On the Performance of Convolutional Neural Networks for Side-Channel Analysis

Stjepan Picek[1], Ioannis Petros Samiotis[1], Jaehun Kim[1], Annelie Heuser[2], Shivam Bhasin[3(✉)], and Axel Legay[4]

[1] Delft University of Technology, Mekelweg 2, Delft, The Netherlands
[2] CNRS, IRISA, Rennes, France
[3] Physical Analysis and Cryptographic Engineering, Temasek Laboratories, Nanyang Technological University, Singapore, Singapore
sbhasin@ntu.edu.sg
[4] Inria, IRISA, Rennes, France

Abstract. In this work, we ask a question whether Convolutional Neural Networks are more suitable for side-channel attacks than some other machine learning techniques and if yes, in what situations. Our results point that Convolutional Neural Networks indeed outperform machine learning in several scenarios when considering accuracy. Still, often there is no compelling reason to use such a complex technique. In fact, if comparing techniques without extra steps like preprocessing, we see an obvious advantage for Convolutional Neural Networks when the level of noise is small, and the number of measurements and features is high. The other tested settings show that simpler machine learning techniques, for a significantly lower computational cost, perform similarly or sometimes even better. The experiments with guessing entropy indicate that methods like Random Forest or XGBoost could perform better than Convolutional Neural Networks for the datasets we investigated.

Keywords: Side-channel analysis · Machine learning · Deep learning Convolutional Neural Networks

1 Introduction

Side-channel analysis (SCA) is a process exploiting physical leakages in order to extract sensitive information from a cryptographic device. The ability to protect devices against SCA represents a paramount requirement for the industry. One especially attractive target for physical attacks is the Internet of Things (IoT) [1] since (1) the devices to be attacked are widespread and in the proximity of an attacker and (2) the available resources to implement countermeasures on devices are scarce. Consequently, we want a setting where the countermeasures are simple (i.e., cheap) and yet being able to protect from the most powerful

A. Chattopadhyay et al. (Eds.): SPACE 2018, LNCS 11348, pp. 157–176, 2018.
https://doi.org/10.1007/978-3-030-05072-6_10

attacks. At the same time, many products have transaction counters which set a limit for the number of side-channel measurements one is able to collect.

The profiled side-channel analysis defines the worst case security assessment by conducting the most powerful attacks. In this scenario, the attacker has access to a clone device, which can be profiled for any chosen or known key. Afterward, he is able to use the obtained knowledge to extract the secret key from a different device. Profiled attacks are conducted in two distinctive phases where the first phase is known as the profiling (or sometimes learning/training) phase, while the second phase is called the attack (test) phase. A well-known example of such an attack is template attack (TA) [2], a technique that is the best (optimal) from an information theoretic point of view if the attacker has an unbounded number of traces [3,4]. Soon after the template attack, the stochastic attack that uses linear regression in the profiling phase was developed [5]. In coming years, researchers recognized certain weaknesses in template attack and they tried to modify it in order to better account for different (usually, more difficult) attack scenarios. 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 [6].

Alongside such techniques, the SCA community recognized that the same general profiled approach is actually used in supervised machine learning. Machine learning (ML) is a term encompassing a number of methods that can be used for tasks like classification, clustering, feature selection, and regression [7]. Consequently, the SCA community started to experiment with different ML techniques and to evaluate whether they are useful in the SCA context, see e.g., [4,8–18]. Although considering different scenarios and often different machine learning techniques (with some algorithms used in a prevailing number of works like Support Vector Machines and Random Forest), all those works have in common that they establish numerous scenarios where ML techniques can outperform template attack and are the best choice for profiled SCA.

More recently, deep learning techniques started to capture the attention of the SCA community. In 2016, Maghrebi et al. conducted the first analysis of deep learning techniques for profiled SCA as well as a comparison against a number of ML techniques [19]. The results were very encouraging with deep learning surpassing other, simpler machine learning techniques and TA. Less than one year later, a paper focusing on Convolutional Neural Networks (CNNs) showed impressive results: this technique was better performing than TA but was also successful against device protected with different countermeasures [20]. This, coupled with a fact that the authors were able to propose several clever data augmentation techniques, boosted even further the confidence in deep learning for SCA.

In this work, we take a step back and investigate a number of profiled SCA scenarios. We compare one deep learning technique that got the most attention in SCA community up to now – CNNs against several, well-known machine learning techniques. Our goal is to examine the strengths of CNNs when compared with different machine learning techniques and to recognize what are the most suitable scenarios (considering complexity, explainability, ease of use, etc.) to use deep

learning. We emphasize that the aim of this paper is not to doubt CNNs as a good approach but to doubt it as the best approach for any profiled SCA setting. The main contributions of this work are:

1. We conduct a detailed comparison between several machine learning techniques in an effort to recognize situations where convolutional neural networks offer clear advantages. We especially note XGBoost algorithm, which is well-known as an extremely powerful technique but has never before been used in SCA. We show results for both accuracy and guessing entropy in an effort to better estimate the behavior of tested algorithms.
2. We design a convolutional neural network architecture that is able to reach high accuracies and compete with ML techniques as well as with the other deep learning architecture designed in [19].
3. We conduct an experiment showing that the topology of measurements does not seem to be the key property for CNNs' good performance.
4. We discuss scenarios where convolutional neural networks could be the preferred choice when compared with other, simpler machine learning techniques.

2 Background

2.1 Profiled Side-Channel Analysis

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 ML domain). To guess the secret key, the attacker first needs to choose a model $Y(T, k)$ 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 $Y(T, k)$.

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 estimate the leakage model. 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.

2.2 Machine Learning Techniques

We select several machine learning techniques to be tested against CNN approach. More precisely, we select one algorithm based on Bayes theorem (Naive Bayes), one tree-based method based on boosting (Extreme Gradient Boosting),

one tree-based method based on bagging (Random Forest), and finally, one neural network algorithm (Multilayer perceptron). We do not use Support Vector Machines (SVM) in our experiments despite their good performance as reported in a number of related works. This is because SVM is computationally expensive (especially when using radial kernel) and our experiments showed problems when dealing with imbalanced data (as occurs here since we consider the Hamming weight model) and large amounts of noise. For all ML techniques, we use scikit-learn library in Python 3.6 while for CNNs we use Keras with TensorFlow backend [21,22].

