Make Some Noise

Unleashing the Power of Convolutional Neural Networks for Profiled Side-channel Analysis

Kim, J.H.; Picek, Stjepan; Heuser, Annelie ; Bhasin, Shivam; Hanjalic, Alan

DOI
10.13154/tches.v2019.i3.148-179

Publication date
2019

Document Version
Final published version

Published in
IACR Transactions on Cryptographic Hardware and Embedded Systems

Citation (APA)
Kim, J. H., Picek, S., Heuser, A., Bhasin, S., & Hanjalic, A. (2019). Make Some Noise: Unleashing the Power of Convolutional Neural Networks for Profiled Side-channel Analysis. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, *2019*(3), 148-179. https://doi.org/10.13154/tches.v2019.i3.148-179

Make Some Noise
Unleashing the Power of Convolutional Neural Networks for Profiled Side-channel Analysis

Jaehun Kim[1], Stjepan Picek[1], Annelie Heuser[2],
Shivam Bhasin[3] and Alan Hanjalic[1]

[1] Delft University of Technology, Delft, The Netherlands
[2] Univ Rennes, Inria, CNRS, IRISA, France
[3] Physical Analysis and Cryptographic Engineering,
Temasek Laboratories at Nanyang Technological University, Singapore
j.h.kim@tudelft.nl,s.picek@tudelft.nl,annelie.heuser@irisa.fr,sbhasin@ntu.edu.sg,
a.hanjalic@tudelft.nl

Abstract. Profiled side-channel analysis based on deep learning, and more precisely Convolutional Neural Networks, is a paradigm showing significant potential. The results, although scarce for now, suggest that such techniques are even able to break cryptographic implementations protected with countermeasures. In this paper, we start by proposing a new Convolutional Neural Network instance able to reach high performance for a number of considered datasets. We compare our neural network with the one designed for a particular dataset with masking countermeasure and we show that both are good designs but also that neither can be considered as a superior to the other one.

Next, we address how the addition of artificial noise to the input signal can be actually beneficial to the performance of the neural network. Such noise addition is equivalent to the regularization term in the objective function. By using this technique, we are able to reduce the number of measurements needed to reveal the secret key by orders of magnitude for both neural networks. Our new convolutional neural network instance with added noise is able to break the implementation protected with the random delay countermeasure by using only 3 traces in the attack phase. To further strengthen our experimental results, we investigate the performance with a varying number of training samples, noise levels, and epochs. Our findings show that adding noise is beneficial throughout all training set sizes and epochs.

Keywords: Side-channel analysis, Convolutional Neural Networks, Machine learning, Gaussian noise

1 Introduction

Profiled side-channel analysis (SCA), especially machine learning-based techniques received a significant amount of attention in the SCA community lately. Such attention seems to be well-deserved since results show a number of situations where machine learning techniques perform (extremely) well and even surpass, for instance, the template attack [LBM14, PHG17, CDP17]. More recently, deep learning techniques (specifically, Convolutional Neural Networks – CNNs) emerged as a powerful alternative to more standard machine learning techniques when conducting side-channel analysis [MPP16, CDP17]. Due to a limited number of published works, it is not clear when deep learning is actually needed in practice and how to effectively use CNNs for SCA.

IACR Transactions on Cryptographic Hardware and Embedded Systems ISSN 2569-2925,
Vol. 2019, No. 3, pp. 148–179
DOI:10.13154/tches.v2019.i3.148-179

CNNs have shown great performance across different domains, like image classification, where it can work with thousands of features and classes and millions of examples [KSH12]. Still, there is one scenario usually not encountered in other domains but SCA: countermeasures. While it is normal (although undesired) to have noise in measurements coming from the environment, countermeasures are deliberately crafted in order to (ideally) prevent attacks by noise addition. Naturally, machine learning techniques, as well as traditional profiling methods, degrade in performance when dealing with implementations with countermeasures. Interestingly, some results indicate that deep learning is able to cope with countermeasures straightforwardly, which is a behavior not observed with other machine learning techniques, to the best of our knowledge. This holds for additional randomness [MPP16] (i.e., masking countermeasure) as well as additional noise [CDP17] (i.e., random delays). Still, deep learning architectures suitable for SCA, and particularly against practical countermeasures, are not well explored.

One important drawback of machine learning techniques and particularly deep learning is the problem of overfitting. Overfitting occurs when the profiling is done "too well", such that the model is predicting the training data very precisely but performs poorly on unseen measurements in the attacking phase. Even though overfitting has a huge influence on the attack efficiency, to the best of our knowledge this problem received limited attention in the SCA community so far.

In this paper, we propose two approaches to achieve high performance with neural networks in SCA where the end goal is to be able to break cryptographic implementations protected with countermeasures. We start with a proposal for a new CNN instance that is able to reach high performance. Next, to boost even further that performance, we show how the addition of noise to the original measurements, without adding new artificial instances by precomputation and using domain-specific knowledge, can improve the behavior of CNN significantly as it prevents overfitting. To the best of our knowledge, this has not been done before in side-channel analysis and it represents a powerful option when using machine learning techniques in SCA. This is particularly interesting since usually, noise is a limiting factor and actually, many types of countermeasures could be regarded as the noise. Finally, we discuss what a new deep learning architecture actually is and why it is important to differentiate it from a new instantiation of a known architecture. Since deep learning in SCA is a relatively novel research direction, we believe such discussion is a necessity that has pedagogical value and will keep future work more inline with the terminology in other domains.

1.1 Related Work

When considering profiled SCA, the template attack (TA) is the best (optimal) technique from an information theoretic point of view if the attacker has an unbounded number of traces [HRG14, LPB$^+$15]. After the template attack, the stochastic attack emerged using linear regression in the profiling phase [SLP05]. In years to follow, researchers recognized certain shortcomings of template attacks and tried to modify them in order to deal better with the complexity and portability issues. One example of such an approach is the pooled template attack where only one pooled covariance matrix is used in order to cope with statistical difficulties [CK13].

Alongside such techniques, the SCA community recognized the similarity between profiling and supervised machine learning. Consequently, some researchers started experimenting with different machine learning techniques and evaluating their effectiveness in the SCA context. As a consequence, there exists an abundance of works considering profiled machine learning-based attacks and block ciphers. There, the most common denominator is the target of attacks – AES. Next, we can divide machine learning-based attacks on those that investigate regular machine learning techniques and those that also use deep learning (in this category, it is still common to compare with regular machine learning techniques.) For the examples of the first category, see e.g., [HZ12, HGDM+11, LBM14, LPB+15, LBM15, PHG17, PHJL17, PHJ+17]. When considering deep learning, the common examples in SCA are multilayer perceptron and Convolutional Neural Networks [GHO15, HPGM17, MPP16, CDP17, PSK+18, PSB+18].

1.2 Our Contributions

This paper investigates the limits of CNNs' performance when considering side-channel analysis. To that end, we consider the AES algorithm and investigate 4 datasets that encompass the main types of measurements as detailed in Section 4. Our main contributions are:

1. a new CNN instance derived from the neural network proposed in the audio domain, which is the 1-dimensional analog to the *VGG* architecture (Section 3.1). We experimentally show that the proposed CNN instance is able to efficiently find the secret key (Section 5). This indicates that CNN architectures developed for other domains (but where the input signal shares commonalities with the SCA domain) can be successfully applied to the SCA domain.
2. investigation how additional, non-task-specific noise can significantly improve the performance of CNNs in SCA (Section 3.2). While adding noise to fight overfitting is a well-known technique in machine learning, we show it also helps in the specific case – SCA. In related works, the authors used domain-specific information when adding noise, but we show that a simple Gaussian noise not only works but can surpass the effect of domain-specific noise. With our new CNN instance and added noise, we are able to break the dataset with the random delay in only a few measurements. More specifically, we need 1 trace to reach guessing entropy less than 10, 2 traces for guessing entropy less than 5, and 3 traces for guessing entropy of 1. To the best of our knowledge, these results surpass by far any in related literature. Moreover, we conduct an in-depth study of the effects of samples in the training dataset, noise level, and epochs. Interestingly, noise addition has a positive effect on all noise levels and epochs.

Besides these contributions, we also show how random splitting of training data results in a drastically different behavior for both TA and neural networks. Finally, we discuss the difference between CNN architectures and instances (Section 6.2). We consider this to be of high importance in order to facilitate future research and comparisons.

2 Background

In this section, we start by introducing the notation we use and then we discuss the profiled side-channel analysis. Afterward, we give details about template attack and convolutional neural networks. Finally, we briefly describe the CNN instance introduced in [PSB+18].

2.1 Notation

Let k^* be the fixed secret cryptographic key (byte), k any possible key hypothesis, and t a plaintext or ciphertext byte of the cryptographic algorithm, which is uniformly chosen. We denote the measured multivariate leakage as $\vec{X} = X_1, \ldots, X_D$, with D being the number of time samples or points-of-interest (i.e., features as called in the machine learning domain). To guess the secret key, the attacker first needs to choose a leakage model $Y(T, k)$ (i.e., label as called in the machine learning domain) depending on the key guess k and on some known text T, which relates to the deterministic part of the leakage. When there is no ambiguity, we write Y instead of $Y(T, k)$.

2.2 Profiled Side-channel Analysis

We consider a scenario where a powerful attacker has a device with knowledge about the secret key implemented and is able to obtain a set of profiling traces in order to characterize the leakage. Once this phase is done, the attacker measures additional traces from the device under attack in order to break the unknown secret key k^*. Although it is usually considered that the attacker has an unlimited number of traces available during the profiling phase, this is of course always bounded, due to practical limitations.

When profiling the leakage, one must choose the number of classes. The Hamming weight (HW) or distance (HD) of sensitive intermediate value are often used as the basis for choosing a number of classes, which has its own advantages and disadvantages. The advantage of choosing HW/HD is a reduced number of classes, which results in reduced training complexity. Unfortunately, HW/HD classes are inherently imbalanced which can seriously hamper the performance of machine learning/deep learning [PHJ$^+$18]. The alternative is to profile directly on the intermediate value which requires a bigger profiling set but does not suffer from imbalance. When considering AES, HW results in 9 classes and intermediate value results in 256 classes as the attack is computed byte-wise. Since 256 classes can still be considered small as compared to the usual application of deep neural networks, we use 256 classes to avoid the imbalance problem.

To assess the performance of the classifiers, we use Guessing entropy (GE) [SMY09]. GE states the average number of key candidates an adversary needs to test to reveal the secret key after conducting a side-channel analysis. In particular, given T_a amount of samples in the attacking phase, an attack outputs a key guessing vector $\mathbf{g} = [g_1, g_2, \ldots, g_{|\mathcal{K}|}]$ in decreasing order of probability with $|\mathcal{K}|$ being the size of the keyspace. So, g_1 is the most likely and $g_{|\mathcal{K}|}$ the least likely key candidate. The guessing entropy is the average position of k_a^* in $\mathbf{g}$ over multiple experiments. Note that we consider leakage models $Y(\cdot)$ which are bijective functions (see Section 4 for definitions regarding each dataset), and thus each output probability calculated from the classifiers for $Y(k)$ directly relates to one single key candidate k. Further, to calculate the key guessing vector $\mathbf{g}$ over T_a amount of samples, we use the log-likelihood principle, i.e.,

$$g_i = \sum_{j=1}^{T_a} \log(\hat{p}_{ij}), \tag{1}$$

where $\hat{p}_{ij}$ is the estimated probability for key candidate $i = \{1, \ldots, |\mathcal{K}|\}$ using sample j.

2.3 Convolutional Neural Networks – CNNs

CNNs represent a type of neural networks which were first designed for 2-dimensional convolutions as it was inspired by the biological processes of animals' visual cortex [LB$^+$95]. They are primarily used for image classification but lately, they have proven to be powerful classifiers for time series data such as music and speech [ODZ$^+$16]. From the operational perspective, CNNs are similar to ordinary neural networks (e.g., multilayer perceptron): they consist of a number of layers where each layer is made up of neurons. CNNs use three main types of layers: convolutional layers, pooling layers, and fully-connected layers.

Convolutional layers are linear layers that share weights across space, where the core operation is the convolution between the filter bank and signal as follows:

$$(\phi * x)(t) = \sum_{a=-\infty}^{\infty} x(t)\phi(t-a) \tag{2}$$

where t indicates each time step, and $\phi \in \mathbb{R}^{i \times o \times s}$ refers a filter bank with i input channels and o output channels, and s filter length respectively. Pooling layers are non-linear layers that reduce the spatial size in order to limit the number of neurons, typically by applying various types of down-sampling the given input over spatial axes. Finally, fully-connected layers are layers where every neuron is connected with all the neurons in the neighborhood layer, which usually is realized by dot product between the weight matrix and the input vector. For additional information about CNNs, we refer interested readers to [GBC16].

