Side-Channel Attacks using Convolutional Neural Networks

Ioannis Petros Samiotis

A Study
on the performance
of Convolutional Neural Networks
on side-channel data

Sample index of data acquisition point
8 spectrogram images
56 MRP images = 7 layers×8 points
USRP N200

TU Delft

Side-Channel Attacks using Convolutional Neural Networks

by

Ioannis Petros Samiotis

to obtain the degree of Master of Science
at the Delft University of Technology
to be defended publicly on Thursday April 26, 2018 at 13:00.

Student number:	4504232	
Thesis committee:	Dr. S. Picek,	TU Delft, supervisor
	Dr. ir. J.C.A. van der Lubbe,	TU Delft
	Prof. dr. A. Hanjalic,	TU Delft

TU Delft

Abstract

Side-Channel Attacks, are a prominent type of attacks, used to break cryptographic implementations on a computing system. They are based on information "leaked" by the hardware of a computing system, rather than the encryption algorithm itself. Recent studies showed that Side-Channel Attacks can be performed using Deep Learning models. In this study, we examine the performance of Convolutional Neural Networks, on four different datasets of side-channel data and we compare our models with conventional Machine Learning algorithms and a CNN model from literature. We found that CNNs have the potential to achieve high accuracy performance (99.3%), although their capacity is heavily influenced by the use case. We also found that certain Machine Learning algorithms can outperform CNNs in certain cases, leaving an open debate on the performance gains of the latter.

Preface

An genuine interest in privacy and classification systems urged me to pursue the topic of this thesis. It was a challenge which helped me broaden my knowledge on Deep Learning, Machine Learning, Optimization and of course, Software Development. While studying the limited literature on the topic, I quickly became curious on the specific performance gains that Convolutional Neural Networks can bring. As such, it became the main premise of this work, where me and my supervisor Dr. Stjepan Picek tested the efficiency of CNNs in various cases.

Many people have supported me through this study. Many colleagues and supervisors from the university and the company where I conducted my internship and of course my family and friends. Naming each and every one would be impossible, but I'm deeply grateful to everyone, as each helped me in a certain way.

Ioannis Petros Samiotis,
April 2018

List of Abbreviations

Abbreviation	Meaning
AES	Advanced Encryption Standard
ANN	Artificial Neural Networks
API	Application Programming Interface
ARM32	Advanced RISK (Reduced Instruction Set Computer) Machine 32-bit architecture
CNN	Convolutional Neural Networks
DL	Deep Learning
DL4J	Deeplearning4j
DPA	Differential Power Analysis
FPGA	Field-programmable Gate Array
GPU	Graphics Processing Unit
LR	Logistic Regression
LSTM	Long-Short Term Memory
ML	Machine Learning
MLP	Multi-Layer Perceptron
NB	Naive Bayes
SCA	Side-channel Attacks
SPACE	Security, Privacy and Applied Cryptographic Engineering
SNR	Signal-to-Noise Ratio
SVM	Support Vector Machine
XGB	Extreme Gradient Boost
0-R$_{ML}$	ZeroR Classifier (Machine Learning)
0-R$_{CNN}$	ZeroR Classifier (Convolutional Neural Networks)

Contents

1 Introduction

Information and connectivity are two of the main characteristics of our society. Computers and smart devices have helped connecting people and sharing information between them, much easier and faster than ever before. Economic transactions and information transfers are all transmitting important data that need established and safe connections between a sender and a receiver. In many cases, this can be achieved by using specialized hardware on both ends, such as smartcards, fingerprint scanners and face recognition cameras. Although cyber security has progressed a lot in the recent years, no infrastructure can ever be guaranteed to be totally secure. More and more sophisticated attacks surface and reveal vulnerabilities that were never thought before.

When describing a cyber security incident, we describe the attempt (succeeded or failed) to steal data or disrupt an infrastructure (in some cases both). Stealing data can be achieved through a passive or intrusive way. Passively, an attacker could bypass the defences of a target and collect the data that are being processed, without leaving any traces of interacting with the target. Gaining information from a target through such means doesn't necessarily imply weaknesses in the software implementation or the encryption algorithm. An attacker could achieve a passive intrusion through the hardware implementation of the system. For example, while one is monitoring the power consumption of a cryptographic device, he could notice differences between the idle consumption and the consumption when an encryption or decryption process is performed. Such variations could indicate minor things such as if a device is active, without any further insight in the implemented algorithm itself. In the case though that an attacker is able to compare that power consumption with the input/output data, he could possibly retrieve the encryption keys of that device. This would enable him to monitor the data transferred through that device in the future.

The above example is a prime example case of a Side-Channel Attack (SCA). SCAs on hardware are being performed and are known since the analog era. Considering they are ignoring the software defences, they are capable of breaking theoretically unbreakable encryption algorithms, through monitoring the hardware's activity. They are difficult to trace and since technology is integrated more and more into our everyday routines, companies are trying to implement sophisticated hardware countermeasures to ensure the security of their systems.

Due to the nature of the mathematical models, which describe the relations between side-channel data and the encryption keys, SCAs could be transformed into a classification problem. Using again the power consumption example, an attacker could collect the power consumption data and relate them, through a model, with appropriate labels. This makes possible the use of classification models, a powerful tool in the hands of an attacker. Machine Learning (ML) has developed and increased the performance of statistical models making systems capable to easily recognize patterns on input data and correlate those with their appropriate labels.

Recent works in the field have shown that Deep Learning (DL) algorithms can be used in side-channel data, achieving rather good performance in key retrieval [26] and in classification accuracy [6]. Through these works, the DL systems demonstrate better performance than ordinary attacks, surpassing well established techniques. This made us interested on the use cases of SCAs that DL could excel and we took a data science approach on the topic. Through this work, we want to understand the characteristics of the data and why specific DL algorithms seem to be the most fitted for the problem.

We examined the use of a specific technique, the Convolutional Neural Networks (CNNs), on side-channel data. CNNs are a Deep Learning technique that in [26] and [6], was found to achieve the best results among the other DL techniques. In this work we study their performance on power consumption traces of four different datasets. In the first chapter, we present a general introduction to Deep Learning and CNNs especially, as well as an overview of Side-Channel Attacks, in order to provide all the necessary background on the topic. Continuing the literature review of this study, we present the related work in the field of SCAs and DL, while presenting an optimization technique that was at the core of our system's functionality. After introducing the main concepts, we show our approach on the problem, the goals we wanted to reach in our study and the system we implemented to achieve these goals. Having introduced our goals and the system's overview, we proceed in explaining our experiments' design alongside the data structures and classification algorithms that were used. Finally, we showcase the results of our experiments and discuss their meaning.

This study was made in conjunction with an industrial internship at Riscure B.V. and the Computer Science MSc study programme in Delft University of Technology. In later sections we discuss design choices and experiments that took place during the study as a whole, but we explicitly remark the parts of the work during the internship. That is due to substantial changes to the codebase and the overall approach on the research topic, between the two parts of the study.

2
Literature Review

We start our work with a review of Deep Learning and Side-Channel Attacks. We give an overview of each respective field, accompanied with descriptions and examples of specific elements which were needed in our study. Finally, we present the work that has been done so far in the field of Side-Channel Attacks using Deep Learning, alongside a specific optimization technique that was pivotal to our research. Through this chapter, we present all the relevant work on which we base our research methodology.

2.1. Deep Learning and Convolutional Neural Networks

Deep Learning is the study and development of neural network architectures for classification tasks. In our study we used a special type of Neural Networks (NN), the Convolutional Neural Networks (CNN), following the works of [26] and [6]. We start this section by presenting some of the research milestones in the field so far, following it with short descriptions of important elements of NN, that we used in our work. Finally, we describe how Convolutional Neural Networks work and some of the classification breakthroughs of certain CNN architectures.

2.1.1. Historical Overview

The field of Artificial Neural Networks (ANN) is old, with a lot of research sparking debates on "thinking machines" and Artificial Intelligence capabilities. In 1943 Warren McCulloch and Walter Pitts published the first study [27] on how actual neural networks could possibly work and modeled a simple neural network using electrical circuits. In later years, we have the first applied artificial neuron ADALINE (ADAptive LINear Element) and artificial neural network MADALINE (Multiple ADAptive LINear Elements) [47], designed by Bernard Widrow and Marcian Hoff in 1960. Better network architectures were introduced through the years that followed and in 1986, a team of researchers including David Rumelhart, introduced the back-propagation technique [37]. This technique, allows classification errors to propagate throughout the neurons of the network and calculates their error contribution per data batch. This results in a better classification model but made the networks become "slower learners", needing more time to train on data.

In the period between the 80s and 2000s, the Artificial Intelligence (AI) research was perceived to be have significantly slowed down. That could be attributed mainly to the low computational capabilities of the available hardware, alongside the high expectations for AI that were not met yet. Nevertheless, AI systems were more and more integrated into systems of our every day lives, like banking systems and air traffic control. During that period we had the birth of many different types of ANN, like Convolutional Neural Networks [22] and Long-Short Term Memory machines (LSTMs) [15], all important neural network architectures in modern Deep Learning research.

During 2010s, through conference competitions and hardware advancements, ANN research grew as a field, becoming a dominant pattern recognition practice. Breakthroughs in image classification (AlexNet [21], GoogleNet [43], VGG-16 [39]) and the integration of GPUs as computing systems for ANN, brought Deep Learning in the spotlight, leading to numerous breakthroughs in the pattern recognition field.

2.1.2. Basic Structural Elements of Artificial Neural Networks

In the following parts, we introduce basic concepts of ANN that are being used throughout our work. Some of them are essential theoretical concepts, while others are practices and techniques that their use is being encouraged in the literature.

Artificial Neurons

Neurons are the core component of an Artificial Neural Network. They are mathematical functions that are inspired from the biological neurons and they are the basic classifying components of the network. Mimicking the biological neurons, the artificial neurons are activated if "stimulated" with the right values. A neuron receives values as input, and outputs values if the sum of the input exceeds the threshold of the neuron [35].

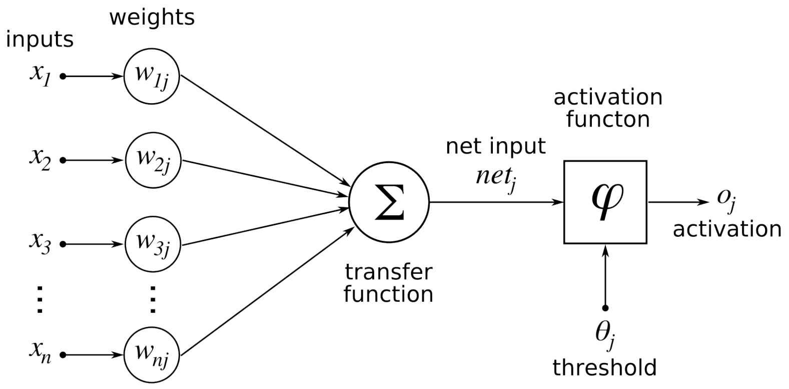


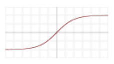
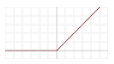
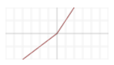
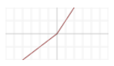
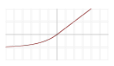
Figure 2.1: Representation of an Artificial Neuron [9]

As we see in figure 2.1, a neuron assigns weights on each input separately and summarizes the results. These weights are very important to the network and they are defined by its Cost function. The weights are being updated through the training process, to reflect the patterns found in the data that the ANN is classifying. As a rule of thumb, the ANNs seem to perform better with more data, and if the training process is long enough, the neurons' weights can become extremely efficient on the training data. This can lead to overfitting issues, meaning the ANN has a poor ability to generalize on datasets other than the training data. Later we discuss how to prevent such issues using regularization. Finally, the sum of the weighted inputs is used in the neuron's Activation function to determine if the neuron "activates", meaning whether it outputs a result or not.

Activation Functions

The type of Activation function, which determines if a neuron activates or not, is the simplest form of activation function. It is called the Binary Step function [36]. Introducing more states in the neuron's activation pattern though, makes the ANN more capable to classify complex data structures. To achieve such intermediate values in neuron's activation, we make use of functions where the neuron's activation values are bound within the range of a function. A simple example of such an Activation function, is the Sigmoid function. In the following Table 2.1, we show the Activation functions that were used in our classification system, alongside their mathematical definitions and graphical representations.

Table 2.1: Activation functions used in this study

Activation Function	Definition	Graph
TanH (Hyperbolic Tangent)	$f(x) = tanh(x) = \frac{2}{1+e^{-2x}} - 1$	
ReLU (Rectified Linear Unit) [29]	$f(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0 \end{cases}$	
LeakyReLU (Leaky Rectified Linear Unit) [24]	$f(x) = \begin{cases} 0.01x & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0 \end{cases}$	
PReLU (Parametric Rectified Linear Unit) [14]	$f(\alpha, x) = \begin{cases} & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0 \end{cases}$	
ELU (Exponential Linear Unit) [11]	$f(\alpha, x) = \begin{cases} \alpha(e^x - 1) & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0 \end{cases}$	
SELU (Scaled Exponential Linear Unit) [19]	$f(\alpha, x) = \begin{cases} \alpha(e^x - 1) & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0 \end{cases}$	-
Softmax [10]	$f_i(\vec{x}) = \frac{e^{x_i}}{\sum_{j=1}^{J} e^{x_j}}$	-

Multilayered networks

A number of neurons in parallel constitute a layer. Feed-forward architectures consist of stacked layers of neurons, where each neuron of the previous layer is connected with all the neurons of the next layer. Such networks are also known as Fully-Connected Network. These kind of networks make no assumptions about the data features and are memory and computationally expensive, due to the number of connections and weights.

A network of two layers, makes use of an *Input layer* where the neurons' input are the values of the target data and an *Output layer* where the neurons' output is the classification result of the network. In the cases where the network has more than two layers, all the intermediate layers are called *Hidden layers*. The values of the Output layer, represent the label that the network associates with the input data. In the case of a Binary classification, the network outputs one of two values, while in Multiclass classification, the network outputs one out of many values. The classification task in our study is a Multiclass classification problem.

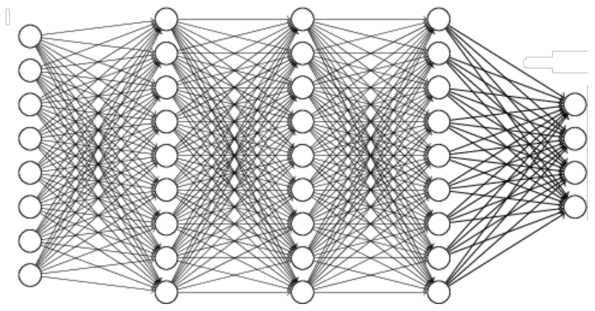


Figure 2.2: Fully Connected Neural Network with 3 Hidden Layers [1]

Cost Functions and Regularization

Due to their mathematical nature, the ANNs performs a nonlinear mapping of the input data vectors to their corresponding values (labels). Cost functions are being used to update the network's weights and their results are an indicator of the network's overall performance. Their result is a non-negative value and it measures the inconsistency between the predicted value

($\hat{y}$) and their ground truth label (y). A Cost function consists of an empirical risk term and a regularization term which penalizes the wrong predictions of the network (if the ANN uses regularization). While the ANN is being trained, it tries to minimize the loss value of its Cost function, in order to increase its classification performance.

Before presenting the specifics of the Cost function used in our study, we will explain the role of Regularization. As explained before, by using Regularization, the network can avoid overfitting on the training data and increase its generalization capabilities. Otherwise, the connections between specific neurons in the network become so important for it (high weights on those connections), that other connections that could associate with different types of data patterns, are being ignored, lowering its overall performance as shown in Figure 2.3.

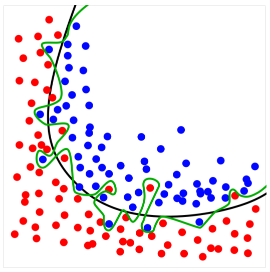


Figure 2.3: Green line represents an overfitting classifier and the black line represents a regularized classifier [7]

The Regularization techniques that are relevant to our work, are:

- L1 regularization
- L2 regularization
- Dropout

L1 and L2 regularization techniques, are both targeted to the neurons' weights; they both penalize large weight values. Their main difference is that L1 shrinks a weight ω by a constant value towards 0, while L2 shrinks ω proportionally to its value. To contrast their resulting output, when the magnitude of $|\omega|$ is large, L1 shrinks the value much less than L2 and when the magnitude of $|\omega|$ is small, L1 shrinks the value much more than L2. This results in L1 having a net result of high weighted connections becoming more and more important, while the rest of the connections are driven to 0.

The L1 regularization update is defined as:

$$\omega \rightarrow \hat{\omega} = \omega - \frac{\eta\lambda}{n} sign(\omega) - \eta\frac{\partial C_0}{\partial\omega}$$

and the L2 regularization update is defined as:

$$\omega \rightarrow \hat{\omega} = \omega(1 - \frac{\eta\lambda}{n}) - \eta\frac{\partial C_0}{\partial\omega}$$

where λ is the regularization parameter, η is the learning rate (more in **??**), C_0 is the initial value of the cost function and n is the total number of neurons in the network. In our work we made use of L_2 regularization instead of L_1, as it met our classification needs.

Dropout [41] is a technique which doesn't modify the Cost function. It instead modifies the network itself, by modifying its topology. Depending on the magnitude of the Dropout effect, the ANN temporarily deactivates a number of random neurons in the hidden layers while training. After the network has parsed through the whole batch of data, the network reactivates the neurons and randomly selects new ones to deactivate, while recalculating the weights and biases of the network for the new batch. This technique can be applied to the convolutional hidden layers as well and the performance gains in our system, were in line with [41].

2.1.3. Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a type of networks that are primarily used for image classification and object recognition. They were inspired by how visual cortex operates in cats [17] and monkeys [18], where overlapping receptive fields respond to visual stimuli in their region.

As their name suggests, the network calculates a series of convolutions on the input data. To achieve this, they use a series of certain filters (n-dimensional arrays), which overlay on the input data and the product of this convolution is the output of the "neuron". They are Multi-layered networks, and they mainly use 3 types of layers, Convolutional layers, Pooling Layers and Fully-connected layers. In the most common architectures, we find a Convolutional layer as an Input layer, a series of Convolutional and Pooling layers for the Hidden layers and a Fully-connected layer as Output layer. To better demonstrate how CNNs perform data classification through these layers, we will use an example on image data.