We follow that line of investigation since the "No Free Lunch Theorem" for supervised machine learning proves there exists no single model that works best for every problem [23]. To find the best model for a specific given problem, numerous algorithms and parameter combinations should be tested. Naturally, not even then one can be sure that the best model is obtained but at least some estimate about trade-offs between the speed, accuracy, and complexity of the obtained models is possible. Besides the "No Free Lunch Theorem" we briefly discuss two more relevant machine learning notions. The first one is connected with the curse of dimensionality [24] and the Hughes effect [25], which states that with a fixed number of training samples, the predictive power reduces as the dimensionality increases. This indicates that for scenarios with a large number of features, we need to use more training examples, which is a natural scenario for deep learning. Finally, the Universal Approximation theorem states that neural network is a universal functional approximator, more precisely, even a feed-forward neural network with a single hidden layer that consists of a finite number of neurons can approximate many continuous functions [26]. Consequently, by adding hidden layers and neurons, neural networks gain more approximation power.

Naive Bayes – NB. The Naive Bayes classifier is a method based on the Bayesian rule. It works under the simplifying assumption that the predictor attributes (measurements) are mutually independent among the features given the target class [27]. The existence of highly correlated attributes in a dataset can thus influence the learning process and reduce the number of successful predictions. NB assumes a normal distribution for predictor attributes and outputs posterior probabilities.

Multilayer Perceptron – MLP. The Multilayer perceptron is a feed-forward neural network that maps sets of inputs onto sets of appropriate outputs. MLP consists of multiple layers of nodes in a directed graph, where each layer is fully connected to the next one. To train the network, the backpropagation algorithm is used, which is a generalization of the least mean squares algorithm in the linear perceptron. An MLP consists of three or more layers (since input and output represent two layers) of nonlinearly-activating nodes [28]. Note, if there is more than one hidden layer, we can already talk about deep learning.

Extreme Gradient Boost – XGBoost. The XGBoost is a scalable implementation of gradient boosting decision tree algorithm [29]. Chen and Guestrin

designed this algorithm where they use a sparsity aware algorithm for handling sparse data and a theoretically justified weighted quantile sketch for approximate learning [30]. As the name suggests, its core part is gradient boosting (since it uses a gradient descent algorithm to minimize the loss when adding new models). Here, boosting is an ensemble technique where new models are added to correct the errors made by existing models. Models are added sequentially until no further improvements can be made. Today, XGBoost is due to his execution speed and model performance one of the top performing algorithms in the ML domain. Since this algorithm is based on decision trees, it has additional advantages as being robust in noisy scenarios.

Random Forest – RF. The Random Forest algorithm is a well-known ensemble decision tree learner [31]. 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. RF is a stochastic algorithm because of its two sources of randomness: bootstrap sampling and attribute selection at node splitting.

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 [32]. 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 [33]. Their usage in side-channel analysis has been encouraged by [19,20]. As we explain in Sect. 3.2, in order to find the most optimized model for the available datasets, we use random search for hyperparameter tuning. This enabled us to study how different architectures behaved on the datasets and compare the results to determine the best candidate model for our experimental setup. As this work is not attempting to propose a new optimal architecture for side-channel data classification, we used the most optimized network found through the Random Search for our benchmarks. The final architecture was chosen after creating hyperparameter constraints based on the literature and tests we conducted, followed by an optimization of their values through a random search. The hyperparameters that are modeled and optimized are number of convolutional/pooling/fully connected layers, number of activation maps, learning rate, dropout magnitude, convolutional activation functions, convolutional/pooling kernel size, and stride and number of neurons on fully connected layers.

During the training, we use early stopping to further avoid overfitting by monitoring the loss on the validation set [34]. Thus, every training session is interrupted before reaching high accuracy on the training dataset. To help the network increase its accuracy on the validation set, we use a learning rate scheduler to decrease the learning rate depending on the loss from the validation set. We initialize the weights to small random values and we use "adam" optimizer [35].

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. We implement the experiments with the Tensorflow [22] computing framework and Keras deep learning framework [21] to leverage the GPU computation.

2.4 Performance Analysis

To assess the performance of the classifiers (and consequently the attacker) we use accuracy: $ACC = \frac{TP+TN}{TP+FP+FN+TN}$. TP refers to true positive (correctly classified positive), TN to true negative (correctly classified negative), FP to false positive (falsely classified positive), and FN to false negative (falsely classified negative) instances. TP, TN, FP, and FN are well-defined for hypothesis testing and binary classification problems. When dealing with the multiclass classification, they are defined in one class–vs–all other classes manner and are calculated from the confusion matrix. The confusion matrix is a table layout where each row represents the instances in an actual class, while each column represents the instances of a predicted class.

Besides accuracy, we use also Success rate (SR) and Guessing entropy (GE) [36]. Given a certain amount of traces, SR is defined as the estimated average probability of success of an attack. In other words, what is the probability on average that the attack predicts the correct secret key given a certain amount of traces. Given a ranking of secret key candidates (i.e., from probabilities or scores) of an attack, the guessing entropy is the average position of the correct secret key in the ranking.

3 Experimental Setting

3.1 Datasets

In our experiments, we use three datasets: one representing an easy target to attack due to a low level of noise, one more difficult target due to a high level of noise, and finally, one with the random delay countermeasure.

DPAcontest v4 Dataset [37]**.** This dataset (denoted DPAv4) gives measurements of a masked AES software implementation but since the mask is known, one can easily transform it into an unprotected scenario. Since it is a software implementation, the most leaking operation is not the register writing but the processing of the S-box operation and we attack the first round:

$$Y(k^*) = \texttt{Sbox}[P_{b_1} \oplus k^*] \oplus \underbrace{M}_{\text{known mask}}, \tag{1}$$

where P_{b_1} is a plaintext byte and we choose $b_1 = 1$. We consider here a setting with 9 classes corresponding to the Hamming weight of the output of an S-box. The SNR for this dataset lies between 0.1188 and 5.8577. For our experiments,

we start with a preselected window of 3 000 features (around the S-box part of the algorithm execution) from the original trace. Note that we maintain the lexicographical ordering (topology) of features after the feature selection (by lexicographical ordering we mean keeping the features in the order they appear in measurements and not, for instance, sorting them in accordance to their relevance).

