Back to the Basics: Seamless Integration of Side-Channel Pre-processing in Deep Neural Networks

Yoo-Seung Won, Xiaolu Hou, Dirmanto Jap, Jakub Breier, and Shivam Bhasin

***Abstract*—Deep learning approaches have become popular for Side-Channel Analysis (SCA) in the recent years. Especially Convolutional Neural Networks (CNN) due to their natural ability to overcome jitter-based as well as masking countermeasures. Most of the recent works have been focusing on optimising the performance on given dataset, for example finding optimal architecture and using ensemble, and bypass the need for trace pre-processing. However, trace pre-processing is a long studied topic and several proven techniques exist in the literature. There is no straightforward manner to integrate those techniques into deep learning based SCA.**

In this paper, we propose a generic framework which allows seamless integration of multiple, user defined pre-processing techniques into the neural network architecture. The framework is based on Multi-scale Convolutional Neural Networks (MCNN) that were originally proposed for time series analysis. MCNN are composed of multiple branches that can apply independent transformation to input data in each branch to extract the relevant features and allowing a better generalization of the model. In terms of SCA, these transformations can be used for integration of pre-processing techniques, such as phase-only correlation, principal component analysis, alignment methods, *etc*. We present successful results on generic network which generalizes to different publicly available datasets. Our findings show that it is possible to design a network that can be used in a more general way to analyze side-channel leakage traces and perform well across datasets.

***Index Terms*—Multi-scale convolutional neural networks, MCNN, Side-channel attacks, Deep learning**

I. Introduction

Deep neural networks (DNN) have gained popularity in the last decade due to advances in available computational resources. While image classification has benefited the most, the capability of DNN is also demonstrated in other domains like natural language processing, bioinformatics, *etc*. Security evaluation of cryptography against classical and implementation level attacks has also seen rapid adoption of DNN. In particular, side-channel attacks (SCA) have received the most attention as being a classification problem, DNN comes as a natural candidate. Various works in the literature have demonstrated the capability of DNN to break protected implementations, triggering a wave of research in understanding their limits and in turn design of strong countermeasures.

Manuscript received April ??, ????; revised APril ??, 2021.

Y.-S. Won, D.Jap, and S. Bhasin are with Temasek Laboratories, Nanyang Technological University, Singapore 637553 (e-mail: yooseung.won@ntu.edu.sg; djap@ntu.edu.sg; sbhasin@ntu.edu.sg).

X. Hou is with Slovak University of Technology in Bratislava, Slovakia (email: xiaolu.hou@stuba.sk).

J.Breier is with Silicon Austria Labs, TU-Graz SAL DES Lab and Graz University of Technology, Graz, Austria (e-mail: jbreier@jbreier.com).

The authors acknowledge the support from the 'National Integrated Centre of Evaluation' (NICE); a facility of Cyber Security Agency of Singapore (CSA). We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan Xp GPU used for this research. This work has been supported in parts by the "University SAL Labs" initiative of Silicon Austria Labs (SAL) and its Austrian partner universities for applied fundamental research for electronic based systems.

A. Related Works

Maghrebi *et al.* [1] first demonstrated the power of Deep Learning (DL)-based SCA to break protected implementations, specially masking countermeasures. Further, Cagli *et al.* [2] showed the advantages of Convolutional Neural Networks (CNN) against jitter based countermeasures. Authors exploited the input invariance property of CNN to perform SCA evaluation on misaligned traces without the need for trace re-alignment. The work of Zhou *et al.* [3] showed that trace re-alignment can still be helpful for deep learning which is also clear by looking at the results of Cagli *et al.* when comparing ASCAD datasets with different misalignments. These works triggered further research into the usage of DL-based techniques for SCA. The methodology to determine suitable hyper-parameters for CNN and Multi-Layer Perceptron (MLP) was investigated by Prouff *et al.* [4]. An observation was later reported by Picek *et al.* [5], which highlighted that accuracy, a widely used metric in the machine learning field, is not an optimal metric in SCA context, instead they proposed to use guessing entropy. Further, Kim *et al.* [6] proposed a VGG([7])-like network, inspired by the similarity of side-channel measurements to time series data like audio signals. The proposed VGG-like network, along with (externally introduced) regularization due to added Gaussian noise, was shown to produce promising results against multiple datasets. In the context of metrics for side-channel, Masure *et al.* [8] theoretically showed that minimization of negative log-likelihood loss (NLL) corresponds to the estimation of perceived information, a classical side-channel metric. Zaid *et al.* [9] proposed a methodology to design efficient CNN for SCA context. The authors study different side-channel datasets and design an optimal CNN for each case, reporting promising results for each studied dataset. The difference between the approach of Kim *et al.* and Zaid *et al.* is that the latter optimizes CNN architecture to each use case, while the former uses the same CNN architecture to evaluate several datasets. It is not a surprise, Zaid *et al.* present better results. Perin et al. [10] use ensemble models to focus on generalization but

their focus lies in model generalization targeting one dataset at a time. Further, Won *et al.* [11] showed that the results of Zaid *et al.* can be further boosted by applying data oversampling technique. Wouters *et al.* [12] showed the importance of pre-processing for DL-based SCA evaluation to reduce network size. They also highlighted the need for study of networks which are optimal across datasets and indicated the existing literature on time-series classification as a direction. Golder *et al.* [13] showed DL-based SCA for cross-device attacks. They also showed that pre-processed traces with Dynamic Time Warping (DTW) and Principal Component Analysis (PCA)-based pre-processing outperforms standalone MLP and CNN in terms of testing accuracy. As mentioned earlier, accuracy is not an optimal metric for evaluating the model performance with regards to SCA. The results in [13] were obtained by using traces collected from an 8-bit ChipWhisperer platform and the dataset was not made public.

B. Motivation

Scanning through the series of previous works, we notice that the majority of the research has been done towards the direction of designing an efficient network that can provide the best attacks against a set of public trace datasets [6], [9], [12] or on techniques to boost the results of existing networks like augmentation or oversampling [5], [2], [11]. The general focus of these works has been to optimize CNN to defeat the underlying countermeasures, leading to designing a specific network for each dataset, tailored for the properties of this dataset. Independently, the advantage of pre-processing the training set for DL based SCA was shown in [3], [13]. To the best of our knowledge, no work has investigated the possibility of strengthening DNN architecture with the capability of integrating existing side-channel pre-processing or filtering techniques. Moreover, there is no general framework up to date that would help users with the overall trace analysis aided by machine learning, thus minimizing the necessity for architecture adjustments by the user. This forms the key motivation of this work, where we would like to propose a framework to seamlessly integrate previously developed and proven techniques for side-channel pre-processing into deep-learning based evaluation.

C. Multi-Scale Convolutional Neural Networks

Multi-Scale Convolutional Neural Networks (MCNNs) were proposed for time series classification (TSC) in [14]. The idea is to incorporate feature extraction and classification in a single framework by using a multi-branch model. The working principle of MCNN is to extract features at different scales and frequencies by transforming the original data and feeding the result to different branches of the model. One convolutional layer is capable of detecting local patterns, while the combination of multiple convolutional layers can recognize more complex patterns. Later, the branches are concatenated and the computation follows a standard CNN architecture.

Overall architecture of MCNN is depicted in Figure 1. The MCNN framework from [14] has three sequential stages:

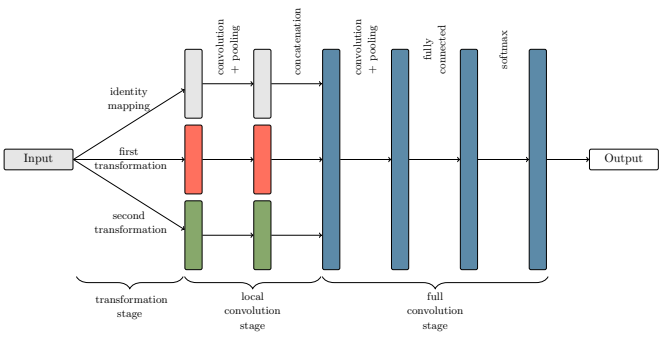


Fig. 1: MCNN architecture proposed in [14] for time series classification.

1) *Transformation stage:* various transformations are applied on the input data. In the TSC domain, the proposed transformations were identity mapping, down-sampling in the time domain, and spectral transformation in the frequency domain. Each part is called a *branch*, and serves as an input to the CNN. Long-term features reflect overall trends and short-term features characterize small changes in local regions, while both of these can be important for the prediction.
2) *Local convolution stage:* several convolutional layers are used in each branch to extract the features. Convolutions for different branches are independent. Max pooling is also performed between the convolutions to prevent overfitting and improve computation efficiency.
3) *Full convolution stage:* extracted features are concatenated and several more convolutional layers are applied, followed by fully connected layers, and a softmax layer to generate the output.

MCNN was applied to 44 time series datasets in [14], achieving better results than a standard CNN on 41 of them. These networks were also successfully used in the past for predicting heart diseases [15] and for speech emotion recognition [16] where they outperformed CNNs. Multi-scale recurrent CNNs were used for financial time series classification [17].

