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, Mekelweg 2, Delft, The Netherlands
[2] Univ Rennes, Inria, CNRS, IRISA, France
[3] Physical Analysis and Cryptographic Engineering, Temasek Laboratories at Nanyang Technological University, Singapore

Abstract. Profiled side-channel attacks 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 that is able to reach high performance for a number of considered datasets. Additionally, for a dataset protected with the random delay countermeasure, our neural network is able to break the implementation by using only 2 traces in the attack phase. We compare our neural network with the one designed for a particular dataset with masking countermeasure and we show how both are good designs but also how 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 improve the number of measurement needed to reveal the secret key by orders of magnitude in certain scenarios for both neural networks. To strengthen our experimental results, we experiment with a number of datasets which differ in the levels of noise (and type of countermeasure) where we show the viability of our approaches.

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

1 Introduction

Profiled side-channel attacks (SCA) and especially those based on machine learning techniques received a significant amount of attention in the SCA community lately. Such attention seems to be well-deserved since the results show a number of situations where machine learning techniques perform (extremely) well and

even surpass for instance, the template attack [1–3]. More recently, deep learning techniques emerged as a powerful alternative to more standard machine learning techniques when conducting side-channel attacks [3, 4]. There, Convolutional Neural Networks (CNNs) found their place as the main candidate to explore. Unfortunately, due to a limited number of published works, it is not clear when deep learning is actually needed in practice.

CNN has shown great performance across different domains, like image classification, where it can work with thousands of features and classes and millions of examples [5]. 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, lose effectiveness 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 [4] (i.e., masking countermeasure) as well as additional noise [3] (i.e., random delays). Still, deep learning architectures suitable for SCA, and particularly against practical countermeasures, are not well explored and compared.

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 a limited attention in the community so far.

In this paper, we propose several mechanisms to achieve high performance 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 the performance, we show how the addition of artificial noise can improve the behavior of CNN significantly as it prevents overfitting. To the best of our knowledge, this has not been done before in SCA and it represents a very 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 [6, 7]. After the template attack, the stochastic attack emerged using linear regression in the profiling phase [8]. 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 [9].

Alongside such techniques, the SCA community realized that a similar approach to profiling is used in other domains in the form of supervised machine learning. Consequently, some researchers started experimenting with different machine learning techniques and evaluating their effectiveness in the SCA context. As a consequence, today there is a plethora 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., [1,2,7,10–14]. When considering deep learning, the common examples in SCA are multilayer perceptron and Convolutional Neural Networks [3, 4, 15–18].

1.2 Our Contributions

This paper investigates what are the limits of CNNs' performance when considering side-channel attacks. To that end, we explore 4 datasets that we believe encompass the main types of measurements as detailed in Section 4. Our main contributions are:

1. a new CNN instance. We introduce a new CNN neural network instance and we experimentally show that it is able to reach high performance over considered datasets.
2. investigation how additional noise can be beneficial for the performance of CNNs in SCA. More precisely, we show that, although maybe counter-intuitive, the addition of noise can help neural networks to avoid overfitting and consequently reach better performance.
3. we show how random splitting of training data results in a drastically different behavior for neural networks. Although the standard approach is to consider the averaged behavior of all data folds, we discuss how to explore the behavior of each fold separately and use separate folds in practical attacks.
4. we discuss the difference between Convolutional Neural Networks architectures and instances. We consider this to be of high importance in order to facilitate future research and comparisons.

2 Background

2.1 Notation

Let calligraphic letters ($\mathcal{X}$) denote sets, capital letters (X) denote random variables taking values in these sets, and the corresponding lowercase letters (x) denote their realizations. Let k^* be the fixed secret cryptographic key (byte), k any possible key hypothesis, and the random variable T the plaintext or ciphertext of the cryptographic algorithm, which is uniformly chosen. We denote the measured leakage as X and consider multivariate leakage $\boldsymbol{X} = X_1, \ldots, X_D$, with D being the number of time samples or points-of-interest (i.e., features as called in the ML 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 N profiling traces $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_N$ in order to characterize the leakage. Once this phase is done, the attacker measures additional traces $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_Q$ 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. Hamming weight (or distance) of sensitive intermediate value is 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 reduced number of classes which result in reduced training complexity. On the other hand, HW/HD classes are inherently imbalanced which can seriously hamper the performance of machine learning/deep learning [19]. The alternative is to profile directly on the intermediate value which requires a bigger profiling set but does not suffer from imbalance. In the case of AES, i.e., our target in this paper, 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 imbalance problem.

To assess the performance of the classifiers, we use Guessing entropy (GE) [20]. GE states the average number of key candidates an adversary needs to test to reveal the secret key after conducting a side-channel attack. In particular, let us assume, given Q 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 a 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 $\mathbf{g}$.