The first results in the SCA domain have shown a significant promise [MPP16, CDP17]. Considering the signal given in SCA is the 1-dimensional signal that shares a substantial amount of characteristics with other 1-dimensional signals such as audio, it implies that if a certain architecture yields a reasonable performance in one field, it might be transferable to the other. To our knowledge, most prominent CNN architectures reported in side-channel analysis domain can be seen as the variants of a specific design principle, which is derived from the particular design strategy introduced in [SZ14], which is often referred as *VGG-like*. The main difference compared to the original architecture is the spatial dimensionality on each convolution and pooling operation, since the model is developed for the image classification tasks where the input signal has 2 spatial dimensions, while the signal given in SCA has only 1 spatial dimension. Still, the core design principle is identical, where the network `net` can be formalized as a following:

$$\texttt{net} = \texttt{fc}_{\theta,\texttt{softmax}} \circ \prod_{p=1}^{P} \texttt{fc}_{\theta^p,\texttt{ReLU}} \circ \prod_{q=1}^{Q} \left(\texttt{pool}_{\texttt{Max}} \circ \prod_{r=1}^{R_q} \texttt{conv}_{\phi^r,\texttt{ReLU}}\right), \tag{3}$$

where P, Q, R_q represent the number of fully-connected layers `fc`, convolution blocks, and convolution layers `conv` in q^{th} convolution block, respectively. The "convolution block" refers to the set of chained operations within the parenthesis in Eq. (3) that consists of R_q convolution layers $\texttt{conv}_{\phi,\sigma}$ and one or less pooling layer `pool`. $\texttt{pool}_t$ indicates the pooling function that sub-samples the signal in its spatial axes in specific pooling strategy t. Typically, *max pooling* that sub-samples the maximum value within the sliding pooling window is applied for many applications [GBC16]. $\texttt{conv}_{\phi,\sigma}$ and $\texttt{fc}_{\theta,\sigma}$ are convolution and fully-connected layers, respectively, which are followed by the non-linear transformation given as follows:

$$\texttt{conv}_{\phi,\sigma}(X) = \sigma(\phi * X), \text{and} \tag{4}$$

$$\texttt{fc}_{\theta,\sigma}(x) = \sigma(\theta^{\intercal} x), \tag{5}$$

where $X \in \mathbb{R}^{i \times d}$ is the input signal that has i channels and length d. $x \in \mathbb{R}^f$ denotes the input vector and $\theta \in \mathbb{R}^{f \times h}$ is the projection matrix that transforms f dimensional input to h dimensional output space. Note that we omit the bias terms for simplicity. σ is the non-linearity function applied after the output of affine transformation, including the activation functions such as rectified linear unit (`ReLU`) or `softmax`, which are formalized as follows, respectively:

$$\sigma_{\texttt{ReLU}}(z)_j = max(0, z_j), \tag{6}$$

$$\sigma_{\texttt{softmax}}(z)_j = \frac{e^{z_j}}{\sum_{f=1}^{F} e^{z_f}} \quad \text{for } j = 1, 2, ..., F, \tag{7}$$

with F being the hidden or output dimensionality of the given layer. Commonly, `ReLU` is used for the hidden layers, and `softmax` is used for the final layer to represent the posterior distribution over each class labels.

2.4 ASCAD Neural Network

To serve as a baseline in our experiments, we use the CNN model introduced in [PSB+18]. For brevity, we refer this model as the ASCAD model in the rest of the paper. This model is developed on the dataset which is also introduced in the paper, by grid search on a number of important model hyper-parameters. One of the biggest difference between this model and the CNN model we present it this paper is the size and depth of the network.

2.5 Template Attack

The template attack [CRR02], which is classical profiling based side-channel attack, relies on the Bayes theorem and considers dependence among the features. In the state-of-the-art, it is mostly assumed that the noise follows a multivariate normal distribution. Accordingly, for each class y an attacker only needs to estimate one mean vector and a covariance matrix to define the distribution of the leakage per class y. The authors of [CK13] propose to use only one pooled covariance matrix averaged over all classes Y to cope with statistical difficulties and thus a lower efficiency. As a comparison to attacks based on neural networks, we use the pooled template attack for all datasets which do not include a masking countermeasure. In the case of masking, we apply the standard version of the template attack as each covariance matrix may not only include noise but also information about the class y and therefore the secret key.

3 CNN & Artificial Noise

In this section, we first present details about our convolutional neural network instance. Afterward, we detail our approach to add noise to the measurement data in order to avoid overfitting and achieve generalization.

3.1 The Design Principle: Sample-level CNN

We adopted the more specific design strategy introduced in [KLN18]. This strategy is the 1-dimensional analog to the *VGG* architecture, characterized by the use of the minimal filter size and highly stacked convolution layers followed by several fully-connected layers. The CNN based on this strategy exhibits better performance in the music-related classification tasks [KLN18]. Surprisingly, by grid search and cross-validation, the authors in [PSB+18] reach a very similar architecture setup compared to our work, while this network is developed on the side-channel analysis dataset. It implies that this design is potentially suitable for a various range of problem domains where the input signal is given as the 1-dimensional signal.

Considering evidence given in the literature, we choose this design to build network architecture. One detailed example of our architecture can be found in Figure 1. The key principles of the applied design strategies are

- a minimal filter size (3),
- the convolution block continues until the spatial dimension is reduced to 1,
- the number of channels starts from a small number and keeps increasing (similarly to *VGG*).

In addition to the above standard architecture, we applied two additional components to improve the model: batch normalization [IS15] and dropout [SHK+14]. The batch normalization is the layer-wise normalization followed by the parametric re-adjustment, which is known to be helpful for faster learning by stabilizing internal covariate shifts [IS15]. Dropout is a simple but powerful technique for the regularization of the model, which simply introduces randomly generated masks on the activation of neurons. Dropout intends to reduce the influence of the data in the training dataset on the model and therefore

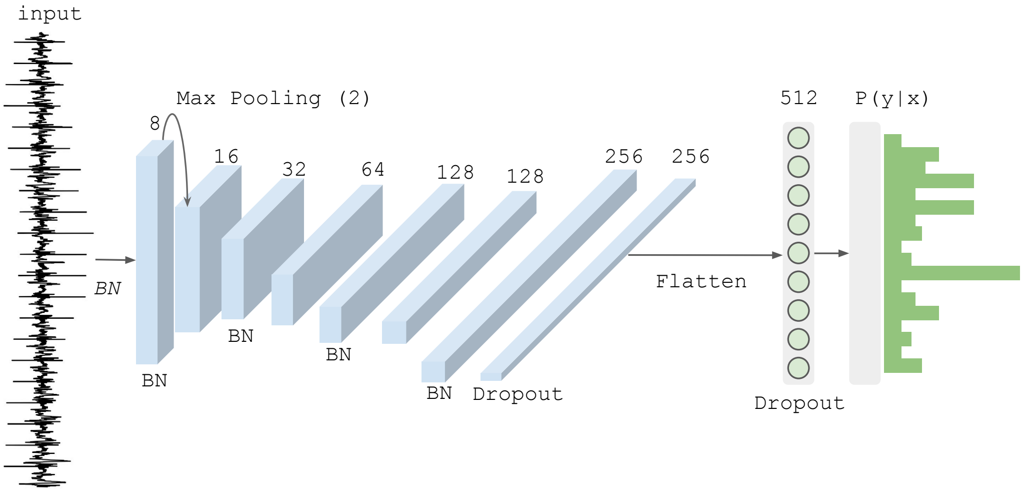


Figure 1: An example schema of the CNN architecture. This particular configuration is used for the AES HD dataset. The model encodes a trace into latent representation such that the final layer can output the posterior probability for each label. The blue tensors indicate the activation after the convolution layer $\texttt{conv}_{\phi^m,\texttt{ReLU}}$. The number on top of each block indicates the number of channels used for the corresponding convolution layer. 'BN' indicates that the batch normalization is applied to the corresponding layer, and 'Dropout' is used in a similar manner referring to the dropout layer. The gray box with green circles refers to the fully-connected layer $\texttt{fc}_{\theta^n,\texttt{ReLU}}$. Finally, the rightmost component in the pipeline is the output of the final fully-connected layer $\texttt{fc}_{\theta,\texttt{softmax}}$ that outputs the probability for each 256 label. As it can be seen, a convolutional neural network is a sequence of layers, and every layer of a network transforms one volume of activation functions to another through a differentiable function. Input holds the raw features. Convolution layer computes the output of neurons that are connected to local regions in the input, each computing a dot product between their weights and a small region they are connected to in the input volume. `ReLU` layer will apply an element-wise activation function, such as the $max(0, x)$ thresholding at zero. Max Pooling performs a down-sampling operation along the spatial dimensions. The fully-connected layer computes either the hidden activations or the class scores. Batch normalization is used to normalize the input layer by adjusting and scaling the activations after applying standard scaling using running mean and standard deviation. To convert the output of a convolution part of CNN (which is 2-dimensional) into a 1-dimensional feature vector that is used in the fully-connected layer, we use the flatten operation. Dropout is a regularization technique for reducing overfitting by preventing complex co-adaptations on training data. The term refers to dropping out units (both hidden and visible) in a neural network.

also helps to prevent overfitting [SHK+14]. We applied the batch normalization for every odd-numbered layers and the input channels, to standardize the input signal channel-wise. The dropout is only applied after the fully-connected layers since convolution layer already has a relatively smaller number of parameters. Finally, we employed the L2 regularization on all of the weight with a small coefficient (10^{-7}) for all models. In order to allow easier notation and comparison, we denote this neural network as the RD network.

Note that the ASCAD network has approximately 130 times larger number of parameters than the RD network, mostly due to the fully-connected layers with large dimensionality in its hidden units. Still, in terms of depth, the RD model is deeper, which means that more abstractions can be conducted by the convolution blocks.

3.2 Adding Artificial Noise to the Input

To reduce the overfitting of the model against the noise (additionally to dropout), we apply a further regularization technique by introducing noise to the traces in the training phase. It is known that adding noise in the input data is equivalent to adding a regularization term to the objective function [Bis95, RGBV11]. Generally speaking, it can be seen as "blurring" high-frequency patterns that are more frequently appearing in the data, which will eventually help the model to concentrate more on low-frequency patterns that are often the more important ones for a given task. Since in our case, the input normalization is also learned during the training process via the batch normalization layer, we added the noise tensor Ψ after the first batch normalization as follows:

$$X^* = BN_0(X) + \Psi, \quad \Psi \sim \mathcal{N}(0, \alpha) \tag{8}$$

where BN_0 indicates the batch normalization applied on the input X. After tuning, we draw a noise tensor Ψ following a normal distribution, i.e., with zero mean and standard deviation $\alpha \in [0, 1]$. The noise tensor is added to the output of the batch normalization. The noise addition for the ASCAD network follows similarly.

Different values for the weight α in Eq. (8) are further discussed and evaluated in Section 5. After the noise injection, deformed signal X^* is processed by upper layers of `net`.

Note, in the related works [CDP17, PYW+18], the authors use the notion of 'data augmentation' to describe the procedure where new measurements are constructed from the original measurements by some data transformation. Then, both original and transformed data are used in the 'augmented' profiling set. Differing from that, we transform our dataset by adding noise where we only use the transformed data in the profiling phase. Additionally, the noise is dynamically applied to every random batch during the training, since the generation of Gaussian noise is fast enough to be included in the training iteration.

Consequently, our technique is a regularization technique and can be seen as 1) a noisy training such as 'dropout' or 'stochastic optimization', and 2) 'data augmentation'. In our opinion, it is closer to the noisy training (i.e., regularization using noisy training). Data augmentation typically applies domain knowledge to deform the original signal into more "plausible" way (as also used in [CDP17, PYW+18]), and uses both the original and transformed measurements in the training phase. In our work we use neither: 1) we do not add transformed measurements and 2) do not use domain-specific knowledge (e.g., knowledge about countermeasures).

