Bachelor's Thesis Computing Science

Radboud University Nijmegen

Fast And Furious: Implementing Optuna and Early Stopping in CNN-based SCA

Author:
Gabriele Serafini
s1060638

First supervisor/assessor:
Dr Lejla Batina

Second assessor:
Dr Stjepan Picek

July 11, 2023

Abstract

This study adapts the capabilities of the open-source Optuna framework, known for its state of art hyperparameters optimization (HO), to enhance the efficiency and performance of Convolutional Neural Network (CNN) models in Side-Channel Analysis(SCA). Particularly, we examine the Sampling Algorithm Tree-structured Parzen Estimator and early stopping based on previous models, combined with the classical patience-based early stopping mechanism, considering their role in fine-tuning the exploration-exploitation trade off in machine learning HO and their computational resource savings in SCA. Through different Optuna studies using two distinct datasets, ASCADf and ASCADr, the warm up values have been systematically varied. This approach allowed their influence on computational resources usage and model performance to be analyzed. The Optuna framework's adaptability, and ready-to-use features significantly streamlined the workflow for this work, illustrating the potential of integrating such established machine learning frameworks into SCA research. The results, suggest a consistent performance across all early stopping, with the best models achieving a Guessing Entropy (GE) of 0 using approximately 200 traces on average for both datasets. Interestingly,for ASCADr, models obtained from experiments with a shorter warm-up value displayed superior performance, reinforcing the notion that aggressive early stopping can still yield optimal models. This approach may also facilitate earlier identification of promising configurations during the optimization process, consequently leading to significant resource savings. However, the effectiveness of the models and the observed effects could be attributed to multiple factors such as the number of models tried in each HO experiment, the relative simplicity of the models required to analyse these datasets, and the vastness of the search space. Therefore, these findings should be interpreted considering these elements. In conclusion, our research lends support to the implementation of 2 different early stopping strategies in HO for CNN-based SCA. It highlights the invaluable benefits of adapting open-source frameworks like Optuna, thus encouraging further integration of such versatile tools into the SCA field.

Contents

Chapter 1

Introduction

1.1 The background of the problem

From personal smart gadgets to sophisticated industrial machinery, devices embedded with sensors, software, and other technologies are becoming an integral part of our daily lives. These devices, collectively referred to as the Internet of Things (IoT), have the ability to connect and exchange data with other devices and systems over the internet, thus becoming a modern necessity that automates various aspects of our daily activities and processes. However, as we integrate these IoT devices further into our lives and systems, we must address growing security concerns. One of the most critical concerns is the potential vulnerability of these devices to cyberattacks, which can lead to unauthorized access, data theft, or even control over the device's functions, posing significant risks to personal privacy and security.
Because of these concerns, many of these devices require rigorous security assessments to exhibit their safety. Side-Channel Analysis (SCA) is a critical component of these evaluations. This technique entails analyzing information that has unintentionally leaked from electronic devices while they are in use. This analysis can provide insights into the device's internal workings, exposing vulnerabilities to security breaches.
In the context of evaluation of cryptographic implementations, SCA represents a class of non-invasive attacks that focuses on the information leaked during the physical implementation of a cryptographic algorithm, rather than the theoretical weaknesses of the latter. In Literature, the concept emerged for the first time with Paul Kocher's groundbreaking work on Differential Power Analysis(DPA)[26] and timing attacks[27]. Ever since, a wide array of side channels have been studied, including electromagnetic emissions[16, 44], sound[3], light production[14], heat emissions[24] and faults[7]. In a traditional scenario, an attacker obtains a sequence of side-channel observations (also called traces), collected during the execution of a cryptographic operation. These traces typically include patterns which

are influenced by the intermediate computations of the cryptographic algorithm and the secret key. The main objective is to statically analyze these traces to successfully retrieve the secret key.
In its early days, SCA relied heavily on simple, straightforward statistical techniques, like DPA cited above, which provided crucial insights into the hidden information processed by electronic devices. These methods were pivotal in the advancement of SCA and continue to be relevant in many contexts.
As the field progressed, new techniques were introduced. One such development was the emergence of Template Attacks (TAs)[10]. In a TAs, a statistical template is created for every possible secret key. This template is then used to determine the most likely key.
In the continued evolution of SCA, Machine Learning (ML) emerged as another key technique. The early integration of ML in SCA saw the application of algorithms such as Random Forests and Support Vector Machines, which were used to predict key bits from side-channel measurements. This shift was set in motion in the study by Hospodar et al. (2011)[19], which was the first to use ML in SCA (even though machine learning was already used in cryptanalysis[46]). With the growing exploitation of ML techniques in SCA[18], the advent of deep learning techniques further propelled the field[43], offering advanced pattern recognition capable of detecting intricate patterns that traditional methods might overlook. A notable advantage of these deep learning methods is that they eliminate the need for extensive preprocessing of the data, streamlining the SCA process.
Among the various deep learning techniques, Convolutional Neural Networks (CNNs) have emerged as a particularly powerful tool for side-channel analysis[4]. CNNs are designed to automatically and adaptively learn spatial hierarchies of features, which make them incredibly effective at recognizing subtle, complex patterns in side-channel data.
By training a CNN on a labeled dataset of side-channel traces, the network can learn to find the secret key associated with the unlabeled side-channel traces. This enables the side channel community to use CNNs models to predict the secret key based on given side-channel traces, providing a highly effective tool for side-channel analysis.

1.2 The problem

A fundamental challenge when using CNNs in side channel analysis is the tuning of hyperparameters. Hyperparameters define the configuration of the model and have a significant impact on the model performance. The sheer size of the hyperparameter space and the expense of evaluating each configuration make the hyperparameter optimization (HO) task computationally demanding and time-consuming.

This situation is further complicated by the well-known "portability problem" in SCA[6]. In essence, the portability problem refers to the issue that a model optimized for one dataset or device might not perform well when applied to another. This means that in practice, it's often necessary to go through the time-consuming process of finding a new, optimized model for each different dataset. Each of these optimized models will likely require a different set of hyperparameters, further increasing the complexity of the task.
This issue represents a significant bottleneck in the field of SCA. As the landscape of the Internet of Things (IoT) continues to grow, the need for efficient and effective SCA in different settings has never been greater. Therefore, the complexity of the hyperparameter optimization problem is of crucial importance in this area.
The difficulty of navigating the vast hyperparameter space and the necessity of addressing the portability problem underline the need for an efficient framework for HO. Such a framework should ideally be capable of consistently and lightly generating highly effective models for different datasets.
Existing tools and methods for hyperparameter optimization are explored in the context of SCA, aiming to improve the efficiency of the HO process, thereby boosting the effectiveness of CNN-based SCA methods. By doing so, this research aim to contribute to improving the reliability and efficiency of cryptographic implementation analysis.

1.3 The solution

In response to the complex issues of HO and the portability problem in SCA, this research solution does not seek to reinvent the wheel but rather, optimizes it. While some efforts have been made within the SCA community to develop new frameworks for these tasks[9, 45, 40, 53], here a different approach is proposed.
Optuna, an established machine learning frameworks [2] which have already seen widespread adoption in the deep learning community is suggested in this research. This existing framework come with support for state-of-the-art hyperparameter optimization algorithms that can easily be integrated and parallelized across a variety of libraries and systems.
Moreover, this framework includes additional features that can further enhance efficiency. One such feature is Early Stopping strategies based on previous models, which allow for the early termination of unpromising models based on intermediate results of previous models[2]. This saves computational resources and allows the HO process to allocate more time to explore other, potentially more fruitful hyperparameters configurations.
In this work the capabilities of state-of-the-art ML framework Optuna are integrated with other resources-friendly strategies, aiming to develop an

effective solution to the challenge of hyperparameter optimization in CNN-based side-channel analysis. In doing so, this research seeks to contribute a valuable tool for improving the security evaluation of embedded devices in our interconnected world.

1.4 The contribution and the outline

The following points provide a more detailed overview of the study's focal areas:

1. The Tree-structured Parzen Estimator (TPE), a Sampling Algorithm, is used as HO algorithm to guide the selection of hyperparameter configurations.

2. An objective value, whose computation is based on the Guessing Entropy (GE), is used to guide the HO process. This objective value is an estimation of the effort in terms of amount of traces needed by the model to reach a GE value of 0.

3. An introductory analysis is conducted with the aim of enhancing the efficiency of hyperparameter optimization. By suggesting the incorporation of early stopping strategies, the study attempts to expedite the optimization process while minimizing the compromise on the quality of results. The integration of these techniques offers more flexibility in dealing with the exploration-exploitation trade-off during hyperparameter tuning. Patience Early stopping is used to halt training models that appear to have converged around a value and so further training is unlikely to significantly enhance performance. Median Early Stopping strategy is used to stop the training of models that appear to have little likelihood of improving previously run models.

4. This study introduces the integration of the Optuna framework into the SCA context. This is an initial attempt to explore the potential benefits of applying widely recognized machine learning frameworks within the distinct landscape of SCA, providing a basis for future research. The utilization of Optuna is aimed at simplifying and standardizing the hyperparameter optimization process in SCA, thus paving the way for further exploration with other optimization algorithms or machine learning frameworks in the SCA setting. Optuna already includes HO Sampling Algorithms and early stopping mechanisms. The objective value described in section 4 can then be computed and used by Optuna to guide the Sampling Algorithm during the HO process.

The remainder of this thesis is structured as follows: Section 2 supplies readers with necessary background knowledge on side-channel analysis (SCA),

hyperparameter optimization (HO), and Convolutional Neural Networks (CNN) needed for this research. Section 3 delves into the exploration of relevant studies and related work in the field. The research undertaken within this thesis is discussed comprehensively in Section 4. Lastly, Section 5 concludes the thesis, summarizing the insights gained and suggesting directions for future research.

Chapter 2

Preliminaries

2.1 Side-Channel Analysis

Side-Channel Analysis (SCA) seeks to exploit information leaked from the physical implementation of a cryptographic algorithm, as opposed to cryptanalysis, which examines the algorithm's theoretical underpinnings. This leakage can take various forms, including timing information, power consumption, electromagnetic emissions, or even sound. By collecting and analyzing this information, an attacker can potentially extract secret data, such as cryptographic keys, that compromise the system's security.
SCA poses a significant threat to both hardware and software cryptographic implementations, especially those found in embedded devices and Internet of Things (IoT) technologies. These systems often use cryptographic algorithms to ensure secure data transmission and storage. However, their physical attributes can betray the secrets they aim to protect, making them potential targets for side-channel attacks.
SCA can be divided in 2 categories: Active and Passive Analysis. Active Side-Channel Analysis involves the attacker actively interacting with the device to induce it into a state that increases the leakage of information. This could be done through methods such as fault injection, where the attacker causes the device to malfunction or make errors. The induced faults might alter the device's normal behavior, causing it to skip certain operations, malfunction, or operate in unintended ways. Even if these faults don't directly lead to traditional side-channel leakages like differences in power consumption or processing time, they could still compromise the security of the device or its data. For instance, a fault might cause the device to skip a critical security step creating exploitable weaknesses. Active analysis is more intrusive than passive analysis and carries a higher risk of detection or even causing permanent damage to the device. In contrast, Passive Attacks Side-Channel Analysis involves observing the behaviour of a device without actively modifying its operation. In a passive analysis, the attacker monitors and records

the data leaked through side-channels, such as power consumption, while the device is operating under normal conditions. Passive Analysis can be more difficult to detect since it doesn't necessarily require interaction with the device. Examples of Passive Side-Channel Analysis techniques are Simple Power Analysis (SPA), Differential Power Analysis (DPA)[26], Correlation Power Analysis (CPA)[8] and Template Attacks (TAs)[10]
Depending on the amount of knowledge and access an attacker has to the target device or a copy of the target device, passive Side-channel analysis generally fall into two categories: profiled and non-profiled. The division between profiled and non-profiled analysis significantly influences the method and feasibility of the attack. Non-profiled analysis often require less prior knowledge and are generally more applicable, but they may struggle with more complex and variable device behaviors. Conversely, profiled analysis can be highly effective but require a copy of the device for the profiling phase, which may not always be accessible.

