Resolving the Doubts: On the Construction and Use of ResNets for Side-channel Analysis

Sengim Karayalçın[1] and Stjepan Picek[2,1][0000−0001−7509−4337]

[1] Delft University of Technology, The Netherlands
[2] Radboud University, Nijmegen, The Netherlands

Abstract. The deep learning-based side-channel analysis gave some of the most prominent side-channel attacks against protected targets in the past few years. To this end, the research community's focus has been on creating 1) powerful and 2) (if possible) minimal multilayer perceptron or convolutional neural network architectures. Currently, we see that computationally intensive hyperparameter tuning methods (e.g., Bayesian optimization or reinforcement learning) provide the best results. However, as targets with more complex countermeasures become available, these minimal architectures may be insufficient, and we will require novel deep learning approaches.
This work explores how residual neural networks (ResNets) perform in side-channel analysis and how to construct deeper ResNets capable of working with larger input sizes and requiring minimal tuning. The resulting architectures obtained by following our guidelines are significantly deeper than commonly seen in side-channel analysis, require minimal hyperparameter tuning for specific datasets, and offer competitive performance with state-of-the-art methods across several datasets. Additionally, the results indicate that ResNets work especially well when the number of profiling traces and features in a trace is large.

Keywords: Deep learning · ResNet · Side-channel Analysis · Residual Blocks

1 Introduction

Many digital devices, from phones to payment terminals, rely on encryption algorithms for security. While modern cryptographic algorithms are (presumed to be) theoretically secure, the practical implementations come with an entirely separate set of concerns. Several types of attacks that rely on some vulnerability in the implementation exist. In this work, we will focus on side-channel analysis (SCA). Side-channel analysis is a non-invasive implementation attack that focuses on extracting leaked information during the algorithm's execution. Examples of these leakages include power consumption [10], electromagnetic emanation [2], sound [1], or cache-timings [27].

In the last few years, the focus of the side-channel community has primarily shifted to attacks using deep learning techniques [12,2,8]. Various works have

looked at different neural network architectures and hyperparameter setups. Still, the general trend for state-of-the-art performance has been to take relatively small architectures [28] and to use automated techniques to find optimal hyperparameter configurations [19,25]. While some works have explored deeper architectures for specific scenarios, like attacking traces with large feature windows [11], we have not often seen larger CNN-like architectures.

Although targets protected with various countermeasures have become relatively straightforward to break with deep learning techniques [28,24,26,15], there are newer datasets that seem more difficult to successfully attack [14]. These recent datasets commonly provide more measurements (and features) for the profiling and the attacking phase. Because of the increased complexity of the datasets due to additional countermeasures, larger architectures may provide a way to break these targets. Therefore, it is essential to provide insights into the construction and performance of these larger architectures to conclude their viability for such tasks.

Since we focus on deeper neural networks, we use Residual networks (ResNets). The ResNets are neural network architectures that include shortcut connections between network layers. These shortcut connections are added as in deep networks, updating the weights in earlier layers becomes difficult as the gradient vanishes in the deeper layers. The shortcut connections then allow the gradient to skip layers and let the weights in these deeper layers be trained [6]. The benefit of ResNets in this context is that the residual connections allow deeper networks to be used without running into gradient vanishing issues. Additionally, some recent works have used ResNets for side-channel analysis and showed promising results [29,7].

1.1 Related Work

Recently, several works investigated ResNets and concluded they have relatively competitive performance with the state-of-the-art SCA. Zhou and Standaert explored the use of a ResNet architecture for dealing with desynchronized traces and showed impressive attacking results [29]. Jin et al. also showcased a ResNet architecture utilizing attention mechanisms, and the resulting architectures performed well across a fairly wide variety of datasets [7]. While both works show good attacking results using ResNets, the motivations for using these ResNets are lacking. Several design choices regarding the architecture are not motivated by experimental results. For the work of Zhou and Standaert, the architecture description is unclear and, thus, does not allow the reproduction of their results. Next, Gohr et al. used ResNets to break the CHES CTF 2018 dataset, where they explored deeper networks with up to 19 residual blocks [5]. The authors also discussed breaking the full key and not only one subkey guess, as commonly done. Finally, Masure et al. did not use ResNets but discussed them and concluded that it is possible to use distinctive features in SCA traces and avoid problems connected with deep architectures (which would then necessitate the usage of ResNets) [13].

Other works have looked at attacking datasets with large numbers of features. Lu et al. attacked the raw traces of several datasets with novel, attention-based architectures and showed impressive results [11]. The authors also required large neural network architectures for this. Indeed, they used architectures with more than 50 layers, while most of the results in the SCA domain are accomplished with architectures with less than ten layers. Perin et al. also explored various feature selection scenarios and showed that even very small architectures could have excellent results against datasets with large numbers of features [17]. Moreover, the authors showed that it is possible to break the targets in certain cases with only a single attack trace. Finally, Masure et al. looked at how the code-polymorphism countermeasure impacts the security of implementations against attacks using deep learning [13]. This countermeasure results in large feature windows, as the sensitive operations are spread across larger periods of time. Because of this, Masure et al. created adapted CNN-like architectures to deal with these large-scale traces.

We can make several observations when we look at the state-of-the-art methods for attacking more commonly used datasets in SCA. The architectures proposed by Zaid et al. are amongst the top performing architectures for several datasets [28]. These small architectures provide a base for the novel model search strategies currently providing top performance for various datasets. Wu et al. showed how Bayesian Optimization could be used to optimize the hyperparameters for small architectures for specific datasets [25]. They also found that random search can provide top performance if the hyperparameter search ranges are adequately limited. Rijsdijk et al. utilized reinforcement learning to generate top-performing architectures automatically [19]. These automated search techniques rely on training and evaluating large numbers of models and on pre-selecting reasonable ranges for hyperparameters to limit the otherwise unreasonable search ranges.

