mathematics

MDPI

Article

Shift-Invariance Robustness of Convolutional Neural Networks in Side-Channel Analysis

Marina Krček [1], Lichao Wu [2], Guilherme Perin [3] and Stjepan Picek [4,*]

[1] Independent Researcher, 2624 Delft, The Netherlands; marina.krcek7@gmail.com
[2] System Security Lab, Technical University of Darmstadt, 64289 Darmstadt, Germany; lichao.wu@tu-darmstadt.de
[3] Leiden Institute of Advanced Computer Science (LIACS), Leiden University, Niels Bohrweg 1, 2333 CA Leiden, The Netherlands; g.perin@liacs.leidenuniv.nl
[4] Digital Security Group, Radboud University, Houtlaan 4, 6525 XZ Nijmegen, The Netherlands
* Correspondence: stjepan.picek@ru.nl

Abstract: Convolutional neural networks (CNNs) offer unrivaled performance in profiling side-channel analysis. This claim is corroborated by numerous results where CNNs break targets protected with masking and hiding countermeasures. One hiding countermeasure commonly investigated in related works is desynchronization (misalignment). The conclusions usually state that CNNs can break desynchronization as they are shift-invariant. This paper investigates that claim in more detail and reveals that the situation is more complex. While CNNs have certain shift-invariance, it is insufficient for commonly encountered scenarios in deep learning-based side-channel analysis. We investigate data augmentation to improve the shift-invariance and, in a more powerful version, ensembles of data augmentation. Our results show that the proposed techniques work very well and improve the attack significantly, even for an order of magnitude.

Keywords: side-channel analysis; deep learning; misalignment; countermeasures; shift-invariance

MSC: 94A60

check for updates

Citation: Krček, M.; Wu, L.; Perin, G.; Picek, S. Shift-Invariance Robustness of Convolutional Neural Networks in Side-Channel Analysis. *Mathematics* **2024**, *12*, 3279. https://doi.org/10.3390/math12203279

Academic Editor: Cheng-Chi Lee

Received: 29 September 2024
Revised: 11 October 2024
Accepted: 11 October 2024
Published: 18 October 2024

1. Introduction

Convolutional neural networks are successfully applied in diverse domains, like image [1] and audio [2] processing. Moreover, CNNs have been successfully applied to profiling side-channel analysis (SCA). From the first paper on CNNs in SCA appearing in 2016, there have been more than 250 papers exploring the topic of deep learning-based side-channel analysis (DLSCA). In [3], the authors reported 183 papers up to 2021. Searching papers published in 2022 and 2023 showed more than 100 additional papers. The primary motivation for using CNNs in DLSCA is their ability to (1) work with raw features, making feature engineering non-mandatory, and (2) deliver excellent attack performance even when the target is protected with countermeasures. While implicitly assumed by the SCA community, both points deserve more discussion.

While CNNs dramatically reduce the requirement of feature engineering, most works in the DLSCA domain consider traces with a pre-selected window of samples. More recently, results demonstrate that such selection is unnecessary: CNNs can directly retrieve leakages from raw measurements and reach state-of-the-art performance [4,5]. Excellent attack performance, even in the presence of countermeasures, is commonly assumed but also confirmed by numerous empirical analyses. However, the rationale for how CNNs bypass these countermeasures must be clarified. Specifically, desynchronization is a commonly investigated hiding countermeasure. For "reasonable" desynchronization values, it is assumed that CNNs inherently handle it due to their shift-invariance property. Understanding where the shift-invariance comes from and how to improve it is a challenging problem. Although multiple works try to explain it from a theoretical perspective or improve shift-invariance by modifying CNN architecture or the dataset used for training, the mechanism is not fully understood. Since multilayer perceptron (MLP) architectures are not shift-invariant [6], we focus our attention on CNNs in this work.

While the core motivation for this work (data augmentation (DA) enhancing the shift-invariance of CNNs for DLSCA) may sound intuitive and well-known in the SCA domain, we argue that this is not the case. We conducted a literature survey for DLSCA and examined around 230 related works. We do not claim this to be an exhaustive search, but we are confident it is representative as it contains more works than a recent systematization of knowledge work [3]. Out of those 230 works, 70 use datasets with desynchronization. Surprisingly, only 17 out of those 70 works use data augmentation. Thus, with 24% of papers using data augmentation, we cannot claim it as a well-known or accepted technique for DLSCA. Furthermore, we analyzed the 230 papers to see if they mention the shift-invariance property of CNNs and what the cause of it is. We found nine papers discussing how the convolutional layer is the key to shift-invariance. Four papers claim it is due to the pooling layer. From a more general perspective, 18 papers claim it is due to CNNs without going into specifics. More importantly, no papers in DLSCA directly connect data augmentation and shift-invariance.

To conclude, despite the numerous related works, the shift-invariance of CNNs in DLSCA remains an open question. More precisely, it needs to be clarified if CNNs have sufficient shift-invariance and, if not, how to extend it. Our main contributions are as follows:

1. We show that commonly used CNNs in DLSCA have a limited shift-invariance and should be carefully used when attacking desynchronized datasets. Additionally, our experiments show that the pooling layer has a negligible effect on the shift-invariance of CNNs. Still, this might need a separate, more thorough investigation.
2. We show how data augmentation can successfully improve the shift-invariance of CNNs in DLSCA.
3. We show how to use ensembles of data augmentation settings to provide superior attack performance using CNNs when dealing with highly desynchronized datasets.

We emphasize that understanding what part of CNN gives shift-invariance and how to design a shift-invariant CNN are problems that have been open for several years. This work does not aim to solve those problems by providing a universal solution regardless of the evaluated dataset. Instead, we propose an efficient data augmentation approach to improve the shift-invariance of an existing model. Although one could potentially reduce the effects of desynchronization by using deeper architectures or custom architecture elements, we decided not to follow those directions as they will (1) make the hyperparameter tuning more challenging and (2) increase the chances for overfitting. On the other hand, using data augmentation and ensembles is a simpler yet very successful approach.

2. Deep Learning-Based Side-Channel Analysis

As is commonly performed in profiling attacks, we consider a setup with two phases: profiling and attack. The profiling phase corresponds to training a machine learning-based classifier, and the attack phase tests its classification performance. Let us assume a side-channel dataset $\mathcal{X}$ that consists of N measurements. $\mathcal{X}$ can be considered as a 2D array with N rows and J columns. Each point in the array $\mathcal{X}$ is an element $t_{i,j}$, where i indicates the side-channel trace index and j indicates the index of a sample (feature) inside a side-channel trace $\mathbf{x}_i$. The profiling phase aims to learn a parameterized function $f_{\boldsymbol{\theta}}$, where f represents a mapping $f : \mathcal{X} \rightarrow \mathcal{Y}$ between the input space $\mathcal{X}$ and the label space $\mathcal{Y}$. The function $f_{\boldsymbol{\theta}}$ corresponds to a specific instance of f defined by parameters $\boldsymbol{\theta}$, which are optimized to minimize the empirical risk, i.e., the expected value of the loss function. The objective of the profiling phase is defined in Equation (1), where L represents the loss function, which takes the model prediction $f_\theta(\mathbf{x})$ and the true labels $\mathbf{y}$ as inputs. The term $\mathbb{E}[\cdot]$ denotes the expected value.

$$\boldsymbol{\theta} = \arg\min_{\theta} \mathbb{E}[L(f_\theta(\mathbf{x}), \mathbf{y})]. \tag{1}$$

In the test phase, the goal is to predict labels $\mathcal{Y}$ based on the traces from the attacked device and the trained model $f_{\boldsymbol{\theta}}$. More precisely, the result of predicting with a model $f_{\boldsymbol{\theta}}$ on the test set is a two-dimensional matrix P with dimensions $Q \times C$, where Q is the number of attack traces and C is the number of labels. Each value in the matrix P denotes the probability that for a specific key $\mathbf{k}$ and input $\mathbf{a}$, we obtain the label $\mathbf{y}$. Finally, the cumulative sum $S(\mathbf{k})$ for any key byte candidate $\mathbf{k}$ is a side-channel distinguisher with a common maximum log-likelihood principle: $S(\mathbf{k}) = \sum_{i=1}^{Q} \log(\mathbf{p}_{i,\mathbf{y}})$. The cumulative sums for each possible key value form a key guessing vector ordered per the probability of the key being correct (the first position in the vector is the most likely key, and the last position is the least likely key). The position of the correct key is called the key rank, which denotes the effort the attacker requires to break the target. To reduce the effects of a random choice of test measurements, it is common to assess the average behavior over many randomly selected traces, called guessing entropy (GE) (i.e., average key rank) [7]. We consider DLSCA as already well-known in the SCA community, so we only briefly overview it. For more details, we refer readers to, e.g., [3].