DPAcontest v2 Dataset [38]**.** DPAcontest v2 (denoted DPAv2) provides measurements of an AES hardware implementation. Previous works showed that the most suitable leakage model (when attacking the last round of an unprotected hardware implementation) is the register writing in the last round:

$$Y(k^*) = \underbrace{\texttt{Sbox}^{-1}[C_{b_1} \oplus k^*]}_{\text{previous register value}} \oplus \underbrace{C_{b_2}}_{\text{ciphertext byte}} . \tag{2}$$

Here, C_{b_1} and C_{b_2} are two ciphertext bytes and the relation between b_1 and b_2 is given through the inverse ShiftRows operation of AES. We select $b_1 = 12$ resulting in $b_2 = 8$ since it is one of the easiest bytes to attack [38]. In Eq. (2), $Y(k^*)$ consists of 256 values but we apply the Hamming weight (HW) on those values resulting in 9 classes. These measurements are relatively noisy and the resulting model-based signal-to-noise ratio $SNR = \frac{var(signal)}{var(noise)} = \frac{var(y(t,k^*))}{var(x-y(t,k^*))}$, lies between 0.0069 and 0.0096. There are several available datasets under the DPAcontest v2 name and we use the traces from the "template" set. This dataset has 3 253 features.

Random Delay Countermeasure Dataset. As our last use case, we use a 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 [39]. 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 (the SNR has a maximum value of 0.0556). We mounted our attacks in the Hamming weight power consumption model against the first AES key byte, targeting the first S-box operation. This dataset has 50 000 traces with 3 500 features each. This countermeasure has been shown to be prone to deep learning based side-channel [20]. The random delay is quite often used countermeasure in commercial products, while not modifying the leakage order (like masking). The dataset is publicly available at https://github.com/ikizhvatov/randomdelays-traces.

3.2 Data Preparation and Parameter Tuning

We denote the training set size as Tr, validation set size as V, and testing set size as Te. Here, $Tr + V + Te$ equals to the total set size S. We experiment with four dataset sizes S – [1 000, 10 000, 50 000, 100 000]. For the Random delay dataset, we use the first three sizes since it has only 50 000 measurements. The ratios for Tr, V, and Te equal 50% : 25% : 25%. All features are normalized

into $[0, 1]$ range. When using ML techniques, instead of validation, we use 5-fold cross-validation. In the 5-fold cross-validation, the original sample is first randomly partitioned into 5 equal sized subsets. Then, a single subsample is selected to validate the data while the remaining 4 subsets are used for training. The cross-validation process is repeated 5 times where each of the 5 subsamples is used once for validation. The obtained results are then averaged to produce an estimation. We select to conduct 5-fold cross-validation on the basis of the number of measurements belonging to the least populated class for the smallest dataset we use. Since the number of features is too large for ML techniques, we conduct feature selection where we select the 50 most important features while we keep the lexicographical ordering of selected features. We use 50 features for ML techniques since the datasets are large and the number of features is one of two factors (the second one is the number of measurements) comprising the time complexity for ML algorithms. Additionally, 50 features is a common choice in the literature [12,14]. To select those features, we use the correlation coefficient where we calculate it for the target class variables HW, which consists of categorical values that are interpreted as numerical values [40]:

$$Pearson(x, y) = \frac{\sum_{i=1}^{N}((x_i - \bar{x})(y_i - \bar{y}))}{\sqrt{\sum_{i=1}^{N}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{N}(y_i - \bar{y})^2}}. \tag{3}$$

Despite the fact it is not the best practice to use such selected features with the CNNs, we also include the experiment using CNNs with the selected features as an input for the purpose of the comparison and completeness. For CNNs, we do not conduct cross-validation since it is too computationally expensive but rather we additionally use the validation set, that serves as an indicator of early stopping to avoid overfitting.

In order to find the best hyperparameters, we tune the algorithms with respect to their most important parameters as described below:

1. The Naive Bayes has no parameters to tune.
2. For MLP, we tune the solver parameter that can be either *adam*, $lbfgs$, or *sgd*. Next, we tune activation function that can be either $ReLU$ or $Tanh$, and the number and structure of hidden layers in MLP. The number of hidden layers is tuned in the range $[2, 3, 4, 5, 6]$ and the number of neurons per layer in the range $[10, 20, 30, 40, 50]$.
3. For XGBoost, we tune the learning rate and the number of estimators. For learning rate, we experiment with values $[0.001, 0.01, 0.1, 1]$ and for the number of estimators with values of $[100, 200, 400]$.
4. For RF, we tune the number of trees in the range $[10, 50, 100, 200, 500]$, with no limit to the tree size.

When dealing with CNNs, in order to find the best fitting model, we optimized 13 hyperparameters: convolutional kernel size, pooling size, stride on convolutional layer, initial number of filters and neurons, learning rate, the number of convolutional/pooling/fully connected layers, type of activation function, optimization algorithm, and dropout on convolutional and fully connected layers.

The hyperparameter optimization was implemented through a random search, where the details on possible parameter ranges are given in Table 1). We tune our CNN architecture for the DPAcontest v4 dataset.

Table 1. Hyperparameters and their value ranges.