1.2 Our Contributions

This paper investigates how ResNets can be used in SCA and, more precisely, how difficult it is to tune them and how they compare with state-of-the-art results. Our main contributions are:

1. We empirically investigate several constructions for deep ResNets and provide recommendations about what type of residual block should be used and how deep the networks should be for state-of-the-art side-channel analysis.
2. From these recommendations, we construct a novel architecture that performs competitively with state-of-the-art model search strategies across several datasets. In several settings, we obtain the best-known results (compared with other types of deep learning and the same number of features).

2 Background

2.1 Residual Neural Networks

A specific family of CNN architectures is Residual Neural Networks (ResNets). One of the main problems deep CNN architectures experience is the gradient vanishing problem. There, weights in the earlier layers of the network cannot be efficiently updated using the gradient as the gradient is too small in these layers [22]. This gradient vanishing results in the training of very deep networks to be very complex. To avoid this problem, ResNets have shortcut connections that skip over some of the layers in the network. These shortcut connections allow the network to have many layers while still not experiencing the problem of vanishing gradients. Often, ResNets are implemented using residual blocks. These are blocks of convolutional layers. Then a shortcut connection is added to the output of the residual block. This connection allows the gradient to flow through the shortcut connection, which helps resolve the problems mentioned above (i.e., gradient vanishing). This shortcut connection is generally realized with a connection without any intermediate operations or a convolutional layer with kernels of size one to match the number of filters for the addition. Examples of typical constructions of residual blocks can be seen in Figure 1. Additional information about deep learning and training of neural networks is provided in Appendix A.

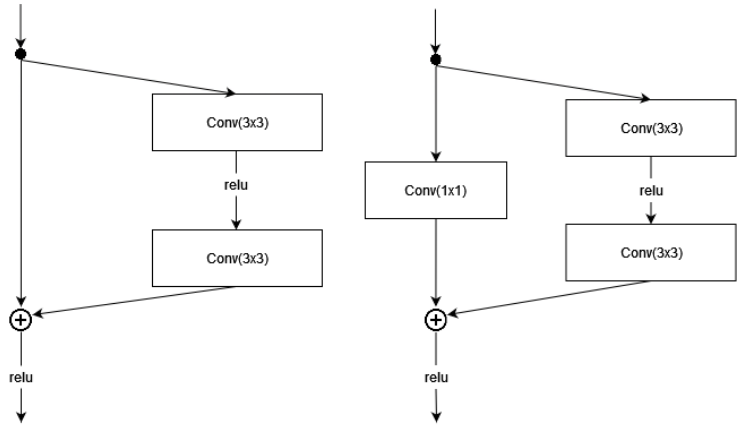


Fig. 1: Typical structure of residual blocks.

2.2 Profiling Side-channel Analysis

The setting we consider is the profiling side-channel analysis. Profiling attacks are generally more powerful than their non-profiling counterparts as the adversary is assumed to have access to (and full control over) a copy of the device

being attacked. The main workflow for profiling SCA is to take a large number of measurements from the copy of the device and create a model of its behavior. Generally, this is done by labeling all measurements using the chosen leakage model. A leakage model describes how the leakage depends on some sensitive internal state dependent on the key. Labeling the traces with this leakage model is possible as the adversary is assumed to have access to the key on the copied device. After the profile of the clone device is built, several measurements are taken from the device that is being attacked. A set of labels for these measurements is generated using each of the possible values for the key bytes, and for each of these, the log-likelihood according to the profile is computed. In cases of a successful attack, the correct key byte will have a high (ideally, the highest) log-likelihood.

Creating these profiles can be seen as a supervised learning task as measurements are collected, labeled, and then used to learn a model to predict new measurements. Over the past years, ML and DL approaches have become prevalent in profiling SCA. Specifically, the DL approach has led to very impressive results, being able to attack datasets protected by various countermeasures successfully [28,8].

SCA Metrics ML approaches for profiling SCA have proven very successful. However, accuracy and loss, metrics that are often used in standard classification tasks, are not necessarily the best indicators of how successful a side-channel attack will be [18]. While very good accuracy scores or loss values indicate a successful attack, this is not a necessary condition. Indeed, models that perform slightly better than random guessing can also generate successful attacks [15]. The models in profiling SCA are usually evaluated based on metrics related to the correct key's rank. Here, the rank of the correct key is the number of key candidates with a higher log-likelihood than the correct key. Generally, metrics are computed using a relatively large number of attacks on random subsets of the full attacking set to accurately depict the realistic (averaged) performance. More formally, we define the vector g_{S_a} as a vector of key guesses ordered by the log-likelihood of each key. Here, $S_a \subset N_a$ is a subset of the full attacking set N_a. Then, we define the index $g_{S_a}(k)$ as the index of a key candidate k. The two main metrics we will consider are:

- *Guessing Entropy*: Guessing entropy (GE) is the average key rank over a number of attacks. This is defined as $GE(N_a) = \mathbf{E}(g_{S_a}(k^*))$, where $\mathbf{E}$ is the mean function. As commonly done, we estimate this over 100 attacks.
- NoT: The NoT (Number of traces) metric refers to the number of measurements required to reduce GE to one (i.e., to see if our best guess is also the correct guess). This is defined as $min\{|N_a|\}$ for which $GE(N_a) = 1$. Then, the NoT metric estimates how many measurements are required to recover the key.

2.3 Datasets

We run our analysis on three datasets. They represent common choices when assessing the performance of deep learning-based SCA, and as such, they are used in numerous papers, see, e.g., [8,28,24].

The ASCAD database [2] provides datasets from a protected AES implementation. The sensitive value attacked for these implementations is generally the output of the third S-box in the first round (as the third key byte is the first masked one). The ASCAD database provides traces measured on an AES implementation implemented on the ATMega8515 device. This implementation is protected with the first-order masking scheme around the S-box, which can be described as:

$$S - box[p_i \oplus k_i] \oplus r_{out}.$$

Here, p_i and k_i represent the i-th byte of the plaintext and the key, respectively. r_{out} represents the additive mask.