Data Normalization. The application of data normalization is crucial to obtain better deep learning performance. The most common form of data normalization adopted in previous papers is *standardization* (see Section 3 in [8] and Section 4-B in [3] for a discussion about data preprocessing). Profiling, validation, and attack sets must be aligned for consistent data normalization. Otherwise, features in validation and attack sets are distorted when normalized with μ_p and σ_p (standardization). In our case, as we aim to investigate the shift-invariance robustness of trained deep neural networks, the profiling and attack sets might contain different desynchronization levels. Therefore, we use the Horizontal MinMax Scaler for data normalization in the range $[-1, 1]$ that was shown to be better overall in [8]. The scaler is trained on and also fits the transposed trace sets individually.

Data Augmentation. Data augmentation refers to increasing the training set's size by artificially generating additional training data with dynamic changes during a model's training. These changes must preserve the class properties of the training set. The training set represents an approximate distribution, given by a finite set $\mathcal{T}$, from a true and unknown distribution $\mathcal{R}$. By augmenting the training set $\mathcal{T}$, one expects that $\mathcal{T}$ becomes a better representation of $\mathcal{R}$. The main idea is to improve class representation inside a dataset, so it is essential to understand what kind of effect the augmentation needs to develop. Inappropriate data augmentation settings can lead to no or even detrimental effects [9].

Metrics. As the goal of this work is to improve the attack efficiency of SCA, the shift-invariance robustness is quantified with the guessing entropy metric and the corresponding number of attack traces to reach a GE equal to 1 ($N_{\mathsf{ge}=1}$), which means successful key recovery. In all our experiments, we provide GE and corresponding $N_{\mathsf{ge}=1}$ results for a single attacked key byte from a 128-bit AES key.

3. Related Works

Cagli et al. used data augmentation to defeat datasets protected with software-based countermeasures (Random Delay Interrupt) and hardware-based countermeasures (jitter) [10]. The authors broke the targets and suggested using data augmentation to reduce overfitting. Interestingly, they did not mention data augmentation as the technique to help defeat desynchronization. B. Timon proposed a non-profiled deep learning-based SCA approach and used data augmentation to improve the learning phase [11]. Perin et al. used data augmentation to improve the performance of the deep learning-based attack on a protected ECC implementation [12]. The authors stated that data augmentation defeats overfitting and the investigated datasets do not have desynchronization. Lu et al. used data augmentation to expand the profiling set and improve the attack performance [5].

The authors mentioned convolutional layers as relevant for shift-invariance. Perin et al. discussed various feature selection scenarios and broke several datasets [4]. The authors used data augmentation for desynchronized datasets and reported significantly fewer attack traces needed if data augmentation was used in the training process. The authors did not discuss the connection between shift-invariance and data augmentation.

Since the publicly available ASCAD datasets provide 50 and 100 window desynchronization samples [13], most papers either use synchronized datasets or desynchronizations up to 100 samples. Naturally, the desynchronization window should be considered relative to the length of the trace. Cagli et al. used desynchronization levels up to 200 samples [10], which they managed to circumvent by using deep neural network architectures and data augmentation. Lu et al. investigated the performance of CNNs against the ASCAD dataset with desynchronization of up to 200 samples [5]. Zotkin et al. used data augmentation and attacked datasets with desynchronization of up to 150 samples [14]. Aligned with several DLSCA works, the authors attributed a shift-invariance property to CNNs. Hajra et al. considered scenarios where the datasets had a significant level of desynchronization, up to 400 samples, and used transformer networks to handle it [15]. The authors tested scenarios where a neural network is trained on synchronized and attacked desynchronized datasets. The good performance is attributed to the shift-invariance property of the used transformer networks. Hajra et al. proposed using a transformer network with linear time and memory complexity that is also shift-invariant [16].

Finally, we note that ensembles were used in various contexts in DLSCA, improving attack performance. For instance, Perin et al. used ensembles to combine predictions from multiple neural networks [17], while Zaid et al. proposed a loss function called Ensembling Loss [18].

4. CNNs and Invariance

In the context of CNNs, three general types of invariance are commonly discussed: translation (shift), rotation, and scale invariance. Of those three, shift-invariance is discussed in the context of DLSCA as it becomes relevant to defeat various desynchronization countermeasures. An operation $\mathcal{G}$ is called shift-invariant if, for an input x and its shifted version x_s, the output remains unchanged, i.e., $\mathcal{G}(x) = \mathcal{G}(x_s)$. Similarly, shift-equivariance means that shifting the input results in an equivalently shifted output, i.e., $\mathcal{G}(x_s) = (\mathcal{G}(x))_s$. In the context of CNNs, which are designed to extract features from the input, the feature representation is said to be shift-invariant if shifting the input x produces the same feature representation. On the other hand, it is shift-equivariant if shifting the input results in an equally shifted output feature representation. In this case, operation $\mathcal{G}$ can represent the feature extraction operation (e.g., the convolutional layers of the CNN). It is a relatively common (and long-lasting) claim that CNNs are invariant due to the convolution layer, e.g., [19,20]. More recently, it was recognized that convolutional layers could have the property of shift-equivariance and not shift-invariance [6,21]. Consider a convolutional layer operation $\mathcal{F}$, which applies a convolution filter k to an input x: $\mathcal{F}(x) = k * x$, where $*$ denotes the convolution operation. If the input x is shifted $x_s = T_s x$, with T_s being the shift operator, the convolution output for the shifted input becomes $\mathcal{F}(x_s) = k * (T_s x)$. Due to the properties of convolution, shifting the input shifts the output by the same amount: $\mathcal{F}(x_s) = T_s(k * x) = T_s \mathcal{F}(x)$. The convolutional layers can lose the perfect equivariance due to downsampling [22]. In CNNs, downsampling happens when the pooling or convolutional stride is greater than one, as the intermediate representation skips samples in the input. It is a pooling layer that provides approximate shift-invariance to small translations of the input [23]. Shift-invariance means that translating the input by a small amount will not cause the values of most of the pooled outputs to change. Intuitively, pooling achieves local translation invariance [24]. Invariance is a particular case of equivariance when the transformation has no effect [25]. Combining the shift-equivariance property of convolutional layers and stability to deform pooling layers, one could expect CNNs to become shift-invariant [26]. Unfortunately, recent results showed that this is not the case since small input shifts can cause drastic changes in the output [22,27]. The reason is the downsampling nature of modern convolutional and pooling layers. Some works also showed that CNNs could encode absolute spatial location in images, resulting from a lack of shift-invariance [21]. Additionally, one cannot definitively answer what part of CNNs brings the most invariance. For instance, it has been shown that the network depth, not the type of layer, contributes more to the shift-invariance [28]. This is relevant for the DLSCA perspective, as most of the results (including state-of-the-art) commonly use relatively shallow neural network architectures, e.g., [4]. Finally, larger convolution filter sizes allow for more shift-invariance [28]. We note that data augmentation is recognized for improving the accuracy and performance of machine learning models across various domains, such as image recognition and segmentation [29–32], and natural language processing (NLP) [33–35]. Additionally, it aids in learning more robust representations that generalize better [36–38]. This robustness is typically achieved by designing transformations that mimic variations observed in the real world [39], making models more likely invariant to such transformations.

5. The Shift-Invariance Robustness of Deep Learning Models in SCA

In this section, we deploy several experiments to empirically verify the shift-invariance robustness of various deep neural networks in SCA. Moreover, we evaluate how data augmentation improves shift-invariance robustness and how this robustness is directly related to data augmentation settings and the shift distribution in side-channel measurements. We define four main scenarios:

1. Neural networks are trained with a synchronized profiling set.
2. Neural networks are trained with a desynchronized profiling set.
3. Neural networks are trained with a synchronized profiling set and data augmentation.
4. Neural networks are trained with a desynchronized profiling set and data augmentation.

The trained networks are tested on an attack set with different levels of desynchronization to showcase the performance.

Datasets

We consider the ASCAD datasets (https://github.com/ANSSI-FR/ASCAD (accessed on 29 September 2024)), which are protected with first-order Boolean masking. In total, we consider four different datasets:

1. `ASCADf`: This dataset contains side-channel measurements consisting of 700 features representing side-channel leakages from processing the third `S-Box` output byte in the first AES encryption round. We split this dataset into profiling, validation, and attack sets with 45,000, 5000, and 5000 traces, respectively. All sets have the same fixed key.
2. `ASCADr`: This dataset contains side-channel measurements where each trace consists of 1400 features. Similar to `ASCADf`, this measurement interval also represents side-channel leakages from processing the third `S-Box` output byte in the first AES encryption round. We split this dataset into profiling, validation, and attack sets with 200,000, 5000, and 5000 traces, respectively. The profiling set has random keys, while the validation and attack sets have the fixed key.
3. `ASCADf_desync100`: Desynchronized version of `ASCADf`, and each trace is randomly shifted inside the interval $[0, 100]$. Shifts are drawn from a uniform distribution.
4. `ASCADr_desync100`: Desynchronized version of `ASCADr`. Each trace is randomly shifted (also from a uniform distribution) inside the interval $[0, 100]$.

