Data Under Siege: The Quest for the Optimal Convolutional Autoencoder in Side-Channel Attacks

van den Berg, D.; Slooff, T.; Brohet, M.; Papagiannopoulos, K.; Regazzoni, F.

DOI
10.1109/ijcnn54540.2023.10191779

Publication date
2023

Document Version
Author accepted manuscript

Published in
IJCNN 2023 conference proceedings

Data Under Siege: The Quest for the Optimal Convolutional Autoencoder in Side-Channel Attacks

Danny van den Berg[†], Tom Slooff[§], Marco Brohet[†], Kostas Papagiannopoulos[†] and Francesco Regazzoni[†§]
[†] University of Amsterdam, The Netherlands,
email: danny@dvandenberg.nl, m.j.a.brohet@uva.nl, k.papagiannopoulos@uva.nl, f.regazzoni@uva.nl
[§] Università della Svizzera italiana, Switzerland,
email: tom.slooff@usi.ch, regazzoni@alari.ch

***Abstract*—Encryption is a method to keep our data safe from third parties. However, side-channel information may be leaked during encryption due to physical properties. This information can be used in side-channel attacks to recover critical values such as the secret encryption key. To this end, it is necessary to understand the robustness of implementations to assess the security of data handled by a device. Side-channel attacks are one such method which allow researchers to evaluate the robustness of implementations using appropriate metrics.**

In the security community, machine learning is playing a prominent role in the study of side-channel attacks. A notable example of this is the use of Convolutional Autoencoders (CAE) as a preprocessing step on the measurements. In this work we study in depth the problem of finding the most suitable architecture of such Convolutional Autoencoders. To this end, Optuna is used to explore the CAE hyperparameter space. This process allows us to identify hyperparameters that outperform state-of-the-art autoencoders, reducing the needed traces for a succesful attack by approximately 37% in the presence of Gaussian noise and reducing the trainable parameters needed to attack desynchronization by a factor of 29. In addition to the promising results, experiments carried out in this paper allow a better understanding of the hyperparameter space in the field of side channel attacks, providing a solid base for future use of CAE in this specific domain.

***Index Terms*—Convolutional Autoencoder, Side-Channel Attack, Hyperparameter search, Encryption, Power Analysis, Desynchronization, Gaussian Noise, Random Delay Interrupts**

I. Introduction

Starting with the advent of internet and the consequent diffusion of digital applications, data has become increasingly central in our way of life. Data transfers can happen at short or long distances, for example from a computer to a mobile phone or from an edge or IoT device to a server in a remote location on the globe. Naturally, this increases the exposure to security risks. To protect our sensitive data from unauthorized access, we generally rely on encryption [1]. When encryption is used correctly, data can not be recovered unless in possess of the correct encryption key.

However, the use of secure encryption algorithms is not sufficient to ensure data protection. Though mathematically sound constructs may be implemented in a device, the physical properties of the implementation can unintentionally leak information to the outside world in what are called *side-channels*. This, in turn, can expose secret information such as the secret encryption key. A few examples of these side-channels are the device's power consumption, timing needed to complete the computation and electromagnetic radiation [2]–[6]. By far, the most studied side channel attack is power analysis, that exploits the correlation between the power consumed by the device and the secret data to infer the latter one [2], [3], [6], [7]. The focus of this paper is also on power analsys attacks. Because of this, in the remainder of the paper, when speaking about Side-Channel Attacks we will refer to Side-Channel Attacks using power analysis, unless differently specified.

To ensure the confidentiality of data, it is necessary to assess the physical implementation against Side-Channel Attacks (SCA) [8]. We generally distinguish SCA in two categories, the first category is *direct attacks* such as simple power analysis and differential power analysis. In direct attacks, the leaked information from the target device is used to directly model the sensitive values targeted, using simple leakage assumptions. The second category is *profiled attacks* such as the template attack and machine learning based attacks [9]. The profiling attacks consists of two phases, the first of which is to identify the target device and acquire a clone device of the target device. Information about the Side-Channel leakage is collected from the clone device and a leakage model is trained. The second phase is the attack phase in which the trained model is used to extract the sensitive values with leakage from the target device [10].

The family of profiling attacks is currently considered the most powerful one. This power comes from the use a clone of the target device to tune the attack before carrying out actual attack [4]. Deep-learning based profiling attacks are fairly new to the field of SCA [11] but are seen as a promising attack vector and therefore should be investigated further. This work is a step in this direction.

To stop or hinder attacks, countermeasures are implemented either in software, hardware or both. Countermeasures can be split into two categories: *Masking* and *Hiding* [10], [12]. Masking is when the sensitive values are split into different shares and processed separately. This is done to decrease the correlation between the sensitive values and the leaked information. The hiding category on the other hand works hiding the leakage in a direct manner. This can be done by adding noise or by adding random delays in the code or

hardware making the information more difficult to find [12].

The countermeasures can be seen as noise because of their randomness [12]. In addition to countermeasures, attacker has to deal also with the inherent noise due to the measurement setup. The problem is that classifiers tend to overfit when they are trained on noisy or high-dimensional traces [10]. This reduces their performance on the validation set as well as during the attack. A solution to the problem is to apply a divide and conquer method: first reducing the noise in the training data and then mounting the attack with a different mode. In this paper this will be done by reducing noise using a Convolutional Autoencoder. The main advantage of preprocessing the traces is less overfitting, however other advantages include: (1) The complexity and computational power of the second model can be reduced [10], (2) the ability to mount more effective attacks, (3) the ability to understand noise better and (4) the ability to understand the attack model better [12].