2.3 Convolutional Neural Networks – CNNs

CNNs are a specific type of neural networks which were first designed for 2-dimensional convolutions as it was inspired by the biological processes of animals' visual cortex [21]. 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 [22]. Their usage in side-channel analysis has been encouraged by [3, 4]. Considering the signal given in side-channel analysis 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 [23], 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 task where the input signal has 2 spatial dimensions, while the signal given in the side-channel analysis has only 1 spatial dimension. Still, the core design principle is identical, which can be formalized as a function h:

$$h = g_{\theta,\texttt{softmax}} \circ \prod_{n=1}^{N} g_{\theta^n,\texttt{ReLU}} \circ \prod_{m=1}^{M} \left(\varphi_{\texttt{Max}} \circ \prod_{l=1}^{L_m} f_{\phi^l,\texttt{ReLU}}\right), \tag{1}$$

where N, M, L_m represent the number of fully-connected layers, convolution blocks, and convolution layers in m^{th} convolution block, respectively. The "convolution block" refers to the set of chained operations within the parenthesis in Eg. (1) that consists of L_m convolution layers $f_{\phi,\sigma}$ and one or less pooling layer φ. φ_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 [24]. $f_{\theta,\sigma}$ and $g_{\theta,\sigma}$ are convolution and fully-connected layers, respectively, which are followed by the non-linear transformation given as follows:

$$f_{\phi,\sigma}(X) = \sigma(\phi * X). \tag{2}$$

$$g_{\theta,\sigma}(x) = \sigma(\theta^\intercal x), \tag{3}$$

where $X \in \mathbb{R}^{i \times l}$ is the input signal that has i channels and length l, and $\phi \in \mathbb{R}^{i \times o \times s}$ is the convolution filter that has an i input channels, o output channels and filter length s. $x \in \mathbb{R}^d$ is the input vector and $\theta \in \mathbb{R}^{d \times k}$ is the projection matrix that transforms d dimensional input to k 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 functions such as `ReLU` or `Softmax`.

2.4 Template Attack

The template attack relies on the Bayes theorem and considers the features as being dependent. In the state-of-the-art, it is mostly assumed that the noise follows a multivariate normal distribution. Accordingly, each $P(\boldsymbol{X} = \boldsymbol{x}|Y = y) \sim \mathcal{N}(\mu_y, \Sigma_y)$ where μ_y is the mean vector and Σ_y the covariance matrix for each class y. The authors of [9] 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 Σ_y 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 and briefly explain the CNN instance introduced in [18]. 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 [25]. 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 a few fully-connected layers. It is reported that the network based on the strategy indicates better performance in the music related classification tasks which the strategy is originally developed on [25]. Surprisingly, by grid search and cross validation, the authors in [18] reach very similar architecture setup, 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 the network architecture. The example of detailed architecture can be found in Figure 1. The key principles of applied design strategy are 1) the minimal filter size (3), 2) convolution block continuing until the spatial dimension reduced to 1 3) the number of channels starts from a small number, and keep increasing similarly to *VGG*. In order to allow easier notation and comparison, we denote this neural network as RD network.

In addition to the above standard architecture, we applied to additional components to improve the model: batch normalization [26] and dropout [27]. 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 [26]. Dropout is a simple but powerful technique for the

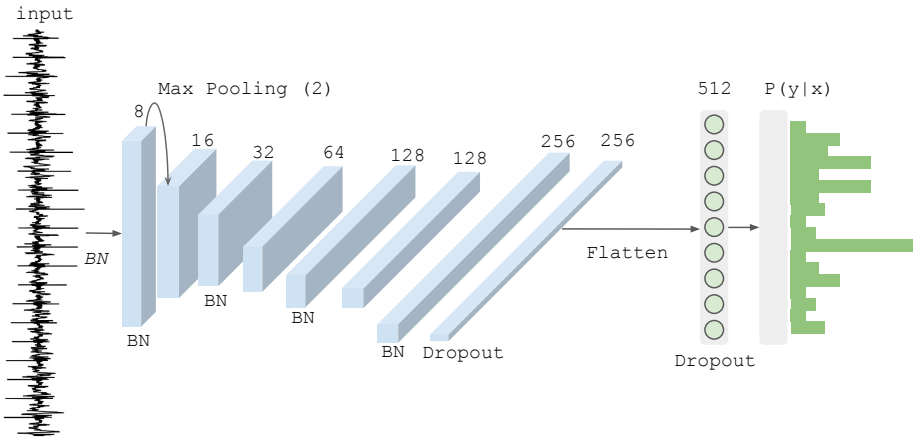


Fig. 1: An example schema of the CNN architecture. This particular configuration is used for the AES HD dataset. The model encodes a trace into latent representation such that the final layer can output the posterior probability for each label. The blue tensors indicate the activation after the convolution layer $f_{\phi^m,\texttt{ReLU}}$. 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 $g_{\theta^n,\texttt{ReLU}}$. Finally, the rightmost component in the pipeline is the output of the final fully-connected layer $g_{\theta,\texttt{softmax}}$ that outputs the probability for each 256 label.

regularization of the model, which simply introduces randomly generated mask on the activation such that the model is not overly dependent on the what is already learned from the training set [27]. 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.