Two versions of this dataset are provided. The first dataset has 50 000 training traces and 10 000 attacking traces. All of the traces were measured with the same key and random plaintexts. These traces are 100 000 samples (features) long, and the authors have provided a pre-selected window of 700 samples for attacking the third key byte for comparisons of attacks. We refer to this version as **ASCADf**.

The second dataset has 200 000 training traces and 100 000 attacking traces. The profiling traces were measured with random keys and plaintexts, and the attacking set has a fixed key. The traces are 250 000 samples long, and the authors have provided a pre-selected window of 1 400 samples for attacking the third key byte. We refer to this version as **ASCADr**.

The **AES_HD** dataset provides an unprotected AES hardware implementation [8]. The dataset was measured on a Xilinx Virtex-5 FPGA. The dataset contains 500 000 traces with 1 250 features each [3]. As the implementation is hardware-based, there is significantly more noise present than is the case for the **ASCAD** datasets. Another difference is that a different leakage model is used for **AES_HD**. The writing of the output to the register in the final round is attacked:

$$InvSbox[C_i \oplus k*] \oplus C_j.$$

Here, C_i and C_j are two of the ciphertext bytes, and the relationship of i and j is based on the inverse ShiftRows operation. Generally, we use $i = 16$ and $j = 12$, as this is one of the easier bytes to attack [8]. Additionally, we use the Hamming distance of this resulting operation as the leakage model.

3 Construction of Deep ResNets for SCA

Our experiments were run on a high-performance computing cluster (HPC) and we used one NVIDIA GeForce GTX 2080Ti GPU with 11 GB of memory. The

required amount of RAM ranges from sixteen to sixty-four GB of RAM, and training one of the models takes from twenty minutes to two hours, depending on the dataset and model size. The AISY Framework [16] was used to implement the experiments with Python 3.7, TensorFlow 2.0, and Keras 2.1.6.

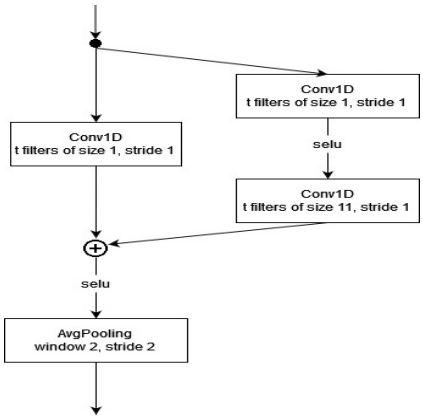
As we wish to determine whether ResNets can be a viable alternative for the more common smaller CNNs currently used in SCA, we first need to create architectures specifically created for SCA. First, we need to determine what residual blocks work well, which is a central part of our architectures. Second, it is helpful to determine how deep the architectures should be to mount successful attacks consistently.

3.1 Residual Block Construction

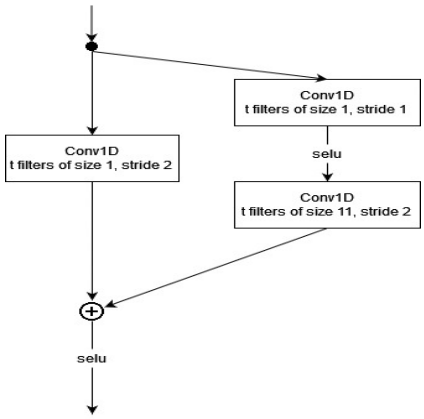
We have chosen an empirical approach to test how residual blocks should be constructed for SCA. We determine six reasonable choices for how to construct the blocks. These choices are based on recent works that explore ResNets for SCA [7,29], and we also choose the Inception block based on recent works in image recognition [21]. Inception blocks are residual blocks where the first convolutional layer has its kernel size set to one. They are included here because they could help mitigate the extra performance overhead of our deeper networks while potentially not harming the attacking performance. The tested blocks can be seen in Figure 2. It is worth noting that we limit ourselves to a relatively small number of basic residual blocks. We choose to only look at relatively simple residual blocks because we want to determine what types of basic constructions function well in deeper networks. Additionally, if even simpler blocks provide good attack performance, then it is intuitive that with more experiments, it will be possible even further improve the results and adjust for specific datasets and settings.

To limit the scope of our search for the best residual block, we decided that each block should include either a convolutional stride or a pooling layer to reduce the feature map size by half. This design decision was made since the earlier ResNets in SCA also use such blocks [7,29,14], and the reduction in feature map size in the network has been a standard in some of the state-of-the-art architectures in SCA [28,8]. We note that this choice limits the maximum number of residual blocks our network can contain. This maximum is determined by the number of features, and it equals $\lfloor log_2(700) \rfloor = 9$ (for the **ASCADf** dataset).

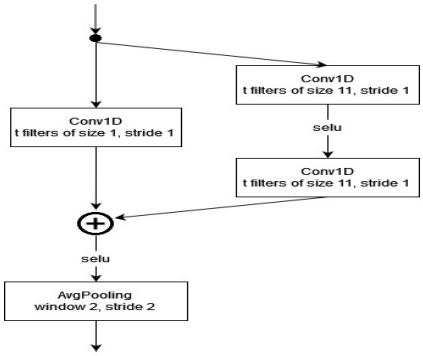
To test the blocks, we attack the **ASCADf** dataset with the pre-selected window of 700 features. We use 50 000 traces for training, and 5 000 traces for both validation and attacking. The network we use can be seen in Figure 7. The filter size is 11, and the number of filters in the i-th residual block is equal to $min(2^{i-1}, 256)$. The activation function is *selu* and the optimizer is the *Adam* [9] optimizer with a cyclic learning rate schedule [20]. The loss function we use is *categorical cross-entropy*. We use 100 epochs and a batch size of 50. We chose these hyperparameter values based on the work of Zaid et al. [28]. While they are not necessarily optimal, it is reasonable to assume good enough performance to test the differences in performance the various residual blocks can achieve.