A. Related work

In SCA, power traces can be used to attack the target device. Power traces are a sequence of power measurements which have been made during the encryption. Recent works have introduced and shown promising results using a Convolutional AutoEncoder (CAE) to preprocess power traces with countermeasures in order to remove these countermeasures [10], [12], [13]. After preprocessing, the attack can be performed on traces which should be cleaner and thus easier to perform the attack with. Some research has been done on CAEs for profiled attacks by Wu et al. and Paguada et al. [10], [12]. Wu et al. made a single CAE to reduce the noise from 5 different countermeasures as well as some of these countermeasures combined. Paguada et al. explore the use of CAE further by setting new criteria for successful convolutional architectures for CAEs in the SCA space such as the use of dilation. Both of these papers however did not dive into how the hyperparameters and CAE architecture were found, therefore leaving a gap in knowledge. In addition to this, as far as the hyperparameter space is discussed in both papers, Wu et al. focused more on the optimizer, activation function, batch size, epochs, training size and validation size. Paguada et al. focused especially on stride and dilation. In our work we will focus on most of the parameters above, however in addition to this the latent space, number of layers, number of filters per layer, pool size and kernel size will be addressed as well due to the lack of prior exploration.

B. Contribution

Our paper improves the existing architectures for Convolutional Autoencoders in Side-Channel Attacks. These autoencoders convert power traces with countermeasures back to raw power traces in order to increase Side-Channel Attack performance. Our work explores a bigger hyperparameter space than previous works using the search framework Optuna. Three architectures will be given for the different countermeasures Gaussian noise, Desynchronization and Random Delay Interrupts. Following, the selection of hyperparameters will be discussed extensively and per countermeasure. This will allow to understand, for different SCA scenario's, which hyperparameters to use, when use it, and in what range should it be. The Gaussian noise architecture in this work is compared with related works [12], needing about 37.5 percent less traces for a successful attack. The desynchronization architecture needs slightly more traces than previous works, however the architecture needs 29 times fewer trainable parameters. Finally the Random Delay Interrupt network gives similar performance but being found by an automated process like the other two networks, it does not require manually searching the hyperparameter space.

II. Background

A. Convolutional neural network

A convolutional neural network (CNN) is a type of network that makes use of convolutions in the form of convolutional layers. Typically it also has pooling layers, and at the end one or more fully-connected layers. By using the convolutional layers the CNN is a solution to reducing the complexity of neural networks which have large data structures as an input [14]. An additional advantage of the use of convolutional layers is that the CNN is spatially or shift invariant. In other words, the position of the object being detected is not a determining factor [15].

The convolutional layer which gives the network its name is a special type of layer. In the case of power traces a 1-dimensional convolution is used. In the 1 dimensional convolutional layer a vector is used which acts as a filter. The vector slides across the data, every time it slides it multiplies the local region with its weights and then sums these. After doing this on the input a new vector consisting of all the regional computed numbers is made. The values of this new vector are first put through an activation function and then passed on to the next layer [15].

Both the size of the filter and the amount the filter or vector slides between its regional computations can be altered. The number of units it moves between each computations is called stride [10]. The stride is set to 1 by default, however by increasing it, the overlap of the filter each time it moves can be decreased as well as the size of the output. One convolutional layer usually consists out of multiple filters. Having multiple of these filters makes the network able to extract different features from the data within each layer [15].

ReLu has recently been the most often used activation function for CNNs. This is first of all because its computational costs is very low as the formula only selects the largest value out of 0 and its input.

$$ReLu(x) = max(0, x) \tag{1}$$

Another important reason for its common use is that the ReLu function has a constant gradient for all of the positive input whereas others have a near 0 gradient everywhere but

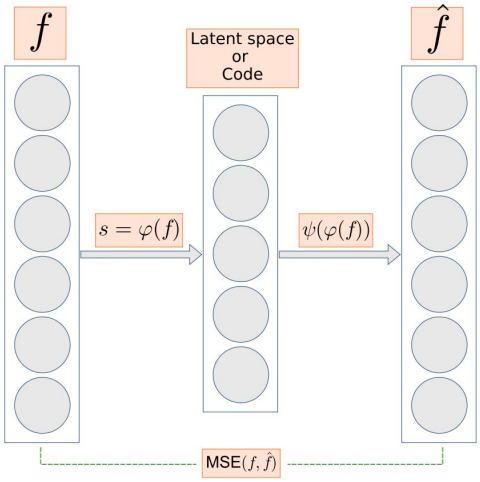

Fig. 1: The inner workings of an autoencoder illustrated by Paguada et al. [10]

near $x = 0$. Therefore the vanishing gradient problem as the network grows deeper can be avoided [15].

The pooling layer is a layer which is used to down-sample the data by using a pooling function, which in its turn decreases the complexity of both the data and the next layers in the model. Some examples of pooling functions are minimum pooling, average pooling and max pooling. The most commonly used pooling function however is the max pooling function. This function takes a subregion of the data and then downscales it by only passing the maximum value of each subregion on to the next layer. A very common size for the subregion is a 2x2 area [15].

Finally, the fully-connected layer usually makes up the last few layers. The fully-connected layers are completely the same as the layers in the multi-layer perceptron neural network, and thus consist out of nodes connected by weights. The fully connected layers are then the part of the CNN with the most parameters and takes long to train [16]. The ReLu function is then recommended again as it is low in computational cost [14]. These final layers compute the final class scores [12].

B. Autoencoders

Autoencoders [17], introduced in the context of training unsupervised networks with backpropagation, are designed to output its input back but with as much noise removed as possible [18]. The autoencoder consists of an encoder (φ) and a decoder (ψ). The encoder maps the input space (f) to the latent space (s). After this the decoder tries to reconstruct the input from the latent space, we call the reconstruction $\hat{f}$. An overview of this can be seen in Fig. 1. The goal is to minimize equation (2).

$$\mathrm{MSE}(f, \hat{f}) = \frac{1}{N} \sum_{i=1}^{N} (f_i - \hat{f}_i)^2 \tag{2}$$

When the latent space is chosen correctly and this formula gets optimized, the output of the autoencoder should resemble the input as much as possible.