3.2 ASCAD Neural Network

Along with the model we introduce in Section 3.1, we also test another CNN model introduced in [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 above model is the size and depth of the network. Compared to a model developed on the same dataset following the above principle, for instance, ASCAD model has approximately 130 times larger number of parameters, mostly due to the fully-connected layers with large dimensionality in its hidden units. Still, in terms of depth, the other model (RD network) is deeper, which means that more abstractions can be conducted by the convolution blocks.

3.3 Adding Artificial Noise to the Input

To enhance the robustness of the model against the noise, we applied a simple regularization 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 [28, 29]. Generally speaking, it can be seen as "blurring" high-frequency patterns that are more frequently appearing in the data, which will eventually make the model concentrating more on the 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 batch normalization layer, we added the noise tensor Ψ after the first batch normalization as follows:

$$X^* = BN_0(X) + \alpha\Psi, \tag{4}$$

where BN_0 indicates the batch normalization applied on the input tensor X. After the tuning process, we set the mean and standard deviation of the Gaussian distribution where the noise tensor Ψ is drawn as 0 and 1, respectively.

For the ASCAD model, we use the average and standard deviation of the entire trace for each dataset to draw the random noise. Also, since ASCAD model does not have the batch normalization layer, the noise application is executed as the following way:

$$X^* = X + \alpha\Psi_d^{ASCAD}, \tag{5}$$

where Ψ_d^{ASCAD} indicates the random noise drawn from the normal distribution whose average and the standard deviation is derived from the dataset d. For all cases, we apply the mixing coefficient $\alpha = 0.5$.

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 [30]. 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 [31] 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 is considered with no countermeasures and has a small level of noise. This dataset represents an ideal attack target (and one not seen often in reality) and is consequently 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 between 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

DPAcontest v4 provides measurements of a masked AES software implementation [32]. As the masking is found to leak first-order information [33], 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

$$Y(k^*) = \texttt{Sbox}[P_i \oplus k^*] \oplus \underbrace{M}_{\text{known mask}} , \tag{6}$$

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)

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 [32], i.e.,

$$Y(k^*) = \underbrace{\texttt{Sbox}^{-1}[C_i \oplus k^*]}_{\text{previous register value}} \oplus \underbrace{C_j}_{\text{ciphertext byte}}, \tag{7}$$

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)

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 [34]. 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.,

$$Y(k^*) = \texttt{Sbox}[P_i \oplus k^*], \tag{8}$$

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 [3]. 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

Finally, we test our architecture on the recently available ASCAD database [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.,

$$Y(k^*) = HW(\texttt{Sbox}[P_i \oplus k^*]). \tag{9}$$

Interested readers can find more information about this dataset at [18]. This dataset is available at https://github.com/ANSSI-FR/ASCAD. Note that, the model given in Eq. (9) does not leak information directly as it is first-order protected, 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.

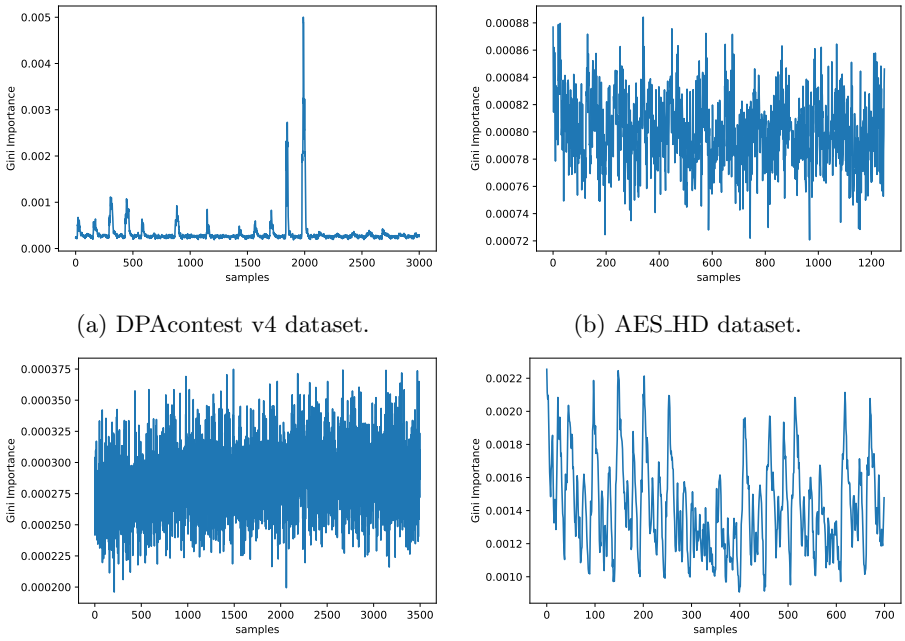
In Figure 2, we depict the most important features for all considered dataset. To investigate the feature relevance, we use the Random Forest classifier. Random Forest is a well-known ensemble decision tree learner [35]. Decision trees choose their splitting attributes from a random subset of k attributes at each internal node. The best split is taken among these randomly chosen attributes and the trees are built without pruning, RF is a parametric algorithm with respect to the number of trees in the forest. For this experiment, we use the number of trees equal to $I = 500$.