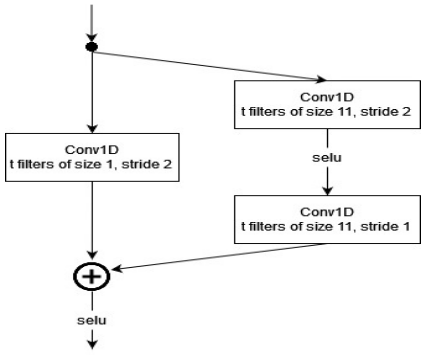
(a) Inception block with pooling for down-sampling.



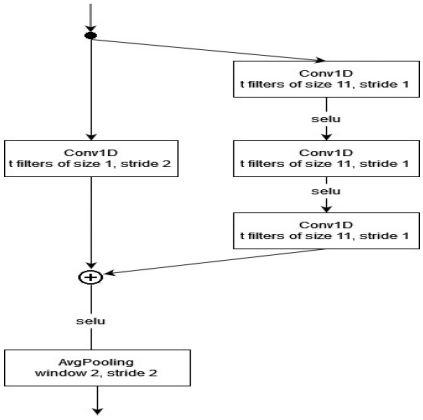
(b) Inception block with stride for down-sampling.



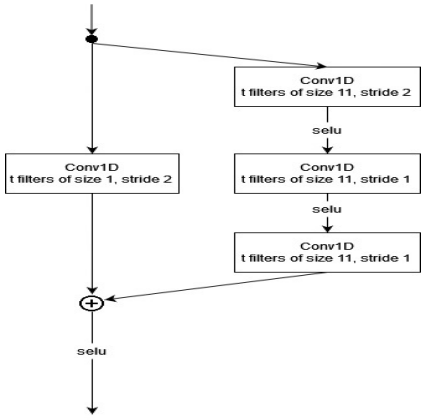
(c) Two layer block with pooling for down-sampling.



(d) Two-layer block with stride for down-sampling.



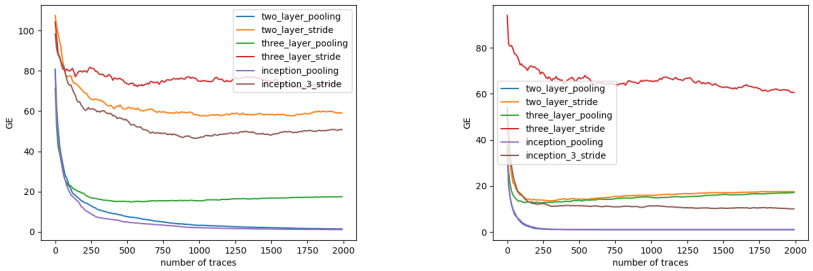
(e) Three-layer block with pooling for down-sampling.



(f) Three-layer block with stride for down-sampling.

Fig. 2: The six types of residual blocks used for testing.

To verify that the networks perform consistently, all of the results in this section are averages over five training runs of the network. Additionally, GE is computed as an average of 100 attacks. Finally, during training, we save the model that achieves the best validation loss and track the number of attacking traces required to reach $GE = 1$. In Figure 3, we see that the average attacking performance of the models with each residual block is significantly worse after the entire 100 epochs than if the model with the best validation loss is used instead. This performance difference is due to the size of the tested networks being larger than necessary for the dataset. Because of this, we see that the models overfit, as can also be seen in Figure 4. Additionally, we see that the models using pooling for feature map size reduction consistently perform better than those using convolutional stride. We believe this may be because when a convolutional stride is used[3], specific patterns might be skipped over in the convolutions. Skipping these patterns can result in specific leakage points being missed entirely, resulting in worse performance. Finally, we see that the *two_layer_pooling*- and the *inception_pooling* blocks perform best out of the chosen residual blocks, with the difference in attacking performance between these two being negligible.



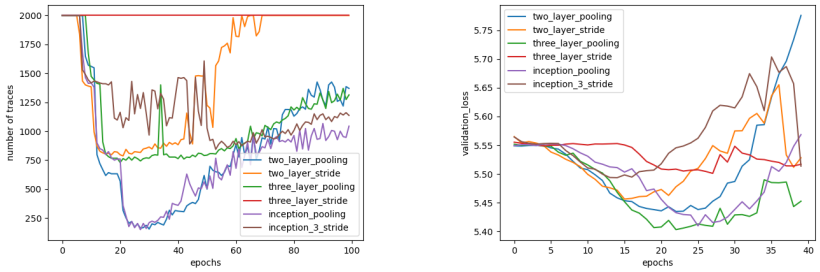
(a) Attack performance of the models after 100 epochs.

(b) Attack performance of models achieving best validation loss.

Fig. 3: Attacking performance of both the final model and the model that had the best validation loss.

In Figure 4, we observe another reaffirmation of the earlier results. We again see that generally, the *two_layer_pooling*- and the *inception_pooling* blocks perform significantly better than their counterparts in terms of the number of traces required for reaching $GE = 1$. When we look at the validation loss, we note that there is no large difference between the various blocks using pooling. However, we again observe that pooling layers are better than convolutional stride.

[3] Stride controls the amount of movement over the data. The larger the stride, the more downsampling of the data.



(a) Evolution of the NOT metric on the validation set over epochs.

(b) Evolution of validation loss across the first 40 epochs.

