On the use of ResNet architectures for Side-Channel Analysis

S. Karayalçın

On the use of ResNet architectures for Side-Channel Analysis

by

S. Karayalçın

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Friday May 13, 2022 at 10:45 AM.

Student number:	4598539	
Project duration:	September 1, 2021 – May 13, 2022	
Thesis committee:	Dr. S. Picek,	TU Delft, supervisor
	Prof. dr. ir. I. Lagendijk ,	TU Delft
	Dr. S. Roos,	TU Delft

TU Delft

Preface

Before you lies the product of about nine months worth of coffee, procrastination, and an occasional bit of research, I am proud to present this thesis. Still, before we get into it, I should express my wholehearted thanks to some people who have made it possible.

Firstly, I would like to thank Stjepan. From suggesting a great topic to the weekly meetings that made sure I kept making steady progress. Without your wonderful supervision, completing this thesis would have been more complicated.

Second, I would like to thank Guilherme, Lichao, and Wolfgang for taking the time in your busy schedules to provide feedback which helped significantly improve the quality of this work.

Finally, I would like to thank all of my family and friends for supporting me during this thesis and during my entire time studying in Delft. Without all of you, this would not have been possible.

S. Karayalçın
Rotterdam, April 2022

Abstract

Some of the most prominent types of attacks against modern cryptographic implementations are side-channel attacks. These attacks leverage some unintended, often physical, leakage of the implementation to retrieve secret information. In recent times, a large part of the focus of side-channel research has been on deep learning methods. These methods operate in a profiled setting where a model is learned based on a copy of the device that is being attacked. This model is subsequently used to create significantly more potent attacks against the target. Attacks using deep learning methods can often defeat even implementations protected with countermeasures, but as implementations become more protected, novel methods are required to successfully generate attacks. Recently, residual neural networks have been used for side-channel attacks, and these networks show promising attacking performance. However, these novel networks are relatively limited, and a more thorough investigation into the construction of residual networks in the side-channel context is required. Our contribution is a more thorough investigation into the construction of these residual architectures. We explore several important factors to the construction of these models and generate insights into various methods for this construction. The resulting architectures we find show attacking performance that is competitive with the state-of-the-art methods across various data sets and feature selection scenarios.

Contents

List of Figures

List of Tables

1 Introduction

The security of almost every interaction in cyberspace relies on the security of the encryption algorithms that ensure data is private. These encryption algorithms are used everywhere, from keeping data sent across the internet secure to authenticating payments on low-powered devices. These algorithms, like the AES and RSA, are theoretically secure. The secret keys that the algorithms use cannot be derived from only the inputs and outputs of the algorithms. However, the implementations of these algorithms do not always have the same guarantees of security.

Therefore, attacks based on the implementations are separate concerns for security evaluators. Several types of these attacks exist. In this thesis, we will focus on side-channel analysis. Side-channel analysis is a non-invasive attack that focuses on extracting leaked information during the execution of the encryption. These leakages can be very varied. Examples include power consumption [21], electromagnetic emanation [3], sound [2], or cache-timings [43]. In this work, we mainly focus on side channels that use traces of electromagnetic emanation during the encryption as this corresponds to the data sets that are commonly used [3].

There are several settings for side-channel analysis. The non-profiled setting aims to find some statistical connection between the traces and the sensitive values generated by a key. Attacks like Correlation Power Analysis [4] and Differential Power analysis [21] are some of the most famous examples of this. In the profiled setting, there is a very different threat model. In this case, an adversary has access and complete control over a copy of the device that is under attack. Having access to the secret keys allows the adversary to build a model of the device's behavior. This model can then be used for significantly more powerful attacks and can break protected targets. Profiled side-channel attacks were initially introduced with the Template Attack (TA) [6] and have since been a large part of the research surrounding SCA.

Several types of countermeasures against these attacks exist. These countermeasures are overall relatively effective at making attacks more difficult. However, the recent introduction of deep learning techniques in the field of profiled side-channel analysis has resulted in even protected targets being relatively trivial to break [24]. In recent times the field of research into profiled side-channel analysis has been focused on improving deep learning techniques. This has resulted in many works that emphasize attacking performance and the computational complexity of attacks. Because of this focus, the effectiveness of deeper neural network architectures has mainly been left unexplored.

Therefore, our work will aim to give a more thorough investigation into whether deeper architectures can result in improved attacking performance for side-channel analysis. The motivation for this is twofold. Deeper neural network architectures should be able to attack more complex data sets. As these networks have more representative power, they can generate more powerful models that could perhaps defeat more complex countermeasures. Additionally, deeper networks could prove to be more effective when traces with larger amounts of features are used [25].

Because our focus will be on deeper networks, we also choose to use Residual networks. These ResNets are

neural network architectures that include shortcut connections between network layers. These shortcut connections are added because updating the weights of earlier layers in deep networks becomes difficult as the gradient vanishes. These shortcut connections then allow the gradient to skip layers and allow the weights in these higher layers to be trained [13]. 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, ResNets have been used for side-channel analysis in some recent works and show promising results [17, 46].

To draw reasonable conclusions about the effectiveness of deeper ResNets, we must first evaluate how these networks should be constructed for side-channel analysis. This leads us to our first research question:

How should ResNets be constructed for the purpose of side-channel analysis?

Then we must evaluate when and where to use these ResNets. We can evaluate whether there are merits to using these architectures at all, and if so, in what context using these methods is better than the current state-of-the-art methods. This then leads us to our second research question:

In what contexts is the use of ResNets over more common CNN architectures useful?

The organization of this thesis is then as follows. First, in chapter 2 we will provide the required background information. Then, in chapter 3, we will give an overview of the related work. In chapters 4 and 5, the research questions will be answered. Finally, in chapter 6, we will give the overall conclusions of our work.

2
Background

2.1. Deep learning
Deep learning (DL) is a form of Machine learning (ML) where the algorithm to learn the function is a network of interconnected layers. Over recent years the DL field has undergone massive developments. With applications in many fields, such as image recognition [42], natural language processing [9], and speech recognition [10]. DL has proven to be a powerful method for learning various classification tasks. The driving forces behind the recent DL boom have been the advances in computational power and fast implementations of DL applications on Graphics Processing Units (GPUs), which have allowed for the training of very deep networks to be trained [5]. Additionally, the availability of larger amounts of data allowed for networks to be trained on larger training sets, which can greatly improve the classifying performance.

These DL algorithms are generally 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 it has not seen before [7]. An example is that a model can be used on pictures of handwritten numbers. The model is trained on already labeled pictures, and then it is tested by feeding it other pictures of handwritten numbers to test whether it can generalize to unseen data.

This section will provide an overview of how DL algorithms are used in this project. This overview will be given by first giving a brief overview of how DL models are trained. Subsequently, an overview of the main DL layers used for this project will be given. Then finally, an overview of some of the main types of architectures which will be used.

2.2. Training of DL networks
As described above, DL networks are trained using a set of training data assigned to classification labels. The training lasts for a number of epochs. Here, one epoch is equivalent to processing all training data once. The data is divided into batches to be processed. 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. These loss functions are used to measure the errors the networks are making during classification. The goal during training is then also to minimize the loss of the network. This is accomplished by computing the gradient of the loss function in relation to the weights in the network. Then the weights are updated per this gradient. The amount by which these updates are scaled is called the learning rate of the network.

Several methods for updating the weights according to the gradient, also called optimizers, have been used. The first of these, Stochastic Gradient Descent (SGD), was developed in 1951 and updates the weights in the direction of the gradient. Newer optimizers, like Adam [20], which we mainly use in this project, have adaptive learning rates and several other optimizations which result in faster convergence.

2.2.1. DL Layers
Here is a brief overview of the DL layers that will mainly be used.

Fully connected layer: Fully connected layers are made up of a number of neurons. These neurons are inspired by mathematical models of the human brain [33]. A neuron has n inputs. The trainable parameters of a neuron are the n weights that are assigned to each of the inputs, and the bias b that is added to the output. Thus, if a neuron with bias b and weights $\mathbf{w} = (w_1, w_2, ..., w_n)$ is fed an input vector $\mathbf{x} = (x_1, x_2, ..., x_n)$, then the output of this neuron is $y = \sum_{i=1}^{n} w_i x_i + b$. This output y is then fed through a non-linear activation function which results in the final output of one neuron. The output of the fully connected layer is then the list of outputs of all of the neurons in the layer. An image of one neuron can be seen in Figure 2.1

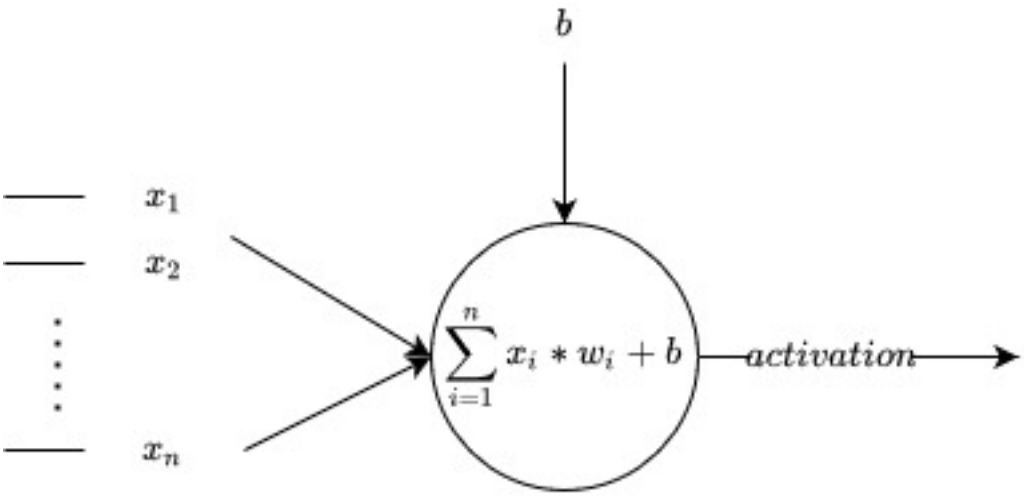


Figure 2.1: Schematic depicting a neuron.

Convolutional layer: Convolutional layers are composed of several matrices, also called kernels or filters. These kernels are slid across the input data to compute the output. At each position, the dot product of every filter is taken with each of these fields, and a new filter map is constructed composed of each of these dot products. In Figure 2.2, an example of this can be seen.

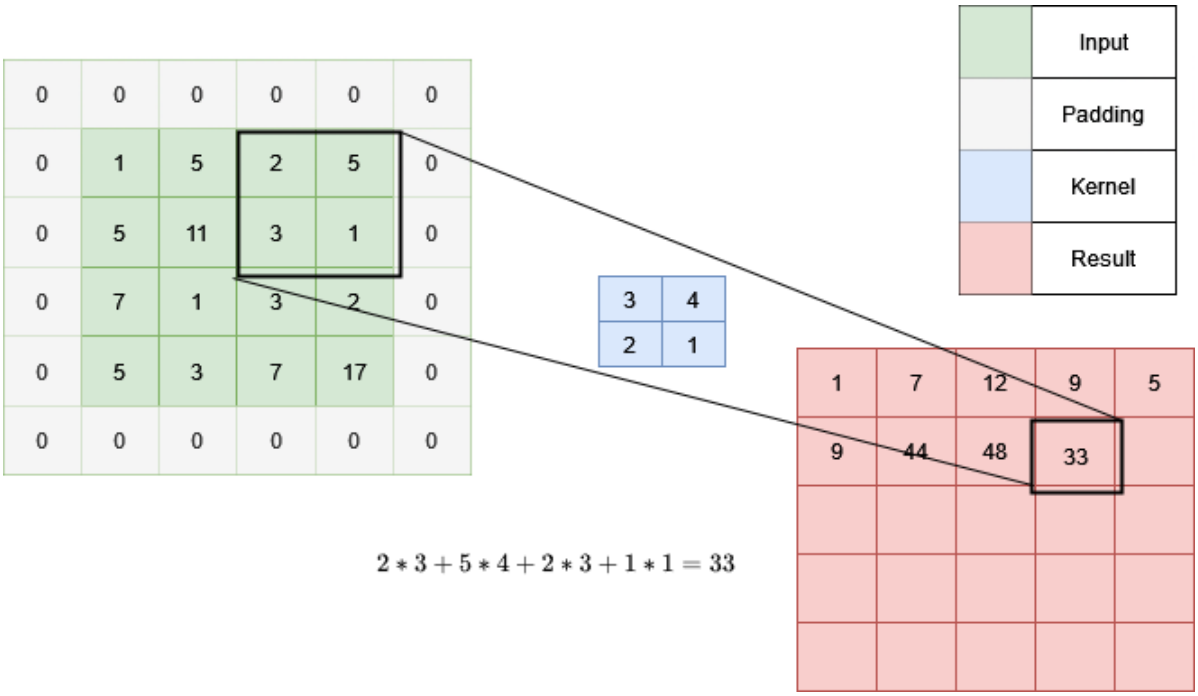


Figure 2.2: Schematic depicting operation of a convolutional layer.

Pooling layers: A pooling layer is used to reduce the size of its inputs. This reduction is made by dividing the input into small, possibly overlapping, fields and reducing each of these fields to a single value. Generally, the average value of a field (average pooling) or the maximum value in a field (max pooling) is taken. Global pooling layers take the average or maximum of the entire feature map for every channel and then return a 1-dimensional output of these averages/maximums.