Notice, 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.

Table 1 sums 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 [18] and perform a PCA and select the 15 most relevant components.

4.5 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



(a) DPAcontest v4 dataset.

(b) AES_HD dataset.

(c) Random Delay dataset.

(d) ASCAD dataset.

Fig. 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.

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* [36], 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

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

a larger number of validation traces per class with 10 folds, which is suggested in [36], 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

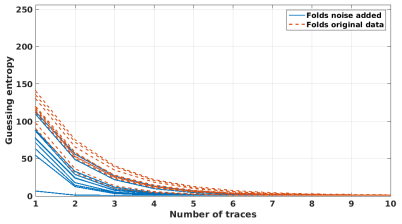
In this section, we start by providing results for our RD neural network, ASCAD neural network, and template attack (pooled) for two noise levels: $\alpha = 0$ and $\alpha = 0.5$. Afterward, we give additional experiments with different levels of noise in order to explore its limitations. As for the 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.

5.1 Results

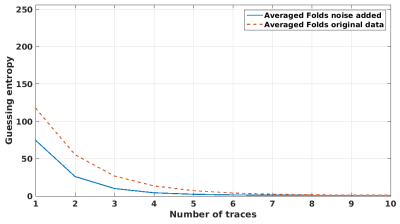
When presenting the results, we adopt the following setting: for both CNNs, we present the results for each fold as well as the averaged results. Note, the averaged results are what we usually use in SCA to assess the performance of a machine learning-based classifier. Still, we believe this not to be a requirement and we discuss in Section 7 how the results of each fold could be used in a practical attack. At the same time, we do not give results per each fold for TA since this is not usual for standard SCA techniques (although there is no technical reason prohibiting such a procedure).

DPAcontest v4 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 measurement only 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

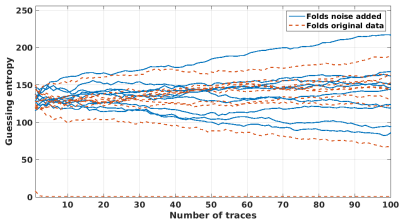
noise addition are almost half the results without artificial noise addition. When considering looking at each fold separately, we see that most of the folds with noise are better than the folds without noise. Interestingly, there is one fold that is several times better than the average behavior with 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. Finally, 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.



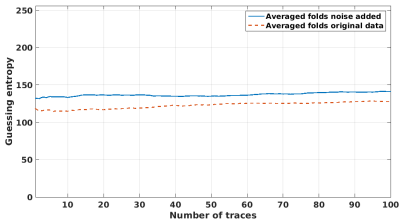
(a) RD network per fold.



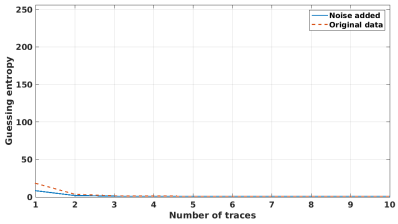
(b) RD network averaged.



(c) ASCAD network per fold.



(d) ASCAD network averaged.



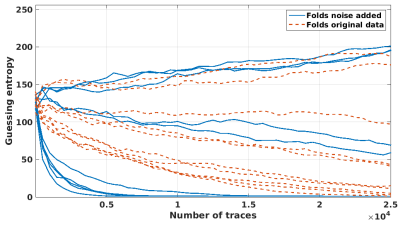
(e) Pooled template attack.

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

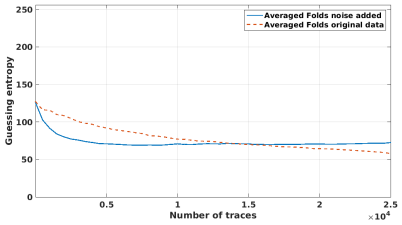
AES_HD In Figure 4, we give the results for AES_HD, which exhibits quite a different behavior from DPAcontest v4. Still, this is not unexpected due to a much higher level of environmental and algorithmic noise. For the RD network, we see for a number of folds very good behavior but unfortunately, there are some folds that actually deteriorate with the increase in the number of traces. 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 with the ASCAD network. When comparing the behavior of the ASCAD network and RD network, we see that the ASCAD network outperform RD. When considering TA, we see that added artificial does not help but GE decreases even in that case with the increase in the number of traces. The behavior of TA is superior when compared with the RD network but much worse when compared with the ASCAD network.

AES_RD 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 artificial noise are much better than those without noise. When averaged, we see that we require only 2 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 150 times more than for the RD network. Finally, considering TA, we see that the results with and without artificial behave similarly and the required number of traces to be around 20 00 to reach the same performance as RD and ASCAD networks.