In this work, the autoencoder will be used for removing the countermeasures which we consider to be noise. The input will therefore be traces with added countermeasures. Instead of using these as label as well as is conventional with autoencoders, clean traces will be used as label. Since the autoencoder can only bring part of the information through the smaller latent space only the most representative input features remain after the reconstruction [12]. This should result in removing the noise and recovering the raw traces as well as possible.

C. Optuna

Optuna [19] is a framework to efficiently search through the hyperparameter space of the machine learning model. It consists of *sampling* and *pruning* strategies.

The sampling strategy determines which next hyperparameter set to try, given the previously used sets and their performance. The default sampler in Optuna is the Tree-structured Parzen Estimator (TPE) [20]. TPE fits two Gaussian Mixture Models. One to the set of hyperparameters with the best objective values $f(x)$, and one to the remaining hyperparameters with worse values $g(x)$. It then calculates the ratio between them $\frac{f(x)}{g(x)}$. The next hyperparameter that will be evaluated is the one that maximizes the calculated ratio.

The pruning strategy determines how the performance is evaluated of a used hyperparameter set. The default pruner in Optuna is the median pruner. This pruner compares the trial's best intermediate result with the median of intermediate results of the previous trials at the same step. If the trial's best intermediate result is worse, it will be pruned, otherwise the trial can continue [19]. Optuna also takes an objective function to suggest hyperparameters.

D. Side-Channel Attacks

The first type of Side-Channel Attacks involves direct attacks such as simple power analysis or differential power analysis, where the target device is directly attacked for example by taking power consumption measurements from it and analysing these under simple model assumptions. This can reveal a lot of information about the target and the encryption code being deployed [21]. However, direct attacks need to make a priori assumptions about the device leakage and this may lead to overestimating the security level. The second type, profiled Side-Channel Attacks, instead offers powerful leakage modeling options [4], [9], [10], [22]–[24] thus improving the attack's success rate. In addition, they can already be effective in attacking with just a few traces available from the target device. In this work only profiled Side-Channel Attacks will then be evaluated. The profiled attacks that will be used are the multi-layer perceptron (MLP), Template Attack (TA) and Convolutional Neural Network (CNN). The reason for this being that these are the models used in both the works from

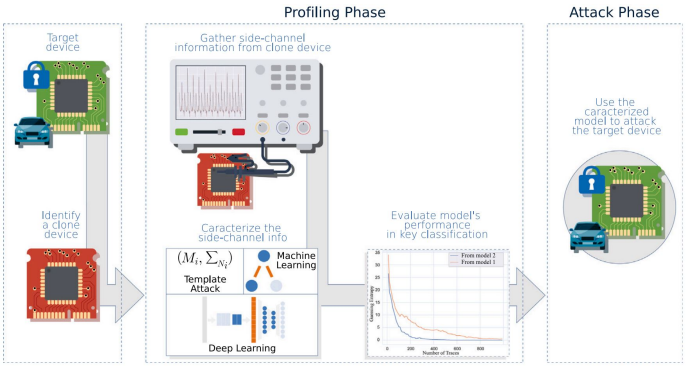

Fig. 2: Overview of a Profiled Side-Channel Attack [10]. Firstly a clone of the target device is acquired. Subsequently power measurements are collected from the device while performing cryptographic operations. Next a attack model is made using these measurements. Finally the model is used to attack the target device.

Wu et al. and Benadjila et al. making the results in this work easier to compare [4], [12].

1) Profiled Side-Channel Attacks: In a profiled Side-channel Attack first of all the target device is selected. When the target device is selected a clone of the target device is acquired in order to start the profiling phase [10], [24]. The profiling phase starts by gathering leakage traces, the input during the leakage trace and the labels corresponding to the leakage trace. These labels represent the sensitive values that the attack aims to discover. After this information is gathered the next step in the profiling phase is to train the model to fit on this information. The model is evaluated and if it performs well enough the attack phase can start. In this phase a limited number of inputs with their corresponding leakage traces is acquired from the target device. This information is fed to the model which should then then be able to construct the sensitive values from this information [10]. A brief overview of Profiled Side-Channel Attacks can be seen in Fig. 2.

2) Multi-layer perceptrons: One model that can be implemented in profiling attacks is the multi-layer perceptron. To be able to use it in SCA the number of nodes in the input layer should be the same as the amount of data points in the leakage traces. The hidden layers should then be designed on a case-by-case basis. The output layer should have the amount of nodes corresponding to the amount of classes that can be predicted in the specific SCA situation.

3) Convolutional neural network: Another model that can be employed in profiling attacks is the Convolutional neural network. The designing of the input layer and the output layer is similar to the MLP implementation. The input layer should also have the same shape as the leakage traces, and as well as the MLP the output layer which is a dense layer should have the same number of nodes as the amount of classes that can be predicted in the corresponding SCA case. The hidden layer can be a bit more difficult in design since it contains more types of layers but should also be designed on a case-by-case basis.

4) Template attack: The Template Attack (TA) [25] as well as the CNN and MLP belongs to the family of profiled attacks. It is a statistical technique which uses Bayes Theorem and mostly relies on the normal (Gaussian) distribution [23]. TA assumes that the features are dependent on each other and uses a multivariate Gaussian distribution to model this. A model using the multivariate Gaussian distribution is made for each key. This can be seen in the following formula.

$$\mathcal{N}(t|\mu_k, \Sigma_k) = \frac{1}{\sqrt{(2\pi)^N |\Sigma_k|}} e^{-\frac{1}{2}(t-\mu_k)^T \Sigma_k^{-1}(t-\mu_k)} \tag{3}$$

In (3) t is a trace, μ_k is the mean vector, and Σ_k is the covariance matrix these last two correspond to traces that belong to a certain encryption key (k).

$$\begin{aligned} \hat{k} &= \arg\max_k P(k|t_{attack}) \\ &\iff \arg\max_k P(t_{attack}|k)P(k) \\ &\iff \arg\max_k \mathcal{N}(t|\mu_k, \Sigma_k) \end{aligned} \tag{4}$$

After a model is made for each key candidate the traces from the target device are used to determine which model is most likely and thus which key has the highest probability. This is done using Bayes rule and can be seen in (4).

