On the Performance of Deep Learning for Side-channel Analysis

Stjepan Picek[1], Ioannis Petros Samiotis[1], Annelie Heuser[2], Jaehun Kim[1], 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 at Nanyang Technological University, Singapore
[4] IRISA/Inria, Rennes, France

Abstract. Profiled side-channel attacks represent the most powerful category of side-channel attacks. There we have a number of methods promising to work well in a number of different scenarios. Still, the area is constantly improving: we started with template attack and then went into different machine learning techniques that outperformed template attack in certain settings. Recently, deep learning techniques brought promise of even better results. In this paper, we ask a question whether deep learning is actually better than machine learning, and if yes, in what situations exactly. To this end, we compare several machine learning techniques and a well-known deep learning technique – convolutional neural networks in a number of scenarios. Our results point that convolutional neural networks indeed outperforms machine learning in several scenarios but that often there is no compelling reason to use such a complex technique. In fact, if comparing techniques without extra steps like pre-processing, we see obvious advantage for deep learning only when the level of noise is small, the number of measurements is high, and the number of features is high. All other tested situations actually show that machine learning, for a significantly lower computational cost, performs the same or even better. Finally, we conduct a small experiment that opens the question whether convolutional neural networks are actually the best choice in SCA context.

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

1 Introduction

Today, in side-channel analysis (SCA) domain, profiled attacks are recognized as the most powerful ones. There, the attacker has access to a device and is consequently able to find out the secret key for a profiling device. Afterwards, he can use that knowledge to extract a secret from a different device. Already from this short description, we can observe that 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 known as the attack (test) phase.

We start with template attack (TA) [1], a technique that is still the best (optimal) from an information theoretic point of view if the attacker has unbounded number of traces and those traces follow Gaussian distribution [2, 3]. After the template attack, soon emerged the stochastic attack that uses linear regression in the profiling phase [4]. In coming years, researchers recognized certain weaknesses in template attacks and they tried to modify them 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 [5].

Alongside such techniques, SCA community recognized that the same general approach is actually used in other domains in the form of supervised machine learning. Machine learning (ML) is a term encompassing a number of methods that can be used for tasks like clustering, classification, regression, feature selection, etc [6]. Consequently, SCA community started to experiment with different ML techniques and to evaluate whether they are useful in the SCA context. When considering other domains the results look somewhat sparse but still there is a number of papers considering machine learning and side-channel attacks, see e.g., [3, 7–9, 9–16]. Although considering different scenarios and often different ML techniques (with some algorithms used in prevailing number of papers like Support Vector Machines and Random Forest), all those papers 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, we are evident that deep learning (DL) techniques start to capture attention of the SCA community. This is quite natural since DL techniques are so successful in other domains and there is no reason why similar behavior should not be observed in the SCA domain. Additionally, if DL techniques are able to surpass ML techniques in other domains, once again there is no reason why not to expect the same behavior in SCA. Accordingly, the first results confirmed that expectations. In 2016, Maghrebi et al. conducted the first analysis of DL techniques for profiled SCA as well as a comparison against a number of ML techniques [17]. The results were very encouraging with deep learning surpassing ML and TA.

After less than one year, another paper focusing on a DL technique called Convolutional Neural Networks (CNNs) showed impressive results: again deep learning was better performing than TA but also it was successful against device protected with different countermeasures [18]. 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 paper, we take a step back and investigate a number of profiled SCA scenarios. We compare one DL technique that got the most attention in SCA community up to now – CNNs against several, well-known “classical” machine learning techniques. Our goal is to assess whether the deep learning approach is truly the best and should be selected without any questions or there could be some more viable techniques (from complexity, explainability, ease of use or any other perspective). We emphasize that the aim of this paper is not to doubt deep learning as a good approach but to doubt it as the best approach in any profiled SCA setting. Consequently, the main contributions of this paper are:

1. We conduct a detailed comparison between deep learning and machine learning techniques in an effort to recognize situations where deep learning offers clear advantages over machine learning. We especially note that we use XGBoost ML technique that is well-known as an extremely powerful technique but has never before been used in SCA.
2. We propose a deep learning architecture called "SCANet" that is able to reach high-quality results and compete with ML techniques as well as with the other deep learning architecture designed in [17]. We hope that this will inspire other researchers to publish their designs so better performance comparisons are possible. To that end, having a publicly available repository of DL architectures designed for SCA would greatly benefit the area and increase the strength and confidence in results.