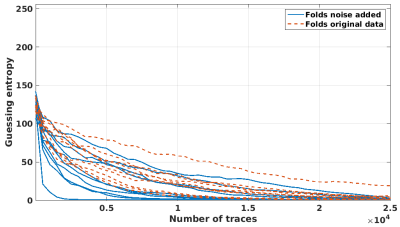
ASCAD Finally, we give results for the ASCAD dataset in Figure 6. Note that the results given here slightly differ from those given in [18]. This is natural since we use fewer traces in the training phase. When considering the RD network, we see that all folds improve with the increase in the number 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. Next, we see that the ASCAD network shows slightly worse performance than 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



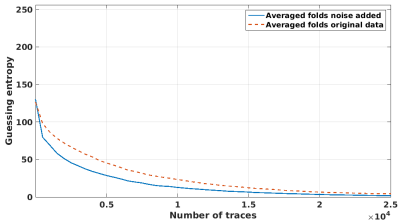
(a) RD network per fold.



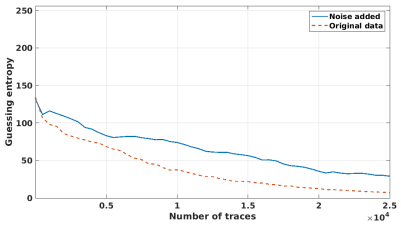
(b) RD network averaged.



(c) ASCAD network per fold.

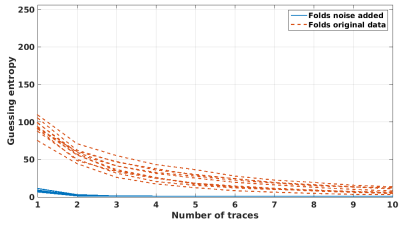


(d) ASCAD network averaged.

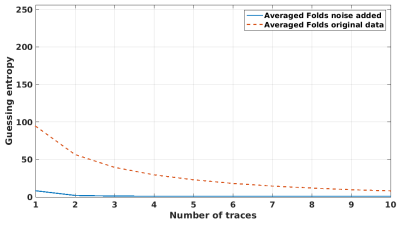


(e) Pooled template attack.

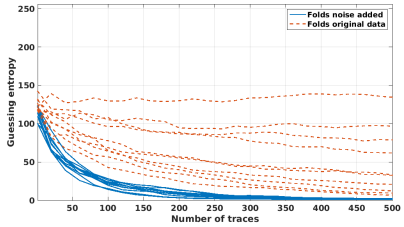
Fig. 4: Guessing entropy results for the AES_HD dataset.



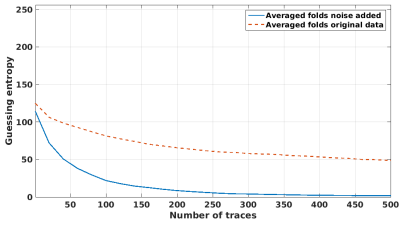
(a) RD network per fold.



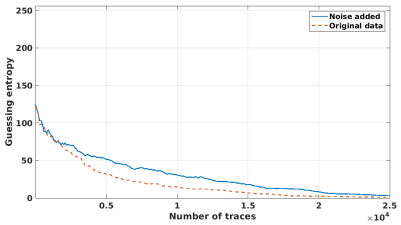
(b) RD network averaged.



(c) ASCAD network per fold.



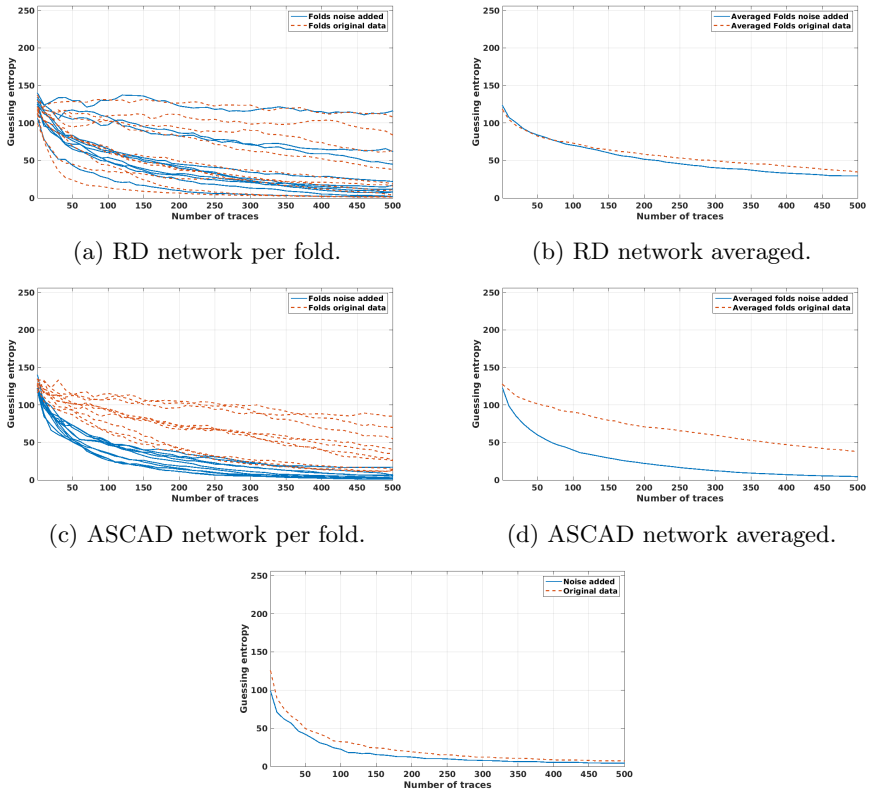
(d) ASCAD network averaged.