Fig. 4: Evolution of early stopping metrics across epochs

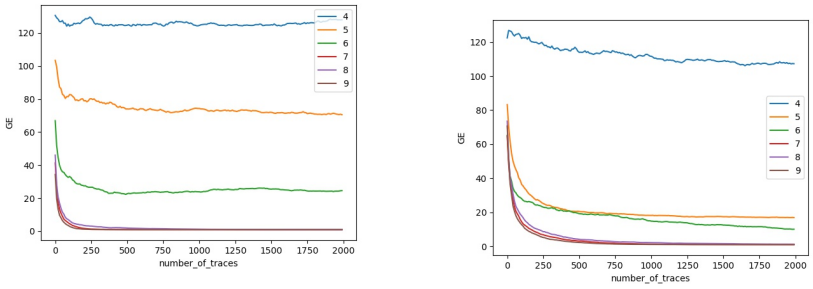
3.2 Depth of ResNets

To test how deep our Resnets should be for SCA, we conduct experiments that vary the number of residual blocks. Varying these blocks is done straightforwardly; the only notable exception is that now, we exclude the pooling layer from the final residual block. This is done as a GlobalAveragePoolingLayer immediately follows this block, and as such, including a pooling layer in the final block is pointless. For these experiments, we only use the *two_layer_pooling* residual block to limit the number of experiments that we have to run. We chose this residual block as it is one of the two best performing residual blocks in our earlier experiments. All other hyperparameters are identical to the earlier experiments, and the network setup is the same as in Figure 7, except for the number of residual blocks.

In Figure 5, we see that adding residual blocks seems to improve the attacking performance of the networks up to a point. From 4 to 7 residual blocks, the best models steadily improve the attacking results. The difference from 7 to 9 residual blocks is negligible. The results are similar for models after the full 100 epochs.

We see some of the same phenomena when we look at the progression of the validation metrics across the epochs in Figure 6. The networks with 4 to 6 residual blocks seem to perform a fair bit worse than the networks with 7 or more residual blocks. Especially for the validation loss, the networks with 9 residual blocks perform slightly better than the rest, but this does not seem to translate to improved attack performance, as observed in Figure 5.

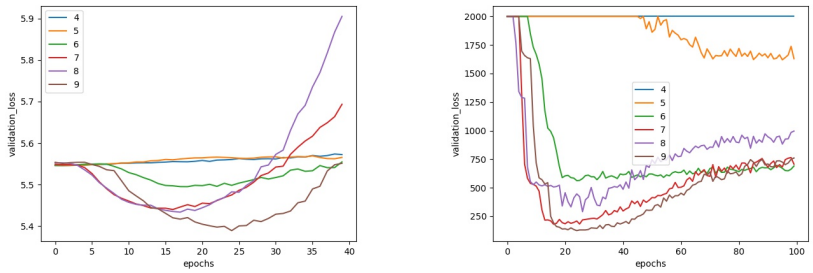
When we combine all of these results, we can provide several observations. First, using pooling layers over a convolutional stride looks like an obvious choice as the performance difference is significant. This differs from several of the other ResNets in SCA [29,14], which suggests these architectures could be improved significantly with this substitution. Second, using more than two convolutional blocks does not look useful in our experiments, and thus, we chose to use the two layer block in the next section, but the Inception version is a viable alternative.



(a) The GE results for networks of varying depth with best model weights according to validation loss.

(b) The final GE results for networks of varying depth.

Fig. 5: GE results for networks of varying depths.



(a) Evolution of the validation loss during training of residual networks of varying depths limited to the first 40 epochs.

(b) Evolution of the NOT metric during training of residual networks of varying depths.

Fig. 6: Evolution of the early stopping metric across epochs for blocks of varying depths.

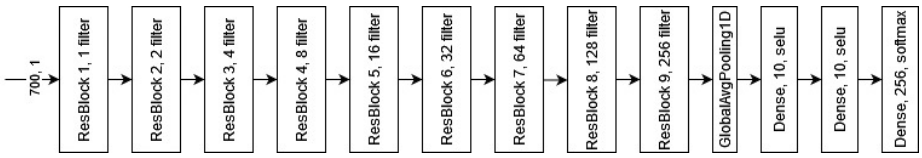


Fig. 7: Network setup for tests of various residual blocks.

Finally, we show that using more residual blocks does help the performance of the models for SCA up to a point. Our experiments show that using more than seven residual blocks does not improve performance. If we generalize this, when a dataset has x features, using $\lfloor log_2(x) \rfloor - y$ residual blocks seems a reasonable choice as using more residual blocks results in additional computational overhead

while, in our experiments, not improving performance. Our experiments showed that $y = 2$ gives reasonable results, but more experiments are needed before giving definitive recommendations. Still, we consider it "safe" to recommend that y should be a small value (e.g., 1 or 2).

4 Comparison With State-of-the-art CNN and MLP Architectures

We have explored how ResNets should be constructed for SCA to provide insights into what types of residual blocks can function well and how deep the ResNets should be. However, it is still unclear how well ResNets compare to other state-of-the-art DL approaches. Therefore, we do not yet know whether ResNets can be a viable alternative and, if they are, in what scenarios.

To remedy this, we aim to utilize the ResNets we developed to attack the **AES_HD** dataset and both versions of the **ASCAD** datasets. These experiments will allow us to conclude the efficacy of ResNets in scenarios with varying numbers of profiling traces and for both protected software and unprotected hardware implementation. Additionally, it allows us to directly compare the results of our ResNets to the results of state-of-the-art model search strategies of Perin et al. [17] on the **ASCAD** datasets and to the state-of-the-art CNN architecture of Zaid et al. [28] on the **AES_HD** dataset. The architectures we compare against from [17] are the best achieved MLP and CNN models for each dataset, using the ID leakage model in the OPOI scenario. OPOI represents optimized points of interest setting, where the attacked dataset consists of an optimized interval, including the main SNR peaks but also several low SNR points. This scenario is commonly considered in related works, see, e.g., [28,15]. In our experiments, we consider only the Identity leakage model.