Hyperparameter	Value range	Constraints
Convolutional kernel	$k_{conv} \in [3, 20]$	-
Pooling kernel	$k_{pool} \in [3, 20]$	$k_{pool} \leq k_{conv}$
Stride	$s \in [1, 5]$	In pooling layers, $s = k_{pool} - 1$
# of convolutional layers	$layers_{conv} \in [2, 6]$	-
# of pooling layers	$layers_{pool} \in [1, 5]$	$layers_{pool} \leq layers_{conv}$
# of fully-connected layers	$layers_{fc} \in [0, 2]$	-
Initial # of activation maps	$a \in [8, 32]$	Follows geometric progression with ratio r = 2, for the # of $layers_{conv}$
Initial # of neurons	$n \in [128, 1024]$	Follows geometric progression with ratio $r = 2$, for the # of $layers_{fc}$
Convolutional layer dropout	$drop_{conv} \in [0.05, 0.10]$	-
Fully-connected layer dropout	$drop_{fc} \in [0.10, 0.20]$	-
Learning rate	$l \in [0.001, 0.012]$	A learning rate scheduler was applied
Activation function	ReLU, ELU, SELU, LeakyReLU, PReLU	The same for all layers except the last which uses Softmax
Optimization algorithm	Adam, Adamax, NAdam, Adadelta, Adagrad, SGD, RMSProp	-

We use the Softmax activation function in the classification layer combined with the Categorical Cross Entropy loss function. For regularization, we use dropout on convolutional and fully connected layers while on the classification layer we use an activity L2 regularization. These regularization techniques help to avoid overfitting on the training set, which in turn help lower the bias of the model. The number of activation maps increases per layer, following a geometric progression with an initial value $a = 16$ and a ratio $r = 2$ (16, 32, 64, 128). The number of activation maps is optimized for GPU training. The network is composed of 4 convolutional layers and 4 pooling layers in between, followed by the classification layer. All convolutional layers use kernel size of 6 and stride 2 creating a number of activation maps for each layer. For pooling we use Average Pooling on the first pooling layer and Max Pooling on the rest, using the kernel of size 4 and stride of 3. The convolutional layers use "Scaled Exponential Linear

Unit" (SELU) activation function, an activation function which induces self-normalizing properties [41]. We depict our architecture in Fig. 1 and give details about it in Table 2.

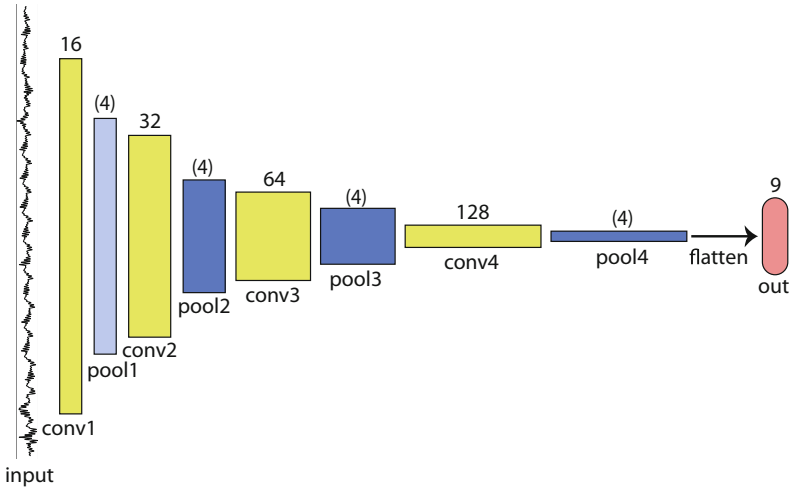


Fig. 1. The developed CNN architecture. The simplified figure illustrates the applied architecture. The yellow rectangular blocks indicate 1-dimensional convolution layer, and the blue blocks indicate pooling layers. The first light blue block indicates average pooling, which is different from the other max pooling blocks. After the flattening of every trailing spatial dimension into a single feature dimension, we apply a fully-connected layer for classification. (Color figure online)

Table 2. Developed CNN architecture.

Layer	Output shape	Weight shape	Sub-sampling	Activation
conv (1)	1 624 × 16	1 × 16 × 6	2	SELU
average-pool(1)	542 × 16	-	(4), 3	-
conv (2)	271 × 32	1 × 32 × 6	2	SELU
max-pool (2)	91 × 32	-	(4), 3	-
conv (3)	46 × 64	1 × 64 × 6	2	SELU
max-pool (3)	16 × 64	-	(4), 3	-
conv (4)	8 × 128	1 × 128 × 6	2	SELU
max-pool (4)	3 × 128	-	(4), 3	-
fc-output	9	384 × 9	-	Softmax

4 Results

It has been already established that accuracy is often not sufficient performance metric in SCA context but something like the key enumeration should be used to really assess the performance of classifiers [13,20]. The problem with accuracy is most pronounced in imbalanced scenarios since high accuracy can just mean that the classifier classified all measurements into the dominant class (i.e., the one with the most measurements). This phenomenon is well-known in the machine learning community. Since we consider in our experiments the Hamming weight model, we have imbalanced data where HW class 4 (all S-box outputs with the Hamming weight equal to 4) is the most represented one. In fact, on average HW4 is 70 times more represented than HW0 or HW8. Consequently, a classifier assigning all measurements into HW4 will have a relatively good accuracy ($70/256 \approx 27.3\%$) but will not be useful in SCA context. To denote such cases, we depict the corresponding accuracies in cells with the gray background color.

First, we briefly address the fact that we do not use template attack. The decision for this is based on previous works as listed in Sect. 1 where it is shown that machine learning and deep learning can outperform TA. Consequently, we keep our focus here only on techniques coming from the machine learning domain.

4.1 Accuracy

In Table 3, we give results for DPAcontest v4 dataset when considering 50 most important features. First, we can observe that none of the techniques have problems with obtaining high accuracy values. In fact, we notice a steady increase in the accuracy values as we add more measurements to the training/testing process. By comparing the methods simply by the accuracy score, we see that XGBoost reaches the highest performance, followed closely by Random Forest. When considering CNN, we see that only Naive Bayes is resulting in smaller accuracies. Interestingly, when considering 1 000 measurements scenario, we see that CNN actually has by far the best accuracy. We believe this to be due to a combination of a small number of measurements and a small number of features. For a larger number of measurements, CNN also needs more features in order to train a strong model.