The rest of this paper is organized as follows. In Section 2 we provide necessary details about profiled side-channel analysis and machine learning. Section 3 gives details about datasets we investigate and results obtained. Section 4 provides discussion on relevance of these results, their significance for profiled side-channel analysis, and possible future research directions. Finally, Section 5 offers a brief conclusion.

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 unlimited number of traces available during the profiling phase, this is of course always bounded.

2.2 Machine Learning Techniques

We select several techniques from machine learning domain to be tested against deep learning approach. The selection is done on the basis of results from previous work and diversity of ML families. More precisely, we select one algorithm based on Bayes theorem (Naive Bayes), one linear method (logistic regression), one instance-based method (Support Vector Machine), one tree-based method (Extreme Gradient Boosting), and finally, one neural network algorithm that does not belong to the deep learning (Multi Layered Perceptron).

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 [19]. To find the best model for any 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 ML notions. The first one is connected with the curse of dimensionality [20] and the Hughes effect [21], which states that with a fixed number of training samples, the predictive power reduces as the dimensionality increases. This indicates clearly 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 [22]. Consequently, by adding hidden layers and neurons, the networks gain more approximation power.

Finally, when giving results we also include a classifier called Zero-R. This is the simplest classification method which relies on the class label only. Zero-R classifier simply predicts the majority category (class). Zero-R results are the baseline case: if some classifier reaches the same accuracy as Zero-R, it means it classifies all records as belonging to the most frequent class.

Naive Bayes – NB. 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. 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. Naive Bayes does not have any parameters to tune. Further information about the Naive Bayes algorithm can be found in [23].

Logistic Regression – LR. Multinomial logistic regression uses a linear predictor function $f(k, i)$ to predict the probability that observation i has the outcome k, of the form $f(k, i) = \beta_{0,k} + \beta_{1,k}x_{1,i} + \ldots + \beta_{M,k}x_{M,i}$ where $\beta_{M,k}x_{M,i}$ is a regression coefficient of the mth variable and the kth outcome. The β coefficients are estimated using the maximum likelihood estimation, which requires finding a set of parameters for which the probability of the observed data is the greatest [24]. Note that logistic regression would have the same model as the single layer perceptron with logistic (sigmoid) activation function. For LR, we tune the margin parameter C in the range $[0.001, 0.01, 0.1, 1, 10]$. A low cost of the margin parameter C makes the decision surface smooth, while a high C aims at classifying all training examples correctly.

Multi Layer Perceptron – MLP. Multi Layer 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. A perceptron is a linear binary classifier applied to the feature vector. Each vector component has an associated weight w_i. A perceptron classifier works only for data that are linearly separable, i.e., if there is some hyperplane that separates all the positive points from all the negative points [6]. Differing from linear perceptron, MLP can distinguish data that are not linearly separable. MLP consists of 3 or more layers (since input and output represent two layers) of nonlinearly-activating nodes [25]. 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 in tested in the range $[1, 2, 3, 4, 5]$ and the number of neurons per layer in the range between 10 and 50 – $[10, 20, 30, 40, 50]$.

Support Vector Machines – SVM. Support Vector Machine is a kernel based machine learning technique used to accurately classify both linearly separable and linearly inseparable data [26]. The general idea for scenarios with not linearly separable data is to transform them to a higher dimensional space by using a transformation kernel function. In this new space, the samples can usually be classified with a higher accuracy. Many types of kernel functions have been developed, with the most used ones being polynomial and radial-based. As a learning method, a Sequential Minimal Optimization (SMO) algorithm is used [27, 28]. For multi-class classification purpose as we consider in this paper, SVM is adapted to perform $n \times (n - 1)/2$ binary classifications. We consider only radial kernel in our experiments since it is known to be a good choice when there is no expert knowledge about the problem, the number of features is not too high, and there is no linear separation between the data [29]. We tune the radial kernel parameter γ and the margin parameter C. The radial kernel parameter γ defines how much influence a single training example has, where the larger γ is, the closer other examples must be to be affected. The margin parameter C has the same meaning as in the case of logistic regression (see Section 2.2). We tune C in the range $[0.001, 0.01, 0.1, 1, 10]$ and γ in the range $[0.001, 0.01, 0.1, 1, 10]$.