An digital image is an array of pixel values. Each pixel mainly holds information on color channels, resulting on images using the RGB color scheme, to be 3 dimensional data. To simplify our example, we assume that the picture is black and white, so each picture can have one value. In the Input Convolutional layer, there are a certain number of filters (else known as kernels), whose values will be used for the convolutions. In this example, the filter has 2 dimensions and a square shape and it detects features on the image by applying convolution from left to right, top to bottom. The size of the step the kernel uses to parse through the image, is called Stride. The result of the convolutions using this filter, constitute an array called Feature Map or Activation Map. If the layer utilizes more than one kernels, then the dimensionality of the layer's output increases. The volume of the Feature Map is layer's output depth. Through this process, the layer has detected the first order features of the image. Each filter applied, detects different types of features (color value, curvature etc.), so the more filters applied, the more features that will be detected. Likewise, the more convolutional layers applied, the higher the level of features detected, as the filters detect patterns on features found by the previous layer. This has the results of CNN being able to detect faces and objects in images. The Figure 2.4 shows a graphical representation of how the filters are applied on the data.

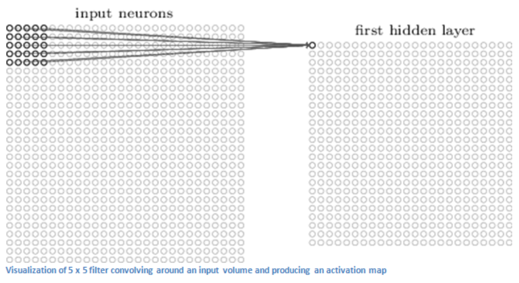

input neurons
first hidden layer
Visualization of 5 x 5 filter convolving around an input volume and producing an activation map

Figure 2.4: Creation of an Activation Map [30]

The main role of Pooling layers is to down-sample the data between the layers of the CNN. They apply kernels on the input data, in the same way that convolutional layers do. This time though, the output of the layer is a smaller version of the input by either summarizing the input or by considering only the max value of it. An example of the Pooling process using MaxPooling, can be found in the Figure 2.5.

Through the use of CNN, there were several breakthroughs in image classification tasks. One of the first breakthroughs was made with LeNet [22]. LeNet is a CNN which can identify handwritten digits in images, needing only minimal data pre-processing. In Figure 2.6, we show how an image of a digit is being classified using LeNet.

Typical examples of Convolutional Neural Networks, whose architectures are optimized for use with GPUs, are the AlexNet which achieved a top-5 error (top-5 label candidates) of 15.3% [21] and GoogleNet which is a 22 layers deep CNN and set a new state-of-the-art in image classification, in the ImageNet Large-Scale Visual Recognition Challenge 2014 [38].

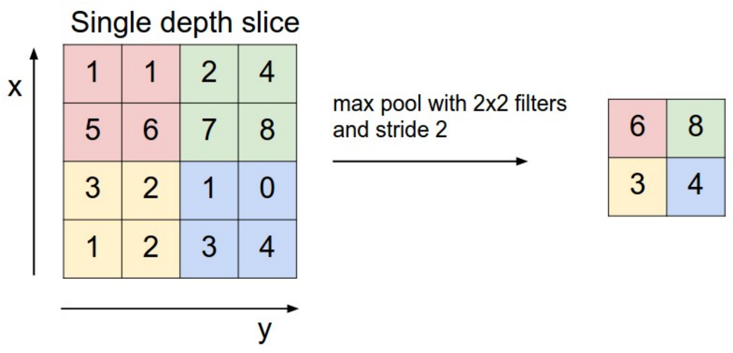


Figure 2.5: MaxPooling output [30]

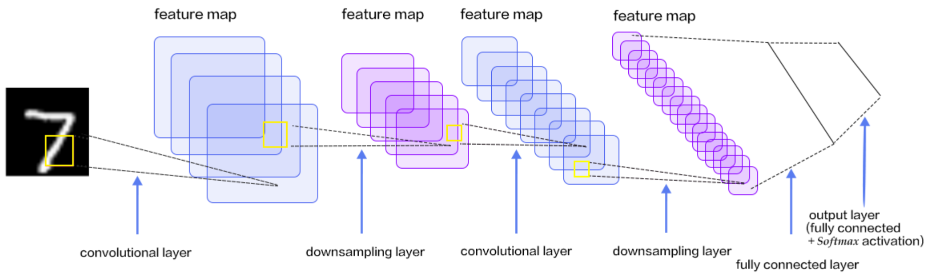


Figure 2.6: LeNet architecture [31]

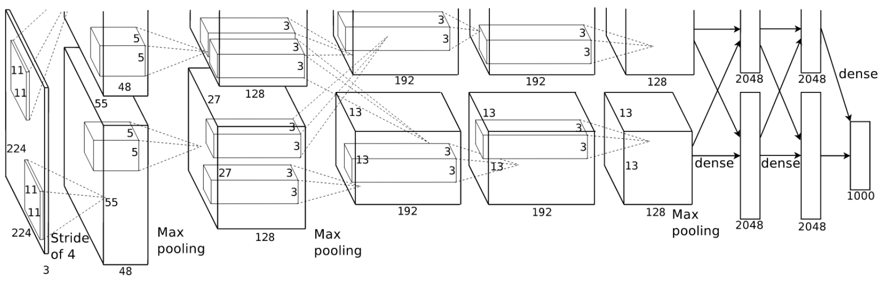


Figure 2.7: AlexNet architecture [21]

2.2. Breaking Cryptographic Implementations Using Power Consumption Data

Throughout this chapter, we will present the necessary background regarding the Side-Channel Attacks (SCAs) as a field. More importantly, the type of Side-Channel Attack that is more relevant to our work, the Power Analysis Attack. We will explain how the power consumption of a cryptographic device can be associated with its encryption key and how this type of attack can be transformed into a classification task.

2.2.1. Overview of Side-Channel Attacks

In order to understand Side-Channel Attacks, we will start by explaining what does the term "side-channel" stand for. Side-Channel is an unintended interface for monitoring or operating a device, resulting from its physical implementation. For the SCAs, the attacker needs access to the hardware of the target device, in order to gather data on unintended information leakage. These leaked information can be fluctuations of the device's temperature, electromagnetic radiation, power consumption, etc, that can indicate when the device is performing a certain process. By combining the leaked information with mathematical models to express this leakage, an attacker can retrieve the keys of the encryption algorithm that is being performed on the target cryptographic device.

Side-Channel Attacks (SCAs) are a serious threat for many cryptographic devices, as they can compromise them, even if the encryption algorithm is strong. Short-comings on secure hardware design can lead to powerful SCAs like the most recent security exploits on CPUs' cache, Meltdown [23] and Spectre [20]. In our specific case study, we focus on SCAs associated with the power consumption of a target device.

2.2.2. Few notes on the AES

The Side-Channel Attacks which we will focus on this study, all make use of the AES algorithm. The AES is an encryption standard (as its name suggests, "Advanced Encryption Standard"), which is widely used to secure digital data. It is a subset of the Rijndael cipher and it has variations of AES-128, AES-192 and AES-256. The different variations depend on the length of the key that is being used, and all use blocks of 128 bits. We should mention that the AES is a symmetric-key algorithm, meaning that the same key is used for encryption and decryption of the data. As in this study we only classify data where AES-128 was used, we will abbreviate AES-128 to AES from now on.

More specifics on the AES can be found in the original work of [12]. As the exact functionality of AES is not relevant to our case study, in the following sections we will only refer to certain parts of AES that are relevant with our work.

2.2.3. Power Analysis Attacks

As previously mentioned, the power consumption of a device can be used in an attack to extract critical information from a target device. It is an non-invasive type of SCA and the attacker needs to have access to the device so he can measure power consumption.

The procedure of a power analysis attack starts with an attacker sending data to the cryptographic device, which triggers the execution of the encryption algorithm on the target. The device then encrypts the input data and ouputs the results. While the encryption takes place, the attacker measures the power consumption of the cryptographic device through an oscilloscope (Figure 2.8). The attack is virtually untraceable, as the encryption device operates as intended and the attacker only measures a side product of the device's operation. We assume that the type of encryption algorithm is known to the attacker, who then applies the proper mathematical model (leakage model) to retrieve the encryption key. The attacker needs to combine theoretical knowledge of the encryption algorithm with the understanding of the specifics of its implementation in order to apply a proper leakage model to the measured power consumption traces.

To better demonstrate a power analysis attack, we will use an example attack on a target device with an implementation of AES-128 encryption. The encryption key in this example is constant for the device and it is secret. The AES implementation in our example, is based on

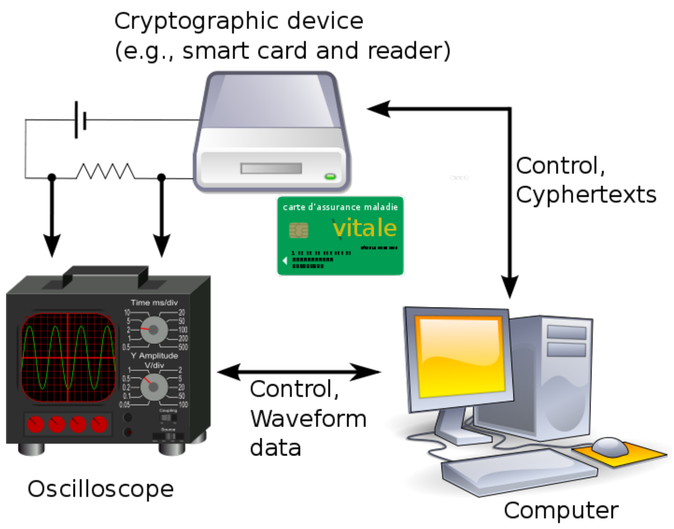


Figure 2.8: Differential Power Analysis [33]

a key with 16 bytes length $K_{16} \in \{k_1, ..., k_{16}\}$. The attacker proceeds on sending data D with 16 bytes length $\{d_1, ..., d_{16}\}$. The data message sent, differs each time. So when the data D are sent to the device, they trigger the AES encryption process on the target device. The electrical power that is being consumed for the process, is being measured by the oscilloscope, which produces a power trace t (Figure 2.10).

The fact that the implementations of AES encryption are taking place per byte of the input data, allows the attacker to perform the attack per byte of the encryption key. The AES procedure for one byte $d_1 \in D$ with $k_1 \in K_{16}$ are the $AddRoundKey = d_1 \oplus k_1$ and then the S-Box (substitution box) function which is applied to the result (see Figure 2.9). There are more operations in AES that could be targeted but in this study we specifically target the S-Box operation.

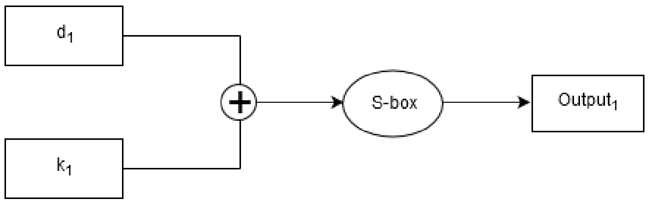


Figure 2.9: AES-128 encryption of 1-byte data (d_1) with 1-byte secret key (k_1) (SubBytes Operation)

As mentioned above, each different data D the attacker is sending to the device, results in a different power trace measured by the oscilloscope. In figure 2.10 we can see a power trace t that was measured when data D_t was sent to the target. In this example, we can observe a pattern repeating 16 times in the power trace. This pattern is the encryption process for each byte.

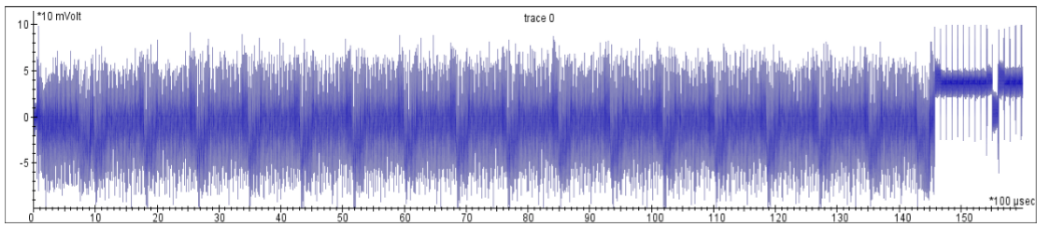


Figure 2.10: Power trace t associated to the AES-128 encryption of each byte of the data $D_t \in \{d_1^t, ..., d_{16}^t\}$

For the sake of example's simplicity, we will focus on the first data byte d_1 and first key byte

k_1 for each data D sent. Following the same method, we isolate roughly the first pattern of the trace, which should contain the power consumption values connected to the encryption of d_1. The resulting Sbox-out of the d_1 encryption, is then incorporated in a leakage model. In our case, we will use the Hamming-weight values of each Sbox-out, as Hamming-weight is an assumed, common leakage model. In short, Hamming-weight is the amount of 1 in a byte, so the range of possible values is from 0 to 8. Each Sbox-out thus has a specific Hamming-weight (HW).

If $d_1^{t_1} \in D_{t_1}$ is the first data-byte of the data $D_{t_1} \in \{d_1^{t_1}, ...d_{16}^{t_1}\}$ sent to the cryptographic device resulting to the first power trace t_1 measured, then for each power trace $(t_1, ...t_n)$ resulted from sending n different $D \in \{D_{t_1}, ..., D_{t_n}\}$, for the specific first key-byte $k_1 \in K_{16}$ of the target we have:

$$\begin{gathered} HW(Sbox(d_1^{t_1} \oplus k_1)) \\ HW(Sbox(d_1^{t_2} \oplus k_1)) \\ \vdots \\ HW(Sbox(d_1^{t_n} \oplus k_1)) \end{gathered}$$

With $d_1^{t_1}, ..., d_1^{t_n}$ known and the encryption key fixed in the target device, the attacker calculates the HW for each of these key-byte values $kv_i \in \{0, ..., 255\}$ with $i \in \{1, ..., 16\}$. This results in 256 different leakages (L) for each of the possible key-byte values, where each leakage L consists of Hamming-weight values for a specific d_1^t $(L_{kv_i} = [HW_0, ..., HW_{256}])$.

In order to find which key-byte value is the best candidate, the attacker proceeds to calculate the correlation of each HW value with the power trace t generated by sending one of the d_1 data-bytes (Hamming-weight values are correlated to the power consumption of the device). If n is the number of traces and s the number of samples, then we calculate the correlation between the power traces and the Hamming-weight values for the first key-byte $k_1 \in K_{16}$. Specifically, for the $kv_1 = 0$ we calculate the correlation of the L_0 vector with each of the columns in $T_{n,s}$:

$$T_{n,s} = \begin{bmatrix} t_{1,1} & t_{1,2} & \cdots & t_{1,s} \\ t_{2,1} & t_{2,2} & \cdots & t_{2,s} \\ \vdots & \vdots & \ddots & \vdots \\ t_{n,1} & t_{n,2} & \cdots & t_{n,s} \end{bmatrix}, \& L_0 = \begin{bmatrix} HW_0^{d_1^{t_1}} \\ HW_0^{d_1^{t_2}} \\ \vdots \\ HW_0^{d_1^{tn}} \end{bmatrix}$$

The L_{kv_i} vector that its $\{HW_{kv_i}^{d_1^{t_1}}, ..., HW_{kv_i}^{d_1^{tn}}\}$ values had the highest correlation with the power consumption values in $T_{n,s}$, indicate the key-value that is the best candidate for the first key-byte $k_1 \in K_{16}$. The same procedure is taking place to extract the key-byte values of the rest key bytes $k_i \in K_{16}$.

2.3. Using Deep Learning for Side-Channel Attacks

Starting this work, there were only two available research papers on the same topic; the works of [26] and [6]. Both of the publications became a strong indicator of the path we would follow in our study.

The two papers showcase two different studies regarding Deep Learning and Side-Channel Attacks. In [26], the research team introduces the concept of Deep Learning and how SCA could benefit from it. After an introduction on both fields, the team presents a comparison of 7 attacks on a publicly available dataset, the DPA contest v2. More specifically, they compare the key recovery results of an Autoencoder, a CNN, a Multilayered Perceptron with PCA, a Multilayered Perceptron without PCA, a Random Forest and an LSTM alongside a Template Attack. The results showed that the CNN model achieved the best key recovery results, significantly outperforming all the other techniques. The research was published in SPACE (Security, Privacy, and Applied Cryptography Engineering) Conference on 2017 and they included the CNN architecture in the publication.

The work in [6], was conducted by the same research team and it was published in CHES (Cryptographic Hardware and Embedded Systems) Conference on 2017. In this work, the researchers focused on using a CNN architecture to classify a proprietary dataset and they studied the influence of data augmentation through adding jitter to the power consumption traces. The network used in this paper is different from the one they used in the previous publication but the information that was shared about it was not sufficient to reproduce the model in our experiments. The CNN architecture they used, achieved good classification performance on the augmented dataset and utilized more layers compared to the CNN in [26].

While examining the above works, we noticed that they both lacked the reasoning behind the use of the specific architectures as well as a proper data science approach to the topic. The first paper didn't include the classification performance of the models, which fails to indicate a true comparison of the classification models. Also, on the second paper, the team used only a proprietary dataset which makes it difficult to replicate the results, especially since the exact CNN architecture they used, was not shared in the paper. Adding to the above, the use of jitter to augment the data in that publication, lacked a sufficient reasoning, following mostly the trends of data augmentation in image classification.

Since the reasoning behind the selection of the specific CNN architectures was not sufficient, we proceeded in experimenting with a variety of different CNN models. As explained in the following sections, 3.1 and 3.3.2, the work of [4] was catalytic in the above process. In this work, the research team is suggesting a simple optimization technique for the hyper-parameters of the ANNs. As ANNs are complex classification functions, they utilize a large number of parameters as described in section 2.1. By using an optimization technique, the burden of finding good values for the ANN's hyper-parameters is alleviated, and a more reproducible and mathematically sound method is used instead. In this work they show that Random Search is a more efficient technique compared to Grid Search. That is because in a high dimensional configuration space, not all the hyper-parameters are important for different datasets, as shown in their study. This makes Grid Search a poor optimization choice and they suggest that Random Search is sufficient to become a benchmark for future optimization techniques.