One of the advantages of the MCNN over the classical CNN is the ability to extract features at different time scales. Each branch can be specified to work on a different scale and frequency, and therefore, help to extract features that are relevant on such scale. It can be thought of as looking at the original data from different viewpoints. It is not a simple data triplication, but each branch looks and extracts the information at a specific property, based on what the network designer aims to analyze. As side-channel leakages come from various operations, working at different frequencies, MCNN naturally fit this problem.

D. Contributions

In this work, we propose a generic framework to integrate side-channel oriented pre-processing into deep learning architecture for side-channel evaluations. The framework is based on MCNN. Each branch of MCNN can be configured to

perform a different transformation of the raw data. These transformations can be from time domain or frequency domain. Each convolution layer in an individual branch is expected to learn local patterns or features which when stacked with other layers result in a more complex learning. This makes the network more generic and consistent across datasets.

The main contributions of this work as follows:

- We propose a *generic framework* based on MCNN to enable seamless integration of side-channel oriented pre-processing techniques into deep learning based side-channel evaluations.
- By choosing a CNN architecture proposed for AS-CAD(desync=100) as a building block from [9], we show that the constructed MCNN performs better across a range of side-channel datasets without the need of fine-tuning the parameters, as compared to the original network which performs only for optimized dataset or its trivial variants.
- We integrate well known methods from side-channel literature like Phase-Only Correlation (POC), Principal Component Analysis (PCA), Elastic Alignment (EA), Moving Average (MA) into MCNN to boost its performance.
- We present successful key recovery results for a masked FPGA implementation of AES-128.
- We also demonstrate that pre-processing alone is not always helpful. Indeed, it is the MCNN architecture that learns different features in each branch to result in a strong classifier that is comparable to the performance of the ensemble method [10], while requiring lesser performance overhead.

The code is made publicly available for reproducibility of results[1].

E. Organization

The rest of the paper is organised as follows. Section 2 recalls general background concepts used in the rest of the paper. Section 3 describes the adaption of MCNN for side-channel evaluation. Section 4 compares the performance of MCNN across public side-channel datasets against the state of art network. Section 5 demonstrates the capability of MCNN to seamlessly integrate well established side-channel pre-processing methods in the evaluation process. Finally, conclusions are drawn in Section 6.

II. Background

This section highlights general background concepts used in the following sections.

A. Time Series

A time series is a real-valued, high-dimensional vector that contains observations that are naturally ordered *w.r.t.* time. A time series often comes from recording time-varying measurements of an underlying process, *e.g.* stock market valuations, electronic health measurements, acoustic signals, *etc*. A *univariate* time series consists of sequentially collected observations of a single time-varying measurement and a *multivariate* time series consists of collected observations of two or more time-varying measurements. Given a collection of side-channel measurements, if the attacker focuses on exploiting one particular time sample (*e.g.* one sample during one XOR operation), the side-channel traces are considered as univariate times series. On the other hand, if the attacker exploits multiple points of interest in each trace, corresponding to one or more operations, we can view a single trace as a multivariate time series.

In time-series analysis, the main objective is to apply algorithms to analyze and extract previously unknown information in time series. In terms of SCA, this usually means recovery of information related to secret key used for encryption. As stated in [12], analyzing network architectures built for time series classification and adopting them for SCA might be beneficial for the community.

B. Profiled Side-Channel Analysis

Considering a strong adversary with access to a clone device, profiled SCA [18] operates in two phases. In the profiling or training phase, the adversary acquires side-channel measurements for known plaintext/ciphertext and known key pairs. This training set is used to characterize or model the device. The adversary then acquires few measurements from the target device, usually identical to the clone device, with known plaintext/ciphertext but the key is secret. These measurements from the target device are then tested against the characterized model from the clone device. For a well-trained model, predicted labels corresponding to measurements from the target device reveal information on the secret key. First known profiled SCA used Gaussian templates [18], commonly referred to as Template Attacks (TA). Later, machine learning [19] and eventually deep learning [1] were shown to be better in practice when the traces are limited in number with intentional disturbance from countermeasures and measurement noise.

C. Performance Metrics

To evaluate the performance of an applied SCA, one must choose a suitable metric. While accuracy is a common metric to evaluate the performance of neural networks, it was shown that it is not optimal for side-channel based key recovery attacks [5]. As a result, we use guessing entropy (GE), a metric commonly used for side-channel evaluations [20], even in deep-learning context. GE can be described as an average rank of the correct key after the attack, where GE = 0 indicates that the correct key is ranked the top, which means the correct key has been recovered by the attacks. Note that, if the deep learning model is unable to learn the dataset during the training phase, the GE results while testing are expected to show a random behavior and in general do not provide much information on the attack ability using that trained model [21].

[1]The code is available at https://github.com/mitMathe/SCA-MCNN

D. Pre-processing Techniques for SCA

While it is common practice in machine learning to pre-process features before training/testing, these pre-processing methods are often limited to range normalization or similar adjustment of the input data. The focus of this work is to integrate side-channel oriented pre-processing into deep neural networks, while default pre-processing like normalization is still in place. We recall four pre-processing techniques for SCA that are utilized in this paper.

a) Moving Average (MA): In the context of SCA, moving average technique is usually combined with the fundamental SCA techniques to resist the jitter-based countermeasure. For example, correlation [22], [23], T-test [24] is combined with moving average for boosting the performance. The original proposal of MCNN uses moving average as one of the transformations to act as a low-frequency filter, reducing the variance of time series [14].

b) Principal Component Analysis (PCA): TA exploits multivariate leakages by exploiting information in the covariance matrix [25]. However, with the increase in number of samples in the trace, the size of co-variance matrix grows quickly beyond computation limits. Thus, PCA [26] was proposed as a technique for dimensionality reduction. PCA finds linear transformation that projects high-dimensional data to a lower dimensional subspace while preserving the data variance. Several variants of TA had used PCA as a pre-processing tool [25], [27].

c) Phase-Only Correlation (POC): POC is used for high-accuracy image matching problems [28]. This technique was adopted as an alignment scheme by [23], [29] in the context of side-channel analysis. POC is based on phase components in the discrete Fourier transform and provides the shift value to properly match with the reference trace for alignment. In the case of EM signal with sharply shaped samples in numerical data, the alignment technique based on correlation might be useless and requires many trial-and-error methods for searching proper parameters. Since the shift value for alignment is based on Peak-to-Sidelobe Ratio in POC, there is no need for parameter adjustments.

d) Elastic Alignment (EA): As desynchronization such as random jitters and random process interrupts are frequently employed to reduce the signal-to-noise ratio in the context of SCA, it is sometimes hard to align using the alignment technique based on correlation. One of the solutions overcoming this obstacle is EA [30], which is based on dynamic time warping algorithm adopted from speech recognition [31]. As a result, the elastic alignment naturally concentrates on resynchronizing the traces. However, it might cause a loss of data leakage since it focuses the synchronization on traces shape and generates artificial samples.

III. Tailoring MCNN for SCA

In SCA domain, the time series data normally comes from leakage measurements like power consumption or electromagnetic (EM) emanation during the execution of the cryptographic algorithm. Different types of SCA countermeasures are usually utilized to prevent the attacker from extracting the secret information from these measurements. It was shown that hiding countermeasures based on random delay insertion (RDI) can be defeated by data pre-processing techniques [32]. These techniques aim at minimizing misalignment either by re-aligning the original traces according to a reference trace or by selecting points of interest that contribute to the information leakage. We notice that the multi-branch structure of MCNN, where each branch undergoes an independent pre-processing, provides us with an opportunity to seamlessly integrate the side-channel pre-processing capability in our deep-learning architecture. The rest of the section describes our approach and rationalises our architecture choices.

A. Main Characteristics of the Framework

In this part, we discuss the basic characteristics of the framework based on MCNN architecture. As shown in Figure 1, MCNN is composed of different branches. Each branch consists of convolutional and pooling layers applied to different transformations of the input data. The branches are then concatenated, followed by a full convolutional stage. In the rest of the paper, we call these branches *plug-in branch components (PBC)*. Concatenation is also an important part of MCNN followed by full convolutional layers, which enables the network to co-learn features from individual branches, together with the following layers. Combined, they strengthen MCNN to learn a more complex model compared to a classical CNN. In the following, we detail the PBC feature, model requirements, and data pre-processing.