Batch Normalization layer: Batch normalization layers are layers that are used to normalize the inputs of a new layer. This normalization is done per mini-batch and allows for faster and more stable convergence of the models.

2.2.2. DL Architectures

Here a brief description of the types of architectures used in this thesis is given.

2.2.3. Multilayer Perceptron

Multilayer Perceptrons (MLPs) are some of the most commonly used architectures used in DL [38]. MLPs consist of an input layer, several hidden layers, and an output layer. Each of these layers is a fully connected layer, which results in MLP architectures being relatively straightforward to implement and understand. The main variations in MLP architectures are the number of hidden layers and the number of neurons within these layers. Besides this, the activation function used in the hidden layers is an important hyper-parameter. In figure 2.3, an example of an MLP can be seen.

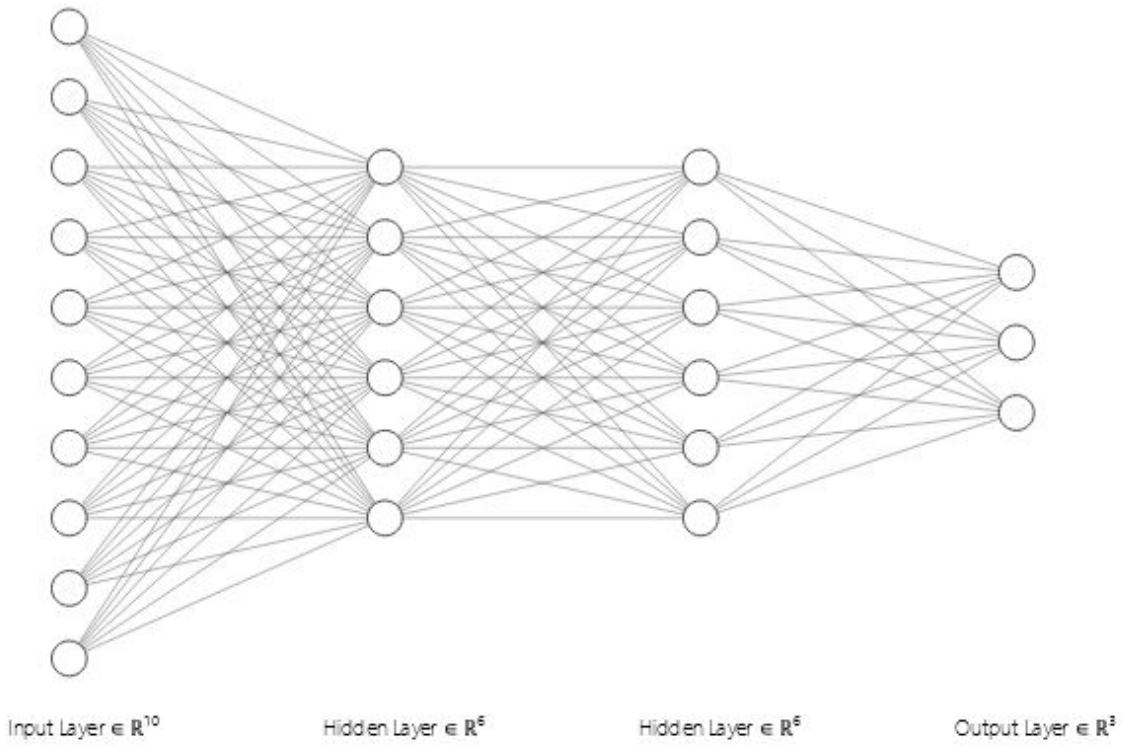


Figure 2.3: Figure depicting an MLP with two hidden layers of five neurons.

2.2.4. Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are another type of network that is very widely used [1]. CNNs are generally made up of three main types of layers: Convolutional-, pooling-, and fully-connected layers. The architectures consist of stacked convolutional layers and pooling layers to extract localized patterns from

the data. The first convolutional layers are used to extract very localized patterns. An example of this is that these layers can detect edges in images. Then later layers are used to extract patterns on a larger scale. For example, in image recognition tasks, this could combine the edges and lines into shapes. The pooling layers are used to reduce the dimensionality of the input features. This size reduction reduces the number of required parameters and the computational complexity. In CNNs, the first part of the network comprises some convolutional layers and then a pooling layer. This pattern is repeated several times. Finally, the fully connected layers are used in the final part of the network to combine the patterns and features extracted in the convolutional part of the network, which then results in the final classification result.

2.2.5. 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 where weights in the earlier layers of the network can not be efficiently updated using the gradient as the gradient is too small in these layers [37]. 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 some convolutional layers. Then a shortcut connection is added to the output of the residual block. This connection allows the gradient to also flow through the shortcut connection, which helps resolve the problems mentioned above of gradient vanishing. This shortcut connection is generally realized with just 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 2.4

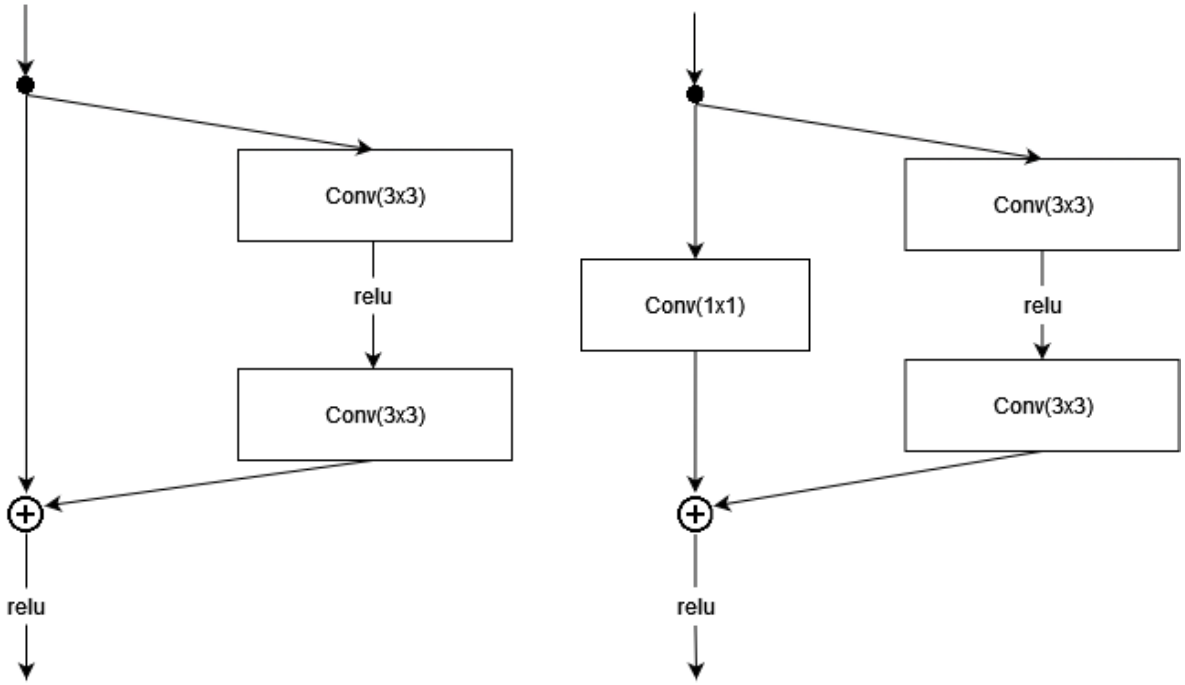


Figure 2.4: Typical structure of residual blocks.

2.3. Cryptography

In this section, a brief overview of cryptography will be given. Additionally, a description of the Advanced Encryption Standard (AES) will be provided as this is the cryptographic algorithm on which the attacks in this thesis will be focused.

2.3.1. Brief Overview of Cryptography

The field of cryptography is a field that is focused on achieving secure communication when someone could be eavesdropping on the information that is being communicated. To accomplish this, usually, the message that is to be sent, or the plaintext, is encrypted using the encryption key. This encryption results in the ciphertext, which is sent to the recipient. Then the recipient can decrypt the ciphertext using the decryption key. The main point here is that even if an attacker can eavesdrop on the ciphertext, they can not deduce any information about the plaintext or the keys used, which allows for confidential data transfer.

Various schemes exist to encrypt/decrypt data in this way. Public key ciphers, like the RSA cipher [32], are based on a system of public and private keys. In this context, a message encrypted with the public key can only be decrypted using the private key, and a message encrypted with the private key can only be decrypted using the public key. Another class of ciphers is the symmetric key ciphers. These ciphers use the same key for encryption and decryption.

2.3.2. Advanced Encryption Standard

The Advanced Encryption Standard (AES) refers to a group of ciphers that were proposed in 2001 for the NIST competition to design a new standard cipher [8]. The AES ciphers are all symmetric key block ciphers operating on 128-bit blocks. The ciphers operate using four operations executed repeatedly over a number of rounds. These operations are applied to the internal state, represented using a 4×4 matrix where each entry represents a byte. The four operations are as follows:

- AddRoundKey: In this operation, the key for the round is added to the internal state.
- SubBytes: In this operation, each entry of the internal state is transformed using a cryptographic Sbox.
- ShiftRows: In this operation, the rows of the internal state matrix are shifted.
- MixColumns: In this operation, the columns of the internal state matrix are mixed using a linear transformation.

Before these operations occur, the key is expanded to provide separate round keys for every round. The number of rounds is dependent on the size of the key that is being used. The number of rounds is 10, 12, or 14 for 128-,192-, or 256-bit keys, respectively.

2.4. Implementation Attacks

For most cryptographic ciphers used today, no information about the key can be deduced using only input-output behavior. For example, some of the best-known attacks against AES result in attacks with time complexity $2^{126.01}$ [36]for full key recovery, compared to 2^{128} which results from enumerating all possible keys. However, implementations of these ciphers are not necessarily as secure. An entire class of implementation attacks aims to exploit details about the implementation of these ciphers to attack them successfully. These implementation attacks can take different forms. Active attacks, like fault injection, aim to trigger some faulty behavior in the processor of the algorithm [15]. These faults are then exploited to result in successful key recovery attacks [11]. Passive attacks, or side-channel attacks (SCA), on the other hand, only observe some leakage of the target device. Examples of these leakages are sound [2], cache-timings [43], or power consumption [21].

2.4.1. Side-channel Analysis

Ever since first being proposed by Kocher et al. [21], side-channel attacks using power traces have seen a large number of works devoted to them. Various attacks, like Differential Power Analysis (DPA) [21] and Correlation Power Analysis (CPA) [4], have been developed. These power attacks rely on the fact that writing some value to a register or memory during execution requires power. This power usage means that information is leaked about this value by analyzing the device's power usage. Generally, these attacks only attack one of the bytes in the full key. This technique results in brute-force attacks on these key bytes as they only consist of $2^8 = 256$ possible values. If every sub-key can then be successfully attacked, combining the resulting guesses will make the recovery of the full-key possible. Often these attacks make use of a leakage model that is dependent on the targeted key byte. The effectiveness of these attacks can depend heavily on whether the chosen leakage model accurately reflects the leakage of the device that is being attacked. Commonly used leakage models are

the Identity- (ID), the Hamming weight (HW), and the Hamming Distance (HD) leakage models. For these, either the direct value of the leaky operation is taken or the number of ones in the binary representation of this value.

The setting that this project is focused on is the profiled side-channel analysis. Profiled attacks are generally much more potent than their non-profiled counterparts as the adversary is assumed to have access, and complete control, to a copy of the device they are attacking. The main workflow for profiled SCA is to take a large number of measurements from the copied device and create a model for its behavior. The creation of this model is generally done by labeling all measurements using the chosen leakage model. This labeling is possible as the adversary is assumed to have access to the key on the copied device. Then, several measurements are taken from the device that is being attacked. Then a set of labels for these measurements is generated using each of the possible values for the key-bytes, and for each, the log-likelihood according to the profile is computed. In cases of a successful attack, the correct key-byte will have the highest log-likelihood.

Creating these profiles can be seen as a supervised learning task as some number of measurements is collected, labeled, and then used to learn a model to predict new measurements. Over the past years, ML and DL approaches have become prevalent. Specifically, the DL approach has led to impressive results, being able to successfully attack data sets that are protected by various countermeasures [12].

2.4.2. SCA Metrics

As discussed before, ML approaches for profiled SCA have proven very successful. However, accuracy and loss, metrics that are often used in standard classification tasks, are not necessarily great indicators of how successful an attack will be [30]. While excellent accuracy scores or loss values indicate a successful attack, this is not required. Models that perform slightly better than random guessing can also generate successful attacks. Therefore, models in profiled SCA are usually evaluated on metrics related to the correct key's rank. Here, the rank of the correct key is the number of key candidates that have 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 real-world 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 three main metrics that are used are:

- Success rate: Success rate (SR) is the fraction of attacks that result in the correct key having key rank 1. This is defined as $SR(N_a) = Pr[g_{S_a}(k^*) = 1]$. Here $Pr[x]$ is the probability of x occurring. Generally, we estimate this over 100 attacks.
- 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. Generally, we estimate this over 100 attacks.
- NoT: The NoT (Number of traces) metric refers to the number of measurements that are required to reduce the GE to a value lower than two. This is defined as $min\{|N_a|\}$ for which $GE(N_a) < 2$. The NoT metric estimates how many measurements are required to recover the key.