Datasets are always labeled according to the identity leakage model for the third `S-Box` output byte in the first AES encryption round.

Deep Neural Network Architectures

The neural network architectures considered in the experiments of this section are described below. The CNN architectures always contain convolution blocks consisting of a convolution layer followed by a batch normalization layer connected to a pooling layer. Both average and max pooling layers are tested. All hyperparameters were obtained with a random hyperparameter search, with the best-found architectures reported in Table 1. The architectures are denoted as `cnn_pooling_2`, `cnn_pooling_4`, `cnn_pooling_6`, and `cnn_-ascadr_desync100`. We train the models for 100 epochs in all experiments. The batch size of each case will be reported in the corresponding section.

Table 1. Hyperparameter values for `cnn_pooling_2`, `cnn_pooling_4`, `cnn_pooling_6`, and `cnn_-ascadr_desync100`. Hyperparameters like weight initializer, activation function, or number of neurons in fully connected (FC) layers apply across all layers if specified with a single value. The number of filters in convolutional layers varies per layer and is specified in sequence.

Hyperparameters	**cnn_pooling_2**	**cnn_pooling_4**	**cnn_pooling_6**	**cnn_ascadr_desync100**
Conv. blocks		2		5
Filters (ordered)		8, 16		12, 24, 36, 48, 60
Kernel size		30		40
Strides		2		15
Pool size	2	4	6	6
Pool strides	2	4	6	6
FC layers		1		2
FC Nb. neurons		40		50
Weight initializer		glorot_normal		glorot_uniform
Activation function		elu		elu
Learning rate		0.001		0.00025
Optimizer		RMSprop		Adam
Epochs			100	

We analyze `cnn_pooling_2`, `cnn_pooling_4`, and `cnn_pooling_6`, which contain different pooling sizes, to understand if small changes in the pooling size have any effect on shift-invariance. Moreover, all models listed above contain large convolution kernel sizes (e.g., 40). Still, in this work, we skip a more detailed evaluation of the effect of filter kernel size (and its strides) on shift-invariance robustness. We evaluate architectures that have provided the best results (i.e., minimum $N_{\texttt{ge}=1}$) from our random hyperparameter search and also architectures with good performances from some related works on the evaluated datasets. Specifically, we test CNN architectures from [8,40] designed for the `ASCADf` and `ASCADf_desync100` datasets. Hyperparameter values are kept the same as in those papers. We denote the architectures as follows:

- `zaid_ascad_desync_0` for architecture from [40] for `ASCADf`.
- `noConv1_ascad_desync_0` for architecture from [8]. The network is a reduced version of `zaid_ascad_desync_0`, where the convolution and batch normalization layers are removed, and the input layer is the pooling layer.
- `zaid_ascad_desync_100` for architecture from [40] for `ASCADf_desync100`.
- `noConv1_ascad_desync_100` for design from [8]. This architecture is a reduced version of `zaid_ascad_desync_100` in the same manner as `noConv1_ascad_desync_0`.

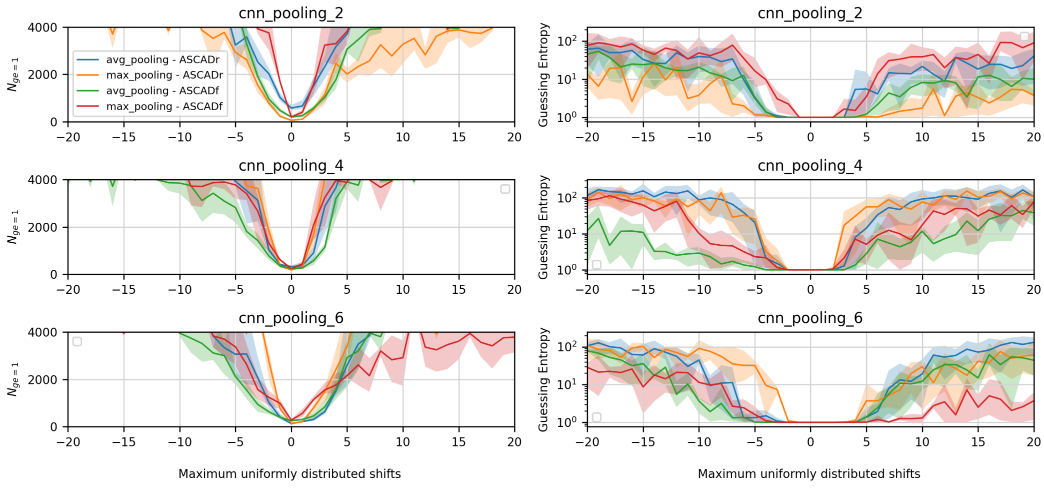
For all architectures, the loss function is always categorical cross-entropy.

Reading the Graphs

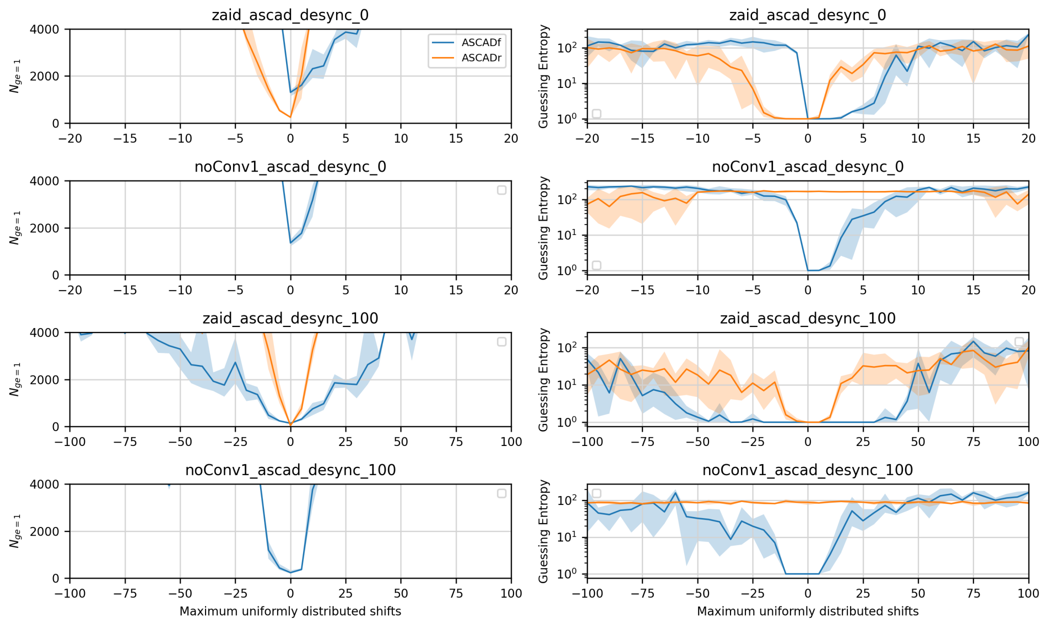
The x-axis in all figures (except those illustrating the shift distributions) indicates the maximum number of shifts δ drawn from a uniform distribution and applied to the set of attack traces. For instance, when the x-axis indicates the value of 100, the set of attack traces is randomly shifted by values inside the interval $[0, 100]$. For negative x-axis values $-\delta$, the interval is $[-\delta, 0]$. To treat the trace samples at the trace boundaries, we first apply random shifts to the attack set and then trim it. The shaded area around lines in the plots indicates the minimum and maximum values obtained from 10-fold experiments, while the main line is the average value. The gray region in the plots indicates the target shift-invariance region.

5.1. Shift-Invariance Robustness of CNNs Trained with Synchronized Datasets

First, we analyze the shift-invariance robustness of `cnn_pooling_2`, `cnn_pooling_4`, and `cnn_pooling_6` architectures with the `ASCADr` and `ASCADf` datasets. Figure 1a shows the shift-invariance robustness of these three models. As we can see, the shift-invariant robustness of these networks, when trained on the synchronized datasets, is very limited in the side-channel context. Note how randomly shifting a few trace samples to the right (positive) or the left (negative) already significantly affects the model performance. With more than five sample shifts, the GE of the attacked key byte substantially increases. The results show no significant difference in the shift-invariance robustness between the tested pooling sizes. Different pooling types (i.e., average or max pooling) show no critical effects on shift-invariance for these three CNN architectures.



(**a**)



(**b**)

Figure 1. Shift-invariance robustness of profiling models trained with synchronized profiling sets. (**a**) Shift-invariance robustness of `cnn_pooling_2`, `cnn_pooling_4`, and `cnn_pooling_6` architectures. Training batch size is set to 400. (**b**) Shift-invariance robustness of `zaid_ascad_desync_0`, `noConv1_ascad_desync_0`, `zaid_ascad_desync_100`, and `noConv1_ascad_desync_100` architectures. Training batch size is set to 50 for `zaid_ascad_desync_0` and `noConv1_ascad_desync_0`, and 400 for `zaid_ascad_desync_100` and `noConv1_ascad_desync_100`.

