpropagation for a modified gradient function. Additionally, they define a new metric (known as *Sensitivity-n*) and compare the previous methods against this metric. In our experiments we use the DeepExplain framework to create the attribution maps using the Gradient * Input method.

2.3.1 Weight Visualization and Heatmaps [ZBHV20]

Despite all the previous visualization techniques available in the literature (Sect. 2.3) to illustrate the Points of Interest (PoI) of an input trace, Zaid et al. [ZBHV20] propose two new visualization techniques in addition to the techniques introduced above. They propose the methods weight visualization and heatmaps, the first one used for synchronized traces, and the second one for desynchronized traces.

[3] https://github.com/marcoancona/DeepExplain

The first method, weight visualization, is based on the weights of the first Fully Connected (FC) layer after the flatten layer. The second one, heatmap, is based on the activations of an intermediate layer.

Weight Visualization. In this method, the weights of the first fully connected layer following the flatten layer are analysed. This technique is defined as follows:

$$W_m^{vis}[i] = \frac{1}{n_f^{[flatten-1]}} \sum_{j=i\times n_f^{[flatten-1]}}^{(i+1)\times n_f^{[flatten-1]}} |W_m^{[flatten+1]}[j]|\,, \tag{1}$$

where $W_m^{vis}[i]$ is the weight visualization at a particular point i, for the m-th neuron of the first FC layer after the flatten layer (denoted as $[flatten+1]$), and $i \in [0, \text{len}([flatten-1])$. Similarly, $[flatten-1]$ denotes the last convolutional layer before the flatten layer. $n_f^{[flatten-1]}$ represents the number of filters in $[flatten-1]$, and $W_m^{[flatten+1]}$ the weights of the first fully connected layer.

Originally, the paper by Zaid et al. refers to the weights of the flatten layer ($W_m^{[flatten]}$). As a flatten layer does not have any trainable weights, we updated the definition to more clearly reflect that the weight visualization technique uses the weights of the first fully connected layer ($W_m^{[flatten+1]}$). Finally, the full weight visualization is defined as:

$$W^{vis}[i] = \frac{1}{n_u^{[flatten+1]}} \sum_{m=0}^{n_u^{[flatten+1]}} W_m^{vis}[i]\,, \tag{2}$$

in which $n_u^{[flatten+1]}$ is the number of neurons in $[flatten+1]$. The magnitude of the average weights is used to assign relevance to an intermediate feature for the classification task.

Heatmaps. The heatmap representation is computed as the average output over all filters of a convolutional operation, defined in [ZBHV20] as:

$$H^{[h]} = \frac{1}{n_f^{[h]}} \sum_{i=0}^{n_f^{[h]}} (\text{input}^{[h]} \circledast f_i^{[h]})\,, \tag{3}$$

where $H^{[h]}$ denotes the heatmap associated with the h-th layer, $f_i^{[h]}$ the i-th filter of the convolution at that layer, and $n_f^{[h]}$ the number of filters in the given convolution. The heatmap helps to understand the role of each convolutional layer in the learning process.

3 Preprocessing

Preprocessing the input data is of paramount importance when training a neural network to ensure that it can converge quickly. In this section, we discuss the preprocessing techniques used by Zaid et al. and show how they can be improved, resulting in networks with up to 69.97% less trainable parameters. We show that the first convolutional block used in the networks by Zaid et al. acts as a network internal preprocessing unit, and hence, that it can be removed given an appropriate preprocessing of the input trace.

3.1 Convolutions with filter size one

One remarkable addition to the networks proposed by Zaid et al. is the use of a convolutional layer with filter size one. A convolutional filter is shifted over the entire input time series. As the filter has a size of one, it only has two trainable parameters, a weight and a bias. The result of this filter is that each sample of each input trace is multiplied by the *same* weight after which the bias is added, followed by the application of the non-linear activation function. More formally, for each input sample x_i, we compute $\sigma(w \times x_i + b)$, where σ is the non-linear activation function, w is the weight, and b is the bias.

Therefore, using a one-dimensional convolution with filter size one at the input of a time series classification network, as proposed by Zaid et al. seems rather counter-intuitive. However, simply removing this layer from the provided networks makes it increasingly difficult for the networks to converge to a working solution. According to the explanation provided by the authors, this layer helps the network to focus its attention on the leaking Points of Interest (PoI). We conjecture that this layer is simply learning to scale the provided input which, in combination with batch normalization, allows the network to learn at an increased rate. Variations of this technique are sometimes referred to as network internal preprocessing [NZ12], but it does not seem to be a widely adopted technique.

Intuitively, if the first hidden layer uses four filters of size one, it will learn four scaled and shifted representations of the network's input (the power trace). Therefore, we hypothesise that this first layer can be omitted by properly preprocessing the input data.

Nevertheless, convolutions with kernels or filters of size one are employed in state-of-the-art classification networks [SLJ+15, FLF+19]. In these cases they are referred to as bottlenecks, used to reduce the dimensionality of the intermediate representation.

3.2 Experiments

In Sect. 3.1 we conjecture that the first convolutional layer with filters of size one can be omitted if the traces are preprocessed properly. In order to verify or disprove this hypothesis, we performed several experiments on the networks proposed by Zaid et al. and on slightly modified versions thereof. We exclusively remove the first convolutional layer (with filter size one) and the accompanying batch normalization. By deleting this first convolutional layer, we also remove the non-linearity resulting from its activation function. Note that a convolutional block is composed of a convolutional layer, batch normalization, and a pooling layer. We do keep the first average pooling layer as it performs dimensionality reduction. We do not perform any hyperparameter tuning on the modified networks and present our experimental results as is.

We train both the modified and the original networks in combination with different preprocessing strategies. In the case of aligned traces, Zaid et al. used the Sklearn *MinMaxScaler* function to scale the input features between 0 and 1. However, when training neural networks it is recommended to either standardise the data (zero mean and unit variance) or to scale it between -1 and 1. Scaling input features between 0 and 1 (all inputs are positive) slows down the rate at which the network can converge [LBOM12].

Recall from Sect. 2.2 that feature based normalization techniques are applied per input sample over all traces whereas instance based techniques are applied on a per trace basis. In these experiments, we compare feature scaling between 0 and 1 (as employed in the original work), feature scaling between -1 and 1, and zero-mean unit-variance standardization. Even though Zaid et al. did not use any preprocessing techniques in the case of misaligned traces, we investigate horizontal or instance based feature scaling between 0 and 1 and between -1 and 1, as well as instance based standardization. While we limit ourselves to these specific preprocessing strategies, it may be worth exploring the combination of techniques (e.g. horizontal standardization followed by feature standardization), providing the network with multiple preprocessed versions of the original data or using non-linear

transformations such as the Box-Cox transform [RHMV19].