2.4.3. Countermeasures

Countermeasures have been developed to break the statistical connection between the sensitive intermediate values and the traces. There are two main classes of countermeasures.

Masking countermeasures are meant to ensure the information about the sensitive value is spread out over various intermediate computations. The implementation of masking countermeasures is often done by adding or multiplying parts of the sensitive values with random masks. First-order masking schemes only have one mask, and higher-order schemes have more parts to the mask, ensuring attacks have to learn to predict all of these shares. An example of a masked substitution operation can be: $SBox[p_i \oplus k_i] \oplus r_{out}$. Here $\oplus$ is an xor operation. Computing the Sbox in this way makes sure that the sensitive value $SBox[p_i \oplus k_i]$ is not directly used and therefore does not leak directly. To determine this value an attacker now has to retrieve both the leaked $SBox[p_i \oplus k_i] \oplus r_{out}$ and the mask value r_{out}.

Hiding countermeasures, on the other hand, try to hide the signal. Hiding the signal can be done in the amplitude domain by adding noise to the signal, resulting in noisier measurements, which increases the difficulty of the attacks. Hiding countermeasures can also be done in the temporal domain, ensuring that leakages of sensitive values do not always occur at the same time during the computation and are therefore harder to locate in the traces. An example of this is introducing delays that take a random amount of time, at random moments, during the computation.

These countermeasures make attacks more complicated and can provide necessary resistance against side-channel attacks, but they also come at a cost. This cost can be a performance cost; random delays during computation make the computation take longer. It can also be an implementation cost as adding additional operations during the computation makes the circuits more complicated and expensive. These costs result in a need to find minimal countermeasures that still provide adequate security.

2.4.4. Data Sets

ASCAD Database

The ASCAD database [3] provides data sets on a protected implementation of AES. The sensitive value that is attacked for these implementations is generally the output of the first SBox. The ASCAD database provides traces measured on an implementation of AES for the ATMega8515. This implementation was protected with a first-order masking scheme around the SBox which can be described as:

$$SBox[p_i \oplus k_i] \oplus r_{out}$$

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 data set are provided. The first has 50,000 training traces and 10,000 attacking traces. All of these traces were measured with the same key and random plaintexts. These traces are 100,000 samples 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 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 to facilitate attacks. These 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 for comparisons of attacks. We refer to this version as **ASCADr**.

AES_HD

The **AES_HD** data set provides an unprotected hardware implementation of AES. The data set was measured on a Xilinx Virtex-5 FPGA. Overall, the data set contains 100,000 traces with 1250 features each. As the implementation is on hardware, there is significantly more noise present than is the case for the **ASCAD** data sets. Another difference is that a very 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 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. Additionally, we use the hamming distance of this resulting operation.

3

Related Work

In recent times the research around profiled SCA has evolved from standard machine learning techniques [14] to the deep learning domain [3, 19, 24]. The focus of recent works has been chiefly on CNN architectures [3, 19, 44], but recently ResNets have been used and show promising performance [17, 46].

3.1. Model Optimization in SCA

Ever since CNNs were first used for SCA [24], a large number of works have investigated improvements that could be made to deep learning models for SCA. Benadjila et al. [3] introduce the **ASCADf** data set to benchmark DL methods in SCA. They also test several types of DL architectures and find that VGG-like architectures provide the best performance for SCA.

The work of Kim et al. [19] looked at incorporating lessons learned in other domains which have similar, 1-dimensional data structures. They find that the VGG-like architectures used in speech recognition can provide top performance for SCA. Additionally, Kim et al. explore methods for adding artificial noise to training data to regularize the neural networks, which provides additional performance benefits.

Then, Zaid et al. [44] propose a methodology for creating minimal CNN architectures for specific data sets. These minimal CNNs result in top performance across several of the most common public data sets in SCA. In addition to the performance improvement these architectures provide, they are also significantly less computationally intensive to train than the architectures used previously.

The use of these architectures resulted in a need to find optimal hyper-parameter configurations. The effects of using various optimizers [27], loss functions [18], weight-initializers [22], and pooling layer configurations [40] were investigated. Furthermore, several methods for automatic optimization hyper-parameters were proposed. Using the previously mentioned investigations and model construction guidelines from [44], the search space for random search and grid search was limited greatly. The use of Bayesian Optimization for choosing hyper-parameters has also proven to be an effective method for finding hyper-parameter configurations that have strong performance [41]. Another option for searching for the best architectures and hyper-parameters is reinforcement learning [31]. The use of ensembles of randomly configured models has also been shown to have excellent results and is less reliant on finding optimal hyper-parameters [28]. These methods train larger amounts of small models with varying hyper-parameters and then either take the best performing of these models in the case of hyper-parameter optimization or combine the top-performing models into a larger ensemble.

These works that focus on automatic hyper-parameter tuning often rely on running a considerable amount of models to find the optimal candidate. These approaches are not always feasible when larger networks are used, as the time required to train the models will become prohibitive.

Other works in the field of SCA have focused on tackling issues that arise in profiling SCA. To tackle the class imbalances that arise when using the Hamming Weight leakage model Picek et al. [30] use SMOTE to balance

classes. Zhang et al. [45] proposed a novel evaluation metric that also serves as a loss function. These methods result in better performance than other approaches when using the HW leakage model.

3.2. Larger Amounts of Features

One of the places where larger architectures could be useful is when traces with a larger amount of features are used. The work of Lu et al. [23] looks at scenarios where the raw measurements of software implementations of AES are used. In this scenario, no pre-processing is applied, and the raw traces, consisting of hundreds of thousands or even millions of features, are fed directly to the DL models. Lu et al. use very deep (for SCA) networks that utilize attention mechanisms, and these networks result in great attacking performance on the tested data sets.

Perin et al. [29] also investigate the effects of using larger amounts of features. They test these effects by using larger amounts of features around the pre-selected windows for **ASCADf** and **ASCADr**, and also subsampling the raw traces for both of these data sets. They use random search to optimize the architectures and find that very small architecture can still yield very strong performance in scenarios with large amounts of features.

Finally, Masure et al. [26] attack data sets that have greater numbers of features than is common. Masure et al. attack an implementation of AES that uses code polymorphism as a countermeasure. Because of this, the window in which sensitive operations could occur becomes very large, which results in large amounts of features. Masure et al. developed adapted architectures from Kim et al. [19] to deal with this increased amount of features.

3.3. ResNets in SCA

Recently, some works have explored the use of ResNets in SCA. Zhou et al. [46] investigate the effects of misalignment of traces on the performance of DL methods. This work presents a ResNet architecture with promising performance on the ASCADF data set. However, the presentation of this architecture is not entirely clear, and for most of the results, proprietary data sets have been used. As such, reproducing the presented results is not straightforward, and how is unclear.

Another work that utilizes a ResNet architecture is the work of Jin et al. [17]. Here, Jin et al. propose using an attention mechanism that aims to make the network focus on the important features. The achieved results are similar to other state-of-the-art results, but the motivation for the use, and construction of the ResNet, is fairly limited.

Finally, Masure et al. [25] use a ResNet architecture to attack the new ASCADv2 data set. In this work, the attacks are executed with some knowledge of the masks during profiling, and a multi-label approach is used to limit the time required to train separate networks for each sub-key. The argument for using a ResNet, in this case, is that the amount of features is too large for conventional CNNs to work. The results here are hard to compare to the state-of-the-art as no other DL attacks against the data set have been published.

3.4. Research Questions

In recent times, the usage of ResNets for SCA has been becoming an interesting new research direction. However, investigations into the construction of architectures have not been conducted as far as we can find. This problem is made more apparent because the automatic hyper-parameter searching techniques that have been recently gaining popularity in the SCA field are not necessarily as feasible for ResNets as training large amounts of ResNets takes considerable computational resources. Additionally, clear investigations on whether to use ResNets over conventional CNNs and, if so, in what circumstances, have not yet been done.

This then leads us to our research questions: How should ResNets be constructed for the purpose of side-

channel analysis?

We then decompose this research question into three sub-questions:

- How should the residual blocks be constructed in the context of side-channel analysis?
- What number of residual blocks should be used for specific data sets in side-channel analysis?
- How do the models we arrive at compare to the ResNet models in other recent works that use ResNets?

Then our second research question, which aims to address the question of when the use of a ResNet architecture is advisable over the more commonly used CNN architectures:

In what contexts is the use of ResNets over more common CNN architectures useful? We then split this into two sub-questions:

- How does the performance of our ResNets compare to the state-of-the-art for some common public data sets?
- Can ResNets be used to create deeper networks that allow attacks on raw traces of software implementations that have very high numbers of features?

4

Construction of ResNets for SCA

In this chapter, we will explore several different ResNets in the context of SCA. The goal of this is to be able to give general recommendations on the construction of ResNets for SCA. To generate these recommendations, we will first look at several different ways of constructing residual blocks. After gaining insights into how residual blocks should be constructed for SCA, we will explore how deep these ResNets should be. Finally, we will compare the ResNets that we arrive at with the ResNets that are used in recent works [17, 46].

4.1. Motivation

The primary motivation for exploring the construction of ResNets in SCA is that there have not been any thorough investigations into how to construct ResNets for SCA in the literature. As our goal is to answer whether using ResNets for SCA is a viable alternative to the smaller CNN architectures currently used, we need to provide architectures that represent how ResNets can perform in the SCA domain. Extensive experimentation is necessary here as the previous works using ResNets in SCA do not explain why the architectures they used had been chosen.

To be able to develop these architectures, we focus on two essential factors. Firstly, the residual block construction is a central part of a ResNet. These blocks form the basis for how a ResNet functions, but there are no clear indications of how to construct these for SCA. The three works using ResNets for SCA all approach this factor differently. Masure et al. [25] use a block that has two convolutional layers and uses convolutional stride to reduce the feature map size. Zhou et al. [46] also use convolutional stride, but they use three convolutional layers and include batch normalization. Finally, Jin et al. [17] use a block with two convolutional layers and use pooling layers to reduce the feature map size. Jin et al. also use a novel attention mechanism in their blocks. Because of these different approaches and the limited motivation that underlies each, making well-motivated choices regarding how a residual block should look for SCA is pivotal.

Determining how deep a network should be is also vital. While the need for residual architectures is only really present when networks become deeper, the works in SCA previously [17, 46] do not utilize very deep architectures. Overall, the state-of-the-art architectures in SCA are relatively shallow [44], and it is, therefore, unclear whether we should limit the depth of our ResNets. To generate insights into this, we need to evaluate networks of varying depth experimentally. Conducting these experiments will allow us to conclude how deep these networks should be and whether it is even necessary to include residual connections if the added depth is not helpful.

In this chapter, we will aim to provide insights into how to construct ResNets for SCA effectively. We will execute extensive experiments to show that the resulting architectures are representative of the performance of ResNets for SCA. This will then allow us to compare the performance of ResNets to other state-of-the-art methods and draw conclusions about how useful ResNets can be for SCA in Chapter 5.

4.2. Residual Block Construction

To determine how residual blocks should be constructed for SCA, we will take several residual block constructions and test these on the **ASCADf**. We determine an initial set of residual block constructions based on recent works in SCA [17, 25, 46]. Additionally, we will look at the blocks inspired by Inception-Resnet [35]. All of the blocks we test here can be seen in Figure 4.1.

The architecture used for the tests is also significantly deeper than the usual architectures used in state-of-the-art SCA [31, 44]. We made this choice as one of the main motivations for using ResNets is that they can be made deeper than conventional CNN architectures without experiencing gradient vanishing issues.

4.2.1. Experimental Setup

The data set we use for the initial experiments is the **ASCADf** data set. We attack the first Sbox output of the second byte as this is the first protected byte. The features we use are the pre-selected window of 700 features.

We use the maximum number of residual blocks possible while still halving the feature map size in each block. This decision results in us using $\log_2(700)$, rounded down to 9 blocks, which amounts to 18, or 27, convolutional layers. We use stride 2 in either pooling or convolutional layers, and the kernel size is 11. The number of filters in the convolutional layer of the i'th block is $min(2^{i-1}, 256)$. We use the Adam optimizer with a cyclic learning rate schedule [34] with a maximum learning rate of 1e-3. These hyper-parameters were chosen based on the state-of-the-art in SCA [3, 44]. We chose to do this as these hyper-parameters are presumed to be 'good enough', and we consider a search for optimal hyper-parameters out of scope for this work. An overview of the network that we use for these experiments can be seen in Figure 4.7.