Table 3. Testing results, DPAcontest v4, 50 features

Dataset size	NB	MLP	XGBoost	RF	CNN
1 000	37.6	44.8	52.0	49.2	60.4
10 000	65.2	81.3	79.7	82.4	77.2
50 000	64.1	86.8	88.8	87.9	81.4
100 000	66.5	91	92.1	90.3	84.5

In Table 4, we present results for DPAcontest v2 with 50 features. As observed in related work (e.g., [13,19,20]) DPAcontest v2 is a difficult dataset for profiled

attacks. Indeed, CNN here always assigns all the measurements into the class HW4. Additionally, although MLP does not assign all the measurements into HW4, by examining confusion matrices we observed that the prevailing number of measurements is actually in that class, with only a few ones belonging to HW3 and HW 5. Finally, we see that the best performing technique is XGBoost. The confusion matrix for XGBoost results reveals that even when the accuracy for XGBoost is similar as assigning all measurements into HW4, the algorithm is actually able to correctly classify examples of several classes. Since for this dataset we have the same imbalanced scenario as for DPAcontest v4, we can assume that the combination of high noise and imbalancedness represents the problem for CNNs. Additionally, our experiments indicate that with this dataset, the more complex the architecture, the easier is to assign all the measurements into HW4. Consequently, simpler architectures work better as there is not enough expressive power in the network to learn perfectly the training set. For this reason, the CNN architecture used in [19] works better for DPAcontest v2 since it is much simpler than the CNN architecture we use here.

Table 4. Testing results, DPAcontest v2, 50 features

Dataset size	NB	MLP	XGBoost	RF	CNN
1 000	14.4	28.8	28.8	25.6	28.8
10 000	10.6	28.3	27.3	25.8	28.2
50 000	12	26.6	26.6	25.3	26.7
100 000	11.7	27.1	27.1	25.8	27.1

Finally, in Table 5, we give results for the Random delay dataset with 50 features. We can observe that the accuracies are similar to the case of DPAcontest v2 but here we do not have such pronounced problems with assigning all measurements into HW4. In fact, that behavior occurs in only one case – CNN with 50 000 measurements.

Table 5. Testing results, Random delay, 50 features

Dataset size	NB	MLP	XGBoost	RF	CNN
1 000	20	22	27.32	26.8	21.2
10 000	22	26.7	24.9	27	28.2
50 000	25.6	27.6	26.3	26.9	27.1

One could ask why setting the limit to only 50 features? For many machine learning techniques, the complexity increases drastically with the increase in the number of features. Combining that fact with a large number of measurements and we soon arrive into a situation where machine learning is simply to slow for

practical evaluations. This is especially pronounced since only a few algorithms have optimized versions (e.g., supporting multi-core and/or GPU computation). For CNNs we do not have such limiting factors. In fact, modern implementations of deep learning architectures like CNNs enable us to work with thousands of features and millions of measurements. Consequently, in Table 6, we depict the results for CNNs for all three considered datasets when using all available features. For DPAcontest v4, we see improvements in accuracy in all cases, where for cases with more measurements we see drastic improvements. It is especially interesting to consider cases with 50 000 and 100 000 measurements where we reach more than 95% accuracy. These results confirm our intuition that CNNs need many features (and not only many measurements) to reach high accuracies. For DPAcontest v2, we see no difference when using 50 features or all the features. Although disappointing, this is expected: if our architecture was already too complex when using only 50 features, adding more features does not help. Finally, when considering the Random delay dataset, we see that the accuracies for two smaller dataset sizes decrease while the accuracy for 50 000 measurements increases where we do not see that all measurements are assigned to HW4 class. Again, this is a clear sign that when working with more complex datasets, having more features helps but only if it is accompanied by the increase in the number of measurements.

Table 6. Testing results for CNN, all features

Dataset size	DPAcontest v4	DPAcontest v2	Random delay
1 000	60.8	28.8	20.3
10 000	92.7	22.6	20.2
50 000	97.4	22.3	28
100 000	96.2	27.1	–

Naturally, a question can be made whether it is really necessary to use deep learning for such a small increase in accuracy when compared with computationally simpler techniques given in Tables 3, 4, and 5. Still, we need to note that while for CNNs having 100 000 measurements is not considered as a large dataset, for many other machine learning techniques this would be already a huge dataset. To conclude, based on the presented results, we clearly see cases where CNNs offer advantages over other machine learning techniques but we note there are cases where the opposite is true.

4.2 Success Rate and Guessing Entropy

Figures 2a until f give guessing entropy and success rate for all three datasets when using 50 000 traces in total. One can see from Figs. 2a and b that the correct secret key is found for nearly all methods already using less than 10 traces when considering DPAcontest v4. Interestingly, we see that the CNN architecture that

uses all the features is less successful than the one using only 50 features, which is opposite from the results on the basis of accuracy. The most efficient techniques are MLP and XGBoost, but in this scenario, we see that even a simple method like Naive Bayes is more than enough.

For DPAcontest v2, we see that NB is significantly outperforming all the other methods. This could be due to a fact that other methods are more prone to classify most of the measurements into HW4 and thus do not contribute significant information to recover the secret key. For the Random delay dataset, we observe that NB, XGBoost, and RF are the most efficient methods when considering guessing entropy. On the basis of the success rate, Naive Bayes and Random Forest are the best. To conclude, we can see that all machine learning techniques display consistent behavior for both metrics. This means that those algorithms have stable behavior in ranking not only the best key candidate but also the other key candidates.

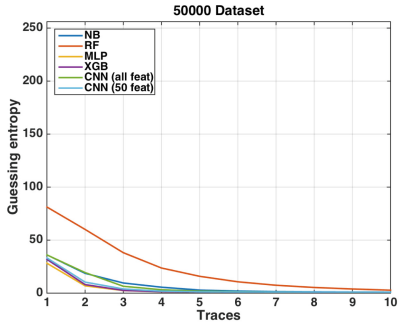
5 Discussion and Future Work

We start this section with a small experiment where we consider DPAcontest v4 and Random delay datasets with all features. We do not give results for DPAcontest v2 since even when using all features, all the measurements are classified into HW4 class. One reason why CNNs are so successful in domains like image classification is that they are able to maintain the topology, i.e., shuffling features in an image would result in a wrong classification. We do exactly that: we shuffle the features uniformly at random. Since the results indicate that the topological information in the trace signal used in experiments is not as important as expected, we tried to investigate such observation in depth by testing the CNN model with the extreme case. Expectedly, running our CNN architecture on such datasets results in decreased accuracy, but one that is still quite high as given in Table 7. When comparing Tables 6 and 7, we see that for DPAcontest v4, accuracy drops 10–15%. For Random delay and 10 000 measurements, the result is even slightly better after shuffling. In Figs. 3a and b, we give results for guessing entropy when using shuffled features. Interestingly, we see that shuffling the features did not significantly decrease the results for guessing entropy.