The above works were a primary influence in our work. Apart from these, research on CNN and signal processing were important to draw inspiration from or learn more about a specific topic of the project. Those studies though were not as catalytic as the ones above, as they didn't affect our study's goals directly. They were rather used in aspects such as the types of hyper-parameters we used, data preprocessing techniques that we applied or to generally learn more about the data classification process for the task at hand.

Our classification task could be summarized as follows:

- It is a Multiclass classification of time-series data using Convolutional Neural Networks
- Each power consumption trace is associated with a specific label
- The data labels are generated from the traces using Hamming weight model (labels between 0 and 8)
- Each classification task is focused on the Hamming weight values for one byte of the encryption key

3 Research Methodology

3.1. Approaching the Research Problem

Due to the limited research on the field, we based our work mainly on two publications. In [26], the research team conducted a comparison of different DL techniques against standard types of SCAs and found that a CNN they created achieved the best performance in key retrieval. This outcome combined with their second publication [6] where they demonstrated that certain data manipulations can increase the performance of CNN, pushed our research towards CNNs.

The CNN architectures used in those studies, where either vaguely described or they were relatively simple architectures. The networks were shallow and the reasoning behind some of the design choices, was not described at the studies. This made us experiment on CNN architectures and link our design choices with research outcomes in related fields. First we tried already established networks such as LeNet [22] and AlexNet [21], both re-purposed for 1-D data. The results of those networks in some preliminary tests, were poor. That was not a surprise as those networks were optimized for image classification tasks. For our next step, we sought performance gains in data manipulation, that were used in audio signals' classification literature, as audio and power signals share structural characteristics. We quickly found though that plain implementations of Fast Fourier Transform (FFT) and Short-Term Fourier Transform (STFT) were not particularly useful, having minimal impact in networks' performances.

Thus we focused our research on finding CNN architectures that could achieve satisfying classifying performance, using minimal data pre-processing. To achieve that we had to test several different CNN architectures, in order to find the most fitting for each of the target datasets. Since this task would have needed a lot of manual hyper-parameter tuning, it would have rendered the assignment non-feasible in the time frame of this work. In order to automate the task, we referred to literature for hyper-parameter tuning and optimization. The system that we wanted to build, would have to compete in speed and performance with already established types of SCAs. The time needed for Grid Search though (as discussed in section 2.3), would not make the system competitive. Random Search [4] though, provided us with a technique which could produce architectures with good performance, in a relative shorter time than Grid Search or Genetic Algorithms [28] would. Our results using this optimization algorithm matched the results in literature [5] in terms of time and performance efficiency.

3.2. Research Goals

With the task at hand, combined with our design patterns and constraints, we formulated our research questions early in our work. The desired outcome was to test several CNN architectures in a number of different datasets and study their performance and characteristics. Then we would compare them with literature's models and other Machine Learning algorithms in order to investigate their performance, which would lead us in valuable insights on the side-channel data and the capabilities of CNNs classifying them.

In an attempt to quantify the above research questions, we planned the following research goals, on which we based all of our experiments:

1. Measure the performance of Convolutional Neural Networks across different side-channel datasets.
2. Identify performance differences between different datasets' sizes.
3. Identify any common structural elements between network topologies of the highest performing models across the datasets.
4. Analyze the hyper-parameter search space for the side-channel classification problem.
5. Research the role of samples' topology, in the side-channel classification problem.

We will address each goal separately below.

Regarding our first goal, it was our main focus in this essay. We would like this work to be a guide for future researchers who will work on the topic, so we conducted a series of experiments on CNNs and side-channel data, to cover as many scenarios as possible. This resulted in insights which we believe they are valuable for the research in the field as they show the average expected performance when using CNN on side-channel data.

A general rule of thumb regarding the amount of data needed for ANN classification is "the more, the better". In the case of SCA though, an attacker might have access to a limited amount of power consumption traces. Common practices in SCAs so far, show that the more difficult an encryption implementation is, the more traces are needed. Through our experiments, we wanted to identify dataset sizes where the CNNs perform the best and the point where any additional data won't affect their performance any more. Thus, we split each dataset in different sizes and classified each subset separately.

As the datasets are sharing a lot of structural similarities, we expect to find commonalities between the different optimized CNN architectures. These insights would also help to identify common structural elements and their association with the type of data that were classified. To contribute more on this claim, the number of layers in a CNN or the size of its kernels, could be affected by the type of data at hand. Thus, by studying the classification models that reached high performance, we sought to identify any common structural characteristics.

When we first implemented Random Search optimization, we established a search space which was large enough, in order to include many possible architectures. That was our approach on the problem since the impact of each hyper-parameter to the classification performance on side-channel data, was not known in the beginning of this work. Once we conducted all the experiments, we had narrowed the space down to ranges which seemed to produce the best results. This, narrowed down search space, could be used in future research to help with further optimizing CNN architectures.

Our last research goal is inspired by CNN's ability to find associations between features, based on the data topology. This ability is crucial in object identification problems and is more obvious when classifying image data. In our case we didn't know if the topology of the power consumption samples is indeed a valuable information, thus we conducted extra experiments for this inquiry.

All of our research goals led to a publication [34], as our performance comparisons and insights on CNNs, were novel in the field.

3.3. System Design

The design and development of the Deep Learning system was the engineering part of this work. The system reflects the needs of our study and was developed using core software developing principles so the final product can have expandable structure and easily maintainable codebase. It supports automatic CNN creation based on 13 different hyper parameters and an optimization which can produce an optimized model for a specific dataset. Through the design of the system, we didn't try to innovate on CNN architectural elements and propose new techniques. We rather studied the characteristics of each generated model alongside their performance and compared them with the characteristics of the target datasets. Through this process we identified some common patterns on CNN architectures as well as we found elements of the datasets that were greatly influencing the classification performance.

3.3.1. Implementation Details and Frameworks

As part of the internship at Riscure, the first version of the Deep Learning system was developed in Java using the Deeplearning4j framework. The system requirements were to produce a fast and well performing system which had key recovering capabilities. The codebase had to be easily maintainable for future use by the company and the development had to take place in a Windows environment.

Since the end of the internship, we focused on further improving the system without the above constraints. Moving the development to Python and to the more established Deep Learning frameworks, Keras and Tensorflow, made the system capable achieving better performance. Key recovery though was dropped in favour of further experimenting on the classification capabilities of CNNs.

In the following sections we explain all the advantages and disadvantages of each development environment, while going into technical details regarding their differences.

GPU Acceleration

The usage of GPUs (Graphics Processing Unit) for Deep Learning methods is common in our days. As computational requirements of algorithms rise, researchers need capable hardware to run their implementations in decent time frames. Deep Learning frameworks, integrate their functionality with GPU computational libraries in order to harness the high parallelism that GPU cores offer.

In both developed systems, CUDA parallel computing platform was used in order to increase the computational performance by harnessing the processing power of the GPUs installed in the development systems. CUDA is a C/C++ library designed and developed by NVIDIA and is compatible with highly computationaly capable NVIDIA GPUs. Mainly two different GPU cards were used, GeForce 1080Ti on the Windows machine and GeForce 1050Ti on the Ubuntu machine. Alongside CUDA, the CUDA Deep Neural Network library (cudNN) was used which provides highly tuned implementations of standard neural network routines such as convolution and pooling. Both of the deep learning frameworks are compatible with CUDA and cudNN libraries.

Deeplearning4j

Deeplearning4j [44] is an open source Deep Learning framework, developed and maintained mainly by a core team from Skymind company. The company develops other open sourced libraries as well which accompany the Deeplearning4j framework. These include ND4j which brings n-dimensional array manipulation in Java, DataVec for data vectorization and JavaCPP for interfacing Java with C++ (mainly for Deeplearning4j-CUDA compatibility).

The developers of the framework are active in their channel in Gitter and help with troubleshooting. At the time of writing, it is in pre version 1.0 (0.9.1 is the stable version). As such, the framework still has bugs which the community and developers try to fix.

Deeplearning4j has features which make it competitive, such as large scale distributed Deep Learning network using Hadoop. Also it offers several performance optimization features such as scheduling or disabling the Java garbage collector and iterating through datasets asynchronously through their libraries.

While developing with this framework, the main advantage was apparent. The Java language helps for a more structured and easily maintainable code while benefitting from the multithreading and immutability features of the language. Although these are definitely advantages from a software developer's point of view, the language and the framework don't appear to help the nature of Deep Learning research. In Deep Learning, researchers need to quickly test architectures and easily manipulate datasets for experimentation, both of which are hindered in this case. For example, creating n-dimensional arrays from CSV data and passing them to CUDA for computations were difficult as this process is not intuitive in Java and creating your own data loaders was impended by obfuscations that were implemented in the framework.

As the framework was still in a pre-release version while developing, we faced bugs that were impossible to fix or needed a lot of system tinkering to resolve. Framework's documentation wasn't helpful in these situations and the user base is relatively small compared to other Deep Learning frameworks, making debugging even more difficult.

Example of such problems was: computer crashes during training, with no log output to pinpoint the error. That happened when hyper-parameter values were higher than a certain value and resulted in memory allocation issues due to Java-CUDA incompatibilities. The example was not easily reproducible from the framework developers' side

Lastly, the performance results of this framework's models were found to be poorer compared to the later Keras' models performance results. Also, there was inconsistency in the accuracy and loss results between the models running in CPU and GPU, while using Deeplearning4j. This was a particularly anomalous incident as there shouldn't be any difference between the results of identical models, independently from the hardware used for computations.

Keras

Keras [8] is a library for high-level design of neural networks in Python. Its API (Application Programming Interface) supports Tensorflow [2] which was used in our system, as well as other frameworks for backends. It allows fast prototyping and supports GPU acceleration libraries (CUDA, cudNN) as mentioned before. Keras at the time of developing, was in version 2.1.2 and it is a well established Deep Learning library, benefiting from an extensive documentation and community support.

Transferring the Deep Learning system from Java to Python was easily achievable due to the plethora of Python libraries and the more intuitive way of programming in Python. Due to the excessive documentation of Python's libraries, debugging the system and adjusting it for the multiple datasets, was an easier task compared to the developing process of the Java system. Finally, while using Keras and Tensorflow, we were able to choose from a wider variety of functions and algorithms for our Deep Learning system, compared to the Deeplearning4j system.

Comparing the performance of the two systems, we found that in many cases, the produced models of the Python system needed less epochs to train on a dataset and attained better metrics' performance when compared to the Java system. In section 5.3 we explicitly show the differences in performance. These differences alongside the ease of development in Python, urge us to drop the development of the Java system in favour of the Python one.

3.3.2. Using Hyper-parameter Optimization

As explained in section 3.1, we implemented a Random Search optimization algorithm in our Deep Learning system. Through this search we were able to produce several different architectures per dataset and compare the models in order to choose the one with the best results. In order to increase performance, we narrowed down the space each time, close to the model with best results. This made the Random Search more efficient every time we re-iterated the search for a dataset, as the hyper-parameter search was narrowed down to a local optimum.

Using this optimization technique, we significantly simplified the building process of neural networks. Through the Random Search we implemented, a possible user could set the hyper-parameter ranges and could let the system run and produce multiple models. By the end of the process, the system would produce scatter plots for each hyper-parameter alongside the accuracy that was achieved by the model with the specific value at that specific hyper-parameter. Understanding that the optimization problem is multivariate, shows that a graph of a single hyper-parameter cannot lead to any meaningful insights. The clusters of models per plot though, can show a general trend per hyper-parameter and an overview comparison between the general trends in each plot can be extremely helpful. The researcher can find the overall ranges which yield better results than others per hyper-parameter and also find a hyper-parameter's contribution to the model's performance (e.g. were the performance results saturated or did they concentrated on high values for a specific range).

Below we incorporated a general description of the algorithm, with the steps that were followed during the optimization:

- Create a set of hyper-parameters for the Random Search
- Set hyper-parameter value ranges
- Leave the system generate models for n number of experiments on the target dataset

- The user interrupts the search or the system finishes a predetermined number of experiments
- Generate plots of models' performance per each hyper-parameter
- Identify trends on performance across the different hyper-parameters
- Narrow the search space by:
  - Eliminating the hyper-parameters that have small influence in the network's performance
  - Setting hyper-parameters' ranges around the values of the best performed models
- Repeat process until the differences in results saturate

During the training of all networks, we used Early Stopping to further avoid overfitting by monitoring the loss on the validation set [13]. Thus every training session is interrupted before reaching high accuracy on the training datasets. To help the network increase its accuracy on the validation set, we use a learning rate scheduler to decrease the learning rate depending on the loss on the validation set. Finally, backpropagation was applied in all the models to help optimizing the weights in the networks.

In the following table, we show the initial hyper-parameter ranges. In a later section we show the final ranges as they were formulated after many applications of the Random Search

Table 3.1: Initial Hyper-parameter Ranges for Random Search Optimization

Hyper-Parameter	Value Range	Constraints
Convolutional Kernel	$k_{conv} \in [2,50]$	-
Pooling Kernel	$k_{pool} \in [2,50]$	$k_{pool} \leq k_{conv}$
Stride	$s \in [1,10]$	in pooling layers, $s = k_{pool} - 1$
Number of Convolutional Layers	$layers_{conv} \in [1,8]$	-
Number of Pooling Layers	$layers_{pool} \in [0,8]$	$layers_{pool} \leq layers_{conv}$
Number of Fully-Connected Layers	$layers_{fc} \in [0,4]$	-
Initial number of Activation Maps	$a \in [2,256]$	follows geometric progression with ratio r = 2, for the number of $layers_{conv}$
Initial number of Neurons	$n \in [64,2048]$	follows geometric progression with ratio $r = 2$, for the number of $layers_{fc}$
Convolutional Layer Dropout	$drop_{conv} \in [0,0.5]$	-
Fully-Connected Layer Dropout	$drop_{fc} \in [0,0.5]$	-
Learning Rate	$l \in [0.001,0.02]$	a learning rate scheduler was applied
Activation Function	ReLU, ELU, SELU, LeakyReLU, PReLU	the same for all layers except the last which uses Softmax
Optimization Algorithm	Adam, Adamax, NAdam, Adadelta, Adagrad, SGD, RMSProp	-

3.3.3. System Overview

The final system that was developed throughout this work, was able to automatically initialize a Random Search and classification on a given dataset, as well as running the designed experiments. The user's input is limited on selecting the target dataset and whether they want to run experiments or Random Search and classification. The system will then proceed to generate models accordingly and run the appropriate procedures. By automating the processes, we wanted to achieve our methods and experiments to be the same across all scenarios, to ensure reproducibility in our experiments and the same conditions for all the models used in our study.

In Figure 3.1, we present a flowchart of the main functionality and the order of the processes in our system. The user starts by choosing the target dataset and then proceeds to select whether they want to run the experimental setup or a Random Search on the target dataset. The system then proceeds to run the processes for the chosen scenario and exports the results accordingly.

In Figure 3.2, we show in more details, the Random Search and the Deep Learning processes through a block diagram. It shows how the main methods interact in the system, alongside the architectural blocks they belong to. More details on the models used in the study, can be found in Section 4.3.