The profiling data was split into training and validation data using a 90-10 split for each dataset. Each combination of model and preprocessing strategy was trained and evaluated ten times, each time using a different split and a random weight initialization. The ten distinct splits are generated in a reproducible manner and reused for each model. For both the modified and unmodified networks we use the same training strategy as documented by Zaid et al. We set the training parameters for each of the ASCAD datasets to be the same (50 epochs, batch size of 50, and a learning rate of 0.005). These parameters provide more stable results for both networks, and were modified because the parameters provided by Zaid et al. for ASCAD_N100 do not work for us when we try to train their models. Furthermore, the goal of our work is to study the network architectures and not the model optimization.

Table 1 shows the number of parameters in the networks proposed by Zaid et al. and the modified networks trained during the presented experiments. Our models entail a reduction of up to 70.0% in the number of trainable parameters, while still producing similar results. In the following, the results shown in Fig. 1 and 2 are discussed for each dataset.

Table 1: The number of trainable parameters for each of the models proposed by Zaid et al. and the number of trainable parameters after removing the first convolutional layer while preserving the first average pooling layer.

Dataset	Network parameters ([ZBHV20])	Network parameters (Sect. 3.2)	Network parameters reduced by
ASCAD	16,960	6,436	62.1%
ASCAD-50	87,279	41,052	53.0%
ASCAD-100	142,044	42,652	70.0%
DPAv4_mod	8,782	4,770	45.7%
AES_HD	3,282	2,020	38.5%
AES_RD	12,760	7,112	44.3%

Aligned datasets (Fig. 1). The results for the aligned datasets (ASCAD, DPAv4_mod and AES_HD) are all quite similar. The first convolutional layer (with filter size one) can be replaced by feature standardization or feature scaling between -1 and 1. These results also corroborate our previously stated hypothesis that the first convolutional block (in the models proposed by Zaid et al.) is learning to properly scale the inputs. On the contrary, feature scaling between 0 and 1 (as used by Zaid et al.) no longer works on any of the datasets when the first layer is removed. Interestingly, we can see that training the model as proposed by Zaid et al. on the aligned ASCAD dataset does not always result in a converging model (as indicated by the error bands in the top-left plot in Fig. 1). Further analysis revealed that this effect is due to the random weight initialisation, as the same weights led to similar results on different splits of the profiling data. This indicates that the proposed model in combination with the used training strategy is not a robust solution, and the same effect can be observed in some of the misaligned datasets.

Misaligned datasets (Fig. 2). Zaid et al. do not perform any preprocessing on the raw measurements in the case of trace desynchronization or misalignment. In each case, properly preprocessing the data leads to more consistent results, even without modifying the models. The error bands show that for some combinations of models and preprocessing strategies the models were unable to consistently converge.

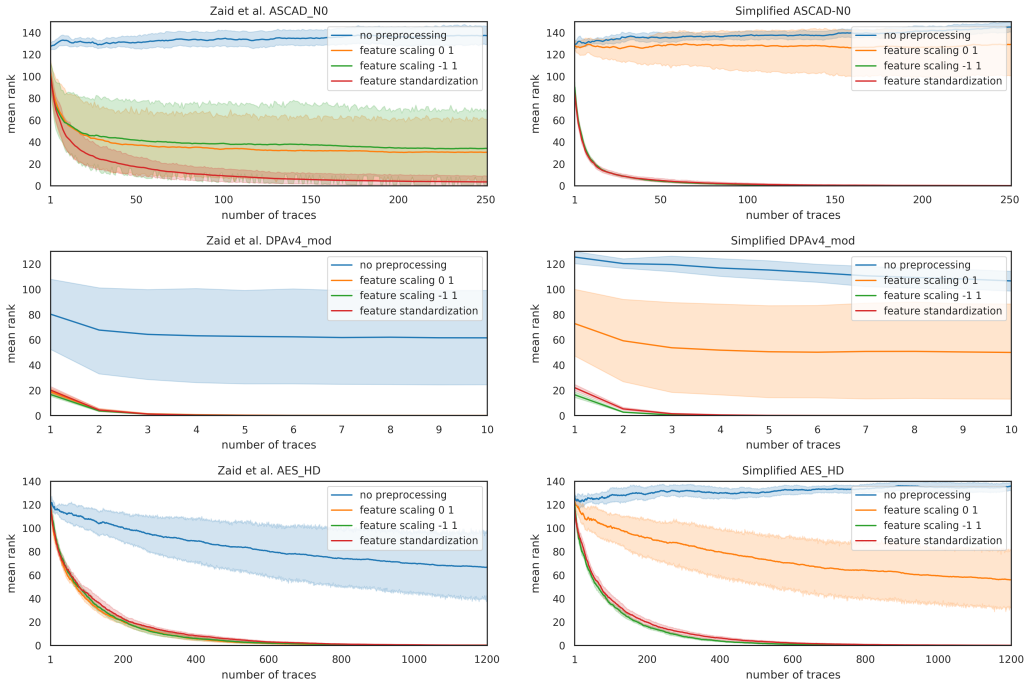

Figure 1: Each plot shows the evolution of the average rank of the correct key hypothesis over the number of traces used for the attack. For each model we performed 100 attacks on random subsets of the attack set. The error bands indicate the difference between multiple experiments on different splits of the profiling dataset. Comparison between the models proposed by Zaid et al. (left) and those same models without the first convolutional layer (right). The models are trained on the aligned datasets using different preprocessing techniques.

In the case of AES_RD, our simplified model performs slightly worse. We found that we can compensate for this by training the model for more epochs. Another interesting observation is that, without any preprocessing, the simplified models were never able to converge. Instead, for the misaligned datasets, not preprocessing the data does work in some cases. The main difference is that, in the aligned case, the simplified models are essentially Multi Layer Perceptron (MLP) models without any form of normalization. On the other hand, the simplified models for the unaligned scenario still have convolutional blocks which include batch normalization. Below, we study the effect of the batch normalization.

3.2.1 Batch Normalization

To study the effect of batch normalization on the models proposed by Zaid et al. and our simplified versions, we remove batch normalization from both models for the ASCAD_N100 dataset. As before, we train each model on the same ten splits. To compensate for removing batch normalization, we train for 200 epochs instead of 50 and reduce the learning rate to 0.001. We save the model state based on the validation accuracy, and the model state after completing 200 epochs. The results of this experiment are shown in Fig. 3.