1) Plug-in Branch Components (PBC): As we focus on modular design for our MCNN SCA framework, we introduce the concept of *Plug-in Branch Components (PBC)*. The design of the original MCNN utilizes 3 PBCs, one of them taking the original data as an input, and the other two transformed data. In the context of SCA, we propose using transformations that were shown to be helpful when analyzing leakage traces, such as PCA, or moving average. Alignment techniques, such as elastic alignment, and signal processing and noise filtering techniques like Fast Fourier Transform can also be used as PBCs. As neural networks naturally select important features during the training phase, it is expected that the PBC providing more relevant features will be prioritized. This unburdens the user from trying out various pre-processing techniques to get the best result. The choice of PBC can also profit from the attacker's expertise who can carefully choose PBC based on the underlying countermeasure. Linking these PBC based transformations to data pre-processing is what enables natural and seamless integration of widely used techniques to DL-based SCA.

a) Data Pre-processing: Pre-processing is a general practice in SCA. Most evaluation labs dealing with real products with countermeasures spend the majority of their effort in data pre-processing. If pre-processing is done correctly, the following process of key recovery is straightforward. Adopting MCNN structure for SCA allows us to feed the pre-processed traces into the neural network. Note that this is a salient feature of MCNN whereas for other used architectures, such pre-processing is applied on training data. Being a time series

data, side-channel traces contain both short term and long term features, while also exhibiting different frequency features. Processing only the transformed data can also lead to loss of information. With pre-processing, we can create training data with more distinct features when exploited together with the original data in another branch, thus increasing the learning power of the trained network. We have discussed several existing pre-processing techniques for SCA in Section II-D. Moving averages are a simple and common type of smoothing used in time series analysis and time series forecasting. For SCA, it helps to remove noise and better expose the signal of the underlying operations within each averaged interval of the trace. Hence, moving average assists in capturing short term features and merging leakages spread over several neighbouring samples together. PCA projects high-dimensional data to a low-dimensional subspace and preserves the most important directions. It helps in identifying important features in different frequency domains. Similarly, POC analyzes the discrete Fourier transforms of waveforms, hence extracting features in various frequency domains. Elastic alignment aims to align the entire trace to counter jitter and random interrupts. The pre-processed traces will then contain enhanced long term features.

b) Model Requirements: We can summarize our requirements on the neural network model based on MCNN in following points:

- Perform well on multiple datasets without the need of hyperparameter tuning.
- Overcome side-channel countermeasures, specially commonly studied jitter-based and masking countermeasures for hardware and software implementations.
- Easy to replace a PBC with a different one, in case a better pre-processing method is available in the future.

Additionally, as we aim to have a general framework that works across various datasets, we integrate different pre-processing techniques in a single package that are designed for extracting important side-channel information preserving distinct features.

B. MCNN Architecture for SCA

While the original MCNN proposal [3] uses CNN as a building block, one can choose to build MCNN like architecture with advanced deep learning techniques like recurrent CNN (RCNN) and long short-term memory (LSTM). We however continue to use CNN for our MCNN architecture for two distinct reasons. Firstly, LSTM and RNN are not currently well studied for SCA use case. While some work do report results with LSTM [33], in current form, the use of such architectures is not shown to be specially advantageous over CNN. Secondly, the wide variety of results available for use of CNN with SCA datasets, help us to benchmark our results. To this end, we choose state of the art CNN architecture known in the SCA literature. In particular we use the the architecture from Zaid *et al.* [9].

Zaid *et al.* [9] have proposed different CNN architectures for different datasets. For example, they fine-tuned the filter size based on the jitter amount. However, in practice, one would not expect the attacker to have this kind of knowledge. Since we aim to propose a generic architecture that can be utilized for any dataset, we choose one particular network structure from [34] for comparison. We have chosen the CNN proposed for ASCAD(desync=100) as it has the most complicated structure and we expect it to perform reasonably well on other datasets as well. We will denote this CNN as Base Network (BN) throughout the rest of the paper. We note that as BN is not optimized for other datasets, we expect sub-optimal performances of BN on other datasets.

In line with the MCNN structure presented in Section I-C and the original paper [14], we consider three plug-in branch components in the transformation stage, with one PBC being the identity. Therefore, the other two PBCs have to be chosen carefully to provide extraction of relevant features. With three branches, the parameter complexity of the network stays within reasonable numbers while the advantage from different feature sets is clearly visible in the result. Naturally, there are a lot of candidates for PBC since many pre-processing techniques have been suggested in the context of side-channel analysis. As a representative example, we consider the moving average and PCA. The moving average techniques are widely combined with the fundamental side-channel analysis to boost the performance against jitter countermeasures. For example, the correlation based on sliding window is employed because the points of interest are normally spread over several points. Hence, moving average is used as one of the PBCs in the transformation stage of our MCNN. For the second PBC, we can choose one of the representative pre-processing techniques performing dimensionality reduction such as PCA. PCA has been used for years in the context of profiled SCA. Especially, this technique is used for noise reduction and overcoming of misalignment, and it has been recently applied to increase the performance of DL [13].

While for our basic MCNN structure, we use identity, moving average, and PCA as PBCs for the transformation stage, we propose MCNN as a generic framework where the PBCs can be user-defined and seamlessly integrated into the network structure. Therefore we show a few examples for variations of the basic MCNN structure in Section V.

C. MCNN Structure

Structure of the MCNN used in this paper is depicted in Figure 2. It closely follows the original proposal from [14], using two convolutional layers in each branch, one convolutional layer after the merging of the branches, and dense layers at the end. As stated in [9], three convolutional layers in a sequence can provide optimal feature extraction for SCA tasks. While the increasing number of convolutional layers can increase the performance according to [12], after three layers, this increase is marginal in terms of traces required for the attack.

To make sure the model architecture is well-fit for SCA, we experimented with a few different candidate models. We used the AES_HD dataset to benchmark the models as it is more challenging compared to software datasets due to low SNR. Therefore, the performance difference can be clearly recognizable between good and bad options. The results of

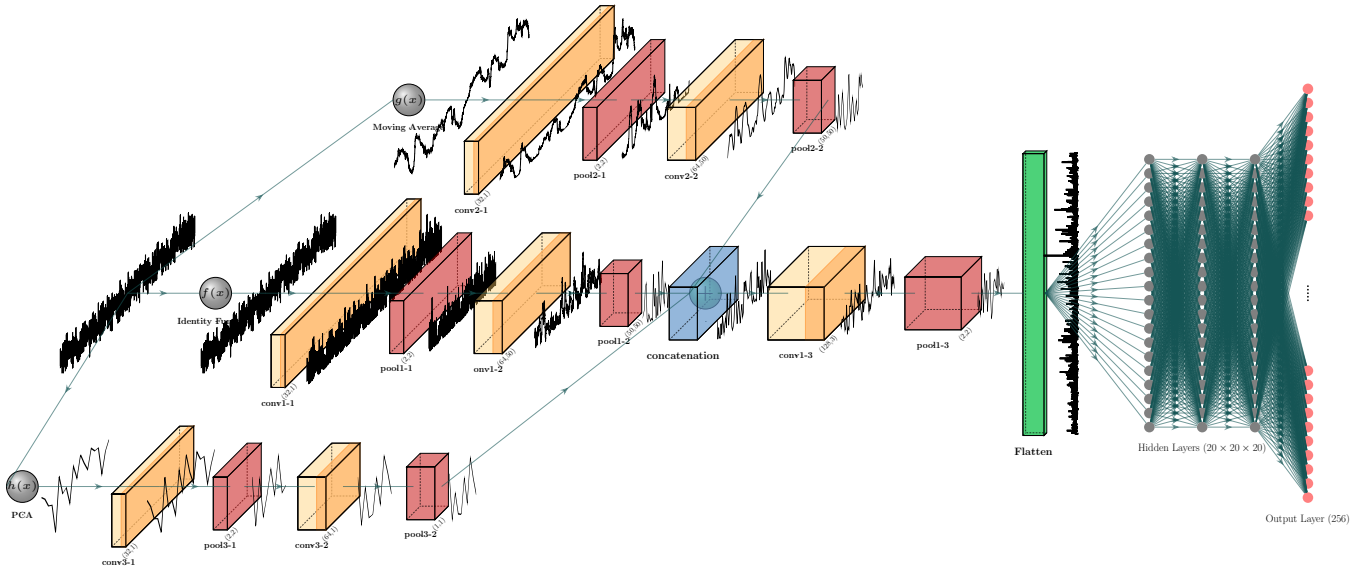


Fig. 2: Main Structure for MCNN.

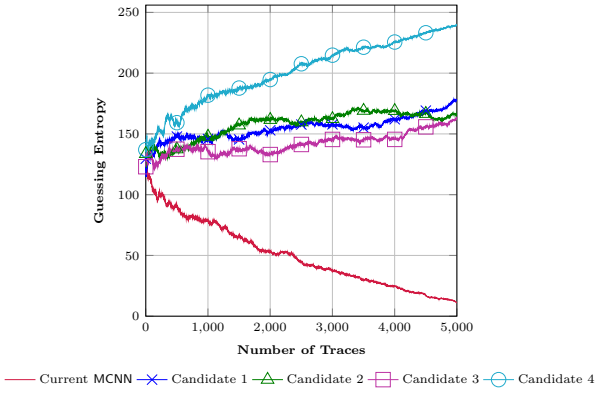


Fig. 3: Results for AES_HD for various MCNN architecture candidates.

these experiments are stated in Figure 3. Differences of the candidate models over the main model are as follows:

- **Candidate 1:** One additional convolutional layer was added to each branch, making the total number of branch layers three.
- **Candidate 2:** Same as Candidate 1, but with removing the convolutional layer after the branch merging.
- **Candidate 3:** Same as Candidate 2, but with adding an extra batch normalization step after the branch merging.
- **Candidate 4:** One additional convolutional layer was added after the branch merging, making the total number of convolutional layers in the full convolutional stage two.