The experimental setup for the **ASCAD** datasets is similar to the one used in the previous section. We use 50 000 and 200 000 profiling traces for **ASCADf** and **ASCADr**, respectively. For both **ASCAD** datasets, we use 5 000 traces for validation and 5 000 traces for attacking, and we use the standard 700 and the 1 400 feature intervals. For the **AES_HD** dataset, we use 50 000 profiling traces and 12 500 validation and attacking traces. Our hyperparameter setups in this section are generally the same as in Section 3. Two minor changes to the hyperparameters have been made for **AES_HD** and **ASCADr**. For **AES_HD**, we use a kernel size of 3 as opposed to 11 to emphasize "smaller" details that are to be expected due to more noise and hardware implementation.For **ASCADr**, we use 200 neurons in both of the fully connected layers as opposed to 10. This is because with the additional training examples this dataset contains, we can use more neurons in these final layers without the network immediately overfitting. Thus, these additional neurons allow our network to mount more powerful attacks. We again save the model weights that result in the best validation loss during training and present the result of this model. The number of residual blocks we used for each dataset depended on the number of features. We use $\lfloor log_2(\text{num\_features}) \rfloor - 2$ residual blocks for each dataset as was recommended in

Section 3. This results in 7 residual blocks for **ASCADf**, and 8 residual blocks for both **ASCADr** and **AES_HD**. For the **ASCAD** datasets, we also attack larger intervals of varying sizes around the optimized intervals. These experiments are used to evaluate the impact of more features on the performance of our ResNets. The indices of an interval of size x are then $[45\,750 - (x/2), 45\,750 + (x/2)]$ for **ASCADf**, and $[81\,645 - (x/2), 81\,645 + (x/2)]$ for **ASCADr**. It should be noted that the larger feature sets here include additional leakage points, and as such, attacks against larger windows can achieve better attack results. We take the best-performing model out of five training runs to compare our models to the state-of-the-art. Training the model several times is done as models are often inconsistent and do not always converge, and in the literature, the best-case performance is commonly showcased.

4.1 Results

ASCADf. When we compare our results to state-of-the-art results from [17], we see that our results are somewhat worse. Indeed, notice that we require almost double the number of traces for CNN and 50% more traces than MLP. However, the attack results for our ResNets are still reasonable, especially considering that we have made limited investigations into optimizing our hyperparameter configurations. We believe that the performance gap we see here can potentially be bridged with enough additional tuning. Another insight we see in Table 1 is the number of trainable parameters for each architecture. Our ResNets have a significantly higher number of trainable parameters than the relatively small architectures that [17] required. This additional size is unnecessary for this dataset and results in our network starting to overfit in the first half of the training. As we save the model with the best validation loss, the performance penalty from this overfitting is mitigated.

	ResNet	CNN [17]	MLP [17]
Nr. of traces to reach $GE = 1$	160	87	104
Nr. of trainable parameters	96 800	7 728	10 266

Table 1: Comparing the performance and size of our ResNets against the models resulting from state-of-the-art model search strategies on **ASCADf**.

ASCADr. When we compare the ResNets to state-of-the-art results in Table 2, we see that the results of our networks are better than the hyperparameter search strategy of Perin et al. [17]. Surprisingly, our models outperform these methods here, as the model search finds a model tuned specifically for this dataset. We expected that our model performed slightly worse than the state-of-the-art, as was the case for **ASCADf**. We presume this difference is because the models are trained with a significantly larger training set than for **ASCADf**. Because of this, our deeper models can learn a better profile, while

the smaller networks that result from model search techniques do not benefit from the additional information as much. An additional consideration here is that training the final models of [17] only takes a few minutes while training the larger ResNet takes about 1.5 hours for this dataset. The improved attacking performance does come at a relatively high computational cost. However, [17] train and evaluate 500 models to find these networks. Training these models can take a couple of hours to complete, or if the search space also includes bigger models, it can take several days. When we consider this, the additional cost to train one of our ResNet models is justified as it minimizes the extensive search for optimal hyperparameters (still, we note that we also required limited tuning to find our architecture). In Table 3, we provide additional results for ASCAD datasets that show more features are beneficial for the performance of ResNets, giving an additional argument in their favor. Notice how for both datasets, we find a setting where we require less than ten traces to break the target. Finally, observe that our recommendation on the required number of residual blocks generalizes well for the cases we tested.

	ResNet	CNN [17]	MLP [17]
Nr. of traces to reach $GE = 1$	47	78	129
Nr. of trainable parameters	489 592	87 520	34 236

Table 2: Comparing the performance and size of our ResNets against the models resulting from state-of-the-art model search strategies on **ASCADr**.

AES_HD. We see some interesting phenomena when attacking the **AES_HD** dataset. In Table 4, the ResNet can successfully recover the key in approximately 4 100 attack traces. This attack is significantly better than the state-of-the-art CNN model of Zaid et al., which can recover the key in about 6 060 traces. It is worth noting that the Zaid et al. network does train in only a few minutes while training our network takes approximately half an hour. Additionally, our model has significantly more trainable parameters.

Nr. of features	Nr. of residual blocks	**ASCADf**	**ASCADr**
1 500	8	36	34
2 000	8	64	37
2 500	9	86	37
3 000	9	42	29
3 500	9	6	21
4 000	9	10	14
4 500	10	9	9
5 000	10	10	8

Table 3: Comparing the number of traces required to reach $GE = 1$ at various feature selection scenarios for both **ASCADf** and **ASCADr**.

	ResNet	CNN [28]
Nr. of traces to reach $GE = 1$	4 100	6 060
Nr. of trainable parameters	111 491	3 278

Table 4: Comparing the performance and size of our ResNets against resulting state-of-the-art CNN model **AES_HD**.

4.2 Discussion

Our experiments to determine the advisable residual block constructions show that models using convolutional stride perform significantly worse than models using pooling layers. We believe this can be attributed to the fact that when a convolutional stride is used, specific patterns might be skipped over in the convolutions. Furthermore, using more than two convolutional layers per residual block seems to worsen the attack performance of the models. Therefore, our experiments indicate that using residual blocks with two convolutional layers and a pooling layer to reduce the feature map size is the best choice for deeper ResNets in SCA. This result is in line with the work of Jin et al. [7]. The residual block they used adheres to these guidelines, but the focus of that work is the addition of attention mechanisms to these blocks. On the other hand, further recent uses of ResNets for SCA do not adhere to our guidelines. Masure et al. used convolutional stride [14], and Zhou and Standaert used residual blocks with three convolutional layers [29]. This leads us to believe that these works could be improved by updating the networks, as our results indicate that some of their design choices could negatively impact the attack performance.