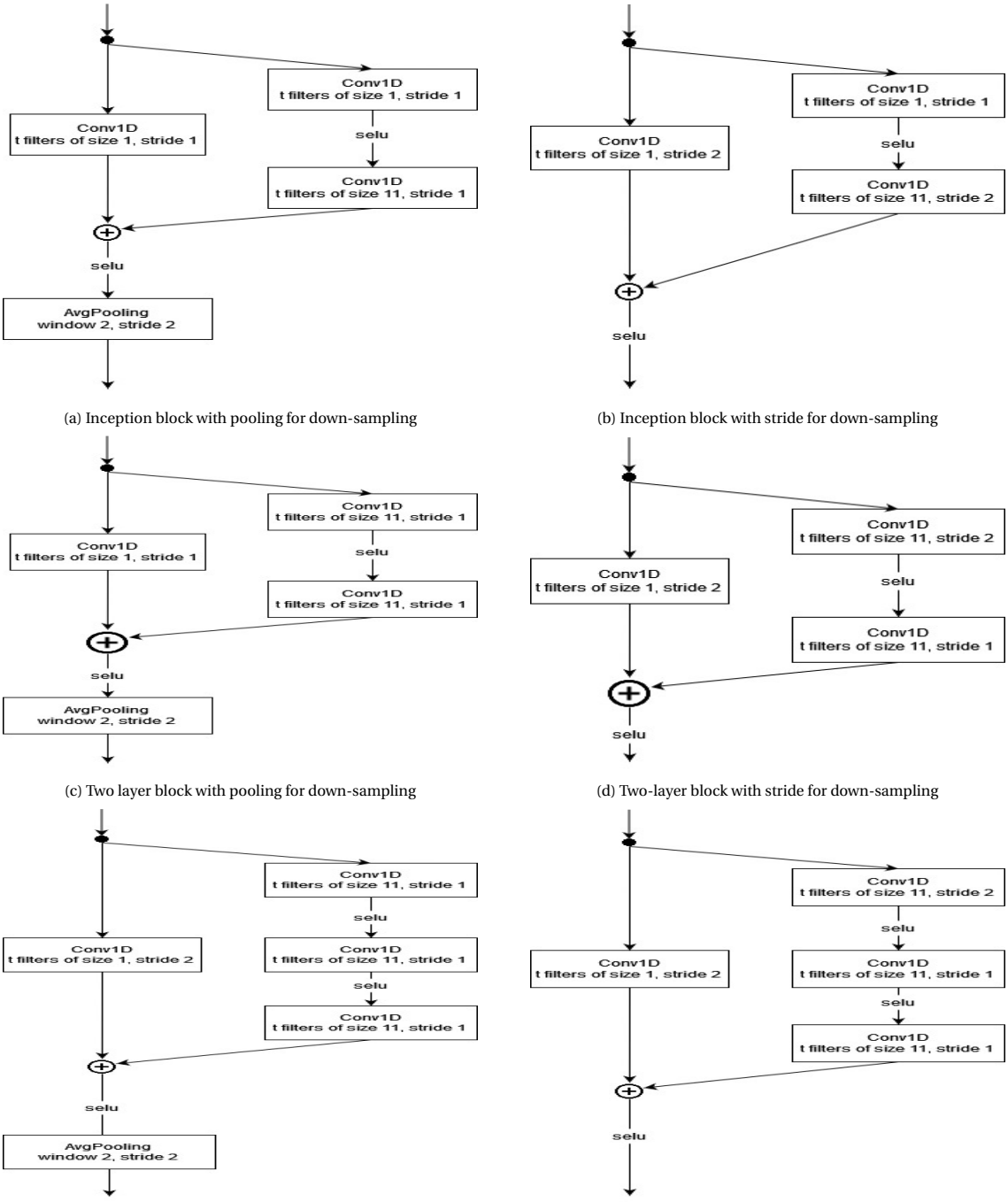
All networks were trained for 100 epochs with a batch size of 50. For training, 50,000 traces were used, 5,000 were used for validation, and 5,000 were saved as an attacking set. During all epochs, the validation loss and the number of traces required to get to $GE < 2$ on the validation set were tracked, and the models that had the best result on validation loss and the NOT metric were saved and used to execute attacks. Additionally, five models were trained for every configuration and all results presented in this chapter are averaged across these five runs. We average the results from several runs as there is a fair amount of variability in the performance of the models. Therefore, running the models only once was not deemed sufficient to get an accurate view of what does and what does not work well.

4.2.2. Results

The first experiments we ran were to determine what types of blocks should be used. The blocks that can be seen in Figure 4.1 were all tested according to several parameters. The final attacking performance, the attacking performance of the best model according to validation loss, and the evolution of both the validation loss and the number of traces to reach $GE < 2$ were tracked.

In Figure 4.2 the attacking performance of all the models can be seen. Here we can see that all models, except the three-layer model without pooling, can generate models that successfully attack the ASCAD data set. The attacking performance in the final model is also significantly worse than the performance of the best model, which means some over-fitting occurs. This can also be seen in Figure 4.3. Here we see that the performance of the models initially improves in both metrics. After several epochs, the performance then starts decreasing. This over-fitting occurs as the number of trainable parameters is fairly high for the data set as we chose to use deeper models to represent the targeted use case for ResNets better. We mitigate this by saving the model weights from the epoch that achieves the best validation loss.

The main distinction we see in performance is that the models using pooling for reducing the feature map size perform significantly better than their respective counterparts that reduce the dimensions using convolutional stride. This result is in line with previous works in the construction of CNNs for SCA where pooling layers were used [19, 44]. Additionally, it can be seen that the inception model and the two-layer model both perform better than the model using three convolutional layers in their residual blocks.



(a) Inception block with pooling for down-sampling

(b) Inception block with stride for down-sampling

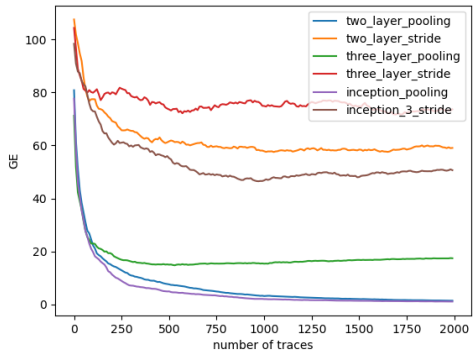
(c) Two layer block with pooling for down-sampling

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

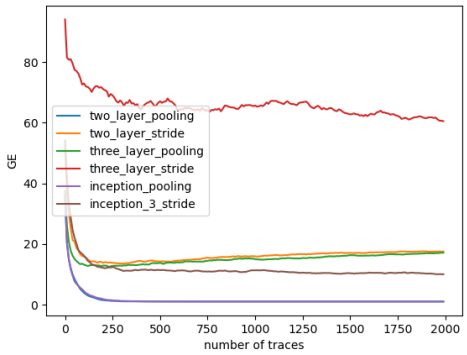
(e) Three-layer block with pooling for down-sampling

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

Figure 4.1: The six types of residual blocks used for testing

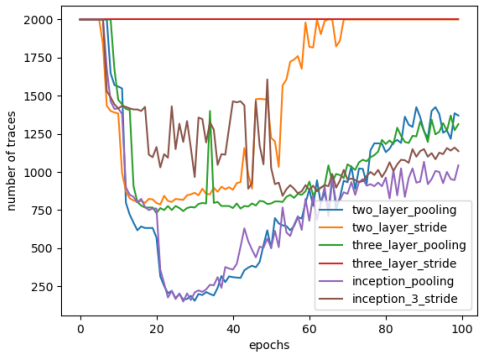


(a) Attack performance of the models after 100 epochs

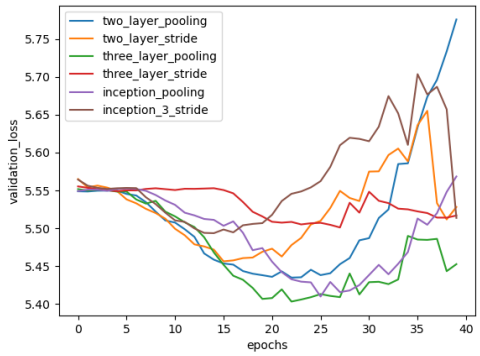


(b) Attack performance of models achieving best validation loss.

Figure 4.2: Attacking performance of both the final model and the model that had the best validation loss



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



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

Figure 4.3: Evolution of early stopping metrics across epochs

4.3. Pre-activation and Batch Normalization

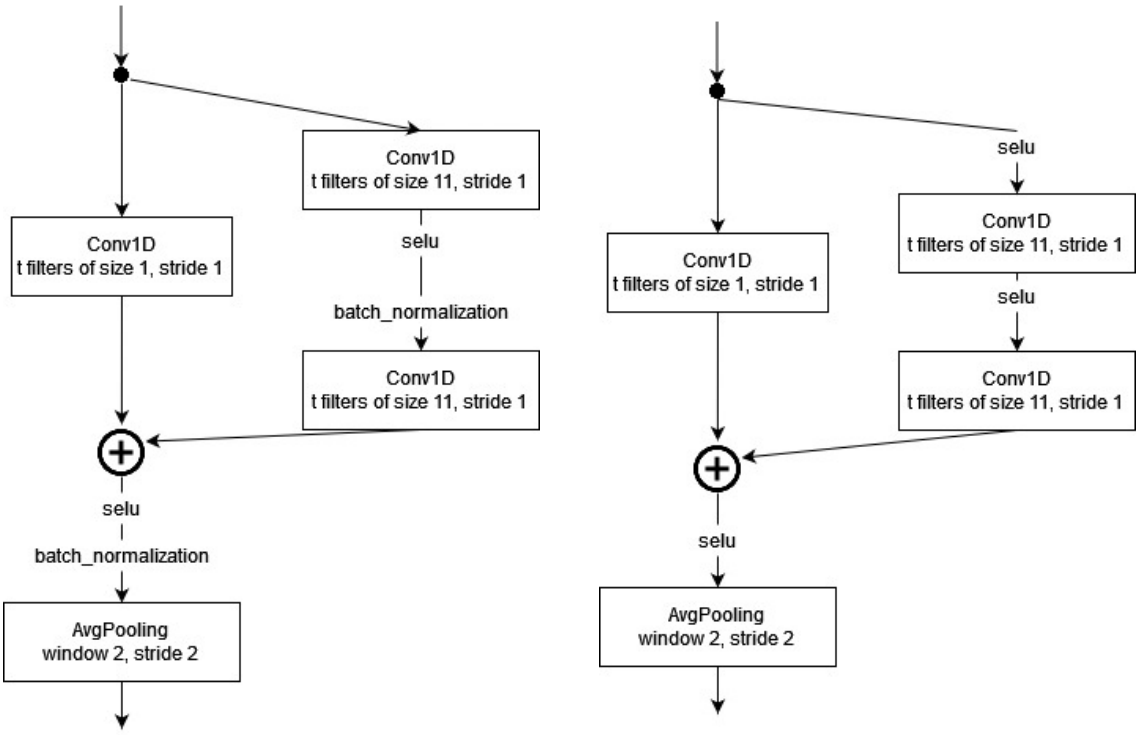
To better understand what types of constructions for residual blocks function best in the context of SCA, we also test the addition of some common mechanisms from image recognition. We limit the tests of these additions to the two residual blocks that function best in the tests described above: the inception block with pooling and the two-layer block with pooling. The mechanisms we test are pre-activation [13], where we add an activation function before the first convolutional layer of the residual block, and batch normalization layers [16], which we add after each activation layer. The resulting blocks can be seen in Figure 4.4. Here, the experimental setup is identical to the experimental setup for the various block constructions.

4.3.1. Results

As can be seen in Figure 4.5, the addition of pre-activation and batch normalization layers does not make a great difference in terms of attacking performance. All of the different models seem to perform approximately the same. Additionally, we see that the difference either of these additions makes in the speed of the convergence and in terms of the amount of over-fitting is fairly minimal in Figure 4.6. Here we see that, on average, all of these models show approximately the same behavior. They first improve the attacking performance quite rapidly in the first 20-30 epochs, and subsequently, the performance worsens.

4.4. Depth of Residual Networks

This section aims to answer our second sub-question: What number of residual blocks should be used for specific data sets in side-channel analysis? To answer this sub-question, we will investigate how different numbers of residual blocks work for attacking the **ASCADf** data set. These experiments will allow us to rec-



(a) Block with added batch normalization layer.

(b) Block with added pre-activation layer.

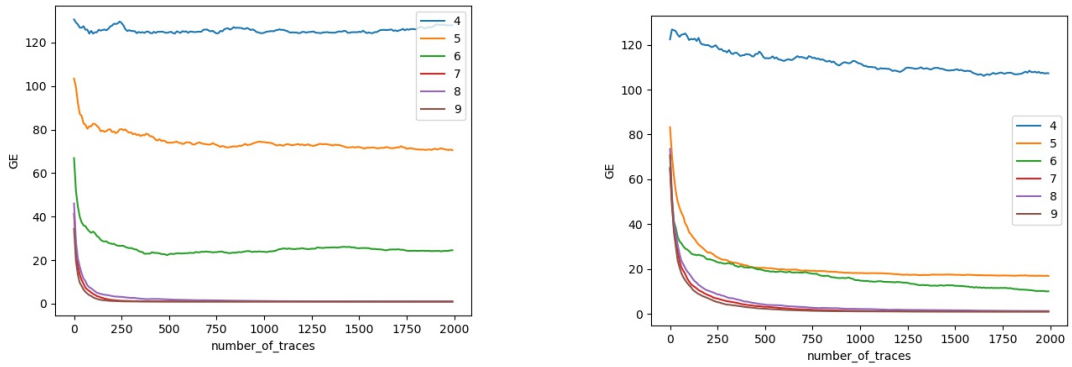
Figure 4.4: Residual blocks used for testing with additional layers.

ommend the number of residual blocks useful for ResNets in SCA.