E. Countermeasures in power traces

As stated in the introduction most countermeasures can be divided into two categories: Masking and hiding [10], [12] this work will focus solely on the hiding category. Out of many hiding tactics a important tactic and one that will be focused on in this work is the addition of Gaussian noise. In the time dimension this work will focus on desynchronization of the trace and random delay interrupts which are implemented in software. Another example could however be clock jitters which are implemented in hardware. Since these countermeasures are implemented at random, we can consider these countermeasures to be noise and try to filter them out [12].

1) Gaussian noise: In Side-Channel leakage traces Gaussian noise is the most common type of noise. This is both due to countermeasures taken as well as the fact that it can occur due to the working environment. Sources can be the transistor, data buses, the record device or the wires to the record device. Some ways to implement Gaussian noise as a countermeasure are by doing parallel operations (algorithmic noise) or a dedicated noise engine or generator [12], [26]. In order to replicate the countermeasure, we added Gaussian noise to the leakage traces. To do so we simply added a value sampled from a Gaussian distribution to each data point in the leakage trace. The Gaussian distribution used in this work has a mean of 0 and a variance of eight. For the countermeasures Gaussian noise, Desynchronization as well as RDI they were added with the same settings and code as in [12].

2) Desynchronization: After leakage data from a device has been recorded the points of interest (POI) where the encryption starts have to be found in the data. Most of the time this process is done by finding a 'trigger' which is a distinctive point in all traces after which the (POI) start. Finding a bad trigger can result in a lot of noise between the trigger and POIs resulting in a desynchronized trace. Countermeasures can be taken so that more desynchronization happens. Since well synchronized traces can significantly improve the correlation between the traces and sensitive values [12] this will make it more difficult to perform SCA. To accomplish desynchronization we added random delays in a device's software code. This countermeasure adds noise to time dimension (x-axis) instead of the power consumption (y-axis). This countermeasure was already in the ASCAD database and has a maximum desynchronization value of 50 points.

3) Random Delay Interrupt (RDI): Another countermeasure which will be focused on in this work is the Random Delay Interrupt (RDI). Where the Desynchronization countermeasure moves all points left or right, the RDI interrupts the encryption process at random points in time. This can be done in practice by using the NOP operation. This countermeasure affects the time dimension more locally than the desynchronization and thus breaks up the trace in more pieces adding significant noise and randomness to the time dimension [12]. The Random Delay Interrupts were added using the Floating Mean method introduced in [27]. Parameter a was equal to 5 and parameter b equal to 3.

III. Method

First in this section, information will be given about the dataset used in the experiments. Secondly the model optimization will be discussed including the hyperparameter range for the autoencoder, how the hyperparamater space will be explored and a restriction which needs to be satisfied by the autoencoder. In addition, the hardware used for the experiments will be discussed and finally we describe the performance metric used to estimate the effectiveness of the CAE.

A. Data set

In this work the ASCAD database will be used. This database was introduced by Prouff et al. to serve as an open and common database for researchers so that improved and new architectures can be easily compared in the same settings [4]. Wu et al. used the same dataset in their work where they created the original CAE architecture [12]. The ASCAD database consists out of two data sets, one with fixed key encryption and one with random keys encryption. In this work only the fixed key dataset will be used. The traces in this dataset were created with a Atmega8515 8-bit microcontroller. The dataset consists out of 50 000 profiling traces and 10 000 attack traces and their corresponding labels. Each of these traces is time-aligned in a preprocessing step and contains 700 datapoints [4]. The profiling traces are used for training and validation for the CAE, CNN and the MLP. The attack traces are then used to simulate an attack after the profiling phase and measure the guessing entropy.

TABLE I: The hyperparameter search space

Hyperparameter	**Range**
Epochs	1-200
Activation function	ReLU, SeLU
Optimization function	RMSprop, SGD, Adam
Latent space (nodes)	1-700
Total convolutional layers	1-12
Number of filters	1-256
$\prod$ Pool size	1-350
Batch size	128
Kernel size	2-16
Dilation	1-4
Training set	40 000
Validation set	10 000

B. Model optimization

In this work, we aim to improve upon the existing Convolutional Autoencoders for SCA. To do so the findings from other papers were taken into account. One of these findings is that in recent works from Paguada et al. it was shown that using dilation in CNNs can be very useful [10]. This is due to the fact that convolutions start mapping redundant features when they are covering too much signal. When they do so they start representing non relevant information [28]. Therefore the amount of dilation will be taken into account when choosing the hyperparameters for the CAE. In Table I an overview is shown of the different hyperparameters and the range used to search in this work. These were mostly restricted due to either time constraints or hardware limitations.

Since the eventual goal of improving the CAE is to improve guessing entropy this would have been the best choice in measuring the performance of the CAE. However, to properly test the guessing entropy multiple attacks have to be performed and averaged which is too costly in terms of time. Additionally every time the guessing entropy would have to be measured for attacks with several different models such as a CNN, MLP and TA all of which would have to be trained as well. All together this would have taken too much time for the scope of this work. So instead the validation loss has been taken as a measure of performance in the search of the best hyperparameters for the CAE.