As can be seen from the figure, the chosen MCNN architecture performs the best among the five models. The number of convolutional blocks for this architecture is the same as in [9], which is in line with their findings. While Candidate 2 also has the same number of convolutional blocks, the performance is degrading because of properties of MCNN architecture – a convolutional layer after the branch merging is beneficial to further extract the features. A different number of branches was explored in [17] (see Table 7) for analyzing financial time-series data, where authors tried 1-3 branches. From their results, 3 branches provide the best accuracy.

For local convolution and full convolution stages, we follow the properties of the BN architecture. Since there are three convolution and pooling layers in BN, it can be split into two convolution and pooling layers for the local convolution stage and the last one convolution and pooling layers for the full convolution stage in MCNN. Moreover, the convolution filters, convolution kernel size, pooling size, and pooling strides are also adopted from BN. For PCA, there are some modifications as the number of points is different compared to other branches. In the second convolution block of the local stage, the convolution kernel size and pooling size are set to 1. Similar to BN, batch normalization is applied to the next pooling layer. The architecture requires the batch normalization before concatenation as data of different dimensions are merged before the full convolution stage. We summarize our MCNN architectures and compare them to BN in Table I. We use Original to indicate the branch with identity function in order to emphasize the traces are without pre-processing. For example, the loss function and optimizer are NLL and Adam, respectively.

IV. Experimental Results

In this section, we evaluate the performance of the proposed MCNN architectures from Table I against BN. The comparison is performed on various publicly available datasets. While [9] recommend different architectures for different datasets with the objective of achieving the best results for all datasets, our MCNN experiments do not aim at minimizing $\bar{N}t_{GE}$, but we aim at demonstrating that MCNN performs well across datasets in general. Therefore, the evaluation is performed with

Arch.	Transformation stage (PBC)	Local convolution stage			Full convolution stage			Multiperceptron layer
		filters	kernel size	pool size	filters	kernel size	pool size	
MCNN	Original MA PCA	(32,64) (32,64) (32,64)	(1,50) (1,50) (1,1)	(2,50) (2,50) (2,1)	128	3	2	$20 \times 20 \times 20$
MCNN	Original MA POC	(32,64) (32,64) (32,64)	(1,50) (1,50) (1,50)	(2,50) (2,50) (2,50)	128	3	2	$20 \times 20 \times 20$
MCNN	Original MA EA	(32,64) (32,64) (32,64)	(1,50) (1,50) (1,50)	(2,50) (2,50) (2,50)	128	3	2	$20 \times 20 \times 20$
MCNN*	Original MA PCA	(64) (64) (64)	(15) (15) (1)	(2,50) (2,50) (2,1)	128	3	2	$20 \times 20 \times 20$
BN	Original	-	-	-	(32,64,128)	(1,50,3)	(2,50,2)	$20 \times 20 \times 20$
BN	MA	-	-	-	(32,64,128)	(1,50,3)	(2,50,2)	$20 \times 20 \times 20$
BN	PCA	-	-	-	(32,64,128)	(1,1,3)	(2,1,2)	$20 \times 20 \times 20$
BN	POC	-	-	-	(32,64,128)	(1,50,3)	(2,50,2)	$20 \times 20 \times 20$
BN	EA	-	-	-	(32,64,128)	(1,50,3)	(2,50,2)	$20 \times 20 \times 20$
BN*	Original	-	-	-	(64,128)	(15,3)	(2,50,2)	$20 \times 20 \times 20$

All hyperparameters for MCNN and BN are employed from [9].
The definition of BN* and MCNN* refers to Section IV-C.

TABLE I: Network Architecture for BN and MCNN

Dataset	#Train	#Validation	#Test
AES_HD	$45,000$	$5,000$	$5,000$
AES_HD_MM	$45,000$	$5,000$	$5,000$
AES_RD	$20,000$	$5,000$	$5,000$
ASCAD	$45,000$	$5,000$	$5,000$
ASCAD(desync=50)	$45,000$	$5,000$	$5,000$
ASCAD(desync=100)	$45,000$	$5,000$	$5,000$

TABLE II: Number of train/test sets in all open datasets to perform BN and MCNN

a fixed network architecture. The chosen BN as a baseline architecture is motivated as the most complex architecture of all the architectures proposed in [9] as it will learn different datasets with ease compared to smaller architectures. Since BN has been reported to perform better than template attacks in [9], we focus on comparison with deep learning models only.

A. Target Dataset & Notations

For the experiments, we consider the following 4 public datasets, which are freely available online, for reproducibility.

a) ASCAD: The dataset[2] contains side-channel measurements of protected AES implementations running on an 8-bit AVR microcontroller. It was introduced by Benadjila *et al.* [4], as a public dataset for comparing the performance of deep-learning based side-channel attacks. The ASCAD database traces correspond to first order masking protected AES with artificially introduced random jitter. In particular, for the experiments, the introduced jitter (desynchronization) are set to range up to 50 and 100 sample points. We represent the desynchronization of 50 and 100 as ASCAD(desync=50) ASCAD(desync=100), respectively. The dataset consists of $60,000$ traces, with 700 features each.

b) AES_RD: The dataset[3] is based on AES software implementation on an 8-bit AVR microcontroller. The implementation is protected with a random delay countermeasure [35] to cause misalignment in the traces, which in turn reduces the SNR. The dataset consists of $50,000$ traces, with $3,500$ features each.

[2]https://github.com/ANSSI-FR/ASCAD
[3]https://github.com/ikizhvatov/randomdelays-traces

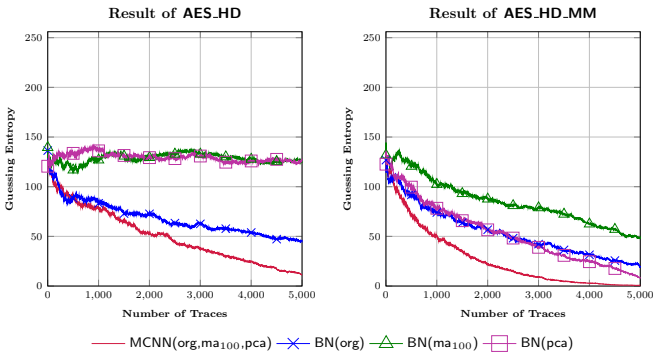


Fig. 4: Results for AES_HD and AES_HD_MM.

c) AES_HD: The dataset[4] includes an unprotected AES hardware implementation on FPGA. Unlike software implementations, the last round is considered as a main target, in order to utilize the register update leakage from last round to output ciphertext. There are $50,000$ traces with $1,250$ points in the dataset.

d) AES_HD_MM: In all the previous works, DL based SCA have focused on countermeasures implemented for software targets. However, AES_HD_MM dataset[5] is based on multiplicative masking countermeasure [36] implemented in hardware. The implementation performed masked AES on SASEBO-GII FPGA board [37]. According to [38], the success rate for this countermeasure is only 90%, even though they launched second-order attacks [38] with $500,000$ traces. Dataset contains $5,600,000$ traces with $3,125$ points are provided in their open URL[5]. For attack, the leakage model is identical to AES_HD dataset, which means second-order attack on a hardware countermeasure.

e) Notations and Parameters.: BN(x) and MCNN($PBC_1(x)$,$PBC_2(x)$,$PBC_3(x)$) indicate the Base Network with original traces x and MCNN with PBCs (PBC_1, PBC_2, PBC_3), respectively. For MCNN, PBC_1 is an identity function (org), PBC_2 is moving average (ma) and PBC_3 as PCA (pca). For later experiments, we change PBC_3 to POC (poc) and EA (ea). In the case of moving average technique, step size is a required parameter when merging from n points to single point. Hence, we represents it as ma_n. For example, BN(ma_{100}) (instead of $\mathsf{BN}_{\mathsf{ma}_{100}}(x)$ for simplicity) means Base Network having input as original traces applying moving average technique with merging 100 points to single point. The original traces are datasets such as AES_HD, AES_HD_MM, ASCAD, ASCAD(desync=50), and ASCAD(desync=100).

Additionally, "+" notation which is used as the input for BN in order to fairly compare with the MCNN indicates merging of datasets. For instance, org+ma means that a training dataset consists of original traces and the output of moving average applied to original traces. If the dimension is not matched, we use a zero padding scheme.