In the second group, we consider the CNN architectures zaid_ascad_desync0 [40] and noConv1_ascad_desync0 [8] that are designed for the synchronized ASCADf dataset. These architectures and the architectures from our first group were not designed to defeat desynchronized datasets. As shown in Figure 1b, zaid_ascad_desync0 and noConv1_-ascad_desync0 provide a more limited shift-invariance robustness compared to CNNs from Figure 1a. The performance of zaid_ascad_desync0 and noConv1_ascad_desync0 models is significantly inferior compared to original results from [8,40]. The main reason comes from data normalization: in original papers, the authors considered (feature) standardization, while here, we apply horizontal MinMax normalization for the reasons mentioned in Section 2. We also show results for zaid_ascad_desync100 and noConv1_-ascad_desync100 that were designed to break the ASCADf_desync100 dataset. In particular, the zaid_ascad_desync100 model provides much better shift-invariance, as randomly shifting the attack set inside the interval $[-60, 40]$ delivers successful key recovery when the attack set contains 4000 traces. In contrast, the improved version proposed in [8], noConv1_ascad_desync100, provides significantly less shift-invariance robustness. Furthermore, noConv1_ascad_desync100 cannot provide successful key recovery with the ASCADr dataset. This might be because the first convolution layer is absent in this architecture. The second convolution block in zaid_ascad_desync_100 and noConv1_ascad_desync_-100 contains a pooling layer with a size and stride of 50, which is significantly higher than the pooling sizes from other CNN architectures analyzed here. As large (e.g., 50) and small (e.g., 6) pooling sizes provide similar results for synchronized datasets, we conclude that large pooling sizes in one of the hidden layers might not be the main reason for better shift-invariance robustness. Still, more research should be performed to investigate the influence of pooling hyperparameters for the shift-invariance robustness and some other hyperparameters like kernel size, which are also not addressed in this work.

5.2. Shift-Invariance Robustness of CNNs Trained with Desynchronized Datasets

An alternative way to improve the shift-invariance robustness of CNN models is by conducting the training with desynchronized profiling traces. Figure 2 shows results for different CNN models trained with the desynchronized ASCADr and ASCADf datasets. Here, we skip results for cnn_pooling_2, cnn_pooling_4, cnn_pooling_6, zaid_ascad_-desync_0, and noConv1_ascad_desync_0 architectures as, with these models, we were unable to successfully recover the key when trained and predicted with desynchronized datasets. Thus, we consider the cnn_ascadr_desync100 architecture that was found in a random hyperparameter search when ASCADr_desync100 was used as a training set, and its shift-invariance robustness is provided in the top part of Figure 2. Since this model is trained with a dataset containing uniformly distributed random shifts inside the interval $[0, 100]$, its shift-invariance robustness is more salient within the same shift-interval $[0, 100]$. This is expected, as the profiling set contains traces that are shifted inside this interval. The model shows poor generalization capacity outside the interval of $[0, 100]$. For the zaid_-ascad_desync_100 and noConv1_ascad_desync_100 architectures, we found similar shift-invariance robustness results. Both models were designed to break the ASCADf_desync100 dataset with traces containing uniform random shifts inside the interval $[0, 100]$. As shown in Figure 2, both CNN models provide satisfactory shift-invariance inside this shift interval. Outside this interval, the performance quickly deteriorates, indicating that shift-invariance is directly related to the shift distribution in the desynchronized profiling set.

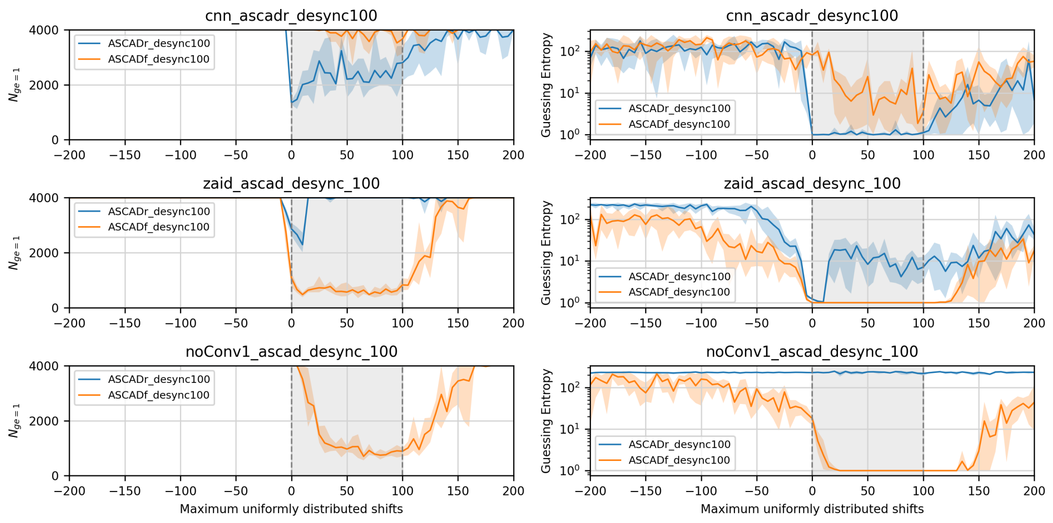


Figure 2. The shift-invariance robustness of different neural network models trained with desynchronized datasets. For the `zaid_ascad_desync_100` and `noConv1_ascad_desync_100`, batch size is set to 200; for `cnn_ascadr_desync100`, the batch size is set to 400.

5.3. DA Effect on Shift-Invariance Robustness of CNNs Trained with Synchronized Datasets

When CNN models are trained with a synchronized trace set, one expects this model to generalize poorly to desynchronized attack sets, as shown by most of the results in Figure 1. One alternative to improve the model's generalization is to train with data augmentation. In this section, we only select `cnn_pooling_2` instead of also analyzing `cnn_pooling_4` and `cnn_pooling_6`, as they showed similar results in the previous section. Also, `cnn_pooling_2` provided slightly less shift-invariance compared to `cnn_pooling_4` and `cnn_pooling_6`, and we want to verify if adding data augmentation improves shift-invariance in the most critical case. The `cnn_pooling_2` architecture is configured with average pooling layers. We skip results for max pooling layers, as our preliminary analysis found no particular benefit of max pooling in shift-invariance robustness, as visible in Figure 1. This model is trained with the `ASCADr` and `ASCADf` datasets. In contrast, `zaid_ascad_desync_0` and `noConv1_ascad_desync_0` models are trained for the `ASCADf` dataset only (results for the `ASCADr` dataset for these two models provided no successful key recovery, so we omit these results). In terms of data augmentation, CNN models are trained with the full profiling set plus double the number of profiling traces as augmented traces (For `ASCADr`, we have 200,000 profiling traces plus 400,000 augmented traces. In comparison, for `ASCADf`, we have 45,000 profiling traces plus 90,000 augmented traces.). We define the number of augmented traces arbitrarily, and the analysis of the optimal number of augmented profiling traces is left for future work.

Figure 3 shows results for three different CNN architectures: `cnn_pooling_2` (with average pooling layer), `zaid_ascad_desync_0`, and `noConv1_ascad_desync_0`. We train these models with synchronized datasets and with data augmentation, which shifts the traces for specific ranges, i.e., $[-100, 0]$, $[0, 100]$, and $[-100, 100]$. The main idea is to verify if all models can become shift-invariant inside specific shift intervals that could be present in a desynchronized attack set. Due to data augmentation, the shift-invariance robustness of all trained CNN models significantly improves compared to the scenario without data augmentation shown in Figure 1. Moreover, the shift-invariance robustness often occurs outside of the augmented shift interval. When data augmentation is implemented for random shifts inside the interval $[0, 100]$ (or $[-100, 0]$), the CNN model tends to become shift-invariant for the interval $[0, 200]$ (or $[-200, 0]$), as is visible for the `cnn_pooling_2` models. The same can be seen in the results on the `ASCADf` dataset with `cnn_pooling_2`. In the case of the models designed for a specific synchronized dataset, data augmentation helped reach good attack results, at least within the data augmentation interval. Thus, training a CNN model with a synchronized dataset and data augmentation may ensure satisfactory shift-invariance robustness. This could prevent the model from presenting poor generalization for cases when the attack set is not perfectly aligned with the profiling set in the time domain.

Another finding in this analysis comes from applying data augmentation for a larger interval, which is the case for the random shifts drawn from the interval $[-100, 100]$. The shift-invariance robustness of the model improves. However, it is not as good for some cases, like with separated intervals $[0, 100]$ and $[-100, 0]$, when looking into GE for these intervals. This is more evident for the `zaid_ascad_desync_0` model: splitting the data augmentation interval from $[-100, 100]$ into two model training for intervals $[0, 100]$ and $[-100, 0]$ provides better convergence of GE inside each smaller shift interval. For `cnn_pooling_2`, the results for the interval $[0, 100]$ are superior if the model is trained with data augmentation providing random shifts inside $[0, 100]$ instead of the entire interval $[-100, 100]$. In Section 6, we propose an ensemble-based strategy to make CNN models highly shift-invariant for larger desynchronization intervals. More precisely, we propose combining multiple data augmentation intervals into a model with better shift-invariant robustness.

