On the Performance of Convolutional Neural Networks 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. In this paper, we ask a question whether convolutional neural networks are more suitable for SCA scenarios than some other machine learning techniques, and if yes, in what situations. Our results point that convolutional neural networks indeed outperforms 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 only 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 similar or even better. The experiments with the guessing entropy metric indicate that simpler methods like Random forest or XGBoost perform better than convolutional neural networks for the datasets we investigated. Finally, we conduct a small experiment that opens the question whether convolutional neural networks are actually the best choice in side-channel analysis context since there seems to be no advantage in preserving the topology of measurements.

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 as powerful as possible

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

Profiled side-channel attacks are recognized as the most powerful ones since they define the worst case security assumptions. There, the attacker has access to a clone device, which can be profiled for any chosen key. Afterwards, he can use that knowledge to extract a secret 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 known as the attack (test) phase. A well-known example of such an attack is template attack (TA) [2], 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 [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, 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 clustering, classification, regression, feature selection, etc [7]. Consequently, SCA community started to experiment with different ML techniques and to evaluate whether they are useful in the SCA context, see e.g., [4, 8–17]. 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, deep learning (DL) techniques started 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. 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 [18]. The results were very encouraging with deep learning surpassing ML 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 [19]. 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 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 paper 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 [18].
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 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 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 (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 [20]. 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 [21] and the Hughes effect [22], 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 [23]. 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 [24]. 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.

Multi Layer Perceptron – MLP. The 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. An MLP consists of three or more layers (since input and output represent two layers) of nonlinearly-activating nodes [25]. 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 [26]. 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 [27]. 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 and being (somewhat) easier to understand.

Random Forest – RF. The Random forest is a well-known ensemble decision tree learner [28]. 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 [29]. 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 [30]. Their usage in side-channel analysis has been encouraged by [18, 19]. As we explain in Section 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 on 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 [31]. 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 [32].

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 the 3 584 of GPU cores. Specifically, we implement the experiment with the Tensorflow [33] computing framework and Keras deep learning framework [34] to leverage the GPU computation.

2.4 Performance Analysis

To assess the performance of the classifiers (and consequently the attacker) we use accuracy: $ACC = \frac{TP+TN}{TP+FP+FN+TN}$.

Besides accuracy, we use also Success rate (SR) and Guessing entropy (GE) [35]. A side-channel adversary $A_{E_K,L}$ conducts experiment $\mathsf{Exp}_{A_{E_K,L}}$, with time-complexity τ, memory complexity m, and making Q queries to the target implementation of the cryptographic algorithm. The attack outputs a guessing vector g of length o, and is considered success if g contains correct key k^*. o is also known as the order of the success rate. The o^{th} order success rate of the side channel attack $\mathsf{A}_{\mathsf{E_K,L}}$ is defined as:

$$\mathrm{SR}^{o}_{A_{E_K,L}}(\tau, m, k^*) = \mathsf{Pr}[\mathrm{Exp}_{A_{E_K,L}} = 1]$$

The Guessing entropy measures the average number of key candidates to test after the attack. The Guessing entropy of the adversary $A_{E_k,L}$ against a key class variable S is defined as:

$$GE_{A_{E_K,L}}(\tau, m, k^*) = \mathsf{E}[\mathrm{Exp}_{A_{E_K,L}}]$$

3 Experimental Setting

3.1 Datasets

In this paper, we consider 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 random delay countermeasure. For all ML techniques, we use scikit-learn library in Python 3.6 while for CNNs we use Keras with TensorFlow backend [33, 34].

DPAcontest v2 Dataset [36]. 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:

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

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 [36]. 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. There are several available datasets under the DPAcontest v2 name, we use here the traces from the "template" set. This dataset has 3 253 features.

DPAcontest v4 Dataset [37]. 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 DPAcontest v2). 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).

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 [38]. 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. Note, this countermeasure has been shown to be prone to deep learning based side-channel [19]. Random delay is quite often used countermeasure in commercial products, while not modifying the leakage order (like masking).