Neither the original model proposed by Zaid et al. nor the simplified model manage to converge when the traces are not preprocessed. However, when the traces are preprocessed using horizontal standardization, both models consistently converge. Using horizontal standardization, the best model according to the validaton loss metric was found after on average 98 epochs for the model by Zaid et al. and after 144 epochs for the simplified

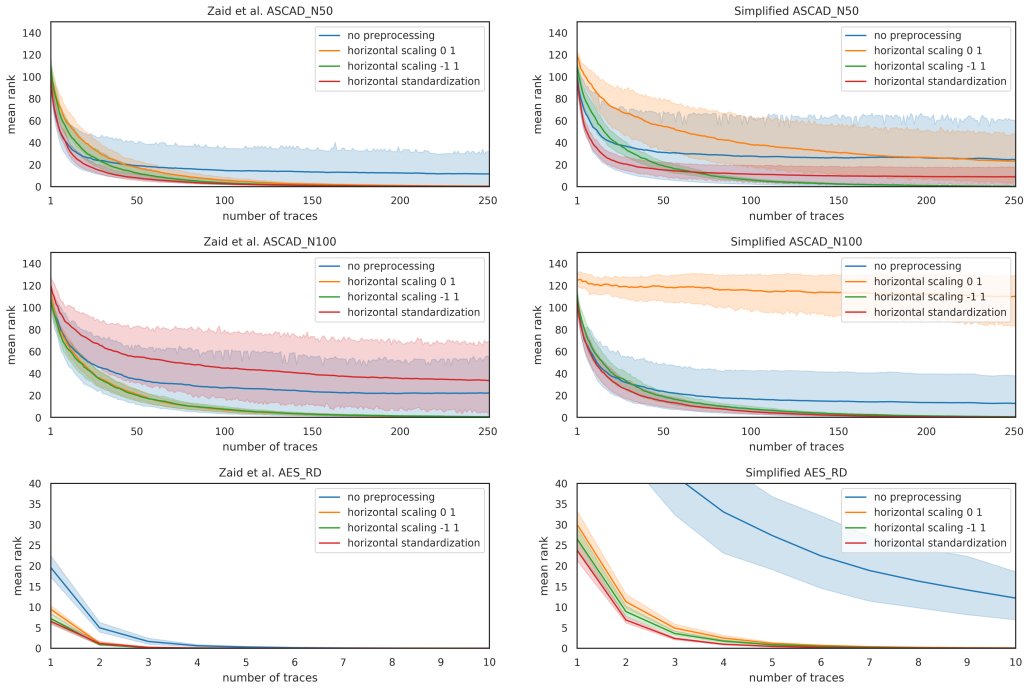

Figure 2: Comparison between the models proposed by Zaid et al. (left) and those same models without the first convolutional layer (right). The models are trained on the misaligned datasets using different preprocessing techniques. Best viewed on screen.

model. Without preprocessing, the best model according to the validaton loss metric was found after on average 32 epochs for the model by Zaid et al. and after 46 epochs for the simplified model.

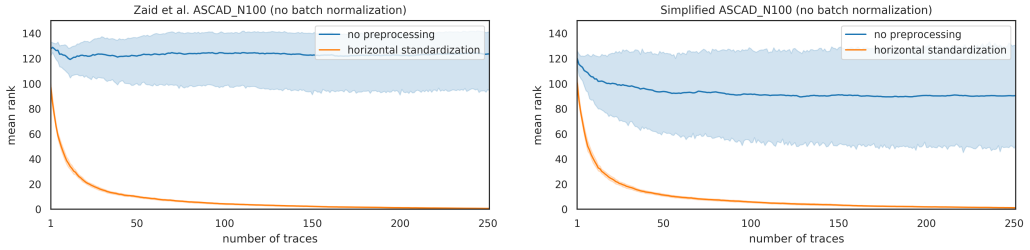

Figure 3: Comparison between the models proposed by Zaid et al. (left) and those same models without the first convolutional layer (right). The models are trained on the ASCAD_N100 dataset. Batch normalization has been removed for both models.

3.3 Conclusion

The experiments presented in this section demonstrate the importance of preprocessing the traces before training the model. We demonstrate that the preprocessing strategies adopted in [ZBHV20] and many other papers are suboptimal for these datasets. Additionally, we demonstrate that the first convolutional block with filters of size one behaves as a network internal preprocessing unit, which does not provide any clear advantage over traditional preprocessing techniques. Finally, our results show that by using batch normalization one

might not always have to perform preprocessing to get good results. However, the datasets, training strategy and models used in this manuscript always benefit from preprocessing.

As shown in Table 1 removing the first convolutional layer reduces the network parameters on average by 52%. This reduction does not provide a significant practical advantage on the used datasets as the original networks are already small by today's standards. However, this reduction will be more pronounced and important on more realistic datasets with longer traces.

4 On the correct use of visualization techniques

In this section, we present how techniques previously introduced in the deep learning community perform equally well or even better than the weight visualization technique introduced in [ZBHV20]. Additionally, we argue that the conclusions drawn from the proposed techniques are inaccurate, which led to erroneous results, on which we elaborate in subsequent sections.

4.1 Visualization techniques

Zaid et al. argue that techniques such as Gradient Visualization [MDP19] are "not appropriate if we want to interpret the convolutional part of the network". The weight visualization proposed by Zaid et al. attempts to visualize how the network learns relevant information by looking at the weights of the first FC layer of the classification part.

However, this methodology is unable to precisely locate the Points of Interest in the actual input trace. To get those samples, backpropagation methods are needed, bringing us to the techniques presented in Sect. 2.3. In our experiments, these methods (as implemented in the DeepExplain framework), obtain similar or even better results. We illustrate this comparison in Fig. 4. From the options available in DeepExplain, we choose the first method, Gradient*Input, but similar results can be obtained with other techniques.

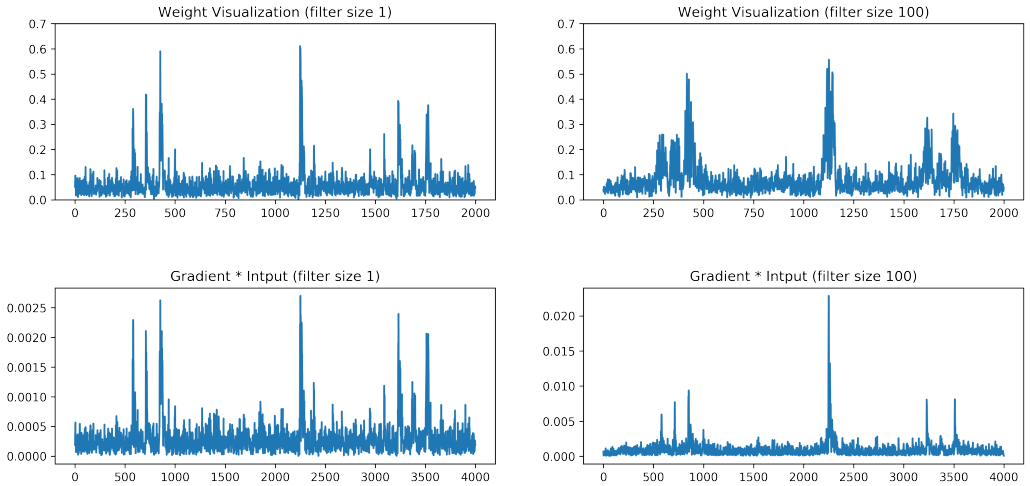

Figure 4: Weight Visualization as used by Zaid et al. (top) versus Gradient*input (bottom) for filter size 1 (left) and filter size 100 (right), using DPAv4_mod traces. Note the differences in x-axis scale between both techniques and the differences in y-axis scale between the different experiments.