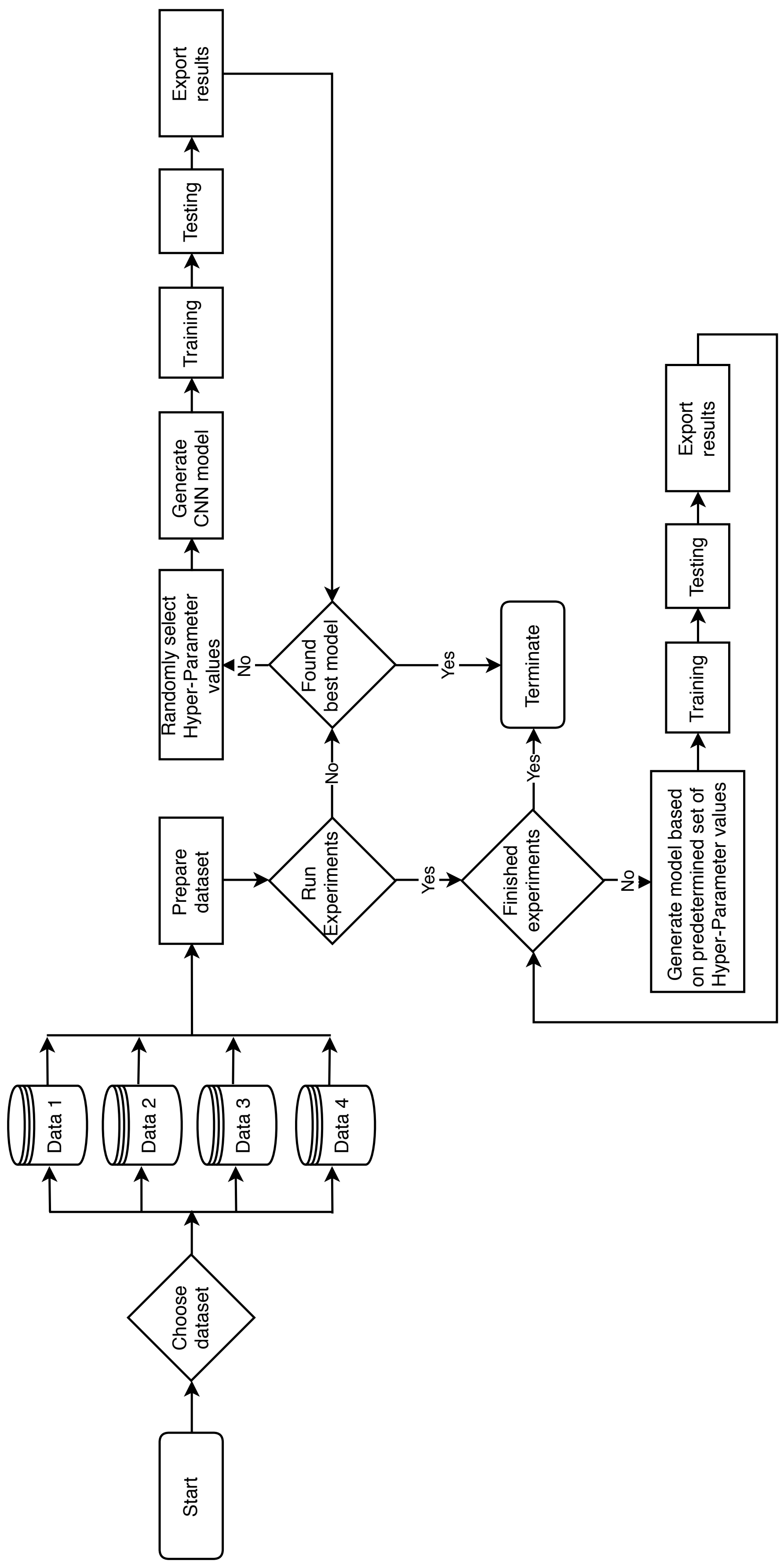


Figure 3.1: Deep Learning System's Flowchart

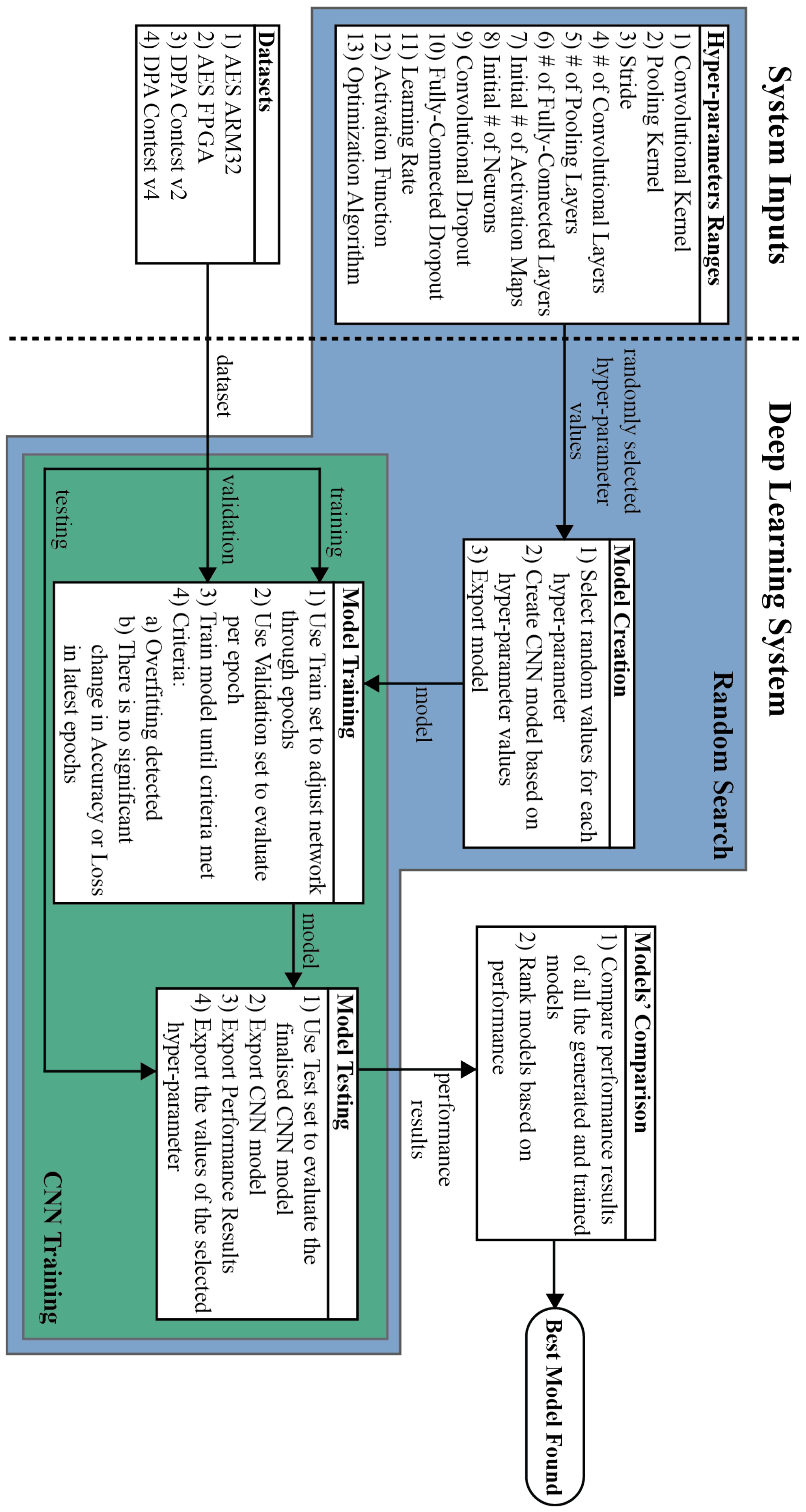

System Inputs
Deep Learning System
Hyper-parameters Ranges
1) Convolutional Kernel
2) Pooling Kernel
3) Stride
4) # of Convolutional Layers
5) # of Pooling Layers
6) # of Fully-Connected Layers
7) Initial # of Activation Maps
8) Initial # of Neurons
9) Convolutional Dropout
10) Fully-Connected Dropout
11) Learning Rate
12) Activation Function
13) Optimization Algorithm
Datasets
1) AES ARM32
2) AES FPGA
3) DPA Contest v2
4) DPA Contest v4
randomly selected hyper-parameter values
dataset
Random Search
Model Creation
1) Select random values for each hyper-parameter
2) Create CNN model based on hyper-parameter values
3) Export model
model
training
validation
testing
Model Training
1) Use Train set to adjust network through epochs
2) Use Validation set to evaluate per epoch
3) Train model until criteria met
4) Criteria:
a) Overfitting detected
b) There is no significant change in Accuracy or Loss in latest epochs
model
Model Testing
1) Use Test set to evaluate the finalised CNN model
2) Export CNN model
3) Export Performance Results
4) Export the values of the selected hyper-parameter
CNN Training
performance results
Models' Comparison
1) Compare performance results of all the generated and trained models
2) Rank models based on performance
Best Model Found

Figure 3.2: Deep Learning System's Block Diagram

4 Experimental Setup

The experiments we conducted were in line with our research questions, as they were described in the section 3.2. Following the topics we wanted to research in this study, we ran multiple Random Searches to optimize our CNN models and we experimented with different dataset sizes and types. We compared the results with our own implementation of a CNN model from the related literature, alongside ML algorithms in order to benchmark our generated models. Finally, we conducted a set of experiments to value the influence of data topology in this classification problem.

All these experiments gave us valuable insights on how CNNs perform on side-channel data and how CNNs compare to ML techniques. In the following sections we break down the design process of the experiments and we discuss their details. This chapter is essential for our discussion on the results, which takes place in the next chapter.

4.1. Designing the Experiments

As explained in the section 3.2, the main focus of our research is to measure the performance of CNN on multiple side-channel datasets. Thus, we used four main datasets in our work, 2 proprietary and 2 publicly available datasets. These datasets have different characteristics and differ in their respective implementations. As we value the reproducibility of our experiments, we conducted the main set of experiments on the public datasets. We used Random Search optimization multiple times, optimizing each time the hyper-parameter search space as discussed in section 3.3.2. Through the search, we created an optimal CNN for both public datasets which, for ease of reference, we will call it SCANet (Side-Channel Attack Network) for the rest of the study. The public datasets were split into subsets of different sizes, to study our second research goal of identifying the influence of different dataset sizes on CNN classification. Once we collected the results of those experiments, we implemented the model from [26] alongside several ML algorithms, in order to compare the performance of our optimized model with other classification models. Finally, we conducting experiments on the whole set of the proprietary datasets, using SCANet and the literature's model. In order to identify if those datasets needed a better CNN architecture, we ran a Random Search optimization on both datasets and we compared the performance of the new optimized models with SCANet and the literature's model.

Working towards our last research goal, we conducted a set of experiments on the public datasets. These experiments were used to give us an indication about the importance of the signals' sample topology on this classification problem. We randomly shuffled the samples of each power consumption signal and we compared the classification results against the performance on the original traces.

More specifically, the methodology of our experiments is presented below:

- We selected 4 datasets, two publicly available and two proprietary
- We used Random Search to find the most optimized model for the public datasets

- We split the public datasets on subsets of different dataset sizes
- The optimized CNN and the literature model were used to classify the datasets
- We implemented Machine Learning algorithms and compared the results with the models above
- We conducted classification with the optimized CNN and the literature model on the full size of the proprietary datasets
- We used Random Search to find if a more optimized CNN architecture exists for the proprietary datasets
- We compared performance of the optimized CNN model on the original and the randomly shuffled samples of the public datasets

4.2. Datasets' Structures

All the datasets used in our classification task, were time-series data. They represent power consumption of a cryptographic device and their labels were generated based on a Hamming weight model (as described in section 2.2.3).

The datasets used are the following:

- AES ARM32: An AES-128 dataset of power consumption, provided by Riscure B.V.
- AES FPGA: An AES-128 dataset of power consumption
- DPA contest v2: An open sourced AES-128 dataset of power consumption, provided by VLSI research group in COMELEC, Télécom ParisTech University [45]
- DPA contest v4: An open sourced AES-128 dataset of power consumption, provided by VLSI research group in COMELEC, Télécom ParisTech University [46]

As a first step, we visualized the clustering of the datasets using t-SNE clustering [25], so we can have an insight on the data structures. In each of the following subsections, we include the clustering visualizations, alongside a more in depth discussion about each dataset's features. We applied minimal data pre-processing, which includes Normalization and Standardization of the power consumption traces for each dataset. Both techniques were applied using the Scikit-Learn Python library [32] for the Python system while in the Java system we only applied Normalization, using the framework's functionalities. In the Python implementation, the traces had their magnitude standardized and then normalized using l2 normalization [16]. We include graphs of how these techniques influenced the power traces for each dataset accordingly. Lastly, all datasets were in CSV format.

4.2.1. AES ARM32

AES ARM32 is an AES-128 hardware implementation and it was the main dataset used during the internship in Riscure BV. The dataset contains 10,000 traces of 1,000 samples each. It is a simple side-channel dataset, where the target didn't have any sophisticated countermeasure.

4.2.2. AES FPGA

This dataset is an AES-128 FPGA implementation. The leakage model is described below:

$$Y(k^*) = \underbrace{C_{b_1} \oplus k^*}_{\text{previous register value}} \oplus \underbrace{C_{b_2}}_{\text{ciphertext byte}}, \tag{4.1}$$

The total number of power consumption traces was 10,000 with 245 samples each.

4.2.3. DPA contest v2

DPA contest v2, is one of the publicly available datasets. It is an AES-128 hardware implementation on an FPGA. The dataset has 100,000 traces with 3,253 samples each. Note these measurements are relatively noisy and the resulting model-based signal-to-noise ratio

$$SNR = \frac{var(signal)}{var(noise)} = \frac{var(y(t,k^*))}{var(x - y(t,k^*))} \tag{4.2}$$

lies between 0.0069 and 0.0096.

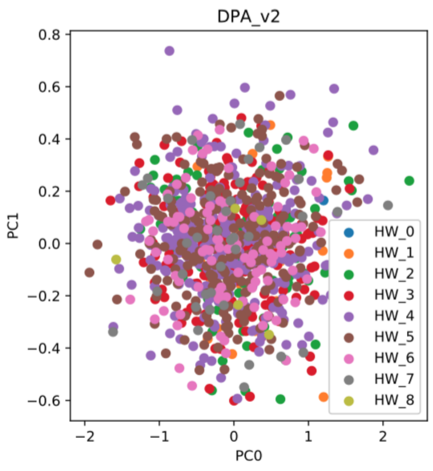


Figure 4.1: DPA contest v2 data clustering

4.2.4. DPA contest v4

DPA contest v4, is a software implementation of a masked AES. The mask is known in this dataset, thus making it equivalent to an unprotected scenario. The SNR here is much higher and lies between 0.1188 and 5.8577.

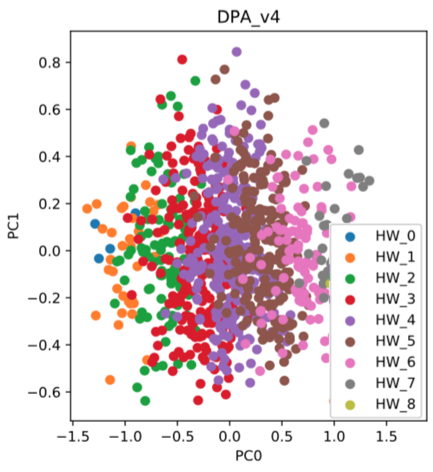


Figure 4.2: DPA contest v4 data clustering

4.3. Network Architectures

Due to the nature of the research (explained in the section 3.2), we explored various different architectures of CNN during the research. From our implementation of Random Search (more details in the section 3.3.2), we found 3 in total optimized CNN architectures. One architecture was optimized for the DPA contest v2 and v4 datasets, which we named SCANet for ease of reference, and one architecture for each of the proprietary datasets.

SCANet is composed of 4 convolutional layers and 4 pooling layers in between, followed by the classification layer. All convolutional layers use kernel size of 6 and stride 1 creating a number of activation maps for each layer. The number of activation maps increases per layer, following a geometric progression with initial value $a = 16$ and a ratio $r = 2$ (16, 32, 64, 128). The number of activation maps was optimized for GPU training. For pooling we use Average Pooling on the first pooling layer and Max Pooling on the rest, using kernel of size 4 and stride 1. The convolutional layers used "Scaled Exponential Linear Unit" (SELU) activation function, an activation function which induces self-normalizing properties and it was first introduced by [19]. In the classification layer, we use Softmax activation function combined with the Categorical Cross Entropy loss function. Finally, for regularization we use dropout on convolutional and fully connected layers while on the classification layer we use an activity l2 regularizer. These regularization techniques help to avoid overfitting on the training set, which in turn help lower the bias of the model.

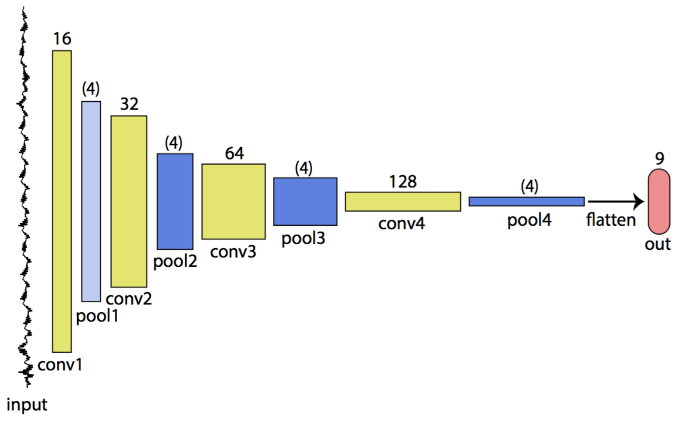


Figure 4.3: SCANet architecture

SCANet is being tested on the other datasets as well for comparison reasons. Other networks that were used in our work are: our own adaptation of the CNN used in [26] (for ease of reference we call it SPACE) and an optimized network for each proprietary dataset. As mentioned before, SCANet is the optimized network for DPA contest v2 and DPA contest v4, thus we don't compare results with another optimized network on those datasets.

Recapping, we compare results of the following architectures:

- SCANet
- SPACE
- "Optimal CNN" (wherever applicable)

Table 4.1: SCANet architecture.

Layer	Weight Shape	Sub-Sampling	Activation
conv(1)	1 x 16 x 6	2	SELU
average-pool(1)	-	(4), 3	-
conv(2)	1 x 32 x 6	2	SELU
max-pool(2)	-	(4), 3	-
conv(3)	1 x 64 x 6	2	SELU
max-pool(3)	-	(4), 3	-
conv(4)	1 x 128 x 6	2	SELU
max-pool(4)	-	(4), 3	-
fc-output	384 x 9	-	Softmax

Table 4.2: SPACE architecture.

Layer	Weight Shape	Sub-Sampling	Activation
conv(1)	1 x 8 x 16	1	ReLU
average-pool(1)	-	(2), 1	-
conv(2)	1 x 8 x 8	1	tanh
fc-output	384 x 9	-	Softmax

Table 4.3: Riscure AES Optimized architecture.

Layer	Weight Shape	Sub-Sampling	Activation
conv(1)	1 x 8 x 4	1	ELU
average-pool(1)	-	(4), 3	-
conv(2)	1 x 16 x 4	1	ELU
average-pool(1)	-	(4), 3	-
fc	1 x 256	-	tanh
fc-output	256 x 9	-	Softmax

Table 4.4: Proprietary AES Optimized architecture.

Layer	Weight Shape	Sub-Sampling	Activation
conv(1)	1 x 8 x 7	1	ELU
average-pool(1)	-	(4), 3	-
conv(2)	1 x 16 x 7	1	ELU
fc	1 x 256	-	tanh
fc-output	256 x 9	-	Softmax

4.4. Detailed Overview of the Experiments

The experiments were conducted in two main parts: one during an internship in Riscure B.V. and one during the period after. During the internship, the experiments were based on the company's needs and they are not following much the experimental methodology of the rest of the study.

On the public datasets of DPA contest v2 and v4, we performed the main body of our experiments. We split the datasets into subsets of different sizes and through Random Search (as explained in section 3.3.2) we produced an optimized CNN model, the aforementioned, SCANet. We then compared its results with the performance of our implementation of SPACE architecture. Following our experimental design described in section 4.1, we also classified these datasets using Machine Learning algorithms. For this specific classification though, we performed Pearson's Correlation Coefficient on the data in order to reduce their dimensionality, as the feature space would be vast for the ML algorithms otherwise. In the end, we compared the performance of SCANet and those ML algorithms on classifying the 50 most relevant features per power trace on the DPA contest v2 and v4 datasets.

On the same datasets, we finally performed experiments to study the importance of samples' topology of the power consumption traces. We randomly shuffled the samples of each signal of the DPA contest v2 and v4 datasets, while keeping intact the association between the traces and their labels. These experiments were inspired by the fact that the importance of the samples' topology is not that intuitive in side channel data, as it might be in audio or image data. This made us question the importance of the samples' topology on our datasets; if the topology mattered, then obviously techniques such as CNN, would be superior to others who don't retain the samples' topology. In the case though that the classification results did not differ much, then it would be an indicator that samples' topology might not be that important. It is a simple set of experiments that while by themselves, cannot prove our research goal, they can nevertheless act as an indicator for future research on the topic.