Extreme Gradient Boost – XGB. XGBoost is a scalable implementation of gradient boosting decision tree algorithm [30]. 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 [31]. As the name suggests, its core part is gradient boosting. 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. Gradient boosting is a technique where new models are created that predict the residuals or errors of prior models and then added together to make the final prediction. The concept is called gradient boosting since it uses a gradient descent algorithm to minimize the loss when adding new models. 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 and being (somewhat) easier to understand. XGBoost has many parameters that are possible to tune but here we select only the two most important (in our experience) ones: the learning rate and the number of estimators. For learning rate, we make a grid search within range $[0.0001, 0.001, 0.01, 0.1, 1]$ and for the number of estimators we tune in the range $[100, 200, 300, 400, 500]$.

2.3 Deep Learning

For our Deep Learning system, we use 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 [17,18]. Note that throughout the paper, when we are talking about deep learning, we actually consider only convolutional neural networks.

Our network is composed of 4 convolutional layers and 4 pooling layers in between, followed by the classification layer. The final architecture was chosen after creating hyper-parameter constraints based on the literature and tests we conducted, followed by an optimization on their values through a random search. The hyper-parameters 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.

All convolutional layers use kernel size of 6 and stride 2 creating a number of activation maps for each layer. 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. For pooling we use Average Pooling on the first pooling layer and Max Pooling on the rest, using kernel of size 4 and stride 3. The convolutional layers use "Scaled Exponential Linear Unit" (SELU) activation function, an activation function which induces self-normalizing properties and it was first introduced by [34]. In the classification layer, we use Softmax activation function combined with the Categorical Cross Entropy loss function. Finally, 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.

During the training, we use Early Stopping to further avoid overfitting by monitoring the loss on the validation set [35]. Thus, every training session is interrupted before reaching high accuracy on 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 [36].

The hardware we used has given us some constraints on the model creation due to the available computational power and memory. The GPU unit is an NVIDIA GTX 1050 Ti with Pascal architecture and 4GB of memory. The unit is capable of fast computation of complex computational tasks but lacks in memory compared to the other NVIDIA cards popular in current deep learning research. This has an impact on the network's architecture (few activation maps and not more than 5 Convolutional layers) and prevent us from exploring deeper and more complex network configurations. For reference purposes, the network architecture introduced above will be called SCANet in the following sections and we depict it in Figure 1. Additionally, we give details about our network in Table 1.

Table 1: SCANet architecture.

Layer	Output Shape	Weight Shape	Sub-Sampling	Activation
conv(1)	1624 x 16	1 x 16 x 6	2	SELU
average-pool(1)	542 x 16	-	(4), 3	-
conv(2)	271 x 32	1 x 32 x 6	2	SELU
max-pool(2)	91 x 32	-	(4), 3	-
conv(3)	46 x 64	1 x 64 x 6	2	SELU
max-pool(3)	16 x 64	-	(4), 3	-
conv(4)	8 x 128	1 x 128 x 6	2	SELU
max-pool(4)	3 x 128	-	(4), 3	-
fc-output	9	384 x 9	-	Softmax