[4]https://github.com/AESHD/AES_HD_Dataset
[5]https://chest.coe.neu.edu/\?current_page=POWER_TRACE_LINK&software=ptmasked

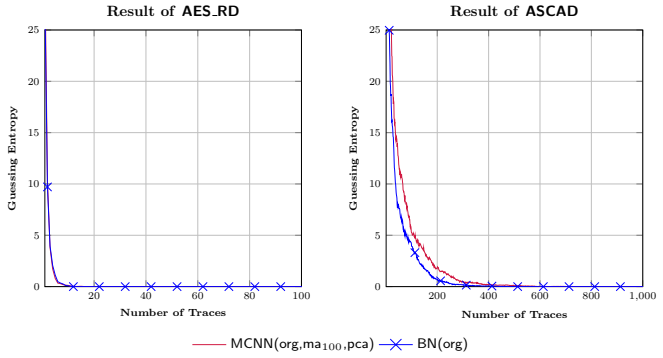


Fig. 5: Results for AES_RD and ASCAD.

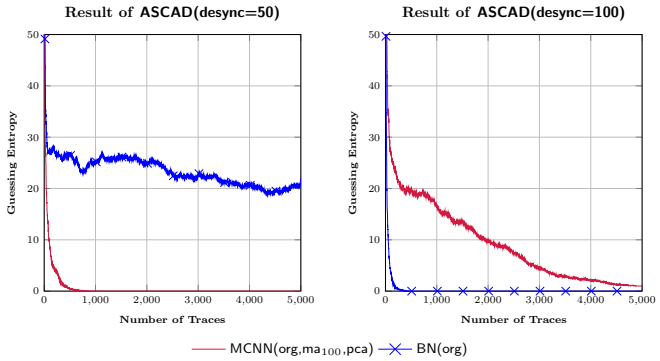


Fig. 6: Results for ASCAD(desync=50) and ASCAD(desync=100).

B. Comparing MCNN with BN

Performance of MCNN and BN is compared across datasets. All experiments in this section are done with MCNN(org, ma_{100}, pca). Traces in each dataset were split for training, validation and testing. For a fair comparison, we followed a similar split as in [9]. We also use a similar dataset split ratio for AES_HD_MM. The ratio is listed in Table II. The attack is repeated 100 times and GE is obtained by averaging the result over these 100 attacks.

In Figure 4, we first report the results for the attacks on AES hardware implementations. For the unprotected implementation, AES_HD, we can see that the proposed model, MCNN, performs better than BN. Here BN only learns from the raw AES_HD traces, MCNN also learns from the PBC branches obtained through moving average with step 100 and PCA. To have a better insight into the results, we repeat the experiments with BN by training it with transformed traces using moving average with step 100 and PCA in two independent experiments. In this case, the transformations are applied to the training set directly. As shown in Figure 4, the transformation of the traces alone does not give good results as shown for BN(ma_{100}) and BN(pca). This shows that it is an inherent property of MCNN which allows it to learn more features than BN alone and result in better attacks. Note that BN is not designed for AES_HD and thus the results are worse than those reported in [9], still we report significant improvements with MCNN. Next, we take a look at AES_HD_MM dataset, where we target traces for a FPGA implementation of AES-128 protected with multiplicative masking. MCNN shows a significantly faster convergence to GE 1 as compared to BN with original traces as well as two sets of transformed traces.

Next, we look into datasets for software implementation of AES. These include AES_RD dataset and ASCAD dataset with different desynchronization. Figure 5 shows the results ASCAD dataset with no desynchronization and AES_RD. These two datasets are easy to break as shown in various previous works [9], [6]. Both MCNN and BN have no trouble in learning these datasets and perform very well.

We next move to test the ASCAD dataset with desynchronization. Here we have two cases, ASCAD(desync=50) and ASCAD(desync=100). Recall that BN, as proposed in [9], is optimized for ASCAD(desync=100) and indeed as shown in Figure 6, BN performs best. While MCNN performs fine in this case, it is not as good as BN. This reinstates the results of [9] that networks optimized for a chosen dataset perform best. Figure 6 shows the results for ASCAD(desync=50). ASCAD(desync=50) is a special case as it can be considered a subset of ASCAD(desync=100) where desynchronization is limited to 50 samples only. Intuitively, we expect BN to perform well in it, however, our results show that BN struggles with this dataset and MCNN performs best. This shows that BN is probably over-optimized for ASCAD(desync=100) dataset. BN(ma_{100}) and BN(pca) GE results for ASCAD and AES_RD did not converge as in previous cases.

Comparing Figure 5 and Figure 6, we can see that MCNN performs well in all the datasets. Since BN is fine-tuned for ASCAD(desync=100), it performs the best on that particular dataset, but its performance on other datasets is not guaranteed. Note that, MCNN does not outperform the results of [9] when considering architectures optimized for each dataset, and neither it is the objective of this work. While, as shown in [9] it is possible to propose efficient attacks for various datasets by fine-tuning the architecture for a chosen dataset, MCNN here is proposed as a common architecture that would generalize better across datasets.

The overall results across all the datasets are stated in Table III. It shows that MCNN can scale to different datasets without the need of changing the network structure or even PBCs.

1) Comparison on the Resistance of Reinforced Jitter-based Countermeasure: To investigate the capability of MCNN in generalizing further, we conduct a special set of experiment. In this case, the training dataset is derived from ASCAD(desync=50), while the testing dataset is derived from ASCAD(desync=100). Note that ASCAD(desync=50)

Arch.	ASCAD	ASCAD (desync=50)	ASCAD (desync=100)	AES_RD	AES_HD	AES_HD_MM
MCNN	Good	Good	Good	Good	Good	Good
BN	Good	Bad	Good	Good	Average	Average

TABLE III: Overall performance of MCNN vs. BN on different datasets. *Good* indicates GE $\leq$ 10 at 5k traces, *Average* indicates GE declining steadily but not reaching GE $\leq$ 10 at 5k traces, and *Bad* means that it is not clear when the GE can converge.

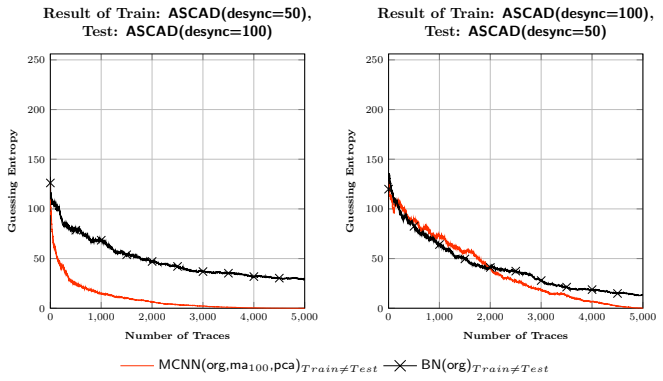


Fig. 7: Results of dissimilarity between train and test datasets based on ASCAD(desync=50) and ASCAD(desync=100).

and ASCAD(desync=100) differ in the jitter offset range only, while everything else remains the same. Thus, we are testing the trained network with cases never seen by the training model. In other words, while the network learns to recognize jitter up to 50 samples in either direction, the testing data can have jitter up to 100 samples. The results are shown in Figure 7 (left). It can be clearly seen that MCNN performs better thus being able to learn the underlying problem with more ease than BN. The opposite *i.e.* training dataset derived from ASCAD(desync=100), while the testing dataset derived from ASCAD(desync=50), is shown in Figure 7 (right). As training datasets already contain samples expected in the testing dataset, the performance of both the networks is comparable.

2) Comparison on the Ensembles in Machine Learning-based Profiled SCA: Hyperparameter tuning remains a major challenge in deep learning as there is no optimal way to find the best model parameters. Ensembles in deep learning aim to combine the decision of several models in order to improve the generalization performance of the overall predictor. In the context of side-channel attack, Perin *et al.* [10] recently showed that ensemble perform better than single models. As MCNN also learns from three branches it can be easily confused or compared with ensembles. However, there are subtle differences between the two methodologies. Ensemble uses multiple models trained separately and combines their output probabilities in an optimal manner to maximize success. MCNN on the other hand acts upon the input features by using branches to extract the relevant features from different input data transformations while training a single model.

We performed experiments to compare ensemble with MCNN on side-channel datasets. We use the publicly available code from [10]. We adjusted the number of models to directly compare with MCNN with 10 models overall and 3 best models. The choice of three best models is to directly compare with three branches of MCNN. The results are shown in Figure 8 and even with the 3 best models, MCNN performs much better. The time required to train 10 ensemble classifiers, used for selecting 3 best models, is also significantly higher than MCNN as shown in table IV. To increase the performance of ensembles, the number of models must be increased as

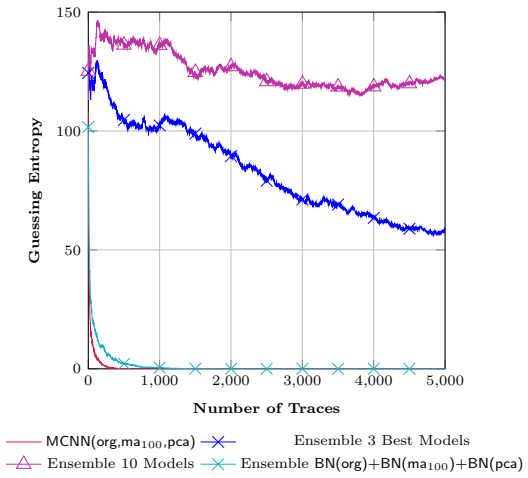


Fig. 8: Results for MCNN and Ensemble on ASCAD dataset.