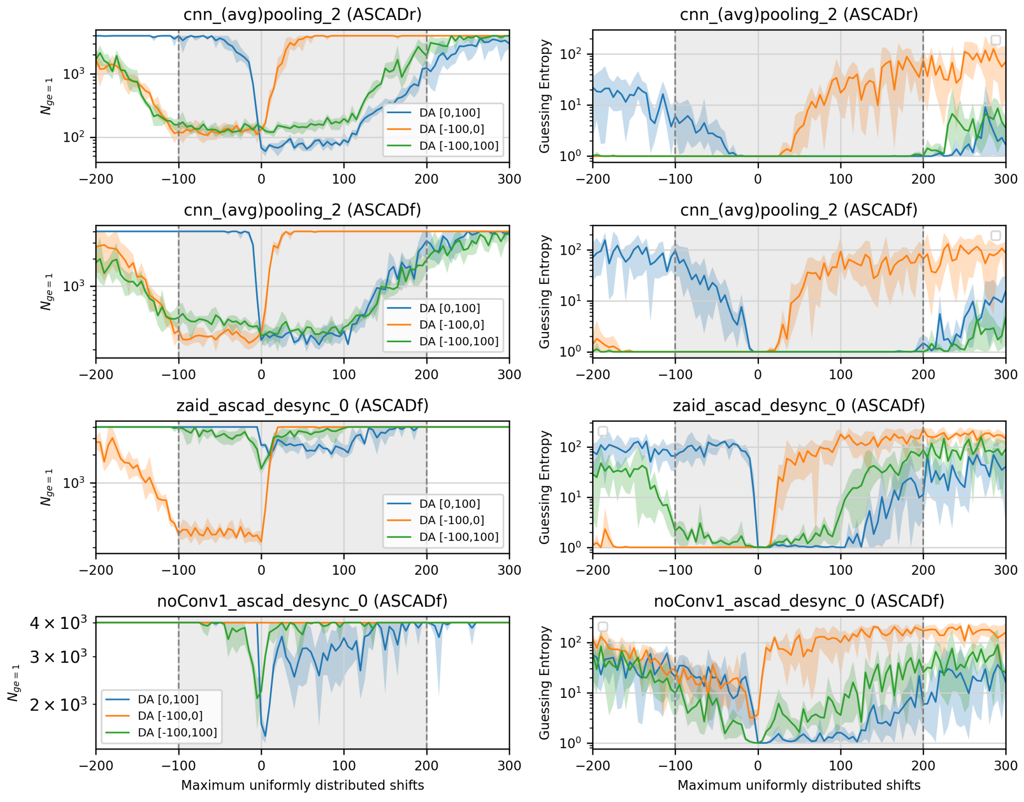


Figure 3. The shift-invariance robustness of different neural network models trained with synchronized datasets. Except for `noConv1_ascad_desync_0`, which has a batch size of 50, all models are trained with a batch size of 400.

5.4. DA Effect on Shift-Invariance Robustness of CNNs Trained with Desynchronized Datasets

A CNN architecture designed to provide an excellent generalization to desynchronized trace sets already shows improved shift-invariance robustness, as shown in Figure 2. However, shift-invariance robustness is still related to the desynchronization in the profiling set. We explore the model's generalization when the attack set contains desynchronization levels that differ from the profiling set. Data augmentation is a solution to improve the shift-invariance robustness of CNN models trained with desynchronized datasets. Our primary goal in this section is to make the CNN models from Section 5.2 shift-invariant inside the trace shift interval $[-100, 200]$ when the profiling set is given by a desynchronized dataset with shifts inside the interval $[0, 100]$. This means that our trained CNN models become shift-invariant outside the shifts contained in the profiling set. Figure 4 shows the shift distribution of original desynchronized profiling traces and the shift distribution of augmented sets. Note that the shifts are applied to already desynchronized profiling traces, so it does not follow a uniform distribution. Still, we ensure the occurrence of shifts from the interval $[-100, 200]$.

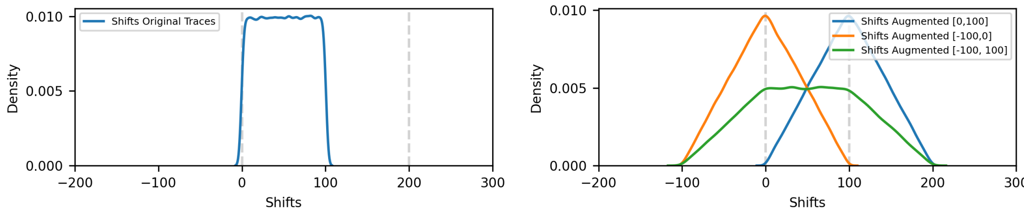


Figure 4. Shift distribution of original (**left**) and augmented profiling sets (**right**).

Results from Figure 5 show the shift-invariance robustness of `cnn_ascadr_desync100`, `zaid_ascad_desync_100`, and `noConv1_ascad_desync_100`. Models were used for predictions on desynchronized attack sets. Although these three models were designed to be shift-invariant to desynchronization of $[0, 100]$, when data augmentation is set to randomly shift the profiling set inside the shift interval $[-100, 100]$, all models become shift-invariant inside the interval $[-100, 200]$. We can also see that `zaid_ascad_desync_100` and `noConv1_ascad_desync_100` become shift-invariant well outside of that interval, achieving low GE but requiring more attack traces than within the $[-100, 200]$ interval.

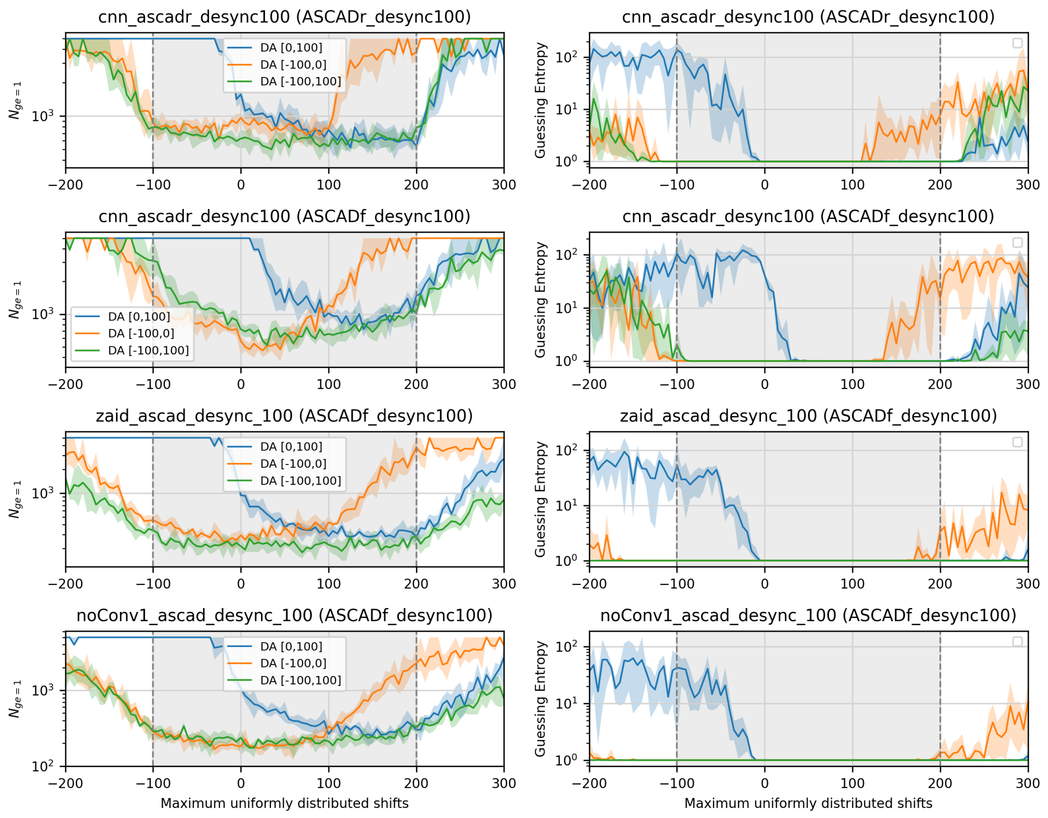


Figure 5. The shift-invariance robustness of different neural network models trained with desynchronized datasets. All models are trained with a batch size of 400.

6. Ensembles to Defeat Larger Desynchronization Levels

In the previous section, we evaluated the shift-invariance robustness when trace sets are desynchronized inside the interval $[0, 100]$. This section shows that a different strategy is required for trace sets containing larger desynchronization levels to improve the shift-invariance robustness of a profiling CNN model. Specifically, we propose a solution to improve the shift-invariance robustness of a CNN model by ensembling multiple CNN models, each trained on a shorter desynchronization interval with data augmentation. We build an ensemble by averaging the output class probabilities from multiple models. The levels of desynchronization that we test are $[0, 200]$, $[0, 400]$, and $[0, 1000]$. To the best of our knowledge, the largest desynchronization level tested in DLSCA-related works is 400 [15].