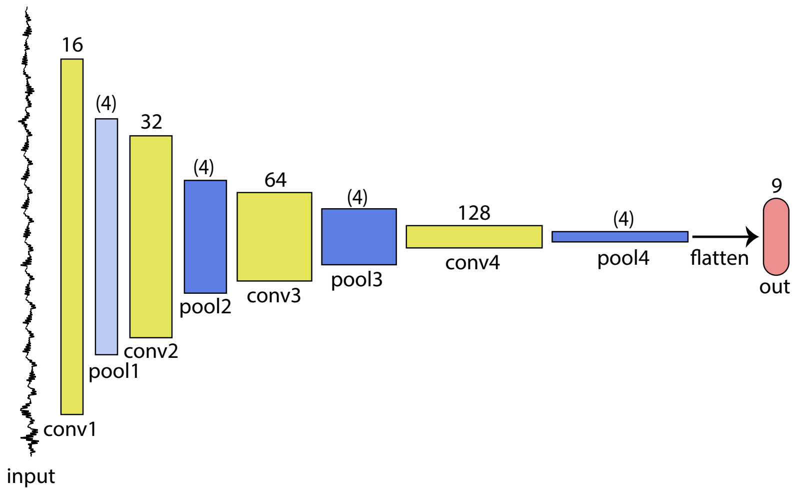

Fig. 1: The SCANet architecture. The simplified figure illustrates the applied architecture. The yellow rectangular blocks indicate 1-dimensional convolution layer, and the blue blocks indicate pooling layers. The first light blue block indicates average pooling, which is different from the other max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

3 Experiments

3.1 Datasets

We consider two datasets that mainly differ in the amount of noise and the side-channel leakage distribution. We use these datasets in order to allow for reproducibility of our results.

DPAcontest v2 [37]. DPAcontest v2 (denoted as 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:

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

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 [37]. In Eq. (1), $Y(k^*)$ consists in 256 values but we apply the Hamming weight (HW) on those values resulting in 9 classes. Note 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. This dataset has 3 253 features.

DPAcontest v4 [38]. The second dataset we investigate gives measurements of a masked AES software implementation (denoted DPAv4) 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:

$$Y(k^*) = \texttt{Sbox}[P_{b_1} \oplus k^*] \oplus \underbrace{M}_{\text{known mask}} , \tag{2}$$

where P_{b_1} is a plaintext byte and we choose $b_1 = 1$. Note that again we 9 classes scenarios corresponding to the Hamming weight of the output of S-box (as for DPAv2). Compared to the measurements from the DPAv2, SNR here is much higher and lies between 0.1188 and 5.8577. For our experiments we start with a preselected window of 3 000 features 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 order they appear in measurements and not for instance sorting them in accordance to their relevance).

3.2 Preparing the Datasets

First, we scale the features into $[0, 1]$ range. For all "classical" ML techniques, we divide measurements into 0.65:0.35 ratio for training and testing sets. While in training phase, the datasets are further divided into 5 stratified folds and evaluated by 5-fold cross-validation procedure. 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. Note that the training phase also contains a tuning phase (see Section 3.2) where we select the best parameters for each algorithm.

For deep learning, we do not conduct cross-validation since it is too computationally expensive but rather divide datasets into three parts in ratio: 0:65:0.20: 0.15. The first set is the training set, the second one is the validation set (that serves as an indicator of early stopping to avoid overfitting), and finally, the third dataset is the testing dataset. There are small differences in dataset sizes between ML and DL cases but since all data is taken from the same distribution, we do not expect negative effects of a such design choice.

Since the number of features for DPAv2 and DPAv4 is too large for ML techniques, we conduct feature selection where we take the 50 most important features where we keep the lexicographical ordering of selected features. To select those features, we use Pearson correlation coefficient where we calculate it for the target class variables HW, which consists of categorical values that are interpreted as numerical values [39].

Algorithm Tuning In order to find the best hyper-parameters, we tune the algorithms with respect to their most important parameters. Note that tuning is done only for 10 000 measurements case and then the same parameters are used in other cases. Naturally, if we would tune parameters for each dataset, we could expect to improve slightly the performance but that improvement would be hardly worth the computational overhead. The results of the tuning phase are given in Table 2.

Table 2: Parameters selected after a tuning phase, 10 000 measurements.

Algorithm	Parameters
	DPAV4
LR	$C = 1$
MLP	$hidden\ layers = 50, 30, 10, 50,\ activation = tanh$
SVM	$C = 10, \gamma = 1$
XGB	$learning\ rate = 0.1,\ number\ of\ estimators = 500$
	DPAV2
LR	$C = 0.01$
MLP	$hidden\ layers = 50, 30, 10, 50,\ activation = relu$
SVM	$C = 10, \gamma = 10$
XGB	$learning\ rate = 0.1,\ number\ of\ estimators = 200$

For SCANet, we use convolutional kernel size of 6, pooling size of 4, stride on convolutional layer equal to 2, initial number of filters of 16, learning rate of 0.007. Next, the number of convolutional layers is equal to 4, number of pooling layers is equal to 4, and there are no fully connected layers. We use SELU activation function, adam optimization algorithm, and convolutional dropout of 0.06.