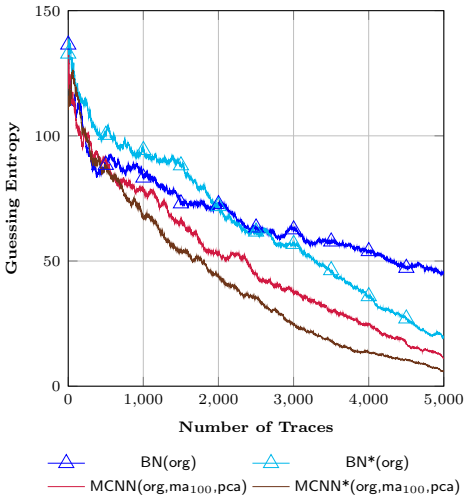


Fig. 9: Comparison of various architectures when trained and tested on AES_HD dataset.

already shown in [10]. Moreover as already pointed out in [14], 3-layer MCNN achieves similar accuracy as an ensemble model with 35 classifiers for other datasets.

Further, we build an ensemble from three chosen models BN(org), BN(ma_{100}), BN(pca). This can be considered a special case of an ensemble, more powerful than proposed in [10], as [10] does not include pre-processing and works with generic models rather than models optimized to a particular dataset. While the ensemble of BN(org), BN(ma_{100}), BN(pca), performs better than previous experiments of ensemble, MCNN outperforms it by a slight margin. This difference can be potentially attributed to the fact that while the ensemble simply combines output probabilities, MCNN combines the three models with an additional convolutional and pooling layer to combine the learning of three models.

C. On Versatility of MCNN

The previous results have shown that MCNN can perform and generalize better across datasets when compared to BN,

Dataset	Arch.	Learning Time (seconds)	Complexity (Trainable Parameter)
ASCAD	MCNN(org,ma$_{100}$,pca)	$4,950$	$267,676$
	BN(org)	$3,000$	$141,596$
	Ensemble	$59,300$	$2,743,860 \sim 6,321,616$
AES_HD	MCNN(org,ma$_{100}$,pca)	$9,500$	$280,476$
	BN(org)	$5,750$	$149,276$
	MCNN*(org,ma$_{100}$,pca)	$1,900$	$70,108$
	BN*(org)	$1,050$	$47,708$
AES_RD	MCNN(org,ma$_{100}$,pca)	$5,120$	$339,356$
	BN(org)	$3,000$	$177,436$
AES_HD_MM	MCNN(org,ma$_{100}$,pca)	$21,500$	$329,116$
	BN(org)	$12,750$	$172,316$

TABLE IV: Learning time and complexity for BN and MCNN

which was known to produce the best results in public literature. Recently, an improvement to BN was proposed by Wouters *et al.* [12], where simplifying the initial convolution layer resulted in reducing the number of parameters while keeping comparable attack performance. We call this updated network from [12] as BN*(refer to Table I for details). To show that MCNN is not limited to one network, we build a variant of MCNN based on BN*(Refer to MCNN*). The attack performance of original and updated networks are reported in Figure 9. As expected from public literature, BN*(org) performs better than BN(org). The baseline MCNN(org,ma$_{100}$,pca) already performs better than both BN(org) and BN*(org). The improved MCNN*(org,ma$_{100}$,pca) built upon BN*(org) shows the best results. These results reinstate two key features of MCNN. Firstly, MCNN can be adapted to any baseline architecture. Secondly, MCNN can learn features from different PBCs and generalize better enabling better performance than the baseline network in most cases.

V. Integrating Proven SCA Pre-Processing Techniques in MCNN

In the previous section, we show on public datasets that MCNN can achieve consistent performance even without modifying the network architecture. While the MCNN architecture used two well known pre-processing techniques, these pre-processing techniques were not chosen based on the dataset. Nevertheless, MCNN performed better across the datasets.

As discussed previously, we propose MCNN as a general framework where the PBCs can be exploited to integrate any pre-processing technique into the framework. Consider an evaluation lab conducting security evaluation of several products on a daily basis. Over the course of years, evaluators in these labs see various countermeasures and develop various pre-processing techniques to optimize the evaluation. The current state of deep learning research for SCA has majorly focused on optimizing architectures so as to bypass the pre-processing phase altogether. With PBC in the proposed MCNN architecture, MCNN provides the opportunity for an evaluator to integrate those tested and proven pre-processing techniques directly into deep learning based SCA evaluation. While the list of pre-processing techniques (including their parameter space) is non-exhaustive, we demonstrate this feature of the proposed MCNN framework by two distinct case studies. In the first case study, we focus on improving the current MCNN architecture by optimizing transformation parameters. The second case study focuses on integrating other well-known pre-processing techniques as PBCs to MCNN.

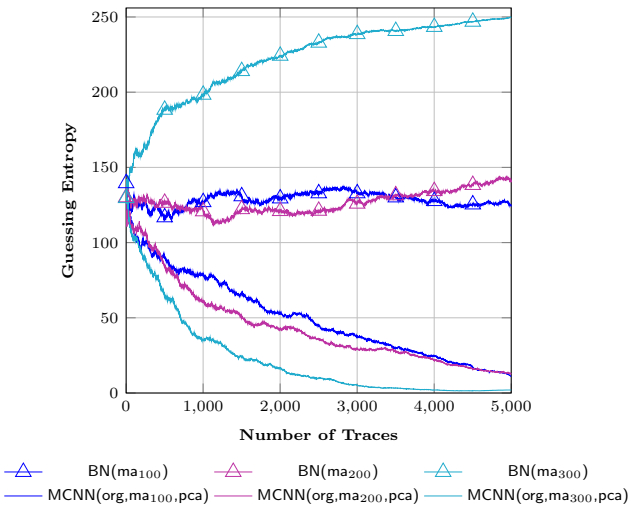


Fig. 10: Results for AES_HD in various moving averages.

A. Case Study 1: Optimizing Transformation Parameters in Existing MCNN Architecture

The choice of moving average as a PBC in MCNN was inspired by the original MCNN [14]. However, in the previous experiments, we did not consider optimization of moving average parameters to suit dataset characteristics, while still showing good results across datasets. In this part, we experiment with moving average parameters to evaluate their effect on AES_HD dataset. As mentioned earlier, AES_HD implements an AES-128 parallel architecture to compute one round per clock. Here, different sub-components of the cipher leak in different parts of the clock cycle. By measuring on a high sampling rate oscilloscope, these leakages might be spread over different points but within a single clock cycle or to a neighbouring clock cycle in some cases. This is unlike software computation where sub-operations might be separated by several clock cycles. Thus, for hardware implementation, simple signal processing techniques like moving average allow the combination of leakage, allowing an attacker to exploit contribution of several leakages spread over a number of points.

For AES_HD, we investigated the effect of the parameters for moving average. We consider the parameters space with varying step size $\in \{100, 200, 300\}$, thus MCNN(org,ma$_{100}$,pca), MCNN(org,ma$_{200}$,pca) and MCNN(org,ma$_{300}$,pca). Here step size refers to the width of the window used for calculating moving average while sliding through the trace. The default step size 100 was chosen to fit BN parameters and adjusted in the chosen range. The results are shown in Figure 10. By choosing a bigger step size for the moving average, the result of MCNN can be largely improved as compared to the previous result in Figure 4. On the other hand, playing with moving average parameters, does not improve the attack results for BN.

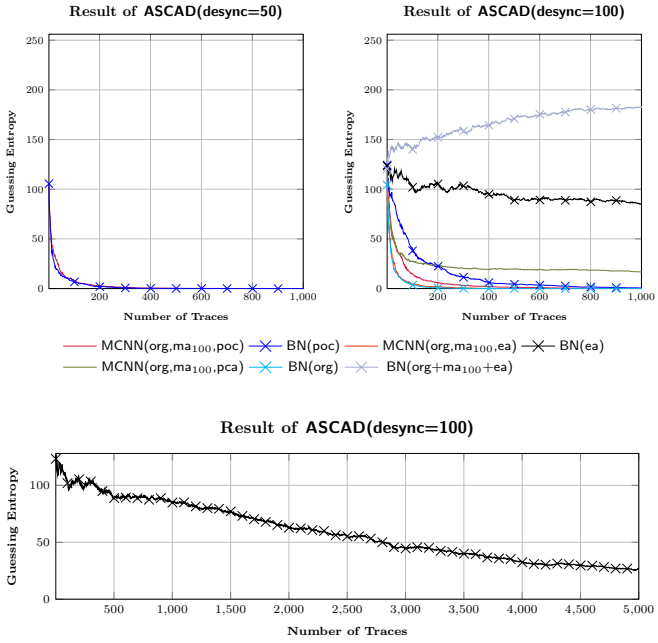
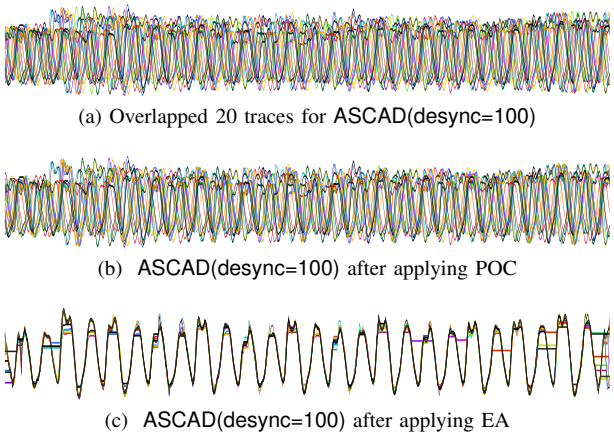


Fig. 11: Comparison of performance of MCNN and BN architectures when working with pre-processed traces of ASCAD(desync=50) and ASCAD(desync=100) datasets.