3.3 Training procedure

The CNN model is trained in an iterative manner, which is illustrated in Algorithm 1. The outer training loop runs to the fixed number of iterations or 'epochs' I, where an epoch is finished when all data points in the training set have been visited in the inner loop. To update the model such that the accuracy is maximized, we choose to minimize the cross-entropy as the main objective function:

$$CrossEntropy(y, \hat{y}) = -\frac{1}{N}\sum_{n=1}^{N}\sum_{c=1}^{C} y_{n,c} log(\hat{y}_{n,c}), \tag{9}$$

where $\hat{y}_{n,c} = \texttt{net}(X_n)$ is the predicted posterior distribution obtained from the network `net` given trace X_n, and $y_{n,c} = \mathbb{1}(X_n; c)$ denotes the binary indicator for corresponding ground truth of given trace X_n that belongs to the class c. N denotes the number of samples in the training dataset and C denotes the number of classes, which is $|\mathcal{K}|$ in SCA context. The parameters are updated in a direction of minimizing the loss function, by the stochastic gradient descent method with a given learning rate ϵ.

To obtain the model that optimally minimizes the generalization error, we adopted a simple algorithm that keeps and returns the best model throughout the training. It can be achieved by monitoring the model at the end of every inner training loop by computing the 'validation error' using the held-out dataset. The best model is updated if the validation error is decreased at the corresponding iteration, or maintained if the error is increased or the same as in the previous iteration. In our experiments, we set the validation accuracy as the monitoring metric.

Algorithm 1: Training a CNN

```
Initialize $\Theta$: $\{\theta, \theta^p, \phi^r\}$ randomly;
Initialize $\mathcal{A} \leftarrow \Theta$ to store best model;
for epoch in 1...I do
    while not all training traces are visited do
        Pick next batch of samples $(X, y)$;
        Forward-pass:;
        $\mathcal{L}(y, X, \Theta) = CrossEntropy(y, \texttt{net}(X; \Theta))$;
        Backward-pass: $\nabla(\Theta)$;
        Update model: $\Theta \leftarrow ADAM(\Theta, \nabla(\Theta), \epsilon)$;
    end
    Calculate accuracy with held-out data;
    if improved then
        $\mathcal{A} \leftarrow \Theta$;
    end
end
```

3.4 Experimental Details

As already introduced in Section 3.1, the detailed configuration can differ across datasets since it depends on the spatial shape of the input signal. The detailed configuration of the models dedicated for each dataset can be found in Appendix A, Table 2. We initialize the weights to small random values and we use the "ADAM" optimizer. We choose to use ADAM because it is shown in practice that gives good results while being robust to hyper-parameter settings [KB14]. In this work, we ran the experiment with computation nodes equipped with 32 NVIDIA GTX 1080 Ti graphics processing units (GPUs). Each of it has 11 Gigabytes of GPU memory and 3 584 GPU cores. Specifically, we implement the experiment with the PyTorch [PGC+17] computing framework in order to leverage the GPU computation.

4 Datasets

We consider four datasets covering the main types of SCA scenarios. The first dataset considered has no countermeasures and a small level of noise. This dataset represents an ideal attack target (and one not seen often in reality) and is a good indicator of the best possible behavior of the attack technique. The second dataset again does not have any countermeasures but has a high level of noise. Consequently, it represents a difficult target for profiled techniques since the high level of noise makes the boundaries among classes fuzzy. The third dataset implements a random delay countermeasure, which makes it a realistic example of a dataset one could encounter in actual implementations. Finally, the last dataset has a masking countermeasure, which is probably the most widespread technique of SCA protection nowadays.

4.1 DPAcontest v4 Dataset

DPAcontest v4 provides measurements of a masked AES software implementation [TEL14]. As the masking is found to leak first-order information [MGH14], we consider the mask to be known and thus turn the implementation into an unprotected scenario. It is a software implementation with most leaking operation not being the register writing but the processing of the S-box operation and we attack the first round. Accordingly, the leakage model changes to

$$Y(k^*) = \texttt{Sbox}[P_i \oplus k^*] \oplus \underbrace{M}_{\text{known mask}} , \tag{10}$$

where P_i is a plaintext byte and we choose $i = 1$. The measured signal to noise ratio (SNR) attains a high maximum value of 5.8577. The measurements consist of 4 000 features around the S-box part of the algorithm execution. This dataset is available at http://www.dpacontest.org/v4/.

4.2 Unprotected AES-128 on FPGA (AES_HD) Dataset

Next, we target an unprotected hardware implementation of AES-128. AES-128 core was written in VHDL in a round based architecture, which takes 11 clock cycles for each encryption. The AES-128 core is wrapped around by a UART module to enable external communication. It is designed to allow accelerated measurements to avoid any DC shift due to environmental variation over prolonged measurements. The total area footprint of the design contains 1 850 LUT and 742 flip-flops. The design was implemented on Xilinx Virtex-5 FPGA of a SASEBO GII evaluation board. Side-channel traces were measured using a high sensitivity near-field EM probe, placed over a decoupling capacitor on the power line. Measurements were sampled on the Teledyne LeCroy Waverunner 610zi oscilloscope. A suitable and commonly used distance leakage model when attacking the last round of an unprotected hardware implementation is the register writing in the last round [TEL14], i.e.,

$$Y(k^*) = \underbrace{\texttt{Sbox}^{-1}[C_i \oplus k^*]}_{\text{previous register value}} \oplus \underbrace{C_j}_{\text{ciphertext byte}} , \tag{11}$$

where C_i and C_j are two ciphertext bytes, and the relation between i and j is given through the inverse ShiftRows operation of AES. We choose $i = 12$ resulting in $j = 8$ as it is one of the easiest bytes to attack. These measurements are relatively noisy and the resulting model-based SNR (signal-to-noise ratio), i.e., $\frac{var(signal)}{var(noise)} = \frac{var(y(t,k^*))}{var(x-y(t,k^*))}$, with a maximum value of 0.0096. In total, 1 000 000 traces were captured corresponding to 1 000 000 randomly generated plaintexts, each trace with 1 250 features. As this implementation leaks in HD model, we denote this implementation as AES_HD. This trace set is publicly available at https://github.com/AESHD/AES_HD_Dataset.

4.3 Random Delay Countermeasure (AES_RD) Dataset

As a use case for countermeasures, we use protected (i.e., with a countermeasure) software implementation of AES. The target smartcard is an 8-bit Atmel AVR microcontroller. The protection uses random delay countermeasure as described by Coron and Kizhvatov [CK09]. Adding random delays to the normal operation of a cryptographic algorithm has as an effect on the misalignment of important features, which in turns makes the attack more difficult to conduct. As a result, the overall SNR is reduced. We mounted our attacks against the first AES key byte, targeting the first S-box operation, i.e.,

$$Y(k^*) = \texttt{Sbox}[P_i \oplus k^*], \tag{12}$$

with $i = 1$. The dataset consists of 50 000 traces of 3 500 features each. For this dataset, the SNR has a maximum value of 0.0556. Recently, this countermeasure was shown to be prone to deep learning based side-channel [CDP17]. However, since its quite often used countermeasure in the commercial products, while not modifying the leakage order (like masking), we use it as a target case study. The trace set is publicly available at https://github.com/ikizhvatov/randomdelays-traces. In the rest of the paper, we denote this dataset as the AES_RD dataset.

4.4 ASCAD Dataset

Finally, we test our architecture on the recently available ASCAD database [PSB+18]. The target platform is an 8-bit AVR microcontroller (ATmega8515) running a masked AES-128 implementation and measurements are made using electromagnetic emanation.

The dataset follows the MNIST database and provides 60 000 traces, where originally 50 000 traces were used for profiling/training and 10 000 for testing. We use the raw traces and use the pre-selected window of 700 relevant samples per trace corresponding to masked S-box for $i = 3$. As a leakage model we use the unprotected S-box output, i.e.,

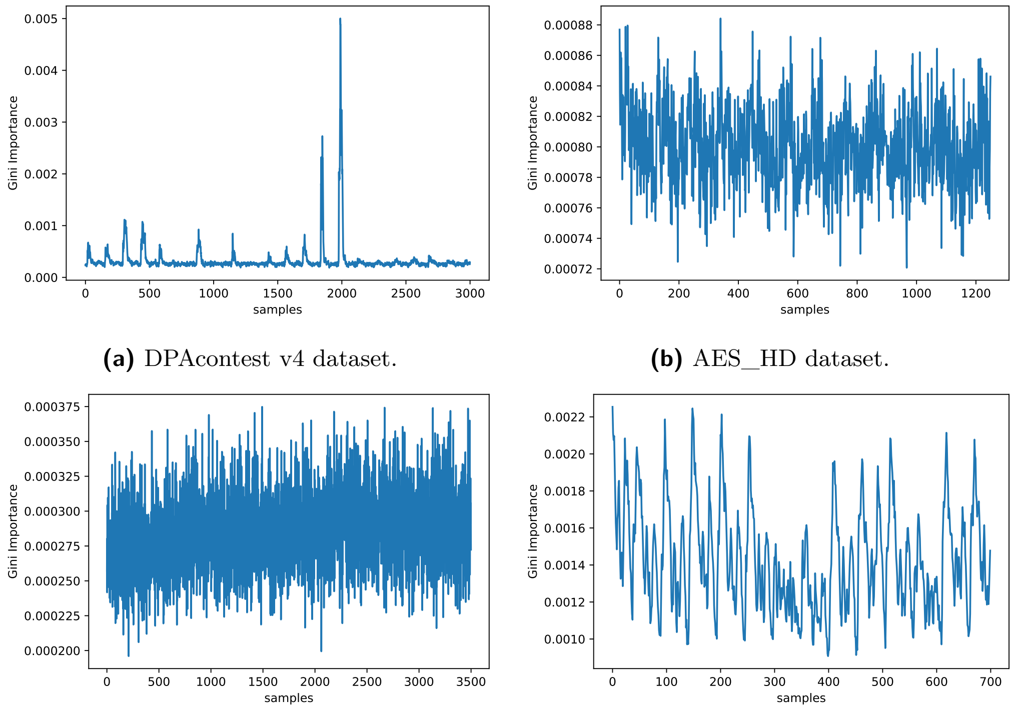
$$Y(k^*) = \texttt{Sbox}[P_i \oplus k^*]. \tag{13}$$

Interested readers can find more information about this dataset at [PSB+18]. This dataset is available at https://github.com/ANSSI-FR/ASCAD. Note that, the model in Eq. (13) does not leak information directly as it is first-order protected and we, therefore, do not state a model-based SNR. The SNR for the ASCAD dataset is ≈ 0.8 under the assumption we know the mask while it is almost 0 with the unknown mask.

4.5 Feature Importance

In Figure 2, we depict the most important features for all considered dataset. To investigate the feature relevance, we use the Random Forest (RF) classifier [Bre01]. Random Forest algorithm enables easy measuring of the relative importance of each feature on prediction. Random Forest (RF) is an ensemble decision tree learner composed of a number of decision trees [Bre01]. A decision tree is a classifier expressed as a recursive partition of the instance space. The decision tree consists of a number of internal nodes. An internal node represents a feature, a branch is the outcome of a condition tested on a node, and a leaf is a node without children. A leaf represents the class label, i.e., the decision taken after computing all features. Decision trees choose their splitting features from a random subset of k features at each internal node. The best split is taken among these randomly chosen features and the trees are built without pruning. Random Forest is a stochastic algorithm because of its two sources of randomness: bootstrap sampling and feature selection at node splitting. Note, RF classifier is often used in the SCA context with very good results, see e.g., [MPP16, PHJ+18]. For this experiment, we use 500 trees, with no limit to the tree size. Note that on the y-axis, we depict the Gini importance. Gini Importance (Mean Decrease in Impurity) calculates each feature importance as the sum over the number of splits and across all trees that include the feature, proportionally to the number of samples it splits.

For DPAcontest v4, there is a region containing all the relevant features. For the ASCAD dataset, we still observe some regions with more important features while for the AES_HD and AES_RD datasets, there appears to be no single region where important features are concentrated. This means that for the DPAcontest v4 dataset, one could easily preselect only the most important features. For other datasets, such step is much more difficult and for simpler machine learning techniques (or pooled TA) it is problematic to cope with a large number of features. Since CNNs work with raw features, they are able to use the wide-spread information much better. Note, one option to reduce the number of features and yet use the information from the whole trace is to use PCA.



(a) DPAcontest v4 dataset.

(b) AES_HD dataset.

(c) Random Delay dataset.

(d) ASCAD dataset.