3.3 Results

For all ML techniques, we use scikit-learn library in Python 3.6 while for CNNs we use Keras with TensorFlow backend [40,41]. In all experiments, we use a set of randomly selected measurements (profiled traces) from DPAcontest v2 and DPAcontest v4 datasets with 9 HW classes. We use the accuracy measure to evaluate the performance of ML/DL techniques, where accuracy is the percentage of correctly classified instances: $ACC = \frac{TP}{TP+TN}$.

Cagli et al. noticed that accuracy is often not enough in SCA context but something like the key enumeration should be used to really assess the performance of classifiers [18]. We agree with that but also offer a slightly different perspective. The problem with the accuracy is most pronounced in imbalanced scenarios since high accuracy can just mean overfitting. This phenomenon is a well-known in ML community. To circumvent it, we give results for the Zero-R classifier to differentiate between cases where the classifier simply classifies all measurements into the most frequent class (in all our experiments, that is HW 4 class) and where it actually classifies into n classes. Finally, if in the training/validation phase one uses accuracy and in the final assessment (testing phase) some other measure, there could be a potential issue with the interpretation of results. Indeed, some suboptimal result with respect to accuracy in training phase could actually be optimal when considering some other measure in testing phase.

Before presenting results, we briefly address the fact that we do not use template attack. The decision for this is based on previous works as listed in Section 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 ML and DL domain. Then, the performance of TA can be estimated to be close but somewhat lower than for SVM, for instance.

Since the test dataset size slightly differs for ML and DL cases, we use one version of Zero-R for ML techniques (denoted $0 - R_{ML}$) and the second version for CNNs (denoted $0 - R_{CNN}$). Finally, besides using our SCANet architecture, we implement the deep learning architecture as given by Maghrebi et al. [17]. To ease the differentiation between that architecture and ours, we call this one SPACE (the conference name were it was published). Note that the SPACE neural network architecture we use is our adaptation based on the available information. We do not use the neural network presented by Cagli et al. [18] since the information provided in that paper can be interpreted in different ways, which could potentially lead to a different model implementation compared to the original model.

We start by depicting DPAv4 and DPAv2 with a scatterplot of each trace per dataset in Figure. Each data point is colored based on its Hamming weight value. We can clearly observe different separability of the features given by each dataset. For the visualization we use PCA to reduce the dimensionality of the features [42]. Notice how DPAv4 has much better class separability, which indicates this case to be much simpler when compared to DPAv2.

In Tables 3 and 4 we give results for DPAv2 when considering 50 best features and 3 253 features, respectively. As observed in related work (e.g., [12, 17, 18]) DPAv2 is a difficult dataset for profiled attacks. Indeed, we can see that for instance, SCANet overfits and classifies everything as HW 4. The same behavior is seen for SVM and two smallest dataset sizes for LR. Additionally, although MLP does not overfit in a sense that classifies all as HW 4, the prevailing number of measurements is actually in HW 4, while only few ones belong to HW3 and HW 5. Finally, we see that the best performing technique is XGB where the confusion matrix (table with classification results for each class) reveals that even when the accuracy for XGB is similar to $0 - R_{ML}$, the algorithm does not overfit and is able to correctly classify examples of several classes.

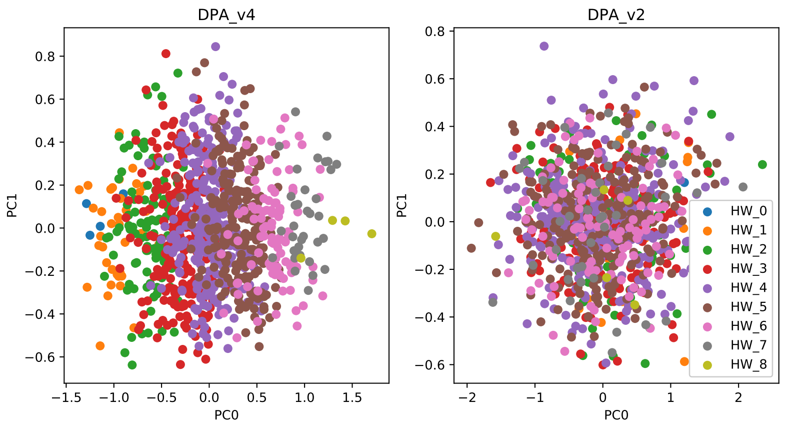

Fig. 2: Scatter plot for DPAv4 and DPAv2. Dimensionality reduction done with PCA.