(e) Pooled template attack.

Fig. 5: Guessing entropy results for the AES_RD dataset.

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. What is particularly interesting is that we see that adding artificial noise benefits TA for a small number of measurements.



(a) RD network per fold.

(b) RD network averaged.

(c) ASCAD network per fold.

(d) ASCAD network averaged.

(e) Template attack.

Fig. 6: Guessing entropy results for the ASCAD dataset.

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. Besides considering the noise level

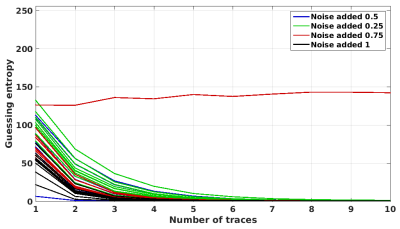
$\alpha = 0.5$ as in the previous section, now we also experiment with $\alpha = 0.25$, $\alpha = 0.75$, and $\alpha = 1$. Figure 7 depicts the obtained results for the RD network and all datasets. For completeness, we also include the results when $\alpha = 0$ for averaged scenarios. For DPAcontest 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 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 actually it works the worst in the case of a very limited number of available traces.

Already the second scenario, AES_HD shows that $\alpha = 1$ is a too significant level of noise and there GE actually increases 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. 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. The scenario with no noise added performs well for this dataset and the only better case is the smallest added noise equal to $\alpha = 0.25$.

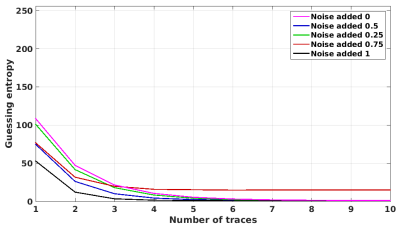
For AES_RD, $\alpha = 1$ exhibits the worst behavior while the other noise levels behave very similarly (notice that $\alpha = 0.25$ and $\alpha = 0.75$ behave almost the same). Still, when considering averaged results, we see that $\alpha = 0.5$ performs the best with slight advantage over $\alpha = 0.25$ and $\alpha = 0.75$. Similar to the DPAcontest v4, he 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 with 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 $\alpha = 0.5$.

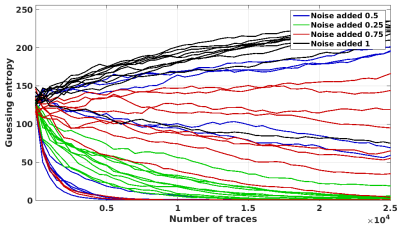
As it can be observed, various noise levels can be successfully applied in order to improve the generalizations of a classifier. Additionally, we see that the classifier is sensitive to the changes in the noise level only to a certain extent, which indicates that for most of the applications, it is not needed to tune the noise level to a 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.



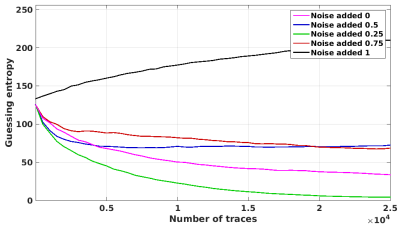
(a) DPAcontest v4 dataset per fold.



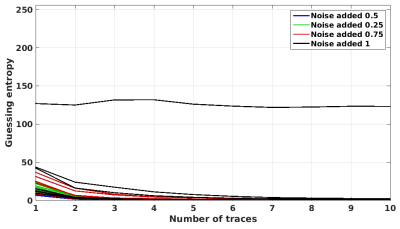
(b) DPAcontest v4 dataset averaged.



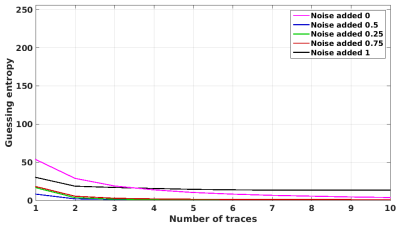
(c) AES_HD dataset per fold.



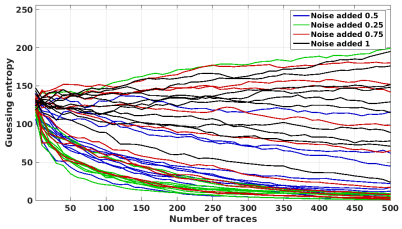
(d) AES_HD dataset averaged.