4.4.1. Experimental Setup

The experimental setup is generally the same as the setup used to investigate what block construction was best. We use the two-layer block with batch normalization, and all hyper-parameters are the same. The main difference is that we limit the number of residual blocks, and in the last residual block, we use a GlobalAveragePooling layer instead of a standard AveragePooling layer. We made this choice to ensure that the feature map size is still reduced to similar sizes for the various depths.

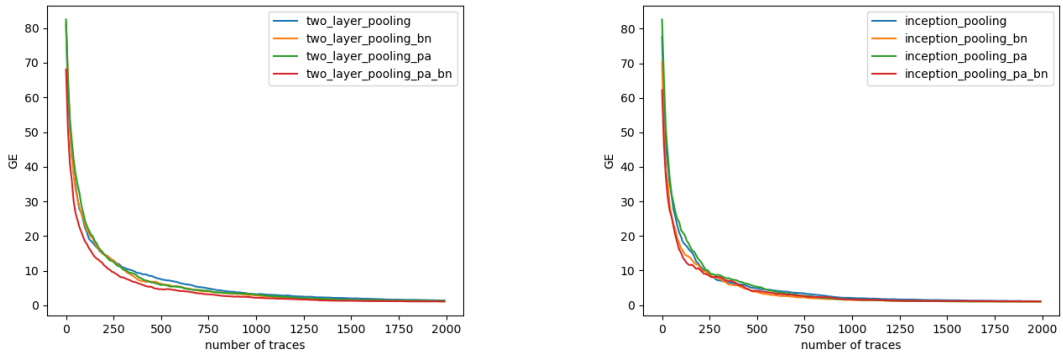
4.4.2. Results



(a) The GE results for networks of varying depths with weights from epoch with best validation loss.

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

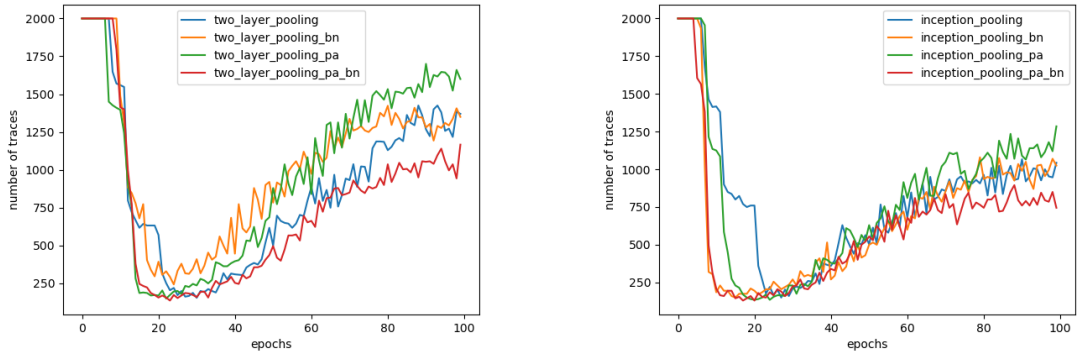
Figure 4.8: GE results for networks of varying depths.



(a) Attack performance of two-layer block

(b) Attack performance of Inception block

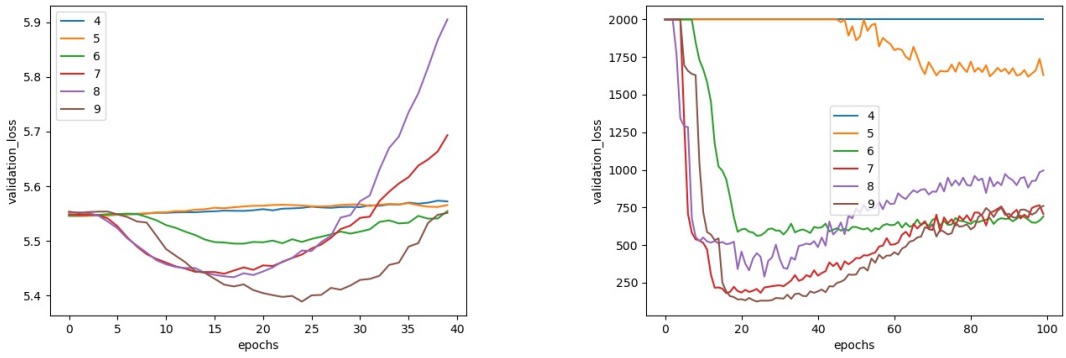
Figure 4.5: Attacking performance of both the final models with the addition of Pre-activation(pa) and Batch normalization(bn).



(a) Evolution of NOT metric during training of the two-layer block.

(b) Evolution of NOT metric during training of the inception block.

Figure 4.6: Evolution of early stopping metrics across epochs with the addition of Pre-activation(pa) and Batch normalization(bn).



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

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

Figure 4.9: Evolution of early stopping metrics across epochs for blocks of varying depths.

From the results in figure 4.8, we can see fairly clearly that too few residual blocks can result in the attacks becoming significantly less powerful. Models with fewer than seven residual blocks do not result in successful attacks consistently on the **ASCADf** data set. The models with only four residual bocks seem unable to improve over random guessing at all, while the models with five and six residual blocks sometimes seem to result in successful attacks. Then, the models with more than six residual blocks show approximately the

same performance when stopped early and for the final models. These models can consistently generate successful attacks with similar performance across the board.

From figure 4.9, we can also see that there is no significant difference between networks with seven through nine residual blocks. All of these networks seem to converge at similar epochs, and we note that there is no need to use more than seven residual blocks for this data set. In Figure 4.9a we do see that models with nine residual blocks result in slightly lower validation loss, but this does not result in improved attack performance.

4.5. Discussion

The main point that we can conclude from our results is that the construction of the residual block is an essential factor in terms of the attacking performance and the consistency of the attacks. We note that with deeper ResNets, it seems advisable to use pooling to reduce feature map size as opposed to convolutional stride, which is in line with the earlier findings around state-of-the-art CNNs. We postulate that this is because when convolutional stride is used, 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.

Additionally, we see that the use of more than two convolutional layers in a residual block harms the performance of ResNets for SCA. In further experiments, we see that both batch normalization and pre-activation do not seem to improve the performance of the networks. The training overhead induced by the addition of either of these layers leads us to conclude that it is preferable not to use them. We note that this limited difference may be due to the relative simplicity of the **ASCADf** data set, and we do not rule out that these techniques may be useful when considering better-protected data sets.

Furthermore, we find that making these networks deeper results in better attack performance. There is a limit, which seems to arise about two residual blocks from the maximum amount of residual blocks that can be used. This result differentiates our work from other recent works on architecture design, which focus on limiting the depth of the network and then finding optimal hyper-parameters [44]. It is also remarkable that the deeper networks perform significantly better than, the shallower architectures, as the hyper-parameter setups we employ are optimized for the smaller architectures of Zaid et al. [44]. This leads us to believe that there could still be significant improvements to the hyper-parameter setups of our networks which could result in even better performance.

Our results differ a fair amount from the other ResNets that have been used in SCA. The final architectures that we arrive at have significantly more residual blocks than both of the works that also attack the **ASCADf** data set [17, 46] which both use three. Our final residual block construction is similar to the one used by Jin et al. [17], but it does not include the attention mechanisms they used. Finally, the architecture presented by Masure et al. [25] is similar but uses convolutional stride to reduce the feature map size as opposed to pooling. In part, these differences can be attributed to varying data sets. However, as we tested all the residual blocks from these works, we postulate that perhaps even for these different data sets, our work can be beneficial. For example, the attacking results of Masure et al. [25] could probably be improved by using pooling layers over convolutional stride.

Another note is that we used a data set with a relatively small 700 feature window for the attack. We chose this window as it is a standard scenario for SCA DL attacks and limits the computational overhead during this more exploratory phase of our work. However, the architectures here are not necessarily being constructed for the much larger feature intervals that we will explore later. This limitation is somewhat mitigated by the fact that we do use deeper architectures. A more extensive exploration of larger feature selection windows can be found in Chapter 5.

In this chapter, we show that deeper ResNets can be effectively used for SCA. Additionally, we provide insights into the construction of these architectures. We show that using more than two convolutional layers per residual block negatively affects the attack performance. Furthermore, we see that using pooling to reduce the feature map size is better than using convolutional stride. 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 data set has x

features, using $\lfloor log_2(x) \rfloor - 2$ residual blocks seems advisable as using more results in some additional computational overhead while, in our experiments, not improving performance.

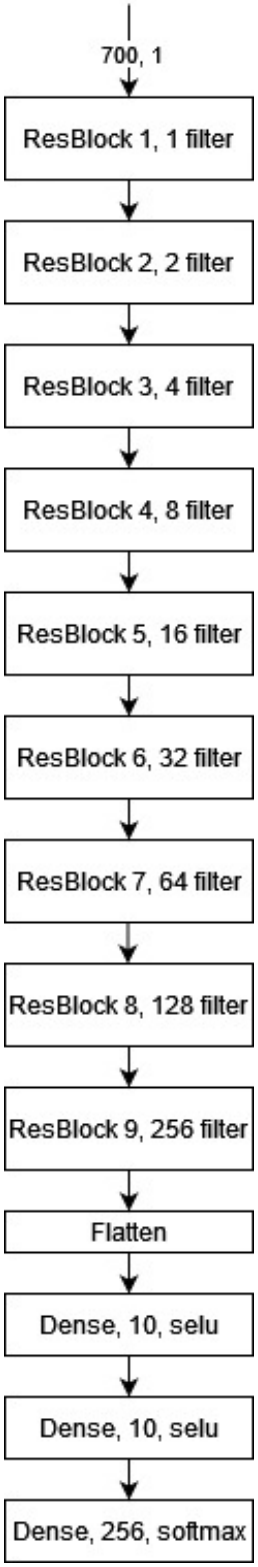


Figure 4.7: Network setup for tests of various residual blocks

5

Comparison with State-of-the-art

This chapter will answer our second research question: "In what contexts is the use of ResNets over more common CNN architectures useful?". We will first aim to answer this by comparing our ResNets with state-of-the-art CNNs on some standard public data sets. In addition to this, we will also compare them to the ResNets that were previously used for SCA. These results will give us insights into how good our ResNets are compared to other DL methods. Secondly, we will explore various feature selection scenarios as presented in [29] to see whether the ResNets architectures perform well when more features are used.

These experiments will let us see whether there are merits to using ResNets over CNN models in standard SCA scenarios. Additionally, the experiments in various feature selection scenarios will generate insights into the usability of ResNets when there are more features.

5.1. Motivation

The primary motivation for running various experiments compared to the state-of-the-art methods is that we need benchmarks to answer whether ResNets provide a viable alternative to other DL methods in SCA. We must first select what methods to compare to our networks to do this. As our networks are singular architectures, it makes sense to compare them to other state-of-the-art network architectures. However, in recent works in DL for SCA, the focus has not necessarily been on improving individual architectures but rather on finding automated methods that run a large number of models with varying hyper-parameter setups to find a model that performs best. To give a better overview of how our networks stack up to the architectures that have been optimized for specific data sets, we take state-of-the-art architectures from Zaid et al. [44] as a point of comparison for the **ASCADf** and the **AES_HD** data sets. For the **ASCADr** data set and the various feature selection scenario we explore for both **ASCAD** data sets, we chose to compare our results to the model search strategies, as in these scenarios, the top results are from model search strategies [29, 41] and ensemble models [28].

The motivation for which data sets we test against is also relevant. Firstly, we test our networks again on the **ASCADf** data set because it has been the primary benchmark for DL methods in SCA over the past years. As we developed our architectures with experimental results from this data set, we also test the larger **ASCADr** data set to make sure our networks are not specific to one data set. In both of these cases, we attack the same implementation. To further investigate the effects of differing amounts of profiling traces, we also do some experiments with data augmentation by adding gaussian noise to the traces, based on the work of Kim et al. [19], as this allows us to vary the amount of artificial traces for **ASCADf**. As this is a masked software implementation, and we focus on the Identity leakage model for both of these data sets, it was also interesting to see how our networks perform in a very different scenario. Therefore we provide some tests on the **AES_HD** data set. With all of these data sets and the various feature selection scenarios, we should be able to conclude how well these ResNets can perform for SCA.

5.2. Experimental Setup

The experimental setup for the **ASCAD** data sets is similar to the one used in the previous chapter. We use 50,000 and 200,000 profiling traces for **ASCADf** and **ASCADr** respectively. For both data sets, we use 5,000 traces for validation and 5,000 traces for attacking. As the main goal in this section is to compare, we also use the standard pre-selected windows of 700 and 1400 features. We run the training with 0.5, 1, and 2 times the standard 50,000 number of artificial traces for the data augmentation tests. For **ASCADf** we also attack the full 100,000 feature set. We do this by using all the 100,000 traces directly as in [23], but we also attack re-sampled versions of the data, as in [29], to limit the computational overhead somewhat. With this re-sampling, we reduce the number of features tenfold. We also do this re-sampled attack against the **ASCADr** data set. For the **AES_HD** data set, we use 50,000 traces for profiling and 12,500 traces for both validation and attacking. We use all 1,250 available features.