Datasets

The experiments are conducted with the `ASCADr` and `DPAv4.2` datasets (https://www.dpacontest.org/v4/42_traces.php, accessed on 28 September 2024). Since we want to analyze larger desynchronization levels, we select larger intervals from the raw `ASCADr` and `DPAv4.2` measurements. For `ASCADr`, we select the trace interval consisting of 5000 features, ranging from sample indexes 79,145 to 84,145 (the raw traces have 250,000 features). This trimmed interval includes processing the third `S-Box` byte in the first AES encryption round. This dataset is referred to as `ASCADr_5000`. For the `DPAv4.2` dataset, we select the trace interval from sample indexes 200,000 to 220,000, which contains the processing of the twelfth `S-Box` byte in the first AES encryption round. The resulting 20,000 samples interval is further resampled into 4000 samples. This resampling process reduces the complexity of CNN models. This dataset is referred to as `DPAv4.2_4000`. To work with desynchronized datasets, before trimming each dataset into 5000 and 4000 samples for `ASCADr` and `DPAv4.2`, we apply random shifts from a uniform distribution and then trim the datasets. Both datasets are labeled with the identity leakage model. For `ASCADr_5000`, using data augmentation, the model processes 200,000 profiling traces plus 400,000 augmented traces. For `DPAv4.2_4000`, we process 70,000 profiling traces plus 140,000 augmented traces.

Deep Neural Networks

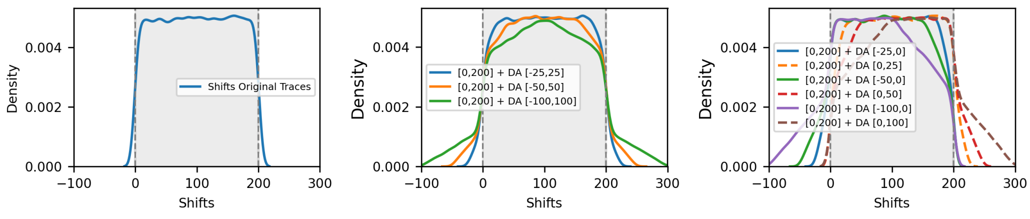
CNN architectures evaluated in the previous section cannot deliver successful key recovery results for the experiments with larger desynchronization intervals. Therefore, we deploy a new random hyperparameter search to find the best possible CNN models for `ASCADr_5000` and `DPAv4.2_4000`. The search considers datasets with desynchronization inside the interval $[0, 100]$. As a result, we found the CNN models reported in Table 2. The models are referred to as `cnn_large_desync_ascadr` and `cnn_large_desync_dpa_v42`. In `cnn_large_desync_dpa_v42`, the fully connected layer is regularized with l1 regularization using $l = 0.00001$.

Table 2. Hyperparameter values for `cnn_large_desync_ascadr` and `cnn_large_desync_dpa_v42`. Hyperparameters like weight initializer, activation function, or number of neurons in fully connected (FC) layers apply across all layers if specified with a single value. The number of filters in convolutional layers varies per layer and is specified in sequence.

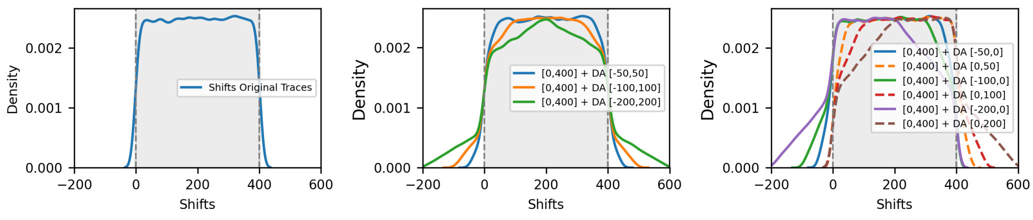
Hyperparameters	**cnn_large_desync_ascadr400**	**cnn_large_desync_dpa_v42**
Conv. blocks	3	4
Filters (ordered)	8, 16, 24	16, 32, 48, 64
Kernel size	40	30
Strides	2	1
Pool size	2	4
Pool strides	2	4
FC layers	1	1
FC Nb. neurons	300	50
Weight initializer	random_normal	random_normal
Activation function	selu	selu
Learning rate	0.00025	0.001
Optimizer	Adam	Adam
Batch size	400	400
Epochs	100	100

Our goal is not to promote the best CNN architectures for large desynchronization cases but to propose a robust ensemble-based data augmentation approach that is expected to be equally efficient with architectures from related works, such as [5,15], which evaluated desynchronization of 200 and 400 samples, respectively.

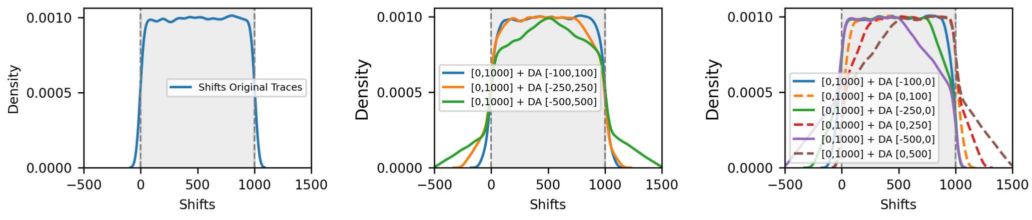
We aim to make described architectures `cnn_large_desync_ascadr` and `cnn_large_-desync_dpa_v42` shift-invariant inside the shift intervals $[0, 200]$, $[0, 400]$, and $[0, 1000]$. For that, we train each model with profiling sets having desynchronized traces with a normal shift distribution inside these intervals, and we add data augmentation to the training process. To keep the model shift-invariant inside, e.g., the interval $[0, 200]$, data augmentation is optimized to preserve the shifts from the original profiling set inside this $[0, 200]$ interval. This means that when we add data augmentation to the profiling set, we shift already desynchronized traces. We carefully choose the minimum and maximum shift ranges for data augmentation. Figure 6a shows (on the left side) the shift distribution of the original profiling set when it is desynchronized inside the interval $[0, 200]$ and also (in the middle) the resulting shift distribution of the augmented traces. Note how our data augmentation process mainly preserves the shift distribution inside the interval $[0, 200]$, with some traces also being shifted outside these boundaries. We perform a similar process when desynchronization is $[0, 400]$ and $[0, 1000]$ (see Figure 6b,c).



(**a**)



(**b**)



(**c**)

Figure 6. Shift distributions in the original and augmented trace sets when the profiling set has desynchronization in $[0, 200]$. (**a**) Profiling set desynchronization: $[0, 200]$. (**b**) Profiling set desynchronization: $[0, 400]$. (**c**) Profiling set desynchronization: $[0, 1000]$.

For each large desynchronization case and each dataset, we train multiple CNN models to combine them into an ensemble. We divide the data augmentation range $[-\delta, \delta]$ into smaller ranges. For instance, when desynchronization in the original profiling set is $[0, 200]$, we divide the data augmentation into the following eight separate ranges: $[-100, -75]$, $[-75, -50]$, $[-50, -25]$, $[-25, 0]$, $[0, 25]$, $[25, 50]$, $[50, 75]$, and $[75, 100]$. It means we have a *data augmentation of step 25* because we implement multiple trainings with data augmentation, shifting the traces with an interval of 25. We train the CNN model for each range with the original desynchronized traces and the data augmentation with the small range. Thus, we train eight separate CNN models and a CNN model with data augmentation with a $[-100, 100]$ shift range. Finally, we build an ensemble from these nine CNN models by averaging their output class probabilities. Similarly, we will test data augmentation with steps of 50 and 100 for the dataset with desynchronization $[0, 200]$; steps 50, 100, and 200 for $[0, 400]$; and steps 100, 250, and 500 for $[0, 1000]$.

Figure 7 shows results for the ensembles obtained for large desynchronization cases in the `ASCADr_5000` dataset. Training a single CNN model with data augmentation inside the shift range $[-\delta, \delta]$ shows inferior results compared to ensembling multiple CNN models with specific data augmentation steps. In the top part of Figure 7, we show results for the desynchronization case of $[0, 200]$. Building an ensemble (with all tested step sizes) allows us to recover the key with approximately 10 attack traces, while using data augmentation with $[-\delta, \delta]$ needs 100 attack traces. This is ten times fewer traces, which indicates better performance. A similar improvement is visible in the case of desynchronization $[0, 400]$. When desynchronization is much larger, in the case of $[0, 1000]$, the only scenario where we can improve shift-invariance is by using an ensemble with a data augmentation step of 100. This means that the shift-invariance robustness of this CNN model is significantly improved with the usage of ensembles.