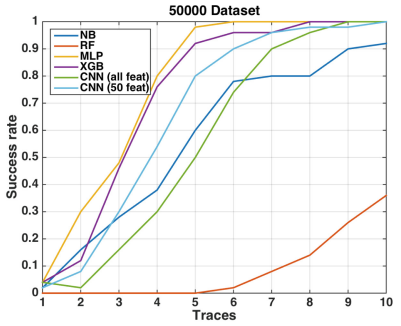
Table 7. Testing results for CNN, features shuffled

Dataset size	DPAcontest v4	Random delay
10 000	77.88	21.44
50 000	84.83	27.3
100 000	84.17	–

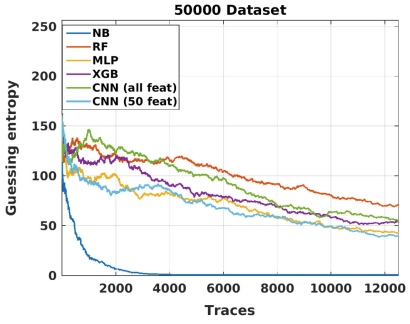
The results imply that the topological information between features of the trace is not more useful than the other characteristics of the signal. Thus, we



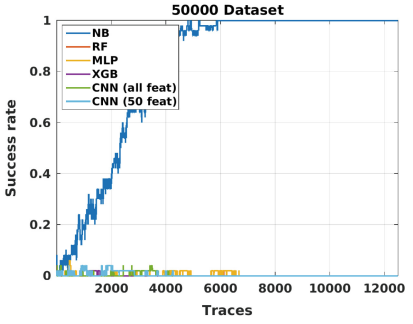
(a) DPAv4, guessing entropy, 50 000 measurements



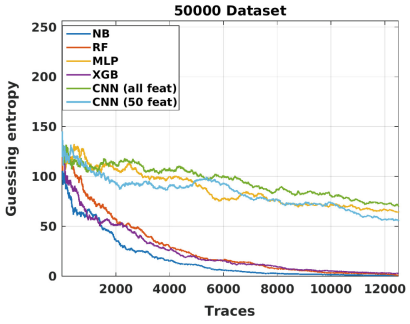
(b) DPAv4, success rate, 50 000 measurements



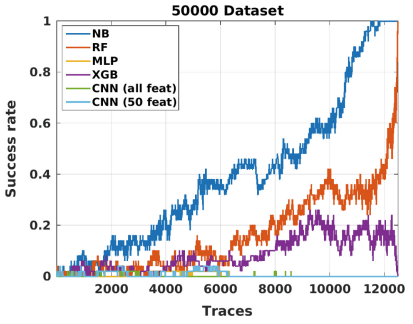
(c) DPAv2, guessing entropy, 50 000 measurements



(d) DPAv2, success rate, 50 000 measurements



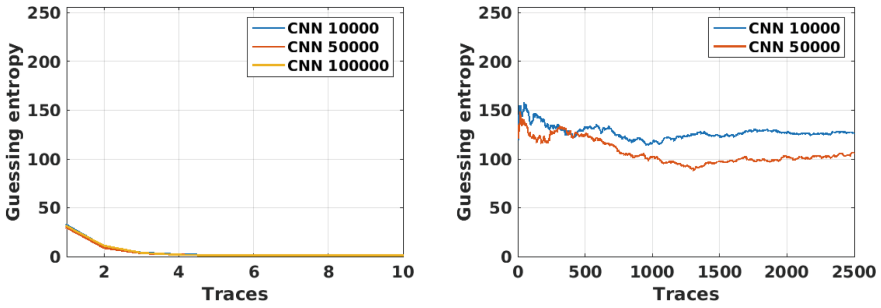
(e) Random delay, guessing entropy, 50 000 measurements



(f) Random delay, success rate, 50 000 measurements

Fig. 2. Guessing entropy and success rate results

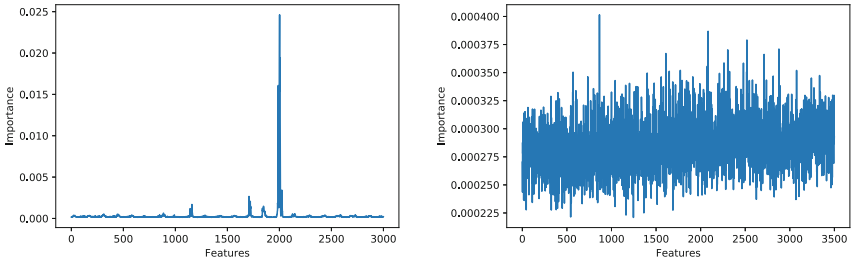
hypothesize that a specific local topology or a certain feature can be a more important factor than global topology, which is coherent to the fact that the independently selected subset of features shows a decent performance in previous



(a) DPAv4 (b) Random delay

Fig. 3. Shuffled features, guessing entropy

experiments. To verify such a hypothesis, we investigated the feature importance which is derived from the Random Forest classifier that is trained on all the features, which is illustrated in Fig. 4. Note that it is reported that the importance analysis is not reliable when it is applied on the features where they are intercorrelated such as the trace signals, compared to the features that are composed of independent variables [42]. To relax such problem, we applied the bootstrap without replacement, which is suggested in [42].



(a) DPAv4 (b) Random delay

Fig. 4. The feature importance derived from the Random Forest classifier trained on DPAcontest v4 dataset. Higher value indicates corresponding feature dimension is relatively more important than others. The values are normalized such that the sum of all the importance is equal to 1.