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 sizes of S – $[1\,000, 10\,000, 50\,000, 100\,000]$[5]. The ratios for Tr, V, and Te equal 50% : 25% : 25%. All features are normalized into $[0, 1]$ range.

When using ML techniques, we do not use the validation part but instead 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

[5] For the Random delay dataset, we experiment with only the first three sizes since it has only 50 000 measurements

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 take the 50 most important features where we keep the lexicographical ordering of selected features. We take 50 features for ML techniques since we use large datasets and the number of features is one of two factors (the second one being the number of measurements) comprising the time complexity for ML algorithms. Additionally, 50 features is also taken in the literature as the design choice [12, 14]. 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]:

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

For CNNs, we do not conduct cross-validation since it is too computationally expensive but rather 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 Random forest, we tune the number of trees in the range $[10, 50, 100, 200, 500]$, with no limit to the tree size.

When dealing with convolutional neural networks, 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. We use 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

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	-

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 kernel of size 4 and stride equals 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 [40]. We depict our architecture in Figure 1 and give details about our it in Table 2.

Table 2: Developed CNN 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

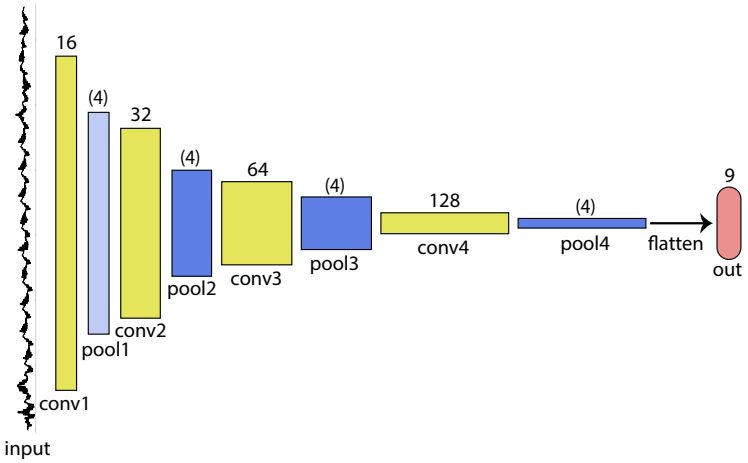


Fig. 1: The developed CNN architecture. The simplified figure illustrates the applied architecture. The yellow rectangular blocks indicate 1-dimensional convolution layer, and the blue blocks indicate pooling layers. The first light blue block indicates average pooling, which is different from the other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

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, 19]. 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 machine learning community. Since we consider in our experiments the Hamming weight model, we have imbalanced data where HW class 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 gray color.

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 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 into 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 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	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, 18, 19]) DPAcontest v2 is a difficult dataset for profiled attacks. Indeed, we can see that for instance, CNN always assigns all measurements into class HW4. Additionally, although MLP does not assign all the measurements into HW4, by examining confusion matrices we see that the prevailing number of measurements is actually in that class, with only few ones belonging to HW3 and HW 5. Finally, we see that the best performing technique is XGBoost where the confusion matrix 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 the high noise and imbalancedness represent the problem for CNN. What is more, our experiments indicate that with this dataset, the more complex the architecture the easier to assign all classes 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 [18] works better for DPAcontest v2 since it is much simpler than the CNN architecture we use here.

Finally, in Table 5, we five results for the Random delay dataset with 50 features. There, we observe that the accuracies are similar to the case of DPAcontest v2 but here we do not have such pronounced problems with assigning all

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

measurements into HW4. In fact, we see that behavior 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 rises drastically with the increase in the number of features. Coupling that with a large number of measurements and we soon arrive to 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 CNN 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 results 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 CNN needs many features (and not only many measurements) to reach high accuracies. For DPAcontest v2, we see no difference when using 50 features or all features. This, although disappointing is expected: if our architecture was already too complex when using only 50 features, adding more features does not make it simpler. 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. Again, this is a clear sign that when working with more complex datasets, having more features helps but only if it is accompanied with the increase in the number of measurements.