2.1.1 Non-profiled Side-Channel Analysis

Non-profiled side-channel analysis primarly rely on patterns found within side-channel emissions of a single target device. Here, the attacker has no prior knowledge or additional access to similar devices for profiling. They must base their entire analysis on observations from the target device. The simplest type of side-channel analysis is Simple Power Analysis (SPA). In this scenario the attacker directly interprets power consumption measurements taken while the target device is operating. By closely inspecting the power traces, they can infer information about the operations that the device is performing.
For example, various operations such as exponentiation in RSA[47] and point additions in ECC[32, 25] may consume different amount of powers or take different amount of time, which can be directly observable in the power traces. This method however requires the cryptographic implementation to be relatively unprotected and the attacker to have a deep understanding of the device's operations and the implemented algorithm.
Another example of non-profiled attack is Differential Power Analysis(DPA). DPA is more sophisticated than SPA and does not requires specific knowledge about the sequence of executed operations in the algorithm. Instead, DPA involves analysing multiple power traces to find statistical patterns. The attacker chooses an intermediate variable and divide the traces into 2 groups based on this sensitive variable (e.g in AES the sensitive variable used is usually the last significant bit in sbox(input $\oplus$ key)). The attacker uses a distinguisher to find statistical difference between the 2 groups in order to reveal the key.
A specific variant of DPA that is widely used for SCA evaluation is Correlation Power Analysis (CPA)[8], which differs in the method used to distin-

guish key hypotheses. In a CPA attack, the attacker considers a sensitive intermediate variable that is computed for each possible key. The attacker then computes the predicted power consumption for each trace corresponding to each different intermediate value (and therefore, each different key). The predicted power consumption is then compared to the actual power consumption using the Pearson correlation coefficient. If the attack is successful, a high correlation will be observed for the correct key hypothesis, thereby revealing the key."

2.1.2 Profiled Side-Channel analysis

As previously mentioned, Profiled Side-Channel Analysis is based on the assumption that the attacker has access to one or more devices that are identical or functionally equivalent to the target device. This allows the attacker to "profile" the device behaviour under known conditions(like known plaintext and key) and use this information to infer secret information from the target device.
One of the most effective profiled side-channel analysis method is the above mentioned TAs.[10]
In this method the Profiling Phase consist on collecting a large number of power traces from the profiling device while it performs cryptographic operations with known keys. The aim is to buld a model or "template" of the device's leakage channel (like power consumption) for each possible key. An important part of this phase is the choice of the leakage model (described in section 2.1.4), as the leakage model is used to describe the relationship between the leakage channel and the processed data.
The result of the profiling phase is a set of templates - one for each possible secret key - which represent the device's leakage channel expected values.
The second Phase is the Matching phase, and it consists of comparing the measurements of the leakage channel from the target device, which is processing an unknown key, to the templates created during the profiling phase. The attacker then infers the key by selecting the template that most closely aligns with the observed trace, assessed using measures like Euclidean Distance or likelihood ratio.
Given the nature of profiled side-channel analysis, machine learning[33] has been found particularly useful because it can uncover hidden correlations in complex data and capture subtle patterns in side-channel leakages[19]. Unlike traditional techniques, machine learning algorithms, especially those based on deep learning, do not require explicit feature extraction or strict statistical assumptions. They can learn complex, non-linear relationships directly from raw data and generalize them to unseen cases, an essential property for effective side-channel analysis.
Profiling a device for side-channel attacks essentially involves understanding the relationship between variations in physical properties (like power con-

sumption or electromagnetic emissions) and secret information processed by the device. This is a pattern recognition problem at its core, which is precisely where machine learning algorithms excel.
Like in Template Attack, the Machine Learning based side-channel analysis approach also requires 2 phases, a profiling phase and a prediction phase:
During the profiling phase, a large number of power traces are collected from a device while it performs cryptographic operations with known keys. This data forms the training dataset, with the power traces serving as input features and the associated keys (or some function of them) serving as output labels. Subsequently, a machine learning model is trained on this dataset. The specific model can range from simpler methods such as Decision Trees, to more complex architectures like Support Vector Machine or Deep Learning models. These models are capable of learning intricate, non-linear relationships directly from the raw data. Particularly with deep learning models, the significant advantage is the capacity for automatic feature extraction, which alleviates the need for manual, pre-defined feature engineering.
During the prediction phase the trained model is deployed to predict the secret key based on new, unseen power traces from the target device. The model's learned associations between specific patterns in the power traces and specific keys come into play here, enabling it to infer the secret key. The accuracy and reliability of these predictions heavily depend on the quality of the training data and the model's capacity to generalize from the training data to new, unseen power traces.

2.1.3 Side-channel analysis: the steps before the attack

In this thesis the focus is on side -hannel analysis based on the power consumption of the hardware implementation of the cryptographic algorithm. Power analysis-based side-channel attacks are grounded in the practical measurement and analysis of a device's power consumption. The process typically involves a series of distinct stages:

1. Setting Up the Measurement Environment: The first step in performing power analysis involves preparing the physical setup for measuring power consumption. This usually requires specialized equipment such as an oscilloscope to capture the power traces, a probing device, and often a resistor inserted in the power supply line of the target device for voltage drop-based measurements. The accuracy and precision of the equipment can significantly influence the quality of the captured power traces.

2. Capturing Power Traces: Once the setup is ready, the target device is made to perform the cryptographic operation while the power consumption is measured. This process results in a series of power traces,

each representing the power consumption of the device over time during a single operation. It's worth noting that a single trace can contain thousands to millions of sample points, depending on the duration of the cryptographic operation and the sampling rate of the oscilloscope.

3. Preprocessing the Data: The raw power traces captured from the device often require preprocessing before they can be used for analysis. This preprocessing can include denoising to eliminate the effects of environmental noise, alignment to correct for any synchronization issues between different traces, and normalization to account for variations in the absolute power levels. Another important aspect of the preprocessing phase is the selection of points-of-interest from the power traces. Given that a single power trace can contain a large number of sample points, it's often impractical and unnecessary to use the entire trace for analysis. Instead, points-of-interest (often regions where the power consumption significantly varies) are identified and used for subsequent analysis. The method of selecting these points can significantly impact the success of the power analysis attack, common selection methods used in side-channel analysis are Difference of Means, Distance of means and Signal to Noise Ratio.

4. Building the Dataset: The final step is the compilation of these power traces into a dataset suitable for analysis. Each trace in the dataset corresponds to a single execution of the cryptographic operation, and the set of all traces represents the power consumption patterns across multiple operations. If a supervised learning method such as Template Attacks is being used, the dataset would also include the associated secret keys or intermediate values for each trace.

2.1.4 Leakage Models

In side-channel analysis, a leakage model is a critical component that links the observed side-channel information (e.g., power consumption or electromagnetic radiation) to the intermediate values of the cryptographic operation being performed. The leakage model thus forms a fundamental bridge between the physical world observations and the mathematical properties of the cryptographic algorithm, allowing for an attacker to make informed deductions about secret data.
The basic premise behind leakage models is that the side-channel emissions of a device are not random, but depend on the device's internal state and operations. In the case of a cryptographic device, these operations are influenced by the data being processed and the secret key. The leakage model aims to describe this relationship, essentially serving as a predictor of side-channel emissions based on the known intermediate values of the cryptographic algorithm.

Leakage models can take various forms, but the most commonly used models in power analysis attacks are the Hamming weight model, the Hamming distance model and the identity leakage model.

Hamming Weight Model

This model posits that the power consumed by a device is proportional to the Hamming weight (the number of '1' bits) of the data being processed. For example, if a device is processing an 8-bit value, the power consumed when processing the value 11110000 (four '1' bits, hence a Hamming weight of 4) would be roughly twice the power consumed when processing the value 00000101 (two '1' bits, Hamming weight of 2).
In software implementations values are frequently loaded into and out of registers using the microcontroller bus. Typically the bus is precharged at all bits being zeros or one and therefore the power consumption of the operation is often proportional to the number of 1 bits that are being moved into a register. Since this leakage model is concerned with the numbers of 1's in a binary representation, this leakage model is often used for software implementations as software systems store and manipulate data in registers.

Hamming Distance Model

The Hamming distance model suggests that the power consumed by a device during a transition between two states is proportional to the Hamming distance (the number of differing bits) between those states. For instance, if a register is transitioning from the value 10110011 to the value 11001100, the Hamming distance is 4, which is expected to cause a relatively high power consumption compared to a transition with a lower Hamming distance.
Most of the chips in the markets nowadays are build using CMOS (Complementary metal-oxide semiconductor)transistors[31]. CMOS transistors work by using 2 complementary transistors together: a p-transistor and a n-transistor. The arrangement ensures that, at any given time, there's a path from the output to either the power supply (through the PMOS when it's conducting) or to ground (through the NMOS when it's conducting), but never both simultaneously. This characteristic helps to minimize power consumption during static operation, since current isn't continuously flowing from power to ground, but makes power consumption dynamic and dependent on the switching activity of the transistors. This design therefore generate dynamic power consumption that is dependent on the switch of these CMOS transistors from state 0 to 1 and vice-versa.
Because of this property, the number of bit changes in a value after an operation is often proportional to the power consumption, making the Hamming Distance Leakage Model suitable for finding a relation between bits and power consumption in hardware implementations.

Identity Leakage Model

This model assumes a direct relationship between the observed side-channel leakage and the secret intermediate values during cryptographic operations. Under the ID model, the observed leakage is assumed to be the identity of the intermediate value. This suggests that the leakage directly reveals the intermediate value, without any noise or distortion.
Mathematically, this can be expressed as:

$$L = V$$

Here, L denotes the observed leakage and V represents the secret intermediate value.
In the context of the Advanced Encryption Standard (AES), for instance, the intermediate value could be the result of the SubBytes operation (the application of the S-box) in the first round, which can be expressed as $V = Sbox(key + input)$.
This would result in 256 possible classes (corresponding to the 256 possible values of an 8-bit output), each representing a distinct intermediate value.

2.1.5 Countermeasures

Side-channel countermeasures are strategies implemented to impede attackers from extracting secret data through side-channel analysis. Various techniques are utilized, including:

- Masking: Masking techniques add random values, or "masks," to secret data during computation. This process obscures correlations between observable characteristics and the secret data, making it difficult for an attacker to extract valuable information. The range of masking techniques includes Boolean masking, Arithmetic masking, and complex methods such as secret sharing.

- Hiding: Hiding techniques aim to minimize the information leakage during computation. This could involve using constant-time algorithms, which don't depend on the input data, or physically isolating sensitive components to reduce their electromagnetic emissions or power consumption variations. The goal is to minimize variations in observable characteristics that could otherwise be exploited by an attacker.

- Noise Addition: Another technique involves the addition of noise to side-channel measurements. This technique confuses an attacker's attempts to correlate measurements with secret data. Noise can be injected into the power supply or introduced through other background processes.

- Shuffling: Shuffling techniques randomize the order of operations within a cryptographic algorithm. By doing this, it becomes difficult for an attacker to correlate specific operations with side-channel observations, thereby protecting the secrecy of the data. [49]

2.1.6 Side-Channel Analysis metrics

Precision, recall, and accuracy are binary metrics, which consider only whether a prediction is correct or not. This approach fails to capture the full picture in side-channel analysis, where not only the correctness but also the confidence of the prediction matters.[42]
In a side-channel attack, an adversary's goal isn't necessarily to get the right key on the first guess, but rather to significantly narrow down the possibilities. Therefore, these binary metrics do not provide insight into how close or far the attacker is from the correct key if the first guess isn't correct.
Precision, which measures the fraction of correct positive predictions among all positive predictions, doesn't provide any insight into how many relevant items (i.e., the correct keys) are missed, which is a critical consideration in side-channel analysis.
Accuracy, which is the ratio of correct predictions over the total predictions, similarly falls short as it doesn't provide information about how much the potential key space has been narrowed down.
All these binary metrics can be misleading in the context of side-channel analysis. Thus, in the context of side-channel analysis, precision and accuracy may not be adequate metrics for evaluating the effectiveness of the attack. More nuanced metrics, such as guessing entropy and success rate, are typically more informative as they consider both the confidence and the correctness of the predictions, providing a more comprehensive picture of the attack's effectiveness.

Guessing Entropy(GE) Guessing entropy[48] is a metric that gives a measure of the average number of guesses an attacker would need to find the correct key. It takes into account both the correctness and the confidence of predictions. Specifically, for each trace, the possible keys are ranked according to their likelihood, and the number of guesses before the correct key is reached is noted. The guessing entropy is then the average of these numbers over all the traces. More specifically, Guessing entropy is the average number of guesses required to correctly guess the key. Mathematically, if we denote by $P(k)$ the probability of key k being the correct key, and $R(k)$ the rank of key k (i.e., its position in the sorted list of keys, starting from the most likely), then the guessing entropy GE is computed as:

$$GE = \sum_{k \in Keys} P(k) * (R(k))$$

The lower the guessing entropy, the fewer attempts are needed, on average, to find the correct key, indicating a more effective attack.