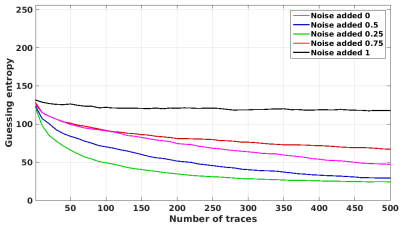
(e) AES_RD dataset per fold.



(f) AES_RD dataset averaged.



(g) ASCAD dataset per fold.



(h) ASCAD dataset averaged.

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

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 an architecture and instance of an architecture. Finally, we discuss 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 [37]. 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 excellent 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 [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 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 in 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.
3. **When conducting the 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 given classification task is relatively high, which eventually makes the accuracy 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 substantial magnitude of the variance. Such variability tends to make the model

tuning process difficult, especially the model we use tends not to robust to the randomness. Although, our empirical result suggests that in general the variability tends to be decreased by applying certain amount of the noise on the input signal. We hypothesize that such results imply that the regularization can be one of the countermeasures reducing such variability.

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 [17,18]. 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 [22].

Hence, every instantiation from a certain design principle can be seen as a "new" architecture unless they have used the exactly same architecture that is developed by other, since architecture may be a mere 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 for CNNs. More precisely, for certain scenarios, some folds exhibit extremely good behavior while other folds actually even deteriorate with the increase in the number of measurements. Then, when averaged, the behavior improves with the increase in the number of traces but much less than for the best fold. The question is how can we use folds information for a successful attack. Since in realistic scenarios we do not know what are the correct labels for the test set, the only way how to test it is to try to obtain the secret key. The same setting can be used for each fold. First, we run the experiments for a single fold. Next, we try to use only that information to break the implementation where we use up to a certain number of traces that we set as a threshold value θ. If the attack is successful, then the procedure is done. Otherwise, we repeat the same procedure for the second fold. If the second fold is again not providing sufficient information to break the implementation with up to θ traces, then we combine the first two folds. This procedure is repeated until we either break the implementation or explore all folds without success. This procedure would be useful in all scenarios where different folds exhibit large variations in behavior. In the worst-case scenario, this technique increases the number of GE evaluations by up to two times the number of folds when compared with the "standard" setting.

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 a same type of signal, i.e., 1 spatial dimension. With this CNN, we are able to reach

high performance over 4 considered datasets where we especially mention the dataset with the random delay countermeasure. There, to break it, we require only 2 traces, which is to the best of our knowledge, a result highly 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 a significantly different behavior for the investigated dataset. We believe that in the SCA context, we can discuss a 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, we believe 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 2 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.

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 attacks. We consider the RD neural network and the ASCAD neural network to be good choices to be included in such a suite.

A The Detailed Configuration of the Models

References

1. Lerman, L., Bontempi, G., Markowitch, O.: Power analysis attack: An approach based on machine learning. Int. J. Appl. Cryptol. **3**(2) (June 2014) 97–115
2. Picek, S., Heuser, A., Guilley, S.: Template attack versus Bayes classifier. Journal of Cryptographic Engineering **7**(4) (Nov 2017) 343–351
3. Cagli, E., Dumas, C., Prouff, E.: Convolutional Neural Networks with Data Augmentation Against Jitter-Based Countermeasures - Profiling Attacks Without Preprocessing. In: Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings. (2017) 45–68
4. Maghrebi, H., Portigliatti, T., Prouff, E.: 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. (2016) 3–26
5. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In Pereira, F., Burges, C.J.C., Bottou, L., Weinberger,

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 [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)			