Our hyper-parameter setup for the **ASCAD** data sets is the same as in the previous chapter. Additionally, we save the model weights that result in the best validation loss during training. In addition to the experiments on the standard interval, 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 and are based on the work of Perin et al. [29]. The hyper-parameter setup for the **AES_HD** data set is also the same as the one used for the **ASCADf** data set other than the fact that we used a kernel size of 2 instead of 11, as this is a data set that is based on a hardware implementation.

The number of residual blocks that we used for each data set depended on the number of features. As we concluded in Chapter 5 we take $\lfloor log_2(\text{num\_features}) \rfloor - 2$ residual blocks for any particular scenario. In addition, as the **ASCADr** set is a lot larger than the **ASCADf** set and as such, we make the fully connected part larger. In this case, we use two dense layers with 200 neurons each instead of 10 neurons each.

Finally, to compare our models to the state-of-the-art, we take the best-performing model out of five training runs. We run five models as models are often inconsistent and do not always all converge, and in the works we are comparing against, the best case performance is often showcased.

5.3. Experimental Results

This subsection will present the results for the various data sets. We first look at both of the **ASCAD** data sets and various scenarios associated with these. Then, we look at the **AES_HD** data set.

ASCADf

On the standard, 700 feature interval of **ASCADf**, our attacking results are relatively good, as can be seen in Figure 5.1. In Table 5.1, we see that our results are significantly better than the results of some other ResNets that have been used for SCA. It should be noted here that the results of Zhou et al. show their ResNet as working, but when we train the presented model, it does not result in successful attacks. In comparison with state-of-the-art CNNs, our methods perform a fair bit worse. In addition, our ResNet models take significantly longer to train.

When we start using more features, the attacking results improve a lot. This improvement is because the sensitive values are leaked at several points which are outside the original interval, which can be seen in Figure 5.3. These additional leakages allow the model to learn to predict the intermediate values more accurately as it has additional leakages that it can use.

In Figure 5.2, we see that the addition of noisy features without too much additional information does impact the performance of the networks quite a bit. The number of traces required for successful attacks increases from 1,500 to 2,500 features. As additional leakages are included in the sets with more than 3,000 features, the attacking performance improves again. We see that at 3,500 features, the attacking performance is best. Then the additional features do not result in significantly worsened attack performance when more features are added after the 3,500 feature mark.

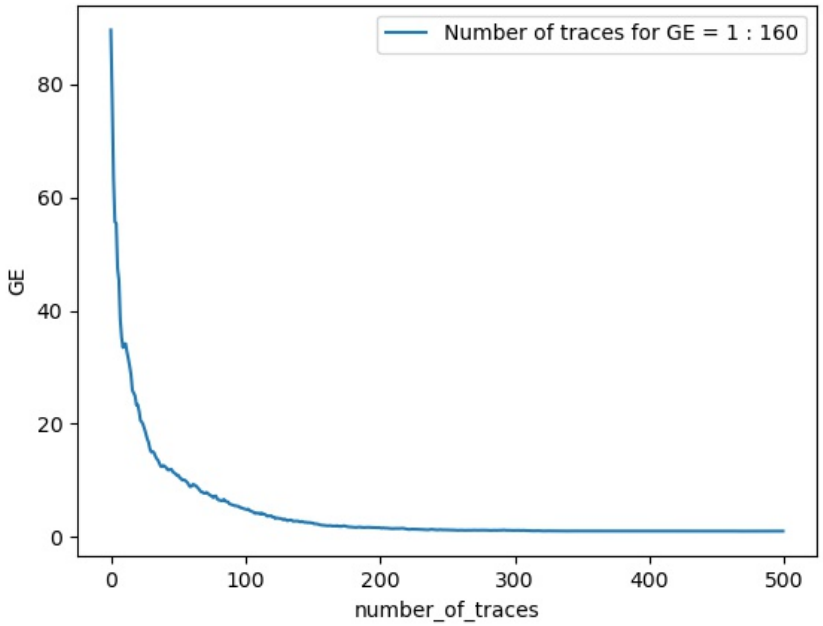
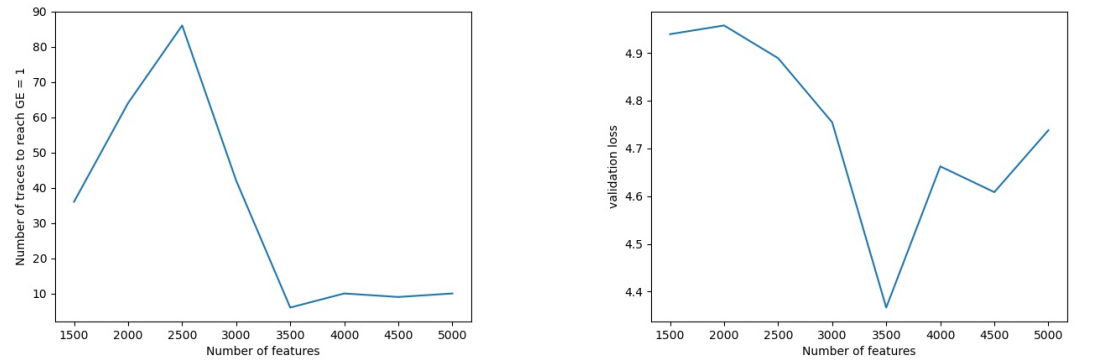


Figure 5.1: GE results for our ResNet on **ASCADf**

	Our Resnet	Zhou et al. [46]	Jin et al. [17]	State-of-the-art CNN [44]
Nr. of traces for GE = 1	160	>2,000	552	83

Table 5.1: Comparison of various different architectures on **ASCADf**

When we then compare our results to the state-of-the-art hyper-parameter search strategies from [29], we see that our attacking results are a bit worse for every amount of features. In Table 5.2 we see that the attacking performance is initially pretty similar at 1,500 features. When the amount of features is then increased, we see that the performance of our ResNets initially deteriorates while both the CNNs and the MLPs show improved performance. When we then hit 3,500 features, we see that the performance of all of the models is relatively similar. The attacking performance is best for all models when the set with 3,500 features is used.



(a) Best validation NoT across several feature selection scenarios

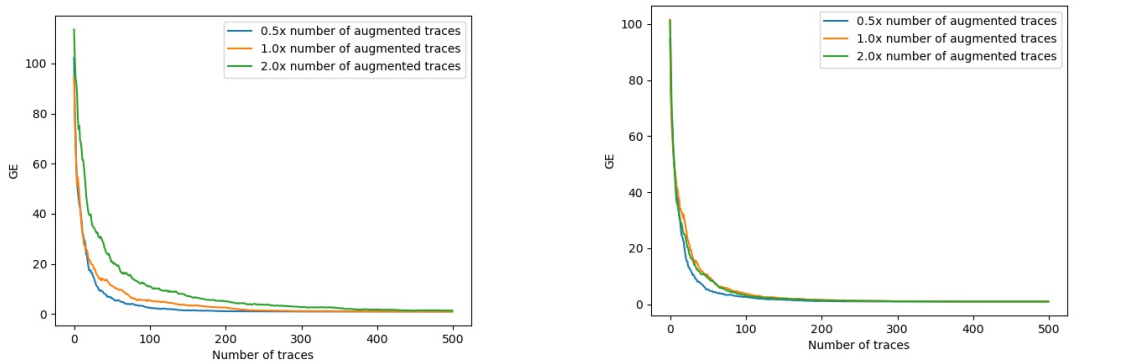
(b) Best validation loss across several feature selection scenarios

Figure 5.2: Best values of metrics across several feature selection scenarios for **ASCADf**

Nt. of features	Nr. of residual blocks	ResNet	CNN [29]	MLP [29]
700	7	160	80	58
1,500	8	36	≈ 40	≈ 30
2,000	8	64	≈ 35	≈ 20
2,500	9	86	≈ 25	≈ 15
3,000	9	42	≈ 10	≈ 10
3,500	9	6	4	4
4,000	9	10	≈ 5	≈ 5
4,500	10	9	≈ 5	≈ 5
5,000	10	10	≈ 5	≈ 5

Table 5.2: Comparing the number of traces required to reach GE = 1 at various feature selection scenarios to the results of Perin et al. [27] for **ASCADf**

Then, we look at the best models utilizing data augmentation in Figure 5.4. Models stopped early all have about the same attacking performance, but we see some more overfitting when more augmented traces are used. Additionally, the performance of models using the augmented traces is better than the original performance on the **ASCADf** data set, but the difference is minimal.



(a) Best final model attack performance.

(b) Best model performance of best model weights according to validation loss.

Figure 5.4: Performance of best models trained with augmented data.

Finally, when we attack the full feature set of **ASCADf** we see some more surprising results. For both the re-sampled and the raw traces we find that our networks cannot generate successful attacks. As can be seen in Figure 5.5, for both of these cases we see that our networks cannot improve over random guessing at all.

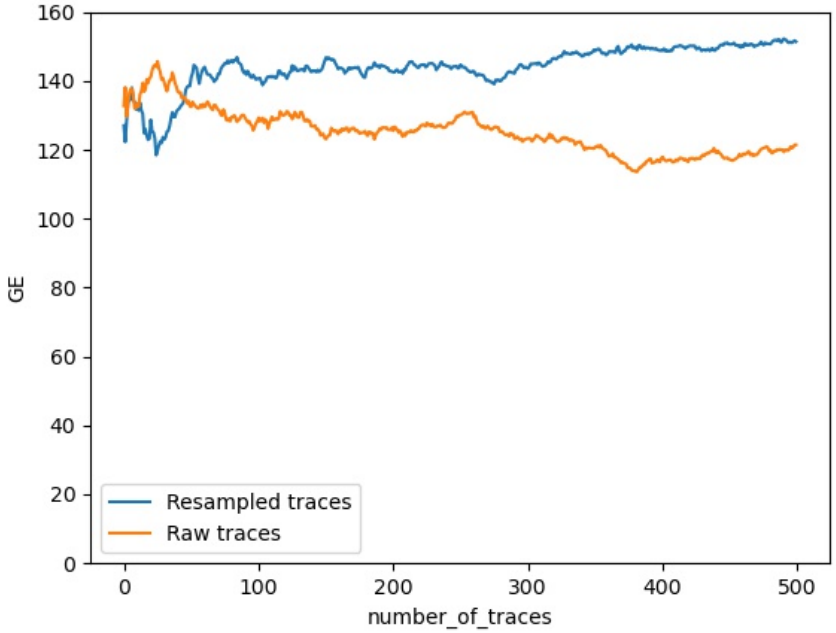


Figure 5.5: Attack performance of models attacking both raw and re-sampled traces.

ASCADr

When we then proceed to look at the **ASCADr** data set we see some similar results. Firstly, when we look at the GE results of our network we see that the attack performance is quite good. In Figure 5.6 we see that our networks can successfully attack the data set in 47 traces. As can be seen in Table 5.3 this is in line with the state-of-the-art methods in SCA. The ensemble methods used in [28] outperform our network, but the model search strategies employed in [29] perform a bit worse.

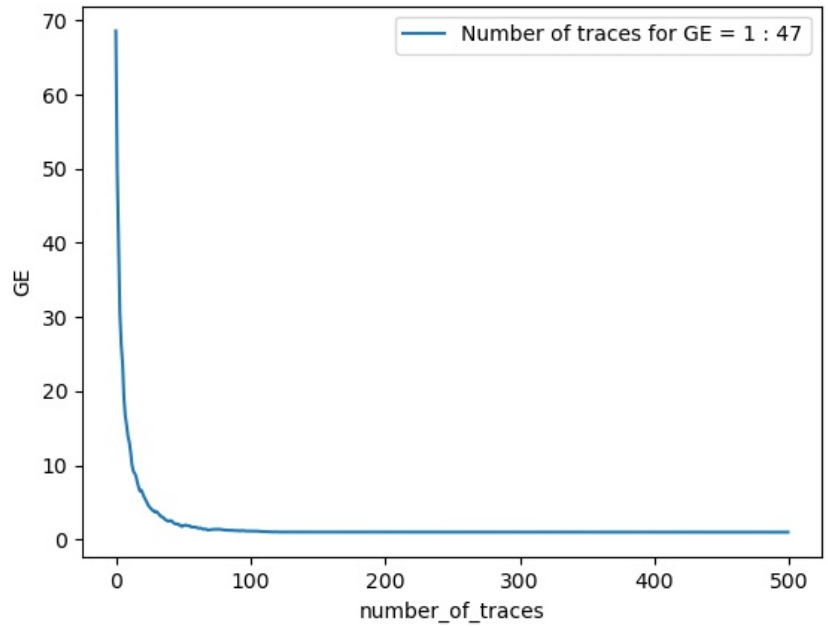


Figure 5.6: GE results for our ResNet on **ASCADr**

	Our Resnet	Ensemble [28]	Model search CNN [29]
Nr. of traces for GE = 1	47	≈ 30	80

Table 5.3: Comparison of various different DL methods on **ASCADr**

When we look at the various feature selection scenarios for **ASCADr**, we see some minor differences from the results with **ASCADf**. In Figure 5.4, we see that the attacking performance is not negatively affected by the addition of non-informative noisy features as it is for **ASCADf**. It seems that with the larger training set of **ASCADr**, noisier input data is less problematic than in cases where the amount of training data is more limited. We see that as additional leakage points, as can be seen in Figure 5.7, get added to the feature set, the attacking performance consistently improves.