Success Rate (SR) Success rate is another commonly used metric in side-channel analysis. It measures the proportion of traces for which the correct key was ranked first (i.e., $R(k) = 0$). If we denote by I the indicator function that is 1 if its argument is true and 0 otherwise, and by N the total number of traces, then the success rate SR is computed as:

$$SR = (1/N) * \sum_{k \in Keys} P(k) * I[R(k) = 0]$$

both GE and SR presume that you have a way to compute or estimate the probabilities P(k) and the ranks R(k) for each key. In practice, this usually involves using your side-channel attack to compute a score for each key (for example, the correlation coefficient in a Correlation Power Analysis attack), and then sorting the keys according to these scores. Therefore, is important to know that calculating exact probabilities and ranks can be computationally intensive, especially for large key spaces. In practice, approximations or sampling methods are often used to compute these metrics more efficiently.

2.2 Deep Learning

2.2.1 Definition

Deep Learning is a subfield of supervised machine learning that focuses on algorithms inspired by the structure and function of the brain called artificial neural networks.
In contrast to traditional machine learning algorithms, which are often engineered to solve specific tasks, deep learning algorithms learn from raw data in a more automatic fashion. This is primarily achieved by using neural networks with several layers - hence the term "deep" learning.
These layers of artificial neurons, or "nodes", have the capability to learn complex representations of the input data. These layers are cascaded, with each layer learning to transform its input data into a more abstract and composite representation.
An important advantage of deep learning lies in its capacity to handle enormous amounts of unstructured data. While traditional machine learning models often necessitate manual feature engineering, deep learning models can learn such features directly from the data, making them particularly effective for complex tasks such as image recognition, natural language processing, speech recognition and side-channel analysis.

2.2.2 deep learning challenges

However, deep learning is not without its challenges. One prominent limitation is its dependence on large volumes of labeled data to achieve optimal performance. Additionally, due to their complex structures, deep learning models are often referred to as "black boxes" due to the inherent difficulty in interpreting and understanding the reasoning behind their decisions or predictions. Moreover, these models are computationally intensive, which might limit their accessibility in certain contexts.
One particular challenge in training deep learning models is the phenomenon known as double descent[35]. This refers to the counter-intuitive behavior where the test error decreases, increases, and then decreases again as we increase model complexity or the number of parameters, diverging from the conventional U-shaped behavior witnessed in traditional bias-variance trade off. Understanding and appropriately handling double descent is critical in developing effective deep learning models.

2.3 Deep feedforward networks

Deep feedforward networks, also often referred to as multilayer perceptrons (MLPs)[17], are the most traditional deep learning models. The goal of a feedforward network is to approximate some function f. For instance, for a classifier, $y = f(x)$ maps an input x to a label y.
A feedforward network defines a mapping $y = f(x; \theta)$ and learns the value of the parameters θ that result in the best function approximation. These networks are called feedforward because information flows through the function being evaluated from the input x, through the intermediate computations used to define f, and finally to the output y. There are no feedback connections in which outputs of the model are fed back into itself.

2.3.1 Forward Propagation

In a deep feedforward network, each layer is composed of multiple neurons or nodes. These neurons take in inputs, apply a weighted sum with a bias term, and pass the result through an activation function to produce an output. This output is then passed on as input to the neurons in the next layer.
The neurons in a given layer operate in parallel but are independent of each other; each has its own set of weights and a bias term, and each produces its own output.
Let's take a single neuron x_i in a hidden layer l for simplicity, with inputs represented as a vector x^{l-1}, weights as a vector w_i^l, and bias as a scalar b_i^l. The output (or activation value) of this single neuron x_i at layer l is computed by the activation function of the weighted sum of all his inputs at

layer $l-1$ plus the bias term, or more formally in vector notation:

$$x_i^l = \sigma(w_i^l \cdot x^{l-1} + b_i^l) \tag{2.1}$$

This activation function σ is typically a nonlinear function such as the softmax function or the rectified linear unit (ReLU) function. The choice of activation function can depend on the specific requirements of the model and the type of data it's working with. The softmax function is typically used in the output layer of a neural network for multi-class classification problems. The reason is twofold: it squashes the outputs for each class to be between 0 and 1, and ensures these values sum up to 1, effectively forming a probability distribution. This allows for an intuitive interpretation of the model's predictions as probabilities of each class, making softmax particularly well-suited to tasks where multiple categories are involved.
For a layer of neurons, this process is done for each neuron independently, with the outputs collected into an output vector.
This output vector then serves as the input to the neurons in the next layer, and this process of applying a weighted sum with bias and then an activation function repeats. In this way, data is propagated forward through the network from the input layer to the output layer.

2.3.2 Backpropagation

Following the feedforward operation in a neural network, where inputs are processed layer by layer until an output is produced, the next stage is backpropagation.
Backpropagation is an algorithm used to adjust the parameters (weights and biases) of a neural network to improve its predictions. The name "backpropagation" stems from the way the algorithm sends information about the error "backwards" through the network from the output layer towards the input layer, propagating the error to each of the weights and biases.
Firstly, the network makes a prediction during the forward pass. Following this, a loss function is used to measure the difference between this prediction and the actual output, producing an error value. The loss function essentially quantifies how 'wrong' the network's prediction is, and this forms the basis for adjustments.
To minimize the error, the backpropagation algorithm calculates the gradient of the loss function with respect to each weight and bias in the network. The gradient is a mathematical concept, representing the direction and rate of fastest increase of a function. In this context, it shows how much a small change in a weight or bias will impact the loss function's value. By considering this, the algorithm figure out how to adjust each parameter to decrease the error, thus improving the network's prediction.
However, it's not just about adjusting the weights and biases. The adjustments have to be proportional to each weight and bias's contribution to

the overall error. By computing the derivative of the loss function with respect to each weight and bias, the algorithm can identify how much each parameter contributed to the error. The derivatives guide the necessary adjustments, so the parameters that contributed more significantly to the error will be adjusted more.
After this process is performed across all the layers, the weights and biases are updated to their new values. The network then goes through another round of forward propagation and backpropagation, continuously learning and updating its parameters until the output error is minimized. In this way, backpropagation enables the network to learn from its mistakes and to make better predictions over time

2.4 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a specific type of neural network that have demonstrated exceptional effectiveness for tasks involving image and signal processing. They stand as a cornerstone of deep learning, especially in the realm of image classification and object detection tasks. The inspiration for CNNs comes from the structure and function of the vertebrate visual nervous system[15, 28], with their design intended to automatically and adaptively learn spatial hierarchies of features directly from data.
A significant aspect of CNNs' architecture is its design to take advantage of the 2D structure of an input image (or other 2D input such as a speech signal). This is achieved with the use of a unique kind of layer: the convolutional layer.
Convolutional layers form the primary building blocks of CNNs. Each layer in a CNN processes the input data using a series of small, square filters (also known as kernels), which move across the input, performing a dot product operation at each position, and creating a feature map. This operation is called convolution. The kernels act as feature extractor, each learning to identify a different pattern in the input data. The patterns can range from simple edges to complex shapes or textures, and they emerge automatically during the training process.
In a Convolutional Neural Network (CNN) model, we typically have two main components: the Convolutional component composed of convolutional blocks, and the Fully Connected component, essentially a Multilayer Perceptron (MLP). The primary function of the Convolutional part is to extract significant features from the input data, while the Fully Connected segment takes over the responsibility of learning how to correctly classify the processed input based on the extracted features.
This ability of CNNs to detect local patterns, irrespective of their position in the input data, is known as translation invariance. This means that once a useful feature has been learnt, the CNN can recognize it no matter where it

appears in the input. This property contributes significantly to the success of CNNs in various tasks.

2.4.1 Kernels

Kernels, also known as filters, are used by Convolutional Neural Networks to extract features from input data. These kernels slide over the input data, conducting dot product at each place (the convolution), and the results are aggregated to generate a feature map. Thus, the kernel size affects the area of the input over which the networks searches for a single value in the output feature map. The kernel slide is defined by the stride value. The value of the stride influences how far the kernel move at each phase of the convolution process. A stride with value one moves the kernel of one value at the time, while a stride with value n moves of n values with the kernel each time it computes a new value for the feature map. Another important part of the feature extraction process implemented by these kernels is the padding: before executing the convolution, additional values are added around the input values. This process aids in controlling the output size and preserving the input's boundary information. **Valid** padding means no padding is applied, while **same** padding means padding at the input boundaries enough values such that the output, assuming stride of 1, has the same dimension of the input.

2.4.2 Downsampling

When applying the convolution operation, depending on the size of the kernel, on the stride value and on the padding, the dimension of the output changes. Thus, downsampling can happen and in the worst case the dimension of the output will be negative, which usually results in a not trainable model. To avoid this, the padding, stride and kernel size must be properly chosen.
The formula to compute the dimension of the output is:

$$D_{out} = \frac{D_{inp} - K + 2P}{S} + 1 \tag{2.2}$$

Where D_{out} and D_{in} are the dimension of output and input respectively, K is the kernel size, P is the padding on the border (so for example with padding set to same $P = 1$, with padding set to valid $P = 0$), finally S is the stride value.

2.4.3 Pooling Layer

Pooling layers serve an essential function in Convolutional Neural Networks (CNNs) by progressively reducing the spatial size (i.e., width and height dimensions) of the input volume. This operation reduces the amount of

parameters and computation in the network, thereby controlling overfitting and reducing computational load. There are 2 main types of pooling operations, the most common of which are Max Pooling and Average Pooling:

- Max Pooling: This operation calculates the maximum value in each patch of the input matrix within a certain window. Max pooling helps to preserve the most active features while reducing the spatial dimensions.
- Average Pooling: This operation calculates the average value for each patch on the input.

These operations are typically performed with a stride larger than one to reduce the size of the output and are often used after convolutional layers. The window size and stride determine how much the input volume is downsampled, again here the values must be chosen such that the model does not end up with a negative dimension. It's worth noting that pooling layers have no learnable parameters, they only perform a fixed operation on the input. These layers greatly help in achieving translation invariance in CNN models, where the network can recognize a feature independently on his position in the data. One potential drawback of pooling operations is the loss of information.

2.4.4 Batch Normalization Layer

Batch Normalization is a deep learning technique that aim to stabilize the learning process by standardizing the inputs to each batch. It aims to solve the problem of internal covariate shift, which occurs when the distribution of each layer's inputs changes during training. It accomplishes this by adding a Batch Normalization layer that normalize the previous activation layer's output, subtracting the batch mean, and dividing by the batch standard deviation. In practice, Batch Normalization aids in regularization, potentially reducing the need for additional techniques such as Dropout.

2.5 Convolutional Neural Networks and Side-Channel Analysis

Side-channel analysis often involves extracting useful information from noisy, high-dimensional data, a process that CNNs are exceptionally well-equipped to handle. The capacity of CNNs to automatically and adaptively learn spatial hierarchies of features from data aligns with the requirements of side-channel analysis. The learnt features are more representative and discriminative, enabling them to effectively identify and classify patterns in side-channel data.

The property of translation invariance of CNNs proves particularly valuable in the context of side-channel analysis. This property ensures that once a useful feature, such as a certain leakage pattern, has been learnt, the CNN can recognize this pattern regardless of its position in the side-channel trace, making CNNs effective tools to work effectively with raw, unprocessed data. This is particularly beneficial in side-channel analysis, where preprocessing steps such as trace alignment can be challenging and time-consuming. Due to the translational invariance property of CNNs, they can handle misaligned traces directly, eliminating the need for a separate alignment step.
Another important notion is that Convolutional Neural Networks are primarily designed to process data that has a grid-like topology (for example with images, they can be viewed as 2 dimensional grid of pixels or 3 dimensional using RGB). In the context of SCA, traces can be interpreted as 1-dimensional grids (i.e., sequences) of data points (or features). Each data point being a power consumption measurement at a particular point in time.