(a) Overlapped 20 traces for ASCAD(desync=100)

(b) ASCAD(desync=100) after applying POC

(c) ASCAD(desync=100) after applying EA

Fig. 12: Overlapped 20 traces ASCAD(desync=100) after pre-processing techniques.

B. Case Study 2: Integrating New Pre-Processing Techniques In MCNN

Finally, in this case study, we investigate the effectiveness of MCNN by plugging in known pre-processing techniques in SCA to replace PCA used as PBC in the original architecture. We choose 2 known techniques. The first technique was proposed by Homma *et al.* [29] in CHES 2006 and is known as POC. The second technique investigated is EA [30] which was proposed at CT-RSA 2011 and also available in few commercial tools for SCA evaluations. To perform the evaluations, we choose ASCAD(desync=50) and ASCAD(desync=100), as these datasets implement jitter or misalignment countermeasure. Both POC and EA are designed to overcome misalignment. It was also shown previously that pre-processing traces can improve efficiency of deep learning based evaluations [3], [13]. MCNN is different from these previous works because, while previous works were modifying the training dataset altogether by pre-processing, MCNN applies these pre-processing on the fly in one of its branches through PBC, while the training set remains unchanged. Few sample traces from ASCAD(desync=100) before and after alignment are shown in Figure 12. It can be seen that EA works better than POC in this case. In essence, a good alignment method is converting these traces close to synchronised ASCAD database and one should expect similar results. The two modified MCNN used in the following experiments are MCNN(org, ma_{100}, poc) and MCNN(org, ma_{100}, ea).

The results are reported in Figure 11. From the figure, we can see that in the case of desynchronization = 50, all methods are working successfully, including BN, which is unsuccessful in recovering the key in the original experiments (see Figure 6). Thus, we confirm the results of [3], [13] that pre-processing helps deep-learning based SCA evaluations.

Now, we look at a comparatively difficult case of ASCAD(desync=100) with higher desynchronization = 100. In this case, MCNN is performing slightly better than BN with POC pre-processing, where we can observe faster convergence and smaller number of traces required to recover the key, and much better than BN with other pre-processing methods. In general, for all the experiments conducted, we can observe that the performance of MCNN is consistent throughout different datasets and different parameter settings. With a good choice of PBC, we were able to match the performance of BN(org) with MCNN(org,ma$_{100}$,poc), where BN(org) is specifically designed to perform best for ASCAD(desync=100).

We note that even though visually EA results in better alignment compared to POC (see Figure 12), BN(ea) performs worse than BN(poc). Indeed, EA introduces distortion to the trace, which can make it more difficult to learn as compared to POC. Moreover, BN(org) is optimized for ASCAD(desync=100) and not for pre-processed traces, thus BN(org) performs better for the original dataset, while MCNN generalizes better. This demonstrates the importance of capturing features in different scales and frequencies. EA helps to enhance the long term features by aligning the traces, while POC analyzes the discrete Fourier transforms of waveforms and extracts features in various frequency domains. The observation further confirms the benefits of data prepossessing done in the PBCs. Finally, we show in Figure 11b (bottom), that BN(ea) does converge, trending towards GE=0 albeit requiring more traces.

Finally, we investigate if MCNN is not simply doing feature space augmentation. To check this, we augment the training dataset with feature transformed traces and see if BN can perform better. In other words, we take ASCAD(desync=100) dataset, transform it using moving average and EA to get two separate datasets, merge it with the original dataset to have an augmented dataset with $3\times$ traces, and use this augmented dataset to train and test BN. As shown in Figure 11, the result is much worse and confirms that MCNN is not simply augmenting the dataset but exploiting salient features from all the transformations to bring a more complex model.

VI. Conclusions

In this paper, we presented a neural network architecture for profiled side-channel attacks based on multi-scale convolutional neural networks (MCNN). We proposed a general framework that can be used for building MCNN models that can effectively perform SCA tasks on various datasets without fine-tuning of parameters. Our results show that MCNN has a great potential to serve as an architecture of choice when the details of the leakage traces are not available to the attacker while providing the power to the attacker to integrate pre-processing seamlessly into the architecture.

a) Future directions: Different architectures based on the idea of MCNN would be interesting to explore. For example, in [17], the authors use multi-scale recurrent CNN (RCNN) and report superior results on financial time series data compared to other models. They claim that using RCNN over CNN improves capturing of temporal dependencies in the data. In time series classification, long short-term memory (LSTM) models are a popular approach to solve tasks that would not be possible to solve with traditional feed-forward networks [39], [40]. Therefore, LSTM might offer additional ways to analyze SCA leakage traces, albeit with proper adaption as works like [40] shifts the focus away from pre-processing step.

Different PBCs could be explored to enhance the feature transformation step. Autoencoders, successfully used for SCA before [41], [42], could be plugged as a PBC to improve the performance of the model. Moreover, we only looked at non-profiled data pre-processing techniques in this work. it would be interesting to investigate methods to integrate profiled pre-processing (like linear discriminant analysis, autoencoders) into MCNN as a PBC.

Automated selection of PBCs with usage of neural architecture search (NAS) [43] could be implemented. NAS approaches iterate over different architectures and try various hyperparameters to find the best model for the task. In terms of SCA, there could be a pool of different PBCs, and the branches would be chosen automatically by NAS based on their performance.

References

[1] H. Maghrebi, T. Portigliatti, and E. Prouff, "Breaking cryptographic implementations using deep learning techniques," in *International Conference on Security, Privacy, and Applied Cryptography Engineering*. Springer, 2016, pp. 3–26.

[2] E. Cagli, C. Dumas, and E. Prouff, "Convolutional neural networks with data augmentation against jitter-based countermeasures," in *International Conference on Cryptographic Hardware and Embedded Systems*. Springer, 2017, pp. 45–68.

[3] Y. Zhou and F.-X. Standaert, "Deep learning mitigates but does not annihilate the need of aligned traces and a generalized resnet model for side-channel attacks," *Journal of Cryptographic Engineering*, pp. 1–11, 2019.

[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*, pp. 1–26, 2019.

[5] S. Picek, A. Heuser, A. Jovic, S. Bhasin, and F. Regazzoni, "The curse of class imbalance and conflicting metrics with machine learning for side-channel evaluations," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2019, no. 1, pp. 1–29, 2019.

[6] J. Kim, S. Picek, A. Heuser, S. Bhasin, and A. Hanjalic, "Make some noise. unleashing the power of convolutional neural networks for profiled side-channel analysis," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 148–179, 2019.

[7] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[8] L. Masure, C. Dumas, and E. Prouff, "A comprehensive study of deep learning for side-channel analysis," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 348–375, 2020.

[9] 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, 2020.

[10] G. Perin, Ł. Chmielewski, and S. Picek, "Strength in numbers: Improving generalization with ensembles in machine learning-based profiled side-channel analysis," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 337–364, 2020.

[11] Y.-S. Won, D. Jap, and S. Bhasin, "Push For More: On Comparison of Data Augmentation and SMOTE With Optimised Deep Learning Architecture For Side-Channel," 2020, https://eprint.iacr.org/2020/655.

[12] L. Wouters, V. Arribas, B. Gierlichs, and B. Preneel, "Revisiting a Methodology for Efficient CNN Architectures in Profiling Attacks," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2020, no. 3, pp. 147–168, Jun. 2020. [Online]. Available: https://tches.iacr.org/index.php/TCHES/article/view/8586

[13] A. Golder, D. Das, J. Danial, S. Ghosh, S. Sen, and A. Raychowdhury, "Practical approaches toward deep-learning-based cross-device power side-channel attack," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 12, pp. 2720–2733, 2019.

[14] Z. Cui, W. Chen, and Y. Chen, "Multi-Scale Convolutional Neural Networks for Time Series Classification," *arXiv preprint arXiv:1603.06995*, 2016.