Next, when we use all 3 253 features we see that DL is not able to improve significantly. Actually, for 1 000 and 10 000 measurements it still overfits while for the 50 000 and 100 000 measurements we have scenario where most of the measurements belong to class HW 4 and small portions to HW 3 and HW 5 (similar to the behavior of XGB for 50 features but with a more pronounced overfitting). Here, SPACE network is able to give somewhat better results than SCANet but we note that still most of the measurements is classified as HW 4. SPACE networks outperforms SCANet due to its simplicity – for noisy settings we see that simpler architectures work better since there is less chance of overfitting (they do not have enough complexity to overfit the data with such a complicated structure).

Table 3: Testing results, DPAcontest v2, 50 features

Dataset	$0-R_{ML}$	NB	LR	MLP	SVM	XGB	$0-R_{CNN}$	SCANet
1 000	0.308	0.131	0.308	0.271	0.308	0.267	0.253	0.253
10 000	0.282	0.067	0.282	0.269	0.282	0.277	0.275	0.275
50 000	0.274	0.116	0.274	0.271	0.274	0.273	0.267	0.267
100 000	0.274	0.108	0.273	0.273	0.274	0.273	0.273	0.273

Tables 5 and 6 give results for DPAv4 dataset when considering 50 features and 3 000 features, respectively. Differing from DPAv2 case, here we see that none of the algorithms (except, of course $0-R$) have any problems reaching high accuracies. Additionally, we see no overfitting which points us that the reason for overfitting in DPAv2 is actually due to the combination of noise and

Table 4: Testing results, DPAcontest v2, all features

Dataset	SCANet	SPACE
1 000	0.253	0.253
10 000	0.275	0.277
50 000	0.244	0.262
100 000	0.271	0.237

highly imbalanced datasets (while having only highly imbalanced dataset would not be a problem as evident from Table 5). When comparing ML and DL results for 50 features, we see that DL cannot compete with more powerful ML techniques. Interestingly, the best DL accuracy is obtained for 50 000 measurements and for the approximately the same accuracy we need SVM with only 1 000 measurements. Finally, on the basis of the presented results, we see that when the level of noise is small and we use only 50 measurements, SVM is by far the best performing technique with XGB as close second best.

Table 6 gives results for SCANet and SPACE architectures for DPAv4 when considering 3 000 features. This scenario reveals the true power of DL where for 50 000 measurements (SCANet) we are able to reach more than 99% accuracy on test set. Naturally, a question can be made whether it is really necessary to use DL for such a small increase in accuracy when compared with much computationally simpler techniques given in Table 5. Still, we need to note that while for DL having 100 000 measurements is not considered as a large dataset, we are approaching the limits for SVM since there the train complexity rises in cube power with the number of examples.

Table 5: Testing results, DPAcontest v4, 50 features

Dataset	$0 - R_{ML}$	NB	LR	MLP	SVM	XGB	$0 - R_{CNN}$	SCANet
1 000	0.297	0.639	0.477	0.834	0.823	0.725	0.267	0.693
10 000	0.272	0.669	0.557	0.867	0.924	0.886	0.268	0.811
50 000	0.275	0.654	0.601	0.866	0.955	0.913	0.273	0.851
100 000	0.274	0.662	0.607	0.866	0.960	0.916	0.269	0.845

When comparing SCANet and SPACE architectures, we see that SCANet performs better for all cases considered here. This is especially pronounced in cases with smaller number of measurements (e.g., 1 000 and 10 000 measurements). To conclude, based on presented results, we clearly see cases where DL offers advantages over ML but we note there are cases where the opposite is true.