The hyperparameter space will be searched with Optuna using the TPE sampler and Median Pruner. In the exploration of this hyperparameter space with Optuna there is one main challenge to be faced. This challenge is that some of the hyperparameters that will be selected also have influence on the output size of the model. The dimensions and size of the output of an autoencoder should however always be the same as the input. Therefore the following formula needs to be taken as a constraint in the search of hyperparameters [12].

$$S_{dense0} = \frac{S_{clean}}{\prod_{i=1}^{n} S_{pool,i}} \cdot N_{filter0} \tag{5}$$

In (5) S_{dense0} is the amount of nodes in the first dense layer in between the encoder and decoder. The variable S_{clean} is

TABLE II: Gaussian noise CAE architecture

Layer	No. filters	Dilation	Kernel size	Pool	No. neurons
Conv	47	1	11	1	-
Conv	70	4	15	2	-
Conv	45	2	7	1	-
Conv	45	1	4	1	-
Conv	19	2	9	10	-
Conv	27	2	14	1	-
Flatten	-	-	-	-	-
Dense	-	-	-	-	635
Dense	-	-	-	-	945
Reshape	-	-	-	-	-
Conv	27	1	4	1	-
Conv	19	1	14	10	-
Conv	45	3	15	1	-
Conv	45	2	12	1	-
Conv	70	3	16	2	-
Conv	47	3	1	1	-

the input size of the autoencoder. The bottom of the fraction $\prod_{i=1}^{n} S_{pool,i}$ is the product of all max pool layers, and $N_{filter0}$ is the amount of filters in the first layer of the decoder.

Since the hyperparameters will be chosen by the sampler in Optuna it would be best to define this restriction in Optuna. However Optuna only supports soft restrictions which means it will try to stay within the restrictions but not guarantee it. In the case it goes outside of the restrictions however the code will error out and valuable training time would be lost. In order to ensure this will not happen, Optuna chooses $\prod_{i=1}^{n} S_{pool,i}$ freely. All valid combinations for the other two variables S_{latent} and $N_{filter0}$ will then be calculated and Optuna can pick one of these combinations.

1) Hardware: The search and training was done on GPUs, of which several were part of the Distributed ASCI Supercomputer (DAS) [29]. The used GPUs include the Nvidia RTX 3080, Nvidia GTX Titan X (pascall generation), Nvidia GTX Titan Xs (Maxwell), Nvidia RTX 2080 Ti and the Nvidia A40.

2) Guessing entropy: Guessing entropy is a standard evaluation metric in SCA [10], [30]. This is the metric that will also be used in this work to be able to compare the optimized autoencoder against other research. In an attack a vector is made with all key candidates sorted from most likely to least likely. The guessing entropy is the average position of the correct key candidate in this sorted vector [10], [31]. When the guessing entropy is equal to 0 it means that the correct key can typically be found by the model. However having a very low guessing entropy can be sufficient, since the attacker would only have to try out a limited number of keys to still complete the attack.

IV. Results

The results are represented by three graphs for every countermeasure. The first graph will be the Validation Loss of the CAE over the training in epochs. The second graph will show the validation loss for the CNN used as attack model. The last graph will show how the guessing entropy converges on average over 100 attacks. The vertical axis represents the guessing entropy and the horizontal axis in this graph is the amount of traces used for the attack.

TABLE III: Desynchronization CAE architecture

Layer	No. filters	Dilation	Kernel size	Pool	No. neurons
Conv	2	1	11	4	-
Flatten	-	-	-	-	-
Dense	-	-	-	-	138
Dense	-	-	-	-	350
Reshape	-	-	-	-	-
Conv	2	3	9	4	-

1) Gaussian Noise: First of all for the Gaussian noise CAE the activation function found by Optuna is Relu, the optimizer Adam and the number of epochs 25. This CAE does seem to have the CNN and MLP converge towards a Guessing entropy of 0 however not within the 10 000 traces attack data set as can be seen in Fig. 3 (c). As illustrated in Fig. 4 (c), training the same CAE for 100 epochs instead of 25 results in better performance compared to the CNN, MLP or TA on their own which all needed over 10 000 traces and previous autoencoder-CNN, MLP or TA combinations. This includes the autoencoder-CNN combination from Wu et al. [12] which was the best performing combination and needed about 8800 traces. Furthermore, the MLP suddenly stops converging towards a GE of 0. In addition to this denoising with averaging as done by Wu et al. still performs better than the CAE found in this work [12]. After denoising with the CAE it takes around 6000 traces for a GE of 0 while around 1800 traces suffice when averaging is used. A further exploration of the epoch hyperparameter has been done in steps of 5 in the range from 5 to 200 epochs. Within this range only spikes can be found in which the CAE delivers a better Guessing Entropy. The best performing models seem to be around the range of 100 epochs, but no clear trend can be found.

2) Desynchronization: As for the desynchronization CAE the activation function found by Optuna is Selu, the optimizer Adam and the number of epochs 171. In Fig. 5 (c) we can see the best found CAE architecture for desynchronization combined with the CNN manages to get a guessing entropy of 0 at roughly 1300 traces. This represents a significant improvement over the more than 9,000 traces required for a successful attack using only the CNN, or more than 10 000 traces required using TA or MLP. As demonstrated in [12], the CAE in their work needs approximately 800 traces to converge to a value of 0. However, the desynchronization CAE found in this paper is less computationally expensive having about 97 000 trainable parameters instead of around 2.8 million trainable parameters. Furthermore the automated search gets close in performance without the need for the researcher to spend valuable time searching.