The split we performed on the DPA contest v2 and v4, followed the subset pattern bellow:

1. Subset 1: 1,000 traces
2. Subset 2: 10,000 traces
3. Subset 3: 50,000 traces
4. Subset 4: 100,000 traces

beginning from the first trace to the full size of the dataset's size category.

Continuing our study on the rest of the datasets, we conducted experiments on the two available proprietary datasets. Regarding the AES ARM32 dataset, we showcase two different sets of experiments. The first set is the performance achieved by an optimized CNN model, using the Deeplearning4j framework. We then showcase the performance results of SCANet, SPACE and an optimized CNN we produced after running Random Search optimization with the Python system. This demonstrates the differences in performance achieved between the two frameworks and provides a further insight into the classification capabilities of different CNN architectures on different datasets.

For the AES FPGA dataset, we followed the same methodology we applied with the AES ARM32. The main difference is that all experiments we performed using the Python framework. In this case, we ran a Random Search for the dataset and compared the results of the optimized model with the performance of SCANet and SPACE architectures.

Finally, following the Machine Learning practices, after shuffling the traces per dataset, we split each of them into Training, Validation and Testing sets. In our experiments, we chose to use ratios of 0.65, 0.2 and 0.15 respectively.

All these experiments, give us an intuition on how the same CNN architectures perform differently in each side channel dataset. Thus we can study the architectural differences of those models and identify common elements, if any exist. Furthermore, they enable us to examine if we actually benefit from using complex classification models such as CNNs, when simpler and less computationally expensive ML techniques exist. Finally, through the random sample shuffling, we learn more about the characteristics of the side channel datasets, by studying the importance of their samples' topology. We believe that through these experiments we overall gain some important insights on CNN classification capabilities on side channel data, while simultaneously learning more about the very nature of the data we try to classify.

To summarize the types of experiments, we have the following list which references the experiments' steps, based on the dataset used:

- DPA contest v2 and v4:
  - Performed Random Search using Keras/Tensorflow and produced SCANet
  - Split the datasets into different size subsets
  - Compared classification results of SCANet with SPACE and ML algorithms
  - Shuffled the samples per power trace and compared the new performance of SCANet with the performance on the original signals
- AES ARM32:
  - Performed Random Search using Deeplearning4j and produced an optimized architecture
  - Performed Random Search using Keras/Tensorflow and produced an optimized architecture
  - Compared results between the optimized architectures of the two frameworks
  - Compared results between the Keras/Tensorflow optimized architecture with SCANet and SPACE
- AES FPGA:
  - Performed Random Search using Keras/Tensorflow and produced an optimized architecture
  - Compared results between the Keras/Tensorflow optimized architecture with SCANet and SPACE

5 Results

In this section we go through the results of our experiments. The experiments were designed in regards to our research goals as discussed in the previous chapter 4. We first show the results on the public datasets, DPA contest v2 and v4. These experiments were the primary set of experiments that bear the most insights on CNNs' performance on side-channel data. Also, as they are publicly available, these experiments can be reproduced and researchers can compare their results with ours.

We showcase the results, following the categories in the section 4.4. For the public datasets, we show a performance comparison between our optimized CNN (SCANet) and the literature architecture (SPACE), in each size subset of the datasets. We proceed to present the results of ML algorithms compared to SCANet and we finally show the performance of SCANet on the original power consumption signals in contrast to the random sample shuffling.

We then proceed on the proprietary datasets where we first present the results on AES ARM32. After showing a performance comparison between the different frameworks, we present the results of SCANet, SPACE and an optimized CNN that we found when we applied Random Search on AES ARM32. Following the same methodology, we showcase a performance comparison of SCANet, SPACE and the corresponding optimized CNN for AES FPGA.

The main metrics we chose to use in order to measure the classification performance, are: accuracy and loss. It was found that the accuracy performance results, didn't differ significantly from recall, precision or f1 score. When a model would achieve high accuracy, the rest of those metrics were also satisfyingly good.

5.1. Results on DPA contest v2

The DPA contest v2 dataset (DPA v2), was one of the most difficult to classify. Our optimized network, SCANet, couldn't achieve better results than what appears to be random classification. As explained in section 4.2, one of the reasons the classification was hindered, was the high overlap between the traces of different classes. The data didn't seem to be easily distinguishable by the CNNs, as even though we run Random Search for DPA v2, none of the generated models had better performance than SCANet. Thus, regardless of SCANet being optimized for DPA contest v4, we used it as our proposed model, to classify and compare the results on DPA v2.

5.1.1. Comparison between CNN Architectures

A first sight on the results of Table 5.1, shows that both SCANet and SPACE achieve near random performance (for a 9 class problem). When going through the individual graphs of Accuracy and Loss though, we can get more insights on the networks' behaviour. Overall, SPACE seems to achieve better performance during training than SCANet in this dataset.

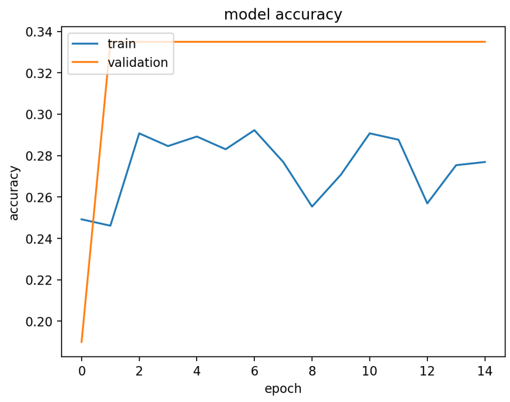
Table 5.1: Testing results, DPA contest v2

Dataset	SCANet	SPACE
1,000	0.253	0.253
10,000	0.275	**0.277**
50,000	0.244	**0.262**
100,000	**0.271**	0.237

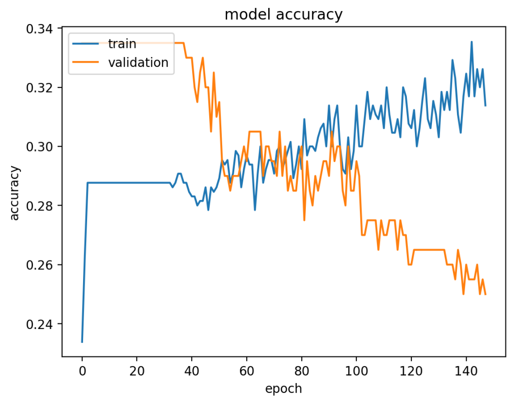
On 1,000 traces

Beginning with 1,000 traces, both CNNs achieve the same accuracy score in the test set. The main difference between the two models becomes more apparent in the graphs of training accuracy (Figure 5.1) and loss (Figure 5.2). SCANet exhibits a random behavior while training, where the score in the validation set seems to achieve a random score and remain unchanged during the whole training process.

SPACE's graphs on the other hand, show that, even though the final score of the model is not good, it achieves better training throughout the epochs. Accuracy and loss seem to gradually increase throughout training for SPACE but still, the model achieves bad results in the test set. The most possible explanation for this behaviour, could be the low amount of data and the fact that SPACE model is a rather shallow network. Thus, due to the high class overlap in DPA contest v2, SPACE seems to be learning the noise of the data, rather useful patterns that would increase its performance.

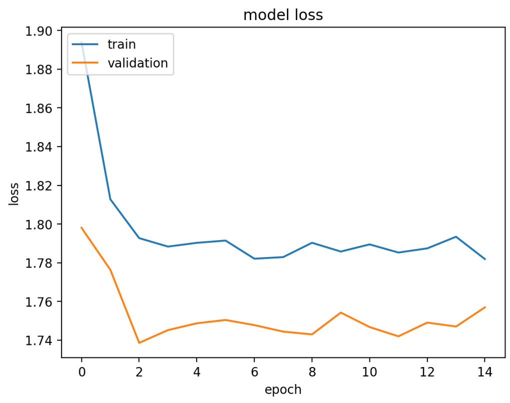


(a) SCANet's Performance

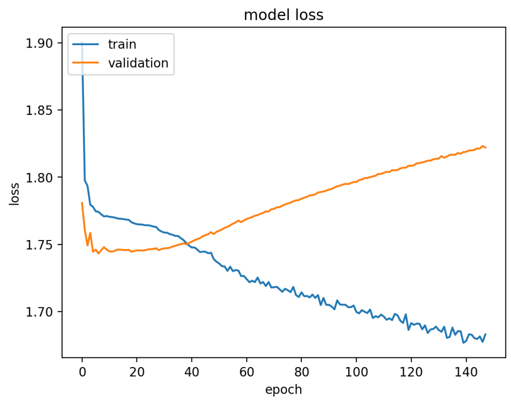


(b) SPACE Performance

Figure 5.1: Models' accuracy in DPA contest v2, 1k scenario



(a) SCANet's Loss



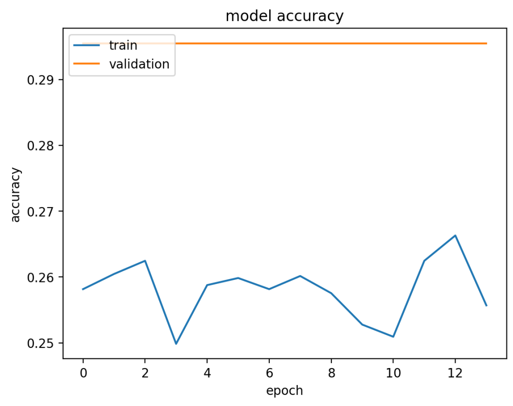
(b) SPACE Loss

Figure 5.2: Models' loss in DPA contest v2, 1k scenario

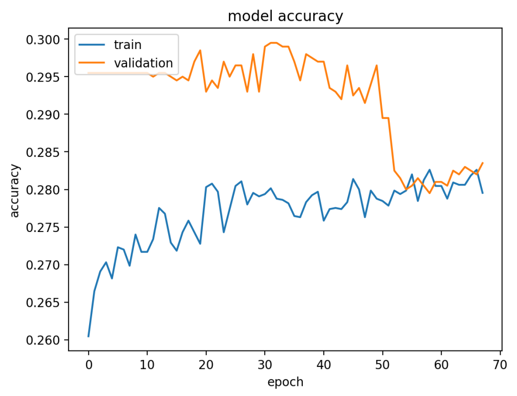
On 10,000 traces

On the subset of 10,000 traces, we observe the same that SPACE is more capable to classify the data, compared to SCANet (see Figures 5.3 and 5.4). Although SCANet's learning behaviour seems more erratic, SPACE is still not capable to achieve good overall classification results. We see that the training and validation accuracy curves of SPACE are smoother than in the 1,000 traces' case, which can be attributed to the 10 times more available data for the classification.

With more traces, the model is capable to find relevant features more easily, but due to the difficulty of this dataset, the scores are still low, achieving high loss values. Even though the performance scores are higher than before, the SCANet appears to be even more inconsistent. This can be ascribed on the high class overlap and the still low availability of data. As SCANet is more complicated as a model compared to SPACE, with so little data it is not capable to learn from the data and distinguish the relevant features from noise.

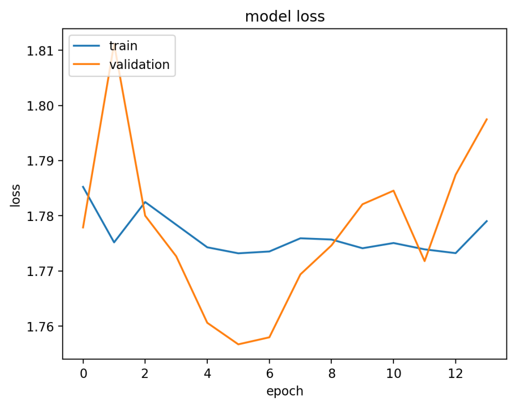


(a) SCANet's Performance

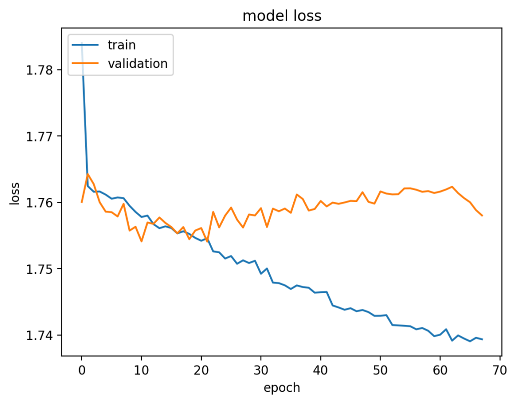


(b) SPACE Performance

Figure 5.3: Models' accuracy in DPA contest v2, 10k scenario



(a) SCANet's Loss



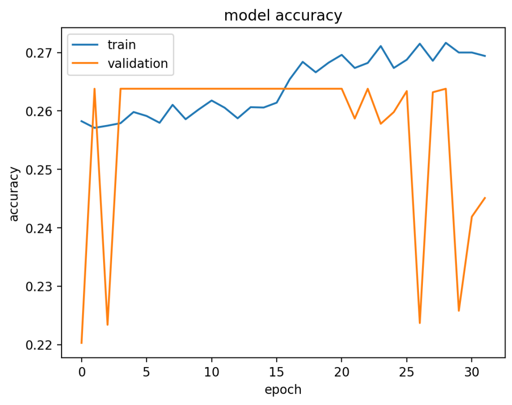
(b) SPACE Loss

Figure 5.4: Models' loss in DPA contest v2, 10k scenario

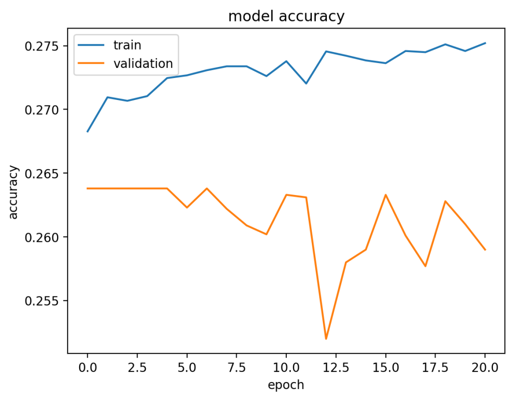
On 50,000 traces

In Figures 5.5 and 5.6, we observe that the SPACE's performance difference between the training set and validation set is higher than in the previous scenarios. The graph on validation set, never approaches the performance on the training set, increasing the difference through the epochs. This change is natural if we think that SPACE is a low complexity model. With more traces available, the noise in the data and the class overlap become more pronounced, leading the SPACE model on a decreased performance as it overfits on the noise of the training set, rather being capable to generalize on new data.

Although SCANet's accuracy performance is still random, a quick view on the loss graph shows that SCANet is less prone to errors and achieves lower loss than SPACE. Regardless though of its lower loss during training, SCANet fails to achieve better accuracy on the given data. The reason of this, becomes more obvious in the next scenario of 100,000 traces.

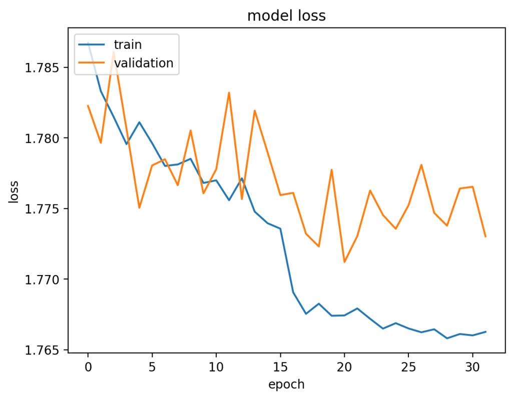


(a) SCANet's Performance

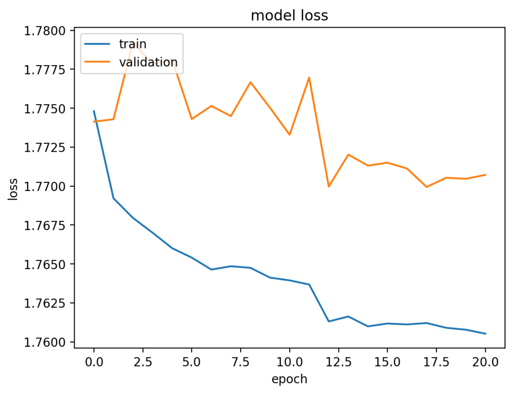


(b) SPACE Performance

Figure 5.5: Models' accuracy in DPA contest v2, 50k scenario



(a) SCANet's Loss



(b) SPACE Loss

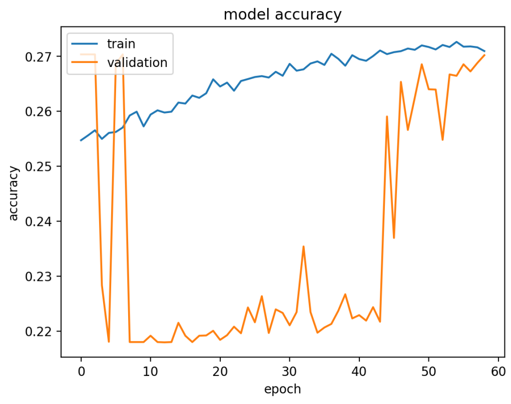
Figure 5.6: Models' loss in DPA contest v2, 50k scenario

On 100,000 traces

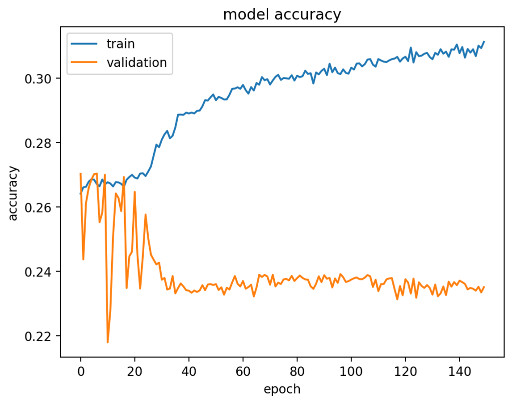
In this scenario, we see a clear different behavior of SCANet, achieving a far better performance compared to SPACE and compared to previous scenarios. The Figures 5.7 and 5.8, show a constant decline of loss for the SCANet model and a clear increase in accuracy after the 40th epoch. Due to SCANet's higher complexity compared to SPACE, in this scenario where more traces were available, the model is more capable generalizing the learned features from the training set thus increasing its performance. The higher accuracy score on the test set, shows also that the learned features are relevant to the particular characteristics of this dataset's traces. Even though the score is lower than the scenario with the 10,000 traces (27,5% compared to 27,1% in this case), we can deduct more useful insights in the model's performance as the behavior is less random, with a lower loss throughout the training.