Figure 2: The feature importance derived from the Random Forest classifier. A higher value indicates corresponding feature dimension is relatively more important than the others. The values are normalized such that the sum of all the importance equals 1.

In Table 1, we sum up the details about the datasets as used in our experiments. Note that, while the CNN architectures take as input the complete measurement trace and therefore all features, TA (pooled) needs a pre-selection. For the datasets of DPAcontest v4, AES_HD, and AES_RD we select 50 features with the highest absolute correlation coefficient between the measurement traces and the corresponding leakage model $Y(k^*)$. As for the ASCAD database the leakage model is not directly correlated with the measurement traces, we follow the approach in [PSB+18] and perform a PCA and select the 15 most

Table 1: Statistical information for all considered datasets. Note, for ASCAD, we show two values for SNR where the first one is for the scenario if the mask is known while the second value is for the unknown mask.

Dataset	Nr measurements	Nr features	SNR	Countermeasure
DPAcontest v4	100 000	4 000	5.8577	–
AES_HD	100 000	1 250	0.0096	–
AES_RD	50 000	3 500	0.0556	Random delay
ASCAD	60 000	700	$\approx 0.8/0$	Masking

relevant components.

4.6 Data Preparation

When considering machine learning techniques, we divide the dataset into training, validation, and testing parts. The testing set in our experiments has 25 000 measurements while the training set equals $0.8 \cdot (T - 25\,000)$, where T is the total size of the dataset. The validation set equals $0.2 \cdot (T - 25\,000)$. We use the 10 randomized splits for the experiments. For every trial, we randomly choose validation and training traces out of the entire trace pool except the held-out test traces. The obtained results are then averaged to produce the final estimation.

This particular splitting method often called as *repeated random subsampling* or *Monte Carlo cross-validation* [Sim07], has an advantage that the number of samples in the validation set is not dependent on the number of trials since in this case, the training and validations sets across the trials are not disjoint as the k-fold cross-validation. Compared to the relatively large number of classes we need to deal with, a number of our benchmark datasets contain limited training sets after holding out 25 000 testing traces. This makes the number of validation traces even smaller to the point where we have only approximately 10 traces per class if 10-fold cross-validation is applied. To reduce the instability occurring by a small validation set, we decided to increase the validation ratio to 20%, which again limits the number of folds to 5 when k-fold cross-validation is applied. By applying the repeated random sampling strategy, we could test our model with a larger number of validation traces per class with 10 folds, which is suggested in [Sim07], where the variability across the folds is sufficiently reduced. Since for TA we do not require a validation set, the training set simply equals $T - 25\,000$.

5 Experiments

In this section, we start by providing results for our RD neural network, ASCAD neural network, and template attack (pooled) for two noise levels: $\alpha = 0$ and $\alpha = 0.5$. Afterward, we give additional experiments with different levels of noise in order to explore its limitations. As for the training, we run both of the neural networks for 75 epochs per each dataset, except the AES_HD dataset, where the networks are trained for 300 epochs. For every training, we monitor the validation accuracy to keep the best model in terms of validation performance during the training even for the case when the validation performance is decreased due to the overfitting. Finally, we explore the influence of the level of noise, the number of epochs, and the profiling dataset size on the performance of the attack.

5.1 Results

When presenting the results, we adopt the following setting: we present the results for each fold as well as the averaged results. Note, the averaged results are what we usually use in SCA to assess the performance of a machine learning-based classifier. We believe this not to be a requirement and we discuss in Section 6.3 how the results of each fold could be used in a practical attack.

5.1.1 DPAcontest v4 Dataset

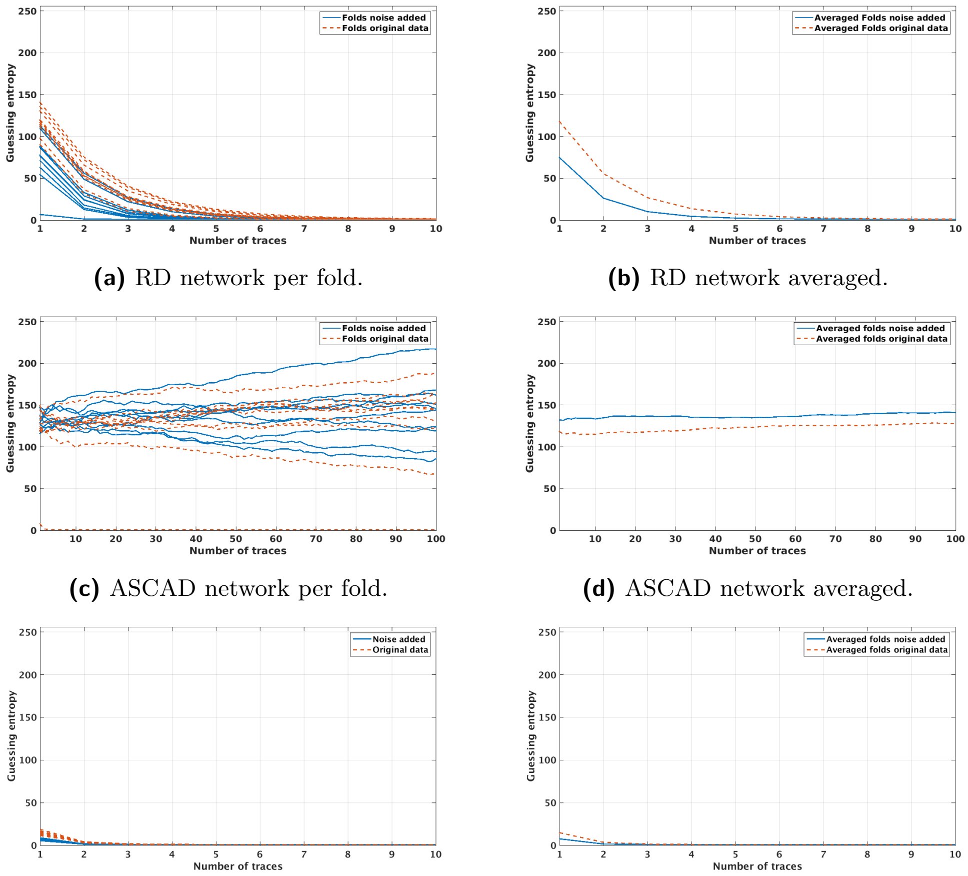
The results on the first dataset we consider are given in Figure 3 and represent DPAcontest v4, which is the easiest dataset since we consider it without countermeasure and the measurements have a small level of noise. The RD network is able to reach very good behavior where it requires only a few traces to break this implementation. Next, we see that the behavior after noise addition improved significantly. More precisely, GE results for RD network with noise addition improve by almost double when compared to the results without artificial noise addition. When looking at each fold separately, we see that most of the folds with noise are better than the folds without noise. Interestingly, there is one fold that is several times better than the average behavior with noise. For the ASCAD network, we see that DPAcontest v4 is not an easy dataset. In fact, we do not see any decrease in GE with the increase in the number of traces. Differing from the RD network, here we see that adding noise actually even slightly decreases the performance. When considering each fold, we see a number of folds improving with the increase in the number of traces and added noise but we also see opposite behavior for certain folds. For the pooled TA, we see that the addition of noise helps and actually we are able to break this implementation with only 2 traces. Finally, we see that the TA behavior per fold is consistent with the averaged results and that all folds with added noise perform better when compared to the scenario without noise.

5.1.2 AES_HD Dataset

In Figure 4, we give the results for AES_HD, which exhibits quite a different behavior from DPAcontest v4. Still, this is not unexpected due to a much higher level of environmental and algorithmic noise. For the RD network, we see for a number of folds very good behavior but unfortunately, there are some folds that actually deteriorate with the increase in the number of traces. Note that, this behavior might be still exploitable (similar to ghost peaks for correlation [BCO04]) or even be preventable. When averaged, we see a slow improvement with the increase in the number of traces but GE stays quite high even for the complete test set. The addition of noise helps only when the number of traces is small and there is a point around 15 000 traces where the behavior is the same for the network with and without noise. A very different behavior is seen for the ASCAD network. Here, we see that for all folds GE reduces with the increase in the number of traces and that artificial noise is beneficial for breaking this dataset. When comparing the behaviors of the ASCAD network and the RD network, we see that the ASCAD network outperforms the RD network. For TA, we see that added noise does not help but both scenarios improve with the increase in the number of traces. For the behavior per fold, we observe that the best behavior is for several folds without noise. Still, there are folds with added noise that exhibit better performance than a number of folds without noise. Note, the behavior of TA is superior when compared with the RD network but much worse when compared with the ASCAD network.

5.1.3 AES_RD Dataset

The results for the AES_RD dataset are given in Figure 5. This dataset contains a countermeasure in the form of random delays and the behavior of the considered classifiers is very interesting. First, we see that the RD network exhibits superior performance where for all folds GE decreases with the increase in the number of traces and the results with



(a) RD network per fold.

(b) RD network averaged.

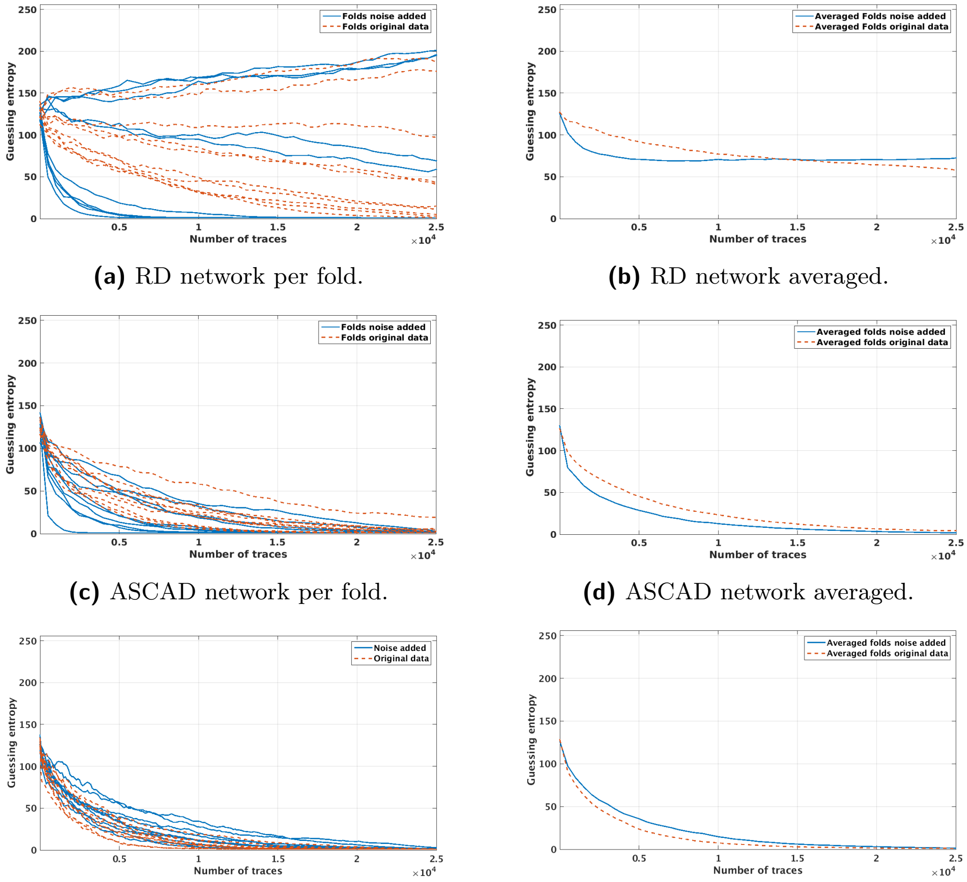
(c) ASCAD network per fold.

(d) ASCAD network averaged.

(e) Pooled template attack per fold.

(f) Pooled template attack averaged.

Figure 3: Guessing entropy results for the DPAcontest v4 dataset.



(a) RD network per fold.

(b) RD network averaged.