Figure 4a suggests the features near 2 000th dimension are treated as substantially more important to the RF model than the others. It partially explains the behavior of the CNN, whose main advantage is the ability to capture meaningful information in topology, which seems a less crucial factor in the DPAcontest v4 dataset. Differing from that, Fig. 4b shows there is no such region for the Random delay dataset that stands out when compared to other areas. Consequently, this implies the random delay applied in the dataset make the positional importance less influential.

Naturally, CNNs also have the implicit feature selection part. It is possible that current good results on SCA stem from that, which would mean we could use separate feature selection and classification to the same goal.

When considering deep learning architectures, and more specifically their sizes, a valid question is whether the architectures currently used in SCA are really deep. For instance, Cagli et al. mention their architecture as being "quite deep CNN architecture" but if we compare that with the state-of-the-art CNNs architectures used today, we see a striking difference. The current "best" architecture for image classification called ResNet has 152 hidden layers [43]. Our architectures look very shallow compared to that. Naturally, the question is if we even need such deep architectures, and if the answer is no, then maybe computationally simpler machine learning techniques could be a good alternative.

We do not need to investigate only the deep learning part. As Cagli et al. showed, using smart preprocessing (e.g., data augmentation) can bring a more striking increase in the performance than by changing the network architecture [20]. Machine learning domain is extensively using various data augmentation techniques for years and there is no reason why some of those, more general methods could not be used in SCA. Additionally, we must mention that data augmentation is not limited to deep learning and it would be interesting to see what would happen if SCA-specific data augmentation would be used with other, simpler machine learning techniques. Finally, in this work, we do not consider masked implementations, which could be the case where convolutional neural networks outperform other techniques. Still, when considering the related work it is not so clear whether this is a trait of CNNs or simply deep architectures [20,44].

When discussing the results on a more general level, we can observe some trends.

1. The number of measurements and the number of features are connected and simply increasing one quantity without the other does not guarantee an improvement in performance.
2. The level of noise in conjunction with the highly imbalanced data seem to affect CNNs more than some simpler machine learning techniques. Naturally, to reduce the level of noise, it is possible to use various forms of preprocessing and to reduce the imbalancedness, a simple solution is to undersample the most represented classes. This could be problematic in scenarios where we require a large number of measurements (but are limited in the amount we can acquire) since undersampling will drastically reduce the number of measurements we have at our disposal.
3. As a measure of performance in all algorithms, we use accuracy. When comparing the performance on the basis of accuracy vs guessing entropy, we can see there are differences and cases when accuracy cannot be used as a definitive measure of performance. Still, our results do not indicate that any of the tested algorithms are less sensitive to this problem.
4. CNNs are more computationally expensive to train and have more parameters than some other (simpler) machine learning techniques. This makes it

a challenging decision whether it is beneficial to invest more resources into tuning for a probably small improvement in the performance.
5. We see that one trained CNN architecture for a specific dataset is suboptimal on some other datasets. This indicates that the obtained models are not easily transferable across scenarios, which even more raises the concern about the computational costs vs. potential performance gains.

6 Conclusions

In this paper, we consider a number of scenarios for profiled SCA and we compare the performance of several machine learning algorithms. Recently, very good results obtained with convolutional neural networks suggested them to be a method of choice when conducting profiled SCA. Our results show that CNNs are able to perform very well but the same could be said for other machine learning techniques. We see a direct advantage for CNN architectures over machine learning techniques for cases where the level of noise is low, the number of measurements is large, and the number of features is high. In other cases, our findings suggest that other machine learning techniques are able to perform on a similar level (with much smaller computational cost) or even surpass CNNs. Of course, stating that CNNs perform well when the level of noise is low does not mean that some other machine learning technique we considered here is very good when the level of noise is high. Rather, when the level of noise is (very) high, we conclude that both CNNs and machine learning techniques have similar difficulties in classifying. When considering the guessing entropy metric, the results favor methods like Random Forest and XGBoost, which is a clear indication more experiments are needed to properly assess the strengths of convolutional neural networks. As discussed in previous sections, there are many possible research directions one could follow, which will, in the end, bring more cohesion to the area and more confidence in the obtained results.

References

1. Ronen, E., Shamir, A., Weingarten, A., O'Flynn, C.: IoT goes nuclear: creating a ZigBee chain reaction. In: IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, 22–26 May 2017, pp. 195–212. IEEE Computer Society (2017)
2. Chari, S., Rao, J.R., Rohatgi, P.: Template attacks. In: Kaliski, B.S., Koç, K., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 13–28. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-36400-5_3
3. Heuser, A., Rioul, O., Guilley, S.: Good is not good enough. In: Batina, L., Robshaw, M. (eds.) CHES 2014. LNCS, vol. 8731, pp. 55–74. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44709-3_4
4. Lerman, L., Poussier, R., Bontempi, G., Markowitch, O., Standaert, F.-X.: Template attacks vs. machine learning revisited (and the curse of dimensionality in side-channel analysis). In: Mangard, S., Poschmann, A.Y. (eds.) COSADE 2014. LNCS, vol. 9064, pp. 20–33. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21476-4_2