[15] Z. Yao, Z. Zhu, and Y. Chen, "Atrial Fibrillation Detection by Multi-Scale Convolutional Neural Networks," in *2017 20th International Conference on Information Fusion (Fusion)*. IEEE, 2017, pp. 1–6.

[16] T. Sivanagaraja, M. K. Ho, A. W. Khong, and Y. Wang, "End-to-end speech emotion recognition using Multi-Scale Convolution Networks," in *2017 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*. IEEE, 2017, pp. 189–192.

[17] L. Guang, W. Xiaojie, and L. Ruifan, "Multi-Scale RCNN Model for Financial Time-series Classification," *arXiv preprint arXiv:1911.09359*, 2019.

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

[19] G. Hospodar, B. Gierlichs, E. De Mulder, I. Verbauwhede, and J. Vandewalle, "Machine learning in side-channel analysis: a first study," *Journal of Cryptographic Engineering*, vol. 1, no. 4, p. 293, 2011.

[20] F.-X. Standaert, T. G. Malkin, and M. Yung, "A unified framework for the analysis of side-channel key recovery attacks," in *Annual international conference on the theory and applications of cryptographic techniques*. Springer, 2009, pp. 443–461.

[21] L. Wu, L. Weissbart, M. Krcek, H. Li, G. Perin, L. Batina, and S. Picek, "On the attack evaluation and the generalization ability in profiling side-channel analysis," Cryptology ePrint Archive, Report 2020/899, 2020. https://eprint.iacr.org/2020/899, Tech. Rep.

[22] D. Fledel and A. Wool, "Sliding-Window Correlation Attacks against Encryption Devices with an Unstable Clock," in *International Conference on Selected Areas in Cryptography*. Springer, 2018, pp. 193–215.

[23] C. H. Gebotys and B. A. White, "A Sliding Window Phase-Only Correlation Method for Side-Channel Alignment in a Smartphone," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 14, no. 4, pp. 1–22, 2015.

[24] A. A. Ding, C. Chen, and T. Eisenbarth, "Simpler, Faster, and More Robust T-test based Leakage Detection," in *International workshop on constructive side-channel analysis and secure design*. Springer, 2016, pp. 163–183.

[25] C. Archambeau, E. Peeters, F.-X. Standaert, and J.-J. Quisquater, "Template attacks in principal subspaces," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2006, pp. 1–14.

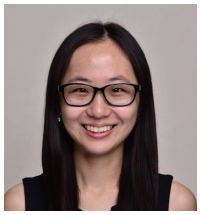
[26] L. Batina, J. Hogenboom, and J. G. van Woudenberg, "Getting more from PCA: first results of using principal component analysis for extensive power analysis," in *Cryptographers' track at the RSA conference*. Springer, 2012, pp. 383–397.

[27] O. Choudary and M. G. Kuhn, "Efficient template attacks," in *International Conference on Smart Card Research and Advanced Applications*. Springer, 2013, pp. 253–270.

[28] Q.-s. Chen, M. Defrise, and F. Deconinck, "Symmetric phase-only matched filtering of fourier-mellin transforms for image registration

and recognition," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 16, no. 12, pp. 1156–1168, 1994.
[29] N. Homma, S. Nagashima, Y. Imai, T. Aoki, and A. Satoh, "High-Resolution Side-Channel Attack using Phase-based Waveform Matching," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2006, pp. 187–200.
[30] J. G. van Woudenberg, M. F. Witteman, and B. Bakker, "Improving differential power analysis by elastic alignment," in *Cryptographers' Track at the RSA Conference*. Springer, 2011, pp. 104–119.
[31] H. Sakoe and S. Chiba, "Dynamic programming algorithm optimization for spoken word recognition," *IEEE transactions on acoustics, speech, and signal processing*, vol. 26, no. 1, pp. 43–49, 1978.
[32] S. Mangard, E. Oswald, and T. Popp, *Power analysis attacks: Revealing the secrets of smart cards*. Springer Science & Business Media, 2008, vol. 31.
[33] H. Maghrebi, "Deep learning based side channel attacks in practice," Cryptology ePrint Archive, Report 2019/578, 2019, https://eprint.iacr.org/2019/578.
[34] G. Zaid, L. Bossuet, A. Habrard, and A. Venelli, "Understanding methodology for efficient CNN architectures in profiling attacks," *IACR Cryptol. ePrint Arch.*, vol. 2020, p. 757, 2020. [Online]. Available: https://eprint.iacr.org/2020/757
[35] J. Coron and I. Kizhvatov, "An Efficient Method for Random Delay Generation in Embedded Software," in *Cryptographic Hardware and Embedded Systems - CHES 2009, 11th International Workshop, Lausanne, Switzerland, September 6-9, 2009, Proceedings*, ser. Lecture Notes in Computer Science, C. Clavier and K. Gaj, Eds., vol. 5747. Springer, 2009, pp. 156–170. [Online]. Available: https://doi.org/10.1007/978-3-642-04138-9_12
[36] M.-L. Akkar and C. Giraud, "An implementation of DES and AES, secure against some attacks," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2001, pp. 309–318.
[37] SASEBO, *Evaluation environment for side-channel attacks*. [Online]. Available: http://www.risec.aist.go.jp/project/sasebo/
[38] A. A. Ding, L. Zhang, Y. Fei, and P. Luo, "A statistical model for higher order DPA on masked devices," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2014, pp. 147–169.
[39] F. A. Gers, D. Eck, and J. Schmidhuber, "Applying LSTM to time series predictable through time-window approaches," in *Neural Nets WIRN Vietri-01*. Springer, 2002, pp. 193–200.
[40] F. Karim, S. Majumdar, H. Darabi, and S. Chen, "LSTM Fully Convolutional Networks for Time Series Classification," *IEEE access*, vol. 6, pp. 1662–1669, 2017.
[41] L. Wu and S. Picek, "Remove Some Noise: On Pre-processing of Side-channel Measurements with Autoencoders," *IACR Cryptol. ePrint Arch.*, vol. 2019, p. 1474, 2019.
[42] D. Kwon, H. Kim, and S. Hong, "Improving Non-Profiled Side-Channel Attacks using Autoencoder based Preprocessing," *IACR Cryptol. ePrint Arch.*, vol. 2020, p. 396, 2020.
[43] B. Zoph and Q. V. Le, "Neural Architecture Search with Reinforcement Learning," *arXiv preprint arXiv:1611.01578*, 2016.

Yoo-Seung Won is currently a Research Scientist at physical and cryptographic engineering lab (PACE), Temasek Laboratories, Nanyang Technological University (NTU), Singapore. He received his PhD from Kookmin University in 2018, Master's from Kookmin University, South Korea in 2014. Before NTU, Yoo-Seung held position of Staff Engineer in foundry business of Samsung Electronics, South Korea from 2018. His research interests include embedded security, secure boot solution, cold boot attack, side-channel analysis and fault attack schemes and countermeasures, practical laser/EM fault injection, and deep learning security.



Xiaolu Hou is currently an Assistant Professor at Faculty of Informatics and Information Technologies, Slovak University of Technology in Bratislava. She received her Ph.D. degree in Mathematics from Nanyang Technological University (NTU), Singapore, in 2017. Her research focus is on fault injection and side-channel attacks. She also has research experience in security of neural networks, location privacy, multiparty computation and differential privacy. With a wide range of research interests, she has published her work at top venues within various fields, ranging from mathematics to computer security.

Dirmanto Jap is currently a Research Scientist at PACE Lab, Temasek Laboratories, Nanyang Technological University (NTU), Singapore. He previously received his Ph.D in Mathematics from NTU in 2016. His main research topics include physical attacks (side-channel and fault attacks) and countermeasures, practical laser/EM fault injection, application of machine learning and deep learning for side-channel attacks and hardware Trojan detection, as well as security of deep learning.

Jakub Breier is currently a Senior Scientist in Embedded Security at Silicon Austria Labs, Graz, Austria. Before that, he worked at Nanyang Technological University, Singapore on hardware security and at Underwriters Laboratories, Singapore on security evaluation of embedded devices. He received his PhD in Applied Informatics from Slovak University of Technology (STU), Slovakia in 2013, Master's in Information Technology Security from Masaryk University, Czech Republic in 2010, and Bachelor's in Informatics from STU, Slovakia in 2008. His research topics include fault and side-channel analysis methods and countermeasures, advanced fault injection techniques, and deep learning security.

Shivam Bhasin is a Senior Research Scientist and Programme Manager (Cryptographic Engineering) at Centre for Hardware Assurance, Temasek Laboratories, Nanyang Technological University Singapore. He received his PhD from Telecom Paristech, France in 2011, Master's from Mines Saint-Etienne, France in 2008. Before NTU, Shivam held position of Research Engineer in Institut Mines-Telecom, France. He was also a visiting researcher at UCL, Belgium (2011) and Kobe University (2013). His research interests include embedded security, trusted computing and secure designs. He has co-authored several publications at recognized journals and conferences. Some of his research now also forms a part of ISO/IEC 17825 standard.