2.6 Hyperparameter optimization

Hyperparameter optimization(HO), also known as hyperparameter tuning, is a critical step in the development of ML algorithms. It entails determining an optimal collection of hyperparameters - the parameters not learned during the training phase - that minimizes or maximizes a preset fit function and so improves the model's performance.
Learning rate, number of layers in a neural network, number of hidden units in each layer, type of activation functions, and many more parameters are examples of hyperparameters. When we initially apply a model, the ideal values for these hyperparameters are often not obvious, and they might be highly problem-dependent.
Common methods for optimizing hyperparameters, also known as Sampling Algorithms, are:

- Grid Search: The simplest strategy, in which we establish a subset of the hyperparameter space and methodically attempt all combinations within the defined grid.
- Random Search: Unlike grid search, random search draws hyperparameter values at random from a given space. When the number of hyperparameters is considerable, this method may be more efficient than grid search.
- Bayesian Optimization: This method creates a posterior distribution of functions that best describes the function to be optimized and then uses it to select the most promising hyperparameters to evaluate in the true function.

The process of optimizing hyperparameters can be time-consuming and computationally expensive. It is, nonetheless, critical for developing strong and accurate ML models.

2.6.1 Exploration-Exploitation trade off

The Exploration-Exploitation trade off is a fundamental concept in HO and, more broadly, in various domains such as reinforcement learning and decision theory. It represents the dilemma faced when deciding whether to utilize existing knowledge (exploitation) or acquire new knowledge (exploration).

- Exploration: This involves searching across a broad range of potential solutions or parameters in order to discover previously unknown but potentially optimal solutions. In the context of hyperparameter optimization, exploration could mean trying out radically different or completely random sets of hyperparameters to see if they might yield better performance.
- Exploitation: This is about refining the current best-known solution or focusing the search around the currently best-known set of hyperparameters.

Balancing the trade-off between exploration and exploitation is crucial in hyperparameter optimization. Too much exploration without enough exploitation can result in spending too much time evaluating poor solutions. Conversely, too much exploitation with insufficient exploration can cause premature convergence to sub-optimal solutions, as it may overlook better solutions lying in unexplored regions of the search space.

2.6.2 Pruning strategies in Hyperparameter Optimization

Pruning strategies in hyperparameter optimization focus on eliminating non-promising hyperparameter configuration as early as possible during the tuning process, thereby saving computational resources. These techniques are particularly crucial for tuning complex machine learning architectures, where the computational cost can be quite high. By eliminating poorly performing configurations early on, pruning strategies allow the hyperparameter search to focus on more promising areas of the search space thus saving both computational resources and time[51]. Different pruning strategies exists [22, 30, 29, 2], These strategies primarily rely on assessing the model's performance at specific intervals during training. Importantly, this performance is not evaluated in isolation, but rather compared to the intermediate performance metrics of other models which were or are being evaluated within the same hyperparameter optimization process. These strategies frequently involve a trade-off between exploration and exploitation. Pruning

strategies were initially viewed as hyperparameter optimization strategies in and of themselves, when applied to grid search or random search, but it has been demonstrated how these strategies can actually enhance Optimization algorithms such as Bayesian Optimization.[50, 51]
While pruning strategies offer clear benefits in terms of computational efficiency, they do present certain risks. One notable concern is the possibility of prematurely dismissing hyperparameter configuartions that may appear suboptimal in the early stages of training but could potentially yield superior performance if further trained. The inherent risk derives from the greedy nature of these strategies, which prioritize hyperparameters that deliver immediate improvements to the objective function. This issue can augment with the double descent phenomena[35], resulting in incorrectly pruning configurations that could be optimal with more training.
Because these Pruning Strategies are effectively a form of early stopping based on previous models, and to avoid confusion with other well-known deep learning pruning techniques [34], in this research they will be referred to as an Early Stopper mechanism rather than pruning strategies.

2.6.3 Tree-structured Parzen Estimator

The Tree-structured Parzen Estimator (TPE)[5] is an algorithm for hyperparameter optimization that has demonstrated its superiority over traditional methods such as grid or random search. As a Sequential Model-Based Optimization (SMBO) approach, TPE constructs a model of the objective function with the goal of suggesting more promising hyperparameters for evaluation.
TPE models two probability distributions: $P(x|y)$ and $P(y)$, where x represents the hyperparameters, and y is the corresponding objective function. Through this modelling, TPE assigns greater weight to regions of the hyperparameter space it deems promising.
The operational procedure of TPE consists of the following steps:

1. TPE starts its process by randomly selecting n hyperparameters configurations from the search space and evaluating the objective function for them.

2. These hyperparameters are then fitted into two distinct distributions: one for those that yielded better outcomes of the objective function (termed "good" values), and another for the remainder of the hyperparameters (termed "bad" values).

3. In subsequent iterations, TPE samples more from regions having a high ratio of good to bad hyperparameter values. This ratio is typically determined by a parameter, γ, which is a quantile threshold separating

the good and bad hyperparameters based on their objective function outcomes.

Through this process, TPE continuously adapts its focus towards the promising areas of the hyperparameter space, while diminishing attention on areas known to yield inferior results.
A significant advantage of TPE is its capability to handle both continuous and discrete hyperparameters, inclusive of conditional hyperparameters. This capability allows TPE to navigate complex search spaces effectively.

2.7 Early Stopping

Early stopping is a regularization strategy used during deep learning model training to prevent overfitting. It consist of ending the training process before the model begins to overfit. The dataset is often separated into three sections to accomplish this: training, validation, and testing. The model's performance on the validation set is evaluated at regular intervals or after each epoch. When performance on the validation set begins to decline or stop improving, even if performance on the training set is still improving, training is terminated. As a result, the model is 'stopped early' to avoid overfitting the training data. The model's state when it performed the best on the validation set is saved and regarded the training outcome. This state is then utilized to evaluate the testing set in order to forecast the model's performance on previously unseen data. Aside from reducing overfitting, early stopping contributes significantly to accelerating hyperparameter optimization. Early stopping can considerably minimize the computing resources (time and hardware) necessary for hyperparameter optimization since it can cease the training process earlier if a model is unlikely to improve. This improves the efficiency of the process of determining ideal hyperparameters, especially when dealing with complex models and large datasets.
However, while early stopping is an effective technique for preventing overfitting and saving computational resources, it's important to note its interaction with phenomena such as the previously mentioned double descent. In certain scenarios, the model's performance on the validation set might temporarily worsen before improving again. If early stopping is applied, the training process might halt prematurely during this phase, potentially preventing the model from reaching an improved performance stage that lies beyond the apparent peak, resulting in a sub-optimal solution.

2.8 Optuna

Optuna is a flexible and user-friendly, open-source software library designed for optimizing hyperparameters in various types of machine learning algorithms.[2]

The Optuna framework offers a structured approach for defining and optimizing hyperparameters, making it suitable for a wide range of machine learning models and frameworks. It provides a unified interface for handling both continuous and categorical hyperparameters.
Optuna's Sampling Algorithms include both traditional strategies like grid search and random search, as well as more sophisticated methods, like Tree-structured Parzen estimator[5], which is used in this study.
One of the distinguishing features of Optuna is its support for automated early stop mechanisms, both based on the learning patience and on the intermediate values of the previous models during the training. These strategies allow Optuna to halt unpromising models early on, saving valuable computational resources. Optuna's early stopping strategies leverage an iterative approach, evaluating the potential of each model based on its performance at various stages during the training process and on the performance at various stages of previous models. This iterative assessment allows Optuna to dynamically allocate resources, focusing more on promising models.
Optuna also includes other key features that contribute to its utility and effectiveness. These include:

- Easy Parallelization: Optuna is designed to support distributed hyperparameter search across multiple processing units, speeding up the HO process.

- Custom Objective Function and Metrics: One of the reasons for using Optuna in this research is its flexibility to integrate custom objective functions for the hyperparameter optimization process. This is beneficial in scenarios where standard metrics may not fully capture the optimization goal. In addition to common metrics like accuracy or F1 score, users can also define and integrate custom metrics based on Guessing Entropy and Mutual Information. This allows a more nuanced control over the optimization process, catering to the specific needs of a project.

- Integration with Machine Learning Libraries: Optuna is designed to be compatible with a wide range of popular machine learning libraries, such as PyTorch, keras, TensorFlow, and scikit-learn, simplifying the process of integrating hyperparameter optimization into existing machine learning workflows.

- Visualization: Optuna provides capabilities for visualizing optimization results, making it easier to track the optimization process and understand the distribution and effects of different hyperparameters

2.9 Ascad Dataset

The ASCAD dataset was introduced in [4] and is now a prominent benchmark in the field of Side-Channel Analysis that is widely used for assessing the effectiveness of SCA countermeasures and comparing different attack methodologies.
The dataset provides a large set of measured traces collected from an 8-bit ATMEGA8515 Microchip, performing AES-128 encryption algotihm. The traces of the subset are cut to only measure the part of the encryption related to the first round of the AES. The version one of the dataset, which is the one utilized in this study, is split into 2 different datasets:

- ASCAD Fixed Key(ASCADf): This dataset is obtained with a unique key used for all the encryptions. This reflects the secnario where the device secret key remains the same over multiple encryptions. The fixed key database is particularly suitable for profiling attacks, where an attacker has access to a device with an identical, fixed key to profile and create a model before attacking the device. This dataset contains 60000 traces, 50000 used for profiling and 10000 for validating and testing.

- ASCAD Random Key (ASCADr): In this dataset, a different key is used for each AES encryption. This simulates a scenario where the device secret key changes frequently between encryptions. This dataset contains 300000 traces, 200000 for profiling and 100000 for validating and testing

Chapter 3

Related Work

In [4] the authors provides an in-depth exploration of deep learning applications for side-channel attacks. Key findings include the effectiveness of CNNs in handling desynchronization in data, suggesting a preference for these models in side-channel analysis despite the complexity in training. To enhance reproducibility and further research, the paper introduces the ASCAD database, which includes measurements from an AES implementation secured against first-order side-channel attacks. This database is now a benchmark for SCA research.
In [23] the authors propose a new training method for CNNs, adapted from models used in the audio domain[36], which achieves high performance across several considered SCA datasets. The paper demonstrates that adding artificial noise to the input traces can significantly improve the performance of the convolutional neural network. It argues that this noise addition equates to a regularization term in the objective function, thus enhancing the model robustness against countermeasures and preventing overfitting. This practice, which was uncommon in side-channel analysis, showed strong efficacy in breaking cryptographic implementations protected with countermeasures. The authors also highlight that it may be impossible to develop a single neural network that will show superior performance over all SCA scenarios due the diversity of leakage patterns that the model needs to learn for different datasets. They therefore suggest that a suite of classifiers tuned for the specific problem might be the most viable solution.
Zaid et al.[54] explores the efficiency and interpretability of CNNs in SCA. CNNs demonstrate significant performance advantages over template attacks, but their performance heavily relies on the selection of hyperparameters. To better understand and optimize these parameters, visualization techniques such as Weight Visualization, Gradient Visualization, and Heatmaps are introduced. They propose a methodology for building efficient CNN architectures even under desynchronization, demonstrating superior performance and significantly reduced complexity compared to previous

state-of-the-art CNN models. The paper also argue that increasing the filter size reduces the effectiveness of the network and slow down training time. On the other hand, they argue smaller filter size concentrates the information better since with a small size you will have more weight associated with the information extracted by the filters.
Wouters et al.[52] reviews the most recently discussed work of Zaid et al in [54]. The authors of the review manage to reduce the complexity of the CNNs proposed in [54] by an average of 52% while maintaining similar performance. The authors argue that increasing the filter size can actually improve the performance of the network. This is contrary to what is argued in [54] where larger filters supposedly led to reduced network confidence. The review also emphasize the importance of pre-processing side-channel traces, and shows that with proper pre-processing the first convolutional block proposed in [54] to extract features can be omitted, which contributes to reducing the complexity of the models.
The paper [38] presents a novel methodology for optimizing the training process in deep learning-based side-channel analysis. It introduces a metric based on the concept of mutual information, which examines how information is propagated through the neural network's hidden layers. The authors validate this approach through extensive experimentation on three masked AES implementations, and the results suggest that the proposed metric offers great performance in determining the optimal training epoch, thus avoiding overfitting. Additionally, it significantly reduces computational overhead, as it doesn't require computation for all key hypotheses like the traditional key ranking metric.
Building upon the approach outlined in [38], the authors of [41] also presents a methodology that enhances the hyperparameter tuning process by addressing the challenge of choosing the optimal epoch at which the networks performs best suggesting the utilization of guessing entropy (GE) as a better metric to determine the best epoch. However, the challenge lies in the computational and temporal costs associated with the calculation of GE for every training epoch. To resolve this, the authors propose a new approach called Fast Guessing Entropy (FGE). FGE significantly reduces the number of validation traces required for the calculation of GE during training, thus making the process faster and more efficient. The results highlight that this method doesn't impact the performance of the trained models but enables the hyperparameter tuning process to be much more rapid, allowing for more detailed tuning and improved model selection.
In [40], the authors introduce AISY, an open-source framework designed to streamline deep learning based side-channel analysis. The key intent of the authors is to address reproducibility issues in SCA research. The platform offers a range of user friendly features that aim to make deep learning side-channel analysis research easier and more standardized. Unfortunately, the project is currently not being updated but it represents a good start for the

side-channel analysis community to find some common frameworks that can be expanded to facilitate the research in this field.
Rijsdijk et al.[45] propose an innovative approach to hyperparameter tuning for deep learning-based Side-Channel Analysis using reinforcement learning. They leverage a popular reinforcement learning paradigm known as Q-Learning to automate the search for effective Convolutional Neural Networks (CNNs). Results indicate that their approach can effectively discover performant CNNs, paving the way for the automated search of optimal hyperparameters in deep learning models.
The authors of [53] also suggest an automated method for modifying the hyperparameters of deep learning models used in side-channel analysis, however instead of employing reinforcement learning, the framework they develop, called autoSCA, employs Bayesian optimization. This framework also supports metrics that are more suitable for SCA such as Guessing Entropy. Their experimentation demonstrates that Bayesian optimization performs well in a variety of settings. They also point out that neural networks generated by random search can perform well, indicating that the available datasets are very easy to break. This is the first paper to use Baesyan optimization to hyperparameter optimization in the context of SCA, providing a stable method for building high-performing neural networks for SCA without prior knowledge of the datasets to be attacked. They also point out that, while reinforcement learning can also produce suitable hyperparameters for a successful side channel [45], the search process takes longer.