5. Schindler, W., Lemke, K., Paar, C.: A stochastic model for differential side channel cryptanalysis. In: Rao, J.R., Sunar, B. (eds.) CHES 2005. LNCS, vol. 3659, pp. 30–46. Springer, Heidelberg (2005). https://doi.org/10.1007/11545262_3
6. Choudary, O., Kuhn, M.G.: Efficient template attacks. In: Francillon, A., Rohatgi, P. (eds.) CARDIS 2013. LNCS, vol. 8419, pp. 253–270. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08302-5_17
7. Mitchell, T.M.: Machine Learning, 1st edn. McGraw-Hill Inc., New York (1997)
8. Heuser, A., Zohner, M.: Intelligent machine homicide. In: Schindler, W., Huss, S.A. (eds.) COSADE 2012. LNCS, vol. 7275, pp. 249–264. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29912-4_18
9. Hospodar, G., Gierlichs, B., De Mulder, E., Verbauwhede, I., Vandewalle, J.: Machine learning in side-channel analysis: a first study. J. Cryptogr. Eng. **1**, 293–302 (2011). https://doi.org/10.1007/s13389-011-0023-x
10. Lerman, L., Bontempi, G., Markowitch, O.: Power analysis attack: an approach based on machine learning. Int. J. Appl. Cryptol. **3**(2), 97–115 (2014)
11. 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. Cryptogr. Eng. **5**(2), 123–139 (2015)
12. Lerman, L., Medeiros, S.F., Bontempi, G., Markowitch, O.: A machine learning approach against a masked AES. In: Francillon, A., Rohatgi, P. (eds.) CARDIS 2013. LNCS, vol. 8419, pp. 61–75. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08302-5_5
13. Picek, S., Heuser, A., Guilley, S.: Template attack versus Bayes classifier. J. Cryptogr. Eng. **7**(4), 343–351 (2017)
14. 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), pp. 106–111, May 2015
15. Heuser, A., Picek, S., Guilley, S., Mentens, N.: Lightweight ciphers and their side-channel resilience. IEEE Trans. Comput. **PP**(99), 1 (2017)
16. Heuser, A., Picek, S., Guilley, S., Mentens, N.: Side-channel analysis of lightweight ciphers: does lightweight equal easy? In: Hancke, G.P., Markantonakis, K. (eds.) RFIDSec 2016. LNCS, vol. 10155, pp. 91–104. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-62024-4_7
17. Picek, S., et al.: Side-channel analysis and machine learning: a practical perspective. In: 2017 International Joint Conference on Neural Networks, IJCNN 2017, Anchorage, AK, USA, 14–19 May 2017, pp. 4095–4102 (2017)
18. 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.) AFRICACRYPT 2017. LNCS, vol. 10239, pp. 61–78. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-57339-7_4
19. Maghrebi, H., Portigliatti, T., Prouff, E.: Breaking cryptographic implementations using deep learning techniques. In: Carlet, C., Hasan, M.A., Saraswat, V. (eds.) SPACE 2016. LNCS, vol. 10076, pp. 3–26. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-49445-6_1
20. Cagli, E., Dumas, C., Prouff, E.: Convolutional neural networks with data augmentation against jitter-based countermeasures. In: Fischer, W., Homma, N. (eds.) CHES 2017. LNCS, vol. 10529, pp. 45–68. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66787-4_3
21. Chollet, F., et al.: Keras (2015). https://github.com/fchollet/keras
22. Abadi, M., et al.: TensorFlow: large-scale machine learning on heterogeneous systems (2015). Software available from tensorflow.org

23. Wolpert, D.H.: The lack of a priori distinctions between learning algorithms. Neural Comput. **8**(7), 1341–1390 (1996)
24. Bellman, R.E.: Dynamic Programming. Dover Publications, Incorporated, Mineola (2003)
25. Hughes, G.: On the mean accuracy of statistical pattern recognizers. IEEE Trans. Inf. Theory **14**(1), 55–63 (1968)
26. Hornik, K.: Approximation capabilities of multilayer feedforward networks. Neural Netw. **4**(2), 251–257 (1991)
27. Friedman, N., Geiger, D., Goldszmidt, M.: Bayesian network classifiers. Mach. Learn. **29**(2), 131–163 (1997)
28. Collobert, R., Bengio, S.: Links between perceptrons, MLPs and SVMs. In: Proceedings of the Twenty-First International Conference on Machine Learning, ICML 2004, p. 23. ACM, New York (2004)
29. Friedman, J.H.: Greedy function approximation: a gradient boosting machine. Ann. Stat. **29**, 1189–1232 (2000)
30. Chen, T., Guestrin, C.: XGBoost: a scalable tree boosting system. CoRR abs/1603.02754 (2016)
31. Breiman, L.: Random forests. Mach. Learn. **45**(1), 5–32 (2001)
32. LeCun, Y., Bengio, Y., et al.: Convolutional networks for images, speech, and time series. In: The Handbook of Brain Theory and Neural Networks, vol. 3361, no. 10 (1995)
33. Van Den Oord, A., et al.: WaveNet: a generative model for raw audio. arXiv preprint arXiv:1609.03499 (2016)
34. Demuth, H.B., Beale, M.H., De Jess, O., Hagan, M.T.: Neural Network Design. Martin Hagan (2014)
35. Kingma, D.P., Ba, J.: Adam: a method for stochastic optimization. CoRR abs/1412.6980 (2014)
36. Standaert, F.-X., Malkin, T.G., Yung, M.: A unified framework for the analysis of side-channel key recovery attacks. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 443–461. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-01001-9_26
37. TELECOM ParisTech SEN research group: DPA Contest, 4th edn (2013–2014). http://www.DPAcontest.org/v4/
38. TELECOM ParisTech SEN research group: DPA Contest, 2nd edn (2009–2010). http://www.DPAcontest.org/v2/
39. Coron, J.-S., Kizhvatov, I.: An efficient method for random delay generation in embedded software. In: Clavier, C., Gaj, K. (eds.) CHES 2009. LNCS, vol. 5747, pp. 156–170. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-04138-9_12
40. James, G., Witten, D., Hastie, T., Tibshirani, R.: An Introduction to Statistical Learning. STS, vol. 103. Springer, New York (2013). https://doi.org/10.1007/978-1-4614-7138-7
41. Klambauer, G., Unterthiner, T., Mayr, A., Hochreiter, S.: Self-normalizing neural networks. arXiv preprint arXiv:1706.02515 (2017)
42. Strobl, C., Boulesteix, A.L., Zeileis, A., Hothorn, T.: Bias in random forest variable importance measures: illustrations, sources and a solution. BMC Bioinform. **8**(1), 25 (2007)
43. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. CoRR abs/1512.03385 (2015)
44. Timon, B.: Non-profiled deep learning-based side-channel attacks. Cryptology ePrint Archive, Report 2018/196 (2018). https://eprint.iacr.org/2018/196