In the case of the SPACE model, we see that once we were increasing the amount of the input data, the model became more and more prone to error and low accuracy performance. The model loses its capability to generalize the learned features in the training set and this problem amplifies in this scenario. As a simpler architecture, SPACE seems to overfit on the training set, meaning that it fails to model the class overlap and the data noise. This makes it unable to learn appropriate features that could be found in newly introduced traces of the dataset. Due to the larger amount of data in this scenario, SPACE is "learning" the noise of the training data, making it to recognize features that are not (or less) present in the validation and test sets.

We hypothesize, based on the SCANet's increased performance in this scenario, that by introducing more traces, SCANet could achieve a higher performance, as it would have sufficient input data to learn features that characterize this dataset.

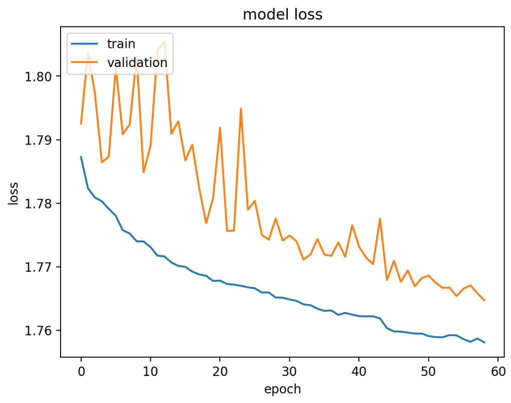


(a) SCANet's Performance

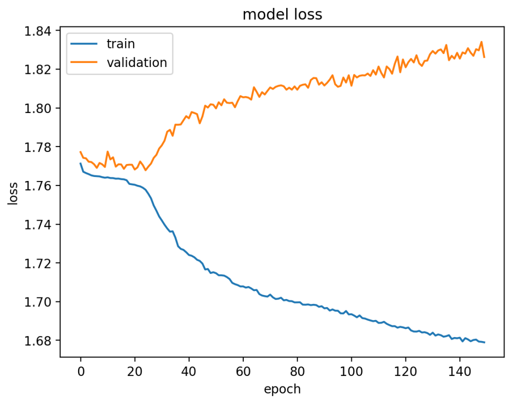


(b) SPACE Performance

Figure 5.7: Models' accuracy in DPA contest v2, 100k scenario



(a) SCANet's Loss



(b) SPACE Loss

Figure 5.8: Models' loss in DPA contest v2, 100k scenario

5.1.2. Comparison between CNN and ML

In this comparison, we have to remind that as explained in section 4.2, each trace is made out of 50 features, all calculated after applying Pearson Correlation Coefficient. Thus the amount of noise was reduced, alongside the dimensionality of the data. SCANet's performance did not change much in this scenario, except the slight increase in performance on the 50,000 and 100,000 traces' datasets. As shown by the random classifier $0-R_{CNN}$, SCANet is unable to meaningfully classify the data (the model classifies the data in the two most populated classes, 4 and 5).

SVM and LR achieve the greatest performance in this dataset. Their higher performance, may still be perceived low though but this fact can be attributed on the high class overlap, class imbalance and noise in the traces. Although surpassing SCANet's classifying capabilities, both SVM and LR fail to surpass in accuracy the random classifier $0-R_{ML}$.

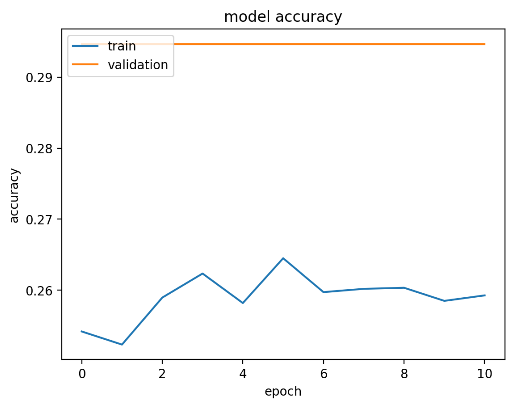
As a final remark for DPA contest v2 dataset, we found that simple ML models or conventional CNN architectures are not able to classify the data with high fidelity. The same type of performance is found in ARM FPGA dataset as well, leading us to acknowledge the limitations of such statistical models in this type of problems, with highly noisy and imbalanced datasets.

Table 5.2: Testing results, DPA contest v2, 50 features

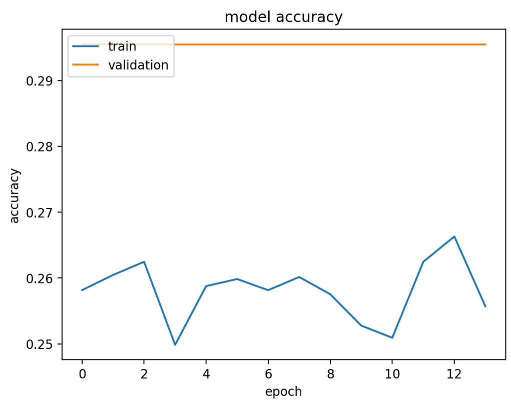
Dataset	$0-R_{ML}$	NB	LR	MLP	SVM	XGB	$0-R_{CNN}$	SCANet
1,000	0.308	0.131	**0.308**	0.271	**0.308**	0.267	0.253	0.253
10,000	0.282	0.067	**0.282**	0.269	**0.282**	0.277	0.275	0.275
50,000	0.274	0.116	**0.274**	0.271	**0.274**	0.273	0.267	0.267
100,000	0.274	0.108	0.273	0.273	**0.274**	0.273	0.273	0.273

5.1.3. Results on Random Sample Shuffling

As the DPA contest v2 dataset is a difficult target for SCANet, when we conducted the experiment with the random sample shuffling, the results were not convincing. The network's performance is still seemingly random as shown in Figures 5.9 and 5.10. Thus we can't conclude in any meaningful insights through this experiment, other than that the performance remained the same.

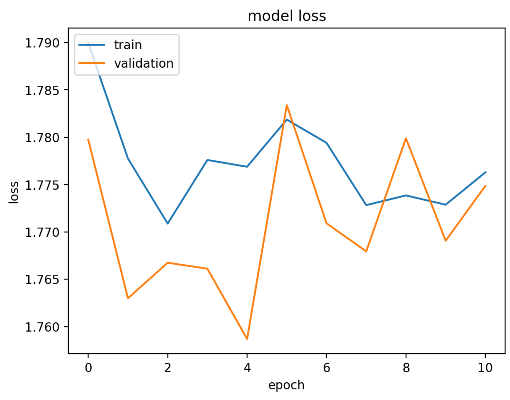


(a) Shuffled samples

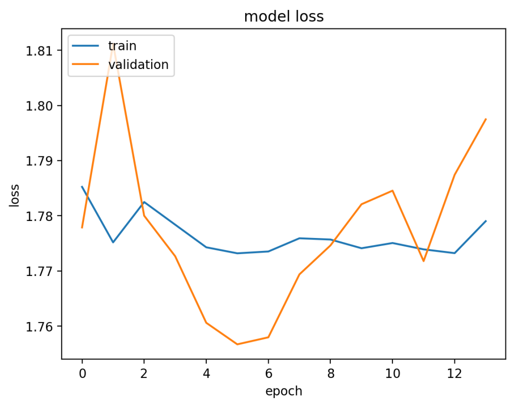


(b) Original signal

Figure 5.9: SCANet performance comparison in shuffled and original samples from DPA contest v2, 10k scenario



(a) Shuffled samples



(b) Original signal

Figure 5.10: SCANet loss comparison in shuffled and original samples from DPA contest v2, 10k scenario

5.2. Results on DPA contest v4

Looking at the data clustering of DPA contest v4 in Figure 4.2, we expected to have better classification results compared to DPA contest v2 dataset. Indeed the results were far better with SCANet achieving high accuracy performance and outperforming (in cases, significantly) other classification models. The results in this dataset were encouraging on the use of CNN for classifying Side-Channel data, making more prominent the fact that datasets such as the v2 version of the contest, can significantly hinder the performance of CNNs.

5.2.1. Comparison between CNN Architectures

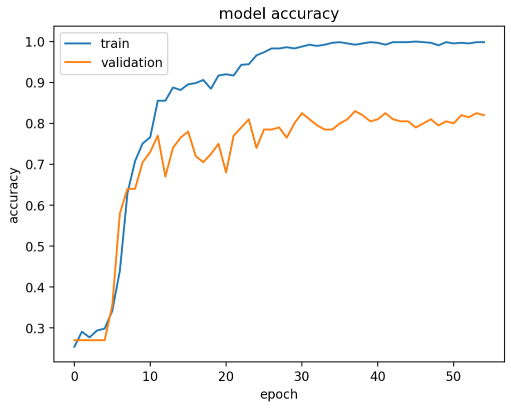
As shown in Figure 5.3, SCANet outperforms the suggested SPACE model. This was to be expected as SCANet was the product of Random Search optimization on DPA contest v4 dataset, leading to an architecture that was highly efficient on this dataset. There is a clear performance difference in both networks, when we used 50,000 traces, compared the other scenarios. We further look into the models' performances in the coming sections, where we examine this difference alongside the effects of each set size to the classification performance.

Table 5.3: Testing results, DPA contest v4

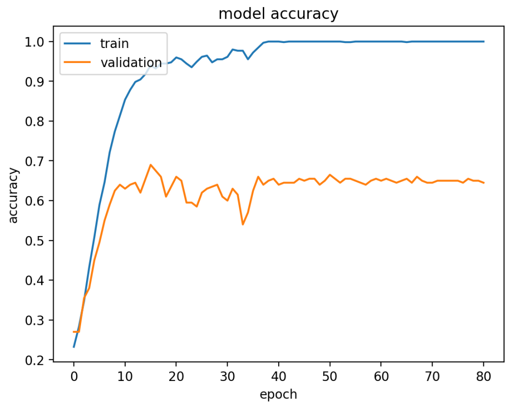
Dataset	SCANet	SPACE
1,000	**0.720**	0.573
10,000	**0.947**	0.927
50,000	**0.993**	0.984
100,000	**0.988**	0.977

On 1,000 traces

In the case of 1,000 traces, we can observe that both models start to overfit early in their training process. The Figures 5.11 and 5.12 show that in both models, the curves of validation accuracy and loss begin to deviate from the training ones in an early point, leading the models in a lower performance on newly introduced traces. That can be attributed to the fact that the amount of training data is low compared to the complexity that both models can represent. That increases the variance of the model, as it identifies random noise fluctuations in the training data, as significant features for classification, leading to poor results when those random fluctuations are not present on new data (in validation and test sets).

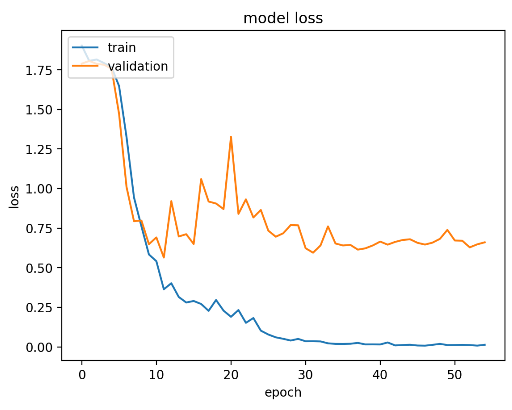


(a) SCANet's Performance

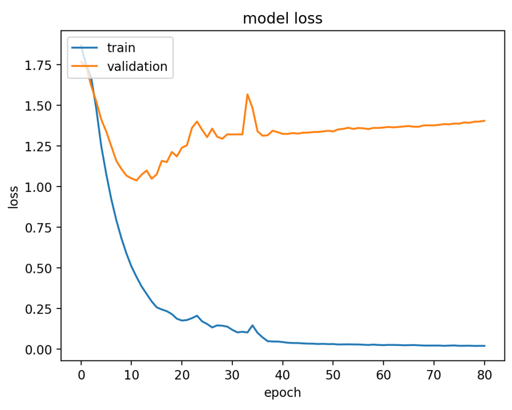


(b) SPACE Performance

Figure 5.11: Models' accuracy in DPA contest v4, 1k scenario



(a) SCANet's Loss



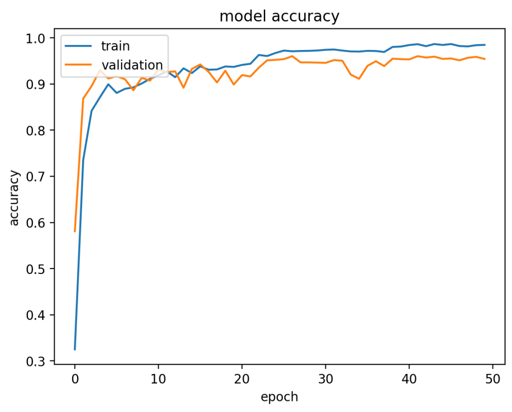
(b) SPACE Loss

Figure 5.12: Models' loss in DPA contest v4, 1k scenario

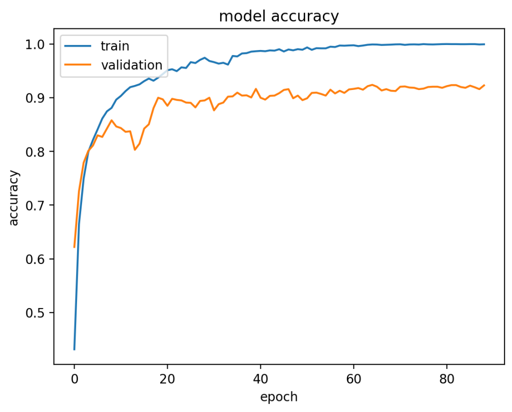
On 10,000 traces

The problems with overfitting seem to already not be apparent in SCANet, when using 10,000 traces. The model has a significant increase in accuracy compared to the previous scenario, as the Figures 5.13 and 5.14 show. SPACE on the other hand, is still not as capable to attain high performance, which can be attributed once more, on the combination of lacking enough data and the lower complexity of SPACE compared to SCANet. As a model, SPACE is still "suffering" of overfitting on the training set, with the difference of training and validation values to deviate noticeably after a certain amount of epochs.

SPACE's lower performance compared to SCANet though, does not change the fact that both models, seemed to benefit from the increase of available data in this scenario. Both of the models achieved accuracy higher of 90% and loss lower than 0.7. This can be perceived as an indicator already, that this dataset is far easier to classify compared to DPA contest v2.

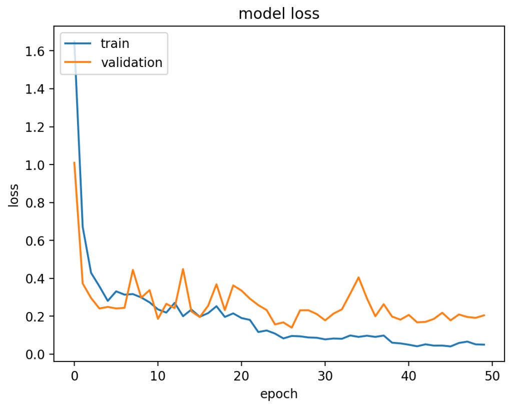


(a) SCANet's Performance

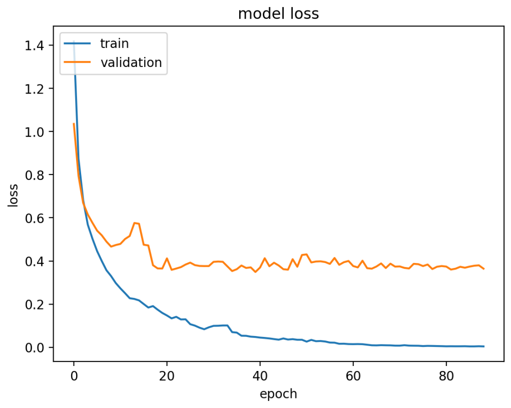


(b) SPACE Performance

Figure 5.13: Models' accuracy in DPA contest v4, 10k scenario



(a) SCANet's Loss



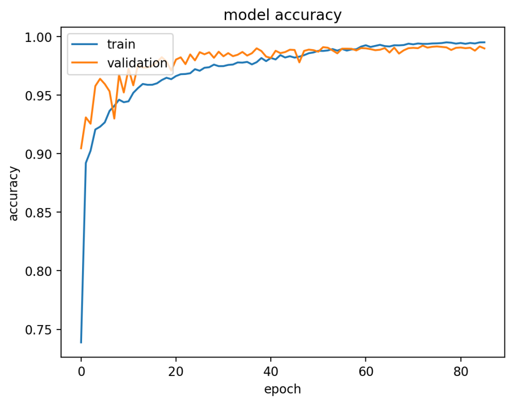
(b) SPACE Loss

Figure 5.14: Models' loss in DPA contest v4, 10k scenario

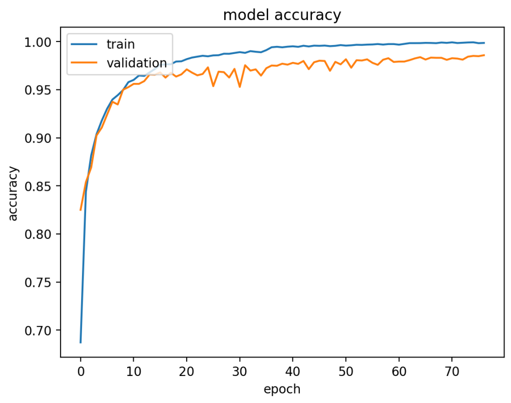
On 50,000 traces

In this scenario, SCANet achieves the highest performance results. With an accuracy of 99,8% on the test set, SCANet achieves the highest accuracy of all the classification models we used in this dataset. In Figures 5.15 and 5.16, we can see that the model has low difference between the performance on the training and validation set, with the training process halting only because of our training criteria of stoping the training after no significant drop in the model's loss.

SPACE achieves comparable performance but with a higher difference between the performance in the training and validation set, compared to SCANet. This indicates that SPACE is slightly overfitting on the given data, not being able to further generalize as well as SCANet.