Chapter 4

Our approach & methodologies

4.1 Study Settings

The experiments utilize both the ASCADf and the ASCADr datasets.
For both we use the label obtained by using the Identity Leakage Model.
For the computational aspect, all the deep learning models were trained on a NVIDIA GeForce GTX 1080 Ti Graphics Processing Unit (GPU).
The implementation of the experiment was conducted using Python. The TensorFlow[1] and Keras[11] libraries served as the foundation for constructing and training the deep learning models, while Optuna[2] was employed for the optimization of hyperparameters. For the purpose of visualizing results and monitoring the progress of our experiments, we used matplotlib[20], TensorBoard, and Optuna's built-in visualization features. These tools allowed us to effectively track, analyze, and present the outcomes of our research in a clear and intuitive manner.
The codebase was built upon an existing framework developed by Léo Weissbart, Member of the Digital Security Group at Radboud University, ensuring a robust foundation for the experimental setup.

4.2 Methodologies

In this research, we leveraged the power of Optuna to perform hyperparameter optimization (HO) experiments on two Side-Channel Analysis (SCA) datasets: ASCADr and ASCADf[4], with the identity leakage model.
The Optuna experiments are executed each for 100 models. In these experiments, the HO process is guided using a customized objective function based on GE. Detailed procedures of this optimization method can be found in the subsection 4.3.1.

A specific search space for the HO was defined, which allowed us to systematically explore various combinations of hyperparameters. The details of this design are discussed in the subsection 4.3.2.
We combined the capabilities of Optuna[2] and Keras[11] to dynamically construct Convolutional Neural Network (CNN) models, the structure and parameters of which were determined by the Tree-structured Parzen Estimator (TPE) Sampling Algorithm used in the HO process. This approach to creating CNN models is further elaborated in the subsection 4.3.3
To enhance the efficiency of our experiments and explore the potential benefits of Optuna's early stop strategies in the realm of SCA, we conducted four tests for each dataset. Each test utilized the Median-Early-Stopping and Patience-Early-Stopping techniques with varying degrees of intensity, adjusted by manipulating the warm-up values. For ASCADf, an additional experiment was conducted without employing any early stopping method, providing a benchmark against which to assess the impact of these resource-saving techniques. Comprehensive information about these strategies is provided in the subsection 4.3.4.
Finally, we assessed the experiments by selecting the best models from each and comparing their performance metrics. We compared the different models using GE. This comprehensive evaluation methodology is further explained in the subsection 4.3.5.

4.2.1 Hyperparameter Optimization Process

Optuna was selected as the primary tool for conducting a series of HO experiments. The Tree-structured Parzen Estimator (TPE) Sampling Algorithm, a Bayesian optimization algorithm provided by Optuna, was employed for the hyperparameter configuration selection.
An important component of the approach was the use of the averaged GE as the objective function to guide the HO process:

$$GE_{average} = \frac{GE(1) + GE(2) + ...GE(n)}{n} \tag{4.1}$$

Where n is the the size of the subset of traces taken from the validation set to estimate the key rank, which correspond to the number of used traces to compute the GE during the training. Utilizing the averaged GE as a metric during the HO process is helpful in comparing the efficiency between different models, since this metric gives an estimation of both the amount of traces needed to reach a low GE, and the amount of attempts needed for guessing the correct value.
Optuna ultimately utilized this number as an objective value to guide the Hyperparameter selection of the future models based on TPE Algorithm.

4.2.2 Hyper parameter search space

Table 4.1 shows the hyperparameters and their respective ranges that have been explored during the hyperparameter optimization. These ranges resulted in 59040000 different combinations (or $2^{25.81518938561}$). For the number of epochs, we used 200 for ASCADf, 300 for ASCADr, but since early stopping strategies were applied, most of the models stopped the training process before this value.

Hyperparameter	Range	Step
batch_size	[100, ..., 1000]	100
learning_rate	[1e-5] (Fixed)	None
activation_function	[relu, selu]	None
filters	[2, ...,64]	$\times 2$
num_conv_layers	[2, ..., 5]	1
kernel_size	[2,3,5,7,11,17] (for each conv layer)	None
num_fc_layers	[1, 2, 3]	1
size_fc_layers	[64, ..., 512] (for each fc layer)	64
epochs	200 or 300 (fixed)	None
strides	1 (fixed)	None

Table 4.1: Hyperparameter space explored in the optimization process

The code in appendix 6.1 shows how we use the Optuna model class to search the hyperparameters space of table 4.1 during the optimization process. A different hyperparameter is optimized in each line. The model suggest techniques are used to recommend values for each hyperparameter within a given range or set of alternatives, while adhering to the optimization algorithm's guidelines, in this case TPE. This process is repeated for each model (or model), after which the hyperparameter probabilities are changed using the TPE algorithm, providing the recommend methods with new probabilities for each value to be picked based on the prior models.

4.2.3 The creation of CNN models

Figure 4.1 give an overview of the general architecture of the dynamically built models. We feed the input to a series of convolutional blocks (2 to 5, as indicated by table 4.1), after the convolutions are applied, a flatten layer is used to reduce the multidimensional input to a single dimension, as commonly required in the transition from the convolution layer to the full connected layer. After the dense layer operations, the output is finally feed to a softmax output layer with number of units equal to the number of output

classes, such that the output corresponds to the probability distribution over the possible classes. Each Convolutional block consist of a Convolutional

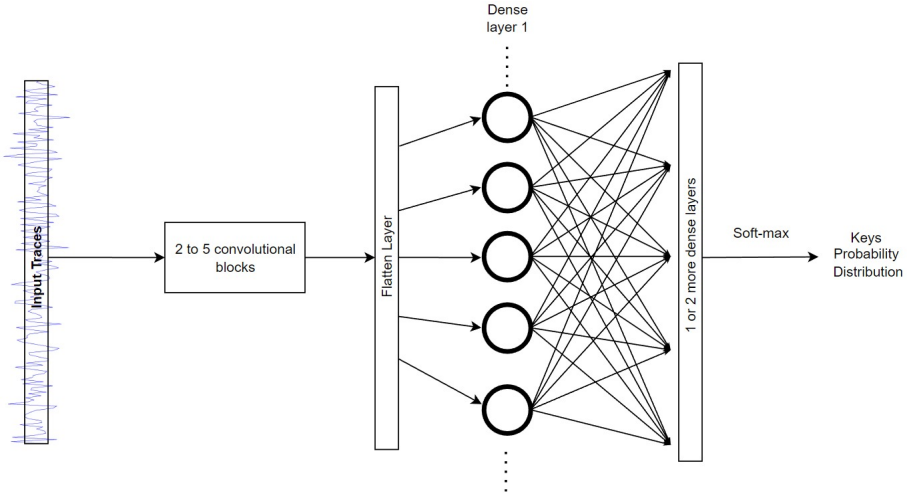


Figure 4.1: General overview of how the convolutional neural networks are built based on the search space of table 4.1

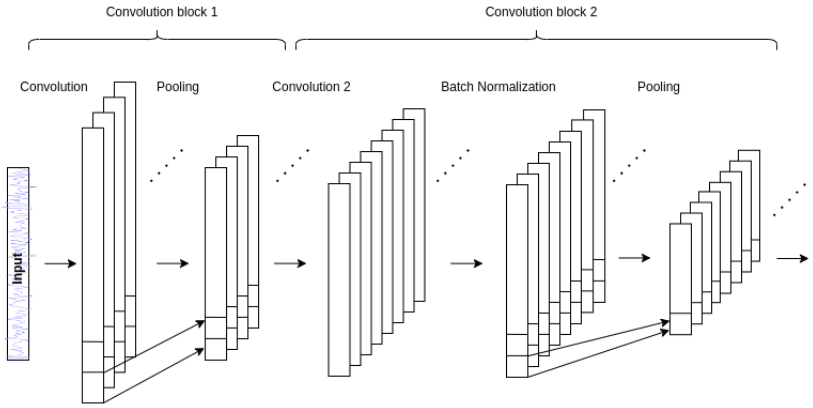


Figure 4.2: First 2 Convolutional blocks from architecture design in 4.1

Layer and an Average pooling layer with size 2, meaning it takes 2 values from the filters and map them to the average of them. Furthermore, every 2 convolutional layer there is also a batch normalization layer between the convolutional and the pooling layer. A graphic representation of the design of the Convolutional Blocks is given in Figure 4.2. The python code used to generate these models is given in 6.2.
The choice to vary the kernel sizes within convolutional layers and the number of units within fully connected layers in the same model allows us to cover a wider variety of architectural possibilities. By offering a range of

values for the kernel sizes and number of units, the optimization process can explore different model structures and find the ones that are the most suitable for the given problem. This approach enhances our model's adaptability and increases the chance of discovering architectures that perform well on the specific dataset.

4.2.4 Early stopping

In this subsection we go into the details on how we decided to implement the cost effectiveness of our framework. Two different early stopping strategies have been used in this study to guide the hyperparameter search towards promising configurations and save computational resources:Early stopping based on patience and Early stopping based on previous models intermediate values

- **Median-Early-Stopper** The Optuna median pruner is utilised in this research, which we refer to as an Median Early Stopper. This mechanism stops unpromising models at the early stages of the training, essentially acting as a form of early stopping. Specifically, the early stopper employed in this research was the median variant. This Early Stopper observes the objective intermediate results at each epoch - in our case, the Averaged Guessing Entropy of queation 4.1 on the validation set. It stops the model at a certain epoch if the model's intermediate result is worse than the median of the intermediate values of the previous models at the same epoch. This mechanism enables us to terminate models that are unlikely to outperform the best-so-far model, focusing more computational resources on other potentially more promising configurations.

  Different warm-up values for the Early Stopper have been experimented with in this study, specifically 30, 50, 75, and 100. The warm-up period defines the number of initial epochs of a model where early stopping does not occur. This is beneficial because it allows models to establish some foundation learning before being considered for early stopping. Different warm-up values were used to analyze the impact of early-stage learning stability on the overall performance of the hyperparameter optimization.

  The Median Early Stopper was chosen for its simplicity and ease of understanding, making it an appropriate choice for the first introduction of early stopping strategies in side-channel analysis. Nonetheless, it's worth noting that the Median Early Stopper is relatively basic, and there exist more advanced early stopping strategies such as hyperband[30]. We encourage future work in this domain to investigate these more sophisticated techniques to further enhance hyperparameter optimization in the context of side-channel analysis.

- **Patience-Early-Stopper**: In conjunction with Median Early Stopper, Early stop based on the patience of the model learning process has also been implemented. More specifically, during a model training phase, if no improvement of the objective function is observed for a patience number of epochs, the training is stopped. This approach offers an additional level of control over resource allocation during training. It enables for the termination of individual models that appear to have converged to a value and that further training does not appear to enhance, even if the achieved performance is not promising.