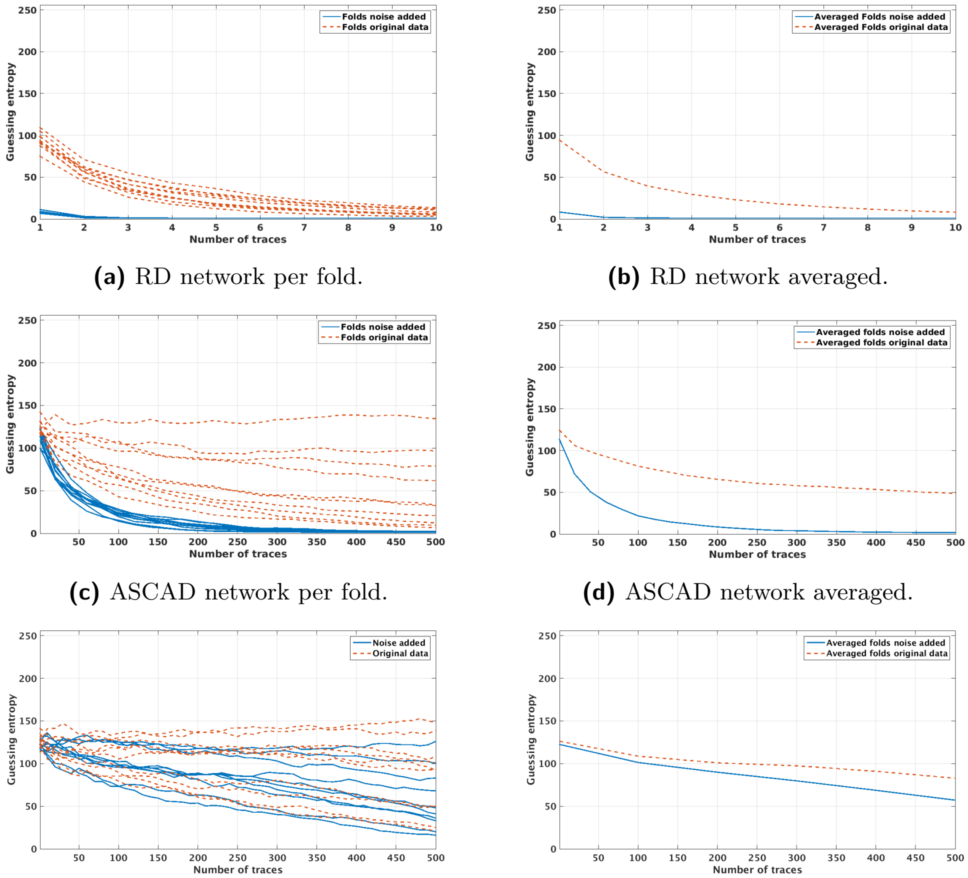
(c) ASCAD network per fold.

(d) ASCAD network averaged.

(e) Pooled template attack per fold.

(f) Pooled template attack averaged.

Figure 4: Guessing entropy results for the AES_HD dataset.



(a) RD network per fold.

(b) RD network averaged.

(c) ASCAD network per fold.

(d) ASCAD network averaged.

(e) Pooled template attack per fold.

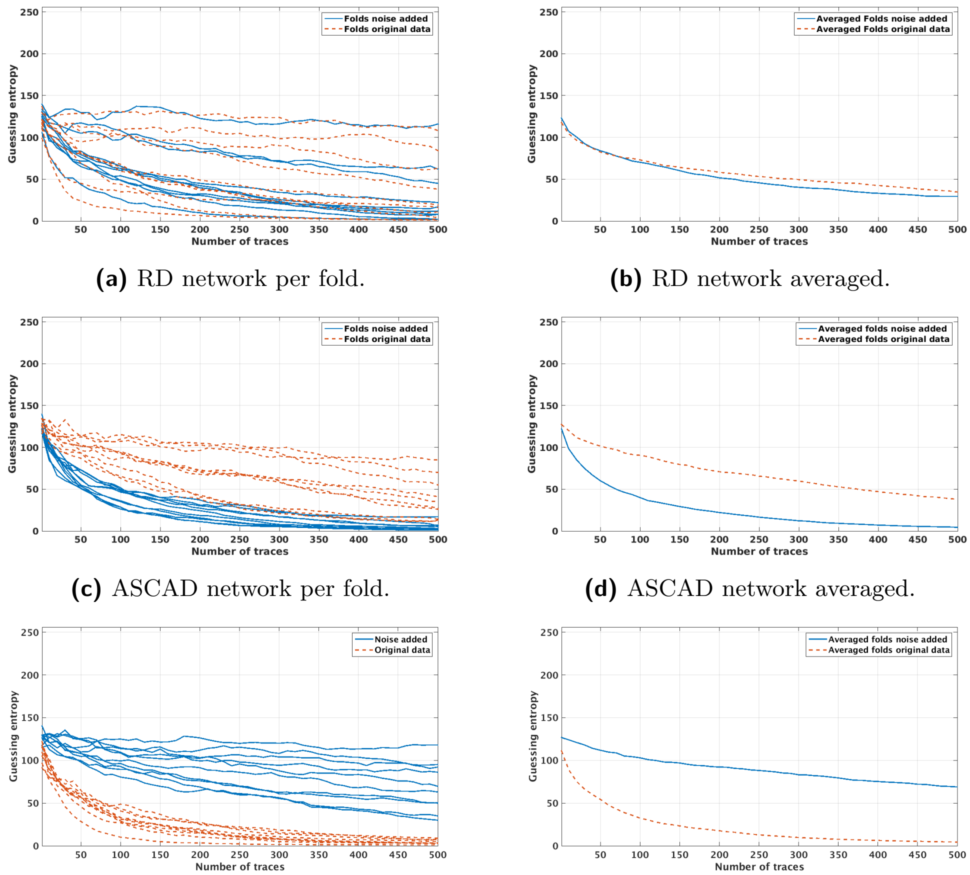
(f) Pooled template attack averaged.

Figure 5: Guessing entropy results for the AES_RD dataset.

artificial noise are much better than those without noise. When averaged, we see that we require only up to 3 traces to break the random delay countermeasure implementation. For the ASCAD network, the noise is beneficial since it reduces GE with the increase in the number of traces for all folds, while for the scenario without noise, we see a number of folds that do not improve with the increase in the number of traces. When considering averaged results, to break this implementation we need around 300 traces, which is 100 times more than for the RD network. Finally, considering TA, we see that the results with noise are somewhat better than the case without noise. For behavior per fold, we see that the best performance is for two folds with noise while the worst performance is for two folds without noise. We note that the required number of traces for TA to reach the same performance as the RD network is more than 20 000.

5.1.4 ASCAD Dataset

Finally, we give results for the ASCAD dataset in Figure 6. Note that the results given here slightly differ from those given in [PSB+18]. This is natural since we use fewer traces in the training phase. When considering the RD network, we see that all folds improve with the increase in the number of traces but for some folds that improvement is very slow. It is difficult to say whether adding noise helps here since the results are very similar and only after a few hundreds of traces we see that added noise behaves better than no noise.



(a) RD network per fold.

(b) RD network averaged.

(c) ASCAD network per fold.

(d) ASCAD network averaged.

(e) Template attack per fold.

(f) Template attack averaged.

Figure 6: Guessing entropy results for the ASCAD dataset.

Next, we see that the ASCAD network shows slightly worse performance than the RD network if we consider scenarios without added noise. This is somewhat surprising since the ASCAD network was developed especially for the ASCAD dataset and intuitively one would expect that a network which is thoroughly tuned for a certain dataset would perform better than a network developed to perform well over a number of datasets. However, we see a substantial increase in the performance of the ASCAD network once the noise is added. Finally, since pooled TA cannot break this implementation, we depict the results for template attack. For a small number of measurements, we see that TA is the most successful method while if considering more than 300 traces, TA and ASCAD perform similarly. TA performs significantly better without added noise for all folds as well as for the averaged performance.

5.2 Noise Addition

In this section, we present results for the RD neural network and all datasets where we investigate different levels of noise. We consider both the averaged behavior and the behavior per folds. Besides considering the noise level $\alpha = 0.5$ as in the previous section, now we also experiment with $\alpha = 0.25$, $\alpha = 0.75$, and $\alpha = 1$. Figure 7 depicts the obtained results for the RD network and all datasets. For completeness, we also include the results when $\alpha = 0$ for averaged scenarios. For DPAcontest v4, we see that all considered noise levels result in the decrease of GE with the increase in the number of traces. The best result for a fold is obtained with $\alpha = 0.5$ but the best averaged result is for $\alpha = 1$. This result is somewhat expected as DPAcontest v4 is easy to classify when given enough training samples, and adding extra noise simply makes the model even more distinguishable among different classes. The noise level $\alpha = 0.75$ shows a slight deviation from the other experiments since the decrease in GE is slower when compared with the other noise levels. Finally, when considering no added noise scenario, we can see that it works the worst in the case of a very limited number of available traces.

The second scenario, AES_HD shows that $\alpha = 1$ is too much noise when considering the full training data and GE is actually increasing with the number of measurements. Since AES_HD is a very noisy dataset, here the smaller level of noise $\alpha = 0.25$ behaves the best when considering the full training data where the model is approximately well defined. The results for noise levels $\alpha = 0.5$ and $\alpha = 0.75$ are similar and they even overlap at around 20 000 measurements. Here, we see the best performance for the smallest noise level and then for each higher noise level, we see a degradation in the results.

For AES_RD, $\alpha = 1$ exhibits the worst mean behavior while the other noise levels behave very similarly (notice that $\alpha = 0.25$ and $\alpha = 0.75$ behave almost the same). Still, when considering averaged results, we see that $\alpha = 0.5$ performs the best with a slight advantage over $\alpha = 0.25$ and $\alpha = 0.75$. Similar to the DPAcontest v4 dataset, the scenario with no added noise exhibits the worst performance when the number of available traces is very limited.

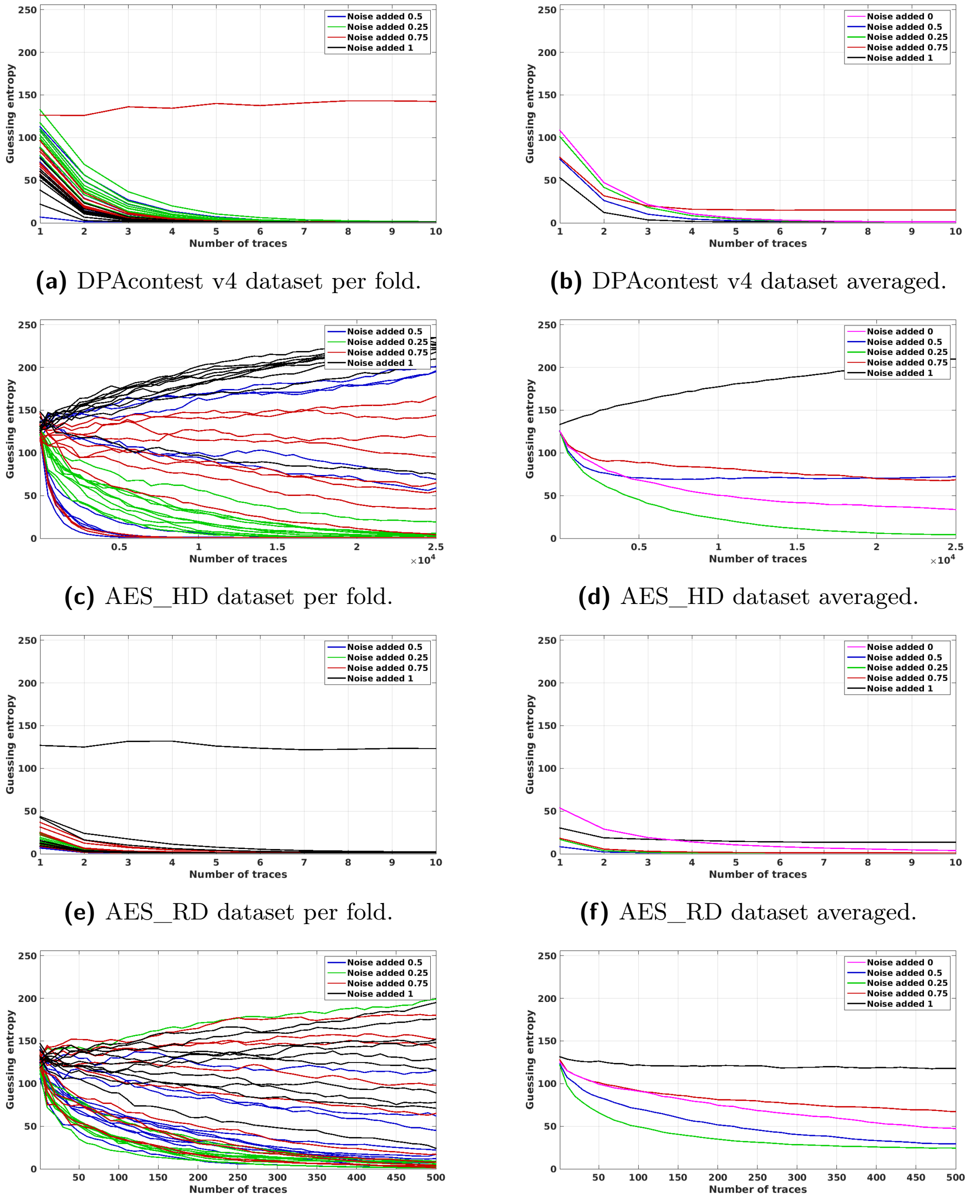
For the ASCAD dataset, we observe that $\alpha = 1$ does not work well as it results in most of the folds with an increased GE for an increased number of traces. The noise level $\alpha = 0.25$ gives the best results here with a small advantage over $\alpha = 0.5$ and $\alpha = 0.75$. Similar as for the AES_HD dataset, we see here that the higher the noise, the worse are the results. The scenario without added noise is positioned in the middle – better than $\alpha = 0.75$ and $\alpha = 1$ but worse than $\alpha = 0.25$ and $\alpha = 0.5$.