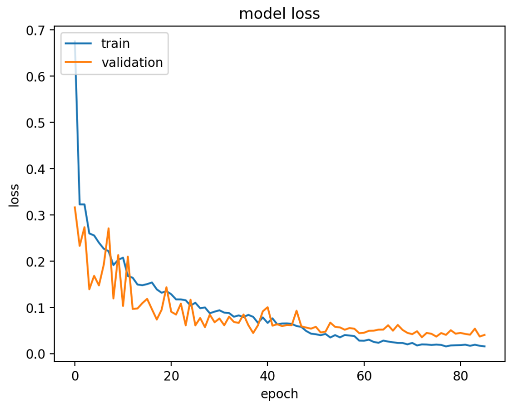


(a) SCANet's Performance

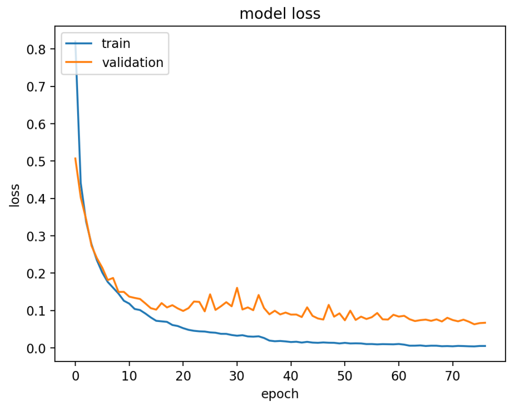


(b) SPACE Performance

Figure 5.15: Models' accuracy in DPA contest v4, 50k scenario



(a) SCANet's Loss



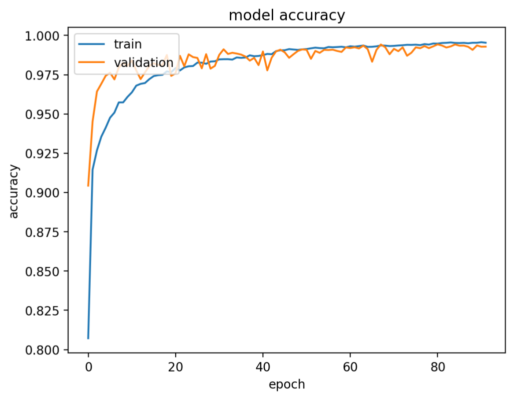
(b) SPACE Loss

Figure 5.16: Models' loss in DPA contest v4, 50k scenario

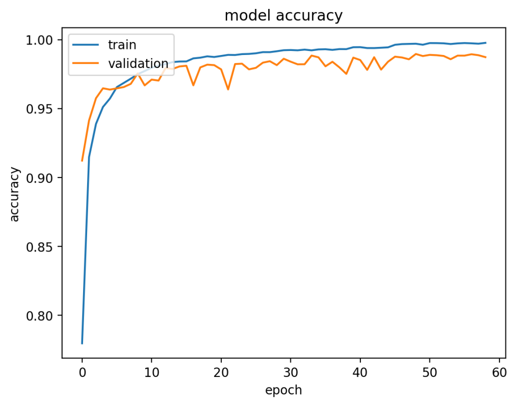
On 100,000 traces

In the 100,000 traces scenario, the results on the test set and the Figures 5.17 and 5.18 show that both CNNs interestingly, don't benefit much from the more provided data. We can observe that SCANet minimizes the performance gap between training and validation set but this doesn't reflect on the test set performance. This could be attributed on the fact that we introduced more data to the model, thus increasing the size of training set, making the model more sensitive on random fluctuations of the DPA contest v4 dataset. The performance difference on the test set though is 0.5%, making it too small to derive more a concrete indication on whether more data actually hindered SCANet's performance.

We see that SPACE on the other hand, is not affected much in the training process by providing more data. Regardless of this observation, SPACE's performance on the test set is still lower compared to the previous score, although by a small margin (0.7%). Both models seem to benefit the most in the 50,000 traces scenario, but don't show clear losses in this scenario that could help to further examine the effects of doubling the data in this scenario.

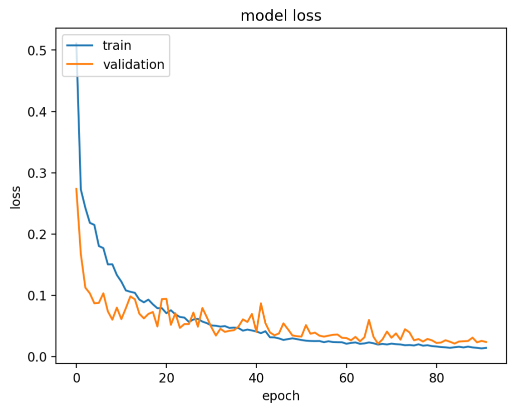


(a) SCANet's Performance

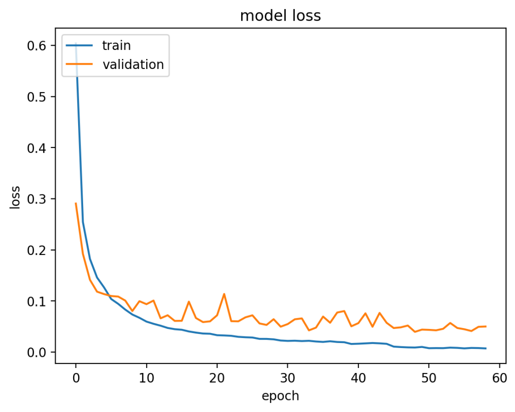


(b) SPACE Performance

Figure 5.17: Models' accuracy in DPA contest v4, 100k scenario



(a) SCANet's Loss



(b) SPACE Loss

Figure 5.18: Models' loss in DPA contest v4, 100k scenario

5.2.2. Comparison between CNN and ML

By lowering the samples per trace, we can observe that SCANet was heavily hindered. It fails to achieve previous performances and it is not able to reach 90% accuracy. This is a clear indication that the model's architecture was optimized on the whole number of traces, finding relevant features which possibly are not prominent on the 50 features that were extracted after Pearson correlation coefficient.

Simpler classification models manage to achieve better accuracy performance (MLP, SVM and XGB surpass SCANet). SVM performs the best yet again, showing that Side-Channel data, after a certain data pre-processing, can be sufficiently classified for the purposes of our problem.

Table 5.4: Testing results, DPA contest v4, 50 features

Dataset	$0-R_{ML}$	NB	LR	MLP	SVM	XGB	$0-R_{CNN}$	SCANet
1,000	0.297	0.639	0.477	**0.834**	0.823	0.725	0.267	0.693
10,000	0.272	0.669	0.557	0.867	**0.924**	0.886	0.268	0.811
50,000	0.275	0.654	0.601	0.866	**0.955**	0.913	0.273	0.851
100,000	0.274	0.662	0.607	0.866	**0.960**	0.916	0.269	0.845

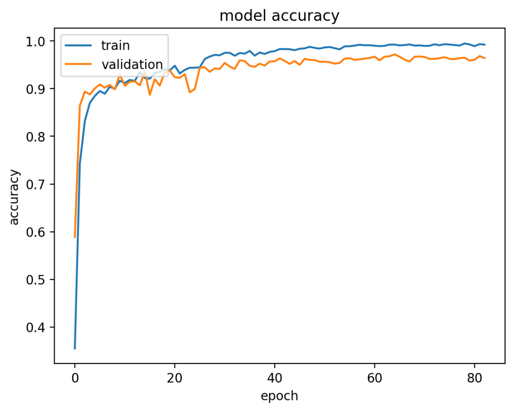
5.2.3. Results on Random Sample Shuffling

In DPA contest v4, we have meaningful results that can help examining our hypothesis. SCANet performance appears to not be hindered by the random sample shuffling and in this case it even achieves better accuracy. We believe that the increase in accuracy isn't an indication that randomly shuffling the traces' samples will result in better results, as it is only one case which was not further studied as it was out of our research goals.

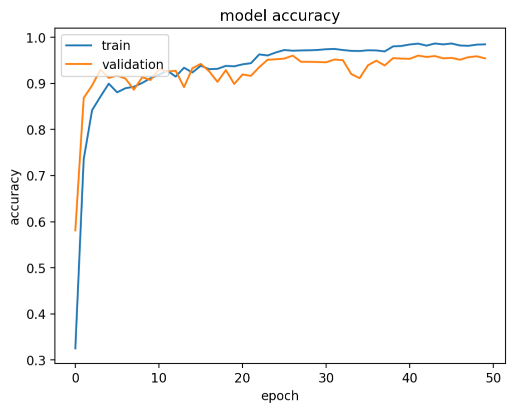
The fact though, that SCANet's performance wasn't affected negatively in both datasets (v2 and v4), is a good indicator that the sample topology of Side-Channel traces, is not a big factor in the CNN classification process. This could be used as an encouragement to future research on the topic, to explore other types of neural networks that don't necessarily retain the input's topology [34].

Table 5.5: SCANet testing results, DPA contest v4, Random shuffling

Dataset	Random Shuffling	Original Signal
10,000	0.966	0.947

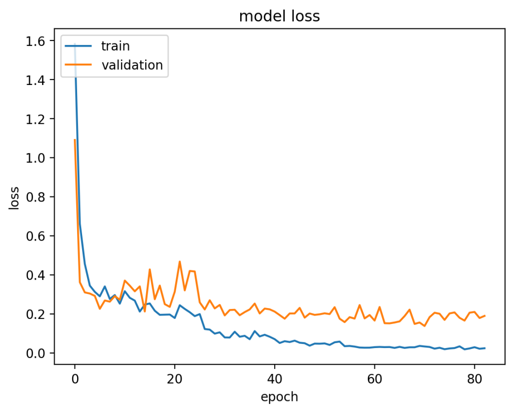


(a) Shuffled samples

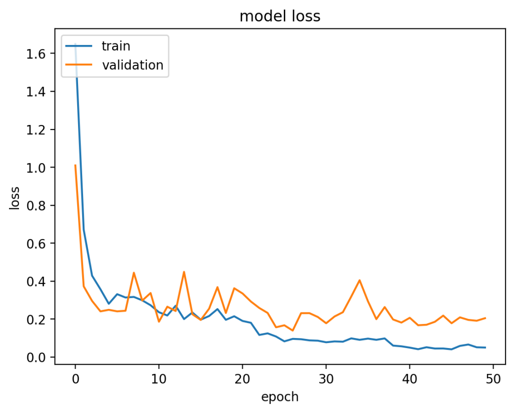


(b) Original signal

Figure 5.19: SCANet performance comparison in shuffled and original samples from DPA contest v4, 10k scenario



(a) Shuffled samples



(b) Original signal

Figure 5.20: SCANet loss comparison in shuffled and original samples from DPA contest v4, 10k scenario

5.3. Results on AES ARM32

As mentioned in chapter 4.2.1, we conducted a separate set of experiments for the AES ARM32 dataset. As this dataset was used during an internship with Riscure B.V., we classified the data with an optimized model using the Java implementation (Deeplearning4j framework), an optimized network using the Python 3 implementation (Keras/Tensorflow backend) and SCANet. Through this set of experiments, we determined differences in frameworks' capabilities as well as the type of architecture that could classify the data with the highest possible accuracy.

5.3.1. Comparison between CNN Architectures

In the Table 5.6, we see that the performance of the three models, differ a lot compared to each other. The optimized model using the Deeplearning4j (DL4J) framework, achieved the lowest performance, while the optimized model using the Python implementation, achieves the highest. Even though SCANet's performance is not that high, it achieves a better than random performance, higher than the DL4J model.

Table 5.6: Testing results, AES ARM32

Dataset (10,000)	DL4J (Opt-model)	Keras (SCANet)	Keras (Opt-model)
1^{st} keybyte	0.325	0.501	**0.902**

We can only speculate for the low performance of the DL4J model. The implemented system was the same between Java and Python, where the Random Search optimization was implemented the same way. As mentioned though in the subsection 3.3.1, during implementing our system, the DL4J framework was still in a pre-1.0 version. This means that the framework was still in development and its implementation was still not stable.

Looking into the Figures 5.21 and 5.22, we can see that the DL4J model needed much longer time to train, with the first ~ 150 epochs being abnormal. The model exhibits a very erratic training behavior regarding its accuracy performance during those epochs. Its loss also is not decreasing significantly during that period. The reasons for this kind of performance are unknown and since is wasn't present in the other models, we speculate that it was a kind of the framework's deficiency.

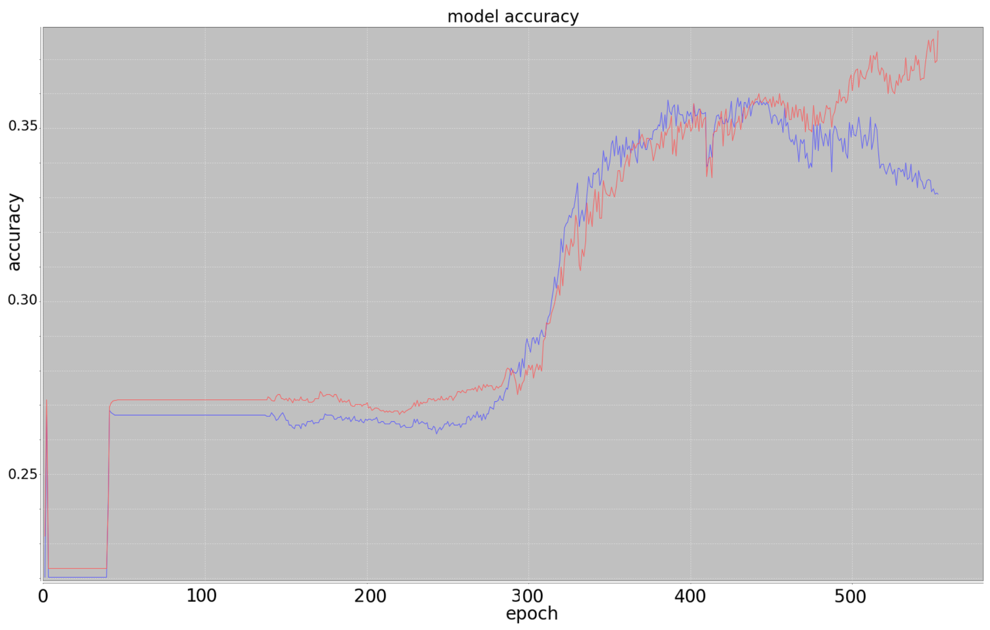


Figure 5.21: Model's performance in AES ARM32, DeepLearning4j

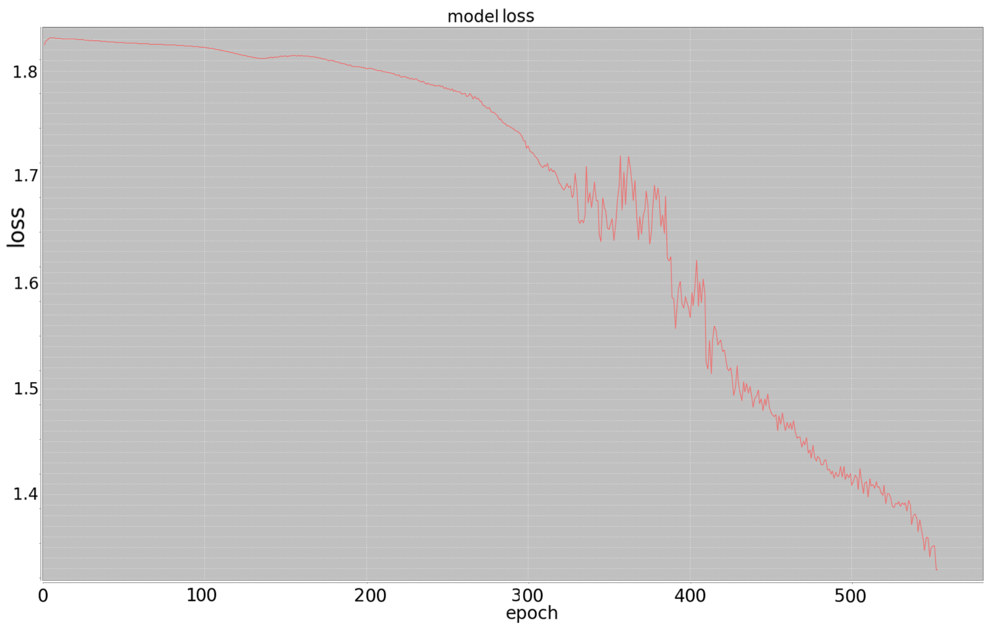
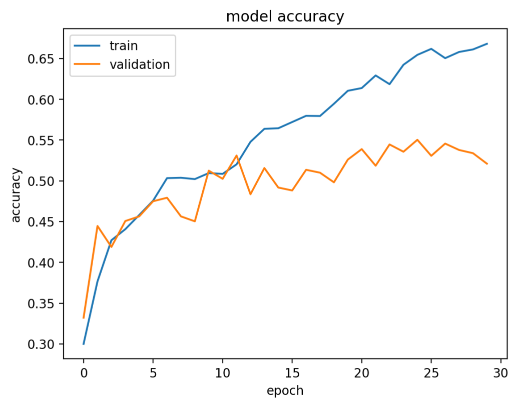
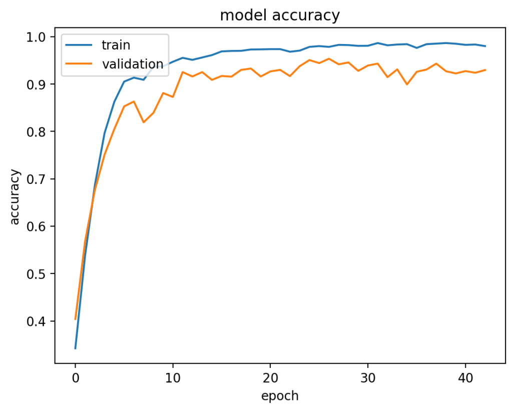


Figure 5.22: Model's loss in AES ARM32, DeepLearning4j

As we can observe in the Figures 5.23 and 5.24, the Keras/Tensorflow CNNs achieve far better performance compared to the Java CNN. The abnormalities found in the previous model's first epochs, are not apparent in these models and the training took much less time (30 to 40 compared to over 500 epochs). SCANet appears to overfit more compared to the optimized CNN, which is apparent on its test set accuracy (50.1%). The optimized CNN achieves far better results compared to the other models. It exhibits litte overfitting and it achieves low loss and high accuracy.