  In this experiments, the patience value for early stopping was set to 50, as we use anyway a relatively small number of epochs to train (200-300). This means that if the Fast Guessing Entropy did not improve for 50 epochs, the training of that specific model was terminated. This choice further encouraged the optimization process to quickly discard models that looked unpromising.

4.2.5 The evaluation of the models

As previously explained, by utilizing the customizable objective function facility provided by Optuna, we could directly embed the averaged GE into the Sampling Algorithm for hyperparameter optimization process. This integration not only helped to assess the performance of the models but also served as an intermediate metric for each training epoch.
The averaged GE given by equation 4.1 was calculated during the model's training for each epoch. This value was then used by both the early stopping mechanisms to decide whether to continue the training of a specific model or to stop it, thereby ensuring resources were allocated more effectively.
More specifically the calculation of the Averaged GE was computed as follows:

- Subset Selection: For each epoch, a random subset of 600 traces was selected from the relatively small validation set (2000 traces) to maintain computational efficiency while still providing a robust estimation of the model's performance. This approach is in accordance with the findings presented in [41]

- Guessing Entropy Computation: Key rank for each trace in the subset was calculated. This measures the difficulty of guessing the correct secret key. The individual guessing entropies were accumulated as per the conventional process in guessing entropy computations.

- Average Guessing Entropy: The sum of guessing entropies was divided by the total number of traces in the subset, resulting in an average guessing entropy. This provided an estimation of the effort (number of traces) the model required to find the correct subkey, as a low value

would correspond to only a few traces before reaching probability of 0 whereas a high value would correspond to a high amount of traces required to reach zero, if reached at all.

At the end of the experiment, the model that reached the lowest averaged GE was selected as best model for that experiment. Figure 4.4 shows the best models for ASCADr, where each models is from a different experiment using a different warm up values. Figure 4.3 shows the ASCADf best models for the experiments with warm up 30, 50, 75, 100 and no early stop respectively.

4.3 Results

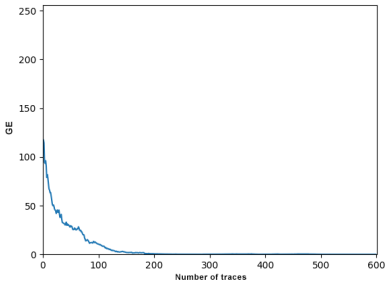
This section of our research delves into the results from our analysis of the two distinct datasets: ASCADf and ASCADr. We focus primarily on the Optuna hyper parameter optimization experiments conducted at various warm-up values for the Median Early Stopper , methodically studying their impact on resource use, measured in number of epochs, and the performance of the best model. This examination allows us to determine the effects of early stopping on the best models' outcomes for each unique experiment.

4.3.1 Experiments for ASCADf

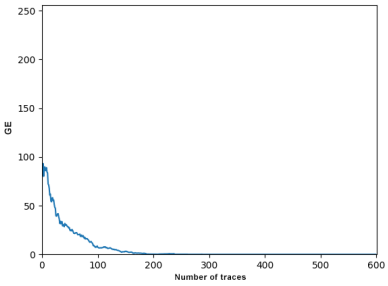
Warm-up	Median-Early-Stopper	Patience-Early-Stopper	Sum
30	49.085	2.68	51.765
50	36.825	7.34	44.165
75	9.975	24.82	34.795
100	4.79	33.27	38.06

Table 4.2: Summary of warm-up values and effects early stop on number of epochs for ASCADf

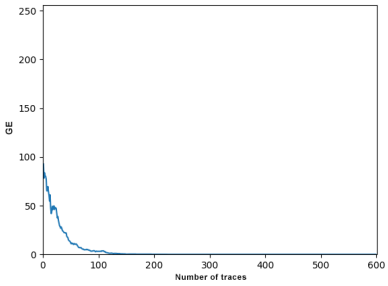
Table 4.2 encapsulates the impact of various warm-up periods on resource utilization during hyperparameter optimization processes. The values represented in the table correspond to the percentages of epochs spared due to the implementation of the 2 different early stopping strategies, and their combined effect.
Specifically, the Median Early Stopper column denotes the percentage of epochs saved because of the application of this strategy, the Patience Early Stopper column represents the percentage of saved epochs attributed to early stopping based on patience. Lastly, the Sum column aggregates these savings, providing an overall view of the resource conservation.
The total savings (represented by the Sum column), which combines the impacts of both the different early stopping strategies, doesn't exhibit a



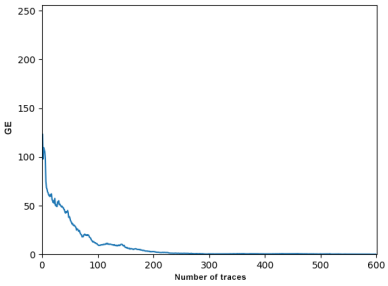
(a) Best model for ASCADf with warm up set to 30



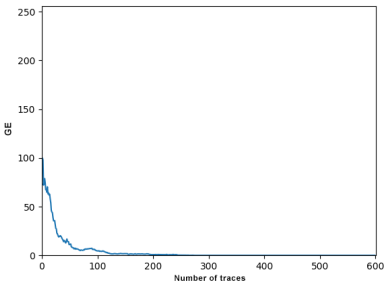
(b) Best model for ASCADf with warm up set to 50



(c) Best model for ASCADf with warm up set to 75



(d) Best model for ASCADf with warm up set to 100



(e) Best model for ASCADf with no early stopping

Figure 4.3: Comparisons of best models for ASCADf with different warm up settings and no early stopping.

direct relationship with the warm-up period. This follows from the fact that a lighter Early stop based on previous models lead to more space for the Early stop mechanism based on training patience to act.
From the data collected in Table 4.2 and corresponding performance drawn from Figure 4.3, we can analyze the relations between the warm-up steps applied to the training and the performance outcomes of the optimal model in the HO process.
The setting with a minimal warm-up phase of 30 steps realizes significant resource savings, with nearly half of the epochs conserved. Notably, the model's performance under this condition doesn't compromise drastically as it recovers the key with approximately 200 traces, like the rest of the configurations. This suggests that even an aggressive early stop schedule can still result in optimal model performance.
Extending the warm-up phase to 50 steps results in a slight decrease in epoch savings. However, the performance of the model remains robust, requiring around 200 traces for key recovery, which aligns with the performance exhibited by other configurations.
Interestingly, brings about a similar performance to the other settings, requiring around 200 traces for key recovery. Despite less pronounced resource savings from early stopping, the configuration with a warm-up phase of 75 steps improves over other configurations by achieving this with less than 200 traces.
Increasing the warm-up phase to 100 steps, or even having no early stop at all, maintains a consistent performance level, necessitating around 200 traces for key recovery. This finding reinforces the premise that strategic early stop doesn't compromise finding optimal configurations, but it might instead ensures efficient resource allocation and exploration-exploitation balance.

Attempt	**Traces to reach GE = 0**
[4]	1476
[54]	191
[45]	202
[53]	257
this study	< 200

Table 4.3: Comparison with other papers of number of traces required to reach a GE of 0 for ASCADf, ID leakage Model

4.3.2 Experiments for ASCADr

Table 4.4 summarizes the effects of different warm-up values on the 2 early stopping strategies for the ASCADr dataset.

Warm-up	Median-Early-Stopper	Patience-Early-Stopper	Sum
30	33.3	14.44	44.74
50	35.63	18.09	53.72
75	24.0	38.12	62.12
100	0.0	39.11	39.11

Table 4.4: Summary of warm-up values and effects of early stop on number of epochs for ASCADr

When the warm-up value is set to 30, the Median Early Stopper contributes to 33.30% saving in epochs, while Patience Early Stopping approximately 14.44%, reaching the total reduction of epochs to approximately 44.74%. Surprisingly, this figure is slightly lower than the reduction seen for some of the higher warm up values (50 and 74), even though a lower warm up value would usually mean more aggressive early stopping.
This counterintuitive result can be attributable to the fact that during this particular experiment, effective configurations were relatively quickly identified, and experiment showed consistent improvements as the models were executed. Considering the Median Early stop mechanism, this scenario naturally led to a decrease and delay of early stopping compared to other scenarios.
Although a lower warm up value generally implies more aggressive early stopping, it is essential to consider the dynamics of the optimization process and the quality of the configurations with the order in which they are found.
At a warm-up value of 50, Median Early Stopper models saved about 35.63% of the total epochs that could have been run, and the early stop saved approximately 18.09% of the epochs. The combined effect led to a reduction of around 53.72% in the number of epochs.
Increasing the warm-up value to 75, the impact of the Median Early Stopper savings decreases to 24.0%, but the effect of early stopping increases to 38.12%, leading to a total reduction of approximately 62.12%.
At a warm-up value of 100, the Median Early Stopper has no effect (0.0% reduction), while the impact of early stopping slightly increases to 39.11%, Leading to the overall reduction in epochs to 39.11%.
It's important to explain that the elevated percentage of pruned epochs in the ASCADr experiments, compared to those in the ASCADf experiments, is primarily attributable to the baseline number of epochs set for each dataset. In the ASCADr experiments, the default number of epochs, was set to 300. In contrast, for the ASCADf experiments, this number was lower at 200.
This discrepancy means that, since the warm-up and early stopping values are kept constant across both sets of experiments, the resource-saving techniques, have a larger pool of epochs from which to cut in the ASCADr

experiments. As a result, these techniques appear to prune a higher percentage of epochs in the ASCADr experiments compared to the ASCADf ones. This difference, however, is simply a reflection of the different default epoch settings for the two datasets, rather than indicating any inherent differences in the effectiveness of the early stopping strategies.

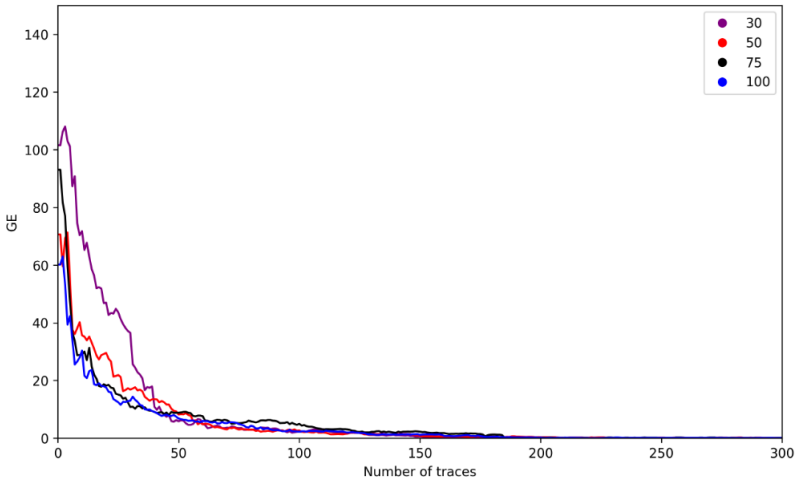


Figure 4.4: Guessing entropy for the ASCADr experiments using warm up value of 50, 75 and 100

Analyzing the outcomes of the best models' Guessing Entropy (GE) from various experiments of the ASCADr dataset, depicted in Figure 4.4, reveals comparable results across the board for different warm-up values, specifically 30, 50, 75, and 100. Interestingly, the model from the Optuna experiment with a warm-up value of 30 exhibited slightly superior performance, reaching a GE of 0 with less than 200 traces. Meanwhile, the best models from experiments with warm-up values of 50, 75, and 100 also demonstrated commendable performance, requiring between 205 and 220 traces to achieve a GE of 0.
Once more, despite the amounts of computational resources saved across the different warm-up values, all experiments resulted in well performing models. These results suggest that the role of early stopping strategies can be effective in the HO process and can help to focus the process on more promising configurations and lead to better results.

Attempt	**Traces to reach GE = 0**
[39]	105
[45]	490
[53]	1568
this study	< 200

Table 4.5: Comparison with other papers of number of traces required to reach a GE of 0 for ASCADr, ID leakage Model

Chapter 5

Conclusions

Regardless of the warm-up values, all the experiments resulted in models that consistently exhibited robust performance, achieving a GE of 0 with approximately 200 traces on average across both datasets. This pattern underscores the potential effectiveness of Optuna's early stopping mechanisms during hyperparameter optimization (HO) in Side-Channel Analysis (SCA). These mechanisms can effectively modify the dynamics of the exploration-exploitation trade-off in HO, directing the process more efficiently towards promising regions of the hyperparameter space.