As it can be observed, various noise levels can be successfully applied in order to improve the generalizations of a classifier. Additionally, we see that the classifier is sensitive to the changes in the noise level only to a certain extent, which indicates that for most of the applications, it is not needed to tune the noise level to high precision. The noise level equal to $\alpha = 0.5$ shows a stable behavior over all datasets and can be considered as a good choice in our experiments. Naturally, taking $\alpha = 0.25$ would be also a viable choice, especially considering the AES_HD dataset. We will make a further conclusion in the next subsection when additionally epoch size and the training sample size is varied.

5.3 On the Levels of Noise, Profiling Set Sizes, and Number of Epochs

In order to investigate the effect of important hyper-parameters regarding model selection, we conducted a series of experiments with different setups of three different hyper-parameters: 1) the number of training traces, 2) the number of epochs, and 3) the effect of noise addition for each case. We ran 10 independent runs for each combination of noise levels and the number of training traces, using stratified random split as discussed in Section 4.6. As for the epoch, we measured the validation and testing accuracy for every epoch and for each run. The results are illustrated in Figures 8 to 11, considering RD Network for each dataset.

Each plot contains 6 subplots with respect to the number of traces used in the training, which spans in the range of $\{1\,024, 2\,048, 4\,096, 8\,196, 16\,384, full\}$. Since we applied the stratified random split approach, the selected number of cases explicitly contain the equal number of traces $\{4, 8, 16, 32, 64\}$ per class. For the case of full, 80% of the entire training set is used. We also indicated the random guess based line with the dotted red horizontal line for each subplot. Each curve in the subplots indicates the evolution of accuracy during the training process per various noise levels. The solid line indicates the mean



(a) DPAcontest v4 dataset per fold.

(b) DPAcontest v4 dataset averaged.

(c) AES_HD dataset per fold.

(d) AES_HD dataset averaged.

(e) AES_RD dataset per fold.

(f) AES_RD dataset averaged.

(g) ASCAD dataset per fold.

(h) ASCAD dataset averaged.

Figure 7: Guessing entropy results for the the RD network with different levels of noise, all datasets.

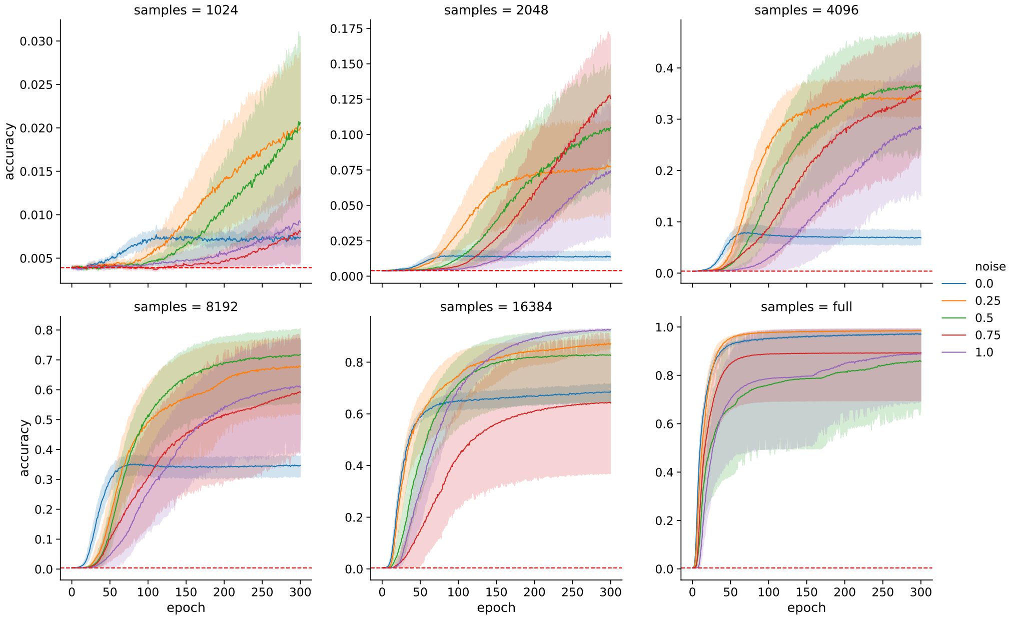


Figure 8: RD Network on DPAv4 dataset

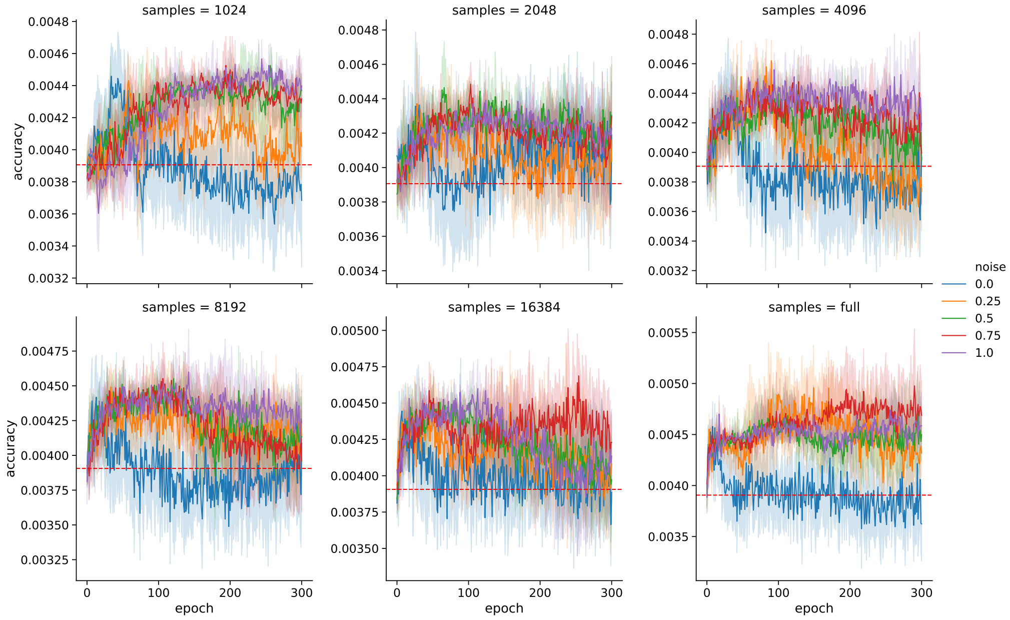


Figure 9: RD Network on AES_HD dataset

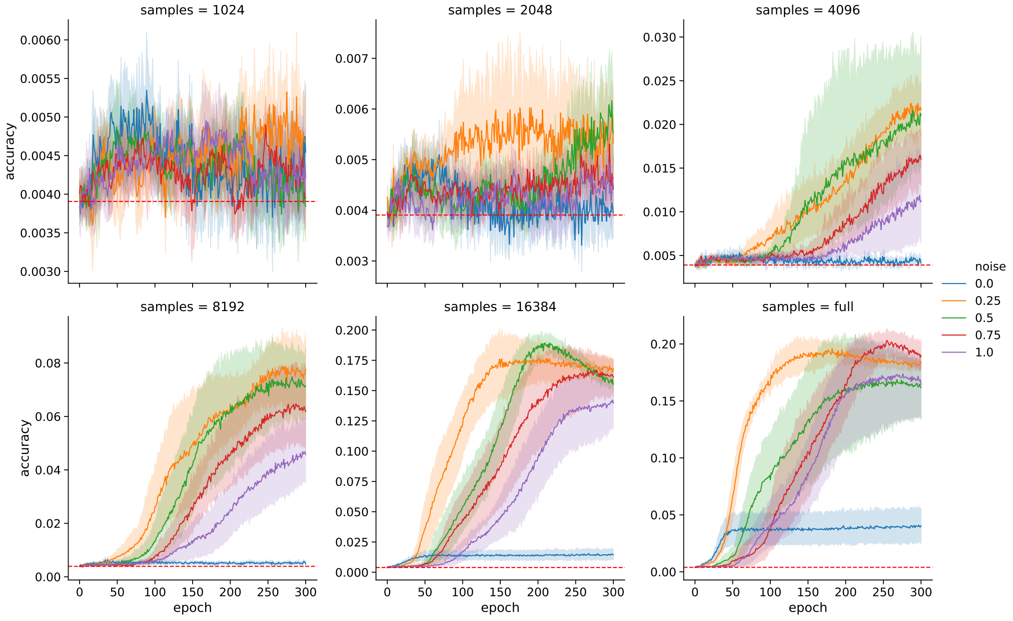


Figure 10: RD Network on AES_RD dataset

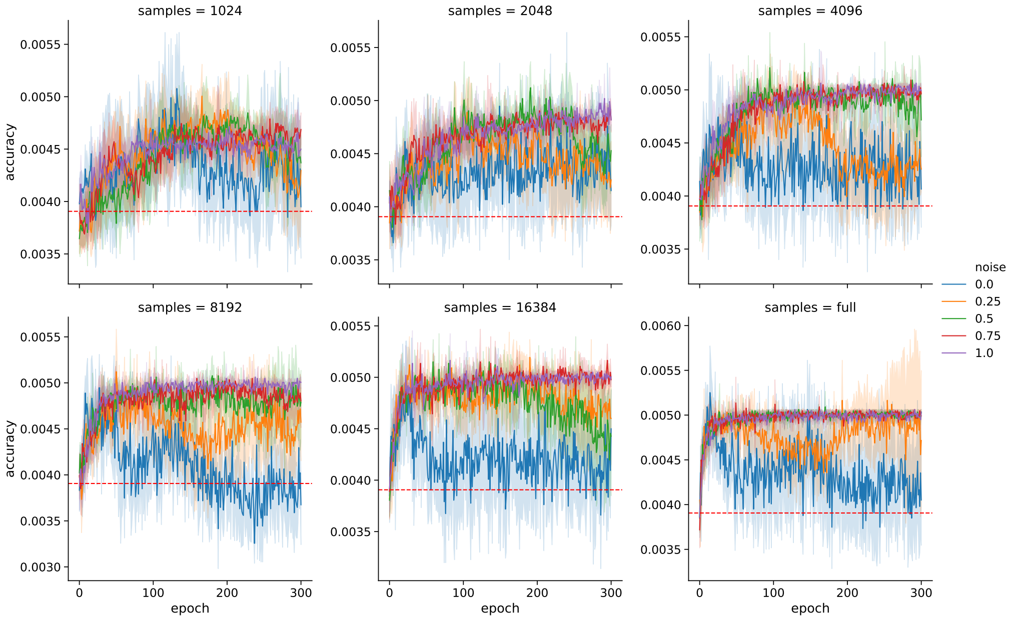


Figure 11: RD Network on ASCAD dataset

accuracy over 10 random splits, and the transparent range around the mean curve is the 95% confidence interval of the averaged curve, which is obtained by 1 000 times of the bootstrap process. For each bootstrap, 10 accuracy values are randomly re-sampled out of the observation of the actual result from 10 folds with replacement. The 97.5% percentile and 2.5% percentile of the mean of each randomly sampled accuracy is picked as the confidence interval.