Figure 4 illustrates the results for the model corresponding to the DPAv4_mod dataset proposed by Zaid et al. which has a single convolutional block with filters of size one. To

generate the plots on the left, we used the exact same model, whereas to generate the plots on the right, we solely changed the filter size to 100. The top row shows the weight visualization as defined in Eqn. (2) (note that we obtain the same results as in [ZBHV20]). The bottom row illustrates the results when applying the Gradient*Input technique from the output (propagating back until the input). We see that both results show similarities, but also important differences.

The first difference we observe is the result of the visualization when applying bigger filter sizes. Due to the size of the filter, we can see in the weight visualization that the relevant data is spread out over multiple points. On the other hand, with Gradient*Input, we can more precisely determine the PoI since the representation goes back completely to the input. The second and most important difference, is the fact that the weight visualization size does not correspond to the input size, hindering the precise localization of the PoI. Note that the input trace has a length of 4000 samples. Figure 15 in [ZBHV20] demonstrates the same effect, if the intermediate representation is reduced too much (because of pooling) the weight vizualization technique is no longer able to accurately identify the leaking input samples.

4.2 Interpretation of the visualization techniques

We believe that the interpretation provided by the authors of [ZBHV20] based on the weight visualization technique is inaccurate, and this led to erroneous conclusions. They use this technique as a conclusive factor to quantify the performance of the network, based on the magnitude of the weights.

Based on the results provided in Fig. 14 of [ZBHV20], Zaid et al. conclude in Sect. 4.2 (of the same work) that: “If we want to precisely define the temporal space where leakages occur, we recommend to minimize the length of the filters in order to reduce the risk of entanglement”. An effect similar to the results presented in Fig. 14 from [ZBHV20] can be observed in the top-right plot of Fig. 4 (this manuscript), where the relevance of the PoI is also spread out. According to the conclusion of Zaid et al., this spread harms the performance of the network leading to the conclusion that a network using bigger filter sizes performs worse than using smaller sizes. However, if one would look at the Gradient*Input results, and following the same reasoning, the conclusions drawn would be the opposite.

An intuitive way to demonstrate that this conclusion might not be correct is to evaluate the attack performance of the two models, the first one with filters of size one in the first convolutional block (as used by Zaid et al.), and the other one with filters of size 100. Figure 5 presents the result for this experiment, on both the DPAv4_mod dataset and ASCAD_N0 dataset. According to the interpretation given by Zaid et al., the model with filter size 100 should perform worse on the attack set than the one with filter size 1. However, the results from Fig. 5 suggest the opposite. Certainly, the model trained with bigger filter size does not perform worse than the one trained with smaller filter size. We further investigate the effect of the filter size in Sect. 5.

Following a similar argument for the number of convolutional blocks, Zaid et al. conclude in Sect. 4.3 of their manuscript that: “the deeper the network, the less confident it is in its feature detection”. This claim is based on Fig. 15, and 16 from Appendix D, where we can see how the magnitude of the weights diminishes with the number of convolutional blocks. We present in-depth investigations regarding the number of convolutional blocks in Sect. 6 to demonstrate that the conclusions from Zaid et al. are erroneous.

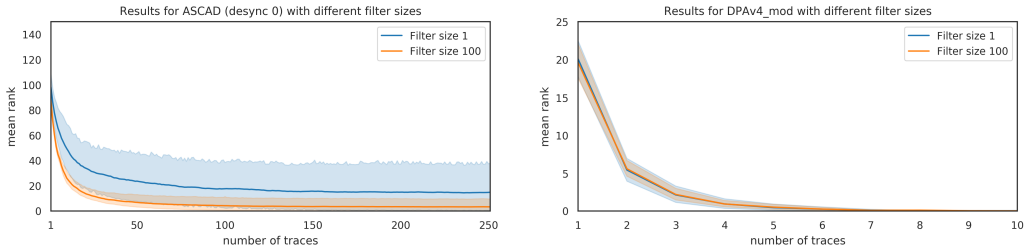

Figure 5: Guessing entropy comparing different filter sizes (1 and 100) on the same model used for the DPAv4_mod dataset.

5 Entanglement and Filter size

In this section, we describe the concept of entanglement introduced in [ZBHV20], analyze its effect, and revisit the conclusions that Zaid et al. draw from this concept on the filter size. Moreover, we show how the performance of a network can improve by increasing the filter size, contrary to the claims of [ZBHV20].

5.1 Entanglement

Zaid et al. coin a new concept called *entanglement*. They introduce this concept in order to explain the effect of a convolutional filter whose goal is to extract relevant input samples, and how these extracted features are distributed in subsequent layers of the network. Entanglement, in this context, means that relevant information (from the input) is shared and mixed within multiple convoluted samples. This effect causes a horizontal spread of the PoI over several convoluted points. According to the authors of [ZBHV20]: "increasing the length of the filter causes entanglement and reduces the weight related to a single information and therefore, the network confidence". In order to locate the leakage source more precisely, they propose to minimize the length of the filters to reduce the effect of entanglement.

Zaid et al. attempt to demonstrate the relevance of entanglement and why, according to their criterion, bigger filter sizes perform worse using their weight visualization method (cf. Fig. 14 from [ZBHV20]). As shown in previous section, we claim that the weight visualization method does not provide suitable results to claim that a network with longer filters performs worse.

In the top graphs of Fig. 4, we can see the effect of the entanglement, and how the leaking samples are more distributed in the horizontal axis when the filter size increases. On the contrary, as we see from the Gradient*Input (bottom plots of Fig. 4), the exact relevant Points of Interest from the actual input trace are successfully revealed, no matter which filter size was used. Moreover, as shown in Fig. 5, the model with bigger filter size performs equally well (if not better) than the model with filter size 1. These examples illustrate, and experimentally validate, that the argument about entanglement diminishing the performance of a neural network is flawed.

Our intuition is that the goal of a filter is to detect a certain pattern. This objective can be achieved by combining multiple layers of small filters (as is often the case in image classification networks) or by training bigger filters, which should also be able to detect a smaller pattern, by learning and fine-tuning the right weights. This matches the experimental results shown in the bottom-right plot of Fig. 4, regardless of the filter size we can still accurately trace back the leaking samples.