3) Random Delay Interrupts: The Activation function for the RDI CAE found by Optuna is Selu, the optimizer Adam and number of epochs 161. In Fig. 6 (c) we see that in combination with the CNN as attack model it manages to converge to a guessing entropy of 0 at about 1500 traces. This is again a great result quite similar to the network of [12] which achieved a guessing entropy of 0 at approximately

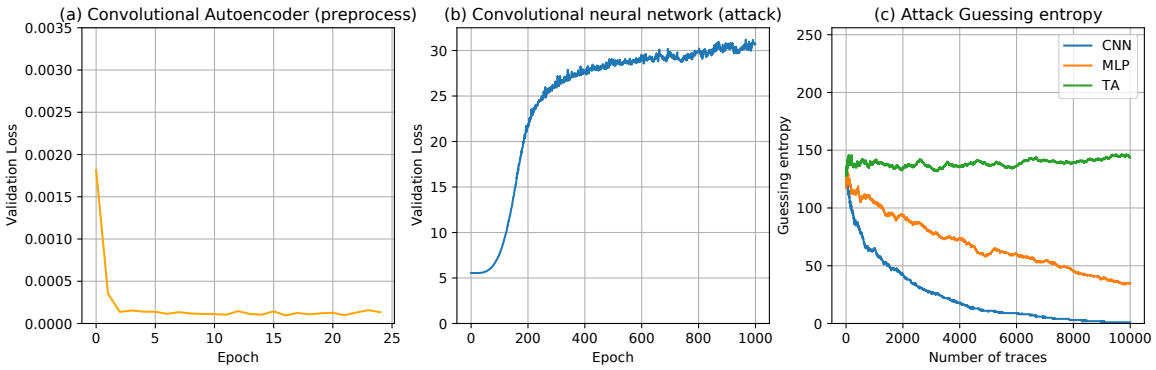

Fig. 3: Results for the countermeasure Gaussian noise (CAE trained 25 epochs)

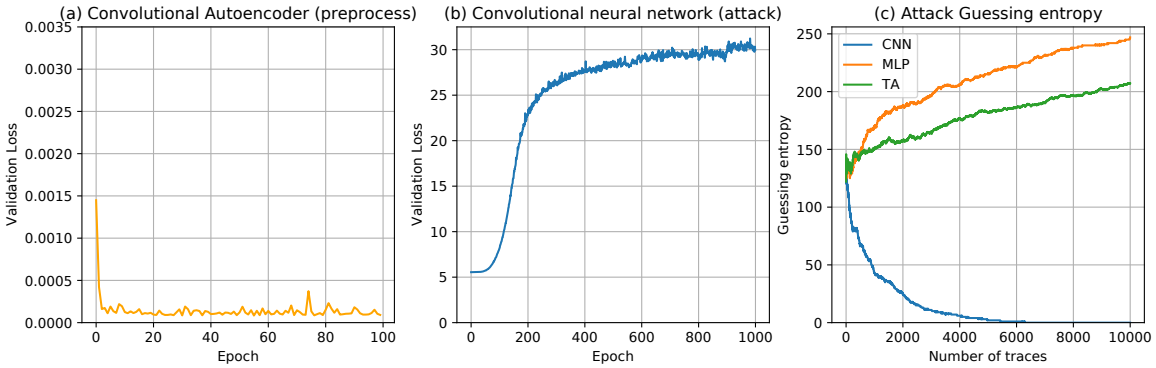

Fig. 4: Results for the countermeasure Gaussian noise (CAE trained 100 epochs)

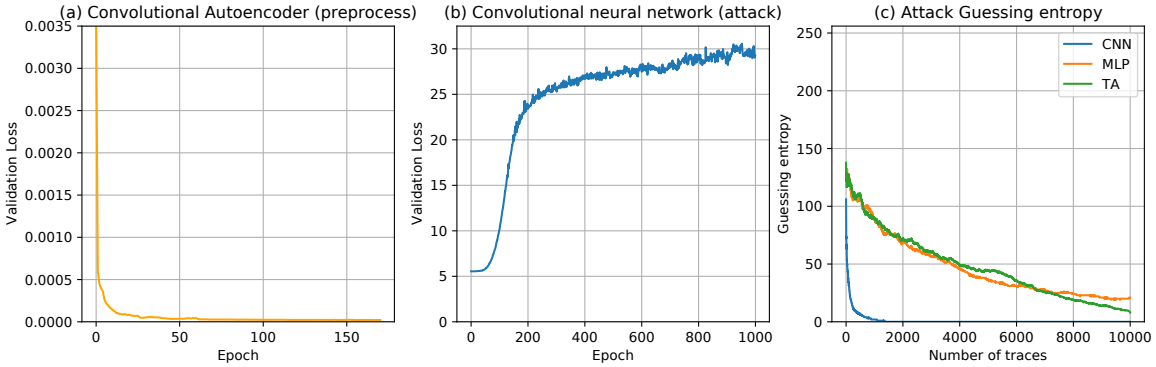

Fig. 5: Results for the countermeasure: Desynchronization

1300 traces. A CNN, MLP or TA on their own do not seem to display any trend towards a GE of 0. This time around the CAE architectures have some common grounds such as the amount of layers and both have the same amount of total max pool when all pool layers are multiplied. Thus the architecture found here has similar results but was found in an automated process eliminating the need for the researcher to put time in manual searching.

As to be expected when finding a CAE architecture for the desynchronization countermeasure, a small network suffices in giving good results. The number of filters, dilation pool size, dense nodes as well as the amount of layers all seem to be at the lower end of their range. This may be explained due to the values of the trace being moved as a group, avoiding the need to filter individual values or parts of individual values. The only parameter which stands out is kernel size, this is likely because a larger kernel can shift values at greater distances, thus improving reconstruction of heavily desynchronized traces.