In general, across all datasets, the results show that the noise addition helps the training process to become more reliable, especially when both the number of training traces ($N > 4\,096$) and the noise levels ($\alpha \geq 0.5$) are higher. The accuracy of the model without noise addition starts to degrade relatively early and the variance is increased, while the models regularized by the noise addition are improved for the longer training period, with less variance. This effect is especially visible on cases where the dataset is relatively easy to predict, such as DPAcontest v4 or AES_RD. For example, in Figure 10 we see that for sample size $N = 16\,384$ the accuracy without noise addition does not further increase the value of 0.01 when using more than 50 epochs, however, when adding additional noise with $\alpha = 0.25$ we observe a large increase in accuracy (from 0.025 to 0.175) between 50 and 150 epochs.

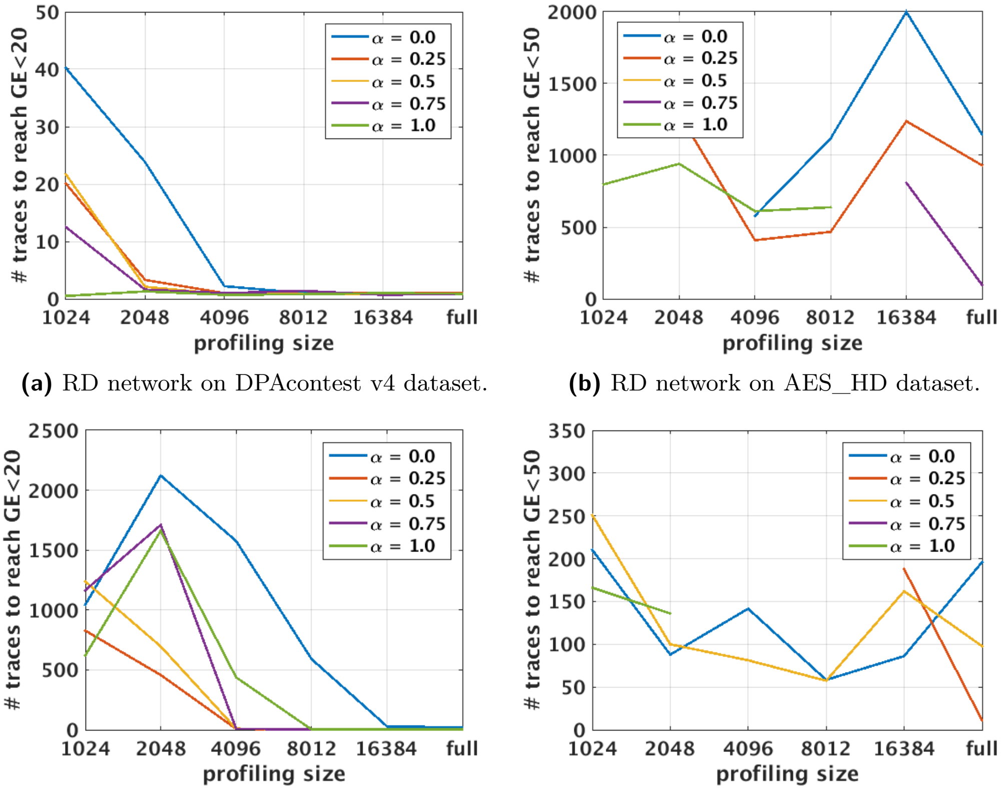
The results also show that the accuracy is improved with similar setups (levels of noise, number of training traces). This is again observable more clearly in the easier datasets but also shown for most of the cases except the ASCAD dataset. Here, the RD network shows a short peak in the earlier part of the training even without noise, which is comparable to the best performance of later improvement from the cases with noise.

Next, we observe that the number of training traces generally suggests a positive effect on accuracy. It is surprising that datasets (especially the easier ones) show good performance even with a small number of traces such as $1\,024, 2\,048,$ or $4\,096$, where the number of traces per class is only 4, 8, and 16. For DPAcontest v4 and AES_RD, the best accuracy achieved with this setup reaches to $\{0.02, 0.125, 0.35\}$ and $\{0.005, 0.006, 0.22\}$, respectively, which are substantially better than the random baseline. In the case where the ASCAD model is used on the ASCAD dataset, while the performance is also both getting more reliable and improved, it shows a boundary approximately at 0.5%, where the model hardly shows better accuracy even at the best scenario we tried.

Finally, the results suggest that a higher regularization with a larger noise can be helpful for obtaining the best model more reliably. As described in Section 3.3, one can obtain a model with a minimal generalization error if the monitoring is ideal. However, the reliability of such monitoring also depends on the stability of the training. If the variance between the monitoring metric is high, there is less chance to obtain the best model. This implies that the properly regularized training is not only advantageous for the distribution of the best possible accuracy values but also helpful for reliably drawing the best model out of such range.

When considering GE, we explore the influence of the levels of noise and profiling dataset size. Since we run neural networks until the performance starts to deteriorate, we have only one number of epochs for each experiment, which represents the best solution for accuracy.

For DPAcontest v4, in Figure 12a we can observe that added noise improves the performance when the profiling dataset size is small. Interestingly, for sample size $N = 1\,024$, the best performance is achieved with $\alpha = 1$, where we only need 1 trace to reach a GE below 20. Without adding noise (i.e., $\alpha = 0.0$), we require 40 traces. For larger sample sizes $N \geq 2\,048$ the effect of adding noise does not depend significantly on the magnitude of the noise, and for $N \geq 8\,012$, we cannot depict any difference between applying noise addition and using original data. This effect may be surprising at first sight, but when considering that for $N = 1\,024$ we only have 4 samples per class, it becomes intuitive that adding a higher amount of noise helps to generalize. The more training samples are



(a) RD network on DPAcontest v4 dataset.

(b) RD network on AES_HD dataset.

(c) RD network on AES_RD dataset.

(d) RD network on ASCAD dataset.

Figure 12: Guessing entropy results for different levels of noise and number of profiling traces.

available to learn, the less the effect of noise addition.

Interestingly, similar effects are visible on the AES_HD dataset. In Figure 12b we can see that for training sizes above 1 024, $\alpha = 0.25$ is able to reach $GE < 50$. This is inline with our previous results (see Figure 7c). Again, for lower sample sizes, $\alpha = 1$ achieves the best results, whereas for $N \geq 4\,096$, $\alpha = 0.25$ is the most efficient one. For $N > 16\,384$ we observe that $\alpha = 0.75$ reaches the best performance, whereas without noise addition the results are worse than for all noise levels that are able to reach the GE threshold.

Figure 12c gives results for the RD Network on the AES_RD dataset. Basically, throughout all sizes of training samples, adding noise is more efficient than using the original dataset. In particular, we see that $\alpha = 0.25$ performs the best. For $N = 4\,096$ all noise levels below 1 become nearly equivalent, where we need only three traces to reach a GE of 20, whereas without adding noise we require 1600. For $N = 8\,012$ all noise level perform similarly by requiring two traces, whereas without adding noise we still require 600 traces. For $N = 16\,384$ and the full dataset, the difference becomes smaller. We would like to emphasize that these results may not be intuitive for the SCA community, but they show the requirement of generalization for deep learning architectures for SCA.

For the ASCAD dataset, we observe again that for $N = 1\,024$ adding noise with $\alpha = 1$ performs the best. For $N = 4\,096$, $\alpha = 0.5$ and for the full dataset, $\alpha = 0.25$ performs the most efficiently, however for $N = 2\,048, 16\,384$ we see that the original dataset has an advantage. This effect may be due to the general reduced performance of the RD network on the ASCAD dataset as seen in Subsection 5.1.4.

6 Discussion

In this section, we start by presenting several conclusions we consider to be relevant for SCA with machine learning techniques. Next, we discuss the design principles of CNNs, with a special emphasis on a difference between architecture and instance of an architecture. Finally, we give several possible future research directions.

6.1 General Findings

1. **It seems impossible to develop a single neural network that will show superior performance over all SCA scenarios.** When discussing the performance of CNN classifiers, we observe that different neural networks perform well when considered over a number of scenarios. Still, the "No Free Lunch" theorem states that there cannot be a single best supervised classifier for all possible problems [Wol96]. Since the scenarios we need to consider in SCA are quite diverse with the respect to the number of measurements and features, levels of noise, and applied countermeasures, we consider as the only viable option to develop a suite of classifiers that are tuned to perform excellently for a specific problem. One example is the RD network and its behavior for the Random Delay dataset. Finally, we note that although we used different design principle for the RD network, the end result is similar to the neural network designed in [PSB+18]. Interestingly, despite the similarities in the design, those two neural networks show very different behavior for different datasets.

2. **It is possible to enhance the robustness of the model against the noise by adding Gaussian mean-free noise.** We show how the addition of noise can be actually beneficial in the process of classification since it enhances the robustness of the model in the presence of noise. Although it might sound counter-intuitive to add more noise in order to fight against noise, it is well-known that adding artificial noise to the input data is equivalent to adding regularization term to the objective function. Our results show that the added noise benefited both CNNs and even in certain cases (pooled) template attack. We emphasize the case of the ASCAD network that was tuned for the ASCAD dataset and yet, additional noise improved the results significantly. Additionally, our results show that our findings hold for both small training datasets as well as the larger ones. Even more, our results indicate that for smaller datasets the amount of noise added should be higher than for larger training datasets.

3. **When conducting experiments with cross-validation, i.e., using a number of folds, it is possible that the performance over folds will be extremely varying.** Our experiments show that a neural network with good performance for a certain dataset can actually exhibit both good and poor performance when considering separate folds. We hypothesize that the difficulty of a given classification task is relatively high, which eventually forces the accuracy to stay very close to the random guess. This means that the variability occurred by the other stochastic components in the experimental framework, such as the random split in the cross-validation can have a substantial magnitude of the variance. Such variability tends to make the model tuning process difficult. Our empirical result suggests that in general, the variability tends to decrease by applying a certain amount of the noise to the input signal. We hypothesize that such results imply the regularization can be used as one of the countermeasures to reduce such variability.

   We also applied cross-validation when using TA/TA pooled, where different folds as well show varying behaviors. We therefore believe that in future works on the evaluation of side-channel attacks the concept of cross-validation should be applied to machine-learning and deep learning algorithms as well as classical side-channel attacks.

4. **Attacking a dataset with a countermeasure (AES_RD) can be easier than attacking unprotected dataset (DPAcontest v4).** In the SCA community, the DPAv4 dataset (assuming the mask is known) is considered as one of the easiest targets known. Adding countermeasures such as the random delay is expected to improve the resistance against SCA. Interestingly, with our new network, we are able to reveal the secret key in the AES_RD dataset with fewer traces than for the DPAcontest v4 dataset. This could imply that the addition of a countermeasure may even help to reveal sensitive information for CNNs.

6.2 CNN - Architectures, Design Principles, and Instances

One of the fundamental differences between the traditional Machine Learning (ML) algorithms and Deep Neural Network (DNN) is the degree of freedom on the parametrization. While many classical machine learning algorithms have relatively small freedom on structuring the parametric model, DNN, on the other hand, can be defined in much more diverse parametrization due to the non-linearity. The non-linearity applied after affine transformation allows the cascading learning layers to have increasingly expressive power as the network getting "deeper". It ultimately allows one to "design" a parametric structure to optimize the expressive power to effectively encode the complex, non-linear input signal.

Consecutively, due to the exponentially increasing number of parameters, it is a common practice to start the architecture from the existing design principles that usually have been developed from the machine learning domain. As we introduced already, *VGG* is one of the most successful architectures adopted for a number of other application domains including speech, video, and music. Even though one starts their model configuration from such a "template", there is still a substantial amount of freedom to reach the optimal architecture according to the given task and dataset. The common practice is the heuristic search on the hyper-parameters of interest such as the previous works already introduced in the side-channel analysis domain [PSK+18, PSB+18]. On the other hand, one can also find a more specific "instance" that is proven as prominent in other domains especially if there is a considerable commonality on the characteristics of the signal between the two domains.

Such instantiation is necessary especially for CNNs, since the details regarding the configurations often depend on the shape of the input signal. Consequentially, it is not trivial to find a single CNN configuration that works optimally on multiple datasets, unless the CNN is designed specifically to have the time-invariance by applying causal convolution [ODZ+16].

Hence, every instantiation from a certain design principle can be seen as a "new" architecture unless they have used the exact same architecture that was developed previously, since an architecture may be a merely another perspective of "parametrization". Likewise, it does not necessarily mean that another instantiation is always "novel" if the design does not have genuine novelty. On the other hand, such novelty is again possibly irrelevant to the practical performance. Our empirical results show that the two models used here are essentially instantiated from a very similar design principle, which means they might not be novel architectures while their performance varies significantly across the datasets.