Our experiments indicate that the problem of entanglement is non existent in this context, and leads to incorrect conclusions. Therefore, we find the concept of entanglement,

which has a suggestive colloquial meaning, unnecessary. [LS19]. In the following section, we provide further experiments on the influence of the filter size to support our claim.

5.2 Filter size

Fawaz et al. [FLF+19] claim that a bigger filter size is capable of detecting longer patterns, which trivially would also detect the smaller patterns. They define the concept of Receptive Field (RF), a theoretical value that quantifies the maximum field of view of a neural network in a one-dimensional space. In other words, it refers to the number of input samples that a weight from further layers in the network sees, or, how many input samples affect one weight deeper in the network. Fig. 6 shows a graphical definition of this concept. This notion is only used for the convolutional parts of the models.

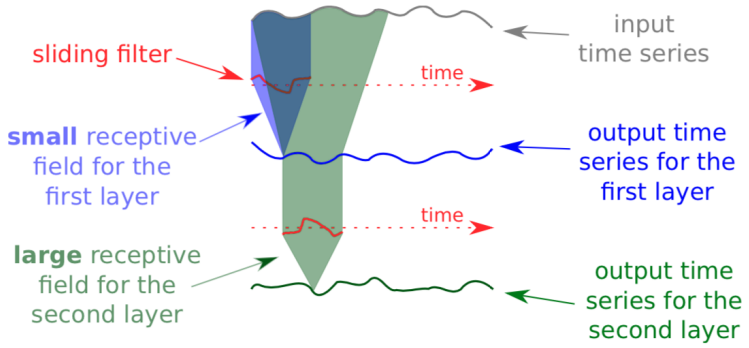

Figure 6: Receptive field representation from a two-layers CNN. Taken from [FLF+19]

According to the previous analysis, bigger filters determine more robust and generic networks, but also increase the number of parameters that have to be trained. Therefore, we want to find a compromise between the feature detection capabilities and the performance of the network.

Experiments

In the following, we present several examples to illustrate the effect of the filter size on the performance of a network on the different datasets with misaligned traces. As demonstrated in Sect. 3, these are the only cases where convolutions are needed.

Following the conclusions from Sect. 3, we use the simplified models for the subsequent experiments. We pick horizontal standardization as the preprocessing method as it gives good results overall. As illustrated in Figs. 1 and 2, the error bands for this preprocessing technique are slim, giving a higher confidence in the results of the experiments below. The three models used in the experiments have the same structure: two convolutional blocks preceded by an average pooling. In these models we only change the filter size as this is the parameter we try to investigate. Thus, we isolate this parameter so the changes are solely due to the changes in the filter size.

In these experiments, we train several models on the ASCAD_N50 and ASCAD_N100 datasets, each of the trained models is based on the simplified ASCAD_N50 model. We conduct similar experiments on the AES_RD datasets using the simplified AES_RD model. We exclusively change the filter size of the first convolutional block (leaving the pooling size and stride as is) to generate new models with smaller or bigger filter sizes.

From the results shown in Fig. 7, we observe that small filter sizes lead to non-converging models for the desynchronized ASCAD datasets. Interestingly, this effect is not observed as much on the AES_RD dataset. Additionally, we note that there is no significant difference

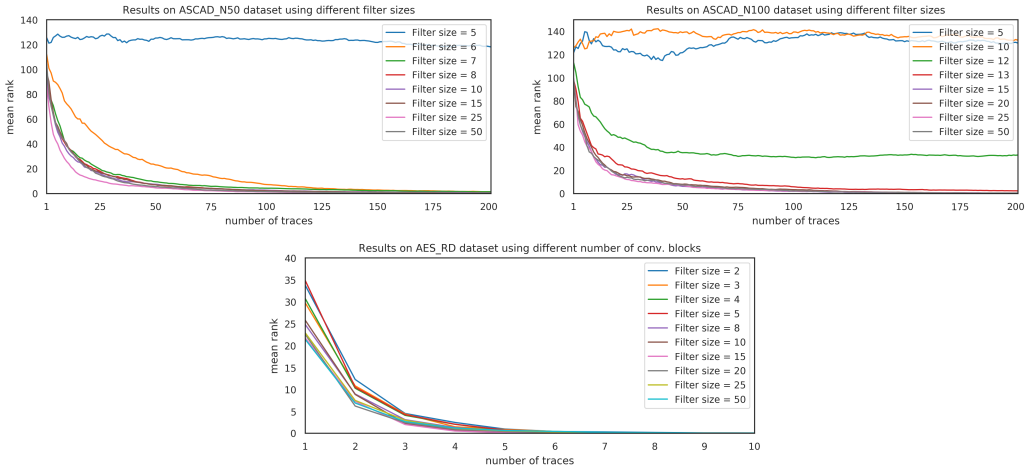

Figure 7: Effect of the filter size on the performance of a network when attacking the datasets ASCAD_N50 (left), ASCAD_N100 (right), and AES_RD (bottom).

in performance when training the same network on the ASCAD_N50 and ASCAD_N100 datasets with a filter size of 15. Moreover, increasing the filter size does not have a negative impact on the network performance for these datasets.

These results again suggest the non-existance of entanglement and are in line with the results of Fawaz et al. since it appears that the network fails to recognize the pattern to make the correct classifications when the filter size is reduced excessively.

We note that the model trained on AES_RD does not seem to be influenced as much by the filter size. One intuitive explanation for this observation is that for this dataset each clock cycle is represented by a single time sample, which might make it easier to detect the leaking pattern using a small filter. We note that this is a speculative statement which we did not thoroughly investigate. It is clear that a good filter size will depend on the underlying features that have to be detected. While increasing the filter size does not have a negative impact on the ASCAD dataset the results could be different in measurements suffering from dynamic misalignment.

5.3 Filter size on desynchronized traces

In [ZBHV20], Zaid et al. propose to use a filter size of $\frac{N^{[0]}}{2}$ for the second convolutional block, where $N^{[0]}$ is the maximum shift in the input traces. In contrast, we believe that the filter size is mostly related to the pattern which has to be detected by the network. Additionally, from the experiment presented above, we can conclude that the boundaries proposed by Zaid et al. on the filter size are not needed to cope with certain degree of misalignment in the traces, but we can confirm that indeed the filter size is affected by the presence of desynchronization.

In Fig. 7, we see how a filter size of 25 behaves best in both ASCAD_N50 and ASCAD_N100 datasets. Moreover, filter sizes 10 and 15, respectively, perform very similarly.