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 computation-

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	–

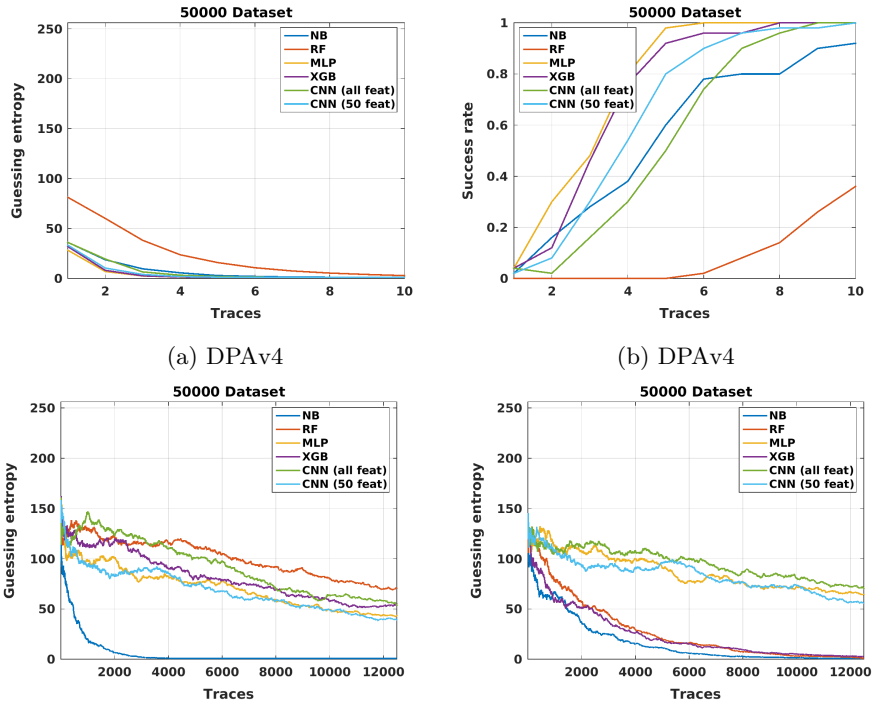
ally simpler techniques given in Table **??**. Still, we need to note that while for DL having 100 000 measurements is not considered as a large dataset, we are approaching the limits for SVM since there the train complexity rises in cube power with the number of examples. To conclude, based on presented 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

Figure 2 gives the guessing entropy of all three datasets and the success rate for DPAcontest v4 when using 50 000 data samples in total. One can see from Fig. 2a and 2a that the correct secret key is found for nearly all methods already using less than 10 traces. 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 for this scenario we see that even a simple method like Naive Bayes is more than enough. For DPAcontest v2 we see that NB is performing more efficient than all other methods. This could be due to a fact that other methods are more prone to classify most of the measurements into HW class 4 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.

5 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 measurements are given in a form to keep the original topology (ordering of features) 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 our CNN architecture on such a dataset results in test accuracy of 91.2%, which is only 1.5% worse than with unshuffled features. If we run the same experiment for DPAcontest v2 and Random delay, we see no change or even an increase of 2.7% in accuracy, respectively. Consequently, we



(a) DPAv4

(b) DPAv4

(c) DPAv2

(d) Random delay

Fig. 2: Side-channel metrics, 50 000 dataset

observe that the ordering of features does not negatively influence the successfulness of 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. These observations are also in accordance with results obtained with MLP in [42] Naturally, CNNs also have the implicit feature selection part. It is possible that current good results on SCA 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 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 more striking increase in the performance than by changing the network architecture [19]. 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 paper 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 [19, 42].