Table 6: Testing results, DPAcontest v4, all features

Dataset	SCANet	SPACE
1 000	0.720	0.573
10 000	0.947	0.927
50 000	0.993	0.984
100 000	0.988	0.977

4 Discussion and Future Work

We start this section with a small experiment. Consider for example the DPAv4 dataset with all features and 10 000 measurements. Note that the traces are given in a form to keep the original topology as much as possible. One reason why CNNs are so successful in image classification is because they are able to maintain the topology, i.e., shuffling features in an image would results in a wrong classification. We do exactly that: we shuffle the features uniformly at random. Running the SCANet on such a dataset results in test accuracy of 0.966, which is even somewhat higher than with unshuffled features. Consequently, we see that the ordering of features does not negatively influence the successfulness of SCANet (or CNNs in SCA domain). This suggests that the topology preservation of CNNs is maybe not needed for SCA, which would mean we should not keep our focus solely on CNNs but to consider other deep learning techniques as well. Naturally, CNNs also have the implicit feature selection part. It is possible that current good results on SCA problems stem from that, which would in turn 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 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 "classical" machine learning or shallow neural networks could be a good alternative.

As shown in our experiment, the current deep models in SCA might not be optimal yet. Consequently, we emphasize the need for more publicly available datasets to conduct experiments. If we do not have enough public datasets for more in-depth investigation of deep learning in side-channel analysis, it could keep the field from the better understanding of the issues we encounter.

We have results up to now for only a few algorithms (where CNNs immediately got the most of attention) and architectures. If the topology can be relevant information after all, this could mean that one can still get certain additional benefit even from (shallow) CNN architectures. Alternatively, other deep learning (or even machine learning) techniques could be a good choice. Deep learning domain is an extremely fruitful field with many designs that are proposed on a regular basis. Unfortunately, it is hard to expect that all novelties can be tested in SCA context but considering at least a part of those would hopefully bring new milestones in profiled SCA.

Finally, we do not need to investigate only the deep learning part. As Cagli et al. showed, using smart pre-processing (they used custom data augmentation) can bring more striking increase in the performance than by changing the network architecture [18]. 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 techniques would be used with "classical" machine learning.

5 Conclusions

In this paper, we consider a number of scenarios for profiled SCA and compare the performance of several ML algorithms and two CNNs. Recently, very good results obtained with deep learning suggested that it should be a method of choice when conducting profiled SCA. We agree that deep learning is able to perform very well but the same could be said for several ML techniques.

We see a direct advantage for CNNs architectures over machine learning techniques only for cases where the level of noise is low, the number of measurements is high, and the number of features is high. In all the other cases, our results suggest that ML is able to perform on a similar level (with much smaller computational cost) or even surpass CNNs. We propose here a CNNs architecture called SCANet that is especially designed for side-channel attacks. We tune our architecture to be more beneficial for settings with low noise since in high noise scenario we see the need for additional data processing. Without additional data processing, high noise settings work better with shallow architectures, which means we require a different architecture for a high noise setting compared to a low noise setting.

Naturally, the experiments we conducted here (and in general, all experiments done up to now with deep learning for SCA) are not sufficient to reach definitive conclusions. 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. Chari, S., Rao, J.R., Rohatgi, P.: Template Attacks. In: CHES. Volume 2523 of LNCS., Springer (August 2002) 13–28 San Francisco Bay (Redwood City), USA.
2. 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)