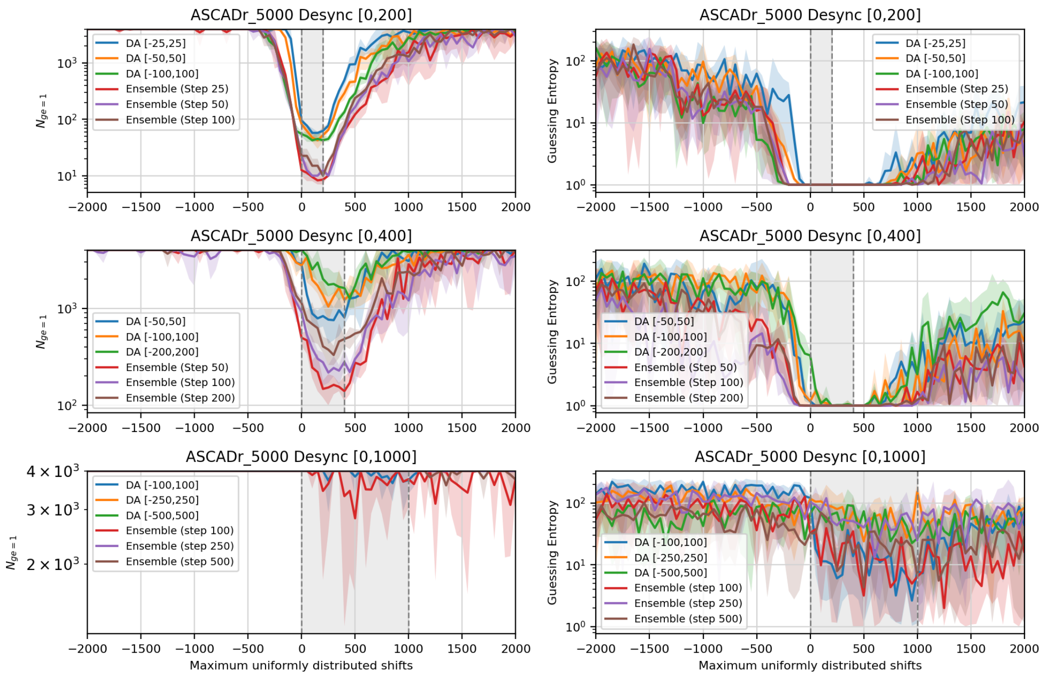


Figure 7. The shift-invariance robustness of the `cnn_large_desync_ascadr` model trained with large desynchronization levels and on the `ASCADr_5000` dataset.

The results for the shift-invariance robustness analysis of the `cnn_large_desync_dpa_v42` model and `DPAv4.2` dataset are shown in Figure 8. When desynchronization in the profiling set is within the range $[0, 200]$, the model becomes shift-invariant for all cases, but ensembles need fewer traces. When the profiling set has desynchronization of $[0, 400]$, we find better results when a single model is trained with data augmentation inside the shift range $[-200, 200]$ and an ensemble of data augmentation with a step of 200. Results for the large desynchronization scenario of $[0, 1000]$ show that we are only able to make the `cnn_large_desync_dpa_v42` model shift-invariant inside the interval of $[0, 1000]$ when we build ensembles of multiple data augmentation steps. The best results were obtained with the data augmentation step of 100. Additionally, the ensembles provide good results outside the targeted shift intervals. Finally, in Figures 7 and 8, if data augmentation is not considered, none of the models can provide successful key recovery results.

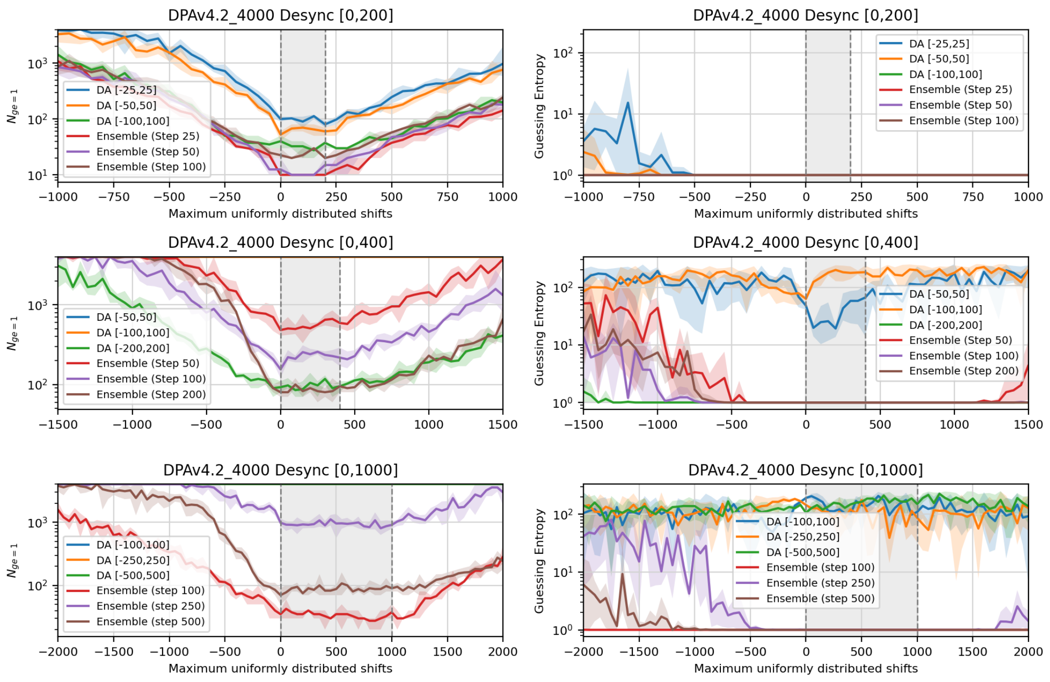


Figure 8. The shift-invariance robustness of the `cnn_large_desync_dpa_v42` model trained with large desynchronization levels and the `DPAv4.2_4000` dataset.

7. Conclusions and Future Work

This paper investigates the shift-invariance of CNNs in DLSCA. First, we show that desynchronized datasets can easily disrupt the shift-invariance of CNNs, leading to unsuccessful attacks. CNNs can learn desynchronization patterns, but this can lead to poor generalization when the attack set is not perfectly aligned with the profiling set in the time domain. Therefore, we empirically demonstrate that the shift-invariance of these CNNs can be enhanced through the use of data augmentation. Data augmentation provides the CNNs with an extended profiling set by introducing potential desynchronization that may occur in the attack set, consequently increasing the model's robustness and improving its generalization to the attack set. Additionally, in many cases, we observe improved shift-invariance beyond the desynchronization levels present within the profiling set. Finally, if the desynchronization levels are large, we propose a novel method based on the ensembles of data augmentation. Our results show superior performance with data augmentation, especially with ensembles of data augmentation. Interestingly, architectural changes (e.g., modifications in the convolutional or pooling layer) provide minor improvements in shift-invariance and represent a more difficult path toward reaching it. This is especially true for more shallow neural network architectures used in DLSCA.

The limitations of this study stem from the reliance on datasets containing the AES measurements, which may not generalize well for other cryptographic algorithms. However, the utilized datasets are widely recognized public benchmarks commonly used in related work to evaluate different methods. Moreover, this work focuses on a single countermeasure, while the methodologies discussed could potentially benefit in the presence of other/multiple countermeasures as well. Thus, for potential future research directions, it would be interesting to evaluate different types of misalignment, like jitter and random delays. Next, here, we arbitrarily set the number of augmented traces. Evaluating the optimal number of augmented traces to be used would be very relevant, especially if some guidelines based on the dataset's properties can be given. Similarly, optimizing model combination techniques for ensembles and the data augmentation steps could be further investigated.