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

1. The number of measurements and number of features are connected and that simply increasing one quantity without the other does not guarantee an improvement in performance.
2. The level of noise in conjunction with highly imbalanced data seem to affect CNN 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 for instance 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 tested algorithms is less sensitive to this problem.
4. CNNs are more difficult 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 dataset. 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 only 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 machine learning is able to perform on a similar level (with much smaller computational cost) or even surpass CNNs. 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: 2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017, IEEE Computer Society (2017) 195–212
2. 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.
3. 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)
4. 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

5. 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.
6. 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
7. Mitchell, T.M.: Machine Learning. 1 edn. McGraw-Hill, Inc., New York, NY, USA (1997)
8. 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
9. 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.
10. 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
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. Cryptographic Engineering **5**(2) (2015) 123–139
12. 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.
13. Picek, S., Heuser, A., Guilley, S.: Template attack versus bayes classifier. Journal of Cryptographic Engineering **7**(4) (Nov 2017) 343–351
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). (May 2015) 106–111
15. Heuser, A., Picek, S., Guilley, S., Mentens, N.: Lightweight ciphers and their side-channel resilience. IEEE Transactions on Computers **PP**(99) (2017) 1–1
16. 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
17. 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
18. 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
19. 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
20. Wolpert, D.H.: The Lack of a Priori Distinctions Between Learning Algorithms. Neural Comput. **8**(7) (October 1996) 1341–1390
21. Bellman, R.E.: Dynamic Programming. Dover Publications, Incorporated (2003)

22. Hughes, G.: On the mean accuracy of statistical pattern recognizers. IEEE Transactions on Information Theory **14**(1) (1968) 55–63
23. Hornik, K.: Approximation capabilities of multilayer feedforward networks. Neural Networks **4**(2) (1991) 251 – 257
24. Friedman, N., Geiger, D., Goldszmidt, M.: Bayesian Network Classifiers. Machine Learning **29**(2) (1997) 131–163
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. Friedman, J.H.: Greedy function approximation: A gradient boosting machine. Annals of Statistics **29** (2000) 1189–1232
27. Chen, T., Guestrin, C.: Xgboost: A scalable tree boosting system. CoRR **abs/1603.02754** (2016)
28. Breiman, L.: Random forests. Machine Learning **45**(1) (2001) 5–32
29. 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)
30. 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)
31. Demuth, H.B., Beale, M.H., De Jess, O., Hagan, M.T.: Neural network design. Martin Hagan (2014)
32. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. CoRR **abs/1412.6980** (2014)
33. 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.
34. Chollet, F., et al.: Keras. `https://github.com/fchollet/keras` (2015)
35. Standaert, F.X., Malkin, T., Yung, M.: A Unified Framework for the Analysis of Side-Channel Key Recovery Attacks. In: EUROCRYPT. Volume 5479 of LNCS., Springer (April 26-30 2009) 443–461 Cologne, Germany.
36. TELECOM ParisTech SEN research group: DPA Contest (2^{nd} edition) (2009–2010) `http://www.DPAcontest.org/v2/`.
37. TELECOM ParisTech SEN research group: DPA Contest (4^{th} edition) (2013–2014) `http://www.DPAcontest.org/v4/`.
38. Coron, J., Kizhvatov, I.: An Efficient Method for Random Delay Generation in Embedded Software. In: Cryptographic Hardware and Embedded Systems - CHES 2009, 11th International Workshop, Lausanne, Switzerland, September 6-9, 2009, Proceedings. (2009) 156–170
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. Klambauer, G., Unterthiner, T., Mayr, A., Hochreiter, S.: Self-normalizing neural networks. arXiv preprint arXiv:1706.02515 (2017)
41. Maas, A.L., Hannun, A.Y., Ng, A.Y.: Rectifier nonlinearities improve neural network acoustic models. In: in ICML Workshop on Deep Learning for Audio, Speech and Language Processing. (2013)

42. Timon, B.: Non-profiled deep learning-based side-channel attacks. Cryptology ePrint Archive, Report 2018/196 (2018) https://eprint.iacr.org/2018/196.
43. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. CoRR **abs/1512.03385** (2015)