To strengthen our argument, we noted that Zaid et al. do not follow their recommendations when choosing the filter size for the model associated with the AES_RD dataset. By using the Gradient*Input technique for a few individual traces we can clearly see the amount of desynchronization present in the dataset. For example, as shown in Fig. 8 the dataset contains traces which are shifted by at least 1000 samples. According to the methodology provided by Zaid et al. the suitable filter size should be 500 provided the

misalignment of 1000 samples. Nevertheless, Zaid et al. choose a filter size of 50. As demonstrated by our previous results (Fig. 7), convolutional filters with a size as small as two still work on this dataset.

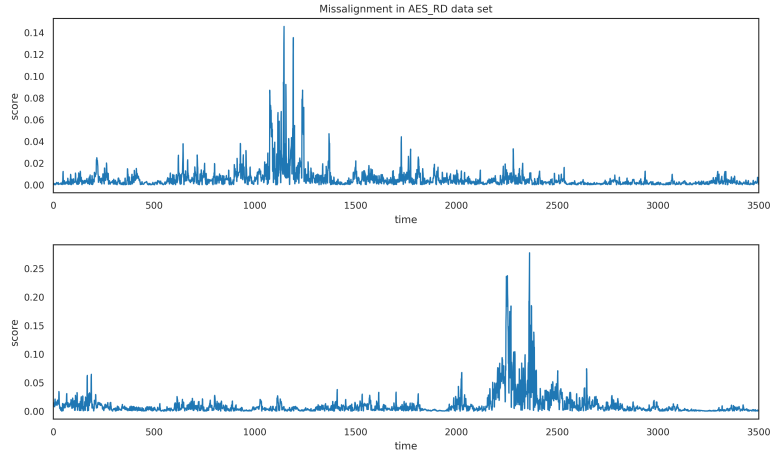

Figure 8: Points of Interest for two traces from the AES_RD dataset showing the misalignment within the different traces. These plots are generated using the Gradient*Input method.

Finally, we see that there is some relationship between the filter size and the degree of misalignment, since for ASCAD_N50 a filter size of 7 works, but not in the case of ASCAD_100. So while there might be some influence from the amount of misalignment on the best filter size there are certainly other factors at play. To decide the filter length for training a network, we have to take into account both the length of the pattern to analyze and the desynchronization, and find a trade-off between the filter size and the number of parameters that this filter entails. This trade-off could additionally be influenced by countermeasures like shuffling and dummy rounds and by the acquisition paramaters.

6 Number of Convolutional blocks

Zaid et al. also provide conclusions on the number of convolutional blocks, and how they affect the performance of a neural network: "we can see that, the deeper the network, the less confident it is in its feature detection" ([ZBHV20], Sect. 4.3 at the end of Page 12). Again, similarly as with the filter size, this claim contradicts the results in [FLF+19]. From their experiments, Fawaz et al. conclude that increasing the depth does not necessarily result in a better model, but it does not seem to diminish performance. They also note that a shallower architecture contains a significantly lower Receptive Field, which entailed lower accuracy in the models tested.

To claim that deeper networks perform worse on the feature detection task, Zaid et al. once again draw their conclusion from the weight visualization and the value of the weights (see Fig. 15 and 16 from Appendix D in [ZBHV20]). As already mentioned before, we believe this is not the correct method to draw these conclusions. In the following experiments, we show that adding convolutional blocks can indeed have a positive impact on the network's performance. We expect that for longer traces the advantage would be more prominent as was also indicated by Fawaz et al. [FFW+19].

6.1 Experiments

For these experiments, we use the simplified models with horizontal standardization preprocessing. The three models have two convolutional blocks, preceded by an initial average pooling. However, to successfully realize these experiments, we need to modify the pooling size of the first convolutional block. This is necessary due to the small size of the traces; applying a pooling operation with size 25 and stride 25, divides the trace by the same amount, making it impossible to use more than one convolutional block.

We replicate the first convolutional block in full (including convolution, batch normalization, and average pooling) several times, keeping the rest of the hyperparameters untouched. This way, the compared models solely differ in the number of intermediate convolutional blocks. To better illustrate the impact of the extra convolutional blocks, we start from the models which, due to their small filter size, did not perform well in Sect. 5.2.

From the experiments performed in previous sections, we noticed that the simplified ASCAD_N50 model with filter size 5 did not work on any of the evaluated ASCAD datasets. In the case of the simplified AES_RD model, we saw that even the model with filter size 2 can successfully attack the AES_RD dataset, although the performance is lower compared to other filter sizes. We employ these models for the subsequent experiments, selecting similar pooling sizes of 5 and 2, respectively. We replicate the first convolutional block two times for the ASCAD models and six times for the AES_RD model.

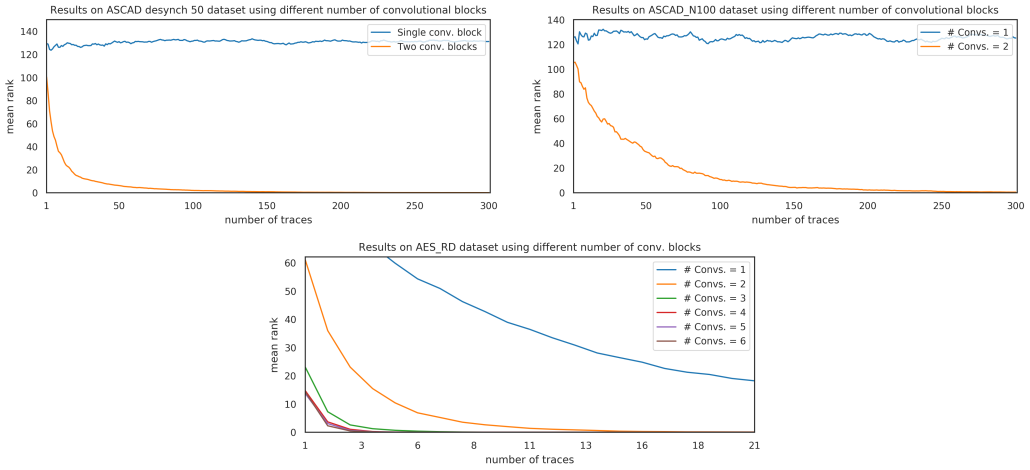

Figure 9: Effect of the number of convolutional blocks on the performance of a network when attacking the datasets ASCAD_N50 (left) and ASCAD_N100 (right), using the model Simplified ASCAD_N50.

6.2 Conclusions

The results in Fig. 9 show that adding one additional convolutional block significantly increases the performance on both ASCAD datasets. Since the pooling size used for the AES_RD traces is smaller, we have room to increase the number of convolutional blocks up until six. Based on the results shown in the bottom plot of Fig. 9, we clearly see that the greater the number of convolutional blocks, the better the model performs.