3. 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 Mangard, S., Poschmann, A.Y., eds.: Constructive Side-Channel Analysis and Secure Design - 6th International Workshop, COSADE 2015, Berlin, Germany, April 13-14, 2015. Revised Selected Papers. Volume 9064 of Lecture Notes in Computer Science., Springer (2015) 20–33
4. 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.
5. 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
6. Mitchell, T.M.: Machine Learning. 1 edn. McGraw-Hill, Inc., New York, NY, USA (1997)
7. 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
8. 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.
9. 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
10. 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
11. Lerman, L., Medeiros, S.F., Bontempi, G., Markowitch, O.: A Machine Learning Approach Against a Masked AES. In: CARDIS. Lecture Notes in Computer Science, Springer (November 2013) Berlin, Germany.
12. Picek, S., Heuser, A., Guilley, S.: Template attack versus bayes classifier. Journal of Cryptographic Engineering **7**(4) (Nov 2017) 343–351
13. 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
14. Heuser, A., Picek, S., Guilley, S., Mentens, N.: Lightweight ciphers and their side-channel resilience. IEEE Transactions on Computers **PP**(99) (2017) 1–1
15. Heuser, A., Picek, S., Guilley, S., Mentens, N.: Side-channel analysis of lightweight ciphers: Does lightweight equal easy? In: Radio Frequency Identification and IoT Security - 12th International Workshop, RFIDSec 2016, Hong Kong, China, November 30 - December 2, 2016, Revised Selected Papers. (2016) 91–104
16. 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
17. 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
18. 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
19. Wolpert, D.H.: The Lack of a Priori Distinctions Between Learning Algorithms. Neural Comput. **8**(7) (October 1996) 1341–1390
20. Bellman, R.E.: Dynamic Programming. Dover Publications, Incorporated (2003)
21. Hughes, G.: On the mean accuracy of statistical pattern recognizers. IEEE Transactions on Information Theory **14**(1) (1968) 55–63
22. Hornik, K.: Approximation capabilities of multilayer feedforward networks. Neural Networks **4**(2) (1991) 251 – 257
23. Friedman, N., Geiger, D., Goldszmidt, M.: Bayesian Network Classifiers. Machine Learning **29**(2) (1997) 131–163
24. Russell, S.J., Norvig, P.: Artificial Intelligence: A Modern Approach. 2 edn. Pearson Education (2003)
25. Collobert, R., Bengio, S.: Links Between Perceptrons, MLPs and SVMs. In: Proceedings of the Twenty-first International Conference on Machine Learning. ICML '04, New York, NY, USA, ACM (2004) 23–
26. Vapnik, V.N.: The Nature of Statistical Learning Theory. Springer-Verlag New York, Inc., New York, NY, USA (1995)
27. Platt, J.: Fast Training of Support Vector Machines using Sequential Minimal Optimization. In Schoelkopf, B., Burges, C., Smola, A., eds.: Advances in Kernel Methods - Support Vector Learning. MIT Press (1998)
28. Keerthi, S.S., Shevade, S.K., Bhattacharyya, C., Murthy, K.R.K.: Improvements to platt's smo algorithm for svm classifier design. Neural Comput. **13**(3) (March 2001) 637–649
29. Keerthi, S.S., Lin, C.J.: Asymptotic Behaviors of Support Vector Machines with Gaussian Kernel. Neural Comput. **15**(7) (July 2003) 1667–1689
30. Friedman, J.H.: Greedy function approximation: A gradient boosting machine. Annals of Statistics **29** (2000) 1189–1232
31. Chen, T., Guestrin, C.: Xgboost: A scalable tree boosting system. CoRR **abs/1603.02754** (2016)
32. 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)
33. 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)
34. Klambauer, G., Unterthiner, T., Mayr, A., Hochreiter, S.: Self-normalizing neural networks. arXiv preprint arXiv:1706.02515 (2017)
35. Demuth, H.B., Beale, M.H., De Jess, O., Hagan, M.T.: Neural network design. Martin Hagan (2014)
36. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. CoRR **abs/1412.6980** (2014)
37. TELECOM ParisTech SEN research group: DPA Contest (2$^{\text{nd}}$ edition) (2009–2010) `http://www.DPAcontest.org/v2/`.
38. TELECOM ParisTech SEN research group: DPA Contest (4$^{\text{th}}$ edition) (2013–2014) `http://www.DPAcontest.org/v4/`.
39. James, G., Witten, D., Hastie, T., Tibsihrani, R.: An Introduction to Statistical Learning. Springer Texts in Statistics. Springer New York Heidelbert Dordrecht London (2001)
40. Chollet, F., et al.: Keras. `https://github.com/fchollet/keras` (2015)

41. Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., Zheng, X.: TensorFlow: Large-scale machine learning on heterogeneous systems (2015) Software available from tensorflow.org.
42. Bohy, L., Neve, M., Samyde, D., Quisquater, J.J.: Principal and independent component analysis for crypto-systems with hardware unmasked units. In: Proceedings of e-Smart 2003. (January 2003) Cannes, France.
43. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. CoRR **abs/1512.03385** (2015)