Nr. of features	Nr. of residual blocks	ResNet	CNN [29]	MLP [29]
1,400	8	47	80	58
1,500	8	34	≈ 190	≈ 140
2,000	8	37	≈ 170	≈ 140
2,500	9	37	≈ 80	≈ 140
3,000	9	29	≈ 40	≈ 130
3,500	9	21	≈ 20	4
4,000	9	14	≈ 10	≈ 10
4,500	10	9	8	≈ 10
5,000	10	8	≈ 10	≈ 15

Table 5.4: Comparing the number of traces required to reach GE = 1 at various feature selection scenarios to the results of Perin et al. [27] for **ASCADr**

When we then compare to state-of-the-art methods in Table 5.4, we see that the results of our networks are almost always slightly better than the hyper-parameter search strategy of Perin et al. [29]. The gap is initially quite significant as the attacking performance of the models' search strategies seems to suffer from the addition of noisy features, while our ResNets do not. From 3,000 features upward, this gap closes, and the performance of our ResNets is comparable to both the CNNs and the MLPs.

Finally, when we look at attacking the full feature set for **ASCADr** we again see that our networks are perhaps not suited for attacking these. In Figure 5.8, we can see the attacking results which do not improve over random guessing.

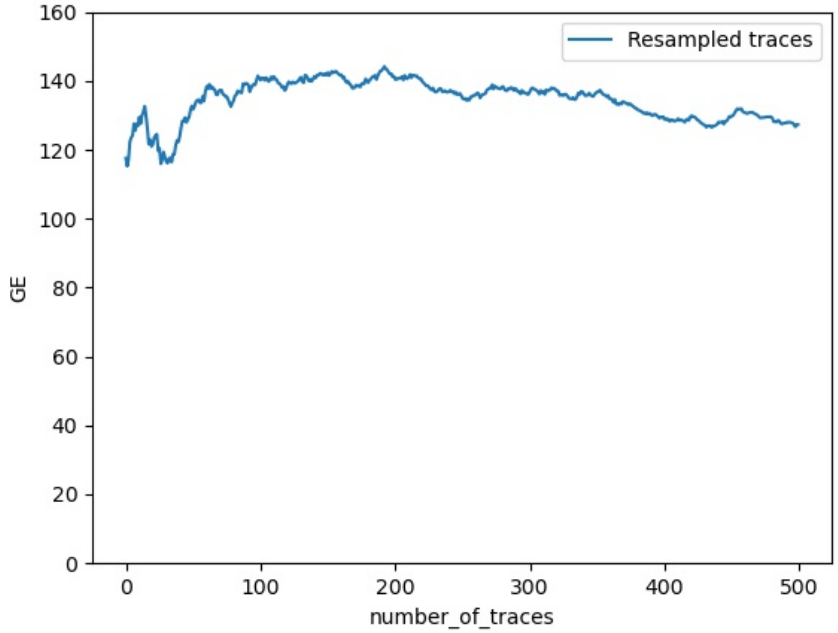


Figure 5.8: Attack performance of models attacking re-sampled traces of **ASCADr**

AES_HD

For the **AES_HD** data set, we see that our results are comparatively worse than in the cases of the **ASCAD** data sets. In Figure 5.9, we can see the guessing entropy results of our residual network. The network can successfully recover the key in approximately 4,100 traces. As can be seen in Table 5.5 this is worse than the ResNet of Jin et al., as this network can recover the key in about half the traces. On the other hand, it is better than the state-of-the-art CNN of Zaid et al..

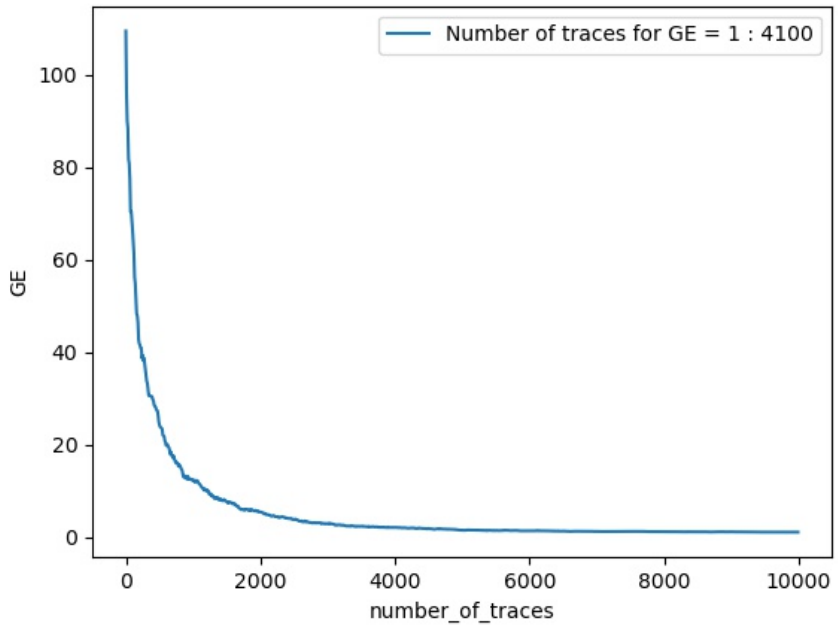


Figure 5.9: GE results for our ResNet on **AES_HD**

	Our Resnet	Jin et al. [17]	State-of-the-art CNN [44]
Number of traces for GE = 1	4,100	2,100	6,060

Table 5.5: Comparison of various different architectures on **AES_HD**

5.4. Discussion

Overall the attacking performance of our ResNet models is relatively good. We see that competitive attacking performance with state-of-the-art CNNs is achieved across various feature selection scenarios. Only on the complete feature set of **ASCADf** and **ASCADr** do we see attacking performance that is very different from the state-of-the-art methods. It is also of note that a singular architecture, which only varies in the number of residual blocks that are used across these scenarios, is comparable to random search methods that result in varying model architectures and hyper-parameter configurations. Additionally, we take the best results out of five runs of our models, while in the cases with automated hyper-parameter tuning, a very large number of architectures are tested, and only the best performance is used. We see that the performance of our ResNet models for **ASCADf** is slightly worse than the state-of-the-art. However, when there are more traces to be used for profiling for **ASCADr**, we see that the deeper ResNet models can perform on par with, or even better than, the state-of-the-art. Additionally, when we look at the **AES_HD** data set, we see that our models outperform the model of Zaid et al. [44] that was specifically made for this data set. This is surprising as our models were initially constructed for a very different data set and, with only minimal adjustments to hyper-parameters, were able to compete with the state-of-the-art models for **AES_HD**.

When we then compare our results with other ResNets that have been used in SCA, we see that our models perform a lot better on the **ASCADf** data set. The model that was used by Zhou et al. [46] does not seem to be able to launch successful attacks at all. The model of Jin et al. [17] does perform relatively well on the **ASCADf** data set, but it is not as good as our model. When we look at the **AES_HD** data set, we do see that the model of Jin et al. is very competitive with the state-of-the-art and outperforms our model by quite a large margin.

Our results show that these deeper networks are viable for use in SCA. As far as we can find, only Lu et al. [23] use networks that are of similar depth to attack SCA data sets, and even in this case, the primary motivation for this was the fact that large feature sets were used. In this work, we show that, contrary to the main trend in DL SCA, deeper networks can be used to achieve state-of-the-art performance on the **ASCAD** database. The fact that our ResNets perform better on the **ASCADr** set than on the **ASCADf** set leads us to reaffirm the need for deeper networks to break more complicated targets. For these original data sets, it is unnecessary to use these deep architectures to attack successfully, but in cases where smaller architectures are not successful, deep ResNets could be used.

An additional consideration is that all of the results presented here have almost identical hyper-parameter configurations. We see that our models provide state-of-the-art performance without needing too much hyper-parameter tuning. In the case of **ASCADr**, we see that the performance of our network compares to the ensembles of Perin et al. [28], and is better than other state-of-the-art model search strategies [29, 31, 41]. Additionally, the comparison to these state-of-the-art techniques is not entirely fair. These methods rely on the training and evaluation of hundreds of models, which comes with a high computational cost. In this chapter, we see that our models can achieve similar performance without the need for computationally intensive hyper-parameter searches.

Additionally, the models we provide converge more consistently than those that often result from these searches. The small model architectures, which result from hyper-parameter search strategies, often do not achieve their state-of-the-art performance when initialized with a different random seed. This consistency is important to consider as it greatly simplifies the evaluation process. If a model almost always converges, we only have to consider doing one or perhaps two training runs. In contrast, if a model is inconsistent, we may need to train it dozens of times to assess a device's security.

Furthermore, as data sets become more complex, our deeper ResNets can be a useful alternative to model search strategies. We postulate that for these newer data sets, like **ASCADv2** [25] and **AES_HD_mm** [39], our networks can provide state-of-the-art results after some tuning of the hyper-parameters. As we have seen

great results with our networks in cases where we see larger amounts of training data, we presume that this should also hold for these newer data sets.

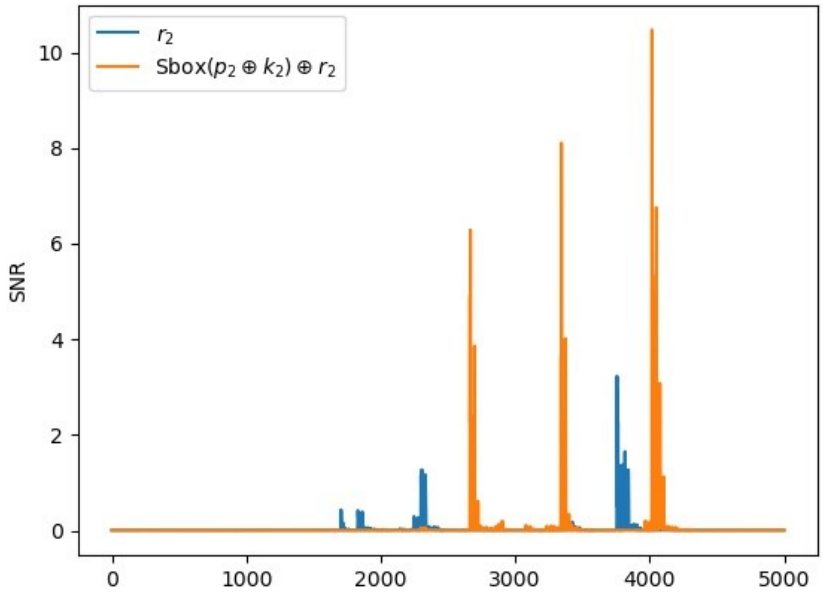


Figure 5.3: The SNR of all 5,000 used features for **ASCADf** with the standard 700 features in the middle.

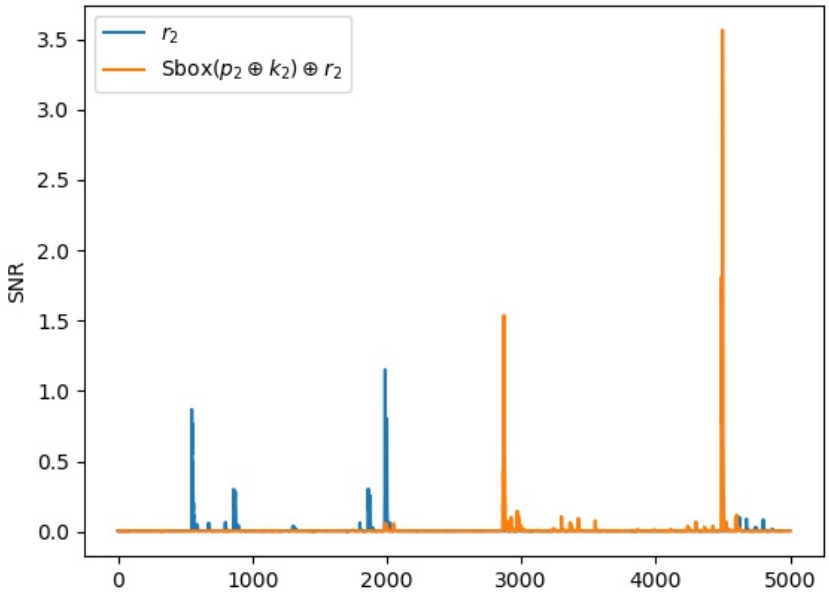


Figure 5.7: The SNR of all 5,000 used features for **ASCADr** with the standard 1,400 features in the middle.

6

Conclusion and Future Work

In this section, we will provide a conclusion to this thesis. We will first discuss the answers to our main research questions. Subsequently, we will address the limitations of this work and what future work we think should be done to explore the use of ResNets for SCA further.

6.1. Research Questions

RQ 1: How should ResNets be constructed for the purpose of side-channel analysis?

In chapter 4 we experimentally worked on finding ResNets that have solid attacking performance for SCA. Finding these ResNets was done by first exploring the construction of the residual blocks that we use. We identified six types of residual blocks that should be tested based on other recent works in the SCA community surrounding ResNets. These were then all tested on the **ASCADf** data set. These experiments were performed using a significantly deeper architecture than the architectures typically used for SCA, as deeper architectures are the context where ResNets could be useful. With these experiments, we find that residual block constructions similar to the ones used by Jin et al. [17] perform best.