Similarly to increasing the filter size, expanding the number of convolutional blocks also broadens the Receptive Field of the deeper points in the network. This in turn increases the size of the pattern recognized in the input trace.

With these experiments, we can experimentally confirm that the addition of more convolutional blocks improves the performance of the network and makes it more confident

regarding feature detection, contrary to what Zaid et al. claimed. Naturally, as with the filter size, the number of convolutional blocks to add in the architecture of a NN entails a new trade-off between the number of trainable parameters and the performance of the network.

A note on pooling size

In the last two sections, we have analyzed in depth the effect of the filter size and the number of convolutional blocks. For these experiments, we always left the pooling size constant (as well as many other parameters). It is important to note that the pooling size also affects the performance of a network. By looking at Figs. 7 and 9 we see that the respective plots "Filter size = 2" and "# Convs. = 1" only differ on the average pooling size, which has values 25 and 2, respectively. From these results, we can observe that the performance of the model with filter size 25 is significantly better. Again, the greater the pooling size, the greater the Receptive Field of the deeper points in the network, which, as we saw before, helps to improve the confidence of the model. Certainly, as Zaid et al. mentioned, a too large pooling size would lead to information loss, so we have to find a compromise.

This suggests that the pooling size (and stride) is also an important factor to take into account when designing an architecture. The analysis of the pooling layer is outside the scope of this paper. However, we believe it is an interesting feature to study in more depth, in combination with the use of global average pooling.

7 Discussion and Conclusions

We have shown how some of the design methodologies outlined in [ZBHV20] are based on misconceptions and misattributions. By providing a more credible explanation for the empirical gains in their work, we assist other scientists who are building on these results.

The experiments and observations outlined in this paper help to form an intuition for how to design a neural network for profiled side-channel attacks. Therefore, our work also shows the importance of performing an ablation study when a new technique or model architecture is proposed.

Specifically, we demonstrated the importance of properly preprocessing the side-channel traces, and that the first convolutional block proposed by Zaid et al. can be omitted. This resulted in a trivial reduction in size of the networks on average by 52% while maintaining similar performance. Additionally, we showed that the filter size is not directly related to the amount of desynchronization in the traces. This observation is consistent with what would be expected from the knowledge of other domains, i.e. the translation invariability of CNNs copes with the misalignment and the filter size is related to the intrinsic nature of the features we want to extract.

It seems that Zaid et al. attempted to condense all relevant information into a few "aligned" samples which are then provided to the fully connected part of the network through the aggressive use of local pooling. We believe that the methodology provided by Zaid et al. provides a first step towards a systematic approach. However, we require a better understanding of different factors, regarding the acquisition framework (operating frequency, sampling rate and the amount of misalignment) in relation to the network parameters.

Our study was possible, in no small part, because Zaid et al. provided their implementations online and data sets were publicly available. In similar fashion, we provide the reader with a few examples which are easy to run online, without requiring any special hardware or software installation.[4] Additionally, supplementary code is available in a

[4] https://colab.research.google.com/drive/1S4ixlEoLm9HqtP3Ku0vqZxm0-S9mq-Cw

Github repository.[5]

For future research, we suggest considering the work from the time-series classification community as a solid foundation. Instead of reinventing the wheel, it would be beneficial to use the techniques or models developed in this research area, and combine those with the information from our community. Thus, obtaining better models and a better intuitive understanding of how they work.

As shown by the results in the work by Zaid et al. and in our work, we have outgrown the datasets which are often used in machine learning based side-channel analysis papers. Therefore, it would be interesting to evaluate more realistic scenarios. For example, the raw ASCAD traces of 100k samples are reduced to 700 samples. While this was likely a good starting point, we feel the reduced dataset (which is arguably the most difficult one from the set of used datasets) has become easy to attack. Therefore, it would be interesting to attempt to attack the raw traces or a bigger subset. As noted earlier, it would be worthwhile to investigate the relation between good network parameters and acquisition parameters. Additionally, investigating the resilience of machine learning based methods to other countermeasures (e.g. clock jitter, instruction shuffling, dummy rounds, etc) could be interesting as these could require a different design methodology.

The networks required to successfully attack these datasets are many orders of magnitude smaller than the networks trained in other domains. Therefore, we believe it would be more interesting to design network architectures work on multiple datasets, similarly to how new network architectures in the time series classification domain are evaluated on many datasets [FFW+19, FLF+19, DBK+18].

Acknowledgments

This work was supported by CyberSecurity Research Flanders with reference number VR20192203. In addition, this work was supported in part by the Research Council KU Leuven C1 on Security and Privacy for Cyber-Physical Systems and the Internet of Things with contract number C16/15/058, in part by the Flemish Government through FWO Sancus G.0876.14N and in part by the European Commission through the Horizon 2020 research and innovation programme under grant agreement Cathedral ERC Advanced Grant 695305.

References

[ACÖG18] Marco Ancona, Enea Ceolini, Cengiz Öztireli, and Markus Gross. Towards better understanding of gradient-based attribution methods for deep neural networks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018.

[BBM+15] Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PLOS ONE*, 10:1–46, 07 2015.

[BLB+17] Anthony J. Bagnall, Jason Lines, Aaron Bostrom, James Large, and Eamonn J. Keogh. The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. *Data Min. Knowl. Discov.*, 31(3):606–660, 2017.

[5] https://github.com/KULeuven-COSIC/TCHES20V3_CNN_SCA

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

[CEM18] Thomas De Cnudde, Maik Ender, and Amir Moradi. Hardware Masking, Revisited. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(2):123–148, 2018.

[CK09] Jean-Sébastien Coron and Ilya Kizhvatov. An efficient method for random delay generation in embedded software. In Christophe Clavier and Kris Gaj, editors, *Cryptographic Hardware and Embedded Systems - CHES 2009, 11th International Workshop, Lausanne, Switzerland, September 6-9, 2009, Proceedings*, volume 5747 of *Lecture Notes in Computer Science*, pages 156–170. Springer, 2009.

[CK14] Omar Choudary and Markus G. Kuhn. Template attacks on different devices. In Emmanuel Prouff, editor, *Constructive Side-Channel Analysis and Secure Design - 5th International Workshop, COSADE 2014, Paris, France, April 13-15, 2014. Revised Selected Papers*, volume 8622 of *Lecture Notes in Computer Science*, pages 179–198. Springer, 2014.

[CK18] Marios O. Choudary and Markus G. Kuhn. Efficient, Portable Template Attacks. *IEEE Trans. Information Forensics and Security*, 13(2):490–501, 2018.

[DBK+18] Hoang Anh Dau, Anthony J. Bagnall, Kaveh Kamgar, Chin-Chia Michael Yeh, Yan Zhu, Shaghayegh Gharghabi, Chotirat Ann Ratanamahatana, and Eamonn J. Keogh. The UCR Time Series Archive. *CoRR*, abs/1810.07758, 2018.