Both the Gaussian noise CAE and the RDI CAE have bigger

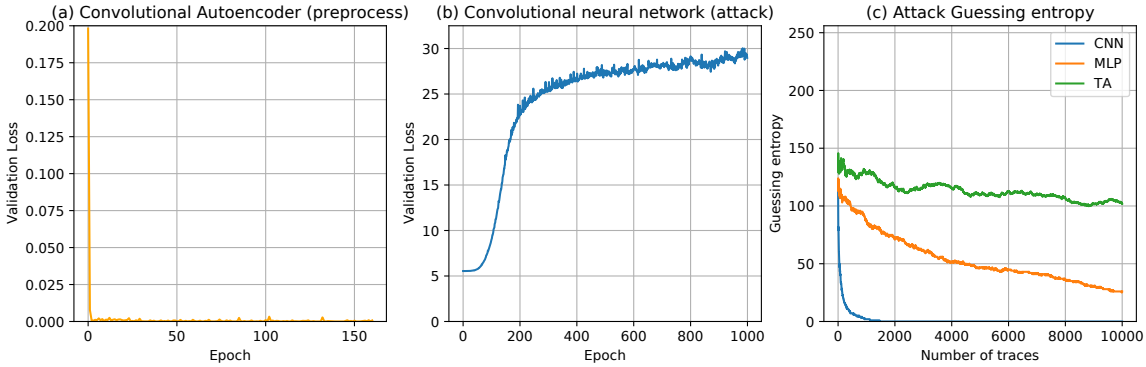

Fig. 6: Results for the countermeasure: Random Delay Interrupts

TABLE IV: Random Delay Interrupts CAE architecture

Layer	No. filters	Dilation	Kernel size	Pool	No. neurons
Conv	26	1	1	1	-
Conv	242	2	14	5	-
Conv	228	2	13	1	-
Conv	107	2	13	1	-
Conv	96	2	16	1	-
Conv	39	3	6	7	-
Flatten	-	-	-	-	-
Dense	-	-	-	-	512
Dense	-	-	-	-	780
Reshape	-	-	-	-	-
Conv	39	2	12	7	-
Conv	96	2	2	1	-
Conv	107	4	16	1	-
Conv	228	2	7	1	-
Conv	242	2	3	5	-
Conv	26	3	13	1	-

architectures expressing the need for more processing that comes with these countermeasures. Both are at the top of the range with regards to layers and may thus benefit from even more layers. In addition to this in both CAEs the kernel size and dilation touch the upper limits in some layers which may indicate it could benefit of bigger kernels and larger dilation. Again similarity is seen in the pool layers where both networks stay far away from the upper boundaries. The main difference seems to be in the number of filters where the Gaussian noise CAE stays clear of the upper bounds, the RDI CAE comes close and thus may benefit of even more filters per layer.

V. Conclusion

Based on our results we conclude that the best optimizer for CAEs in this setting is Adam. As for the activation function, both Relu and Selu work well depending on the countermeasure. Finally, epochs seem to be a different matter as performance improved with increased epochs in the Gaussian noise setting. The validation loss remaining nearly constant in all experiments suggests that as the CAE training continues, the reconstructed traces do not become more similar to the raw traces. However, attack improvements indicate that something positive is happening. Longer training possibly causes the most important features to become more prominent in the reconstructed traces due to the compression in the latent space.

Our results further show the hyperparameter space needed for CAEs in Side-Channel Attacks differ per countermeasure. Where one layer suffices for desynchronization, RDI and Gaussian noise benefit from six or possibly more layers. These differences are seen in nearly all hyperparameters except for the kernel size which is on the upper side of the boundary set at 16 on all three countermeasures. The results of this work may be used as guidance for researchers who want to perform a hyperparameter search of CAEs for Side-Channel Attacks, and could be partially generalized to any model which utilizes a convolutional layer to process the traces.

VI. Future work

In future research, more countermeasures will be tested as well as variable key encryption to further generalize the findings of this paper. Another interesting research direction to be explored is the use of different pruning and sampling strategies. Furthermore, future research should attempt to satisfy the restraint formula in (5) by choosing S_{latent} or $N_{filter0}$ first instead of $\prod_{i=1}^{n} S_{pool,i}$.

In this work validation loss is the metric used to evaluate how successful the CAE hyperparameter settings were. Another interesting research direction is to explore the use of other metrics, such as guessing entropy. To make this possible though, either innovative solutions would have to be found to decrease search and attack time such as pre-training the attack models or more computational power and time are required compared to what is currently done by the research community.

Finally future research should address the validation loss problem to further verify the findings of this work. Research on why networks can still improve after more epochs even though the validation loss does not improve or even deteriorates may help find better metrics, design better loss functions and so help build better neural networks.

Acknowledgement

This project is partially funded by the EU Horizon 2020 Programme under grant agreement No 957269 (EVEREST).

REFERENCES

[1] S. D. Rihan, A. Khalid, and S. E. F. Osman, "A Performance Comparison of Encryption Algorithms AES and DES," *International Journal of Engineering Research*, vol. 4, no. 12, p. 5, 2015.

[2] P. Kocher, J. Jaffe, and B. Jun, "Differential Power Analysis," in *Advances in Cryptology — CRYPTO' 99*, Lecture Notes in Computer Science, (Berlin, Heidelberg), pp. 388–397, Springer, 1999.

[3] D. Agrawal, B. Archambeault, J. R. Rao, and P. Rohatgi, "The EM Side—Channel(s)," in *Cryptographic Hardware and Embedded Systems - CHES 2002*, Lecture Notes in Computer Science, (Berlin, Heidelberg), pp. 29–45, Springer, 2003.

[4] R. Benadjila, E. Prouff, R. Strullu, E. Cagli, and C. Dumas, "Deep learning for side-channel analysis and introduction to ASCAD database," *Journal of Cryptographic Engineering*, vol. 10, pp. 163–188, Nov. 2019.

[5] A. Sayakkara, N.-A. Le-Khac, and M. Scanlon, "A survey of electromagnetic side-channel attacks and discussion on their case-progressing potential for digital forensics | Elsevier Enhanced Reader," 2019.