Additional tests were also performed with various additions to the residual blocks. Batch normalization layers and pre-activation mechanisms were added to both the two-layer block and the inception block and again tested on the **ASCADf** data set. We show that these mechanisms do not provide significant improvements to the attacking performance of the networks. The additional computational costs for these mechanisms then result in the conclusion that they are not helpful additions for our ResNets for the currently used data sets.

Finally, we looked at how deep we should make the architecture. The main thing we explored is the number of useful residual blocks. We tested networks with varying depths on the **ASCADf** data set. With these experiments, we find that adding residual blocks does help the attacking performance of the ResNets (up to a point). The fact that deeper models perform better than smaller models is an exciting finding as most research in SCA since the work of Zaid et al. [44] has focused on using minimal amounts of convolutional layers. We conclude that the ResNet performance is best when the number of residual blocks is at most two smaller than the maximum amount that can be used based on the amount features.

In conclusion, we find architectures that are very different from the ResNets that were previously used in SCA [17, 46]. Our architectures are significantly deeper but also use fewer filters in the initial layers, leading to the relatively low computational cost of training our networks.

RQ 2: In what contexts is the use of ResNets over more common CNN architectures useful?

In chapter 5 we compare our networks' attacking performance to the other ResNets used in SCA and also the state-of-the-art DL methods currently being employed in SCA. We first look at data sets and intervals commonly used in the SCA field. Here we see that for the **ASCADf** with the standard 700 feature interval, our networks outperform the other ResNets used in SCA. The attacking performance of our network is also relatively close to the state-of-the-art network of Zaid et al. [44]. When we then look at the **ASCADr** data

set with the standard 1,400 feature interval, we see that the performance of our network is in line with the state-of-the-art ensemble models of Perin et al. [28]. We also note that on this data set, we outperform other state-of-the-art model search methods using reinforcement learning [31] and grid/random search [29].

Subsequently, we also explore various feature selection scenarios for both the **ASCADf** and the **ASCADr** data sets. This was done to investigate whether our ResNets perform better with more features. We take larger feature windows surrounding the standard features and test the ResNets on several windows. We find that for **ASCADf** we are again a bit worse than the model search results of Perin et al. [29]. For **ASCADr** however, our results are a fair bit better across the board. These variations in the performance lead us to conclude that ResNets are not better at handling larger amounts of features when compared to MLPs and CNNs.

These results then lead us to conclude that these ResNets are not strictly better than the smaller models that have been used in SCA in recent years. We see that the ResNets achieve attacking performance similar to the state-of-the-art with minor variations in the architecture across various feature selection scenarios and data sets. Additionally, while the ResNets are computationally more intensive to train than the individual models used in the state-of-the-art methods, these methods often require the training of many of these models to achieve their best performance. This then leads us to conclude that ResNets can provide a useful alternative to the state-of-the-art in all explored SCA scenarios.

6.2. Contributions

- In this thesis, we propose novel ResNets specifically made for SCA data sets. These architectures are significantly deeper than the other state-of-the-art network architectures currently used for SCA.

- We explore several variations of residual blocks and find the best versions to use experimentally. We conclude that using convolutional stride seems significantly less effective than pooling layers. Additionally, we see that using at most two convolutional layers per residual block is best for attacking SCA data sets.

- We provide insights into how deep these residual architectures should be to provide improved attacking performance. This conclusion is not in line with the current trend in the research community to only use relatively shallow architectures. It makes it clear that deeper architectures should be considered for future works.

- We show our architectures provide competitive attacking performance with the state-of-the-art DL methods on various public data sets. Our architectures perform on par with the state-of-the-art model search strategies while requiring only minimal hyper-parameter tuning.

6.3. Limitations and Future Work

One of the main limitations of this thesis is that we only explore a limited set of hyper-parameter configurations. We chose to limit our search of optimal hyper-parameters and instead set these to some 'good-enough' values based on literature. This was done as the focus of this work was to explore how to construct residual architectures for SCA. We note that the hyper-parameter values that we used were found to be optimal for smaller architectures by Zaid et al. [44]. However, our architectures are very different, and as such, it could be the case that better setups can be found. As our results were already competitive, we did not attempt to further improve these 'good-enough' hyper-parameter setups. However, in future work, this is an interesting area to explore.

Another limitation of our work is that we did not attack data sets that have employed various hiding countermeasures. From previous work, it seems that our architectures should be able to attack data sets with these countermeasures as architectures that employ similar VGG-like structures have been used for this [19]. As such, this could be an interesting direction for future work.

Other directions for future work could be to attack newer data sets with more countermeasures. For example, the **ASCADv2** data set with higher-order masking schemes and shuffling countermeasures could be an attractive target as it has not yet been broken in a black-box threat model. Another data set that could be

explored is the **AES_HD_mm** data set. This data set is similar to the **AES_HD** data set as it is a hardware implementation that leaks in the HD leakage model, but it also implements a masking scheme making it more difficult to attack.

Bibliography

[1] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. Understanding of a convolutional neural network. In 2017 International Conference on Engineering and Technology (ICET), pages 1–6. Ieee, 2017.

[2] S Abhishek Anand and Nitesh Saxena. 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, pages 346–364. Springer, 2016.

[3] Ryad Benadjila, Emmanuel Prouff, Rémi Strullu, Eleonora Cagli, and Cécile Dumas. 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.

[4] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In International workshop on cryptographic hardware and embedded systems, pages 16–29. Springer, 2004.

[5] Vincenzo Carletti, Antonio Greco, Gennaro Percannella, and Mario Vento. Age from faces in the deep learning revolution. IEEE transactions on pattern analysis and machine intelligence, 42(9):2113–2132, 2019.

[6] Suresh Chari, Josyula R Rao, and Pankaj Rohatgi. Template attacks. In International Workshop on Cryptographic Hardware and Embedded Systems, pages 13–28. Springer, 2002.

[7] Pádraig Cunningham, Matthieu Cord, and Sarah Jane Delany. Supervised learning. In Machine learning techniques for multimedia, pages 21–49. Springer, 2008.

[8] Joan Daemen and Vincent Rijmen. The design of Rijndael, volume 2. Springer, 2002.

[9] Li Deng and Yang Liu. Deep learning in natural language processing. Springer, 2018.

[10] Li Deng, Geoffrey Hinton, and Brian Kingsbury. New types of deep neural network learning for speech recognition and related applications: An overview. In 2013 IEEE international conference on acoustics, speech and signal processing, pages 8599–8603. IEEE, 2013.

[11] Pierre Dusart, Gilles Letourneux, and Olivier Vivolo. Differential fault analysis on aes. In International Conference on Applied Cryptography and Network Security, pages 293–306. Springer, 2003.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 770–778, 2016.

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In European conference on computer vision, pages 630–645. Springer, 2016.

[14] Gabriel Hospodar, Benedikt Gierlichs, Elke De Mulder, Ingrid Verbauwhede, and Joos Vandewalle. Machine learning in side-channel analysis: a first study. Journal of Cryptographic Engineering, 1(4):293, 2011.

[15] Mei-Chen Hsueh, Timothy K Tsai, and Ravishankar K Iyer. Fault injection techniques and tools. Computer, 30(4):75–82, 1997.

[16] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In International conference on machine learning, pages 448–456. PMLR, 2015.

[17] Minhui Jin, Mengce Zheng, Honggang Hu, and Nenghai Yu. An enhanced convolutional neural network in side-channel attacks and its visualization. arXiv preprint arXiv:2009.08898, 2020.

[18] Maikel Kerkhof, Lichao Wu, Guilherme Perin, and Stjepan Picek. No (good) loss no gain: Systematic evaluation of loss functions in deep learning-based side-channel analysis. Cryptology ePrint Archive, 2021.

[19] Jaehun Kim, Stjepan Picek, Annelie Heuser, Shivam Bhasin, and Alan Hanjalic. Make some noise. unleashing the power of convolutional neural networks for profiled side-channel analysis. IACR Transactions on Cryptographic Hardware and Embedded Systems, pages 148–179, 2019.

[20] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014.

[21] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Annual international cryptology conference, pages 388–397. Springer, 1999.

[22] Huimin Li, Marina Krček, and Guilherme Perin. A comparison of weight initializers in deep learning-based side-channel analysis. In International Conference on Applied Cryptography and Network Security, pages 126–143. Springer, 2020.

[23] Xiangjun Lu, Chi Zhang, Pei Cao, Dawu Gu, and Haining Lu. Pay attention to raw traces: A deep learning architecture for end-to-end profiling attacks. IACR Transactions on Cryptographic Hardware and Embedded Systems, pages 235–274, 2021.

[24] Houssem Maghrebi, Thibault Portigliatti, and Emmanuel Prouff. Breaking cryptographic implementations using deep learning techniques. In International Conference on Security, Privacy, and Applied Cryptography Engineering, pages 3–26. Springer, 2016.

[25] Loïc Masure and Rémi Strullu. Side channel analysis against the anssi's protected aes implementation on arm. 2018.

[26] Loïc Masure, Nicolas Belleville, Eleonora Cagli, Marie-Angela Cornélie, Damien Couroussé, Cécile Dumas, and Laurent Maingault. Deep learning side-channel analysis on large-scale traces. In European Symposium on Research in Computer Security, pages 440–460. Springer, 2020.

[27] Guilherme Perin and Stjepan Picek. On the influence of optimizers in deep learning-based side-channel analysis. 12804:615–636, 2020. doi: 10.1007/978-3-030-81652-0_24. URL https://doi.org/10.1007/978-3-030-81652-0_24.

[28] Guilherme Perin, Łukasz Chmielewski, and Stjepan Picek. Strength in numbers: Improving generalization with ensembles in machine learning-based profiled side-channel analysis. IACR Transactions on Cryptographic Hardware and Embedded Systems, pages 337–364, 2020.

[29] Guilherme Perin, Lichao Wu, and Stjepan Picek. Exploring feature selection scenarios for deep learning-based side-channel analysis. Cryptology ePrint Archive, 2021.

[30] Stjepan Picek, Annelie Heuser, Alan Jovic, Shivam Bhasin, and Francesco Regazzoni. 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.

[31] Jorai Rijsdijk, Lichao Wu, Guilherme Perin, and Stjepan Picek. Reinforcement learning for hyperparameter tuning in deep learning-based side-channel analysis. IACR Trans. Cryptogr. Hardw. Embed. Syst., 2021(3):677–707, 2021. doi: 10.46586/tches.v2021.i3.677-707. URL https://doi.org/10.46586/tches.v2021.i3.677-707.

[32] Ronald L Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM, 21(2):120–126, 1978.

[33] Frank Rosenblatt. The perceptron, a perceiving and recognizing automaton Project Para. Cornell Aeronautical Laboratory, 1957.

[34] Leslie N Smith. Cyclical learning rates for training neural networks. In 2017 IEEE winter conference on applications of computer vision (WACV), pages 464–472. IEEE, 2017.

[35] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In Thirty-first AAAI conference on artificial intelligence, 2017.

[36] Biaoshuai Tao and Hongjun Wu. Improving the biclique cryptanalysis of aes. In Australasian Conference on Information Security and Privacy, pages 39–56. Springer, 2015.

[37] Sasha Targ, Diogo Almeida, and Kevin Lyman. Resnet in resnet: Generalizing residual architectures. arXiv preprint arXiv:1603.08029, 2016.

[38] H Taud and JF Mas. Multilayer perceptron (mlp). In Geomatic Approaches for Modeling Land Change Scenarios, pages 451–455. Springer, 2018.

[39] Yoo-Seung Won, Xiaolu Hou, Dirmanto Jap, Jakub Breier, and Shivam Bhasin. 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.

[40] Lichao Wu and Guilherme Perin. On the importance of pooling layer tuning for profiling side-channel analysis. 12809:114–132, 2021. doi: 10.1007/978-3-030-81645-2_8. URL https://doi.org/10.1007/978-3-030-81645-2_8.

[41] Lichao Wu, Guilherme Perin, and Stjepan Picek. I choose you: Automated hyperparameter tuning for deep learning-based side-channel analysis. IACR Cryptol. ePrint Arch., 2020:1293, 2020.

[42] Meiyin Wu and Li Chen. Image recognition based on deep learning. In 2015 Chinese Automation Congress (CAC), pages 542–546. IEEE, 2015.

[43] Yuval Yarom and Katrina Falkner. Flush+ reload: A high resolution, low noise, l3 cache side-channel attack. In 23rd {USENIX} Security Symposium ({USENIX} Security 14), pages 719–732, 2014.

[44] Gabriel Zaid, Lilian Bossuet, Amaury Habrard, and Alexandre Venelli. Methodology for efficient cnn architectures in profiling attacks. IACR Transactions on Cryptographic Hardware and Embedded Systems, 2020(1):1–36, 2020.

[45] Jiajia Zhang, Mengce Zheng, Jiehui Nan, Honggang Hu, and Nenghai Yu. A novel evaluation metric for deep learning-based side channel analysis and its extended application to imbalanced data. IACR Transactions on Cryptographic Hardware and Embedded Systems, pages 73–96, 2020.

[46] Yuanyuan Zhou and François-Xavier Standaert. 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.