6.3 Future Work

When considering noise addition, one could envision it as a form of adversarial learning where we add noise to make the classifier more robust, i.e., resilient against perturbations in the input. This adversarial learning could be considered relatively weak since the noise is a random Gaussian and not specifically crafted to make the classification process as difficult as possible. Naturally, if the attacker on the machine learning system (which would be in this case an SCA countermeasure) would use a random Gaussian noise, then our noise addition could be considered as adversarial learning. But we can extend this line of thought further to what we believe to be even more interesting perspectives. It is well-known that TA is the most powerful attack in the information theoretic way but in practice, we see that is often not the case. Consequently, for many difficult (and hence) realistic scenarios we see that machine learning techniques are actually the most powerful attacks. Then, a countermeasure specifically designed against machine learning attacks could be a very powerful option although already random delay or masking could be considered as forms of adversarial examples. As a future work, we plan to investigate how to use more advanced types of adversarial examples as countermeasures in SCA.

As mentioned in Section 5, we observe an interesting behavior when considering folds. More precisely, for certain scenarios, some folds exhibit extremely good behavior while other folds even deteriorate with the increase in the number of measurements. When averaged, the behavior improves with the increase in the number of traces but much less than for the best fold. The question is whether it would be possible to use separate folds information for a successful attack. Since in realistic scenarios we do not know the correct labels for the test traces, the only way to test the performance of the model is to try to obtain the secret key. The same setting can be used for each fold (thus, in a way, considering a smaller profiling dataset). There are two variations how to conduct such experiment: 1) after each fold is constructed, try to break the implementation by using the model for that fold, or 2) obtain all folds and use the best one to try to break the implementation. The first technique could improve on computational complexity since less calculation is needed. The second scenario could improve on attack performance since one could decide not to use the fold that exhibits poor performance.

Finally, as Figure 9 shows, an extremely unoptimized model might result in a performance level near random guess. This leads the model's predictive power in a random way so that guessing entropy is not improved regardless of the number of traces that are used during the attack phase. For the future work, more in-depth investigation of the condition where such underfitting happens seems required.

7 Conclusions

In this paper, we concentrate on one type of deep learning that showed potential in the context of profiled SCA – Convolutional Neural Networks. We adapt a CNN design principle successfully applied in a different domain but to the same type of signal, i.e., 1-dimensional signal. With this CNN, we are able to reach high performance over 4 considered datasets where we emphasize the behavior on the dataset with the random delay countermeasure. There, to break it, we require only 3 traces, which is, to the best of our knowledge, a result surpassing anything that can be found in the literature. At the same time, we experiment with another CNN that has a similar design but exhibits significantly different behavior for the investigated dataset. We believe that in the SCA context, we can discuss general design principles that will work well over a number of datasets but still the specific instantiations need to be tuned for specific scenarios. Consequently, there is a need for a suite of high performing CNN instances (and possibly other types of deep learning and design principles) that will cover relevant scenarios in SCA.

Besides considering different CNN architectures/instances, we also explore how to make such designs more robust, i.e., how to enable them to better generalize to previously unseen data. There, we see that adding the Gaussian noise can help significantly in the classification process. In this context, we especially mention two scenarios, the Random Delay dataset with the RD neural network where after the addition of noise we require only 3 traces to break the cryptographic implementation. The second case is the ASCAD neural network and the ASCAD dataset. There, despite the fact that the ASCAD network was designed for the ASCAD dataset, we still see a significant boost in the performance obtained if we add noise. We conduct additional experiments with the RD network and differing levels of noise, number of epochs, and training dataset sizes to show the validity of our approach for a wide variety of settings.

To conclude, we propose to consider noise addition as a standard technique in the SCA evaluation for deep learning techniques. Besides that, we believe there is a need to build a suite of CNN-based side-channel analysis. We consider the RD neural network and the ASCAD neural network to be good choices to be included in such a suite.

References

[BCO04] Éric Brier, Christophe Clavier, and Francis Olivier. Correlation Power Analysis with a Leakage Model. In *CHES*, volume 3156 of *LNCS*, pages 16–29. Springer, August 11–13 2004. Cambridge, MA, USA.

[Bis95] Chris M Bishop. Training with noise is equivalent to tikhonov regularization. *Neural computation*, 7(1):108–116, 1995.

[Bre01] Leo Breiman. Random Forests. *Machine Learning*, 45(1):5–32, 2001.

[CDP17] Eleonora Cagli, Cécile Dumas, and Emmanuel Prouff. Convolutional Neural Networks with Data Augmentation Against Jitter-Based Countermeasures - Profiling Attacks Without Pre-processing. In *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, pages 45–68, 2017.

[CK09] Jean-Sébastien Coron and Ilya Kizhvatov. An Efficient Method for Random Delay Generation in Embedded Software. In *Cryptographic Hardware and Embedded Systems - CHES 2009, 11th International Workshop, Lausanne, Switzerland, September 6-9, 2009, Proceedings*, pages 156–170, 2009.

[CK13] Omar Choudary and Markus G. Kuhn. Efficient template attacks. In Aurélien Francillon and Pankaj Rohatgi, editors, *Smart Card Research and Advanced Applications - 12th International Conference, CARDIS 2013, Berlin, Germany, November 27-29, 2013. Revised Selected Papers*, volume 8419 of *LNCS*, pages 253–270. Springer, 2013.

[CRR02] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template Attacks. In *CHES*, volume 2523 of *LNCS*, pages 13–28. Springer, August 2002. San Francisco Bay (Redwood City), USA.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[GHO15] Richard Gilmore, Neil Hanley, and Maire O'Neill. Neural network based attack on a masked implementation of AES. In *2015 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 106–111, May 2015.

[HGDM+11] Gabriel Hospodar, Benedikt Gierlichs, Elke De Mulder, Ingrid Verbauwhede, and Joos Vandewalle. Machine learning in side-channel analysis: a first study. *Journal of Cryptographic Engineering*, 1:293–302, 2011. 10.1007/s13389-011-0023-x.

[HPGM17] Annelie Heuser, Stjepan Picek, Sylvain Guilley, and Nele Mentens. Lightweight ciphers and their side-channel resilience. *IEEE Transactions on Computers*, PP(99):1–1, 2017.

[HRG14] Annelie Heuser, Olivier Rioul, and Sylvain Guilley. Good is Not Good Enough — Deriving Optimal Distinguishers from Communication Theory. In Lejla Batina and Matthew Robshaw, editors, *CHES*, volume 8731 of *Lecture Notes in Computer Science*. Springer, 2014.

[HZ12] Annelie Heuser and Michael Zohner. Intelligent Machine Homicide - Breaking Cryptographic Devices Using Support Vector Machines. In Werner Schindler and Sorin A. Huss, editors, *COSADE*, volume 7275 of *LNCS*, pages 249–264. Springer, 2012.

[IS15] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pages 448–456, 2015.

[KB14] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[KLN18] Taejun Kim, Jongpil Lee, and Juhan Nam. Sample-level CNN architectures for music auto-tagging using raw waveforms. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2018, Calgary, AB, Canada, April 15-20, 2018*, pages 366–370, 2018.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[LB$^+$95] Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10), 1995.

[LBM14] Liran Lerman, Gianluca Bontempi, and Olivier Markowitch. Power analysis attack: An approach based on machine learning. *Int. J. Appl. Cryptol.*, 3(2):97–115, June 2014.

[LBM15] Liran Lerman, Gianluca Bontempi, and Olivier Markowitch. A machine learning approach against a masked AES - Reaching the limit of side-channel attacks with a learning model. *J. Cryptographic Engineering*, 5(2):123–139, 2015.

[LPB$^+$15] Liran Lerman, Romain Poussier, Gianluca Bontempi, Olivier Markowitch, and François-Xavier Standaert. Template Attacks vs. Machine Learning Revisited (and the Curse of Dimensionality in Side-Channel Analysis). In *COSADE 2015, Berlin, Germany, 2015. Revised Selected Papers*, pages 20–33, 2015.

[MGH14] Amir Moradi, Sylvain Guilley, and Annelie Heuser. Detecting Hidden Leakages. In Ioana Boureanu, Philippe Owesarski, and Serge Vaudenay, editors, *ACNS*, volume 8479. Springer, June 10-13 2014. 12th International Conference on Applied Cryptography and Network Security, Lausanne, Switzerland.

[MPP16] Houssem Maghrebi, Thibault Portigliatti, and Emmanuel Prouff. Breaking cryptographic implementations using deep learning techniques. In *Security, Privacy, and Applied Cryptography Engineering - 6th International Conference, SPACE 2016, Hyderabad, India, December 14-18, 2016, Proceedings*, pages 3–26, 2016.

[ODZ+16] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

[PGC+17] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[PHG17] Stjepan Picek, Annelie Heuser, and Sylvain Guilley. Template attack versus Bayes classifier. *Journal of Cryptographic Engineering*, 7(4):343–351, Nov 2017.

[PHJ+17] Stjepan Picek, Annelie Heuser, Alan Jovic, Simone A. Ludwig, Sylvain Guilley, Domagoj Jakobovic, and Nele Mentens. Side-channel analysis and machine learning: A practical perspective. In *2017 International Joint Conference on Neural Networks, IJCNN 2017, Anchorage, AK, USA, May 14-19, 2017*, pages 4095–4102, 2017.

[PHJ+18] 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 Transactions on Cryptographic Hardware and Embedded Systems*, 2019(1):209–237, Nov. 2018.

[PHJL17] Stjepan Picek, Annelie Heuser, Alan Jovic, and Axel Legay. Climbing down the hierarchy: Hierarchical classification for machine learning side-channel attacks. In Marc Joye and Abderrahmane Nitaj, editors, *Progress in Cryptology - AFRICACRYPT 2017: 9th International Conference on Cryptology in Africa, Dakar, Senegal, May 24-26, 2017, Proceedings*, pages 61–78, Cham, 2017. Springer International Publishing.

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

[PSK+18] Stjepan Picek, Ioannis Petros Samiotis, Jaehun Kim, Annelie Heuser, Shivam Bhasin, and Axel Legay. On the performance of convolutional neural networks for side-channel analysis. In Anupam Chattopadhyay, Chester Rebeiro, and Yuval Yarom, editors, *Security, Privacy, and Applied Cryptography Engineering*, pages 157–176, Cham, 2018. Springer International Publishing.

[PYW+18] Sihang Pu, Yu Yu, Weijia Wang, Zheng Guo, Junrong Liu, Dawu Gu, Lingyun Wang, and Jie Gan. Trace augmentation: What can be done even before preprocessing in a profiled sca? In Thomas Eisenbarth and Yannick Teglia, editors, *Smart Card Research and Advanced Applications*, pages 232–247, Cham, 2018. Springer International Publishing.

[RGBV11] Salah Rifai, Xavier Glorot, Yoshua Bengio, and Pascal Vincent. Adding noise to the input of a model trained with a regularized objective. *CoRR*, abs/1104.3250, 2011.

[SHK+14] Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[Sim07] Richard Simon. Resampling strategies for model assessment and selection. In *Fundamentals of data mining in genomics and proteomics*, pages 173–186. Springer, 2007.

[SLP05] Werner Schindler, Kerstin Lemke, and Christof Paar. A Stochastic Model for Differential Side Channel Cryptanalysis. In LNCS, editor, *CHES*, volume 3659 of *LNCS*, pages 30–46. Springer, Sept 2005. Edinburgh, Scotland, UK.

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

[SZ14] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

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

[Wol96] David H. Wolpert. The Lack of a Priori Distinctions Between Learning Algorithms. *Neural Comput.*, 8(7):1341–1390, October 1996.

A The Detailed Configuration of the Models

Table 2: Detailed configuration of each network used for datasets.

	Convolution			Pooling		Fully-Connected		Training	
Datasets	Size	Padding	Channels	Type	Size	Units	Dropout	Batch Size	Learn Rate
Baseline [PSB+18]	11	`same`	$(64, 128, 256, 512, 512)$	`average`	2	$(4\,096, 4\,096)$	N/A	100	0.0001
DPAv4	3	`valid`	$(8, 16, 32, 64, 128, 128, 128, 256, 256, 256)$	`max`	2	(256)	0.5	256	0.0001
Random Delay			$(8, 16, 32, 64, 128, 128, 128, 256, 256, 256)$						
ASCAD			$(8, 16, 32, 64, 64, 128, 256, 256)$						
AES_HD			$(8, 16, 32, 64, 128, 128, 256, 256)$			(512)			