[6] P. C. Kocher, "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems," in *Advances in Cryptology — CRYPTO '96*, Lecture Notes in Computer Science, (Berlin, Heidelberg), pp. 104–113, Springer, 1996.

[7] S. Mangard, O. Elisabeth, and T. Popp, *Power Analysis Attacks: Revealing the Secrets of Smart Cards*, vol. 31. Springer Science & Business Media, 2008.

[8] R. Mayer-Sommer, "Smartly Analyzing the Simplicity and the Power of Simple Power Analysis on Smartcards," in *International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 78–92, Springer, Jan. 2002.

[9] L. Wu, G. Perin, and S. Picek, "I Choose You: Automated Hyperparameter Tuning for Deep Learning-based Side-channel Analysis," *IEEE*, p. 23, 2020.

[10] S. Paguada, L. Batina, and I. Armendariz, "Toward practical autoencoder-based side-channel analysis evaluations | Elsevier Enhanced Reader," *Elsevier*, 2021.

[11] D. Das, A. Golder, J. Danial, S. Ghosh, A. Raychowdhury, and S. Sen, "X-DeepSCA: Cross-Device Deep Learning Side Channel Attack," in *Proceedings of the 56th Annual Design Automation Conference 2019*, (Las Vegas NV USA), pp. 1–6, ACM, June 2019.

[12] L. Wu and S. Picek, "Remove Some Noise: On Pre-processing of Side-channel Measurements with Autoencoders," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 389–415, Aug. 2020.

[13] D. Kwon, H. Kim, and S. Hong, "Improving Non-Profiled SCA using Autoencoder-based Preprocessing," *Cryptology ePrint Archive*, p. 26, 2020.

[14] K. O'Shea and R. Nash, "An Introduction to Convolutional Neural Networks," *arXiv preprint arXiv:1511.08458*, Nov. 2015.

[15] S. Albawi, T. A. Mohammed, and S. Al-Zawi, "Understanding of a convolutional neural network," in *2017 International Conference on Engineering and Technology (ICET)*, pp. 1–6, Aug. 2017.

[16] V. Dumoulin and F. Visin, "A guide to convolution arithmetic for deep learning," Mar. 2016. arXiv:1603.07285 [cs, stat].

[17] D. Rumelhart, G. Hinton, and R. Williams, "Learning internal representations by error propagation," *California Univ San Diego La Jolla Inst for Cognitive Science*, 1985.

[18] P. Baldi, "Autoencoders, Unsupervised Learning, and Deep Architectures," in *Proceedings of ICML workshop on unsupervised and transfer learning*, pp. 37–49, JMLR Workshop and Conference Proceedings, 2012.

[19] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A Next-generation Hyperparameter Optimization Framework.," *Optuna*, 2019.

[20] J. S. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for Hyper-Parameter Optimization," *Advances in neural information processing systems*, 2011.

[21] M. Randolph and W. Diehl, "Power Side-Channel Attack Analysis: A Review of 20 Years of Study for the Layman," *Multidisciplinary Digital Publishing Institute*, 2020.

[22] G. Zaid, L. Bossuet, A. Habrard, and A. Venelli, "Methodology for Efficient CNN Architectures in Profiling Attacks," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 1–36, Nov. 2019.

[23] S. Picek, A. Heuser, and S. Guilley, "Template attack versus Bayes classifier," *Journal of Cryptographic Engineering*, vol. 7, pp. 343–351, Nov. 2017.

[24] F.-X. Standaert, F. Koeune, and W. Schindler, "How to Compare Profiled Side-Channel Attacks?," in *Applied Cryptography and Network Security*, Lecture Notes in Computer Science, (Berlin, Heidelberg), pp. 485–498, Springer, 2009.

[25] S. Chari, J. R. Rao, and R. Pankaj, "Template attacks," in *International Workshop on Cryptographic Hardware and Embedded Systems.*, pp. 13–28, Springer, 2002.

[26] T.-H. Le, J. Clediere, C. Serviere, and J.-L. Lacoume, "Noise Reduction in Side Channel Attack Using Fourth-Order Cumulant," *IEEE Transactions on Information Forensics and Security*, vol. 2, pp. 710–720, Dec. 2007.

[27] J.-S. Coron and I. Kizhvatov, "An Efficient Method for Random Delay Generation in Embedded Software," in *Cryptographic Hardware and Embedded Systems - CHES 2009* (C. Clavier and K. Gaj, eds.), Lecture Notes in Computer Science, (Berlin, Heidelberg), pp. 156–170, Springer, 2009.

[28] B. Hettwer, S. Gehrer, and T. Güneysu, "Deep Neural Network Attribution Methods for Leakage Analysis and Symmetric Key Recovery," in *Selected Areas in Cryptography – SAC 2019*, Lecture Notes in Computer Science, (Cham), pp. 645–666, Springer International Publishing, 2020.

[29] H. Bal, R. Bhoedjang, R. Hofman, C. Jacobs, T. Kielmann, J. Maassen, R. Van Nieuwpoort, J. Romein, L. Renambot, T. Rühl, *et al.*, "The distributed asci supercomputer project," *ACM SIGOPS Operating Systems Review*, vol. 34, no. 4, pp. 76–96, 2000.

[30] G. Goos, J. Hartmanis, J. van Leeuwen, D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, A. Kobsa, F. Mattern, J. C. Mitchell, M. Naor, O. Nierstrasz, C. P. Rangan, and B. Steffen, "A unified framework for the analysis of side-channel key recovery attacks," *Springer*, p. 625, 2009.

[31] J. Massey, "Guessing and entropy," in *Proceedings of 1994 IEEE International Symposium on Information Theory*, p. 204, June 1994.