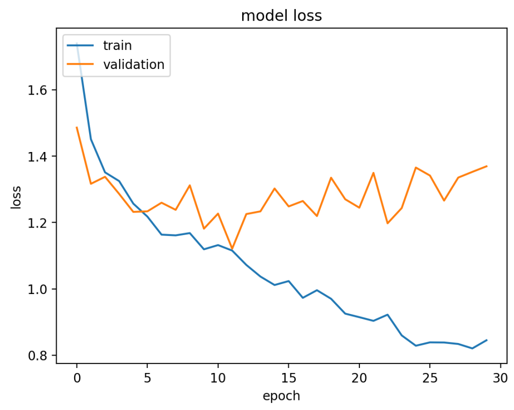


(a) SCANet's Performance

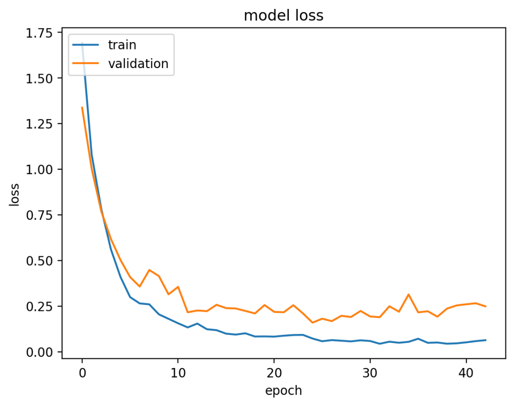


(b) Opt-model Performance

Figure 5.23: Models' accuracy in AES ARM32, Keras/Tensorflow



(a) SCANet's Loss



(b) Opt-model Loss

Figure 5.24: Models' loss in AES ARM32, Keras/Tensorflow

The AES ARM32 dataset was a relatively easy Side-Channel target. That being said, the low performance of the DL4J framework compared to the combination of Keras/Tensorflow, shows that Deeplearning4j was not yet ready for reliable experiments. The training time was much longer and the classification results (even after Random Search optimization) were really low compared to the Keras/Tensorflow CNNs. As it was the first framework where we ran experiments (during the internship in Riscure B.V.), once the Python implementation was ready and we witnessed the difference in the results, we switched our focus on Keras/Tensorflow which were used for the main work of this study.

5.4. Results on AES FPGA

The AES FPGA dataset, was a rather difficult target for classification. Even after Random Search optimization, the optimized model barely achieved better accuracy than SCANet or SPACE. The results resemble a lot the results found in DPA contest v2. That could be attributed on the fact that both datasets share some common implementation characteristics.

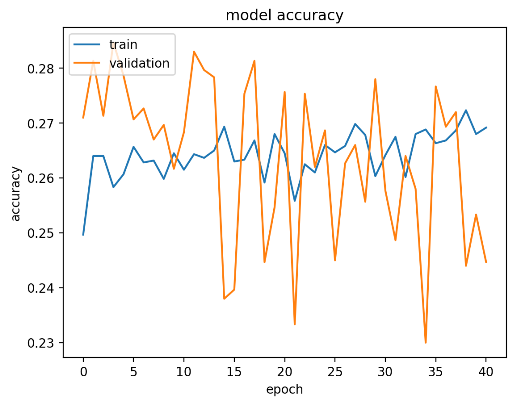
5.4.1. Comparison between CNN Architectures

None of the models were able to achieve high classification performance in this dataset (Table 5.7). SCANet and the optimized model achieved almost the same accuracy, but there is a clear difference in the learning process between the two models. The optimized model had a less random behavior compared to SCANet, which showed an erratic accuracy and loss performance (see Figures 5.25 and 5.26). SCANet's results could be perceived as totally random due to the non-stable learning performance.

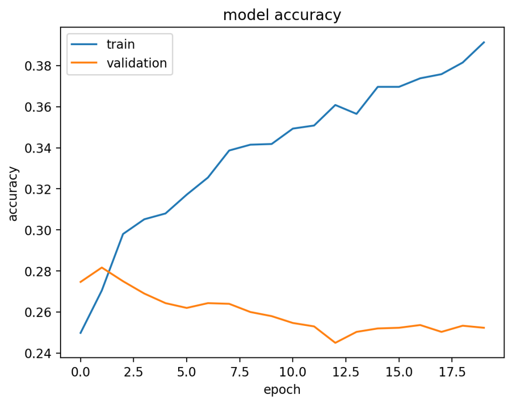
SPACE model is obvious that overfits on the training set, since the training and validation performance gaps are too wide. This could again be attributed on the model's simpler architecture, which makes it highly sensitive in random fluctuations and unable to generalize outside the input training data.

Table 5.7: Testing results, AES FPGA

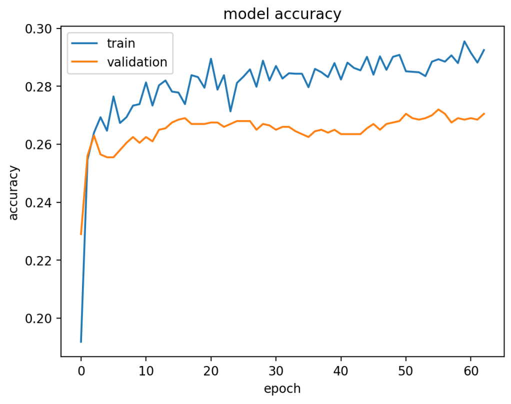
Dataset	SCANet	SPACE	Opt-model
10,000	0.261	0.244	**0.265**



(a) SCANet's Performance

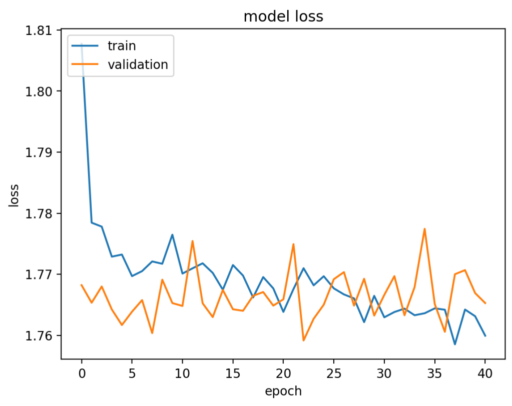


(b) SPACE Performance

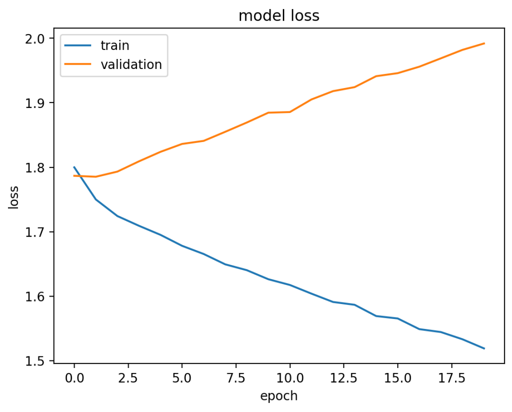


(c) Opt-model Performance

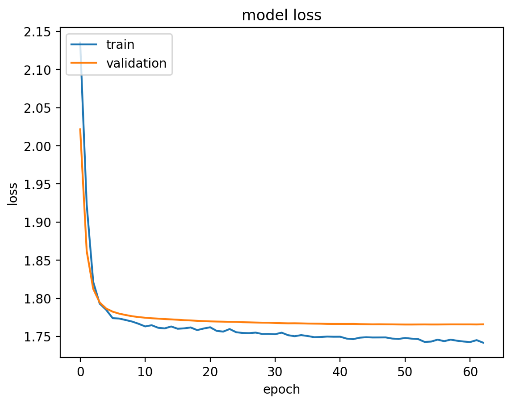
Figure 5.25: Models' accuracy in AES FPGA, 10k scenario



(a) SCANet's Loss



(b) SPACE Loss



(c) Opt-model Loss

Figure 5.26: Models' loss in AES FPGA, 10k scenario

5.5. Final Hyper-parameter Ranges

For each of the datasets, we used Random Search optimization before conducting each of the experiments. Once our experiments had finished, we finalized a range of values for each hyper-parameter we used in the search. We found that the search space could be narrowed from the initial ranges (see Table 3.1) into a search space that could yield optimized models for all the target datasets. The new value ranges of the hyper-parameters can be found in the Table 5.8.

Table 5.8: Final Hyper-parameter Ranges for Random Search Optimization

Hyper-Parameter	Value Range	Constraints
Convolutional Kernel	$k_{conv} \in [3,8]$	-
Pooling Kernel	$k_{pool} \in [3,4]$	$k_{pool} \leq k_{conv}$
Stride	$s \in [1,3]$	in pooling layers, $s = k_{pool} - 1$
Number of Convolutional Layers	$layers_{conv} \in [2,4]$	-
Number of Pooling Layers	$layers_{pool} \in [1,3]$	$layers_{pool} \leq layers_{conv}$
Number of Fully-Connected Layers	$layers_{fc} \in [0,2]$	-
Initial number of Activation Maps	$a \in [8,16]$	follows geometric progression with ratio r = 2, for the number of $layers_{conv}$
Initial number of Neurons	$n \in [256,512]$	follows geometric progression with ratio $r = 2$, for the number of $layers_{fc}$
Convolutional Layer Dropout	$drop_{conv} \in [0.05, 0.08]$	-
Fully-Connected Layer Dropout	$drop_{fc} \in [0.10, 0.13]$	-
Learning Rate	$l \in [0.001, 0.009]$	a learning rate scheduler was applied
Activation Function	ReLU, ELU, SELU, LeakyReLU, PReLU	the same for all layers except the last which uses Softmax
Optimization Algorithm	Adam, Adamax, NAdam, Adadelta, Adagrad, SGD, RMSProp	-

6 Conclusion and Discussion

Side-Channel Attacks oppose a real threat for the security of our computing systems in our everyday lives. With smartcards and micro-controllers being the primary targets, the attackers have the potential to decipher encrypted transaction data, commit identity theft and other types of malicious attacks. Through this study, we examined how SCAs can be performed and how Deep Learning could potentially enhance them. By surveying the available literature, we found two available works [6, 26] on the topic, where Convolutional Neural Networks were introduced as an obvious candidate for such types of attacks. Being such a novel topic though, we questioned CNNs' capabilities and in order to further study their performance, we applied them in four different datasets.

6.1. Overview of CNN-based Side-Channel Attacks

Through our experiments, we wanted to cover a variety of classification scenarios in order to study CNNs' performance on side-channel data. By splitting each dataset into sets of different sizes, we could examine how the amount of data influences the classification process. Alongside the application of various Machine Learning algorithms, we observed how CNNs compare to conventional ML techniques leading to a more objective view of CNNs efficiency in Side-Channel Attacks.

Due to the nature of side-channel data, the datasets suffer from high class imbalance, making the classification process even more difficult. In all of our classification scenarios using CNNs, we used the raw power usage traces of the targets, without applying any pre-process that would change the type of input representations. That ensured that we study CNN performance, without using techniques that could potentially enhance their classification capabilities.

We found that CNNs are a formidable type of SCA, for certain types of targets. In the cases where noise was relatively low and the data could be clustered more easily, CNNs excelled by achieving high accuracy and outperforming other Machine Learning techniques. That was not always the case though; in specific datasets such as DPA contest v2, simple ML models (e.g. SVM) outperformed CNNs, showing that further research is needed to find the proper classification technique for different types of targets.

By studying the experimental outcomes and comparing them with the available literature, we believe that there is a big research potential on other types of Deep Learning and Machine Learning algorithms. The field is relatively new to have definitive answers on the best classification technique, leaving room for experimentation.

6.2. Recommendations on Future Work

As explained in section 4.2, the only publicly available datasets at the time of study, were from the DPA contest. DPA contest has in total 4 dataset versions, with the latest one still having an open challenge since July 9, 2013 [46]. When studying the classification models and possible use cases, the lack of publicly available datasets can hinder the research potential. As such, we believe that people in the field should contribute on open data for Side-Channel Attacks, to help the research continue. By using proprietary datasets, the experimental results of new studies cannot be reproduced, leaving the outcomes disputable and scientifically questionable. Following the same paradigm, we would encourage future researchers, to make their classification models more available, to further support reproducibility and fair comparisons.

Through this study, we found that CNNs are not the best option for every classification scenario with side-channel data. Further experiments with the topology of the traces, indicated that the order of samples in the time domain, is not strongly correlated to the characteristics of each trace. By randomly shuffling the samples of each trace, we questioned the importance of keeping their topology intact (a prime characteristic of CNNs) thus leaving an open research question. We believe that through our study, we showed that future research should study other types of ANN and ML models and compare their results with CNNs as a benchmark. This would lead in more concrete correlations between types of SCA targets and classification models, thus helping establishing the field.

Data pre-processing is a part of the classification process that we didn't explore in this work. Using normalization and standardization on the power traces was the minimum pre-process that could be used, without changing the input representation. Common signal processing techniques such as Fast Fourier Transform (FFT) and Short-Time Fourier Transform (STFT), which are being used in other similar classification problems, could be explored in later research. Also, techniques such as data augmentation have already been shown to improve the classification [6].

Finally, there are many optimization algorithms that are being used in the Machine Learning. In our study we used Random Search which is a simple optimization algorithm but yields much better results than Grid Search or Manual tuning. The research on optimization algorithms is rich and implementations of more complex techniques such as Bayesian Optimization [40], Reinforcement Learning [42] and Genetic Algorithms [3], show great advantages. We believe that by exploring these techniques in a future work, the field could benefit greatly by generating even better classification models than the existing ones.

As a final note, we believe that Deep Learning could potentially enrich Side-Channel Attacks. Simple and conventional implementations of Convolutional Neural Networks, as the ones in our study, can already perform in a high classification standard. As deeper and more complex networks have been shown to solve hard classification problems [38], we believe that research should examine integrating Deep Learning into the Side-Channel Analysis field, to harness its potential. Not all the use cases are good targets for DL-based Side-Channel Attacks though and researchers should avoid using them as a default approach. As we found in this study, in certain cases, simple Machine Learning algorithms can already outperform the CNN implementations. Thus, we believe future research should focus on identifying scenarios where we truly benefit by the use of Deep Learning.

Bibliography

[1] Fully Connected Neural Network. https://math.stackexchange.com/questions/2048722/a-name-for-layered-directed-graph-as-in-a-fully-connected-neural-network, 2017. [Online; last accessed 19-March-2018].

[2] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL https://www.tensorflow.org/. Software available from tensorflow.org.

[3] Christine M Anderson-Cook. Practical genetic algorithms, 2005.

[4] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012.

[5] James S Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Advances in neural information processing systems*, pages 2546–2554, 2011.

[6] Eleonora Cagli, Cécile Dumas, and Emmanuel Prouff. Convolutional neural networks with data augmentation against jitter-based countermeasures. In *International Conference on Cryptographic Hardware and Embedded Systems*, pages 45–68. Springer, 2017.

[7] Chabacano. Overfitting. https://commons.wikimedia.org/wiki/File:Overfitting.svg, 2008. [Online; last accessed 19-March-2018].

[8] François Chollet et al. Keras. https://github.com/keras-team/keras, 2015.

[9] Chrislb. Artificial Neuron Model. https://commons.wikimedia.org/wiki/File:ArtificialNeuronModel_english.png, 2015. [Online; last accessed 19-March-2018].

[10] M Bishop Christopher. *PATTERN RECOGNITION AND MACHINE LEARNING*. Springer-Verlag New York, 2016.

[11] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

[12] Joan Daemen and Vincent Rijmen. *The design of Rijndael: AES-the advanced encryption standard*. Springer Science & Business Media, 2013.

[13] Howard B Demuth, Mark H Beale, Orlando De Jess, and Martin T Hagan. *Neural network design*. Martin Hagan, 2014.

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[15] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[16] Roger A Horn and Charles R Johnson. *Matrix analysis*. Cambridge university press, 1990.

[17] David H Hubel and Torsten N Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of physiology*, 160(1):106–154, 1962.

[18] David H Hubel and Torsten N Wiesel. Receptive fields and functional architecture of monkey striate cortex. *The Journal of physiology*, 195(1):215–243, 1968.

[19] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks. *arXiv preprint arXiv:1706.02515*, 2017.

[20] Paul Kocher, Daniel Genkin, Daniel Gruss, Mike Haas, Werner an d Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Sc hwarz, and Yuval Yarom. Spectre attacks: Exploiting speculative execution. *ArXiv e-prints*, January 2018.

[21] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[22] Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

[23] Moritz Lipp, Michael Schwarz, Daniel Gruss, Tho mas Prescher, Werner Haas, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. Meltdown. *ArXiv e-prints*, January 2018.

[24] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3, 2013.

[25] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

[26] 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.

[27] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[28] Melanie Mitchell. *An introduction to genetic algorithms*. MIT press, 1998.

[29] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[30] Michael Nielsen. Neural Networks and Deep Learning. `http://neuralnetworksanddeeplearning.com`, 2017. [Online; last accessed 19-March-2018].

[31] PaddlePaddle. LeNet. `http://www.paddlepaddle.org/docs/develop/book/02.recognize_digits/index.html`, 2017. [Online; last accessed 19-March-2018].

[32] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[33] Mark Pellegrini. Differential Power Analysis.

[34] Stjepan Picek, Ioannis Petros Samiotis, Annelie Heuser, Jaehun Kim, Shivam Bhasin, and Axel Legay. On the performance of deep learning for side-channel analysis. Cryptology ePrint Archive, Report 2018/004, 2018. `https://eprint.iacr.org/2018/004`.

[35] Raúl Rojas. *Neural networks: a systematic introduction*. Springer Science & Business Media, 2013.

[36] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[37] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533, 1986.

[38] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[39] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[40] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951–2959, 2012.

[41] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[42] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[43] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[44] Deeplearning4j Development Team. Deeplearning4j: Open-source distributed deep learning for the jvm. URL `http://deeplearning4j.org`. Apache Software Foundation License 2.0.

[45] TELECOM ParisTech SEN research group. DPA Contest (2nd edition), 2009–2010. `http://www.DPAcontest.org/v2/`.

[46] TELECOM ParisTech SEN research group. DPA Contest (4th edition), 2013–2014. `http://www.DPAcontest.org/v4/`.

[47] Bernard Widrow and Michael A Lehr. 30 years of adaptive neural networks: perceptron, madaline, and backpropagation. *Proceedings of the IEEE*, 78(9):1415–1442, 1990.