The second main result is that deeper variants of our networks show significantly improved attack results over the shallower variants. Our results indicate that adding residual blocks does not negatively impact attack performance and improves it up to a point. We see that the networks that perform the best in our experiments are deeper than some of the other ResNets used for SCA [7,29] [4].The networks are also significantly deeper than those used in state-of-the-art works [28,17,19]. Recently, the general trend has been to utilize hyperparameter search strategies to find small architectures that perform well. These search methods involve running large numbers of models across predefined ranges of hyperparameters. These ranges are often (loosely) based on the work of Zaid et al. [28], which outlines a methodology for constructing relatively small architectures with excellent attack performance for specific SCA datasets. Because of this, and because smaller architectures train faster, which makes training large numbers of them less cumbersome, the resulting architectures are often very shallow. This has resulted in a trend of very small architectures continually improving the state-of-the-art. Thus, while we show reasonable results, we use hyperparameter setups designed for significantly smaller networks. It

[4] Masure et al. did use a network of similar depth, but the dataset they attack has significantly more features [14]

seems reasonable to assume that even better performance is expected with a more fine-tuned setup.

The ResNet attack performance is competitive with the state-of-the-art CNNs and is achieved across various datasets. It is also worth emphasizing that a single architecture (with minor variations, like the number of residual blocks used across these scenarios or the kernel size and the number of neurons in the fully connected layers) is comparable to random search methods that result in varying model architectures and hyperparameter configurations. Additionally, while training ResNets is more computationally expensive than current state-of-the-art architectures [28], it is significantly faster than the model search strategies. Training one of the ResNets can take up to two hours in the scenarios tested here, which is a significant step up from the mere minutes it takes to train the small CNNs and MLPs that result from the searches. However, hundreds of models need to be trained and evaluated during the search for this specific architecture, which can take several hours or days to complete [19]. Additionally, the resulting architectures cannot be easily repurposed for different attack scenarios, while the ResNets only require minimal adjustments to be adapted (thus, they "generalize" better). The main use case for ResNets in SCA seems to be with larger datasets (both in terms of profiling set size and the number of traces). We see that the networks perform significantly better when the **ASCADr** dataset is used as opposed to the **ASCADf** dataset. Additionally, it is relatively intuitive that our larger networks can fit more complex models and are, therefore, more suited to more complex tasks with more training data. With new and more protected targets (where the measurements are also significantly larger), like **AES_HD_mm** [23] and the new **ASCAD** dataset [14], presumably becoming the focus of the SCA community in the coming years.

5 Conclusions and Future Work

This work developed and evaluated novel ResNet architectures for SCA. We conducted several experiments to determine suitable constructions for residual blocks and established how deep the networks should be. With these novel architectures, we attacked the **ASCAD** datasets and the **AES_HD** dataset. Our ResNets are competitive with state-of-the-art model search strategies and even surpass previous best results for the **ASCADr** dataset. Thus, we can conclude that ResNets provide an interesting new avenue for future research in SCA, especially if considering large datasets (i.e., with many features or training traces) and protected with countermeasures. As our networks provide competitive performance without much hyperparameter tuning, this leads us to believe that more tuning can result in better performance but also that limited tuning can still be a powerful option for diverse settings/datasets. Additionally, deeper ResNets could provide a way for attacking more difficult targets that current deep learning state-of-the-art cannot break. For instance, the **ASCADv2** [14] could be an interesting target to explore in future work.

References

1. Anand, S.A., Saxena, N.: A sound for a sound: Mitigating acoustic side channel attacks on password keystrokes with active sounds. In: International Conference on Financial Cryptography and Data Security. pp. 346–364. Springer (2016)
2. Benadjila, R., Prouff, E., Strullu, R., Cagli, E., Dumas, C.: Study of deep learning techniques for side-channel analysis and introduction to ascad database. ANSSI, France & CEA, LETI, MINATEC Campus, France. Online verfügbar unter https://eprint. iacr. org/2018/053. pdf, zuletzt geprüft am **22**, 2018 (2018)
3. Bhasin, S., Jap, D., Picek, S.: AES HD dataset - 500 000 traces. AISyLab repository (2020), `https://github.com/AISyLab/AES_HD_2`
4. Cunningham, P., Cord, M., Delany, S.J.: Supervised learning. In: Machine learning techniques for multimedia, pp. 21–49. Springer (2008)
5. Gohr, A., Jacob, S., Schindler, W.: Efficient solutions of the CHES 2018 AES challenge using deep residual neural networks and knowledge distillation on adversarial examples. IACR Cryptol. ePrint Arch. p. 165 (2020)
6. He, K., Zhang, X., Ren, S., Sun, J.: Identity mappings in deep residual networks. In: European conference on computer vision. pp. 630–645. Springer (2016)
7. Jin, M., Zheng, M., Hu, H., Yu, N.: An enhanced convolutional neural network in side-channel attacks and its visualization. CoRR **abs/2009.08898** (2020), `https://arxiv.org/abs/2009.08898`
8. Kim, J., Picek, S., Heuser, A., Bhasin, S., Hanjalic, A.: Make some noise. unleashing the power of convolutional neural networks for profiled side-channel analysis. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2019**(3), 148–179 (2019). https://doi.org/10.13154/tches.v2019.i3.148-179, `https://doi.org/10.13154/tches.v2019.i3.148-179`
9. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. In: Bengio, Y., LeCun, Y. (eds.) 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings (2015), `http://arxiv.org/abs/1412.6980`
10. Kocher, P., Jaffe, J., Jun, B.: Differential power analysis. In: Annual international cryptology conference. pp. 388–397. Springer (1999)
11. 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**(3), 235–274 (2021). https://doi.org/10.46586/tches.v2021.i3.235-274, `https://doi.org/10.46586/tches.v2021.i3.235-274`
12. Maghrebi, H., Portigliatti, T., Prouff, E.: Breaking cryptographic implementations using deep learning techniques. In: International Conference on Security, Privacy, and Applied Cryptography Engineering. pp. 3–26. Springer (2016)
13. Masure, L., Belleville, N., Cagli, E., Cornélie, M.A., Couroussé, D., Dumas, C., Maingault, L.: Deep learning side-channel analysis on large-scale traces. In: European Symposium on Research in Computer Security. pp. 440–460. Springer (2020)
14. Masure, L., Strullu, R.: Side channel analysis against the anssi's protected AES implementation on ARM. IACR Cryptol. ePrint Arch. p. 592 (2021), `https://eprint.iacr.org/2021/592`
15. 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**(4), 337–364 (2020)