[DKK+18] Hoang Anh Dau, Eamonn Keogh, Kaveh Kamgar, Chin-Chia Michael Yeh, Yan Zhu, Shaghayegh Gharghabi, Chotirat Ann Ratanamahatana, Yanping, Bing Hu, Nurjahan Begum, Anthony Bagnall, Abdullah Mueen, Gustavo Batista, and Hexagon-ML. The UCR Time Series Classification Archive, October 2018. https://www.cs.ucr.edu/~eamonn/time_series_data_2018/.

[FFW+19] Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, and Pierre-Alain Muller. Deep learning for time series classification: a review. *Data Min. Knowl. Discov.*, 33(4):917–963, 2019.

[FLF+19] Hassan Ismail Fawaz, Benjamin Lucas, Germain Forestier, Charlotte Pelletier, Daniel F. Schmidt, Jonathan Weber, Geoffrey I. Webb, Lhassane Idoumghar, Pierre-Alain Muller, and François Petitjean. Inceptiontime: Finding alexnet for time series classification. *CoRR*, abs/1909.04939, 2019.

[HGG19] Benjamin Hettwer, Stefan Gehrer, and Tim Güneysu. Deep neural network attribution methods for leakage analysis and symmetric key recovery. In Kenneth G. Paterson and Douglas Stebila, editors, *Selected Areas in Cryptography - SAC 2019 - 26th International Conference, Waterloo, ON, Canada, August 12-16, 2019, Revised Selected Papers*, volume 11959 of *Lecture Notes in Computer Science*, pages 645–666. Springer, 2019.

[LBOM12] Yann LeCun, Léon Bottou, Genevieve B. Orr, and Klaus-Robert Müller. Efficient backprop. In Grégoire Montavon, Genevieve B. Orr, and Klaus-Robert Müller, editors, *Neural Networks: Tricks of the Trade - Second Edition*, volume 7700 of *Lecture Notes in Computer Science*, pages 9–48. Springer, 2012.

[LP07] Kerstin Lemke-Rust and Christof Paar. Gaussian mixture models for higher-order side channel analysis. In Pascal Paillier and Ingrid Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems - CHES 2007, 9th International Workshop, Vienna, Austria, September 10-13, 2007, Proceedings*, volume 4727 of *Lecture Notes in Computer Science*, pages 14–27. Springer, 2007.

[LS19] Zachary C. Lipton and Jacob Steinhardt. Troubling trends in machine learning scholarship. *ACM Queue*, 17(1):80, 2019.

[MBTL13] David P. Montminy, Rusty O. Baldwin, Michael A. Temple, and Eric D. Laspe. Improving cross-device attacks using zero-mean unit-variance normalization. *J. Cryptographic Engineering*, 3(2):99–110, 2013.

[MDB17] Gábor Melis, Chris Dyer, and Phil Blunsom. On the state of the art of evaluation in neural language models. *CoRR*, abs/1707.05589, 2017.

[MDP19] Loïc Masure, Cécile Dumas, and Emmanuel Prouff. Gradient visualization for general characterization in profiling attacks. In *COSADE*, volume 11421 of *Lecture Notes in Computer Science*, pages 145–167. Springer, 2019.

[MLB+17] Grégoire Montavon, Sebastian Lapuschkin, Alexander Binder, Wojciech Samek, and Klaus-Robert Müller. Explaining nonlinear classification decisions with deep taylor decomposition. *Pattern Recognit.*, 65:211–222, 2017.

[MMR20] Thorben Moos, Amir Moradi, and Bastian Richter. Static power side-channel analysis - an investigation of measurement factors. *IEEE Trans. Very Large Scale Integr. Syst.*, 28(2):376–389, 2020.

[NSGD12] Maxime Nassar, Youssef Souissi, Sylvain Guilley, and Jean-Luc Danger. RSM: A small and fast countermeasure for AES, secure against 1st and 2nd-order zero-offset SCAs. In Wolfgang Rosenstiel and Lothar Thiele, editors, *2012 Design, Automation & Test in Europe Conference & Exhibition, DATE 2012, Dresden, Germany, March 12-16, 2012*, pages 1173–1178. IEEE, 2012.

[NZ12] Ralph Neuneier and Hans-Georg Zimmermann. How to train neural networks. In Grégoire Montavon, Genevieve B. Orr, and Klaus-Robert Müller, editors, *Neural Networks: Tricks of the Trade - Second Edition*, volume 7700 of *Lecture Notes in Computer Science*, pages 369–418. Springer, 2012.

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

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

[PSB+18] Emmanuel Prouff, Rémi Strullu, Ryad Benadjila, Eleonora Cagli, and Cécile Dumas. Study of Deep Learning Techniques for Side-Channel Analysis and Introduction to ASCAD Database. *IACR Cryptology ePrint Archive*, 2018:53, 2018.

[RHMV19] Guillaume Richard, Georges Hébrail, Mathilde Mougeot, and Nicolas Vayatis. DenseNets for Time Series Classification: towards automation of time series pre-processing with CNNs. *Workshop on Mining and Learning from Time Series.*, 2019, 2019.

[SCD+17] Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, pages 618–626, 2017.

[SDBR15] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin A. Riedmiller. Striving for simplicity: The all convolutional net. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Workshop Track Proceedings*, 2015.

[SGK17a] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. Learning important features through propagating activation differences. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 3145–3153. PMLR, 2017.

[SGK17b] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. Learning important features through propagating activation differences. In *ICML*, volume 70 of *Proceedings of Machine Learning Research*, pages 3145–3153. PMLR, 2017.

[SLJ+15] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, pages 1–9. IEEE Computer Society, 2015.

[STY17] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. In *ICML*, volume 70 of *Proceedings of Machine Learning Research*, pages 3319–3328. PMLR, 2017.

[SVZ14] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. In *ICLR (Workshop Poster)*, 2014.

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

[ZBHV20] Gabriel Zaid, Lilian Bossuet, Amaury Habrard, and Alexandre Venelli. Methodology for efficient CNN architectures in profiling attacks. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(1):1–36, 2020.

[ZCAW17] Luisa M. Zintgraf, Taco S. Cohen, Tameem Adel, and Max Welling. Visualizing deep neural network decisions: Prediction difference analysis. *CoRR*, abs/1702.04595, 2017.

[ZF14] Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In David J. Fleet, Tomás Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *Computer Vision - ECCV 2014 - 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I*, volume 8689 of *Lecture Notes in Computer Science*, pages 818–833. Springer, 2014.

[ZT15] Jian Zhou and Olga G. Troyanskaya. Predicting effects of noncoding variants with deep learning–based sequence model. *Nat Methods*, 12:931–934, 2015.