K.Q., eds.: Advances in Neural Information Processing Systems 25. Curran Associates, Inc. (2012) 1097–1105
6. Heuser, A., Rioul, O., Guilley, S.: Good is Not Good Enough — Deriving Optimal Distinguishers from Communication Theory. In Batina, L., Robshaw, M., eds.: CHES. Volume 8731 of Lecture Notes in Computer Science., Springer (2014)
7. Lerman, L., Poussier, R., Bontempi, G., Markowitch, O., Standaert, F.: Template Attacks vs. Machine Learning Revisited (and the Curse of Dimensionality in Side-Channel Analysis). In: COSADE 2015, Berlin, Germany, 2015. Revised Selected Papers. (2015) 20–33
8. Schindler, W., Lemke, K., Paar, C.: A Stochastic Model for Differential Side Channel Cryptanalysis. In LNCS, ed.: CHES. Volume 3659 of LNCS., Springer (Sept 2005) 30–46 Edinburgh, Scotland, UK.
9. Choudary, O., Kuhn, M.G.: Efficient template attacks. In Francillon, A., Rohatgi, P., eds.: Smart Card Research and Advanced Applications - 12th International Conference, CARDIS 2013, Berlin, Germany, November 27-29, 2013. Revised Selected Papers. Volume 8419 of LNCS., Springer (2013) 253–270
10. Heuser, A., Zohner, M.: Intelligent Machine Homicide - Breaking Cryptographic Devices Using Support Vector Machines. In Schindler, W., Huss, S.A., eds.: COSADE. Volume 7275 of LNCS., Springer (2012) 249–264
11. Hospodar, G., Gierlichs, B., De Mulder, E., Verbauwhede, I., Vandewalle, J.: Machine learning in side-channel analysis: a first study. Journal of Cryptographic Engineering **1** (2011) 293–302 10.1007/s13389-011-0023-x.
12. Lerman, L., Bontempi, G., Markowitch, O.: A machine learning approach against a masked AES - Reaching the limit of side-channel attacks with a learning model. J. Cryptographic Engineering **5**(2) (2015) 123–139
13. Picek, S., Heuser, A., Jovic, A., Legay, A.: Climbing down the hierarchy: Hierarchical classification for machine learning side-channel attacks. In Joye, M., Nitaj, A., eds.: Progress in Cryptology - AFRICACRYPT 2017: 9th International Conference on Cryptology in Africa, Dakar, Senegal, May 24-26, 2017, Proceedings, Cham, Springer International Publishing (2017) 61–78
14. Picek, S., Heuser, A., Jovic, A., Ludwig, S.A., Guilley, S., Jakobovic, D., Mentens, N.: 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. (2017) 4095–4102
15. Gilmore, R., Hanley, N., O'Neill, M.: Neural network based attack on a masked implementation of AES. In: 2015 IEEE International Symposium on Hardware Oriented Security and Trust (HOST). (May 2015) 106–111
16. Heuser, A., Picek, S., Guilley, S., Mentens, N.: Lightweight ciphers and their side-channel resilience. IEEE Transactions on Computers **PP**(99) (2017) 1–1
17. Picek, S., Samiotis, I.P., Heuser, A., Kim, J., Bhasin, S., Legay, A.: On the performance of convolutional neural networks for side-channel analysis. Cryptology ePrint Archive, Report 2018/004 (2018) `https://eprint.iacr.org/2018/004`.
18. Prouff, E., Strullu, R., Benadjila, R., Cagli, E., Dumas, C.: Study of deep learning techniques for side-channel analysis and introduction to ASCAD database. IACR Cryptology ePrint Archive **2018** (2018) 53
19. Picek, S., Heuser, A., Jovic, A., Bhasin, S., Regazzoni, F.: The curse of class imbalance and conflicting metrics with machine learning for side-channel evaluations. Cryptology ePrint Archive, Report 2018/476 (2018) `https://eprint.iacr.org/2018/476`.

20. Standaert, F.X., Malkin, T., Yung, M.: A Unified Framework for the Analysis of Side-Channel Key Recovery Attacks. In: EUROCRYPT. Volume 5479 of LNCS., Springer (April 26-30 2009) 443–461 Cologne, Germany.
21. LeCun, Y., Bengio, Y., et al.: Convolutional networks for images, speech, and time series. The handbook of brain theory and neural networks **3361**(10) (1995)
22. Oord, A.v.d., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., Kavukcuoglu, K.: Wavenet: A generative model for raw audio. arXiv preprint arXiv:1609.03499 (2016)
23. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. CoRR **abs/1409.1556** (2014)
24. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press (2016) `http://www.deeplearningbook.org`.
25. Kim, T., Lee, J., Nam, J.: 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. (2018) 366–370
26. Ioffe, S., Szegedy, C.: 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. (2015) 448–456
27. Srivastava, N., Hinton, G.E., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: a simple way to prevent neural networks from overfitting. Journal of Machine Learning Research **15**(1) (2014) 1929–1958
28. Bishop, C.M.: Training with noise is equivalent to tikhonov regularization. Neural computation **7**(1) (1995) 108–116
29. Rifai, S., Glorot, X., Bengio, Y., Vincent, P.: Adding noise to the input of a model trained with a regularized objective. CoRR **abs/1104.3250** (2011)
30. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. CoRR **abs/1412.6980** (2014)
31. Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., Lerer, A.: Automatic differentiation in pytorch. (2017)
32. TELECOM ParisTech SEN research group: DPA Contest (4^{th} edition) (2013–2014) `http://www.DPAcontest.org/v4/`.
33. Moradi, A., Guilley, S., Heuser, A.: Detecting Hidden Leakages. In Boureanu, I., Owesarski, P., Vaudenay, S., eds.: ACNS. Volume 8479., Springer (June 10-13 2014) 12th International Conference on Applied Cryptography and Network Security, Lausanne, Switzerland.
34. Coron, J., Kizhvatov, I.: 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. (2009) 156–170
35. Breiman, L.: Random Forests. Machine Learning **45**(1) (2001) 5–32
36. Simon, R.: Resampling strategies for model assessment and selection. In: Fundamentals of data mining in genomics and proteomics. Springer (2007) 173–186
37. Wolpert, D.H.: The Lack of a Priori Distinctions Between Learning Algorithms. Neural Comput. **8**(7) (October 1996) 1341–1390