16. Perin, G., Wu, L., Picek, S.: AISY - Deep Learning-based Framework for Side-channel Analysis. Cryptology ePrint Archive, Paper 2021/357 (2021), https://eprint.iacr.org/2021/357, https://eprint.iacr.org/2021/357
17. Perin, G., Wu, L., Picek, S.: Exploring feature selection scenarios for deep learning-based side-channel analysis. IACR Cryptol. ePrint Arch. p. 1414 (2021), https://eprint.iacr.org/2021/1414
18. Picek, S., Heuser, A., Jovic, A., Bhasin, S., Regazzoni, F.: The curse of class imbalance and conflicting metrics with machine learning for side-channel evaluations. IACR Transactions on Cryptographic Hardware and Embedded Systems **2019**(1), 1–29 (2019)
19. Rijsdijk, J., Wu, L., Perin, G., Picek, S.: Reinforcement learning for hyperparameter tuning in deep learning-based side-channel analysis. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2021**(3), 677–707 (2021). https://doi.org/10.46586/tches.v2021.i3.677-707, https://doi.org/10.46586/tches.v2021.i3.677-707
20. Smith, L.N.: Cyclical learning rates for training neural networks. In: 2017 IEEE Winter Conference on Applications of Computer Vision, WACV 2017, Santa Rosa, CA, USA, March 24-31, 2017. pp. 464–472. IEEE Computer Society (2017). https://doi.org/10.1109/WACV.2017.58, https://doi.org/10.1109/WACV.2017.58
21. Szegedy, C., Ioffe, S., Vanhoucke, V., Alemi, A.A.: Inception-v4, inception-resnet and the impact of residual connections on learning. In: Singh, S., Markovitch, S. (eds.) Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA. pp. 4278–4284. AAAI Press (2017), http://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14806
22. Targ, S., Almeida, D., Lyman, K.: Resnet in resnet: Generalizing residual architectures. arXiv preprint arXiv:1603.08029 (2016)
23. Won, Y.S., Hou, X., Jap, D., Breier, J., Bhasin, S.: Back to the basics: Seamless integration of side-channel pre-processing in deep neural networks. IEEE Transactions on Information Forensics and Security **16**, 3215–3227 (2021)
24. 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**(3), 147–168 (2020). https://doi.org/10.13154/tches.v2020.i3.147-168, https://doi.org/10.13154/tches.v2020.i3.147-168
25. Wu, L., Perin, G., Picek, S.: I choose you: Automated hyperparameter tuning for deep learning-based side-channel analysis. IACR Cryptol. ePrint Arch. p. 1293 (2020), https://eprint.iacr.org/2020/1293
26. Wu, L., Picek, S.: Remove some noise: On pre-processing of side-channel measurements with autoencoders. IACR Transactions on Cryptographic Hardware and Embedded Systems **2020**(4), 389–415 (Aug 2020). https://doi.org/10.13154/tches.v2020.i4.389-415, https://tches.iacr.org/index.php/TCHES/article/view/8688
27. Yarom, Y., Falkner, K.: Flush+ reload: A high resolution, low noise, l3 cache side-channel attack. In: 23rd {USENIX} Security Symposium ({USENIX} Security 14). pp. 719–732 (2014)
28. Zaid, G., Bossuet, L., Habrard, A., Venelli, A.: Methodology for efficient CNN architectures in profiling attacks. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2020**(1), 1–36 (2020). https://doi.org/10.13154/tches.v2020.i1.1-36, https://doi.org/10.13154/tches.v2020.i1.1-36

29. Zhou, Y., Standaert, F.: Deep learning mitigates but does not annihilate the need of aligned traces and a generalized resnet model for side-channel attacks. J. Cryptogr. Eng. **10**(1), 85–95 (2020). https://doi.org/10.1007/s13389-019-00209-3, `https://doi.org/10.1007/s13389-019-00209-3`

A Deep Learning

Deep learning (DL) is a form of machine learning (ML) where the algorithm to learn the function is a neural network of interconnected layers. Such DL algorithms are generally (in the context of SCA) used to perform classification in a *supervised learning* setting. Supervised learning is a setting where a set of already labeled data is taken, and a model is created from this labeled data to predict the labels for data that have not been used before [4].

As described above, DL networks are trained using a set of training data that have been assigned classification labels. The training lasts for a number of *epochs*. One epoch is equivalent to processing all training data once (forward and backward pass with the backpropagation algorithm). The data is divided into *batches*, and each batch is utilized in one iteration. When the network has generated predictions for a batch, the models' predictions are compared to the actual labels. This comparison is made by computing a *loss function*. Various loss functions are used to measure the errors the networks are making during classification. The goal during training is then to minimize the loss of the network. This is accomplished by computing the *gradient* of the loss function concerning the weights in the network. Then, the weights are updated according to this gradient by an amount scaled with the *learning rate*. Several methods exist for updating the weights, and we refer to these as *optimizers*.