Author Contributions: All authors contributed to all steps in the preparation of this article. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The original contributions presented in the study are included in the article, and further inquiries can be directed to the corresponding authors. The datasets utilized in this study are publicly available, and corresponding links and references can be found in the article.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep Residual Learning for Image Recognition. In Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 27–30 June 2016; pp. 770–778. [CrossRef]
2. Kim, T.; Lee, J.; Nam, J. Comparison and Analysis of SampleCNN Architectures for Audio Classification. *IEEE J. Sel. Top. Signal Process.* **2019**, *13*, 285–297. [CrossRef]
3. Picek, S.; Perin, G.; Mariot, L.; Wu, L.; Batina, L. SoK: Deep Learning-Based Physical Side-Channel Analysis. *ACM Comput. Surv.* **2022**, *55*, 227. [CrossRef]
4. Perin, G.; Wu, L.; Picek, S. Exploring Feature Selection Scenarios for Deep Learning-based Side-channel Analysis. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* **2022**, *2022*, 828–861. [CrossRef]
5. Lu, X.; Zhang, C.; Cao, P.; Gu, D.; Lu, H. Pay Attention to Raw Traces: A Deep Learning Architecture for End-to-End Profiling Attacks. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* **2021**, *2021*, 235–274. [CrossRef]
6. Bronstein, M.M.; Bruna, J.; Cohen, T.; Velickovic, P. Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges. *arXiv* **2021**, arXiv:2104.13478. [CrossRef]
7. Standaert, F.X.; Malkin, T.G.; Yung, M. A Unified Framework for the Analysis of Side-Channel Key Recovery Attacks. In Proceedings of the Advances in Cryptology—EUROCRYPT 2009, Santa Barbara, CA, USA, 16–20 August 2009; Joux, A., Ed.; Springer: Berlin/Heidelberg, Germany, 2009; pp. 443–461.
8. Wouters, L.; Arribas, V.; Gierlichs, B.; Preneel, B. Revisiting a Methodology for Efficient CNN Architectures in Profiling Attacks. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* **2020**, *2020*, 147–168. [CrossRef]
9. Cubuk, E.D.; Zoph, B.; Mane, D.; Vasudevan, V.; Le, Q.V. Autoaugment: Learning augmentation strategies from data. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Long Beach, CA, USA, 15–20 June 2019; pp. 113–123.
10. Cagli, E.; Dumas, C.; Prouff, E. Convolutional Neural Networks with Data Augmentation Against Jitter-Based Countermeasures. In Proceedings of the Cryptographic Hardware and Embedded Systems—CHES 2017, Taipei, Taiwan, 25–28 September 2017; Fischer, W., Homma, N., Eds.; Springer: Cham, Switzerland, 2017; pp. 45–68.
11. Timon, B. Non-Profiled Deep Learning-Based Side-Channel Attacks. Cryptology ePrint Archive, Paper 2018/196. 2018. Available online: https://eprint.iacr.org/2018/196 (accessed on 1 October 2024).
12. Perin, G.; Chmielewski, L.; Batina, L.; Picek, S. Keep it Unsupervised: Horizontal Attacks Meet Deep Learning. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* **2020**, *2021*, 343–372. [CrossRef]
13. Benadjila, R.; Prouff, E.; Strullu, R.; Cagli, E.; Dumas, C. Deep learning for side-channel analysis and introduction to ASCAD database. *J. Cryptogr. Eng.* **2020**, *10*, 163–188. [CrossRef]
14. Zotkin, Y.; Olivier, F.; Bourbao, E. Deep Learning vs Template Attacks in front of fundamental targets: Experimental study. *IACR Cryptol. ePrint Arch.* **2018**
15. Hajra, S.; Saha, S.; Alam, M.; Mukhopadhyay, D. TransNet: Shift Invariant Transformer Network for Side Channel Analysis. In *Progress in Cryptology—AFRICACRYPT 2022: 13th International Conference on Cryptology in Africa, AFRICACRYPT 2022, Fes, Morocco, July 18–20. 2022, Proceedings*; Springer: Berlin/Heidelberg, Germany, 2022; pp. 371–396. [CrossRef]
16. Hajra, S.; Chowdhury, S.; Mukhopadhyay, D. EstraNet: An Efficient Shift-Invariant Transformer Network for Side-Channel Analysis. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* **2024**, *2024*, 336–374. [CrossRef]
17. Perin, G.; Chmielewski, L.; Picek, S. Strength in Numbers: Improving Generalization with Ensembles in Machine Learning-based Profiled Side-channel Analysis. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* **2020**, *2020*, 337–364. [CrossRef]
18. Zaid, G.; Bossuet, L.; Habrard, A.; Venelli, A. Efficiency through Diversity in Ensemble Models applied to Side-Channel Attacks: A Case Study on Public-Key Algorithms. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* **2021**, *2021*, 60–96. [CrossRef]
19. LeCun, Y. Generalization and Network Design Strategies. In *Connectionism in Perspective*; Pfeifer, R., Schreter, Z., Fogelman, F., Steels, L., Eds.; Elsevier: Zurich, Switzerland, 1989; An extended version was published as a technical report of the University of Toronto.

20. Gens, R.; Domingos, P.M. Deep Symmetry Networks. In Proceedings of the Advances in Neural Information Processing Systems, Montreal, QC, Canada, 8–13 December 2014; Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N., Weinberger, K., Eds.; Curran Associates, Inc.: Red Hook, NY, USA, 2014; Volume 27.
21. Kayhan, O.S.; van Gemert, J.C. On Translation Invariance in CNNs: Convolutional Layers Can Exploit Absolute Spatial Location. In Proceedings of the 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Los Alamitos, CA, USA, 13–19 June 2020; pp. 14262–14273. [CrossRef]
22. Azulay, A.; Weiss, Y. Why do deep convolutional networks generalize so poorly to small image transformations? *arXiv* **2018**, arXiv:1801.01450. [CrossRef]
23. Goodfellow, I.J.; Bengio, Y.; Courville, A. *Deep Learning*; MIT Press: Cambridge, MA, USA, 2016. Available online: http://www.deeplearningbook.org (accessed on 1 October 2024).
24. Marcos, D.; Volpi, M.; Tuia, D. Learning rotation invariant convolutional filters for texture classification. *arXiv* **2016**, arXiv:1604.06720. [CrossRef]
25. Lenc, K.; Vedaldi, A. Understanding Image Representations by Measuring Their Equivariance and Equivalence. *Int. J. Comput. Vis.* **2018**, *127*, 456–476. [CrossRef] [PubMed]
26. Chaman, A.; Dokmanic, I. Truly shift-invariant convolutional neural networks. In Proceedings of the 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Los Alamitos, CA, USA, 20–25 June 2021; pp. 3772–3782. [CrossRef]
27. Zhang, R. Making Convolutional Networks Shift-Invariant Again. In Proceedings of the 36th International Conference on Machine Learning, Long Beach, CA, USA, 9–15 June 2019; PMLR, Proceedings of Machine Learning Research; Chaudhuri, K., Salakhutdinov, R., Eds.; Curran Associates, Inc.: Red Hook, NY, USA, 2019; Volume 97, pp. 7324–7334.
28. Kauderer-Abrams, E. Quantifying Translation-Invariance in Convolutional Neural Networks. *arXiv* **2018**, arXiv:1801.01450. [CrossRef]
29. Shorten, C.; Khoshgoftaar, T.M. A survey on image data augmentation for deep learning. *J. Big Data* **2019**, *6*, 60. [CrossRef]
30. Ho, D.; Liang, E.; Chen, X.; Stoica, I.; Abbeel, P. Population based augmentation: Efficient learning of augmentation policy schedules. In Proceedings of the International Conference on Machine Learning, Long Beach, CA, USA, 9–15 June 2019; PMLR; Curran Associates, Inc.: Red Hook, NY, USA, 2019; pp. 2731–2741.
31. Ghiasi, G.; Cui, Y.; Srinivas, A.; Qian, R.; Lin, T.Y.; Cubuk, E.D.; Le, Q.V.; Zoph, B. Simple copy-paste is a strong data augmentation method for instance segmentation. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Virtual, 19–25 June 2021; pp. 2918–2928.
32. Buslaev, A.; Iglovikov, V.I.; Khvedchenya, E.; Parinov, A.; Druzhinin, M.; Kalinin, A.A. Albumentations: Fast and flexible image augmentations. *Information* **2020**, *11*, 125. [CrossRef]
33. Feng, S.Y.; Gangal, V.; Wei, J.; Chandar, S.; Vosoughi, S.; Mitamura, T.; Hovy, E. A survey of data augmentation approaches for NLP. *arXiv* **2021**, arXiv:2105.03075.
34. Park, D.S.; Chan, W.; Zhang, Y.; Chiu, C.C.; Zoph, B.; Cubuk, E.D.; Le, Q.V. Specaugment: A simple data augmentation method for automatic speech recognition. *arXiv* **2019**, arXiv:1904.08779.
35. Meng, L.; Xu, J.; Tan, X.; Wang, J.; Qin, T.; Xu, B. Mixspeech: Data augmentation for low-resource automatic speech recognition. In Proceedings of the ICASSP 2021–2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Toronto, ON, Canada, 6–11 June 2021; pp. 7008–7012.
36. Min, J.; McCoy, R.T.; Das, D.; Pitler, E.; Linzen, T. Syntactic data augmentation increases robustness to inference heuristics. *arXiv* **2020**, arXiv:2004.11999.
37. Sun, S.; Yeh, C.F.; Ostendorf, M.; Hwang, M.Y.; Xie, L. Training augmentation with adversarial examples for robust speech recognition. *arXiv* **2018**, arXiv:1806.02782.
38. Zheng, S.; Song, Y.; Leung, T.; Goodfellow, I. Improving the robustness of deep neural networks via stability training. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016; pp. 4480–4488.
39. Hernandez-Garcia, A. Data augmentation and image understanding. *arXiv preprint* **2020**, arXiv:2012.14185.
40. Zaid, G.; Bossuet, L.; Habrard, A.; Venelli, A. Methodology for Efficient CNN Architectures in Profiling Attacks. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* **2019**, *2020*, 1–36. [CrossRef]