Models with a warm-up value of 30 demonstrated a slight edge in performance, achieving a GE of 0 with less than 200 traces for ASCADr and about 200 traces for ASCADf. This observation suggests that aggressive Early Stop strategies, characterized by a lower warm-up value, can still yield optimal models. This approach could expedite the optimization process by directing it towards promising configurations earlier.

However, these observations should be interpreted with caution. Numerous factors could influence the observed effects, such as the number of trials in the Optuna studies, the relatively simple nature of the models, and the vastness of the search space. In addition, the models used in SCA are relatively simple, which could contribute to their resilience against common challenges in machine learning such as overfitting. Given the less complex nature of the datasets, it's possible that a wide array of configurations could result in high-quality models.

One noteworthy aspect of these findings is the facilitation offered by the Optuna framework, which comes with state-of-the-art HO, such as the Tree-structured Parzen Estimator (TPE), used in this study. The usage of this robust, well-maintained, open-source framework simplified the overall workflow. Optuna's adaptability demonstrates that 'reinventing the wheel' with bespoke frameworks for SCA may not be necessary. Utilizing established ML frameworks, which can be readily adapted to the specific needs of SCA, can save time and streamline research, further promoting progress in this

field.
In conclusion, these findings lend support to the application of early stopping strategies in HO studies within the context of SCA and underscore the value of leveraging existing ML frameworks like Optuna. The findings also underscore the potential benefits of integrating such open-source frameworks, given their robust, ready-to-use, and well-tested features.

5.1 Future work

Given the exploratory nature of this study, with several new elements integrated into CNN-based SCA within the project's constraints, it lays down a groundwork for future explorations. The following are potential directions for future work seeking to further expand on this research::

- One significant avenue for future research lies in the exploration and investigation of a wider variety of early stopping strategies. The scope of the present study was limited to a few values for the warm-up and a singular fixed value for the patience parameter in the early stopping mechanism. To better comprehend the dynamics and interactions of these parameters, a more granular and comprehensive investigation could prove to be beneficial. Identifying the optimal balance and specific values for these parameters could significantly enhance the efficiency and performance of the hyperparameter optimization process. Furthermore, a comparative analysis between studies employing only one of the early stopping strategies and those utilizing both could yield relevant insights. Such research could reveal how these strategies individually and collectively influence the performance and resource utilization of the hyperparameter optimization.

- A second area for future research would be to delve deeper into various Early Stopping strategies. The current study was restricted to the application of the basic median Early Stopper. However, more sophisticated and diverse Early stopping techniques exist and could be investigated. For instance, percentile Early Stop with differing parameters could be evaluated to uncover their potential benefits and drawbacks. More advanced Early Stopping strategies, such as Hyperband[30], should also be examined.

- Thirdly, the exploration of other state-of-the-art hyperparameter optimization algorithms, which are already integrated within Optuna, presents an exciting direction for future work. In particular, the application of multi-objective hyperparameter optimization algorithms like MOTPE(Multi Objective Tree Parzen Estimator)[37], NSGA-II[12], and NSGA-III[13, 21] could be incredibly valuable in the context of Side-Channel Analysis (SCA). These advanced algorithms allow the

simultaneous consideration of multiple SCA metrics to guide the hyperparameter optimization process. It would be interesting to see if using MI and FGE at the same time could bring to a better Hyperparameter Optimization process

Bibliography

[1] 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, Dandelion 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. Software available from tensorflow.org.

[2] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 2623–2631, 2019.

[3] Mohammad Abdullah Al Faruque, Sujit Rokka Chhetri, Arquimedes Canedo, and Jiang Wan. Acoustic side-channel attacks on additive manufacturing systems. In *2016 ACM/IEEE 7th International Conference on Cyber-Physical Systems (ICCPS)*, pages 1–10, 2016.

[4] Ryad Benadjila, Emmanuel Prouff, Rémi Strullu, Eleonora Cagli, and Cécile Dumas. Deep learning for side-channel analysis and introduction to ASCAD database. *Journal of Cryptographic Engineering*, 10(2):163–188, 2020.

[5] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. *Advances in neural information processing systems*, 24, 2011.

[6] Shivam Bhasin, Anupam Chattopadhyay, Annelie Heuser, Dirmanto Jap, Stjepan Picek, and Ritu Ranjan. Mind the portability: A warriors guide through realistic profiled side-channel analysis. In *NDSS 2020-Network and Distributed System Security Symposium*, pages 1–14, 2020.

[7] Eli Biham and Adi Shamir. Differential fault analysis of secret key cryptosystems. In *Advances in Cryptology—CRYPTO'97: 17th Annual International Cryptology Conference Santa Barbara, California, USA August 17–21, 1997 Proceedings 17*, pages 513–525. Springer, 1997.

[8] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In *Cryptographic Hardware and Embedded Systems-CHES 2004: 6th International Workshop Cambridge, MA, USA, August 11-13, 2004. Proceedings 6*, pages 16–29. Springer, 2004.

[9] Martin Brisfors and Sebastian Forsmark. DLSCA: a tool for deep learning side channel analysis. *Cryptology ePrint Archive*, 2019.

[10] Suresh Chari, Josyula R Rao, and Pankaj Rohatgi. Template attacks. In *Cryptographic Hardware and Embedded Systems-CHES 2002: 4th International Workshop Redwood Shores, CA, USA, August 13–15, 2002 Revised Papers 4*, pages 13–28. Springer, 2003.

[11] François Chollet et al. Keras. `https://keras.io`, 2015.

[12] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

[13] Kalyanmoy Deb and Himanshu Jain. An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part i: Solving problems with box constraints. *IEEE Transactions on Evolutionary Computation*, 18(4):577–601, 2014.

[14] Julie Ferrigno and M Hlaváč. When AES blinks: introducing optical side channel. *IET Information Security*, 2(3):94–98, 2008.

[15] Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, 36(4):193–202, 1980.

[16] Karine Gandolfi, Christophe Mourtel, and Francis Olivier. Electromagnetic analysis: Concrete results. In *Cryptographic Hardware and Embedded Systems—CHES 2001: Third International Workshop Paris, France, May 14–16, 2001 Proceedings 3*, pages 251–261. Springer, 2001.

[17] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[18] Benjamin Hettwer, Stefan Gehrer, and Tim Güneysu. Applications of machine learning techniques in side-channel attacks: a survey. *Journal of Cryptographic Engineering*, 10:135–162, 2020.

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

[20] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[21] Himanshu Jain and Kalyanmoy Deb. An evolutionary many-objective optimization algorithm using reference-point based nondominated sorting approach, Part II: Handling constraints and extending to an adaptive approach. *IEEE Transactions on Evolutionary Computation*, 18(4):602–622, 2014.

[22] Kevin Jamieson and Ameet Talwalkar. Non-stochastic best arm identification and hyperparameter optimization. In *Artificial intelligence and statistics*, pages 240–248. PMLR, 2016.

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

[24] Taehun Kim and Youngjoo Shin. Thermalbleed: A practical thermal side-channel attack. *IEEE Access*, 10:25718–25731, 2022.

[25] Neal Koblitz. Elliptic curve cryptosystems. *Mathematics of computation*, 48(177):203–209, 1987.

[26] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *Advances in Cryptology—CRYPTO'99: 19th Annual International Cryptology Conference Santa Barbara, California, USA, August 15–19, 1999 Proceedings 19*, pages 388–397. Springer, 1999.

[27] Paul C Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In *Advances in Cryptology—CRYPTO'96: 16th Annual International Cryptology Conference Santa Barbara, California, USA August 18–22, 1996 Proceedings 16*, pages 104–113. Springer, 1996.

[28] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[29] Liam Li, Kevin Jamieson, Afshin Rostamizadeh, Ekaterina Gonina, Jonathan Ben-Tzur, Moritz Hardt, Benjamin Recht, and Ameet Talwalkar. A system for massively parallel hyperparameter tuning. *Proceedings of Machine Learning and Systems*, 2:230–246, 2020.

[30] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *The Journal of Machine Learning Research*, 18(1):6765–6816, 2017.

[31] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power analysis attacks: Revealing the secrets of smart cards*, volume 31. Springer Science & Business Media, 2008.

[32] Victor S Miller. Use of elliptic curves in cryptography. In *Conference on the theory and application of cryptographic techniques*, pages 417–426. Springer, 1985.

[33] Tom Michael Mitchell et al. *Machine learning*, volume 1. McGraw-hill New York, 2007.

[34] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. *arXiv preprint arXiv:1611.06440*, 2016.

[35] Preetum Nakkiran, Gal Kaplun, Yamini Bansal, Tristan Yang, Boaz Barak, and Ilya Sutskever. Deep double descent: Where bigger models and more data hurt. *Journal of Statistical Mechanics: Theory and Experiment*, 2021(12):124003, 2021.

[36] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

[37] Yoshihiko Ozaki, Yuki Tanigaki, Shuhei Watanabe, and Masaki Onishi. Multiobjective tree-structured parzen estimator for computationally expensive optimization problems. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, GECCO '20, page 533–541, New York, NY, USA, 2020. Association for Computing Machinery.

[38] Guilherme Perin, Ileana Buhan, and Stjepan Picek. Learning when to stop: a mutual information approach to fight overfitting in profiled side-channel analysis. *Cryptology ePrint Archive*, 2020.

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

[40] Guilherme Perin, Lichao Wu, and Stjepan Picek. AISY-deep learning-based framework for side-channel analysis. *Cryptology ePrint Archive*, 2021.

[41] Guilherme Perin, Lichao Wu, and Stjepan Picek. The need for speed: A fast guessing entropy calculation for deep learning-based SCA. *Algorithms*, 16(3):127, 2023.

[42] Stjepan Picek, Annelie Heuser, Alan Jovic, Shivam Bhasin, and Francesco Regazzoni. The curse of class imbalance and conflicting metrics with machine learning for side-channel evaluations. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2019(1):1–29, 2019.

[43] Stjepan Picek, Guilherme Perin, Luca Mariot, Lichao Wu, and Lejla Batina. Sok: Deep learning-based physical side-channel analysis. *ACM Computing Surveys*, 55(11):1–35, 2023.

[44] Jean-Jacques Quisquater and David Samyde. Electromagnetic analysis (EMA): Measures and counter-measures for smart cards. In *Smart Card Programming and Security: International Conference on Research in Smart Cards, E-smart 2001 Cannes, France, September 19–21, 2001 Proceedings*, pages 200–210. Springer, 2001.

[45] Jorai Rijsdijk, Lichao Wu, Guilherme Perin, and Stjepan Picek. Reinforcement learning for hyperparameter tuning in deep learning-based side-channel analysis. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 677–707, 2021.

[46] Ronald L Rivest. Cryptography and machine learning. In *Advances in Cryptology—ASIACRYPT'91: International Conference on the Theory and Application of Cryptology Fujiyosida, Japan, November 1991 Proceedings 2*, pages 427–439. Springer, 1993.

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

[48] François-Xavier Standaert, Tal G Malkin, and Moti Yung. A unified framework for the analysis of side-channel key recovery attacks. In *Advances in Cryptology-EUROCRYPT 2009: 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cologne, Germany, April 26-30, 2009. Proceedings 28*, pages 443–461. Springer, 2009.

[49] Nicolas Veyrat-Charvillon, Marcel Medwed, Stéphanie Kerckhof, and François-Xavier Standaert. Shuffling against side-channel attacks: A comprehensive study with cautionary note. In *Advances in Cryptology–ASIACRYPT 2012: 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings 18*, pages 740–757. Springer, 2012.

[50] Jiazhuo Wang, Jason Xu, and Xuejun Wang. Combination of hyperband and bayesian optimization for hyperparameter optimization in deep learning. *arXiv preprint arXiv:1801.01596*, 2018.

[51] Martin Wistuba, Nicolas Schilling, and Lars Schmidt-Thieme. Hyperparameter search space pruning–a new component for sequential model-based hyperparameter optimization. In *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2015, Porto, Portugal, September 7-11, 2015, Proceedings, Part II 15*, pages 104–119. Springer, 2015.

[52] Lennert Wouters, Victor Arribas, Benedikt Gierlichs, and Bart Preneel. Revisiting a methodology for efficient CNN architectures in profiling attacks. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 147–168, 2020.

[53] Lichao Wu, Guilherme Perin, and Stjepan Picek. I choose you: Automated hyperparameter tuning for deep learning-based side-channel analysis. *IEEE Transactions on Emerging Topics in Computing*, 2022.

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

Chapter 6

Appendix

6.1 Selecting Hyperparameters From the Search Space

6.1.1 retrieve hyperparameters

Listing 6.1: Code that uses the Optuna trial class to retrieve values for the hyperparameters to build new configurations in the hyperparameter optimization process

```
num_epochs = 300
batch_size = trial.suggest_int("batch_size",
                               low=100,
                               high=1000,
                               step=100,
                               log=False)
learning_rate = 1e-5
activation_function = trial.suggest_categorical(
                        "activation_function",
                        ["relu", "selu"])
filters = trial.suggest_categorical(
                        "filters",
                        [2,4,8,16,32,64])
num_conv_layers = trial.suggest_int(
                        "num_conv_layers",
                        low=2,
                        high=5,
                        step=1,
                        log=False)



kernel_size =[trial.suggest_categorical(f"kernel_size{i+1}",
    [2,3,5,7,11,17]) for i in range(num_conv_layers)]

num_fc_layers = trial.suggest_int("num_fc_layers", low=1, high
    =3, step=1, log=False)
```

```
size_fc_layers = [trial.suggest_int(f"fc_layer{i+1}", low=64,
    high=512, step=64) for i in range(num_fc_layers)]
```

In each convolutional layer, the hyperparameter that defines the kernel size is chosen individually. The same holds for the fully connected layers, where the number of units is also suggested individually for each fully connected layer.

6.1.2 create CNN models

Listing 6.2: this function take the given hyper parameters from Optuna Trial class to dynamically build CNN architectures with those hyperparameters

```
def create_convnet(self,
            trace_length,
            n_output,
            activation='relu',
            kernel_size=3,
            filters=8,
            num_conv_layers=3,
            num_fc_layers=2,
            size_fc_layers=512,
            strides=1,
            learning_rate=1e-05,
            loss_func='categorical_crossentropy'):

    # Define the input shape
    input_shape = (trace_length, 1)

    # Create the model
    model = Sequential()

    # Add convolutional layers
    for i in range(num_conv_layers):
        if i == 0:
            model.add(Conv1D(filters=filters,
                    kernel_size=kernel_size[i],
                    kernel_initializer='he_uniform',
                    activation=activation,
                    padding='same',
                    input_shape=input_shape))
        else:
            if i % 2 == 1:
                model.add(BatchNormalization())
            model.add(Conv1D(filters=filters,
                    kernel_size=kernel_size[i],
                    kernel_initializer='he_uniform',
                    activation=activation,
                    strides=strides))
        model.add(AveragePooling1D(pool_size=2, strides=strides)
            )

    # Flatten the output
    model.add(Flatten())

    # Add fully connected layers
    for i in range(num_fc_layers):
        model.add(Dense(units=size_fc_layers[i],
            kernel_initializer='he_uniform', activation=
            activation))

    # Add an output layer
```

```
model.add(Dense(units=n_output, activation='softmax'))

# Compile the model
model.compile(loss=loss_func, optimizer=Adam(learning_rate),
    metrics=['accuracy'])

return model
```

6.2 Optuna studies for ASCADf

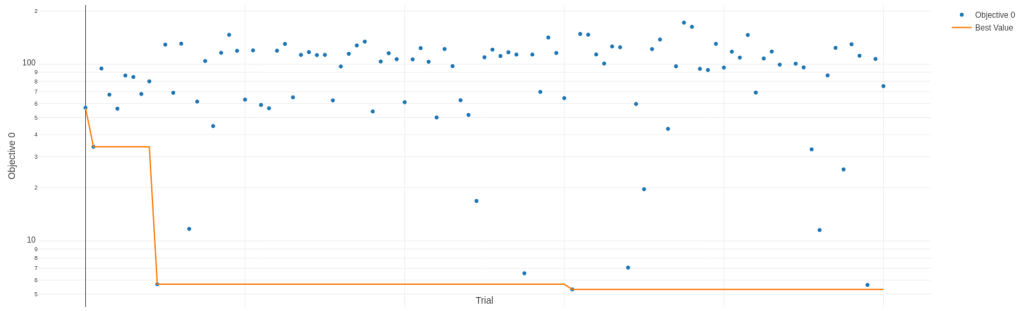


Figure 6.1: Optuna study for Ascadf with warm up set to 30

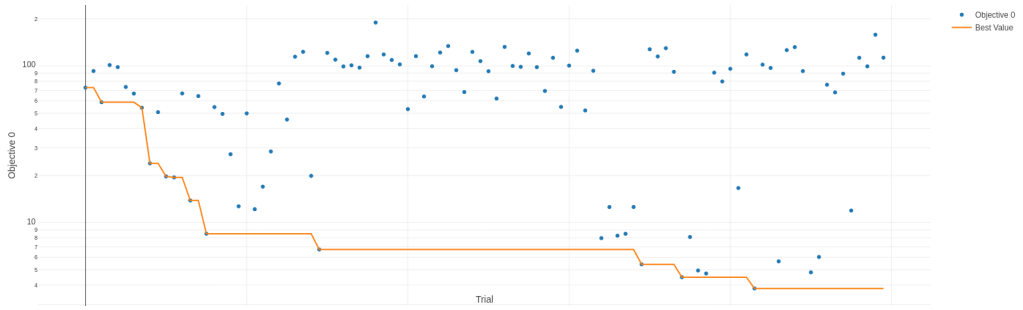


Figure 6.2: Optuna study for Ascadf with warm up set to 50

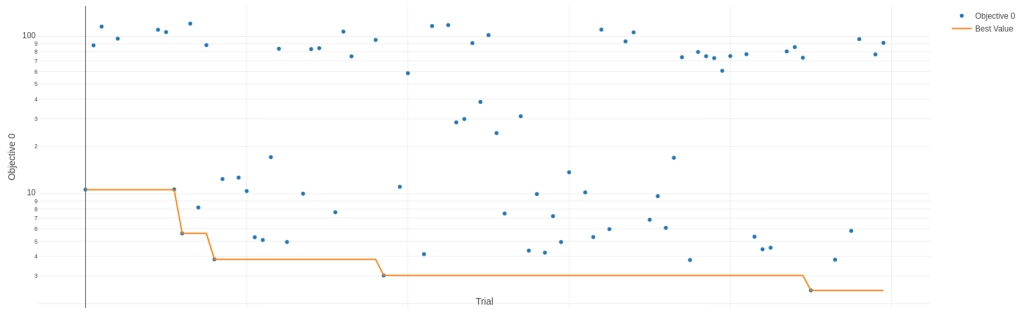


Figure 6.3: Optuna study for Ascadf with warm up set to 75

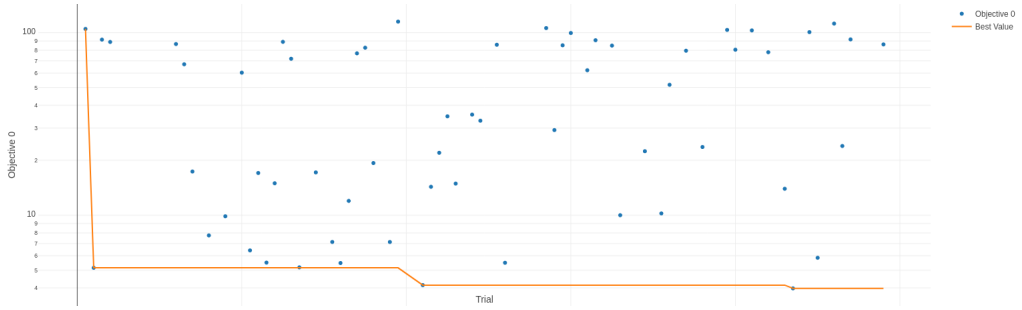


Figure 6.4: Optuna study for Ascadf with warm up set to 100

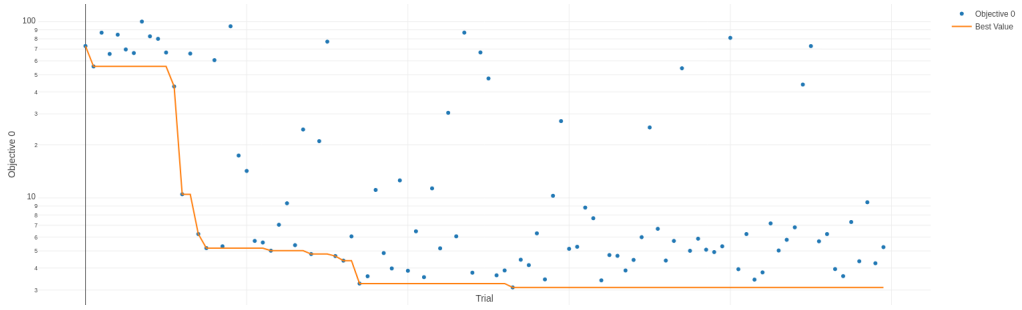


Figure 6.5: Optuna study for Ascadf without early stop

6.3 Optuna studies for ASCADr

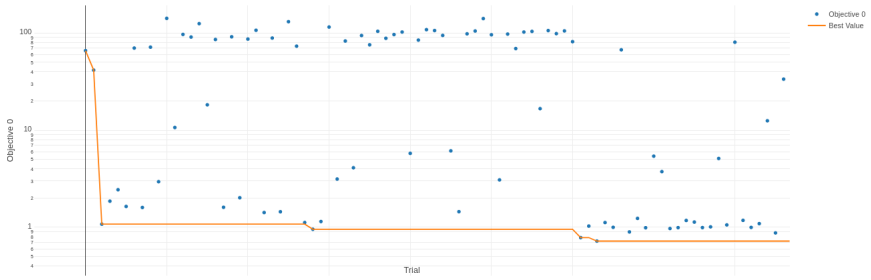


Figure 6.6: Optuna study for ASCADr with warm up set to 30

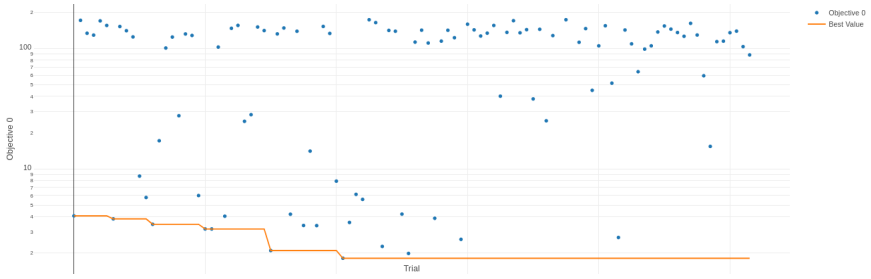


Figure 6.7: Optuna study for ASCADr with warm up set to 50

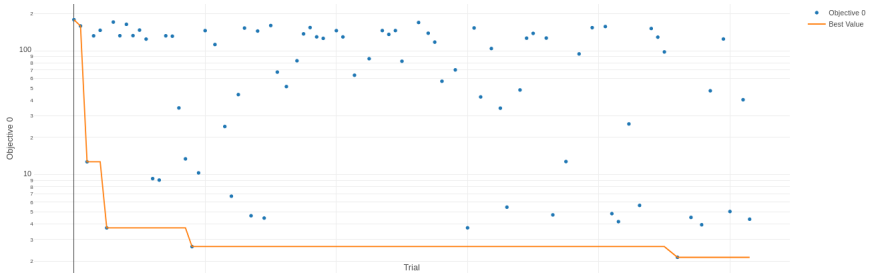


Figure 6.8: Optuna study for ASCADr with warm up set to 75

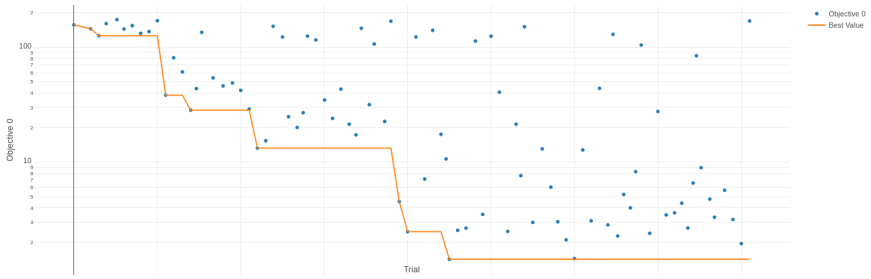


Figure 6.9: Optuna study for ASCADr with warm up set to 100